

**Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования «Казанский (Приволжский) федеральный университет»  
Институт вычислительной математики и информационных технологий**

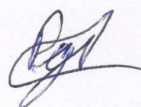
**Кафедра системного анализа и информационных технологий**

Направление подготовки: 02.03.02 — Фундаментальная информатика и  
информационные технологии

Профиль: Системный анализ и информационные технологии

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА  
WEB-ПРИЛОЖЕНИЕ ДЛЯ КОМАНДНОГО ВЗАИМОДЕЙСТВИЯ И  
СОВМЕСТНОЙ РАБОТЫ**

Обучающийся 4 курса  
группы 09-033



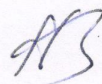
(Рамазанов Р.М.)

Руководитель  
канд. физ.-мат. наук, доцент



(Шаймухаметов Р. Р.)

Заведующий кафедрой системного анализа  
и информационных технологий,  
канд. физ.-мат. наук, доцент



(Васильев А.В.)

Казань – 2024

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ .....	3
1. Анализ предметной области .....	5
1.1. Анализ аналогов .....	5
1.2. Выдвинутые нефункциональные требования .....	6
1.3. Выдвинутые функциональные требования .....	6
2. Инструменты разработки .....	8
3. Архитектура приложения .....	10
4. База данных.....	12
5. Разработка системы мгновенного обмена сообщениями.....	16
5.1. Long Pooling .....	16
5.2. Event Sourcing .....	17
5.3. WebSockets.....	18
5.4. Выбор библиотеки.....	19
5.5. Программная реализация .....	20
6. Механизм аутентификации и автоматизации пользователя .....	22
6.1. Основные токены в NextAuth .....	22
6.2. Работа с токенами и запросами .....	23
6.3. Алгоритм аутентификации и авторизации пользователя.....	25
7. Тестирование API через Postman.....	30
8. Подготовка проекта для развертывания на VPS.....	33
8.1. Sentry для мониторинга ошибок .....	33
8.2. Настройка виртуального выделенного сервера .....	33
9. Анализ результатов разработки.....	35
9.1. Оценка результатов относительно цели и задач.....	35
9.2. Проблемы, возникшие в ходе разработки, и их решение.....	35
ЗАКЛЮЧЕНИЕ .....	37
СПИСОК ЛИТЕРАТУРЫ .....	43
ПРИЛОЖЕНИЕ .....	44

## ВВЕДЕНИЕ

В современных условиях цифровой трансформации, необходимость эффективного командного взаимодействия и совместной работы становится все более актуальной. Разработка web-приложений, предназначенных для этих целей, приобретает особое значение. Такие инструменты позволяют значительно упростить и улучшить процессы коммуникации и координации внутри учебных заведений, где качественное взаимодействие между студентами и преподавателями играет ключевую роль в достижении образовательных целей.

Особенно важной становится данная тема в свете недавних событий, связанных с блокировкой Microsoft Teams в России. Microsoft Teams является одним из наиболее популярных инструментов для командной работы и онлайн-обучения, и его недоступность создает значительные препятствия для образовательных учреждений. В этой связи разработка отечественного аналога, способного обеспечить схожий функционал, становится не только актуальной, но и крайне необходимой. Это позволит учебным заведениям продолжить эффективную организацию учебного процесса, независимо от внешних политических и экономических факторов.

Целью выпускной квалификационной работы является разработка полноценного клиент-серверного web-приложения для командного взаимодействия и совместной работы.

Для достижения поставленной цели были поставлены следующие задачи:

- 1) изучение существующих web-приложений для командного взаимодействия и совместной работы;
- 2) сбор требований от потенциальных пользователей для определения основных функций и возможностей web-приложений для командного взаимодействия и совместной работы;
- 3) проектирование архитектуры web-приложения, включая выбор подходящих технологий и протоколов;

- 4) разработка пользовательского интерфейса и серверной части для работы с серверами, каналами и пользователями, включая синхронизацию и обмен файлами между устройствами, а также обеспечение безопасности данных и аутентификацию пользователей;
- 5) разработка системы мгновенного обмена сообщениями;
- 6) тестирование API через Postman;
- 7) проведение тестирования и отладки разработанного web-приложения для проверки его функциональности и надежности;
- 8) подготовка и настройка проекта для развертывания на VPS виртуальном выделенном сервере: настройка сервера, настройка переменных среды (env), CI/CD для автоматизации развертывания и Sentry для мониторинга ошибок;
- 9) оценка эффективности разработанного web-приложения на основе проведенных тестов и сравнение с существующими решениями.

## **1. Анализ предметной области**

В данном разделе произведен анализ предметной области, связанной с разработкой web-приложения для командного взаимодействия и совместной работы. Были рассмотрены следующие аналоги: Discord, Microsoft Teams и Messenger. Анализ аналогов помог определить преимущества и недостатки существующих решений и выделить ключевые темы, которые необходимо рассмотреть при разработке предлагаемого web-приложения.

### **1.1. Анализ аналогов**

Discord является платформой, изначально созданной для геймеров, но широко используется для общения и сотрудничества в различных сообществах. Он обеспечивает голосовую и текстовую коммуникацию, а также возможности организации серверов и обмена файлами. Discord отличается высокой производительностью и широким набором интегрированных функций. Однако, некоторые пользователи могут отмечать его сложность в использовании и наличие ряда ненужных функций для бизнес-коммуникаций [1].

Microsoft Teams — это интегрированное решение для командной работы, разработанное Microsoft. Оно включает в себя возможности видеоконференций, обмена сообщениями, совместной работы над документами и интеграцию с другими приложениями и сервисами Microsoft. Teams предоставляет широкий набор инструментов для организации рабочего процесса и коммуникаций в команде. Однако, для полноценного использования Teams требуется подписка на платные планы, что может быть недоступно для некоторых пользователей [1].

Messenger — это мессенджер, разработанный Facebook, предназначенный для обмена сообщениями, файлами и звонков. Messenger обеспечивает простоту в использовании и интеграцию с другими сервисами Facebook, что делает его удобным выбором для общения. Однако, он может оставлять желать лучшего в плане организации рабочих процессов и возможностей совместной работы.

## **1.2. Выдвинутые нефункциональные требования**

На основе анализа данных выдвинуты следующие нефункциональные требования:

- безопасность и конфиденциальность данных: одной из ключевых тем, вытекающих из анализа аналогов, является безопасность и конфиденциальность данных. Важно применять механизмы шифрования и защиты данных пользователей, а также предусмотреть меры по предотвращению несанкционированного доступа;

- интерфейс и удобство использования: другой важной темой является разработка удобного и интуитивно понятного пользовательского интерфейса. Необходимо обеспечить простоту и удобство взаимодействия с web-приложением, чтобы пользователи могли легко загружать, организовывать и совместно работать с файлами;

- масштабируемость и производительность: также важно учесть масштабируемость и производительность web-приложения. Предполагается, что количество пользователей и объем хранимых файлов могут значительно возрасти, поэтому необходимо разработать архитектуру, способную эффективно масштабироваться и обеспечивать высокую производительность.

## **1.3. Выдвинутые функциональные требования**

На основе анализа данных выдвинуты следующие функциональные требования:

1) авторизация и управление аккаунтом:

- возможность создания аккаунта и профиля пользователя;
- аутентификация через электронную почту или социальные сети;
- возможность восстановления доступа к учетной записи;
- управление настройками конфиденциальности и безопасности профиля, поддержка двухфакторной аутентификация (2FA);

2) чат и обмен сообщениями:

- мгновенная отправка текстовых сообщений между пользователями и группами,

- поддержка отправки вложений и эмодзи;

### 3) голосовая и видеосвязь:

- возможность совершения голосовых и видеозвонков между пользователями и группами,

- поддержка групповых видеоконференций,

- опции настройки качества и пропускной способности;

### 4) управление серверами:

- возможность создания и настройки собственных серверов;

- ролевая система с разделением прав доступа;

- возможность создания и настройки каналов на сервере;

- поиск по сообщениям, пользовательским профилям и серверам;

- фильтрация контента по категориям и ключевым словам;

### 5) управление сообществом:

- возможность модерации сообщений и пользователей,

- система жалоб и блокировок,

- возможность создания и настройки каналов на сервере.

## 2. Инструменты разработки

В данном проекте использовались различные технологии, специально подобранные для обеспечения эффективной работы как на серверной, так и на клиентской стороне. Каждая из этих технологий играет важную роль в создании современных web-приложений, обеспечивая базовую безопасность, производительность, удобство разработки и множество других возможностей.

Основные библиотеки, использованные в серверной части приложения:

- **Bcrypt** — это криптографический алгоритм хеширования паролей. Он обеспечивает безопасное хранение и проверку паролей на сервере путем хеширования и сравнения хэшей паролей. Bcrypt широко применяется для защиты пользовательских паролей;

- **Dotenv** — это модуль, который позволяет загружать переменные окружения из файла `.env` на сервере. Файл `.env` содержит конфигурационные переменные, такие как секретные ключи, адреса баз данных и другие настройки, которые могут изменяться в разных средах разработки;

- **Express** — это минималистичный и гибкий фреймворк для создания web-приложений на сервере с использованием языка JavaScript. Он предоставляет набор инструментов и маршрутизацию, чтобы упростить разработку серверной части приложения;

- **Jsonwebtoken** — это библиотека, которая позволяет создавать и проверять JSON Web Tokens (JWT) на сервере. JWT — это формат для представления утверждений между двумя сторонами в виде JSON-объекта. Они широко используются для аутентификации и обмена данными между клиентом и сервером;

- **Winston** — это модуль для регистрации событий и журналирования на сервере. Он предоставляет гибкий и настраиваемый механизм регистрации сообщений различных уровней, таких как отладка, информация, предупреждения и ошибки. Winston позволяет сохранять журналы в различных форматах и местах, таких как файлы или базы данных;



— Nodemon — это инструмент разработки, который облегчает процесс разработки на сервере. Он автоматически перезапускает сервер при изменении файлов, что позволяет сразу видеть результаты внесенных изменений без ручного перезапуска сервера.

Технологии на клиенте:

— TypeScript — это язык программирования, который является надмножеством JavaScript. Он добавляет статическую типизацию к JavaScript, позволяя выявлять и предотвращать ошибки на этапе разработки. TypeScript повышает надежность и читаемость кода на клиентской стороне приложения;

— React — это JavaScript-библиотека для разработки пользовательского интерфейса. Она позволяет создавать компоненты, которые являются независимыми и многократно используемыми блоками кода, отвечающими за отображение данных на web-странице. React использует виртуальный DOM для эффективного обновления пользовательского интерфейса [2];

— Next — это фреймворк для разработки web-приложений, основанный на React.js. Он позволяет создавать универсальные приложения, которые могут выполняться как на стороне сервера, так и на стороне клиента.

— Axios — это библиотека для выполнения HTTP-запросов на клиентской стороне. Она обеспечивает простой и удобный интерфейс для отправки запросов на сервер и обработки ответов. Axios поддерживает множество функций, таких как установка заголовков, обработка ошибок и прогресс загрузки;

— Prisma — это современный ORM (Object-Relational Mapping) и инструмент для работы с базами данных, который предоставляет удобный способ взаимодействия с базой данных в приложениях, написанных на TypeScript или JavaScript.

### 3. Архитектура приложения

Для обеспечения эффективной работы приложения была выбрана и реализована следующая архитектурная модель (рисунок 1).

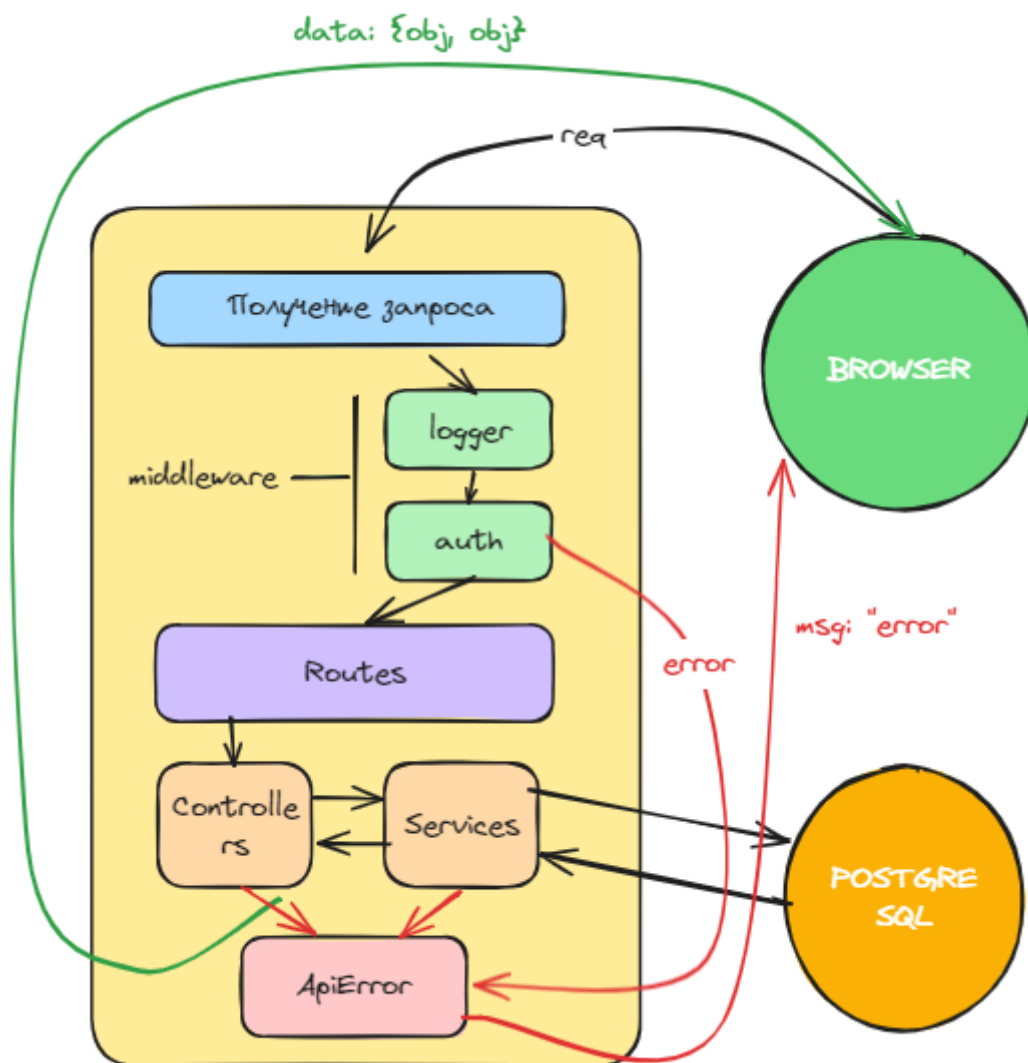


Рисунок 1 – Архитектура серверной части приложения

Слой доступа к данным, известный как DAL (data access layer), был построен с использованием Prisma ORM. Этот слой ответственен за выполнение всех операций с базой данных, обеспечивая ее эффективное использование. В случае использования SQL-запросов напрямую, предполагается выделение логики доступа к данным в отдельный слой или модуль.

Следующий слой архитектуры Contoller, осуществляет обработку клиент-серверных запросов, таких как params, body, headers и другие. Кроме того, контроллер возвращает ответ с сервера на клиент и указывает соответствующий статус код.

Последний слой - Service, отвечает за логику обработки данных. Здесь происходит получение данных из базы данных, их обработка и возврат. Конечное назначение данных уже не имеет значения, поскольку в этом слое они готовы к передаче клиенту или для дальнейшей обработки [3].

Предложенная архитектура была успешно применена в разработке серверной части web-приложения, что обеспечило эффективную обработку данных и высокую отзывчивость системы.

Next с App роутингом представляет собой современную архитектуру клиентской части приложения, основанную на компонентах React и динамическом маршрутизации. Эта архитектура обеспечивает гибкость и масштабируемость разработки, позволяя организовывать интерфейс приложения через компоненты и эффективно управлять навигацией между страницами. Поддержка серверного рендеринга и статической генерации контента позволяет улучшить производительность и SEO-параметры приложения. Такая архитектура является основой для создания современных web-приложений с отзывчивым пользовательским интерфейсом и высокой производительностью (рисунок 2).

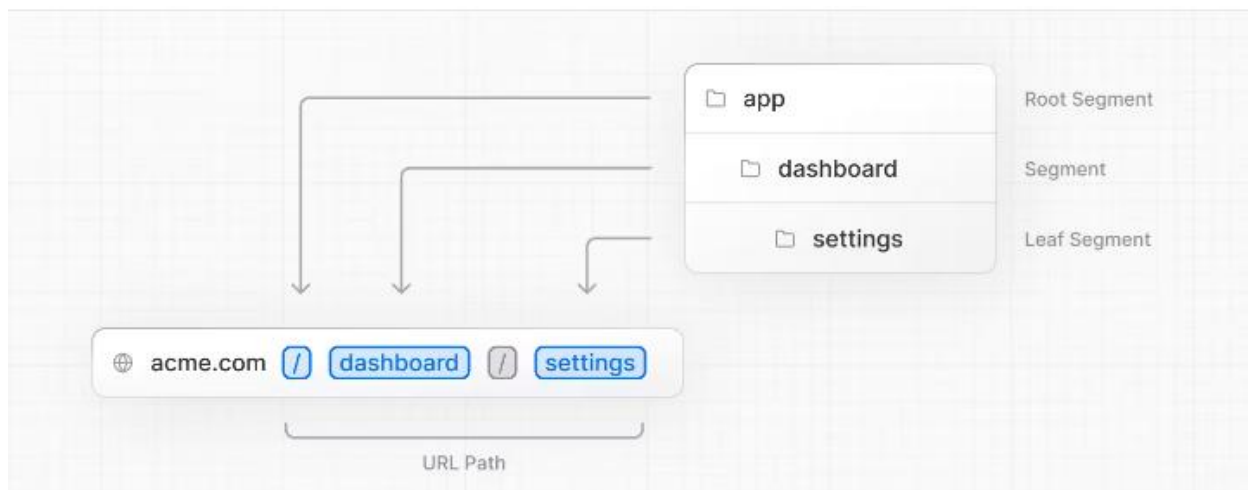


Рисунок 2 – App Routing в Next

#### 4. База данных

При выборе базы данных и ORM принято решение использовать PostgreSQL и Prisma по нескольким причинам:

1) PostgreSQL обладает высокой производительностью и надежностью, поддерживает широкий набор функций для целостности данных, транзакций и масштабируемости. Это делает его подходящим для различных типов приложений [4];

2) Prisma обеспечивает удобство и эффективность разработки, предоставляя средства для определения моделей данных, выполнения запросов и управления схемой базы данных с использованием TypeScript. Он также автоматизирует создание SQL-запросов, снижая вероятность ошибок;

3) PostgreSQL с Prisma обеспечивает безопасность данных приложения благодаря механизмам безопасности PostgreSQL, таким как роли, разрешения и SSL-шифрование, а также защищает от атак SQL-инъекций и других уязвимостей.

Ниже предоставлена схема базы данных (рисунок 3).

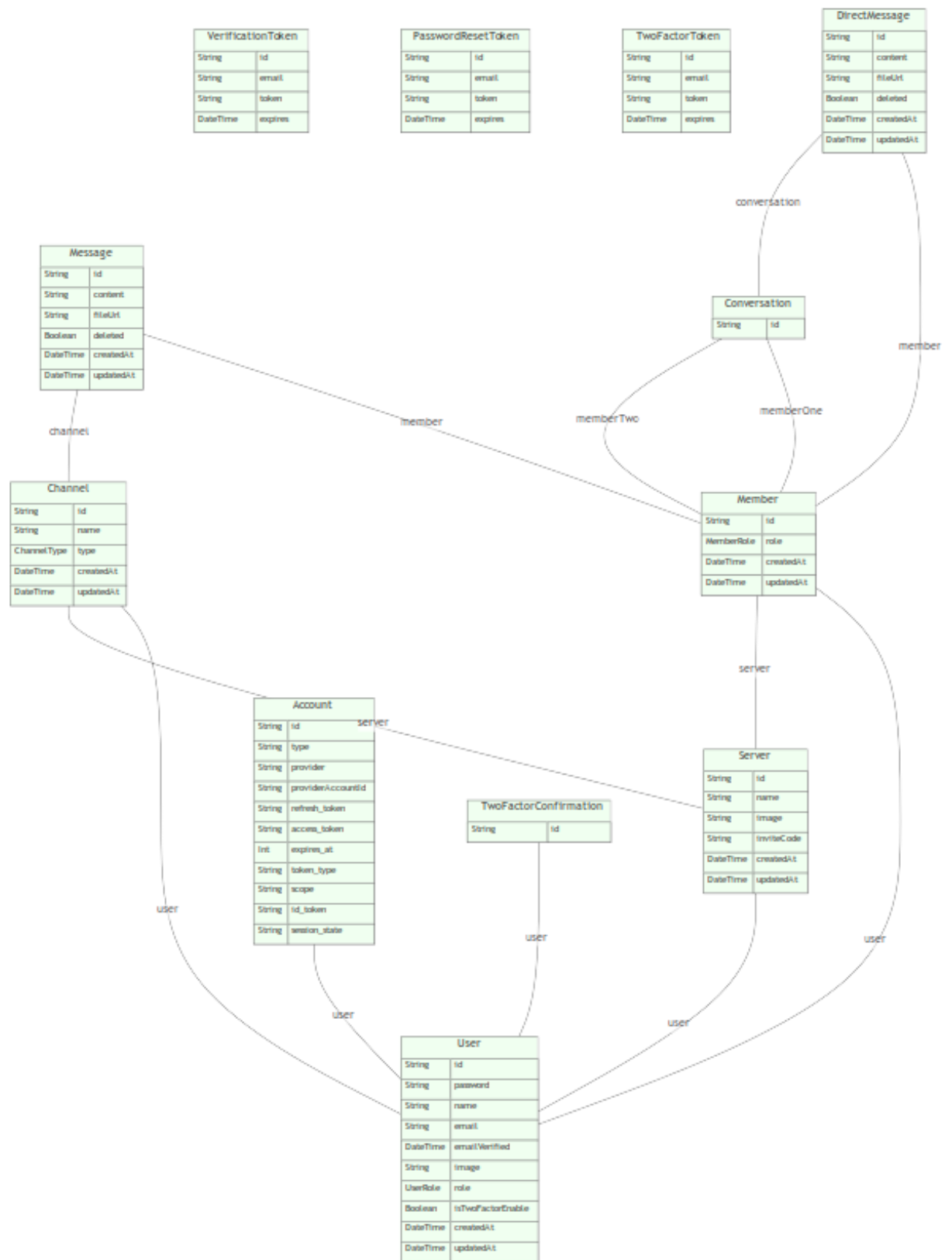


Рисунок 3 – Схема базы данных

Приведенная схема базы данных описывает приложение для обмена сообщениями с использованием Prisma ORM и базы данных PostgreSQL. В приложении предусмотрены сущности, представленные ниже:

— **User:** таблица содержит информацию о пользователях. Каждый пользователь имеет уникальный идентификатор, пароль, имя, адрес электронной почты, дату подтверждения адреса электронной почты, изображение профиля, роль (администратор или обычный пользователь), а также информацию о включенной двухфакторной аутентификации. Кроме того, здесь хранятся связанные с пользователем данные о серверах, участниках и каналах;

— **Server:** в этой таблице содержится информация о серверах. Каждый сервер имеет уникальный идентификатор, имя, изображение, код приглашения и связан с пользователем, который является его владельцем. Также здесь хранится информация о участниках и каналах сервера;

— **Member:** эта таблица содержит информацию об участниках серверов. Каждый участник имеет уникальный идентификатор, роль на сервере (администратор, модератор или гость), и связан с конкретным пользователем и сервером. Здесь также хранятся сообщения, инициированные участником, а также прямые сообщения;

— **Channel:** здесь хранится информация о каналах сервера. Каждый канал имеет уникальный идентификатор, имя, тип (текстовый, аудио или видео), и связан с конкретным пользователем и сервером. Также здесь хранятся сообщения, отправленные в канал;

— **Account:** эта таблица содержит информацию об учетных записях OAuth. Каждая учетная запись имеет уникальный идентификатор, а также связана с конкретным пользователем. Здесь хранится информация о типе, провайдере, токенах доступа и обновления, а также о сроках действия токена;

— **VerificationToken:** в этой таблице хранятся токены подтверждения адреса электронной почты. Каждый токен связан с адресом электронной почты пользователя и имеет срок действия;

— PasswordResetToken: здесь хранятся токены сброса пароля. Каждый токен связан с адресом электронной почты пользователя и имеет срок действия;

— TwoFactorToken: эта таблица содержит токены двухфакторной аутентификации. Каждый токен связан с адресом электронной почты пользователя и имеет срок действия;

— TwoFactorConfirmation: здесь хранятся подтверждения двухфакторной аутентификации. Каждое подтверждение связано с пользователем и указывает, включена ли у него двухфакторная аутентификация;

— Message: содержит информацию о сообщениях в чате. Каждая запись представляет собой отдельное сообщение и включает уникальный идентификатор сообщения, его содержание, ссылку на прикрепленный файл (если есть), а также идентификаторы участника чата и канала. Эти идентификаторы используются для связи с другими таблицами, такими как «Member» (Участник) и «Channel» (Канал);

— Conversation: представляет собой диалог между двумя участниками чата. Она хранит информацию о диалоге, включая идентификаторы обоих участников. Эти идентификаторы используются для связи с таблицей «Member». Также таблица содержит ссылку на прямые сообщения (DirectMessage), отправленные в этом диалоге. Для обеспечения уникальности каждого диалога введен индекс, а также уникальное ограничение на пару идентификаторов участников;

— DirectMessage: содержит информацию о прямых сообщениях между участниками чата. Каждое сообщение имеет уникальный идентификатор, текстовое содержание, ссылку на прикрепленный файл (если есть) и идентификаторы отправителя и диалога. Эти идентификаторы используются для связи с таблицами «Member» и «Conversation». Также как и в «Message», здесь есть флаг удаления и метки времени создания и обновления.

## **5. Разработка системы мгновенного обмена сообщениями**

Реализация чата в современном программном обеспечении включает в себя различные технологии и методы, направленные на обеспечение надежности, масштабируемости и моментальной передачи сообщений. Среди таких технологий можно выделить Long polling, Event Sourcing и WebSockets. Эти подходы позволяют создавать чаты, которые отвечают на запросы пользователей в реальном времени, обеспечивая непрерывное взаимодействие и быструю доставку сообщений.

### **5.1. Long Pooling**

Long Polling - это метод взаимодействия между клиентом и сервером, который позволяет клиенту отправить запрос на сервер и ожидать ответа в течение определенного времени. В отличие от традиционных методов, где сервер сразу же отвечает на запросы, в случае с long polling сервер задерживает ответ до тех пор, пока не произойдет определенное событие.

Принцип работы long polling довольно прост: клиент отправляет запрос на сервер (например, GET /messages), и сервер начинает ожидать события. Если в течение установленного времени не происходит событие, сервер все равно должен ответить клиенту, но может сделать это пустым ответом или с кодом ожидания. После этого клиент снова отправляет запрос, и процесс повторяется.

Однако, когда происходит событие (например, отправка сообщения другим пользователем в чате), сервер немедленно возвращает ответ на запрос клиенту. После этого клиент сразу же отправляет новый запрос, чтобы ожидать следующего события.

Этот метод является одним из самых простых в реализации, поскольку не требует установки постоянного соединения между клиентом и сервером, как в случае с протоколами, такими как WebSocket. Более того, он экономит серверные ресурсы, так как не поддерживает постоянное активное соединение. Однако, он может приводить к задержкам в обработке запросов и не является



наилучшим выбором для приложений, требующих мгновенной реакции на события (рисунок 4).

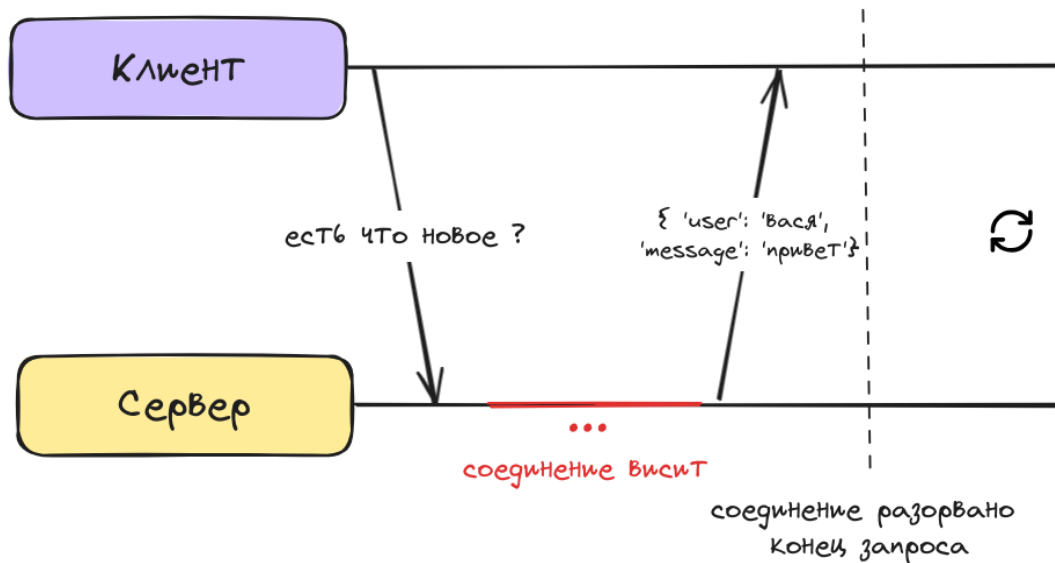


Рисунок 4 – Long Pooling

## 5.2. Event Sourcing

Event sourcing - это методика работы с данными в web-приложениях, которая предполагает использование HTTP протокола для обмена информацией между клиентом и сервером. В отличие от традиционных методов, event sourcing устанавливает постоянное одностороннее подключение к серверу, что позволяет серверу отправлять данные на клиент без необходимости запроса со стороны клиента (рисунок 5).

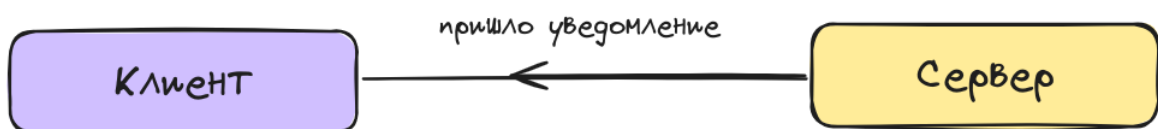


Рисунок 5 – Event Sourcing

Основное преимущество event sourcing заключается в его простоте и эффективности. В отличие от более сложных протоколов, таких как WebSocket, event sourcing не требует двустороннего обмена сообщениями между клиентом и сервером. Это делает его идеальным выбором для ситуаций,

когда необходимо отправлять уведомления или информацию о событиях на клиентскую сторону без необходимости активного участия клиента.

Кроме того, event sourcing не требует дополнительного сервера для его реализации, в отличие от WebSocket, что делает его более легким в развертывании и поддержке.

Однако, важно отметить, что event sourcing не подходит для всех случаев использования. Например, если требуется сложная логика обмена данными между клиентом и сервером или если необходима двусторонняя коммуникация, то лучше использовать более мощные протоколы, такие как WebSocket.

Тем не менее, благодаря своей простоте и эффективности, event sourcing остается важным инструментом в разработке уведомлений и обработки событий.

### 5.3. WebSockets

WebSocket - это мощный протокол, который обеспечивает реальное время взаимодействия между клиентом и сервером в web-приложениях. Он отличается от традиционных HTTP запросов тем, что устанавливает постоянное двустороннее соединение между клиентом и сервером, позволяя обмениваться данными в режиме реального времени (рисунок 6).

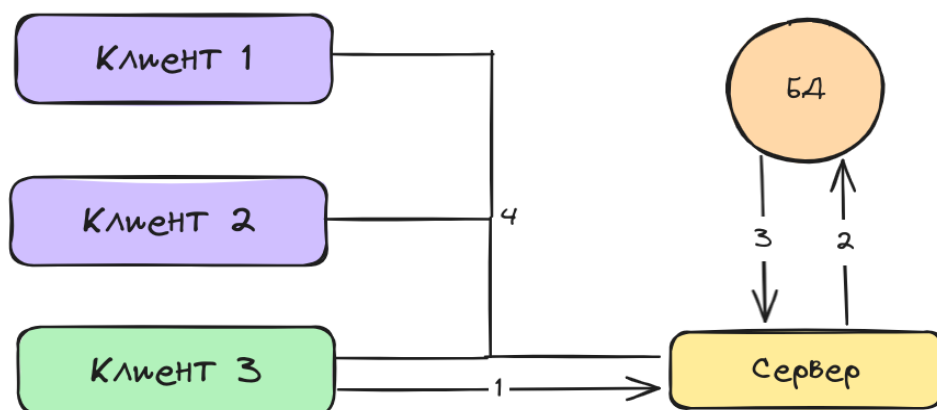


Рисунок 6 – WebSocket

Основное преимущество WebSocket заключается в его способности обеспечивать двунаправленную передачу данных между клиентом и сервером. Это означает, что как клиент, так и сервер могут инициировать передачу данных друг другу без необходимости ожидания запроса от другой стороны. Эта возможность делает WebSocket идеальным выбором для приложений, где требуется мгновенная и синхронизированная передача данных в обе стороны, например, в онлайн-играх, чатах и совместной работе над документами.

Однако следует учитывать, что установка и поддержка WebSocket сервера может потребовать дополнительных ресурсов по сравнению с другими методами взаимодействия, такими как long polling или event sourcing. Это связано с необходимостью постоянного поддержания открытых соединений с клиентами и обработкой потенциально большого количества одновременных подключений.

Несмотря на это, благодаря своей мощности и эффективности, WebSocket остается наиболее предпочтительным выбором для реализации приложений, где требуется высокая степень реактивности и реального времени взаимодействия между клиентом и сервером.

#### **5.4. Выбор библиотеки**

Разобрав основные способы создания real-time чата, было принято решение использовать все три метода. Это достигается путем использования специальной библиотеки Socket.io. Выбор данной библиотеки обусловлен несколькими факторами.

Во-первых, Socket.io предоставляет удобный и единый интерфейс для работы с различными методами обмена данными в реальном времени: long polling, event sourcing и WebSocket. Это позволяет обеспечить поддержку real-time обновлений для всех пользователей, независимо от того, какой метод подключения использует их клиентское приложение.

Во-вторых, библиотека Socket.io предоставляет ряд дополнительных функций, которые делают разработку real-time чата более гибкой и эффективной. К ним относятся автоматическое переподключение при потере

соединения, масштабируемость для работы с большим количеством одновременных подключений и поддержка кросс-браузерности.

Наконец, Socket.io активно поддерживается и обновляется сообществом разработчиков, что гарантирует надежность и актуальность использования данной библиотеки в нашем проекте.

В целом, выбор Socket.io в качестве основной библиотеки для реализации real-time чата позволяет объединить все преимущества трех основных методов взаимодействия, обеспечивая высокую производительность, надежность и удобство использования нашего приложения для всех пользователей.

### **5.5. Программная реализация**

Для реализации был создан обработчик API-запросов для взаимодействия с сокетами (WebSockets). В начале файла импортируются необходимые модули и типы данных. Затем определяется обработчик `ioHandler`, который принимает запросы и ответы для API. Внутри обработчика проверяется, инициализирован ли серверный сокет. Если нет, то создается новый экземпляр сокета и связывается с сервером. После этого функция завершает обработку запроса.

Далее был создан еще один API-запросов, но уже для отправки сообщений в чат. Он импортирует необходимые модули и типы данных, а затем определяет асинхронный обработчик `handler`. Этот обработчик ожидает POST-запросы и проверяет их метод. Если метод не POST, возвращается ошибка «Method not allowed». Далее идет блок `try-catch`, в котором обрабатываются запросы на создание сообщения в чате. Проверяется аутентификация пользователя, наличие необходимых данных (например, содержание сообщения, ID сервера и канала), а также наличие сервера и канала в базе данных. Если все проверки проходят успешно, создается новое сообщение в базе данных и отправляется в определенный канал через сокеты. В случае возникновения ошибки в блоке `try`, ошибка логируется, а клиенту возвращается статус 500 с сообщением об ошибке.

Также важно учитывать, что если клиент не поддерживает работу с WebSockets, то подключение автоматически переходит с использованием Long Pooling.

## **6. Механизма аутентификации и авторизации пользователя**

В данной главе рассматривается процесс реализации механизмов аутентификации и авторизации пользователя web-приложения для командного взаимодействия и совместной работы. Основное внимание уделяется выбору и использованию инструмента NextAuth - библиотеки, предоставляющей простое и эффективное решение для управления аутентификацией в современных web-приложениях [5].

В процессе разработки web-приложения было решено выбрать NextAuth в качестве основного механизма для обеспечения безопасности пользовательских данных и контроля доступа к функционалу приложения. Данное решение было принято на основе анализа требований проекта, возможностей библиотеки и ее применимости к конкретной задаче.

### **6.1. Основные токены в NextAuth**

NextAuth — это мощная библиотека для аутентификации в приложениях Next, предоставляющая удобные методы для управления сессиями пользователей и безопасности. Важными элементами этой системы являются различные токены, обеспечивающие защиту данных и корректное функционирование аутентификационных процессов. В NextAuth используются три основных токена:

1) `authjs.callback-url` — токен представляет URL-адрес обратного вызова (callback URL), который используется для перенаправления пользователя после успешной аутентификации или авторизации. После того как пользователь прошел процесс аутентификации (ввода учетных данных), они перенаправляются обратно на этот URL с дополнительной информацией о результате аутентификации. Этот URL-адрес обычно является страницей приложения, которая обрабатывает успешную аутентификацию и выполняет необходимые действия (например, обновление пользовательского интерфейса);

2) `authjs.csrf-token` — токен CSRF (Cross-Site Request Forgery) используется для защиты от атак, связанных с подделкой межсайтовых

запросов. Он генерируется и включается в запросы на сервер для проверки подлинности запроса. При выполнении операций аутентификации и авторизации, особенно когда данные отправляются на сервер, токен CSRF предотвращает возможность подделки запросов. При получении запроса сервер проверяет, соответствует ли предоставленный токен CSRF ожидаемому значению. Это помогает предотвратить атаки, в которых злоумышленник может отправить запросы от имени пользователя без его согласия;

3) `authjs.session-token` — токен представляет собой уникальный идентификатор сеанса пользователя после успешной аутентификации. После успешной аутентификации пользователю присваивается сеансовый токен, который затем используется для идентификации его при последующих запросах к серверу. Сеансовый токен позволяет серверу отслеживать и управлять сеансом пользователя, обеспечивая ему доступ к защищенным ресурсам или функциям приложения в течение определенного периода времени.

Использование этих трех токенов вместе обеспечивает безопасный и надежный механизм аутентификации и авторизации в приложениях, построенных с использованием библиотеки NextAuth (рисунок 7).

<code>authjs.callback-url</code>	<code>http%3A%2F%2Flocalhost%3A3000%2Fservers</code>
<code>authjs.csrf-token</code>	<code>055fd7775849c57f43bbf4b1c9b057b16b684dd72f28343e16...</code>
<code>authjs.session-token</code>	<code>eyJhbGciOiJIJzI1LCJlbnMiOiJBMjU2Q0JDLUhTNTYliiwia2lkI...</code>

Рисунок 7 – Три основных токена

## 6.2. Работа с токенами и запросами

Теперь разберем взаимодействие между клиентом и сервером в рамках работы с токенами, аутентификации и авторизации пользователя.

### 1) аутентификация:

- клиент отправляет запрос на сервер с данными аутентификации (email + пароль или OAuth);
- если данные верны, сервер возвращает токены `authjs.callback-url`, `authjs.csrf-token` и `authjs.session-token`;

## 2) запрос данных:

- клиент, чтобы получить доступ к защищенным ресурсам на сервере, прикрепляет токены к запросу (например, в заголовке или как параметры запроса);

- сервер проверяет валидность токенов при получении запроса. Если токены действительны, сервер обрабатывает запрос и возвращает данные;

## 3) истечение срока действия токена:

- по прошествии времени токены истекают;
- если клиент пытается использовать устаревшие токены для доступа к защищенным ресурсам, сервер возвращает статус ошибки 401 (например, «Forbidden!») и сообщение о том, что доступ запрещен;

## 4) обновление токенов:

- клиент обнаруживает истечение срока действия токенов и инициирует процесс их обновления;

- обычно для обновления токенов используется механизм, предоставляемый библиотекой NextAuth;

- сервер возвращает новые токены, которые клиент сохраняет и использует для последующих запросов.

Описание отражает типичный процесс работы с аутентификацией и токенами в Next с использованием библиотеки NextAuth (рисунок 8).



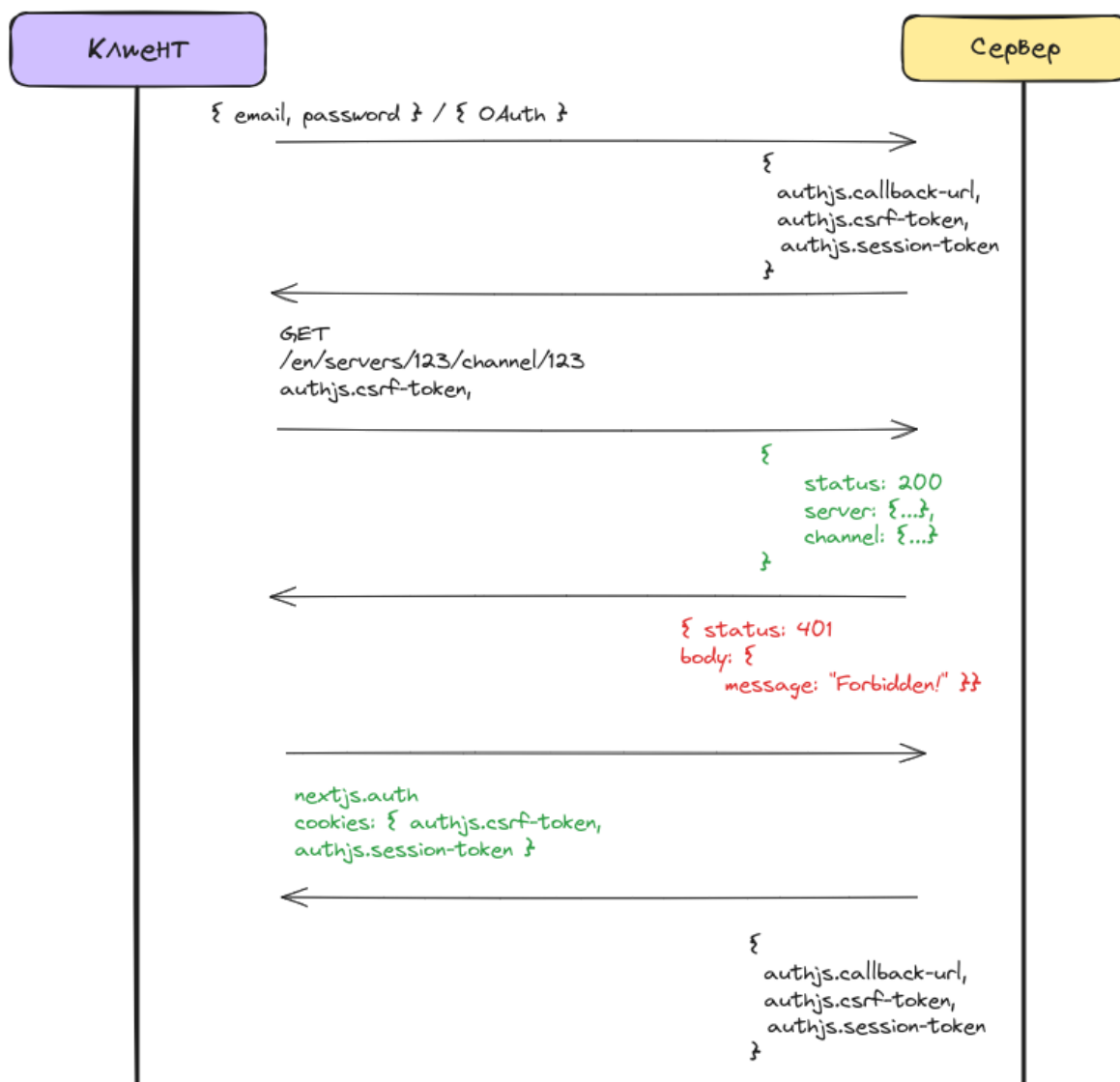


Рисунок 8 – Работа с токенами и запросами

### 6.3. Алгоритм аутентификации и авторизации пользователя

Существует множество готовых алгоритмов авторизации, однако, с учетом особенностей и требований к проекту, они не всегда подходят в идеальном виде. В рамках выпускной квалификационной работы посвященной созданию web-приложения для командной работы, было выяснено, что стандартные алгоритмы авторизации не могут удовлетворить всем потребностям проекта. Сложность настроек и требования к гибкости в управлении пользователями, их ролями и доступом к данным требовали более глубокого и индивидуального подхода. В связи с этим было принято решение разработать собственный алгоритм, используя возможности библиотеки NextAuth.





коды, которые отправляются на почту. В случае наличия двухфакторной аутентификации, происходят дополнительные проверки на валидность кода, срок действия токена, а также работа с базой данных для удаления устаревших токенов и временного подтверждения сеанса.

На стороне клиента происходит работа с сессией, вызов обратных функций для корректной логики NextAuth, после чего пользователю предоставляется доступ к приватным страницам.

Алгоритм также охватывает процессы регистрации новых пользователей и входа в систему (рисунок 11).

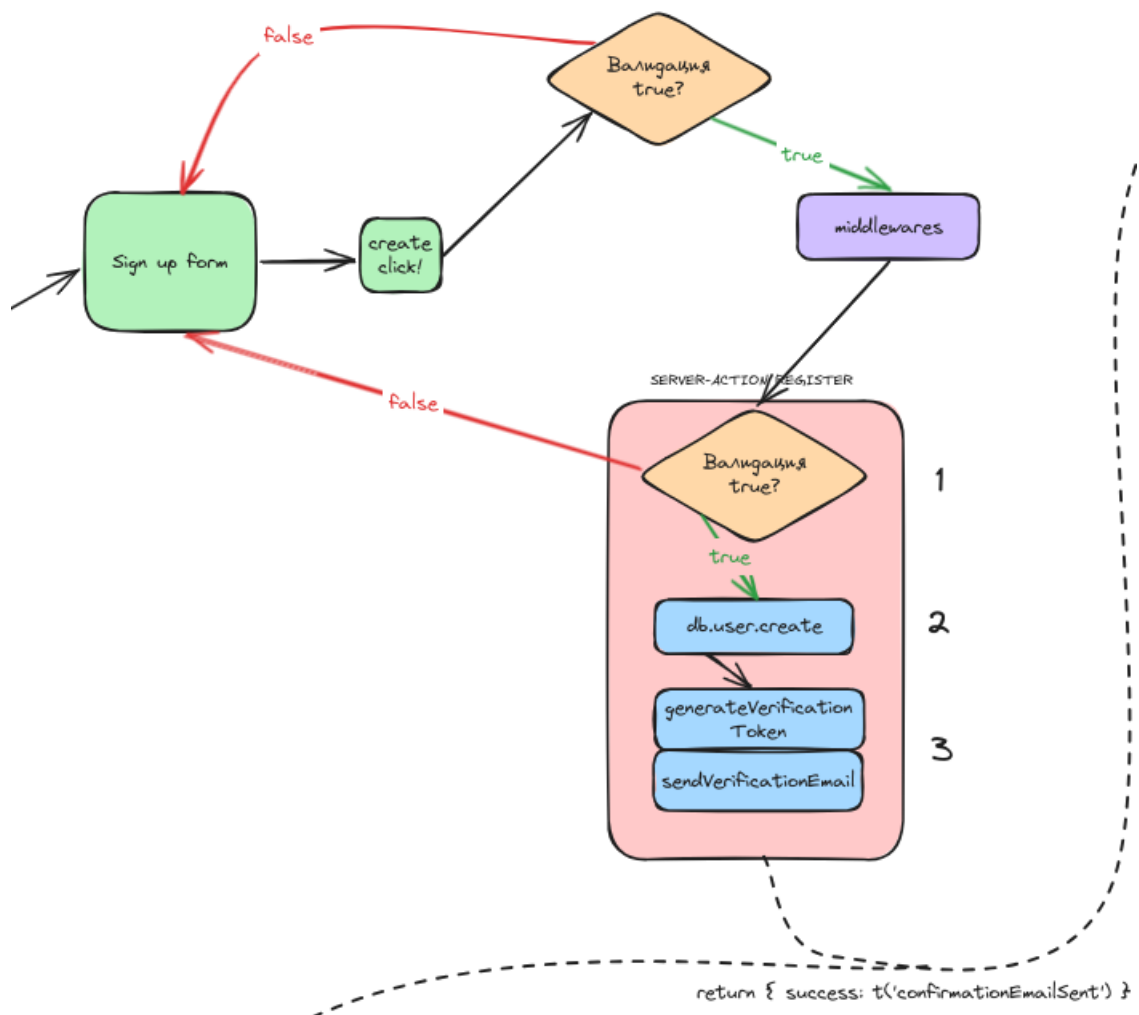


Рисунок 11 – Алгоритм регистрации в системе

Процесс регистрации начинается с валидации формы на стороне клиента, чтобы предотвратить возможность отправки вредоносного кода. После этого данные передаются на сервер, где происходит дополнительная валидация полей. Если все проверки прошли успешно, сервер создает соль для последующего хэширования пароля, после чего сам пароль хэшируется. Если пользователь уже зарегистрирован, клиенту отправляется ошибка. В случае нового пользователя создается новая запись в базе данных с хэшированным паролем. Затем автоматически генерируется токен верификации, который отправляется на указанный адрес электронной почты.

А также аутентификации через популярные социальные сети. Дополнительно в алгоритме реализованы механизмы подтверждения почты. Двухфакторной аутентификации. Восстановления пароля. Кроме того, была предусмотрена система ролей для пользователей, обеспечивающая гибкое управление правами доступа и функциональностью приложения.

## 7. Тестирование API через Postman

Postman - это мощный инструмент для тестирования API, который предоставляет интуитивно понятный интерфейс для создания, отправки и отслеживания HTTP-запросов и ответов. Он позволяет автоматизировать тестирование API, создавать коллекции запросов для повторного использования и обеспечивает широкие возможности для настройки запросов и проверки ответов.

Для тестирования будут отправляться GET или POST запросы. Будут иметься ожидаемые данные и фактические данные, которые пришли, для последующего сравнения с базой данных и проверки их корректности. Отправляемые запросы будут иметь различные параметры и тела, в зависимости от функциональности, которую необходимо протестировать. Важно будет осуществить анализ ответов на запросы и проверить их соответствие ожидаемым данным.

И так, для начала необходимо задать соответствующие Cookies, так как сервер допускает только авторизованных пользователей (рисунок 12).



Рисунок 12 – Cookies

Проверим, что API возвращает правильную активную сессию, то есть берет актуальные данные из базы данных о пользователе и возвращает их. Для этого отправим GET запрос по адресу «http://localhost:3000/api/auth/session» с необходимыми Cookies файлами и заголовками.

Ожидаемый ответ должен содержать данную запись из базы данных (рисунок 13).



Рисунок 13 – Данные пользователя в базе данных

Фактический результат содержит нужные нам поля. Все работает правильно (рисунок 14).

```
"user": {  
  "name": "nikita",  
  "email": "rodion-web@yandex.ru",  
  "image": null,  
  "id": "clvjfcvnd0000mvy1eakvsatd",  
  "role": "ADMIN",  
  "isTwoFactorEnable": false,  
  "isOAuth": false  
},  
"expires": "2024-05-30T13:29:43.468Z"
```

Рисунок 14 – Ответ от сервера

Далее проведем тестирования назначения определенного участника сервера новыми правами, в данном случае сделаем из гостя модератора сервера. Для этого отправим POST запрос по адресу «<http://localhost:3000/ru/servers/clvjfu6ef000amvy1uts2k36f/channels/clvjfu6ef000smvy1otp665dj>». В полезной нагрузке будут содержаться следующие данные (рисунок 15).

▼ Запрос сведений о полезной нагрузке      посмотреть ресурс

▼ ["clvjfz2ef000gmvy1toi6f0v4", "MODERATOR", "clvjfu6ef000amvy1uts2k36f"]

0: "clvjfz2ef000gmvy1toi6f0v4"  
1: "MODERATOR"  
2: "clvjfu6ef000amvy1uts2k36f"

Рисунок 15 – Тело запроса на сервер

В ответе ожидается полностью обновленную информацию о сервера, также к данным о сервере должны быть еще и данные о участниках данного сервера (рисунок 16).

```

{
  "id": "clvjfu6ef000amvyluts2k36f",
  "name": "my nikita server",
  "image": "https://utfs.io/f/644e3838-8ae4-42aa-9bfe-6474ee2b805",
  "inviteCode": "fc4490f5-44ff-4b9a-979a-436933819e92",
  "userId": "clvjfcvnd0000mvyleakvsatd",
  "createdAt": "$D2024-04-28T11:19:17.991Z",
  "updatedAt": "$D2024-04-30T13:40:30.041Z",
  "members": [
    {
      "id": "clvjfz2ef000gmvy1toi6f0v4",
      "role": "MODERATOR",
      "userId": "clvjfcvnd0000mvyleakvsat1",
      "serverId": "clvjfu6ef000amvyluts2k36f",
      "createdAt": "$D2024-04-28T11:23:06.087Z",
      "updatedAt": "$D2024-04-30T14:30:47.285Z",
      "user": {
        "id": "clvjfcvnd0000mvyleakvsat1",
        "password": null,
        "name": "Rodion Ramazanov",
        "email": "rodion.rv111@icloud.com",
        "emailVerified": "$D2024-04-28T11:05:53.016Z",
        "image": "https://avatars.githubusercontent.com/u/6025706",
        "role": "USER",
        "isTwoFactorEnable": false,
        "createdAt": "$D2024-04-28T11:05:50.906Z",
        "updatedAt": "$D2024-04-28T11:05:53.017Z"
      }
    }
  ],
},

```

Рисунок 16 — Ответ сервера

Полученные данные полностью соответствуют ожидаемым, вместе с участником также пришли и данные о самом пользователе. Назначение новой роли участнику сервера работает верно.

Теперь протестируем функционал восстановления, а именно обработку ошибки, что введенный адрес электронной почты не существует. Для этого отправим POST запрос на адрес «<http://localhost:3000/en/auth/reset>» с данными {email: «rodion-web@yandex.ru»} ожидаем получить ошибку на английском языке, что данный адрес электронной почты не найден (рисунок 17).

error: "Email not found!"

Рисунок 17 — Адрес электронной почты не найден



## 8. Подготовка проекта для развертывания на VPS

### 8.1. Sentry для мониторинга ошибок

Sentry — это инструмент для мониторинга и отслеживания ошибок в программном обеспечении. Он позволяет разработчикам отлавливать и анализировать ошибки, исключения и проблемы производительности в реальном времени. Sentry помогает выявлять проблемы, которые могут возникнуть в приложении или на web-сайте, и предоставляет подробную информацию о них, что помогает разработчикам быстро реагировать и устранять проблемы, улучшая стабильность и качество программного продукта.

Из недостатков важно отметить, что библиотека достаточно большая по размерам. Однако в нашем случае пользователю достаточно подгрузить все один раз, закешировать и в дальнейшем все будет работать быстро.

И так для его настройки был создан аккаунт на соответствующем сервисе. Инициализирована библиотека в проекте. Была произведена дополнительная настройка для отлавливания ошибок серверных компонентов. Важно отметить, что ошибки будут отлавливаться только на production (рисунок 18).

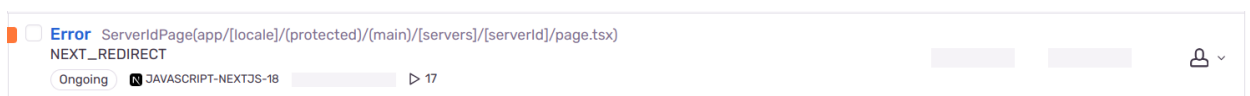


Рисунок 18 — Пример отлавливания ошибки с помощью Sentry

### 8.2. Настройка виртуального выделенного сервера

Важно отметить, что выбор хостинга играет ключевую роль в скорости и стабильности работы приложения. Было принято решение использовать Railway, были учтены не только его технические характеристики, но и доступность цен и удобство в настройке. Это помогло обеспечить оптимальное сочетание мощности, производительности и бюджетных ограничений.

При подготовке сервера к работе, осуществлялись ключевые шаги: приобретение домена, настройка DNS записей и поддоменов для обеспечения доступности приложения, а также приобретение базы данных необходимого размера. Важно подчеркнуть, что для обеспечения безопасности, доступ к базе данных был ограничен только к выбранным IP-адресам, что помогло предотвратить несанкционированный доступ.

Дополнительно, для обеспечения безопасного удаленного доступа к серверу, была проведена процедура аутентификации по SSH с использованием публичного ключа, что повышает уровень защиты системы. Кроме того, создание пользователей и правильная настройка их прав также являются важными шагами для обеспечения безопасности и эффективности работы сервера.

Также, в процессе настройки сервера, была внедрена технология Docker и Docker Compose, что позволяет упростить управление приложением и его зависимостями, а также обеспечить изолированную среду для работы приложения, что способствует стабильности и надежности его работы.

## **9. Анализ результатов разработки**

### **9.1. Оценка результатов относительно цели и задач**

Разработка web-приложения для командного взаимодействия и совместной работы успешно завершена в соответствии с изначально поставленными целями и задачами. Приложение предоставляет все необходимые функциональные возможности для эффективного взаимодействия и общения между пользователями.

В процессе развития приложения были осуществлены значительные улучшения и добавлены ключевые функциональности. В частности, были внедрены настройки конфиденциальности и безопасности профиля, включая поддержку двухфакторной аутентификации (2FA). Дополнительно была создана удобная система управления серверами, каналами и данными пользователей, а также добавлена функциональность присоединения к серверам. Важным аспектом стало обеспечение возможности обмена сообщениями между пользователями для организации коммуникации. Помимо этого, приложение было адаптировано под два языка — русский и английский, что способствует повышению доступности для широкой аудитории. В результате всех этих изменений удалось достичь высокой производительности и стабильности работы приложения.

### **9.2. Проблемы, возникших в ходе разработки, и их решение**

В процессе разработки проекта возникал ряд проблем, связанных как с техническими, так и с организационными аспектами. Рассмотрим наиболее значимые из них:

1) обеспечение безопасности данных - одной из ключевых задач было гарантировать безопасность данных пользователей, включая их личную информацию, резюме и сообщения. Для решения этой проблемы были внедрены механизмы аутентификации и авторизации на основе стандартов OAuth и JWT, с применением библиотеки NextAuth. Это позволило обеспечить надежный контроль доступа к конфиденциальным данным;

2) для обеспечения синхронизации данных между клиентом и сервером в проекте, были применены несколько методов. Во-первых, были созданы API конечные точки на серверной стороне для обработки запросов от клиентской части и передачи соответствующих данных. Затем, на клиентской стороне, для получения и отправки данных к серверу были использованы AJAX/HTTP запросы. Наконец, для управления состоянием данных на клиенте была выбрана библиотека Zustand. Этот инструмент позволил эффективно хранить и обновлять данные в приложении, обеспечивая их актуальность;

3) обеспечение удобного пользовательского интерфейса - создание интуитивно понятного и эргономичного пользовательского интерфейса было важной задачей. После проведения тестов юзабилити были внесены изменения в дизайн и навигацию приложения, что улучшило общее впечатление пользователей.

Таким образом, в ходе разработки были успешно решены различные технические и пользовательские проблемы. Применение современных подходов, архитектурных решений и методов тестирования позволило создать надежное, эффективное и удобное в использовании web-приложения для командного взаимодействия и совместной работы.

## ЗАКЛЮЧЕНИЕ

В процессе выполнения данной выпускной квалификационной работы был приобретён значительный и ценный опыт в области разработки web-приложений с использованием современных технологий, таких как Node.js, Express, React, Next.js, Prisma ORM и TypeScript. В ходе работы было осознано, насколько важно обеспечивать безопасность при работе с файлами и сообщениями, а также применять соответствующие меры для защиты данных пользователей от различных угроз. В частности, была реализована продвинутая система авторизации, включающая восстановление пароля, подтверждение адреса электронной почты, двухфакторную аутентификацию и вход через различные сторонние сервисы.

Работа над выпускной квалификационной работой также включала создание клиент-серверного web-приложения, целью которого было улучшение командной работы и совместного взаимодействия. Для достижения поставленных целей были определены и решены конкретные задачи. Одной из них была разработка системы мгновенного обмена сообщениями, что потребовало тщательного тестирования API с помощью инструмента Postman. Также была проведена подготовка проекта к развертыванию на виртуальном выделенном сервере (VPS), что включало настройку сервера, определение переменных среды (env), внедрение CI/CD для автоматизации процесса развертывания и установку системы мониторинга ошибок Sentry.

Помимо этого, в процессе работы удалось глубже понять архитектуру современных web-приложений и изучить лучшие практики по их разработке и поддержке.

Полученные результаты свидетельствуют о том, что все поставленные цели были успешно достигнуты. Выпускная квалификационная работа завершена с положительными результатами, что подтверждает эффективность использованных технологий и методов. Более того, разработанное web-приложение обладает потенциалом для дальнейшего развития и применения в

реальных условиях, что открывает новые возможности для его использования и совершенствования.

За период выполнения выпускной квалификационной работы были приобретены следующие компетенции (таблица 1).

Таблица 1 – Таблица компетенций

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
УК-1	Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	Способен искать, анализировать и объединять информацию, применять системный подход для решения задач
УК-2	Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	Способен выделять основные задачи для достижения цели и выбирать наилучшие методы их решения с учетом правовых норм, доступных ресурсов и ограничений
УК-3	Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде	Способен работать в коллективе и выполнять свои обязанности внутри команды
УК-4	Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах)	Способен общаться в рабочих ситуациях как на русском, так и на иностранном языке, как устно, так и письменно
УК-5	Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах	Способен понимать различия и особенности различных культур в контексте их исторического, этического и философского значения

Продолжение таблицы 1

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
УК-6	Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	Способен эффективно управлять своим расписанием, создавать и достигать цели по саморазвитию, опираясь на принципы непрерывного образования
УК-7	Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности	Способен поддерживать необходимую физическую форму для успешного выполнения обязанностей как в общественной, так и профессиональной сфере
УК-8	Способен создавать и поддерживать безопасные условия жизнедеятельности, в том числе при возникновении чрезвычайных ситуаций	Способен обеспечивать безопасность и защиту в экстренных ситуациях, обеспечивая безопасные условия для жизни и работы
ОПК-1	Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности	Способен использовать фундаментальные знания из математики и естественных наук в профессиональной деятельности
ОПК-2	Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение, в том числе отечественного происхождения, для решения задач профессиональной деятельности	Способен использовать современные компьютерные и методы, также программное обеспечение, включая отечественное, для выполнения профессиональных задач

Продолжение таблицы 1

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
ОПК-3	Способен к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям	Обладает навыками разработки алгоритмов и программных решений в различных областях программирования, создания математических и информационных моделей, а также образовательного контента. Кроме того, он способен создавать прикладные базы данных, тесты и средства тестирования для проверки соответствия стандартам и требованиям
ОПК-4	Способен участвовать в разработке технической документации программных продуктов и комплексов с использованием стандартов, норм и правил, а также в управлении проектами создания информационных систем на стадиях жизненного цикла	Способен принимать участие в создании технической документации для программных продуктов и комплексов, соблюдая стандарты и правила, а также управлять проектами разработки информационных систем на всех этапах их жизненного цикла
ОПК-5	Способен устанавливать и сопровождать программное обеспечение информационных систем и баз данных, в том числе отечественного происхождения, с учетом информационной безопасности	Способен устанавливать и поддерживать программное обеспечение для информационных систем и баз данных, включая отечественное программное обеспечение, с учетом безопасности информации
ПК-1	Преподавание по дополнительным общеобразовательным программам	Способен преподавать по дополнительным общеобразовательным программам



Продолжение таблицы 1

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
ПК-2	Проверка работоспособности и рефакторинг кода программного обеспечения	Способен проверять работоспособность кода программного обеспечения и проводить тестирование
ПК-3	Интеграция программных модулей и компонент и верификация выпусков программного продукта	Способен совмещать программные модули и компоненты, а также проверять готовые версии программных продуктов
ПК-4	Разработка требований и проектирование программного обеспечения	Способен разрабатывать требования и проектировать программное обеспечение
ПК-5	Оценка и выбор варианта архитектуры программного средства	Умею оценивать и выбирать варианты архитектуры программных средств
ПК-6	Разработка тестовых случаев, проведение тестирования и исследование результатов	Способен тестировать программное обеспечение и проводить анализ на основе результатов
ПК-7	Обеспечение функционирования баз данных	Способен обеспечивать полное функционирование баз данных
ПК-8	Оптимизация функционирования баз данных	Способен оптимизировать полное функционирование баз данных
ПК-9	Обеспечение информационной безопасности на уровне базы данных	Способен обеспечивать защиту информации на уровне базы данных
ПК-10	Выполнение работ по созданию (модификации) и сопровождению информационных систем, автоматизирующих задачи организационного управления и бизнес-процессы	Способен выполнить разработку, изменение и поддержку информационных систем, которые помогают автоматизировать управленческие задачи и бизнес-процессы организации

Продолжение таблицы 1

Компетенция	Расшифровка компетенции	Описание приобретенных знаний, умений и навыков
ПК-11	Создание и сопровождение требований и технических заданий на разработку и модернизацию систем и подсистем малого и среднего масштаба и сложности	Способен создавать и обновлять требования и спецификации для создания и улучшения систем и компонентов небольших и средних проектов с умеренным уровнем сложности
ПК-12	Способен находить организационно-управленческие решения в нестандартных ситуациях и готов нести за них ответственность	Способен принимать нестандартные организационно-управленческие решения и нести за них ответственность
ПК-13	Способен к коммуникации, восприятию информации, умению логически верно, аргументировано и ясно строить устную и письменную речь на русском языке для решения профессиональных задач	Способен эффективно общаться, понимать информацию, логически и аргументированно выражать свои мысли как устно, так и письменно на русском языке для выполнения профессиональных задач
ПК-14	Способен использовать действующее законодательство и другие правовые документы в своей деятельности, демонстрировать готовность и стремление к совершенствованию и развитию общества на принципах гуманизма, свободы и демократии	Способен применять законы и другие правовые акты в своей работе, а также проявлять готовность к развитию общества на основе гуманизма, свободы и демократии

## СПИСОК ЛИТЕРАТУРЫ

- 1) Discord vs Microsoft Teams : Versus [Электронный ресурс] – URL: <https://versus.com/ru/discord-vs-microsoft-teams> (дата обращения: 20.02.24).
- 2) Documentation : react.dev [Электронный ресурс] – URL: <https://react.dev/learn> (дата обращения: 17.02.24).
- 3) Архитектура современных корпоративных Node.js-приложений : Habr [Электронный ресурс] – URL: <https://habr.com/ru/companies/yandex/articles/514550/> (дата обращения: 20.02.24).
- 4) Documentation : PostgreSQL [Электронный ресурс] – URL: <https://www.postgresql.org/docs/16/index.html> (дата обращения: 19.02.24).
- 5) Upgrade Guide (v5) : Auth.js [Электронный ресурс] – URL: <https://authjs.dev/guides/upgrade-to-v5> (дата обращения: 18.02.24).

## ПРИЛОЖЕНИЕ

```
import bcryptjs from 'bcryptjs'
import Github from 'next-auth/providers/github'
import Google from 'next-auth/providers/google'
import Credentials from 'next-auth/providers/credentials'
import { NextResponse } from 'next/server'
import type { NextAuthConfig } from 'next-auth'

import { LoginSchema } from '@/schemas'
import { getUserByEmail, getUserById } from '@/data/user'
import { DEFAULT_LOGIN_REDIRECT, authRoutes, publicRoutes } from
'@/routes'
import { locales } from '@/navigation'

const publicPagesPathnameRegex = RegExp(
  `^(/(${locales.join('|')}))?(${[...publicRoutes,
...authRoutes]
  .flatMap((p) => (p === '/' ? [' ', '/'] : p))
  .join('|')})/?$`,
  'i'
)

const authPagesPathnameRegex = RegExp(
  `^(/(${locales.join('|')}))?(${authRoutes
  .flatMap((p) => (p === '/' ? [' ', '/'] : p))
  .join('|')})/?$`,
  'i'
)

// возможно здесь идет установка нужных провайдеров (входов,
github, google , credentials)
export default {
  providers: [
    Google({
      clientId: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
      allowDangerousEmailAccountLinking: true
    }),
    Github({
      clientId: process.env.GITHUB_CLIENT_ID,
      clientSecret: process.env.GITHUB_CLIENT_SECRET
    }),
    Credentials({
      async authorize(credentials, request) {
        const validatedFields =
LoginSchema.safeParse(credentials)

        if (validatedFields.success) {
          const { email, password } = validatedFields.data

          const user = await getUserByEmail(email)
```

```

        // если вход был выполнен через Google или Github
        // у пользователя не будет password, тогда
        // мы не используем authorize по credentials.
        if (!user || !user.password) {
            return null
        }

        const passwordsMatch = await
bcryptjs.compare(password, user.password)

        if (passwordsMatch) {
            return user
        }
    }
    return null
}
}))
],
callbacks: {
    authorized: async ({ auth, request }) => {
        const { nextUrl } = request

        // console.log(nextUrl.pathname)

        const isAuthenticated = !!auth
        const isPublicPage =
publicPagesPathnameRegex.test(nextUrl.pathname)
        const isAuthPage =
authPagesPathnameRegex.test(nextUrl.pathname)

        if (isAuthenticated && isAuthPage) {
            return NextResponse.redirect(new
URL(DEFAULT_LOGIN_REDIRECT, nextUrl))
        }

        if (!(isAuthenticated || isPublicPage)) {
            return NextResponse.redirect(
                // new URL(`/signin?callbackUrl=${nextUrl.pathname}`,
nextUrl)
                new URL('/auth/signin', nextUrl)
            )
        }

        return isAuthenticated || isPublicPage
    },
    session: async ({ token, session }) => {
        if (token.sub && session.user) {
            session.user.id = token.sub
        }

        if (token.role && session.user) {
        }
        session.user.role = token.role
    }
}

```

```

    return session
  },
  jwt: async ({ token }) => {
    if (!token.sub) {
      return token
    }

    const existingUser = await getUserById(token.sub)

    if (!existingUser) {
      return token
    }

    token.role = existingUser.role

    return token
  }
}
} satisfies NextAuthConfig

'use server'

import * as z from 'zod'
import { currentUser } from '@lib/auth'
import { db } from '@lib/db'
import { CreateChannelSchema } from '@schemas'
import { MemberRole } from '@prisma/client'

export const createChannel = async (
  values: z.infer<typeof CreateChannelSchema>,
  serverId: string
) => {
  try {
    const validatedFields =
      CreateChannelSchema.safeParse(values)

    if (!validatedFields.success) {
      return { error: 'Invalid fields!' }
    }

    const { name, type } = validatedFields.data

    const user = await currentUser()

    if (!user || !user.id) {
      return { error: 'Unauthorized!', status: 401 }
    }

    if (!serverId) {
      return { error: 'Server ID Missing!', status: 400 }
    }
  }
}

```

```

    if (name === 'general') {
    return { error: 'Name cannot be "general"!', status: 400 }
    }

    const server = await db.server.update({
    where: {
    id: serverId,
    members: {
    some: {
    userId: user.id,
    role: {
    in: [MemberRole.ADMIN, MemberRole.MODERATOR]
    }
    }
    },
    data: {
    channels: {
    create: {
    userId: user.id,
    name,
    type
    }
    }
    }
    })

    return server
    } catch (error) {
    console.error('Error:', error)
    return { error: 'Something went wrong!', status: 500 }
    }
    }

    'use server'

    import { currentUser } from '@lib/auth'
    import { db } from '@lib/db'
    import { MemberRole } from '@prisma/client'

    export const deleteChannel = async (serverId: string,
    channelId: string) => {
    try {
    const user = await currentUser()

    if (!user) {
    return { error: 'Unauthorized!', status: 401 }
    }

    if (!serverId) {
    return { error: 'Server ID Missing!', status: 400 }
    }
    }
    }

```

```

if (!channelId) {
return { error: 'Channel ID Missing!', status: 400 }
}

const server = await db.server.update({
  where: {
    id: serverId,
    members: {
      some: {
        userId: user.id,
        role: {
          in: [MemberRole.ADMIN, MemberRole.MODERATOR]
        }
      }
    },
  },
  data: {
    channels: {
      delete: {
        id: channelId,
        name: {
          not: 'general'
        }
      }
    }
  }
})

return server
} catch (error) {
  console.error('Error:', error)
  return { error: 'Something went wrong!', status: 500 }
}
}

```

'use server'

```

import { currentUser } from '@/lib/auth'
import { db } from '@/lib/db'

export const deleteServer = async (serverId: string) => {
  try {
    const user = await currentUser()

    if (!user) {
      return { error: 'Unauthorized!', status: 401 }
    }

    if (!serverId) {
      return { error: 'Server ID Missing!', status: 400 }
    }

    const server = await db.server.delete({

```



```

    where: {
      id: serverId,
      userId: user.id
    }
  })

  return server
} catch (error) {
  console.error('Error:', error)
  return { error: 'Something went wrong!', status: 500 }
}
}

'use server'

import * as z from 'zod'
import { EditChannelSchema } from '@schemas'
import { currentUser } from '@lib/auth'
import { db } from '@lib/db'
import { MemberRole } from '@prisma/client'

export const editChannel = async (
  values: z.infer<typeof EditChannelSchema>,
  serverId: string,
  channelId: string
) => {
  try {
    const validatedFields = EditChannelSchema.safeParse(values)

    if (!validatedFields.success) {
      return { error: 'Invalid fields!', status: 400 }
    }

    const { name, type } = validatedFields.data

    const user = await currentUser()

    if (!user) {
      return { error: 'Unauthorized!', status: 401 }
    }

    if (!serverId) {
      return { error: 'Server ID Missing!', status: 400 }
    }

    if (!channelId) {
      return { error: 'Channel ID Missing!', status: 400 }
    }

    if (name === 'general') {
      return { error: 'Name cannot be "general"', status: 400 }
    }
  }
}

```

```

const server = await db.server.update({
  where: {
    id: serverId,
    members: {
      some: {
        userId: user.id,
        role: {
          in: [MemberRole.ADMIN, MemberRole.MODERATOR]
        }
      }
    },
  },
  data: {
    channels: {
      update: {
        where: {
          id: channelId,
          NOT: {
            name: 'general'
          }
        },
      },
    },
    data: {
      name,
      type
    }
  }
})

return server
} catch (error) {
  console.error('Error:', error)
  return { error: 'Something went wrong!', status: 500 }
}
}

```

```

const EVENTS = {
  SET_USERNAME: "set_username",
  USER_JOINED: "user_joined",
  USER_LEFT: "user_left",
  CHAT_MESSAGE: "chat_message",
  USER_TYPING: "user_typing",
  ROOM_CHANGE: "room_change",
};

export class Socket {
  constructor() {
    /* global io */
    this.socket = io({
      auth: {
        serverOffset: 0,
      },
      ackTimeout: 10000,
    });
  }

  onHandler = (eventName) => (handler) => {
    this.socket.on(eventName, handler);
  };

  createHandler =
    (eventName) =>
    (message = undefined) => {
      this.socket.emit(eventName, message);
    };

  /* handlers */
  onSetUsername = this.onHandler(EVENTS.SET_USERNAME);
  onUserJoined = this.onHandler(EVENTS.USER_JOINED);
  onUserLeft = this.onHandler(EVENTS.USER_LEFT);
  onUserTyping = this.onHandler(EVENTS.USER_TYPING);
  onChatMessage = this.onHandler(EVENTS.CHAT_MESSAGE);

  /* commands */
  emitChatMessage = this.createHandler(EVENTS.CHAT_MESSAGE);
  emitUserTyping = this.createHandler(EVENTS.USER_TYPING);
  emitRoomChange = this.createHandler(EVENTS.ROOM_CHANGE);
}

const EVENTS = {
  SET_USERNAME: "set_username",
  USER_JOINED: "user_joined",
  USER_LEFT: "user_left",
  CHAT_MESSAGE: "chat_message",
  USER_TYPING: "user_typing",

```

```

ROOM_CHANGE: "room_change",
};

export class Socket {
  constructor() {
    /* global io */
    this.socket = io({
      auth: {
        serverOffset: 0,
      },
      ackTimeout: 10000,
      retries: 3,
    });
  }

  onHandler = (eventName) => (handler) => {
    this.socket.on(eventName, handler);
  };

  createHandler =
    (eventName) =>
    (message = undefined) => {
      this.socket.emit(eventName, message);
    };

  /* handlers */
  onSetUsername = this.onHandler(EVENTS.SET_USERNAME);
  onUserJoined = this.onHandler(EVENTS.USER_JOINED);
  onUserLeft = this.onHandler(EVENTS.USER_LEFT);
  onUserTyping = this.onHandler(EVENTS.USER_TYPING);
  onChatMessage = this.onHandler(EVENTS.CHAT_MESSAGE);

  /* commands */
  emitChatMessage = this.createHandler(EVENTS.CHAT_MESSAGE);
  emitUserTyping = this.createHandler(EVENTS.USER_TYPING);
  emitRoomChange = this.createHandler(EVENTS.ROOM_CHANGE);
}

'use server'

import * as z from 'zod'
import bcryptjs from 'bcryptjs'

import { RegisterSchema } from '@schemas'
import { db } from '@lib/db'
import { getUserByEmail } from '@data/user'
import { getTranslations } from 'next-intl/server'
import { generateVerificationToken } from '@lib/tokens'
import { sendVerificationEmail } from '@lib/mail'

```

```

    export const register = async (values: z.infer<typeof
RegisterSchema>) => {
    const validatedFields = RegisterSchema.safeParse(values)
    const t = await
getTranslations('RegisterForm.registerErrorSuccessMessages')

    if (!validatedFields.success) {
    return { error: t('invalidFields') }
    }

    const { email, name, password } = validatedFields.data

    const salt = await bcryptjs.genSalt(10)
    const hashedPassword = await bcryptjs.hash(password, salt)

    const existingUser = await getUserByEmail(email)

    if (existingUser) {
    return { error: t('emailAlreadyInUse') }
    }

    await db.user.create({
    data: {
    email,
    name,
    password: hashedPassword
    }
    })

    const verificationToken = await
generateVerificationToken(email)
    await sendVerificationEmail(verificationToken.email,
verificationToken.token)

    return { success: t('emailsent') }
    }

'use server'

import * as z from 'zod'
import { AuthError } from 'next-auth'
import { getTranslations } from 'next-intl/server'

import { signIn } from '@/auth'
import { LoginSchema } from '@/schemas'
import { DEFAULT_LOGIN_REDIRECT } from '@/routes'
import { getUserByEmail } from '@/data/user'
import { generateVerificationToken, generateTwoFactorToken }
from '@/lib/tokens'
import { sendVerificationEmail, sendTwoFactorTokenEmail }
from '@/lib/mail'
import { getTwoFactorTokenByEmail } from '@/data/two-factor-
token'

```

```

import { db } from '@/lib/db'
import { getTwoFactorConfirmationByUserId } from '@/data/two-
factor-confirmation'

export const login = async (values: z.infer<typeof
LoginSchema>) => {
  const validatedFields = LoginSchema.safeParse(values)
  const t = await getTranslations('LoginForm.loginErrorSuccessMessages')

  if (!validatedFields.success) {
    return { error: t('invalidFields') }
  }

  const { email, password, twofacode } = validatedFields.data

  const existingUser = await getUserByEmail(email)

  if (!existingUser || !existingUser.email ||
!existingUser.password) {
    return { error: t('emailDoesNotExist') }
  }

  if (!existingUser.emailVerified) {
    const verificationToken = await generateVerificationToken(
existingUser.email
)

    await sendVerificationEmail(
verificationToken.email,
verificationToken.token
)

    return { success: t('confirmationEmailSent') }
  }

  if (existingUser.isTwoFactorEnable && existingUser.email) {
    if (twofacode) {
      const twoFactorToken = await getTwoFactorTokenByEmail(existingUser.email)

      if (!twoFactorToken) {
        return { error: t('invalidcode') }
      }

      if (twoFactorToken.token !== twofacode) {
        return { error: t('invalidcode') }
      }

      const hasExpired = new Date(twoFactorToken.expires) < new
Date()

      if (hasExpired) {

```

```

    return { error: t('codeexpired') }
  }

  await db.twoFactorToken.delete({
    where: { id: twoFactorToken.id }
  })

  const existingConfirmation = await
getTwoFactorConfirmationByUserId(
  existingUser.id
)

  if (existingConfirmation) {
    await db.twoFactorConfirmation.delete({
      where: { id: existingConfirmation.id }
    })
  }

  await db.twoFactorConfirmation.create({
    data: {
      userId: existingUser.id
    }
  })
} else {
  const twoFactorToken = await
generateTwoFactorToken(existingUser.email)
  await sendTwoFactorTokenEmail(twoFactorToken.email,
twoFactorToken.token)

  return { twoFactor: true }
}

try {
  await signIn('credentials', {
    email,
    password,
    // redirectTo: callbackUrl || DEFAULT_LOGIN_REDIRECT
    redirectTo: DEFAULT_LOGIN_REDIRECT
  })

  return { success: t('success') }
} catch (error) {
  console.error(error)

  if (error instanceof AuthError) {
    switch (error.type) {
      case 'CredentialsSignin': {
        return {
          error: t('invalidCredentials')
        }
      }
      default: {

```

```

    return { error: t('somethingWentWrong') }
  }
}

throw error
}
}

// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily
with your serverless or edge functions?
// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

// User -> это email + password (credentials)
// Account -> это OAuth, также создается User

// prisma/schema.prisma
datasource db {
  provider = "postgresql"
  url = env("DATABASE_URL")
  // uncomment next line if you use Prisma <5.10
  // directUrl = env("DATABASE_URL_UNPOOLED")
}

generator client {
  provider = "prisma-client-js"
}

enum UserRole {
  ADMIN
  USER
}

// ----- squad

model User {
  id String @id @default(cuid())
  password String?
  name String?
  email String? @unique
  emailVerified DateTime?
  image String? @db.Text
  role UserRole @default(USER)
  accounts Account[]

  isTwoFactorEnable Boolean @default(false)
  twoFactorConfirmation TwoFactorConfirmation?
  servers Server[]

```



```

members Member[]
channels Channel[]

createdAt DateTime @default(now())
updatedAt DateTime @updatedAt
}

model Server {
  id String @id @default(cuid())
  name String
  image String @db.Text
  inviteCode String @unique

  userId String
  user User @relation(fields: [userId], references: [id],
onDelete: Cascade)

  members Member[]
  channels Channel[]

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  @@index([userId])
}

enum MemberRole {
  ADMIN
  MODERATOR
  GUEST
}

model Member {
  id String @id @default(cuid())
  role MemberRole @default(GUEST)

  userId String
  user User @relation(fields: [userId], references: [id],
onDelete: Cascade)
  serverId String
  server Server @relation(fields: [serverId], references: [id],
onDelete: Cascade)

  messages Message[]

  conversationInitiated Conversation[] @relation("MemberOne")
  conversationReceived Conversation[] @relation("MemberTwo")

  directMessages DirectMessage[]

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

```

```

    @@index([userId])
    @@index([serverId])
}

enum ChannelType {
    TEXT
    AUDIO
    VIDEO
}

model Channel {
    id String @id @default(cuid())
    name String
    type ChannelType @default(TEXT)

    userId String
    user User @relation(fields: [userId], references: [id],
onDelete: Cascade)

    serverId String
    server Server @relation(fields: [serverId], references: [id],
onDelete: Cascade)

    messages Message[]

    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt

    @@index([userId])
    @@index([serverId])
}

// -----

model Account { // OAuth account
    id String @id @default(cuid())
    userId String
    type String
    provider String
    providerAccountId String
    refresh_token String? @db.Text
    access_token String? @db.Text
    expires_at Int?
    token_type String?
    scope String?
    id_token String? @db.Text
    session_state String?

    user User @relation(fields: [userId], references: [id],
onDelete: Cascade)

    @@unique([provider, providerAccountId])

```

```

    }

    model VerificationToken {
        id String @id @default(cuid())
        email String
        token String @unique
        expires DateTime

        @@unique([email, token])
    }

    model PasswordResetToken {
        id String @id @default(cuid())
        email String
        token String @unique
        expires DateTime

        @@unique([email, token])
    }

    model TwoFactorToken {
        id String @id @default(cuid())
        email String
        token String @unique
        expires DateTime

        @@unique([email, token])
    }

    model TwoFactorConfirmation {
        id String @id @default(cuid())

        userId String
        user User @relation(fields: [userId], references: [id],
onDelete: Cascade)

        @@unique([userId])
    }

    // messenger

    model Message {
        id String @id @default(cuid())
        content String @db.Text

        fileUrl String? @db.Text

        memberId String
        member Member @relation(fields: [memberId], references: [id],
onDelete: Cascade)

        channelId String

```

```

    channel Channel @relation(fields: [channelId], references:
[id], onDelete: Cascade)

    deleted Boolean @default(false)

    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt

    @@index([channelId])
    @@index([memberId])
}

model Conversation {
  id String @id @default(cuid())

  memberOneId String
  memberOne Member @relation("MemberOne", fields:
[memberOneId], references: [id], onDelete: Cascade)

  memberTwoId String
  memberTwo Member @relation("MemberTwo", fields:
[memberTwoId], references: [id], onDelete: Cascade)

  directMessages DirectMessage[]

  // @@index([memberOneId])
  @@index([memberTwoId])

  @@unique([memberOneId, memberTwoId])
}

model DirectMessage {
  id String @id @default(cuid())
  content String @db.Text
  fileUrl String? @db.Text

  memberId String
  member Member @relation(fields: [memberId], references: [id],
onDelete: Cascade)

  conversationId String
  conversation Conversation @relation(fields:
[conversationId], references: [id], onDelete: Cascade)

  deleted Boolean @default(false)

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  @@index([memberId])
  @@index([conversationId])
}

```

```

/**
 * findConversation находит в бд ранее созданную конференцию
или вернет null
 * createNewConversation создает новую конференцию в бд
 *
 * getOrCreateConversation ф-ция, которая пытается сначала
найти конференцию по айдишникам
 * иначе создает новую конференцию и ее возвращает
 */

import { db } from '@lib/db'

export const getOrCreateConversation = async (
  memberOneId: string,
  memberTwoId: string
) => {
  let conversation =
    (await findConversation(memberOneId, memberTwoId)) ||
    (await findConversation(memberTwoId, memberOneId))

  if (!conversation) {
    conversation = await createNewConversation(memberOneId,
memberTwoId)
  }

  return conversation
}

const findConversation = async (memberOneId: string,
memberTwoId: string) => {
  try {
    return await db.conversation.findFirst({
      where: {
        AND: [{ memberOneId: memberOneId, memberTwoId: memberTwoId }]
      },
      include: {
        memberOne: {
          include: {
            user: true
          }
        },
        memberTwo: {
          include: {
            user: true
          }
        }
      }
    })
  } catch (error) {
    console.error('ERROR:', error)
    return null
  }
}

```

```

const createNewConversation = async (
  memberOneId: string,
  memberTwoId: string
) => {
  try {
    return await db.conversation.create({
      data: {
        memberOneId: memberOneId,
        memberTwoId: memberTwoId
      },
      include: {
        memberOne: {
          include: {
            user: true
          }
        },
        memberTwo: {
          include: {
            user: true
          }
        }
      })
  } catch (error) {
    console.error('ERROR:', error)

    return null
  }
}

/**
 * sendTwoFactorTokenEmail - отправляет на почту письмо с 2FA
Code
 * sendPasswordResetEmail - отправляет ссылку на
восстановление пароля
 * sendVerificationEmail - отправляет ссылку на подтверждение
почты
 */

import { Resend } from 'resend'

const resend = new Resend(process.env.RESEND_API_KEY)

export const sendTwoFactorTokenEmail = async (email: string,
token: string) => {
  await resend.emails.send({
    from: 'onboarding@resend.dev',
    to: email,
    subject: '2FA Code',
    html: `

Your 2FA code ${token}</p>`
  })
}


```

```

    export const sendPasswordResetEmail = async (email: string,
token: string) => {
    const resetLink = `http://localhost:3000/auth/new-
password?token=${token}`

    try {
    await resend.emails.send({
    from: 'onboarding@resend.dev',
    to: email,
    subject: 'Reset your password',
    html: `

Click <a href="${resetLink}">here</a> to reset
password</p>`
    })
    } catch (error) {
    console.error('Error:', error)
    return { error }
    }
    }

    export const sendVerificationEmail = async (email: string,
token: string) => {
    const confirmLink = `http://localhost:3000/auth/new-
verification?token=${token}`

    try {
    await resend.emails.send({
    from: 'onboarding@resend.dev',
    to: email,
    subject: 'Confirm your email',
    html: `

Click <a href="${confirmLink}">here</a> to confirm
email.</p>`
    })
    } catch (error) {
    console.error('Error:', error)
    return { error }
    }
    }

/**
 * generatePasswordResetToken - токен-восстановления-пароля:
 * задает дату когда токен истекает,
 * далее если находит токен-восстановления-пароля в бд,
удаляет его,
 * далее создает в бд новый токен-восстановления-пароля и
возвращает его
 *
 * generateVerificationToken - такой же принцип, только теперь
токен-подтверждения-почты
 *
 * generateTwoFactorToken - такой же принцип, только теперь
2FA-токен
 */


```

```

import crypto from 'crypto'
import { v4 as uuidv4 } from 'uuid'

import { getVerificationTokenByEmail } from
'@/data/verification-token'
import { getPasswordResetTokenByEmail } from
'@/data/password-reset-token'
import { getTwoFactorTokenByEmail } from '@/data/two-factor-
token'
import { db } from '@/lib/db'

export const generatePasswordResetToken = async (email:
string) => {
  const token = uuidv4()

  // Mon Mar 04 2024 19:10:22 GMT+0300 (Москва, стандартное
  время)
  const expires = new Date(new Date().getTime() + 3600 * 1000)

  const existingToken = await
getPasswordResetTokenByEmail(email)

  if (existingToken) {
    await db.passwordResetToken.delete({
      where: { id: existingToken.id }
    })
  }

  const passwordResetToken = await
db.passwordResetToken.create({
    data: {
      email,
      token,
      expires
    }
  })

  return passwordResetToken
}

export const generateVerificationToken = async (email:
string) => {
  const token = uuidv4()

  // Mon Mar 04 2024 19:10:22 GMT+0300 (Москва, стандартное
  время)
  const expires = new Date(new Date().getTime() + 3600 * 1000)

  const existingToken = await
getVerificationTokenByEmail(email)

  if (existingToken) {

```



```

    await db.verificationToken.delete({
      where: {
        id: existingToken.id
      }
    })
  }

  const verificationToken = await db.verificationToken.create({
    data: {
      email,
      token,
      expires
    }
  })

  return verificationToken
}

export const generateTwoFactorToken = async (email: string)
=> {
  const token = crypto.randomInt(100_000, 1_000_000).toString()

  // 5 минут
  const expires = new Date(new Date().getTime() + 5 * 60 * 1000)

  const existingToken = await getTwoFactorTokenByEmail(email)

  if (existingToken) {
    await db.twoFactorToken.delete({
      where: {
        id: existingToken.id
      }
    })
  }

  const twoFactorToken = await db.twoFactorToken.create({
    data: {
      email,
      token,
      expires
    }
  })

  return twoFactorToken
}

```