Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт Вычислительной математики и информационных технологий

ОТЧЕТ

по проектно-технологической (производственной) практике

Обучающийся Рамазанов Родион Михайлович гр.09-033 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО студента) (Группа) (Подпись)

Научный руководитель:  канд.физ.-мат.наук, доцент,

доцент кафедры САИТ Шаймухаметов Р.Р. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Руководитель практики от кафедры:

ст.преподаватель кафедры САИТ Тихонова О.О. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Оценка за практику \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Дата сдачи отчета \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Казань – 2024

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc160725700)

[1. Анализ предметной области 4](#_Toc160725701)

[1.1. Анализ аналогов 4](#_Toc160725702)

[1.2. Выдвинутые нефункциональные требования 5](#_Toc160725703)

[1.3. Выдвинутые функциональные требования 5](#_Toc160725704)

[2. Инструменты разработки 7](#_Toc160725705)

[3. Программная реализация 9](#_Toc160725706)

[3.1. Архитектура приложения 9](#_Toc160725707)

[3.2. База данных 11](#_Toc160725708)

[3.3. Авторизация 14](#_Toc160725709)

[ЗАКЛЮЧЕНИЕ 16](#_Toc160725710)

[СПИСОК ЛИТЕРАТУРЫ 18](#_Toc160725711)

[ПРИЛОЖЕНИЕ 19](#_Toc160725712)

# ВВЕДЕНИЕ

Производственная практика проходила на кафедре системного анализа и информационных технологий Института вычислительной математики и информационных технологий КФУ с 09 февраля 2024 года по 09 марта 2024 года.

Целью практики 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 файлами, папками и сообщения в мессенджере, включая синхронизацию и обмен файлами между устройствами, а также обеспечение безопасности данных и аутентификацию пользователей;
5. проведение тестирования и отладки разработанного web-приложения для проверки его функциональности и надежности;
6. 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. Он широко используется для личного и делового общения благодаря своей популярности и доступности на различных платформах. 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:

* пользователи должны иметь возможность зарегистрироваться на платформе путем указания своего адреса электронной почты, имени пользователя и пароля;
* должна быть возможность создания учетной записи через сторонние платформы, такие как Google и Github;
* пользователи должны иметь возможность восстановить доступ к своей учетной записи в случае утери пароля путем отправки временного пароля на зарегистрированный адрес электронной почты;
* платформа должна поддерживать возможность настройки двухфакторной аутентификации для усиления безопасности учетной записи;
* пользователи должны иметь возможность управлять своими учетными записями, изменять пароль, адрес электронной почты и другие связанные с ними данные;
* пользователи могут создавать собственные сервера для общения в различных тематиках;
* платформа должна поддерживать возможность настройки различных параметров сервера, таких как название, логотип;
* наличие текстовых, голосовых и видео каналов для общения. Возможность создания временных каналов для конкретных целей или мероприятий. Поддержка передачи файлов и встраивания медиа контента в чате.

# 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.

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.js — 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. Программная реализация

## 3.1. Архитектура приложения

Для обеспечения эффективной работы приложения была выбрана и реализована следующая архитектурная модель.

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 (рисунок 1).

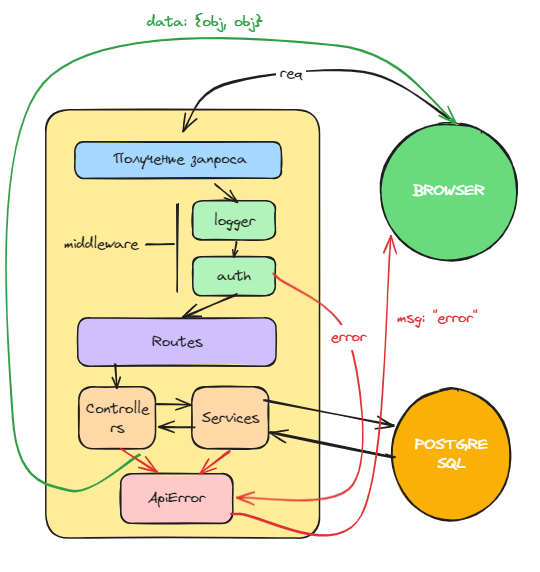


Рисунок 1 – Архитектура серверной части приложения

Next.js 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).

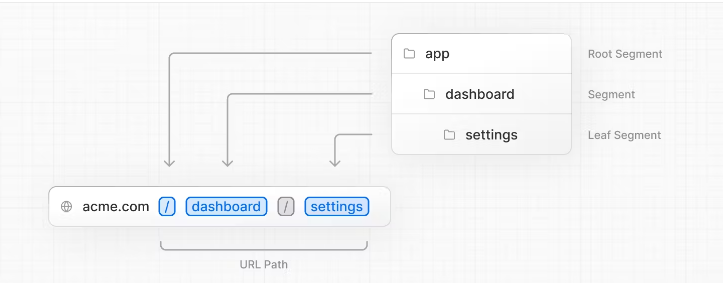


Рисунок 2 – App routing в Next

## 3.2. 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).

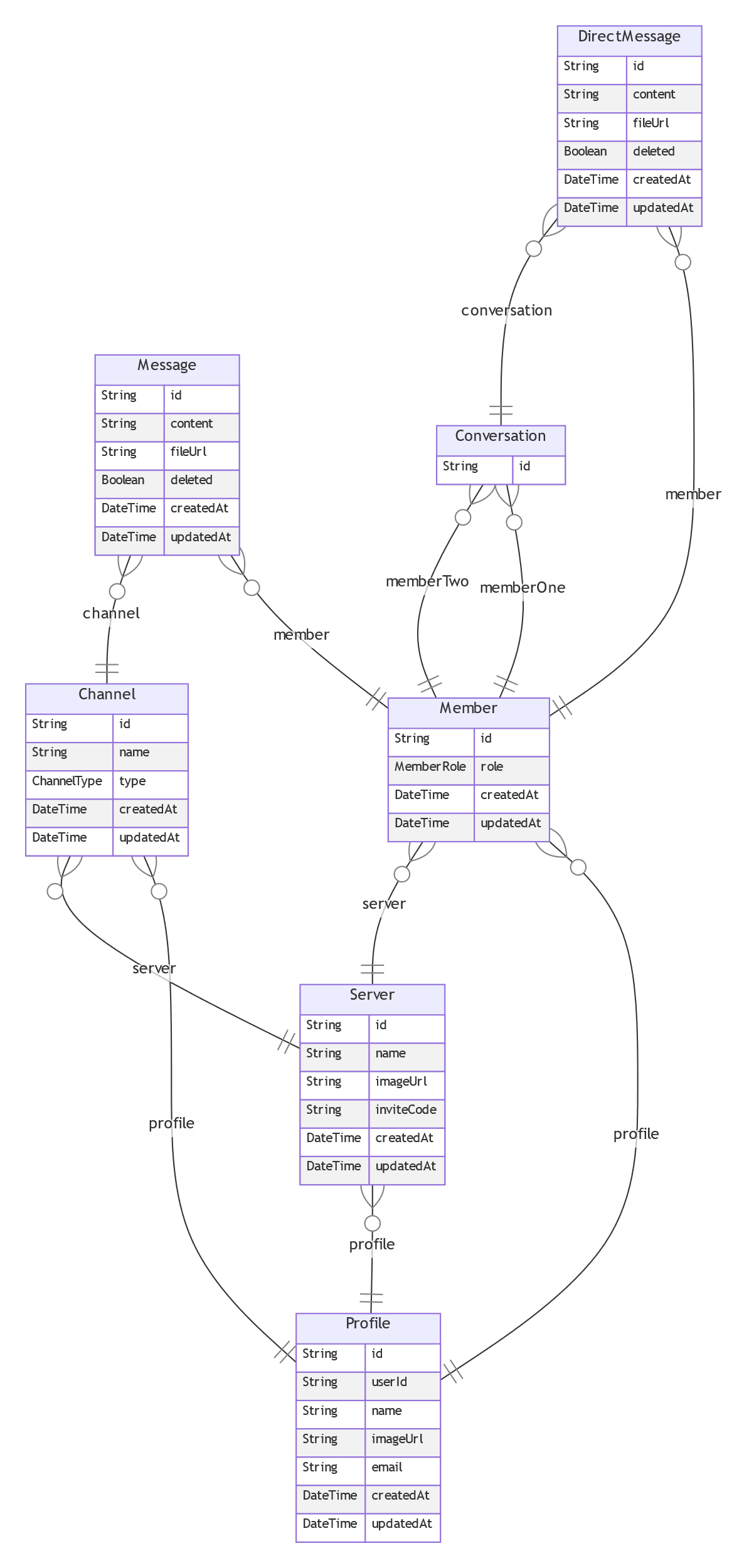


Рисунок 3 – Схема базы данных

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

* Profile: представляет пользовательский профиль с уникальным идентификатором, именем, изображением профиля, адресом электронной почты, а также временными метками создания и обновления. Каждый профиль связан с серверами, участниками и каналами;
* Server: определяет сервер с уникальным идентификатором, названием, изображением, уникальным кодом приглашения и временными метками создания и обновления. Связан с профилем, участниками и каналами;
* Member: представляет участника сервера с уникальным идентификатором, ролью (администратор, модератор или гость), временными метками создания и обновления. Связан с профилем, сервером, сообщениями и прямыми сообщениями;
* Channel: описывает канал с уникальным идентификатором, названием, типом (текстовый, аудио или видео) и временными метками создания и обновления. Связан с профилем, сервером и сообщениями;
* Message: представляет сообщение с уникальным идентификатором, текстовым содержанием, URL-ссылкой на файл (при наличии), а также временными метками создания и обновления. Связано с участником и каналом;
* Conversation: определяет беседу между двумя участниками с уникальным идентификатором и временными метками создания и обновления. Связана с двумя участниками и прямыми сообщениями;
* DirectMessage: представляет прямое сообщение с уникальным идентификатором, текстовым содержанием, URL-ссылкой на файл (при наличии) и временными метками создания и обновления. Связано с участником и беседой.

## 3.3. Авторизация

Процесс разработки системы авторизации с использованием Next.js и NextAuth, который представляет собой универсальное средство аутентификации для web-приложений. Разработанная система позволяет пользователям входить в систему с использованием различных методов аутентификации, таких как учетные записи Google, GitHub, а также стандартная аутентификация по учетным данным.

В ходе разработки были использованы следующие ключевые технологии:

* Next.js: фреймворк React для создания мощных web-приложений с использованием серверного рендеринга и статической генерации;
* NextAuth: библиотека, предоставляющая гибкие средства аутентификации для web-приложений на базе Next.js. NextAuth интегрируется с различными провайдерами аутентификации, такими как Google, GitHub, а также предоставляет возможность настройки собственной аутентификации через учетные данные [5];
* Zod: библиотека для проверки и валидации данных в TypeScript. Используется для валидации форм входа и других данных, отправляемых на сервер;
* Bcryptjs: библиотека для хэширования паролей. Применяется для безопасного хранения паролей пользователей в базе данных.

Система авторизации разработана с учетом следующих основных функциональных требований:

* регистрация и аутентификация пользователей с использованием различных провайдеров, таких как Google, GitHub, а также по учетным данным (email и пароль);
* валидация вводимых пользователем данных на стороне клиента и сервера;
* защита маршрутов, доступ к которым требует аутентификации пользователя;
* проверка подлинности пользовательских данных перед входом в систему;
* перенаправление пользователей после успешной аутентификации.

В ходе работы были реализованы и протестированы серверные и клиентские функции, обеспечивающие безопасность и удобство использования системы авторизации. Полученный результат демонстрирует эффективное использование технологий Next.js и NextAuth для создания безопасных и масштабируемых web-приложений с системой аутентификации.

# ЗАКЛЮЧЕНИЕ

В ходе работы над данным проектом приобрелся ценный опыт в разработке web-приложений, используя Node.js, Express, React, Next, Prisma ORM и TypeScript. Была осознана важность обеспечения безопасности при работе с файлами, сообщениями и применении соответствующих мер для защиты данных пользователей. Также была реализована продвинутая авторизация с возможность восстановления пароля, подтверждением адреса электронной почты, двухфакторной аутентификацией и входом через различные сервисы.

За период практики были приобретены следующие компетенции (таблица 1):

Таблица 1 – Таблица компетенций

|  |  |  |
| --- | --- | --- |
| Шифр компетенции | Расшифровка приобретаемой компетенции | Описание приобретенных знаний, умений и навыков |
| УК-1 | Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни | Научился составлять план выполнения заданий на день |
| ОПК-1 | Проверка работоспособности и рефакторинг кода программного обеспечения | Научился тестировать различные модули приложения и проводить рефакторинг кода |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Шифр компетенции | Расшифровка приобретаемой компетенции | Описание приобретенных знаний, умений и навыков |
| ОПК-2 | Интеграция программных модулей и компонент и верификация выпусков программного продукта | Научился проводить интеграцию программных модулей, а также различных частей приложения |
| ОПК-3 | Разработка требований и проектирование программного обеспечения | Научился определять требования и создавать эффективные решения в области программной разработки |
| ОПК-4 | Оценка и выбор варианта архитектуры программного средства | Научился оценивать имеющиеся архитектурные решения и принимать обоснованный выбор в пользу оптимального варианта для конкретного проекта |

# 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 validatedFileds = LoginSchema.safeParse(credentials)

if (validatedFileds.success) {

const { email, password } = validatedFileds.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