МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Web-приложение турагентство для заказа туров в разные страны»

Выполнил студент Дмитрук Илья Игоревич

(Ф.И.О.)

Руководитель проекта преп.–ст. Некрасова А.П.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доцент Смелов В.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой

Минск 2024

**Содержание**

[Введение 6](#_Toc186442304)

[1 Постановка задачи и обзор аналогичных решений 7](#_Toc186442305)

[1.1 Постановка задачи 7](#_Toc186442306)

[1.2 Аналитический обзор аналогов 7](#_Toc186442307)

[1.2.1 Мессенджер telegram 7](#_Toc186442308)

[1.2.2 Мессенджер whatsApp 8](#_Toc186442309)

[1.2.3 Мессенджер viber 9](#_Toc186442310)

[1.2 Выводы 10](#_Toc186442311)

[2 Проектирование веб–приложения 12](#_Toc186442312)

[2.1 Функциональные возможности. 12](#_Toc186442313)

[2.2 Проектирование базы данных. 16](#_Toc186442314)

[2.2 Выводы 20](#_Toc186442315)

[3 Кодирование веб–приложения 21](#_Toc186442316)

[3.1 Архитектура веб–приложения 21](#_Toc186442317)

[3.2 ORM и база данных 22](#_Toc186442318)

[3.3 Реализация функциональности 25](#_Toc186442319)

[3.3.1 Реализация метода авторизации 25](#_Toc186442320)

[3.4.2 Реализация метода регистрации 25](#_Toc186442321)

[3.4.3 Реализация метода получения актуальных токенов 26](#_Toc186442322)

[3.4.4 Реализация метода выхода из аккаунта 27](#_Toc186442323)

[3.4.5 Реализация метода создания чатов 27](#_Toc186442324)

[3.4.6 Реализация метода выхода из чата 28](#_Toc186442325)

[3.4.7 Реализация метода получения списка участников чата 29](#_Toc186442326)

[3.4.8 Реализация метода ввода шифр ключа 29](#_Toc186442327)

[3.4.9 Реализация метода просмотра чатов и сообщений 30](#_Toc186442328)

[3.4.10 Реализация метода отправки сообщения 31](#_Toc186442329)

[3.4.11 Реализация метода удаления сообщений 32](#_Toc186442330)

[3.4.12 Реализация метода редактирования профиля 33](#_Toc186442331)

[3.4.13 Реализация метода перевода сообщения 33](#_Toc186442332)

[3.4.14 Реализация метода поиска чата 34](#_Toc186442333)

[3.4.15 Реализация метода установки чата в закрепленные 34](#_Toc186442334)

[3.4.16 Реализация метода обноваления имени чата 35](#_Toc186442335)

[3.4.17 Реализация метода добавления пользователя в чат 35](#_Toc186442336)

[3.4.18 Реализация метода отправки сообщения пользователю 36](#_Toc186442337)

[3.4.19 Реализация метода назначения роли и добавления участника чата 37](#_Toc186442338)

[3.4.20 Реализация метода Encrypt 37](#_Toc186442339)

[3.4.21 Реализация метода Decrypt 38](#_Toc186442340)

[3.5 Архитектурный подход 38](#_Toc186442341)

[3.6 Используемые форматы передачи данных 40](#_Toc186442342)

[3.7 Сторонние сервисы 40](#_Toc186442343)

[3.8 Выводы 40](#_Toc186442344)

[4 Тестирование веб–приложения 42](#_Toc186442345)

[4.1 Функциональное тестирование 42](#_Toc186442346)

[4.2 Автоматизированное тестирование 43](#_Toc186442347)

[4.3 Выводы 44](#_Toc186442348)

[5 Руководство пользователя 45](#_Toc186442349)

[5.1 Руководство пользователя 45](#_Toc186442350)

[5.2 Установка веб–приложения 50](#_Toc186442351)

[5.3 Выводы по разделу 50](#_Toc186442352)

[Заключение 51](#_Toc186442353)

[Список используемых источников 52](#_Toc186442354)

[Приложение А 53](#_Toc186442355)

[Приложение Б 54](#_Toc186442356)

[Приложение В 58](#_Toc186442357)

[Приложение Г 63](#_Toc186442358)

# Введение

Проект посвящён разработке веб-приложения, которое позволит пользователям заказывать туры в разные города и страны. Основная идея веб-приложения заключается в предоставлении туров для бронирования, включающих в себя бронь транспорта и отеля в определённом городе на определённую дату. Всё что включено в тур, будет забронировано вместе с ним.

Целью проекта является упрощение организации путешествий за счёт единовременного бронирования всего того, что необходимо для отправки в путешествия в разные города.

Задачи, поставленные для достижения целей, включают:

* постановку задач и аналитический обзор решений (раздел 1);
* проектирование веб-приложения (раздел 2);
* реализация веб-приложения (раздел 3);
* тестирование веб-приложения (раздел 4);
* создание руководства пользователя (раздел 5).

Веб-приложение предназначено для различных людей, которые любят путешествовать, например, любителям отдохнуть на море, любителям посмотреть на культурные и исторические объекты разных народов, любителям увидеть разнообразную природу различных мест нашей планеты, да и просто любителям узнать что-то новое, а также людям, которым необходимо куда-то отправиться по работе или по личным причинам.

Для разработки веб-приложения была выбрана монолитная архитектура. Серверная часть реализована на платформе ASP.NET Core, клиентская часть — с использованием библиотеки React. В качестве СУБД используется MariaDB, с подключением через ORM Entity Framework Core. Для передачи данных используется REST API. Приложение развёрнуто в Docker.

Процесс разработки включал обзор существующих решений для формирования оптимального набора функций, проектирование архитектуры приложения, реализацию серверной и клиентской частей, а также тестирование для проверки функциональности и стабильности.

# 1 Постановка задачи и обзор аналогичных решений

## **Постановка задачи**

Приложение предназначено для общения пользователей, в функциональные возможности которого входит сквозное шифрование сообщений, перевод сообщений, создание чатов, приглашение пользователей в данные чаты и назначение им ролей с правами управления данным чатом.

## 1.**2 Аналитический обзор аналогов**

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных аналогов были сформулированы требования к данному программному средству.

### 1.2.1 Мессенджер telegram

Telegram сочетает в себе широкий спектр возможностей и простоту использования. Он позволяет пользователям отправлять текстовые и голосовые сообщения, совершать видеозвонки, участвовать в групповых чатах, а также обмениваться файлами и мультимедиа.

Одной из ключевых особенностей Telegram является поддержка секретных чатов, где сообщения автоматически удаляются через определённое время, что особенно привлекательно для тех, кто ценит конфиденциальность.

Интерфейс Telegram выделяется своей минималистичностью и удобством. Все ключевые элементы управления расположены так, чтобы обеспечить максимальную доступность функций. Нейтральный дизайн, который позволяет сосредоточиться на содержании, не отвлекаясь на лишние детали.

Чаты организованы так, что пользователи всегда видят актуальную информацию о переписке, включая последнее сообщение и фото профиля собеседника.

Скорость и надёжность работы являются одними из главных достоинств Telegram. Сообщения доставляются практически мгновенно. Эта особенность делает его подходящим инструментом как для личного общения, так и для корпоративного взаимодействия которая должны быть и в Cryptogram. Однако, некоторые возможности Telegram, такие как поддержка ботов и огромный выбор натсроек, могут быть избыточными для приложений, сосредоточенных исключительно на персональном и защищённом общении.

Cryptogram, как современный мессенджер, ориентируется на внедрение ключевых достоинств безопастности Telegram, но с акцентом на собственные уникальные особенности. Например, использование клиентского шифрования которым пользователи могут сами управлять, внедряя собственные ключи для защиты сообщений. Функция перевода сообщений в Cryptogram открывает новые возможности для международного общения, что делает приложение привлекательным для пользователей из разных стран. Простота интерфейса и минималистичный подход в дизайне помогают сосредоточиться на главном, удобном и безопасном общении.Интерфейс приложения представлен на рисунке 1.1.

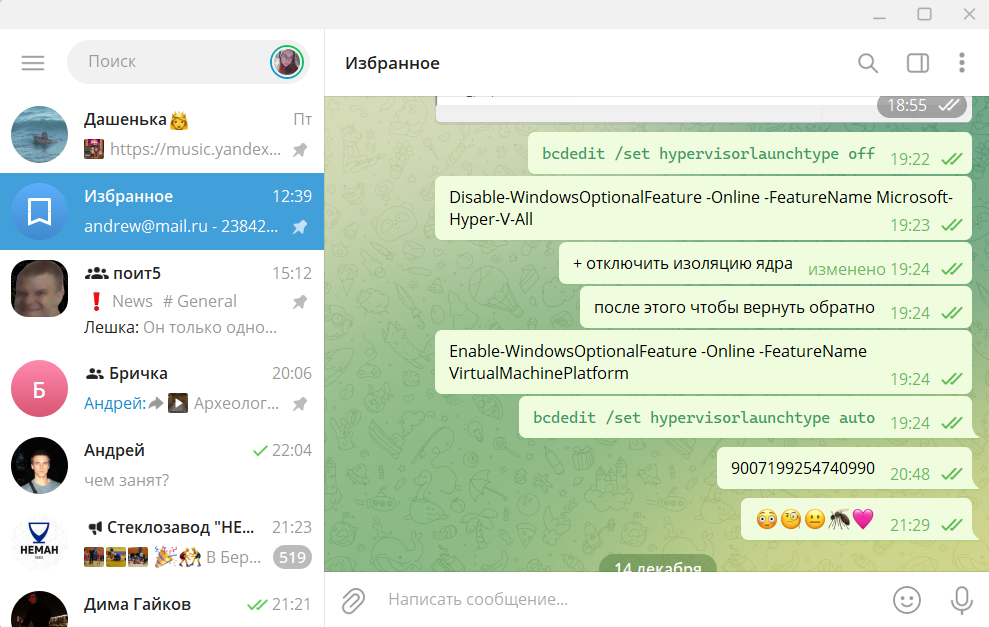


Рисунок 1.1 – Приложение «Telegram»

### 1.2.2 Мессенджер whatsApp

WhatsApp – это популярный мессенджер, известный своей простотой и доступностью. Он предоставляет функции текстового общения, голосовых и видеозвонков, обмена файлами и создания групповых чатов. Интерфейс выполнен в минималистичном стиле, с акцентом на удобство использования. Зеленые и белые тона создают уютную атмосферу, а основные функции всегда находятся под рукой. Одной из заметных особенностей является возможность удалять и редактировать отправленные сообщения, что помогает пользователям корректировать ошибки или удалять нежелательные сообщения.

Тем не менее, WhatsApp имеет свои ограничения. Например, несмотря на наличие сквозного шифрования, вопрос о полной безопасности данных возникает при резервном копировании чатов в облаке. Также отсутствуют продвинутые функции для улучшения международных коммуникаций, такие как встроенный перевод сообщений. Кроме того, WhatsApp не поддерживает клиента с серверным шифрованием, что может вызвать опасения у пользователей, ориентированных на высокую степень безопасности.

В Cryptogram можно учесть сильные стороны WhatsApp, такие как удобство интерфейса и функции редактирования сообщений. Однако, в отличие от WhatsApp, Cryptogram может предложить улучшенные возможности для обеспечения безопасности данных с клиентским шифрованием, что исключает доступ к данным со стороны сторонних серверов. Также, интеграция с сервисами перевода сообщений в реальном времени станет отличной альтернативой отсутствующим возможностям WhatsApp для международных коммуникаций. Cryptogram предложит более гибкую систему настройки прав доступа и более широкий набор возможностей для управления чатом, что сделает его более привлекательным для пользователей, ценящих приватность и функциональные возможности.Интерфейс приложения представлен на рисунке 1.2

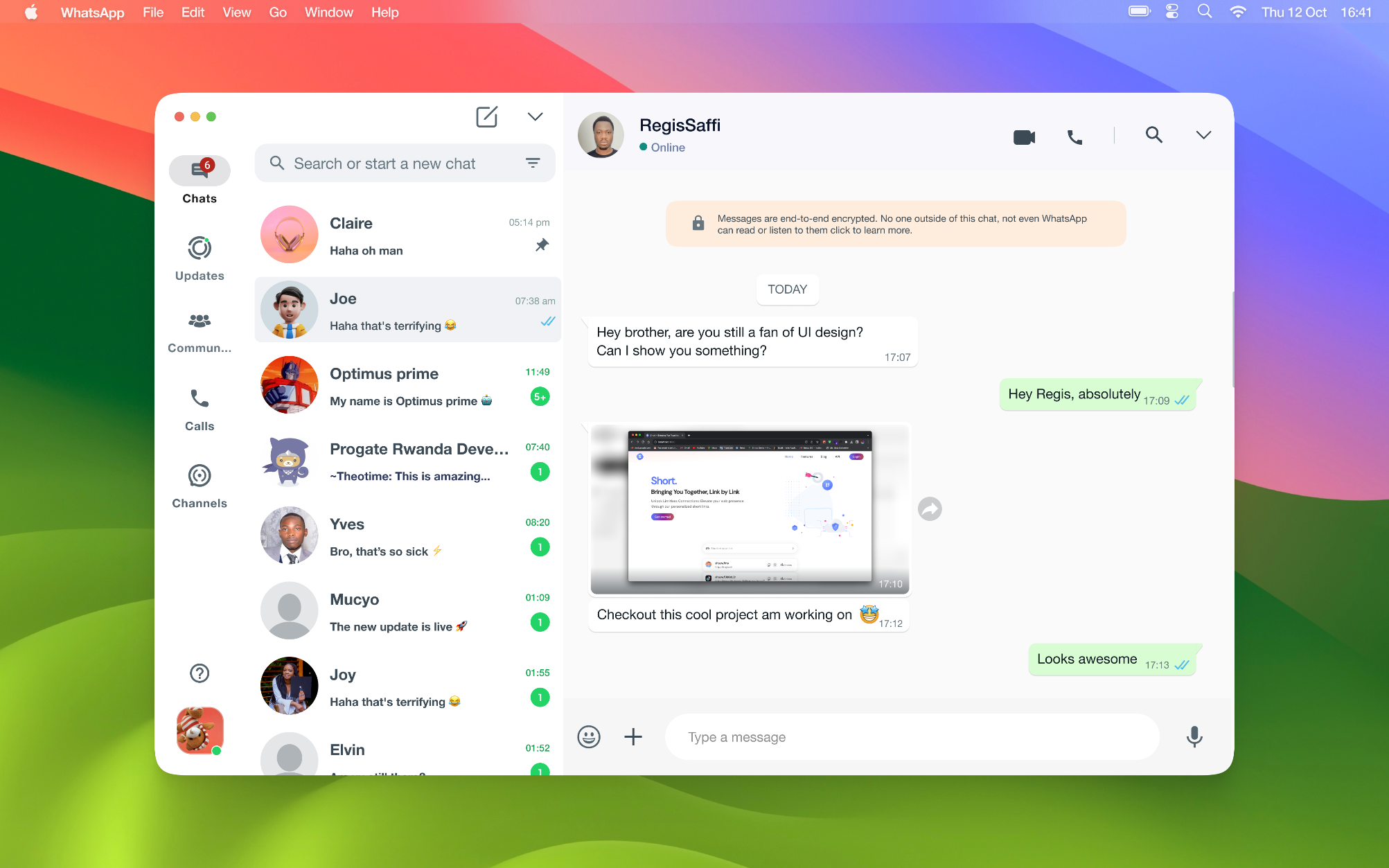


Рисунок 1.2 – Приложение «WhatsApp»

### 1.2.3 Мессенджер viber

Viber – популярный мессенджер, известный своей универсальностью и удобством использования. Он поддерживает обмен текстовыми сообщениями, голосовые и видеозвонки, отправку файлов и изображений, а также возможность создавать групповые чаты. Кроме того, Viber предлагает широкую коллекцию стикеров, которые позволяют пользователям выразить свои эмоции более живо и ярко.

Дизайн приложения отличается простотой и дружелюбностью. Пользовательский интерфейс ориентирован на максимальную доступность функций, при этом элементы управления интуитивно понятны. Основные инструменты сосредоточены в нижней части экрана, что упрощает навигацию.

Cryptogram, ориентированный на создание безопасного и минималистичного мессенджера, может использовать успешные элементы Viber, такие как удобный интерфейс и акцент на защите данных. Однако Cryptogram делает упор на уникальные особенности, например, клиентское шифрование, где ключи шифрования хранятся исключительно на устройствах пользователей.

Целевая аудитория Cryptogram – это пользователи, которым требуется защищённая среда общения. Сюда входят корпоративные команды, малые и средние предприятия, а также индивидуальные пользователи, ценящие приватность и минимализм. В отличие от Viber, Cryptogram исключает элементы, которые могут быть восприняты как отвлекающие, фокусируясь на удобстве, безопасности и скорости работы, что делает его идеальным решением для современного цифрового общения.Интерфейс приложения представлен на рисунке 1.3

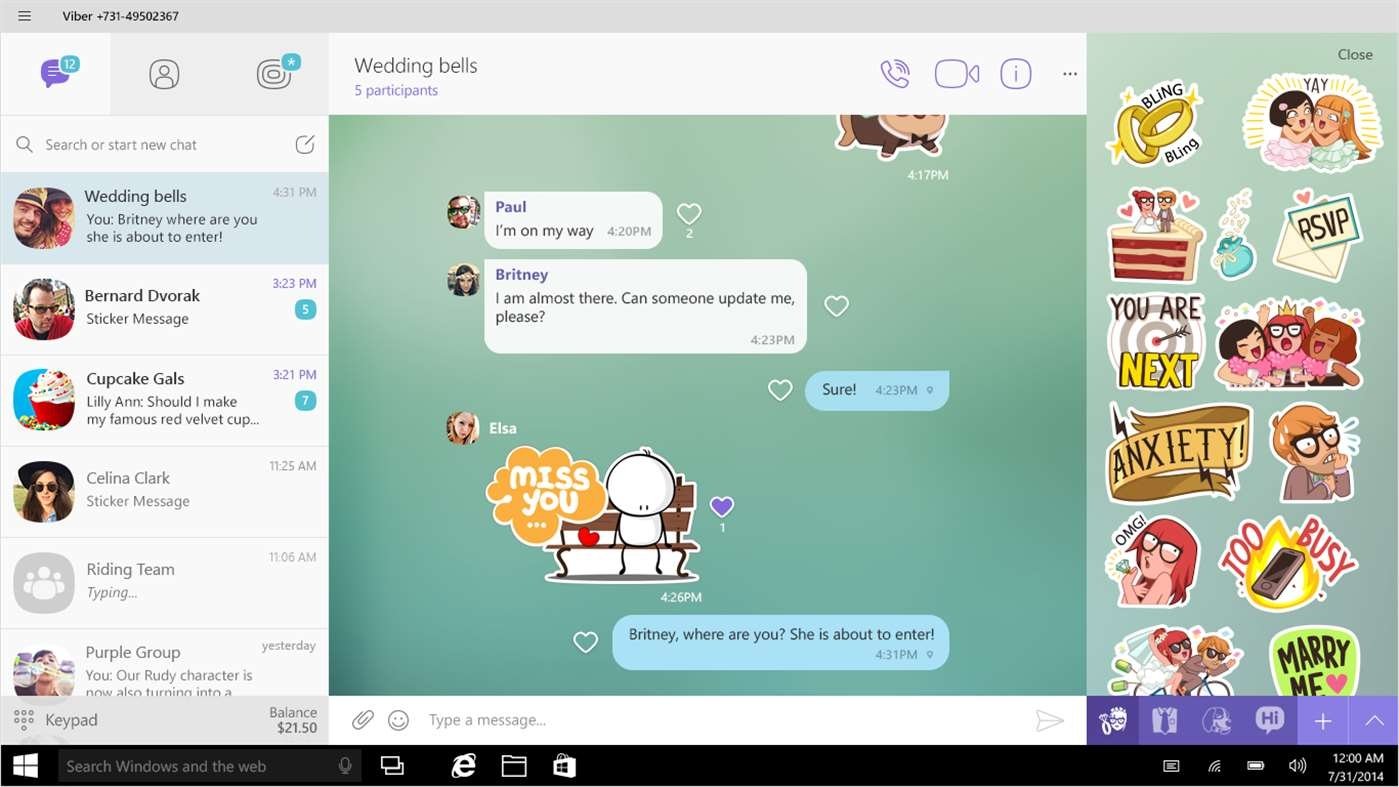


Рисунок 1.3 – Приложение «Viber»

## 1.2 Выводы

Рассмотрены аналогичные веб–приложения, которые стали источником идей для веб–приложения Cryptogram:

1. удобный интерфейс, а также возможность удаления и редактирования сообщений, как в Viber и Telegram;
2. высокий уровень безопасности, как в Telegram, за счет сквозного шифрования, но с особенностью, которая позволит пользователям самим выбирать свои ключи и хранить их только на своих устройствах;
3. возможность назначения ролей в чатах как во всех вышеуказанных мессенджерах;
4. возможность перевода сообщений, что выделяет Cryptogram среди других мессенджеров.
5. создание и управление групповыми чатами.

# 2 Проектирование веб–приложения

## 2.1 Функциональные возможности.

Диаграмма демонстрирует ключевые роли в системе и действия, доступные для каждой из них.

На диаграмме отражено, что гость может просматривать туры, а также регистрироваться и авторизоваться. Пользователь может бронировать туры и оставлять отзывы к турам. Менеджер может добавлять, изменять, удалять туры, а также принимать заявки на бронь туров. Администратор может выполнять функционал менеджера, а также удалять отзывы к турам и удалять, блокировать и изменять роль пользователей. Роли приложения представлены в таблице 2.1.

Таблица 2.1 – Описание ролей приложения

|  |  |
| --- | --- |
| Роль | Описание |
| Гость | Может регистрироваться и авторизоваться. Имеет доступ только к просмотру туров |
| Пользователь | Может бронировать туры и оставлять к ним отзывы |
| Менеджер | Может добавлять, удалять, редактировать туры, а также принимать заявки на туры. |
| Администратор | Может удалять отзывы к турам, а также удалять, блокировать пользователей и изменять им роль |

Описание функций ролей приложения представлены в таблице 2.2.

Таблица 2.2 – Описание функций

|  |  |  |
| --- | --- | --- |
| Функция | Роль | Описание |
| Регистрация | Гость | Позволяет создать учетную запись, указав email, пароль, повторный пароль, имя и имя пользоваетял. |
| Авторизация | Гость | Позволяет войти в систему с помощью email и пароля. |
| Выход из аккаунта | Пользователь | Позволяет пользователю выйти из своей учетной записи. |
| Создание чата | Пользователь | Дает возможность создавать новые чаты. |
| Выход из чата | Пользователь | Дает возможность выхода из чата. Если пользователь был последним участником чата то чат и сообщения удаляються |
| Просмотр чатов | Пользователь | Дает возможность просматривать список своих чатов. |
| Поиск чата | Пользователь | Дает возможность искать в своем списке чатов определенные чаты по названию или по части названия. |
| Редактирование чата | Пользователь | Дает возможность редактировать название чата, аватарку чата и удалять или добавлять участников в чат. |
| Ввод шифр ключа | Пользователь | Дает возможноть пользователю указать шифр ключ, который используется при шифровании сообщений в чате. |
| Просмотр списка сообщений | Пользователь | Дает возможность просматривать список сообщений определенного чата, который выбрал пользователь. |
| Отправка сообщений | Пользователь | Дает возможность отправлять текстовые сообщения. |
| Удаление сообщения | Пользователь | Дает возможность удалять сообщения, отправленные данным пользователем. |
| Редактирование сообщения | Пользователь | Дает возможность редактировать текстовые сообщения, отправленные данным пользователем. |
| Редактирование фотографии | Пользователь | Дает возможность изменить фотографию в своем профиле. |
| Редактирование Имени | Пользователь | Дает возможность изменить имя. |
| Редактирование Имени пользователя | Пользователь | Дает возможность изменить имя пользователя. |
| Редактирование пароля | Пользователь | Дает возможность изменить пароль пользователя. |

Роли чатов представлены в таблице 2.3.

Таблица 2.3 – Описание ролей чатов

|  |  |
| --- | --- |
| Роль | Описание |
| Владелец | Может читать, отправлять и переводить сообщения, добавлять в чат новых пользователей или удалять старых с ролью участник, читатель, участник+ или администратор. Может добавить другого владельца. Может изменять название и аватар чата. Может покинуть данный чат. |
| Администратор | Может читать, отправлять и переводить сообщения, добавлять в чат новых пользователей или удалять старых с ролью участник, читатель или участник+. Может изменять название и аватар чата. Может покинуть данный чат. |
| Участник+ | Может читать, отправлять и переводить сообщения, добавлять в чат пользователей с ролью участник или читатель. Может изменять название и аватар чата. Может покинуть данный чат. |
| Участник | Может читать, отправлять и переводить сообщения. Может покинуть данный чат. |
| Читатель | Может читать и переводить сообщения. Может покинуть чат. |

Описание функций ролей чатов представлены в таблице 2.4.

Таблица 2.4 – Описание функций участников чата

|  |  |  |
| --- | --- | --- |
| Функция | Роль | Описание |
| Отправка сообщений | Владелец, администратор, участник+ и участник | Позволяет отправлять сообщение в чат |
| Удаление сообщений | Владелец, администратор, участник+ и участник | Позволяет удалять собственные сообщения из чата |
| Редактирование сообщений | Владелец, администратор, участник+ и участник | Позволяет редактировать собственные уже отправленные сообщения |
| Добавления пользователей в чат | Владелец, администратор и участник+ | Позволять добавлять пользователей в чат. |
| Удаление пользователя из чата | Владелец и администратор | Позволяет удалять пользователей из чата |
| Перевод сообщений | Владелец, администратор, участник+, участник и читатель | Позволяет переводить сообщения |
| Редактирование названия чата | Владелец, администратор и участник+ | Позволяет изменять название чата |
| Редактирование аватарки чата | Владелец, администратор, участник+ и участник | Позволяет изменять аватарку чата |
| Установка чатов в закрепленные | Владелец, администратор, участник+, участник и читатель | Позволяет закреплять чаты чтобы они были сверху в списке чатов |

Диаграмма вариантов использования представленнна на рисунке 2.1

|  |
| --- |
|  |

Рисунок 2.1 – Диаграмма вариантов использования

## 2.2 Проектирование базы данных.

Диаграмма базы данных – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма логической схемы базы данных представлена на рисунке 2.2.

|  |
| --- |
|  |

Рисунок 2.2 – Логическая схема базы данных

Для веб–приложения была спроектирована база данных, включающая **6 таблиц**, обеспечивающих надежное хранение информации о пользователях, чатах, участников данных чатов их сообщения и информацию необходимую для логики стикеров. В таблице 2.5 показано описание таблиц базы данных

Таблица 2.5 – Описание таблиц базы данных

|  |  |
| --- | --- |
| Название таблицы | Описание таблицы |
| Users | Хранит данные о пользователях. |
| Chats | Хранит данные о чатах. |
| ChatMembers | Описывает связь между пользователями и их участием в чате. А также хранит данные о конкретном участнике конкретного чата. |
| Messages | Хранит данные сообщений. |
| StickerGroup | Хранит данные о группе стикеров. |
| Stickers | Хранит данные стикеров. |

В таблице 2.6 показано описание полей таблицы «Users».

Таблица 2.6– Описание структуры таблицы «Users»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| UserId | TEXT | PRIMARY KEY | Уникальный идентификатор пользователя. |
| Name | TEXT | NOT NULL | Имя пользователя. |
| AvatarPath | TEXT | NOT NULL DEFAULT '/static/defaults/userAvatars /defaultUserAvatar.jpg' | Путь к аватару пользователя на сервере |
| UserName | TEXT | NOT NULL, UNIQUE | Учетное имя пользователя |
| Email | TEXT | NOT NULL, UNIQUE | Электронная почта пользователя. |
| Language | TEXT | NOT NULL DEFAULT 'ru' | Язык пользователя. |
| Role | INTEGER | NOT NULL DEFAULT 1 | Роль пользователя в приложении |
| PasswordHash | TEXT | NOT NULL | Хэшированный пароль пользователя. |
| CreatedAt | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Дата и время создания пользователя. |
| UpdatedAt | TIMESTAMP | NOT NULL DEFAULT CURRENT\_TIMESTAMP | Дата и время изменения пользователя. |

В таблице 2.7 показано описание полей таблицы «Chats».

Таблица 2.7 – Описание структуры таблицы «Chats»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| ChatId | SERIAL | PRIMARY KEY | Уникальный идентификатор чата. |
| ChatName | TEXT | NOT NULL | Название чата. |
| IsGroup | BOOLEAN | NOT NULL | Параметр, отвечающий за определение личного или групового чата. |
| KeyHash | TEXT | NOT NULL | Хэш ключа чата. |
| AvatarPath | TEXT | NOT NULL DEFAULT '/static/defaults/chatAvatar /defaultChatAvatar .png' | Путь к аватарке чата на сервере. |
| CreatedAt | TIMESTAMP | NOT NULL DEFAULT CURRENT\_TIMESTAMP | Дата и время создания чата. |
| UpdatedAt | TIMESTAMP | NOT NULL DEFAULT CURRENT\_TIMESTAMP | Дата и время изменения чата. |

В таблице 2.8 представлена структура таблицы «ChatMembers».

Таблица 2.8 – Описание структуры таблицы «ChatMembers»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| ChatMemberId | SERIAL | PRIMARY KEY | Уникальный идентификатор участника чата. |
| ChatId | INTEGER | NOT NULL, FOREIGN KEY | Идентификатор чата (ссылается на таблицу Chats). |
| UserId | TEXT | NOT NULL, FOREIGN KEY | Идентификатор пользователя (ссылается на таблицу Users). |
| Role | INTEGER | NOT NULL | Роль пользователя. |
| IsFixed | BOOLEAN | NOT NULL | Параметр указывающий зафиксирован ли чат. |
| JoinedAt | TIMESTAMP | NOT NULL DEFAULT CURRENT\_  TIMESTAMP | Дата и время, в которое был добавлен данный пользователь. |

В таблице 2.9 представлена структура таблицы «Messages».

Таблицы 2.9 – Описание структуры таблицы «Messages»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| MessageId | SERIAL | PRIMARY KEY | Уникальный идентификатор сообщения. |
| SenderId | TEXT | NOT NULL, FOREIGN KEY | Идентификатор пользователя, который отправил сообщение (ссылается на таблицу Users). |
| ChatId | INTEGER | NOT NULL, FOREIGN KEY | Идентификатор чата, в который было отправлено сообщение (ссылается на таблицу Chats). |
| Content | TEXT | NOT NULL | Содержимое отправленного пользователем сообщения. |
| MessageType | TEXT | NOT NULL | Тип сообщения (например, текстовое). |
| IsUpdate | BOOLEAN | NOT NULL DEFAULT false | Параметр указывающий было ли изменено сообщение. |
| IsRead | BOOLEAN | NOT NULL DEFAULT false | Параметр указывающий было ли прочитанно сообщение. |
| TimeCreated | TIMESTAMP | NOT NULL | Время создания сообщения. |
| TimeUpdated | TIMESTAMP | NULLABLE | Время последнего обновления сообщения. |

В таблице 2.10 представлена структура таблицы «Stickers».

Таблица 2.10 – Описание структуры таблицы «Stickers»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| StickerId | SERIAL | PRIMARY KEY | Уникальный идентификатор стикера. |
| StickerGroupId | INTEGER | NOT NULL | Идентификатор группы стикеров (ссылается на таблицу StickerGroups). |
| Path | TEXT | NOT NULL | Путь к изображению стикера на сервере.. |

В таблице 2.11 представлена структура таблицы «StickerGroups».

Таблица 2.11 – Описание структуры таблицы «StickerGroups»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| StickerGroupId | SERIAL | PRIMARY KEY | Уникальный идентификатор группы стикеров. |
| GroupName | TEXT | NOT NULL | Название группы стикеров |

В таблице 2.12 описаны связи сущностей.

Таблица 2.12 – Описание связей сущностей

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 1 | Поле | Таблица 2 | Поле | Тип связи | Описание |
| Chats | СhatId | ChatMembers | ChatId | Один–ко–многим | Один чат может включать нескольких участников. |
| Users | UserId | ChatMembers | ChatMemberId | Один–ко–многим | Один пользователь может участвовать в нескольких чатах. |
| Chats | ChatId | Messages | ChatId | Один–ко–многим | Один чат может содержать множество сообщений. |
| Users | UserId | Messages | SenderId | Один–ко–многим | Один пользователь может отправить множество сообщений. |
| Stickers | StickerId | StickerGroups | StickerGroupId | Один–ко–многим | Одна группа может содерать множество стикеров. |

## 2.2 Выводы

Рассмотрены основные роли веб–приложения:

1. гость и пользователь – ключевые роли с базовыми функциями и взаимодействием с системой;
2. 5 вспомогательных ролей для чатов, которые определяют доступные операции в зависимости от статуса пользователя в чате;
3. описаны основные функции, которые доступны каждой роли.

Представлена структура базы данных, включающая 6 ключевых таблиц:

1. Users хранит информацию о пользователях;
2. Chats хранит данные о чатах;
3. ChatMembers отслеживает участников чатов и их роли;
4. Messages отвечает за хранение сообщений;
5. Stickers содержит данные о стикерах;
6. StickerGroups группирует стикеры для удобства их использования.

Рассмотрены связи между таблицами, которые обеспечивают целостность данных и их корректную обработку.

# 3 Кодирование веб–приложения

## 3.1 Архитектура веб–приложения

Система представляет собой архитектуру, включающую серверную и клиентскую части, а также базы данных, развёрнутые в контейнере Docker. Диаграмма развёртывания приведена в приложении А.

Серверная часть приложения реализована на платформе ASP .NET CORE[1] версии XXX. Серверная часть обрабатывает запросы HTTP/1.1. Серверная часть развёрнута в Docker–контейнере на операционной системе Linux Ubuntu 22.04 LTS.

Для работы с базой данных используется ORM Entity Framework версии xxx [3] и сама СУБД MariaDB [4]. MariaDB также развёрнута в Docker–контейнере, что упрощает её интеграцию в общую архитектуру приложения.

Клиентская часть построена на React версии 18.3.1 [5]. Этот фреймворк позволяет создать отзывчивый и удобный интерфейс, оптимизированный для разных устройств. Для сборки проекта используется сборщик WebPack [6]. Визуальный стиль приложения разработан с использованием Scss [8]. Важным компонентом системы является Docker Compose [9]. Для отправки HTTP-запросов использовалась библиотека Axios [10].

Таким образом, архитектура приложения объединяет современные технологии и проверенные решения, обеспечивая стабильность, масштабируемость за счет модульго подхода, безопасность, быстрый доступ к полям таблиц в базе данных и высокую производительность в процессе работы.

## 3.2 ORM и база данных

Для управления данными используется Prisma ORM, что обеспечивает удобство работы с базой данных и строгую типизацию.

В основе приложения лежит модель Users, которая представляет данные пользователей. Каждый пользователь имеет уникальный идентификатор UserId, сгенерированный с помощью cuid(), и уникальный UserName, который служит для идентификации пользователя в системе. Помимо этого, модель включает в себя такие поля, как Name для хранения имени, AvatarPath для управления аватаром пользователя, Email для электронной почты, которая также должна быть уникальной, и PasswordHash, обеспечивающий безопасное хранение паролей. Для отслеживания активности пользователя предусмотрены временные метки CreatedAt и UpdatedAt, которые фиксируют дату создания и последнего обновления учетной записи. Для реализации многоязычности в приложении используется поле Language, которое по умолчанию имеет значение «ru». В листинге 3.1 представлено описание модели «User».

model Users {

  UserId       String        @id @default(cuid())

  Name         String

  AvatarPath   String        @default("/static/defaults/userAvatars/defaultUserAvatar.jpg")

  UserName     String        @unique

  Email        String        @unique

  Language     String        @default("ru")

  Role         Int           @default(1)

  PasswordHash String

  CreatedAt    DateTime      @default(now())

  UpdatedAt    DateTime      @default(now()) @updatedAt

  UserMessages Messages[]

  ChatMembers  ChatMembers[]

}

Листинг 3.1 – Модель «Users»

Система чатов представлена моделью Chats, которая поддерживает как групповые чаты, так и личные. Основной идентификатор чата, ChatId, генерируется автоматически. Каждый чат имеет название ChatName, определяющее его идентичность, а также флаг IsGroup, указывающий на тип чата (групповой или личный). Для безопасности предусмотрено поле KeyHash, которое может использоваться для шифрования данных или управления доступом. Для каждого чата также хранится путь к аватару AvatarPath, а временные метки CreatedAt и UpdatedAt фиксируют изменения, связанные с чатом, например создание чата или изменение его имени. Описание модели «Chats» представлено в листинге 3.2.

model Chats {

  ChatId       Int           @id @default(autoincrement())

  ChatName     String

  IsGroup      Boolean

  KeyHash      String

  AvatarPath   String        @default("/static/defaults/chatAvatars/ defaultChatAvatar.png")

  CreatedAt    DateTime      @default(now())

  UpdatedAt    DateTime      @default(now()) @updatedAt

  ChatMessages Messages[]

  ChatMembers  ChatMembers[]

}

Листинг 3.2 – Модель «Chats»

Модель Messages отвечает за хранение сообщений, которые могут быть текстовыми. Каждое сообщение связано с конкретным чатом через идентификатор ChatId и с участниками чатов с помощью поля SenderId которые предстваляют каждого пользователя в чате. Помимо основного текста сообщения, хранимого в поле Content, модель поддерживает флаг IsUpdate, указывающий, было ли сообщение изменено, и IsRead, который фиксирует статус прочтения. Это позволяет системе эффективно управлять коммуникацией между пользователями. Описание модели «Messages» представлено в листинге 3.3.

model Messages {

  MessageId   Int      @id @default(autoincrement())

  ChatId      Int

  SenderId    String

  Chat        Chats    @relation(fields: [ChatId], references: [ChatId])

  Sender      Users    @relation(fields: [SenderId], references: [UserId])

  Content     String

  MessageType String

  IsUpdate    Boolean  @default(false)

  IsRead      Boolean  @default(false)

  CreatedAt   DateTime @default(now())

  UpdatedAt   DateTime @default(now()) @updatedAt

}

Листинг 3.3 – Модель «Messages»

Управление участниками чатов реализовано через модель ChatMembers, которая связывает пользователей с чатами. В этой модели определяется роль участника, задаваемая через поле Role, а также фиксируется дата присоединения пользователя к чату в поле JoinedAt. Дополнительно поддерживается флаг IsFixed, который может использоваться для закрепления участника в определённом чате. Описание модели «ChatMembers» представлено в листинге 3.4.

model ChatMembers {

  ChatMemberId Int      @id @default(autoincrement())

  UserId       String

  Role         Int

  ChatId       Int

  IsFixed      Boolean  @default(false)

  JoinedAt     DateTime @default(now())

  Chat         Chats    @relation(fields: [ChatId], references: [ChatId])

  Member       Users    @relation(fields: [UserId], references: [UserId])

}

Листинг 3.4 – Модель «ChatMembers»

Поддержка мультимедиа реализована через модели StickerGroup и Stickers, которые отвечают за управление коллекциями стикеров. Группы стикеров, такие как наборы смайликов или уникальные коллекции, определяются моделью StickerGroup. Конкретные стикеры привязаны к группам через идентификатор StickerGroupId. Описание модели представлено в листинге 3.5.

model StickerGroup {

  StickerGroupId Int        @id @default(autoincrement())

  GroupName      String

  Stickers       Stickers[]

}

model Stickers {

  StickerId      Int          @id @default(autoincrement())

  StickerGroupId Int

  StickerGroup   StickerGroup @relation(fields: [StickerGroupId], references: [StickerGroupId])

}

Листинг 3.5 – Модель «StickerGroup и Stickers»

Таким образом, архитектура приложения сочетает в себе продуманное проектирование моделей данных, гибкость в управлении бизнес–логикой и использование современных инструментов для работы с базой данных. Это позволяет не только эффективно решать текущие задачи, но и обеспечивает простор для дальнейшего расширения и развития функциональности.

## 3.3 Реализация функциональности

### 3.3.1 Реализация метода авторизации

Метод «login» обрабатывает процесс входа пользователя в систему. Он начинает с вызова метода «ValidateUser», который проверяет, существует ли пользователь с указанными данными (например, с правильным паролем). После успешной проверки вызывается метод «IssueTokens» описанный чуть ниже в данном разделе для генерации токенов, которые связаны с идентификатором пользователя. В конце метод возвращает объект, содержащий идентификатор пользователя (userId) и сгенерированные токены для авторизации. Метод авторизации представлен в листинге 3.6.

async login(dto: AuthDto) {

const user = await this.ValidateUser(dto);

        const tokens = await this.IssueTokens(user.UserId);

        return {

            //user: this.ReturnUserFields(user),

            userId: user.UserId,

            ...tokens,

        };

    }

Листинг 3.6 – Метод авторизации

### 3.4.2 Реализация метода регистрации

Метод «register» отвечает за регистрацию нового пользователя. Сначала он вызывает метод «CheckUserExists», чтобы проверить, существует ли уже пользователь с таким же именем или email. Если пользователь существует, выполнение прерывается. Далее, если пароли не совпадают (сравниваются «password» и «repeatPassword»), выбрасывается исключение с сообщением «passwords must match». Если пароли совпадают, создается новый пользователь в базе данных с помощью метода Prisma «users.create», где сохраняются имя пользователя (UserName), имя (Name), email (Email) и хеш пароля (PasswordHash). Хеширование пароля происходит с помощью функции «hash». После успешного создания пользователя метод вызывает «IssueTokens», чтобы получить и выдать токены для авторизации. В итоге метод возвращает объект с идентификатором пользователя (userId) и сгенерированными токенами. Метод для регистрации представлен в листинге 3.7.

async register(dto: RegisterDto) {

        await this.CheckUserExists(dto);

        if (dto.password != dto.repeatPassword)

            throw new BadRequestException({

                error: true,

                show: true,

                message: 'passwords must match',

            });

        const user = await this.prisma.users.create({

            data: {

                UserName: dto.username,

                Name: dto.name,

                Email: dto.email,

                PasswordHash: await hash(dto.password),

            },

        });

        const tokens = await this.IssueTokens(user.UserId);

        return {

            //user: this.ReturnUserFields(user),

            userId: user.UserId,

            ...tokens,   }; }

Листинг 3.7 – Метод для регистрации

### 3.4.3 Реализация метода получения актуальных токенов

Метод «IssueTokens» генерирует три токена для пользователя: «accessToken», «refreshToken» и «wsToken». Все токены создаются с помощью библиотеки JWT, и в качестве данных для подписывания используется идентификатор пользователя (userId). Каждый токен имеет свой срок действия: «accessToken» действует 1 день и обновляется при истечении его срока годности, а «refreshToken» и «wsToken» – 7 дней. С помощью рефреш токена пользователь может получать новые токены доступа. Токен для вебсокета необходим для верификации клиента, когда тот подключается по вебсокету, иначе любой злоумышленик смог бы воспользоваться незащищенным каналом. Метод возвращает объект, содержащий все три токена, которые могут быть использованы для различных целей, например, для аутентификации пользователя или авторизации WebSocket–соединений. Метод для получения актуальных токенов представлен в листинге 3.8.

    private async IssueTokens(userId: string) {

        const data = { userId: userId };

        const accessToken = this.jwt.sign(data, {

            expiresIn: '1d',

        });

        const wsToken = this.jwt.sign(data, {

            expiresIn: '7d',

        });

        const refreshToken = this.jwt.sign(data, {

            expiresIn: '7d',

        });

        return { accessToken, refreshToken, wsToken };}

Листинг 3.8 – Метод для получения актуальных токенов

### 3.4.4 Реализация метода выхода из аккаунта

Метод Logout удаляет токены доступа и обновления, установленные в куках, с помощью пустых значений. Он также настраивает параметры безопасности для куков, такие как httpOnly, secure, и sameSite. После этого метод отправляет ответ с сообщением о успешном выходе из системы. Метод выхода из аккаунта представлен в листинге 3.9.

    @ApiOperation({ summary: 'logout in the app' })

    @Post('/logout')

    @HttpCode(200)

    async Logout(@Res() res: Response) {

        res.cookie('accessToken', '', {

            httpOnly: false,

            secure: false, // Рекомендуется для HTTPS

            sameSite: 'strict', // Защита от CSRF

        });

        res.cookie('refreshToken', '', {

            httpOnly: false,

            secure: false, // Рекомендуется для HTTPS

            sameSite: 'strict', // Защита от CSRF

        });

        return res.send({ message: 'Logout successfully' });    }

Листинг 3.9 – Метод выхода из аккаунта

### 3.4.5 Реализация метода создания чатов

Метод «CreateChat» создает новый чат и добавляет пользователей в него. Сначала он использует транзакцию Prisma для атомарности операции, чтобы все действия выполнялись одновременно. Внутри транзакции создается новый чат с использованием метода «prisma.chats.create», где сохраняются название чата (ChatName), флаг группы (IsGroup) и хеш ключа (KeyHash). Затем, в зависимости от типа чата, добавляются участники.

Если чат является группой (dto.isGroup), то текущий пользователь (userId) получает роль 1 и добавляется в чат. Если чат не является группой, то текущий пользователь и дополнительный пользователь (dto.userId) добавляются с ролью 4. После выполнения всех операций метод возвращает созданный чат. Метод для создания чатов в листинге 3.10.

async CreateChat(dto: CreateChatDto, userId: string) {

const chat = await prisma.$transaction(async (prisma) => {

const chat = await prisma.chats.create({ data: {

                    ChatName: dto.chatName,

                    IsGroup: dto.isGroup,

                    KeyHash: dto.keyHash,

},

});

           await prisma.chatMembers.create({

               data: {

                 UserId: dto.userId,

                 ChatId: chat.ChatId,

                 Role: 4,

},

});

}

Листинг 3.10 – Метод для создания чатов

### 3.4.6 Реализация метода выхода из чата

Он сначала валидирует возможность выхода с помощью метода ValidateLeaveFromChat. Затем, в рамках транзакции, удаляет пользователя из списка участников чата. Если после удаления участника в чате не остается других членов, метод удаляет все сообщения чата и сам чат. Если транзакция не удалась, выбрасывается исключение с ошибкой, которая вернется клиенту и отобращится на странице с ошибками. В конце метод возвращает сообщение о том, что пользователь покинул чат. Метод выхода из чата представлен в листинге 3.11.

    p async LeaveFromChat(dto: LeaveFromChatDto, userId: string) {

        const member = await this.ValidateLeaveFromChat(dto, userId);

        await this.prisma

            .$transaction(async (prisma) => {

                await prisma.chatMembers.delete({ where: {

                        ChatMemberId: member.ChatMemberId,},});

                const coutMembers = await prisma.chatMembers.count({

                    where: {

                        ChatId: member.ChatId, },  });

                if (coutMembers === 0) {

                    await prisma.messages.deleteMany({ where: {

                            ChatId: member.ChatId, }, });

                    await prisma.chats.delete({

                        where: {

                            ChatId: member.ChatId,

                        },  });  }  })

            .catch((err) => {

                throw new BadRequestException({ error: true, message: err });

            });

        return { message: 'You leave from this chat' };

    }

Листинг 3.11 – Метод выхода из чата

### 3.4.7 Реализация метода получения списка участников чата

Метод «GetChatMembers» возвращает список участников чата. Он выполняет запрос к базе данных с помощью метода Prisma «prisma.chats.findUnique», чтобы найти чат по его идентификатору (chatId). В запросе указывается выборка (select) только тех полей, которые необходимы – идентификатор пользователя (UserId) и идентификатор чата (ChatId) из связанной таблицы «ChatMembers». После выполнения запроса метод возвращает объект с данными о членах чата. Метод для получения списка участников чата в листинге 3.12.

    private async GetChatMembers(chatId: number) {

        const chatMembers = await this.prisma.chats.findUnique({

            where: {

                ChatId: chatId,

            },

            select: {

                ChatMembers: {

                    select: {

                        UserId: true,

                        ChatId: true,

                    }, }, }, });

return chatMembers;

    }

Листинг 3.12 – Метод для получения списка участников чата

### 3.4.8 Реализация метода ввода шифр ключа

При наличии ошибки отображается сообщение, иначе показывается стандартная метка. Введенный ключ сохраняется в объект setCryptoKey с добавлением идентификаторов чата и пользователя. После отправки формы вызывается метод для закрытия модального окна. Метод ввода шифр ключа представлен в листинге 3.13.

<div>

{errors.key?.message ? (

<label className="labelError">

{errors.key?.message}

</label>

) : (

<label className="label">

Chat private key

</label>)}

<input

type="number"

className="input"

min={1212112121212121}

max={9006554331798799}

{...register('key')}

placeholder="Chat private key" />

</div>

const onSubmit = async (data: any) => {

setCryptoKey(

{

type: 'add', key: 'KeyForChat–' + currentChatId + '–user–' + currentUserId, value: data.key,

});

props.handleSwitchModal(null);

};

Листинг 3.13 – Метод ввода шифр ключа

### 3.4.9 Реализация метода просмотра чатов и сообщений

Метод GetChatInfo получает информацию о чате включая сами сообщения. Он выполняет запрос в базу данных с помощью Prisma для получения данных о чате, проверяя, что пользователь является членом этого чата. При появлении ошибки она вернеться клиенту и отобразится на специальной странице с ошибками, это позволит избежать неправильной работы приложения. В ответ возвращаются данные о чате, его участниках, а также сообщения чата, включая идентификаторы, роли участников, имена пользователей и информацию о сообщениях, таких как содержание, тип, статус прочтения и дата создания. Метод получения информации чата и сообщения представлен в листинге 3.14.

    async GetChatInfo(dto: GetChatInfoDto, userId: string) {

        const chatInfo = await this.prisma.chats.findUnique({

            where: {

                ChatId: dto.chatId,

                ChatMembers: {

                    some: {

                        UserId: userId,

                    }, }, },

            select: {

                ChatId: true,

                ChatName: true,

                IsGroup: true,

                KeyHash: true,

                AvatarPath: true,

                CreatedAt: true,

                UpdatedAt: true,

                ChatMembers: {

                    select: {

                        ChatMemberId: true,

                        Role: true,

                        ChatId: true,

                        JoinedAt: true,

                        Member: {

                            select: {

                                UserId: true,

                                Name: true,

                                AvatarPath: true,

                                UserName: true,

                            }, }, }, },

                ChatMessages: {

                    select: {

                        MessageId: true,

                        Content: true,

                        MessageType: true,

                        IsUpdate: true,

                        IsRead: true,

                        CreatedAt: true,

                        SenderId: true,

                    }, }, }, });

Листинг 3.14 – Метод получения информации чата и сообщений

### 3.4.10 Реализация метода отправки сообщения

Метод «AddMessage» добавляет новое сообщение в чат, начиная с валидации данных через «ValidateAddMessage», чтобы убедиться, что пользователь имеет право отправить сообщение. После проверки создается запись о новом сообщении в базе данных с помощью метода Prisma «prisma.messages.create». В данных сообщения указываются чат (dto.chatId), отправитель (userId), содержимое (dto.content) и тип сообщения (dto.messageType). В ответ метод выбирает важные поля сообщения: идентификатор (MessageId), содержимое (Content), тип сообщения (MessageType), флаги обновления (IsUpdate) и прочтения (IsRead), дату создания (CreatedAt) и идентификатор отправителя (SenderId).

В конце метод возвращает объект нового сообщения, включая все поля, сохраненные в базе данных, что позволяет использовать их для дальнейших операций и отображения в интерфейсе. Метод отправки сообщения в листинге 3.15.

async AddMessage(dto: NewMessageDto, userId: string) {

        await this.ValidateAddMessage(dto, userId);

        const newMessage: ChatMessage = await this.prisma.messages.create({

            data: {

                ChatId: dto.chatId,

                SenderId: userId,

                Content: dto.content,

                MessageType: dto.messageType,

            },

            select: {

                MessageId: true,

                Content: true,

                MessageType: true,

                IsUpdate: true,

                IsRead: true,

                CreatedAt: true,

                SenderId: true,

            },

        });

        const chatMembers = await this.GetChatMembers(dto.chatId);

chatMembers.ChatMembers.forEach((member) => {

            this.wsGateway.SendMessageToUser(

                newMessage,

                member.UserId,

                member.ChatId,

            );

        });

        return newMessage;

}

Листинг 3.15 – Метод для отправки сообщения

### 3.4.11 Реализация метода удаления сообщений

Метод DeleteMessage удаляет сообщение из чата. Сначала выполняется валидация, проверяющая, имеет ли пользователь право удалить это сообщение. Затем сообщение удаляется из базы данных, и для каждого участника чата отправляется уведомление о удалении сообщения через WebSocket с помощью метода wsGateway.DeleteMessageToUser. В конце возвращается сообщение об успешной операции. Метод удаления сообщений представлен в листинге 3.16.

   async DeleteMessage(dto: DeleteMessageDto, userId: string) {

        await this.ValidateUserMessage(dto, userId);

        const chat = await this.prisma.messages.delete({

            where: {

                MessageId: dto.MessageId,             },

            select: {

                ChatId: true,             }, });

        const chatMembers = await this.GetChatMembers(chat.ChatId);

        chatMembers.ChatMembers.forEach((member) => {

            this.wsGateway.DeleteMessageToUser(

                dto.MessageId,

                member.UserId,

                member.ChatId,             ); });

        return { message: 'successful' }; }

Листинг 3.16 – Метод удаления сообщений

### 3.4.12 Реализация метода редактирования профиля

Методы UpdateName и UpdateUserName обновляют имя и имя пользователя в базе данных. Оба метода используют Prisma для обновления соответствующего поля (Name или UserName) пользователя по его идентификатору. После обновления возвращается объект с новым значением имени или имени пользователя. Метод редактирования профиля представлен в листинге 3.17.

   async UpdateName(dto: UpdateNameDto, userId: string) {

        const user = await this.prisma.users.update({

            where: { UserId: userId },

            data: {

                Name: dto.name, },

});

        return { newName: user.Name };

}

    async UpdateUserName(dto: UpdateUserNameDto, userId: string) {

        const user = await this.prisma.users.update({

            where: { UserId: userId },

            data: {

                UserName: dto.username,

            },

});

        return { newUsername: user.UserName };

}

Листинг 3.17 – Методы редактирования профиля

### 3.4.13 Реализация метода перевода сообщения

Этот код использует хук useApi для выполнения запроса на API перевода. Хук возвращает данные перевода (translateObject), функцию для обновления этих данных (setResData), состояние загрузки (translating), которое показывает, выполняется ли запрос, и функцию для выполнения запроса (getTranslateObject). Запрос выполняется с помощью axios.get к API MyMemory [10], передавая параметры из объекта data для перевода текста. Метод перевода сообщения представлен в листинге 3.18.

    const {

        resData: translateObject,

        setResData,

        loading: translating,

        execute: getTranslateObject,

    } = useApi<ResponseTranslationDto, RequestTranstalionDto>(async (data) => {

        return await axios.get('https://api.mymemory.translated.net/get', {

            params: data,

}); });

Листинг 3.18 – Метод перевода сообщения

### 3.4.14 Реализация метода поиска чата

Этот код использует useMemo для сортировки и фильтрации списка чатов. Если задано значение для переменной search, список чатов фильтруется по названию чата, проверяя, содержится ли искомое слово в названии. Если search не задано, список чатов сортируется по полю IsFixed в убывающем порядке. В useEffect выполняется запрос для получения списка чатов при монтировании компонента, вызывая функцию executeChatsList. Метод поиска чата представлен в листинге 3.19.

    const sortedChatsList = useMemo(() => {

        if (search) {

            return chatsListData?.filter((el) =>

       el.ChatName.toLowerCase().includes(search.toLowerCase()),

);}

        if (!chatsListData) return [];

        return [...chatsListData].sort(

            (a, b) => Number(b.IsFixed) – Number(a.IsFixed),

        );

    }, [chatsListData, search]);

    useEffect(() => {

        executeChatsList();

    }, []);

Листинг 3.19 – Метод поиска чата

### 3.4.15 Реализация метода установки чата в закрепленные

Метод FixChat изменяет статус закрепления чата для пользователя. Сначала выполняется валидация с помощью метода ValidateFixChat. Затем обновляется поле IsFixed для указанного члена чата в базе данных, меняя его значение на противоположное (если чат уже закреплен, он будет откреплен, и наоборот). В конце метод возвращает сообщение, идентификатор члена чата и новый статус закрепления. Метод установки чата в закрепленные представлен в листинге 3.20.

    async FixChat(dto: FixChatDto, userId: string) {

        const isFixed = await this.ValidateFixChat(dto, userId);

        await this.prisma.chatMembers.update({

            where: {

                ChatMemberId: dto.chatMemberId, },

            data: {

                IsFixed: !isFixed,             }, });

        return {

            message: '!Fix chat',

            chatMemberId: dto.chatMemberId,

            status: !isFixed, }; }

Листинг 3.20 – Метод установки чата в закрепленные

### 3.4.16 Реализация метода обноваления имени чата

Метод «UpdateChatName» обновляет название чата. Он выполняет запрос с помощью метода Prisma «prisma.chats.update», чтобы найти чат по его идентификатору (dto.chatId) и обновить его название (ChatName), используя новое значение, переданное в dto.chatName. После успешного обновления возвращается объект с сообщением «Chat name updated», подтверждающим, что операция была выполнена. Код в листинге 3.21.

async UpdateChatName(dto: UpdateChatNameDto) {

        await this.prisma.chats.update({

            where: {

                ChatId: dto.chatId,

            },

            data: {

                ChatName: dto.chatName,

            },

        });

        return {

message: 'Chat name updated'

};

    }

Листинг 3.21 – Метод для обноваления имени чата

### 3.4.17 Реализация метода добавления пользователя в чат

Метод «AddMember» добавляет нового участника в чат. Сначала он проверяет, начинается ли имя пользователя (dto.username) с символа «@», и если это так, удаляет его. Затем выполняется поиск пользователя в базе данных с помощью метода Prisma «prisma.users.findUnique», чтобы найти пользователя по имени. Если пользователь не найден, выбрасывается исключение с сообщением, что такой пользователь не существует. После этого в объект dto добавляется идентификатор найденного пользователя (dto.userId). Далее метод вызывает «ValidateAddMember», чтобы убедиться, что текущий пользователь имеет права на добавление нового участника. После успешной валидации добавляется новый участник в чат через метод «prisma.chatMembers.create», где указывается идентификатор чата, пользователя и роль, переданные в dto. Метод также вызывает «wsGateway.AddUserToChat» для обновления WebSocket–соединения с новым участником чата. В результате метод возвращает информацию о новом члене чата. Код метода добавления пользователя в чат представлен в листинге 3.22.

async AddMember(dto: AddMemberDto, userId: string) {

        if (dto.username[0].toLocaleLowerCase() == '@')

            dto.username = dto.username.slice(1);

        const addedUser = await this.prisma.users.findUnique({

            where: {

                UserName: dto.username,

            select: {

                UserId: true, },

        });

        if (!addedUser)

            throw new BadRequestException({

                error: true,

                show: true,

                message: 'User @' + dto.username + ' not found',

            });

        dto.userId = addedUser.UserId;

        await this.ValidateAddMember(dto, userId);

        const NewMember = await this.prisma.chatMembers.create({

            data: {

                ChatId: dto.chatId,

                UserId: dto.userId,

                Role: dto.role, },

        });

        this.wsGateway.AddUserToChat(dto.userId);

        return NewMember;

    }

Листинг 3.22 – Метод для добавления пользователя в чат

### 3.4.18 Реализация метода отправки сообщения пользователю

Метод «SendMessageToUser» отправляет сообщение конкретному пользователю через WebSocket. Сначала он ищет идентификатор сокета (socketId) пользователя, используя его идентификатор (userId) из хранилища подключенных клиентов (this.connectedClients). Если для данного пользователя найден socketId, то с помощью метода WebSocket–сервера «this.server.to(socketId).emit» отправляется событие 'NewMessage' с данными сообщения (message) и идентификатором чата (chatId). Если socketId не найден, сообщение не отправляется. Код метода для отправки сообщения пользователю в листинге 3.23.

async SendMessageToUser(

        message: ChatMessage,

        userId: string,

        chatId: number,) {

    const socketId = this.connectedClients.get(userId);

    if (socketId) {

    this.server.to(socketId).emit('NewMessage', { message,chatId }); } }

Листинг 3.23 – Метод для отправки сообщения пользователю

### 3.4.19 Реализация метода назначения роли и добавления участника чата

Метод AddMember добавляет нового участника в чат. Сначала проверяется, что имя пользователя не начинается с символа '@', и при необходимости удаляется этот символ. Затем выполняется поиск пользователя по имени в базе данных. Если пользователь не найден, выбрасывается исключение с ошибкой. После этого вызывается валидация для добавления участника в чат, и если все проверки пройдены, создается новая запись в таблице членов чатов с указанием чата, пользователя и его роли. Также отправляется уведомление через WebSocket о добавлении пользователя в чат. В конце метод возвращает информацию о новом участнике. Метод назначения роли в листинге 3.24.

    async AddMember(dto: AddMemberDto, userId: string) {

        if (dto.username[0].toLocaleLowerCase() == '@')

            dto.username = dto.username.slice(1);

        const addedUser = await this.prisma.users.findUnique({

            where: {

                UserName: dto.username, },

            select: {

                UserId: true, }, });

        if (!addedUser)

            throw new BadRequestException({

       message: 'User @' + dto.username + ' not found', });

        dto.userId = addedUser.UserId;

        await this.ValidateAddMember(dto, userId);

        const NewMember = await this.prisma.chatMembers.create({

            data: {

                ChatId: dto.chatId,

                UserId: dto.userId,

                Role: dto.role,             }, });

        this.wsGateway.AddUserToChat(dto.userId);

        return NewMember;

Листинг 3.24 – Метод назначения роли

### 3.4.20 Реализация метода Encrypt

Метод «Encrypt» использует ключ для шифрования сообщения, где этот ключ извлекается из локального хранилища. Ключ хранится как число и служит основным элементом для выполнения операции XOR, его минимальное значение должно быть не менее чем 1000200030004000, а максимальное не более чем 9006554331798799. Данные извлекаются из локального хранилища, где ключ сохраняется, что позволяет использовать его для шифрования сообщений в текущем сеансе работы. Важно, что этот ключ используется на клиентской стороне.

Процесс шифрования начинается с преобразования сообщения в двоичный код, а затем для каждого бита этого кода выполняется побитовая операция XOR с соответствующим битом ключа. В случае, если длина сообщения превышает длину ключа, последний повторяется циклично. Это позволяет поддерживать шифрование, несмотря на различие в длине данных и ключа.

После выполнения операции XOR результат снова преобразуется в символы, и итоговое зашифрованное сообщение возвращается в виде строки. Таким образом, алгоритм использует побитовые операции для создания зашифрованной версии исходного текста.

Этот подход к шифрованию данных может быть полезен на клиентской стороне. Простота алгоритма делает его быстрым и эффективным. Метод использования XOR с ключом позволяет обеспечить достаточный уровень безопастности.

Алгоритм легок в реализации, не требует использования внешних библиотек. Также, его устойчивость к атакам может быть достаточно хорошей, если ключ используется разумно и соблюдаются меры безопасности. Код представлен в приложении В.

### 3.4.21 Реализация метода Decrypt

Метод «Decrypt» реализует процесс расшифровки сообщения, которое было зашифровано с помощью ранее описанного алгоритма. Расшифровка происходит по аналогии с шифрованием, поскольку операция XOR является симметричной. То есть для расшифровки достаточно применить тот же ключ и тот же алгоритм, что и при шифровании, чтобы восстановить исходное сообщение.

Процесс расшифровки начинается с того, что зашифрованное сообщение (в виде строки) преобразуется в двоичный код, как это было сделано при шифровании. Затем для каждого бита в этом двоичном коде выполняется операция XOR с соответствующим битом ключа. Ключ, представленный в двоичном виде, используется циклично, повторяясь, если длина сообщения превышает длину ключа.

После выполнения операции XOR, результат преобразуется обратно в символы, восстанавливая исходное сообщение.

Важно отметить, что для успешной расшифровки необходимо, чтобы ключ был точно таким же, как и при шифровании. В противном случае восстановить исходное сообщение невозможно. Таким образом, ключ играет ключевую роль в обоих процессах – и шифровании, и расшифровке. Код представлен в листинге В.

Методы, представленные выше, реализуют базовую функциональность для работы с шифрованием и расшифровкой сообщений, а также вычислением оптимального положения контекстного меню в пользовательском интерфейсе.

Остальные методы и хуки, используемые на клиентской стороне представленны в приложении В.

## 3.5 Архитектурный подход

В архитектуре REST API используется для работы с различными сущностями, такими как аутентификация, блокировки, чаты и другие, используются различные маршруты и методы HTTP. Каждый маршрут связан с определенным контроллером и методом обрабатывающий данный запрос.

В таблице 3.1 описаны все маршруты HTTP–сервера.

Таблица 3.1 – Описание маршрутов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL | Метод | | Контроллер | Описание |
| /api/auth/register | GET | | Auth | Регистрация нового пользователя. |
| /api/auth/login | POST | | Auth | Вход пользователя с логином и паролем |
| /api/auth/logout | POST | | Auth | Выход пользователя с удалением используемых токенов. |
| /api/auth/testauth | POST | | Auth | Проверка авторизации. |
| /api/profile/getMyProfile | GET | | Profile | Позволяет получить данные профиля. |
| /api/profile/updateName | POST | | Profile | Позволяет изменить имя. |
| /api/profile/updateUsername | PUT | | Profile | Позволяет изменить имя пользователя. |
| /api/profile/updateLanguage | PUT | | Profile | Позволяет изменить язык. |
| /api/profile/updatePassword | | PUT | Profile | Позволяет изменить пароль. |
| /api/profile/uploadAvatar | | PUT | Profile | Позволяет изменить аватар. |
| /api/chat/createChat | | POST | Chat | Создание нового чата. |
| /api/chat/AddMember | | POST | Chat | Добавления нового участника чата. |
| /api/chat/deleteMember | | PUT | Chat | Удаление участника из чата. |
| /api/chat/leaveFromChat | | POST | Chat | Выход из чата. |
| /api/chat/getMyChatsList | | GET | Chat | Получение списка чатов пользователя. |
| /api/chat/fixChat | | PUT | Chat | Перовед чата в закрепленный или наоборот. |
| /api/chat/GetChatInfo | | POST | Chat | Получение информации чата включая всю историю сообщений. |
| /api/chat/sendMessage | | POST | Chat | Создание и отправка сообщения в чат. |
| /api/chat/uploadAvatar | | POST | Chat | Загрузка аватара чата. |
| /api/chat/updateChatName | | PUT | Chat | Изменение имени чата. |
| /api/chat/deleteMessage | | DELETE | Chat | Удаление сообщения, отправленного данным пользователем. |
| /api/chat/updateMessage | | PUT | Chat | Редактирование сообщения, отправленного данным пользователем. |

Тем самым мы рассмотрели маршруты и соответствующие контроллеры, обеспечивающие ключевую функциональность системы. Каждый из представленных методов играет важную роль в построении логики взаимодействия между клиентской и серверной сторонами.

## 3.6 Используемые форматы передачи данных

Форматы передачи данных в API могут быть различными в зависимости от типа операции, а также от специфики данных, которые необходимо передать. В общем случае, для большинства запросов и ответов используется формат JSON. Это позволяет эффективно передавать структурированные данные, такие как объекты, массивы, строки, числа и другие базовые типы.

В случае с DTO (Data Transfer Object), как правило, используется формат JSON для структурирования данных, которые необходимы для различных операций. Однако для загрузки фотографии используется формат «multipart/form–data». Это позволяет передавать файлы в составе HTTP–запроса наряду с другими данными, такими как текстовые поля.

Для ответа на запросы в основном используется JSON, который удобно анализировать и обрабатывать на сервере и клиенте. Тем не менее, для некоторых операций, таких как загрузка изображения, может быть возвращен сам файл в виде бинарного потока или конкретного медиа–формата. Например, при запросе фотографии – изображение в формате JPEG или PNG.

Для других запросов, таких как запросы на получение информации о пользователях, чатах или сообщениях, ответ будет в формате JSON, представляющим структурированные данные.

## 3.7 Сторонние сервисы

Для перевода сообщений используется сторонний API MyMemory [10] – это один из крупнейших онлайн–ресурсов для перевода текстов и фраз. Платформа предоставляет доступ к миллионам фраз, переведенных на различные языки, а также предлагает инструменты для машинного перевода. Платформа предлагает API, которое позволяет интегрировать MyMemory в собственные приложения и инструменты для перевода. Таким образом, MyMemory является хорошим выбором для игтеграции в Cryptogram.

## 3.8 Выводы

В данном разделе подробно проанализирован выбор технологий для разработки приложения Cryptogram:

1. серверная часть: использование NestJS обосновано его модульной архитектурой, встроенной поддержкой WebSocket и полной интеграцией с TypeScript, что упрощает разработку и тестирование;
2. клиентская часть: React был выбран за его гибкость при разработке пользовательского интерфейса и эффективное управление состоянием благодаря библиотеке Jotai;
3. база данных: PostgreSQL выбрана за надёжность, высокую производительность при работе со сложными запросами и поддержку транзакций. Для упрощения взаимодействия с базой данных используется Prisma ORM, предоставляющая объектно–ориентированный подход и ускоряющая процесс разработки;
4. архитектура: приложение построено на монолитном подходе. Взаимодействие между клиентом и сервером осуществляется через REST API и WebSocket, что позволяет поддерживать работу в реальном времени;
5. безопасность: реализована аутентификация с использованием токенов JWT, управление пользовательскими ролями и клиентское шифрование данных. Пользователи сами выбирают ключи шифрования, которые хранятся исключительно на их устройствах;
6. проектирование базы данных: реляционная структура базы данных оптимизирована для обеспечения целостности и быстрого выполнения запросов. Таблицы охватывают управление пользователями, чатами, мультимедиа и их связями;
7. функциональность приложения: еализованы механизмы работы с пользователями, чатами, мультимедиа и другими элементами;
8. рассмотрены примеры работы с API, взаимодействие с базой данных и методы обработки событий в реальном времени;
9. шифрование сообщений: представлены подходы к шифрованию и расшифрованию сообщений. Скрипты обеспечивают безопасность хранения данных, делая систему надёжной и устойчивой к взлому;
10. работа с данными: использование форматов JSON и «multipart/form–data» позволяет оптимизировать взаимодействие компонентов системы и упрощает передачу данных.

В результате создано приложение, отвечающее как техническим, так и бизнестребованиям, с продуманной архитектурой, надёжной безопасностью и модульным подходом, который упрощает добавление новых функций и масштабирование.

# 4 Тестирование веб–приложения

## 4.1 Функциональное тестирование

**Функциональное тестирование** проводилось для проверки реализации заявленного функционала, такого как авторизация и регистрация, отправка сообщений их редактирование и удаление атакже и перевод сообщений, создание чатов и добавление в них участников. Для каждого сценария тестирования были заданы ожидаемые и фактические результаты. Пример тестов представлен в таблице 4.1.

Таблица 4.1 – Примеры функциональных тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат | Тест прошёл успешно |
| Регистрация нового пользователя | Пользователь успешно зарегистрирован выданы токены | Пользователь зарегистрирован и получил токены | Да |
| Авторизация существующего пользователя | Пользователь успешно авторизован и получил токены | Пользователь успешно авторизован и получил токены | Да |
| Авторизация несуществующего пользователя | Вывод сообщения о несуществующей регистрации | Вывод сообщения о несуществующей регистрации | Да |
| Отправка сообщения | Созданно новое сообщение в бд | Созданно новое сообщение в бд | Да |
| Получение сообщения | Получени сообщение в реальном времени то вебсокету | Получени сообщение в реальном времени то вебсокету | Да |
| Перевод сообщения с английского на русский язык | Сообщение успешно переведено | Сообщение успешно переведено | Да |
| Перевод сообщения без интернета | Переход на страницу с пояснением ошибки | Переход на страницу с пояснением ошибки | Да |
| Редактирование сообщения | Сообщение отредактированно и отправленно участникам чат | Сообщение отредактированно и отправленно участникам чата | Да |
| Удаление сообщения | Сообщение пропало у всех участников чата | Сообщение пропало у всех участников чата | Да |
| Создание чата | Содался новый чат | Содался новый чат | Да |
| Добавление пользователя в чат | В чат добавлен новый пользователь | В чат добавлен новый пользователь | Да |
| Удаление пользователя из чата | Пользователь пропал из участников чата и у него пропали права доступа к чату | Пользователь пропал из участников чата и у него пропали права доступа к чату | Да |

Функциональное тестирование подтвердило корректность реализации заявленного функционала системы. Для каждого сценария тестирования были определены ожидаемые результаты, которые сравнивались с фактическими. Все тесты, представленные в таблице 4.1, завершились успешно, что указывает на соответствие системы требованиям.

## 4.2 Автоматизированное тестирование

Автоматизированное тестирование было выполнено с использованием тестового фреймворка, такого как Swagger для серверной части и Jest для клиентской части. Основное внимание уделялось тестированию API и компонентов пользовательского интерфейса. Описание тестов в таблице 4.2.

Таблица 4.2 – Примеры автоматизированных тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат | Тест прошёл успешно |
| Тестирование авторизации API с валидными данными | Код ответа 200, возвращён токен | Код ответа 200, токен возвращён | Да |
| Тестирование авторизации API с невалидными данными | Код ответа 401, сообщение об ошибке | Код ответа 401, ошибка корректна | Да |
| Проверка обновления данных профиля через API | Код ответа 200, данные обновлены | Код ответа 200, данные успешно обновлены | Да |
| Проверка списка чатов | На фронтенде отображается список чатов | Список корректно отображается | Да |
| Тест рендера кнопки отправки сообщения | Кнопка отображается в активном состоянии | Кнопка корректно отображена | Да |
| Проверка функциональности кнопки «Покинуть чат» | Участник покидачет чат через API | Участник успешно покинул чат | Да |
| Тест на валидацию формы регистрации | Ошибка отображается при некорректном вводе email | Ошибка корректно отображается | Да |

Код тестов для клиентской части представлены в приложении Г.

## 4.3 Выводы

1. В разделе тестирования веб-приложения проведены различные виды тестов, направленные на проверку корректности работы системы. Это позволило выявить и устранить ключевые ошибки на этапе разработки, а также оценить, насколько приложение соответствует заявленным требованиям;
2. функциональное тестирование подтвердило работоспособность таких основных функций, как регистрация и авторизация пользователей, отправка текстовых сообщений, управление списком контактов и доступ к файлам. Эти проверки показали, что функционал системы полностью соответствует ожидаемым результатам и требованиям технического задания;
3. автоматизированное тестирование проводилось с использованием современных инструментов Swagger и Jest. Оно продемонстрировало высокую эффективность в выявлении недочётов и ошибок на ранних этапах разработки. Это позволило существенно повысить общую надёжность системы;
4. нагрузочное тестирование, выполненное с помощью инструмента JMeter, позволило определить пределы производительности приложения при интенсивной нагрузке. Тесты имитировали одновременную работу большого числа пользователей, оценивая, как система реагирует на пиковые нагрузки. Полученные результаты подтвердили, что приложение способно стабильно работать даже при высокой интенсивности запросов;
5. по итогам всех тестов установлено, что система обладает высокой устойчивостью, надёжностью и производительностью. Она демонстрирует стабильную работу в различных условиях эксплуатации, что подтверждает её готовность к использованию в реальных условиях.

# 5 Руководство пользователя

## 5.1 Руководство пользователя

При первом посещении данного приложения вам будет доступна астраница авторизации, если у вас еще нет аккаунта вам необходимо зарегистрироваться для этого вам необходимо нажать на кнопку регистрации, и перейти к следующему пункту. Если у вас уже есть аккаунт, то вам необходимо ввести свою поту и пароль для проверки вашей личности и нажать на кнопку отправить. Рисунок страницы авторизации представлен на рисунке 5.1.

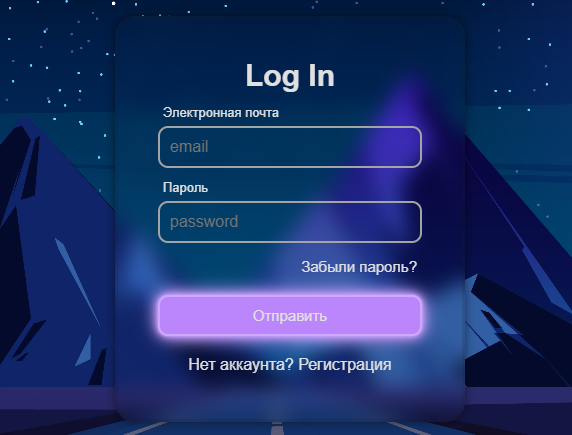


Рисунок 5.1 – Страница авторизации

Для регистрации вамнеобходимо заполнить все поля с вашими данными и нажать на кнопку отправить после чего система зарегистрирует вас и перенаправит на страницу чатов. Рисунок страницы регистрации представлен на рисунке 5.2.

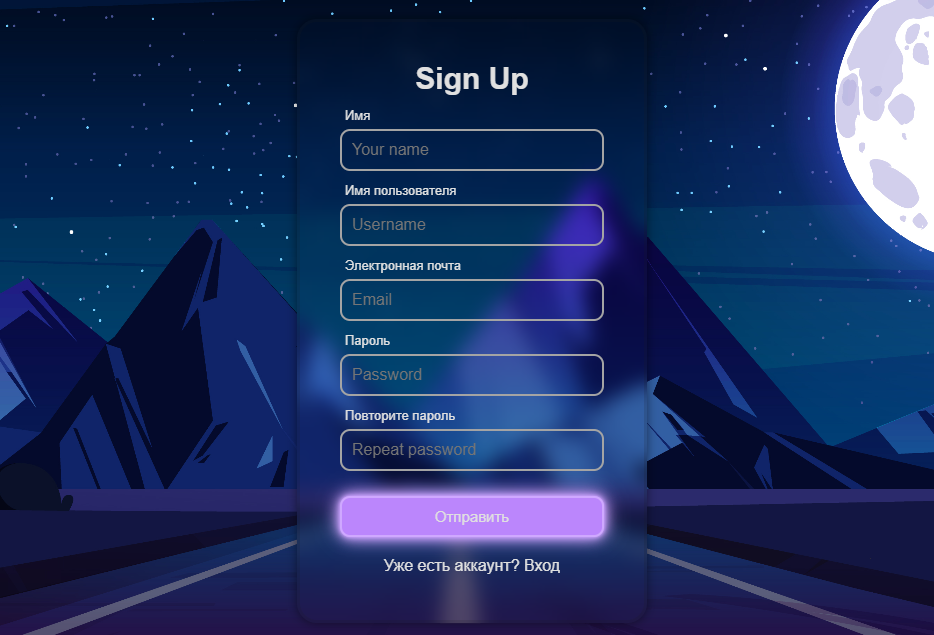


Рисунок 5.2 – Страница регистрации

После успешной регистрации или авторизации вас перенаправиьт на страницу ваших чатов в левой части экрана есть вертикальное меню для перехода на страницу профиля, списка чатов или страницу создания нового чата. На данной странице вы можете выбрать чат из списка при наличии для просмотра истории сообщений и отправки новых сообщений. Рисунок страницы чатов представлен на рисунке 5.3.

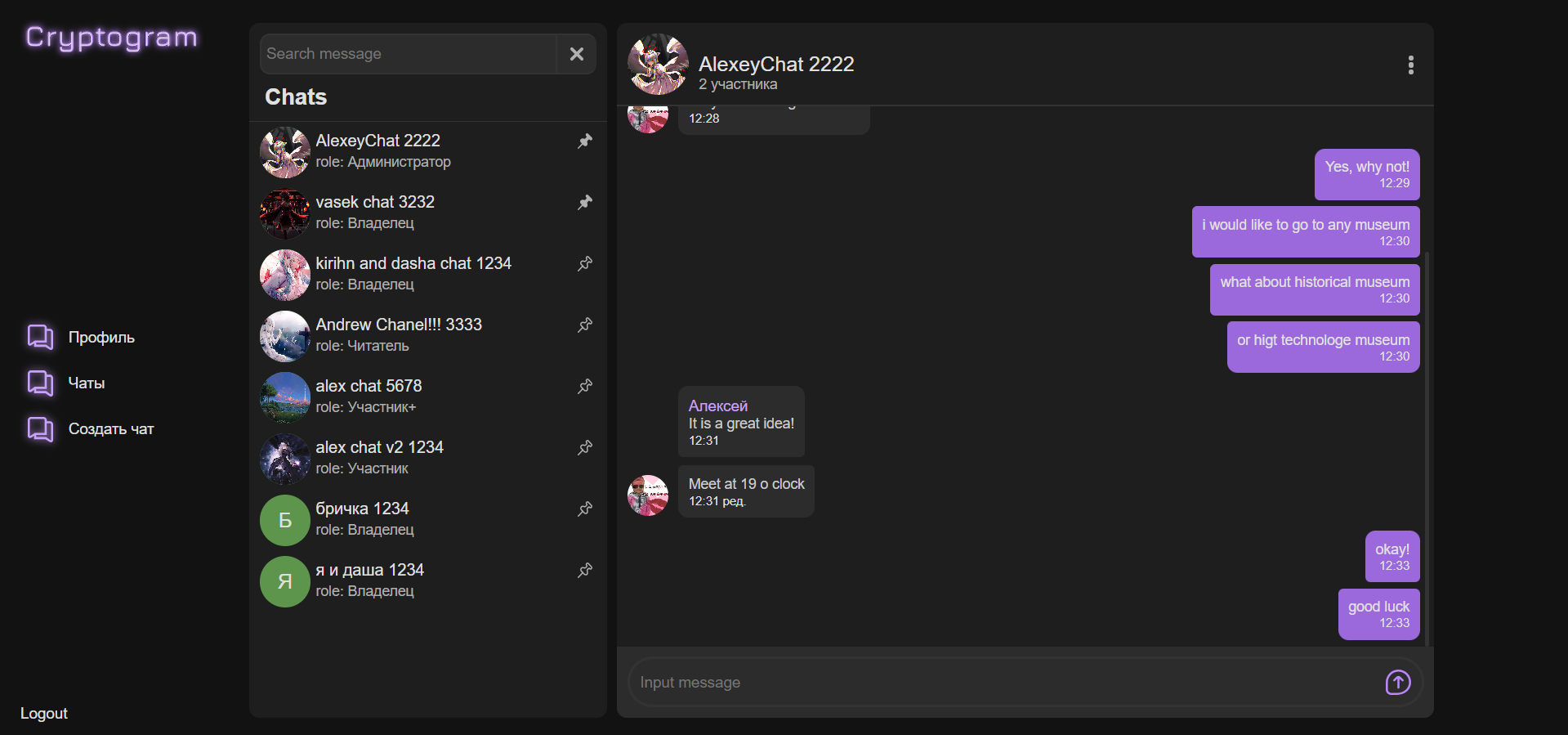


Рисунок 5.3 – Страница чатов.

При желании чат можно закрепить, тогда он будет выше в списке чатов, это позволяет не искать частоиспользуемые чаты. Для этого вам необходим нажать на кнопку закрепа. Рисунок кнопки закрепа представлен на рисунке 5.1.



Рисунок 5.4 – Кнопка закрепа

Для создания и отправки сообщений воспользуйтесь нижней панелью ввода, после написания сообщения нажмите на кнопку отправки сообщения, так же стоит учитывать, что максимальный размер отправляемого сообщения равен 499 символам. После того как вы нажмете на кнопку отправки это сообщение немедленно будет доставленно и другим участникам чата, которые смогут просмотреть его. Рисунок панели написания сообщения представлен на рисунке 5.4.

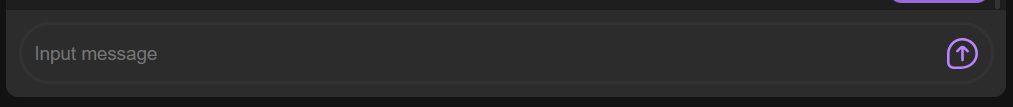


Рисунок 5.5 – Панель написания сообщения

Для изменения, удаления или копирования вашего сообщения воспользуйтесь контекстным меню вызвав его нажатием ПКМ по повему сообщению, с которым вы хотите работать. Рисунок контекстного меню отправленного сообщения представлен на рисунке 5.1.

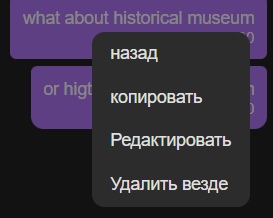


Рисунок 5.5 – Контекстное меню отправленного сообщения

При необходимости перевода собщения полученного от другого пользователя вам необходимо также вызвать контекстное меню этого сообщения нажав по нему ПКМ и выбрать соответствующий вашим требованиям пункт, после чего будет создан перевод сообщения или оно будет скопированно. Рисунок контекстного меню полученного сообщенияпредставлен на рисунке 5.6.

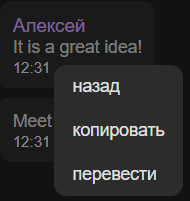


Рисунок 5.6 – Контекстное меню полученного сообщения

Для управления чатом вам необходимо вызвать окно управления чатом нажав на три вертикальных точки в правом верхнем углу чата. После чего у вас появится модальное окно где вы можете увидеть аватар чата, его имя, участников и при налиции достаточной роли кнопки которые предоставят вам возможность управлять данным чатом, например приглашать других пользоватеелй удалять их, менять имя чата его аватарку или вы сможете покинуть чат при желании. Рисунок модального окна параметров чата представлен на рисунке 5.7.

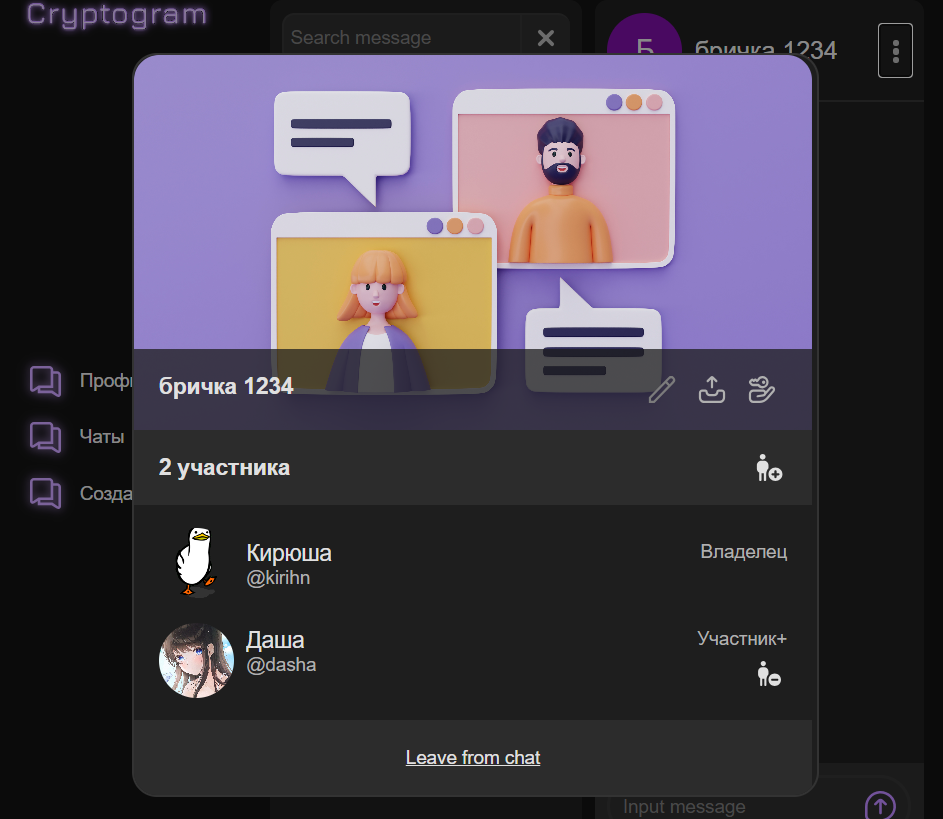


Рисунок 5.7 – Модальное окно параметров чата

Для просмотра своего профиля вам необходимо нажать на левом боковом меню кнопку профиля после чего вы перейдете на другую страницу с вашим профилем. Тут вы видете свой аватар сверху в самом центе и настройки которые разделяются на 2 списка, в первом вы видите общедоступные настройки которые видны всем пользователям, при желании вы можете изменить свое имя, имя пользователя или ваш аватар нажав на кнопку рядом с каждым пунктом или во втором списке вам дуступна функция смены пароля если вы решите что ваш стал недостаточно надежным по аналогии для его замены вам необходимо нажать на кнопку рядом с этим пунктом. После открытия любого модального окна вам необходимо ввести новые данные об новом имени, выбрать новый аватар, данные нового пароля, нажать на кнопку сохранить, так вы сможете отрадактировать ваши данные. Рисунок страницы профиля представлен на рисунке 5.8.



Рисунок 5.8 – Страница профиля

Для создания нового чата вам необходима в левом боковом меню выбрать создания чата после чего у вас откроется страница с формой для ввода данных нового чата. Тут необходимо ввести имя чата и используемый в нем ключ. Ключ должен быть не менее чем 1000200030004000 и не более чем 9006554331798799 этот депозон значений ключа гарантирует надежную и достаточную бинарную последовательность, которая в последствии будет использоваться при шифровании сообщений. Рисунок Страницы создания чата представлен на рисунке 5.9.

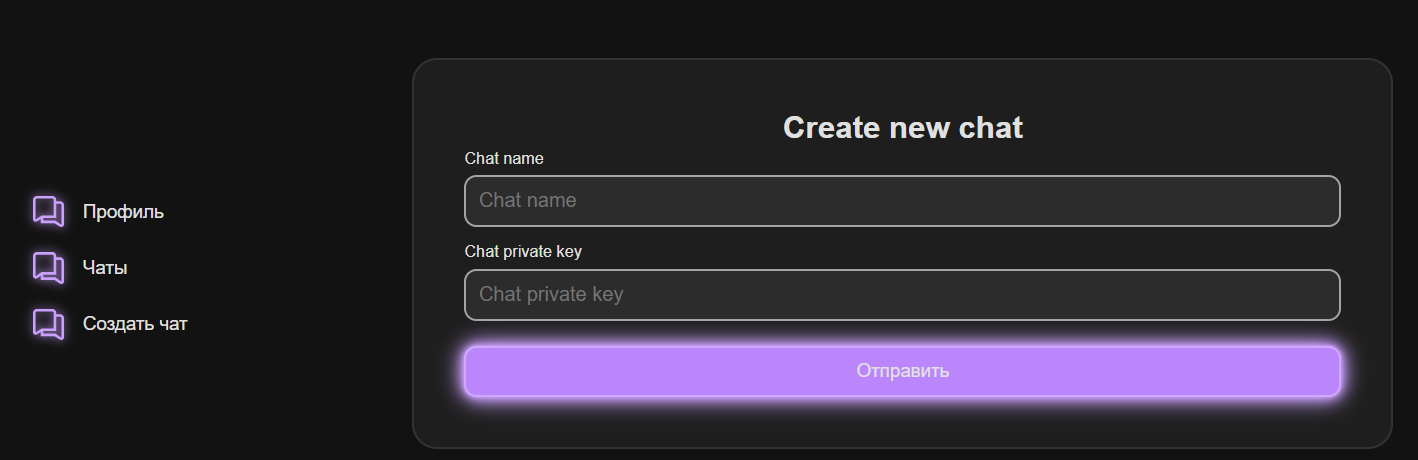


Рисунок 5.9 – Страница создания чата

## 5.2 Установка веб–приложения

Установка приложения происходит в несколько этапов:

1. Сначала нужно клонировать все веб–приложение из Github репозитория.
2. Дальше необходимо создать образы веб–приложения в Docker.
3. Дальше необходимо запустить контейнеры на Docker на основе собранных образов.

## 5.3 Выводы по разделу

В данном разделе рассмотрен функционал и процесс работы с веб–приложением Cryptogram, также предоставлена пошаговая инструкция по его установке и эксплуатации.

1. В первой части раздела рассмотрен пользовательский интерфейс и основные действия, которые доступны пользователям;
2. При открытии приложения пользователю предлагается форма авторизации, где можно войти в систему или зарегистрироваться;
3. После успешного входа пользователь попадает на главную страницу, где отображаются доступные чаты и переписки. Здесь предусмотрена возможность открыть конкретный чат для чтения и отправки сообщений или перейти на другие страницы, например страницу профиля или создания нового чата
4. Функционал чата включает отправку текстовых сообщений и их перевод.;
5. Особое внимание уделено удобству использования данного мессенджера, при помощи инструментов передова сообщений, их копирования, удаления и редактирования. Блогодоря этому пользователям всегда будет комфортно и удобно при общении;

Таким образом, данный раздел предоставляет пользователю полное руководство по работе с веб–приложением Cryptogram, описывая как взаимодействие с функционалом, так и технические аспекты его установки. Четкость описания делают взаимодействие с приложением интуитивно понятным для любого пользователя.

# Заключение

Архитектура приложения включает серверную и клиентскую части и базу данных, развёрнутые в контейнерах Docker, что гарантирует стабильность и безопасность работы системы.

Серверная часть реализована на базе NestJS, что позволяет эффективно обрабатывать асинхронные запросы и расширять приложение в будущем за счет модульности, а для работы в реальном времени используется WebSocket.

Клиентская часть построена на ReactJS и использование Typescript, что обеспечивает отзывчивый, динамичный и строготипизированный пользовательский интерфейс.

В качестве систем хранения данных используются PostgreSQL, что обеспечивает надёжность и высокую скорость получения данных.

Система поддерживает 2 роли приложения, такие как Гость и Пользователь что позволяет гибко управлять доступом к различным частям приложения, повышая безопасность и удобство работы пользователей.

Система поддерживает 5 ролей для чатов, такие как владелец, администратор, участник+, участник и читатель. что позволяет гибко управлять доступом к чату для разных участников чата.

Реализованы все необходимые для работы приложения методы и API–маршруты. Всего приложение Cryptogram насчитывает более 20 функций

База данных приложения содержит несколько таблиц, представляющих ключевые сущности системы, такие как пользователи, роли, чаты, сообщения, файлы и отчёты, с обеспечением целостности данных и эффективной работы с большим объёмом информации.

Архитектура приложения строится на контейнеризации с использованием Docker, что упрощает управление инфраструктурой и улучшает производительность при масштабировании. Docker Compose упрощает управление зависимостями между сервисами и ускоряет процесс развертывания и тестирования.

Проект насчитывает около 8000 строк кода.

Покрытие тестами составляет 65%, что свидетельствует о достаточной степени тестирования основных функций системы.

Система представляет собой сочетание современных технологий и архитектурных подходов, что обеспечивает её стабильность, безопасность и высокую производительность.

Разработанное приложен полностью соответствует заявленным требование и включает в себя всю необходимую функциональность.

# Список используемых источников

1. Node.js Documentation [Электронный ресурс] / Режим доступа: <https://nodejs.org/docs/latest/api/> – Дата доступа: 22.10.2024.
2. NestJS Documentation [Электронный ресурс] / Режим доступа: https://docs.nestjs.com/– Дата доступа: 22.10.2024.
3. Prisma ORM Documentation [Электронный ресурс] / Режим доступа: https://www.prisma.io/docs/orm/overview/databases/postgresql – Дата доступа: 22.10.2024.
4. PostgreSQL Documentation [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/enterprise/16/ – Дата доступа: 22.10.2024.
5. React Official Documentation. [Электронный ресурс] – Режим доступа: https://react.dev/learn – Дата обращения: 26.11.2024.
6. Webpack Documentation. [Электронный ресурс] – Режим доступа: https://webpack.js.org/concepts/. – Дата обращения: 22.10.2024.
7. Jotai Documentation. [Электронный ресурс] – Режим доступа: https://jotai.org/. – Дата обращения: 22.10.2024.
8. Scss Documentation. [Электронный ресурс] – Режим доступа: https://sass-scss.ru/guide/. – Дата обращения: 12.12.2024.
9. Docker Official Documentation. [Электронный ресурс] – Режим доступа: https://docs.docker.com. – Дата обращения: 12.12.2024.
10. Axios GitHub Repository. [Электронный ресурс] – Режим доступа: <https://github.com/axios/axios>. – Дата обращения: 26.11.2024.
11. Swagger OpenAPI Documentation. [Электронный ресурс] – Режим доступа: https://swagger.io/docs. – Дата обращения: 01.11.2024.
12. MyMemory: API technical specifications. [Электронный ресурс] – Режим доступа: https://mymemory.translated.net/doc/spec.php. – Дата обращения: 25.11.2024.
13. Json standart, technical specifications. [Электронный ресурс] – Режим доступа: https://www.json.org/json-en.html. – Дата обращения: 25.11.2024.
14. Jmetet documentstion. [Электронный ресурс] – Режим доступа: https://jmeter.apache.org. – Дата обращения: 26.11.2024.

# Приложение А

|  |
| --- |
|  |

Рисунок 2 – Диаграмма развертывания

# Приложение Б

Контекст «UsersDbContext»

generator client {

  provider = "prisma–client–js"

  binaryTargets = ["native", "debian–openssl–3.0.x"]

}

datasource db {

  provider = "postgresql"

  url      = env("DATABASE\_URL")

}

model Users {

  UserId       String        @id @default(cuid())

  Name         String

  AvatarPath   String        @default("/static/defaults/userAvatars/defaultUserAvatar.jpg")

  UserName     String        @unique

  Email        String        @unique

  Language     String        @default("ru")

  Role         Int           @default(1)

  PasswordHash String

  CreatedAt    DateTime      @default(now())

  UpdatedAt    DateTime      @default(now()) @updatedAt

  UserMessages Messages[]

  ChatMembers  ChatMembers[]

}

model Chats {

  ChatId       Int           @id @default(autoincrement())

  ChatName     String

  IsGroup      Boolean

  KeyHash      String

  AvatarPath   String        @default("/static/defaults/chatAvatars/defaultChatAvatar.png")

  CreatedAt    DateTime      @default(now())

  UpdatedAt    DateTime      @default(now()) @updatedAt

  ChatMessages Messages[]

  ChatMembers  ChatMembers[]

}

model Messages {

  MessageId   Int      @id @default(autoincrement())

  ChatId      Int

  SenderId    String

  Chat        Chats    @relation(fields: [ChatId], references: [ChatId])

  Sender      Users    @relation(fields: [SenderId], references: [UserId])

  Content     String

  MessageType String

  IsUpdate    Boolean  @default(false)

  IsRead      Boolean  @default(false)

  CreatedAt   DateTime @default(now())

  UpdatedAt   DateTime @default(now()) @updatedAt

}

model ChatMembers {

  ChatMemberId Int      @id @default(autoincrement())

  UserId       String

  Role         Int

  ChatId       Int

  IsFixed      Boolean  @default(false)

  JoinedAt     DateTime @default(now())

  Chat         Chats    @relation(fields: [ChatId], references: [ChatId])

  Member       Users    @relation(fields: [UserId], references: [UserId])

}

model StickerGroup {

  StickerGroupId Int        @id @default(autoincrement())

  GroupName      String

  Stickers       Stickers[]

}

model Stickers {

  StickerId      Int          @id @default(autoincrement())

  StickerGroupId Int

  StickerGroup   StickerGroup @relation(fields: [StickerGroupId], references: [StickerGroupId])

}

Код для создания базы данных

–– CreateTable

CREATE TABLE "Users" (

    "UserId" TEXT NOT NULL,

    "Name" TEXT NOT NULL,

    "AvatarPath" TEXT NOT NULL DEFAULT '/static/defaults/userAvatars/defaultUserAvatar.jpg',

    "UserName" TEXT NOT NULL,

    "Email" TEXT NOT NULL,

    "Language" TEXT NOT NULL DEFAULT 'ru',

    "Role" INTEGER NOT NULL DEFAULT 1,

    "PasswordHash" TEXT NOT NULL,

    "CreatedAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    "UpdatedAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    CONSTRAINT "Users\_pkey" PRIMARY KEY ("UserId")

);

–– CreateTable

CREATE TABLE "Chats" (

    "ChatId" SERIAL NOT NULL,

    "ChatName" TEXT NOT NULL,

    "IsGroup" BOOLEAN NOT NULL,

    "KeyHash" TEXT NOT NULL,

    "AvatarPath" TEXT NOT NULL DEFAULT '/static/defaults/chatAvatars/defaultChatAvatar.png',

    "CreatedAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    "UpdatedAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    CONSTRAINT "Chats\_pkey" PRIMARY KEY ("ChatId")

);

–– CreateTable

CREATE TABLE "Messages" (

    "MessageId" SERIAL NOT NULL,

    "ChatId" INTEGER NOT NULL,

    "SenderId" TEXT NOT NULL,

    "Content" TEXT NOT NULL,

    "MessageType" TEXT NOT NULL,

    "IsUpdate" BOOLEAN NOT NULL DEFAULT false,

    "IsRead" BOOLEAN NOT NULL DEFAULT false,

    "CreatedAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    "UpdatedAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    CONSTRAINT "Messages\_pkey" PRIMARY KEY ("MessageId")

);

–– CreateTable

CREATE TABLE "ChatMembers" (

    "UserId" TEXT NOT NULL,

    "ChatMemberId" SERIAL NOT NULL,

    "Role" INTEGER NOT NULL,

    "ChatId" INTEGER NOT NULL,

    "IsFixed" BOOLEAN NOT NULL DEFAULT false,

    "JoinedAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    CONSTRAINT "ChatMembers\_pkey" PRIMARY KEY ("ChatMemberId")

);

–– CreateTable

CREATE TABLE "StickerGroup" (

    "StickerGroupId" SERIAL NOT NULL,

    "GroupName" TEXT NOT NULL,

    CONSTRAINT "StickerGroup\_pkey" PRIMARY KEY ("StickerGroupId")

);

–– CreateTable

CREATE TABLE "Stickers" (

    "StickerId" SERIAL NOT NULL,

    "StickerGroupId" INTEGER NOT NULL,

    CONSTRAINT "Stickers\_pkey" PRIMARY KEY ("StickerId")

);

–– CreateIndex

CREATE UNIQUE INDEX "Users\_UserName\_key" ON "Users"("UserName");

–– CreateIndex

CREATE UNIQUE INDEX "Users\_Email\_key" ON "Users"("Email");

–– AddForeignKey

ALTER TABLE "Messages" ADD CONSTRAINT "Messages\_ChatId\_fkey" FOREIGN KEY ("ChatId") REFERENCES "Chats"("ChatId") ON DELETE RESTRICT ON UPDATE CASCADE;

–– AddForeignKey

ALTER TABLE "Messages" ADD CONSTRAINT "Messages\_SenderId\_fkey" FOREIGN KEY ("SenderId") REFERENCES "Users"("UserId") ON DELETE RESTRICT ON UPDATE CASCADE;

–– AddForeignKey

ALTER TABLE "ChatMembers" ADD CONSTRAINT "ChatMembers\_ChatId\_fkey" FOREIGN KEY ("ChatId") REFERENCES "Chats"("ChatId") ON DELETE RESTRICT ON UPDATE CASCADE;

–– AddForeignKey

ALTER TABLE "ChatMembers" ADD CONSTRAINT "ChatMembers\_UserId\_fkey" FOREIGN KEY ("UserId") REFERENCES "Users"("UserId") ON DELETE RESTRICT ON UPDATE CASCADE;

–– AddForeignKey

ALTER TABLE "Stickers" ADD CONSTRAINT "Stickers\_StickerGroupId\_fkey" FOREIGN KEY ("StickerGroupId") REFERENCES "StickerGroup"("StickerGroupId") ON DELETE RESTRICT ON UPDATE CASCADE;

# Приложение В

Хук отправки запросов и обработки ошибок «useApi»

import { useState } from 'react';

import { useNavigate } from 'react–router–dom';

import { AxiosError } from 'axios';

interface ServerError {

    error: boolean;

    show: boolean;

    message: string;

}

export function useApi<T, D = undefined>(request: (data?: D) => Promise<any>) {

    const [resData, setData] = useState<T | null>(null);

    const [loading, setLoading] = useState<boolean>(false);

    const navigate = useNavigate();

    const execute = async (body?: D) => {

        setLoading(true);

        try {

            const response = await request(body);

            setData(response.data);

        } catch (err) {

            const error = err as AxiosError<ServerError | any>;

            let errorMessage = 'Неизвестная ошибка';

            setData(null);

            if (error.response) {

                if (error.response.status == 401) {

                    navigate('/authorization');

                    return;                }

                const serverError = error.response.data;

                if (serverError && serverError.show === true) {

                    alert(serverError.message);

                    return;                }

                errorMessage =

          serverError.message || 'Отсутствует сообщение об ошибке';

               console.error('Ошибка ответа сервера:',errorMessage);

                navigate(

           `/error?message=${encodeURIComponent(  errorMessage,

                    )}&statusCode=${error.response.status}&statusText=${                        error.response.statusText   }`,);

            } else if (error.request) {

                errorMessage =

                'Сервер не ответил. Проверьте подключение к сети.';

                console.error('Сервер не ответил:', error.request);

                navigate(`/error?message=${encodeURIComponent(errorMessage)}`);

            } else {

                errorMessage = error.message || 'Ошибка запроса';

                console.error('Ошибка запроса:', errorMessage);

                navigate(`/error?message=${encodeURIComponent(errorMessage)}`);

            }

        } finally {

            setLoading(false);        }    };

    return { resData: resData, setResData: setData, loading, execute

Реализация HTTP–сервера на NodeJS

async function bootstrap() {

    const app = await NestFactory.create(AppModule);

    app.setGlobalPrefix('api');

    app.enableCors();

    app.use(cookieParser());

    const swaggerConfig = new DocumentBuilder()

        .setTitle('CryptogramWeb API')

        .setDescription('use this document to explore this API')

        .setVersion('1.0')

        .build();

    const swaggerDocument = SwaggerModule.createDocument(app, swaggerConfig);

    SwaggerModule.setup('swagger', app, swaggerDocument);

    await app.listen(3000);}

bootstrap();

Реализация подключения по протоколу «WebSocket»

handleConnection(client: Socket) {

        try {

            const token = client.handshake.query.token as string;

            if (!token) {

                throw new Error('Token not provided');

            }

            const secret = this.configService.get('JWT\_SECRET');

            if (!secret) {

                throw new Error('JWT\_SECRET is not defined');

            }

            const payload = this.jwtService.verify(token, { secret}

            const userId = payload.userId;

            if (!userId) {

                throw new Error('Invalid token payload');

            }

            this.connectedClients.set(userId, client.id);

        } catch (error) {

            client.disconnect();

            console.log('ConnectError – ' + error);

        }    }

Jotai–хранилище

export const darkModeAtom = atomWithStorage<boolean>('darkMode', true);

export const ruModeAtom = atomWithStorage<string>('langMode', 'ru');

export const wsTokenAtom = atomWithStorage<string>('wsTokenAtom', '');

export const openStickerPanelAtom = atomWithStorage<boolean>(

    'sticketPanel',

    false,

);

export const currentChatAtom = atomWithStorage<number>('currentChat', –1); // –1 mean that user don't choice chat else

export const myUserIdAtom = atomWithStorage<string>('myUserid', '');

export const socketAtom = atom<Socket | null>(null);

export const createSocketAtom = atom(

    (get) => get(socketAtom),

    (\_, set) => {

        const URI = window.location.origin;

        //const URI = 'http://localhost:3000';

        const token = localStorage.getItem('wsTokenAtom');

        const socket = io(URI, {

            query: {

                token: token?.slice(1, token.length – 1),

            },

        });

        set(socketAtom, socket);

        socket.on('connect', () => {

            console.log('Connected to WebSocket');

        });

        socket.on('disconnect', () => {

            console.log('Disconnected from WebSocket');

        });

    },

);

const safeLoad = (): Record<string, number> => {

    try {

        const data = localStorage.getItem('keyValueStorage');

        return data ? JSON.parse(data) : {};

    } catch (e) {

        console.error('Error parsing localStorage data:', e);

        return {};

    }

};

export const keyValueAtom = atomWithStorage<Record<string, number>>(

    'keyValueStorage',

    safeLoad(),

);

export const keyValueActionsAtom = atom(

    (get) => ({

        getCryptoKey: (key: string): number | null =>

            get(keyValueAtom)?.[key] ?? null,

    }),

    (

        get,

        set,

        action: { type: 'add' | 'delete'; key: string; value?: number },

    ) => {

        const prevState = get(keyValueAtom) || {};

        if (action.type === 'add' && typeof action.value === 'number') {

            set(keyValueAtom, {

                ...prevState,

                [action.key]: action.value,

            });

        } else if (action.type === 'delete') {

            const { [action.key]: \_, ...newState } = prevState;

            set(keyValueAtom, newState);

        }

    },

);

Yup схема валидации на примере регистрации

export const registrationSchema = Yup.object().shape({

    name: Yup.string().required('This field is required'),

    email: Yup.string()

        .required('This field is required')

        .email('Invalid email format')

        .matches(

            /^[a–zA–Z0–9.\_%+–]+@[a–zA–Z0–9.–]+\.[a–zA–Z]{2,}$/,

            'Invalid email format',

        ),

    username: Yup.string()

        .required('This field is required')

        .matches(

            /^[a–zA–Z0–9\_]+$/,

            'The user name can contain only English letters, numbers, and underscores',

        ),

    password: Yup.string()

        .required('This field is required')

        .min(8, 'password must be 8 least characters long'),

    repeatPassword: Yup.string()

        .required('This field is required')

        .min(8, 'password must be 8 least characters long')

        .oneOf([Yup.ref('password')], 'Passwords must match'),

});

Код метода расшифрования сообщения

c export function Decrypt(msg: string, key: number) {

    let encryptCodeMsg = [];

    let decryptCodeMsg = [];

    let decryptMsg = '';

    //msg = Buffer.from(msg, 'base64').toString();

    const codeKey = key.toString(2);

    const codeKeylength = codeKey.length;

    for (let i = 0; i < msg.length; i++) {

        encryptCodeMsg.push(msg.charCodeAt(i).toString(2).padStart(14, '0'));

    }

    let index = 0;

    decryptCodeMsg = encryptCodeMsg.map((encryptCodeMsgEl) => {

        let decryptCodeMsgEl = '';

        for (let i = 0; i < encryptCodeMsgEl.length; i++) {

            decryptCodeMsgEl += XoR(encryptCodeMsgEl[i], codeKey[index]);

            index++;

            if (index == codeKeylength) index = 0;

        }

        return decryptCodeMsgEl;

    });

    for (let i = 0; i < decryptCodeMsg.length; i++) {

        decryptMsg += String.fromCharCode(parseInt(decryptCodeMsg[i], 2));

    }

    return decryptMsg;

}

function XoR(el1: string, el2: string): string {

    return el1 === el2 ? '0' : '1';

}

Код метода зашифрования сообщения

export function Encrypt(msg: string, key: number) {

    let codeMsg = [];

    let encryptCodeMsg = [];

    let encryptMsg = '';

    const codeKey = key.toString(2);

    const codeKeylength = codeKey.length;

    for (let i = 0; i < msg.length; i++) {

        codeMsg.push(msg.charCodeAt(i).toString(2).padStart(14, '0')); }

    let index = 0;

    encryptCodeMsg = codeMsg.map((codeMsgEl) => {

        let encryptCodeMsgEl = '';

        for (let i = 0; i < codeMsgEl.length; i++) {

            encryptCodeMsgEl += XoR(codeMsgEl[i], codeKey[index]);

            index++;

            if (index == codeKeylength) index = 0;

        }

        return encryptCodeMsgEl;

    });

    for (let i = 0; i < encryptCodeMsg.length; i++) {

        encryptMsg += String.fromCharCode(parseInt(encryptCodeMsg[i], 2));

    }

    return encryptMsg;

}

function XoR(el1: string, el2: string): string {

    return el1 === el2 ? '0' : '1';

}

Конфигурация WebPuck для продакшона

const { merge } = require('webpack–merge');

const common = require('./webpack.common.js');

const MiniCssExtractPlugin = require('mini–css–extract–plugin');

module.exports = merge(common, {

  mode: 'production',

  output: {

    filename: '[name].[contenthash].js',

  },

  module: {

    rules: [

      {

        test: /\.scss$/i,

        use: [MiniCssExtractPlugin.loader, 'css–loader', 'sass–loader'],

      },

    ],

  },

  plugins: [new MiniCssExtractPlugin()],

});

# Приложение Г

Автоматизированные тесты для клиентской части приложения

// Import необходимых библиотек

import { render, screen, fireEvent } from '@testing–library/react';

import '@testing–library/jest–dom';

import ChatList from '../components/ChatList';

import SendMessageButton from '../components/SendMessageButton';

import RegistrationForm from '../components/RegistrationForm';

import DeleteChatButton from '../components/DeleteChatButton';

// Тест для проверки отображения списка чатов

test('Отображение списка чатов', () => {

const chats = ['Chat 1', 'Chat 2', 'Chat 3'];

render(<ChatList chats={chats} />);

chats.forEach(chat => {

expect(screen.getByText(chat)).toBeInTheDocument();

});

});

// Тест для кнопки отправки сообщения

test('Кнопка отправки сообщения отображается и активна', () => {

render(<SendMessageButton isActive={true} />);

const button = screen.getByRole('button', { name: /отправить/i });

expect(button).toBeInTheDocument();

expect(button).toBeEnabled();

});

// Тест для кнопки удаления чата

test('Кнопка удаления чата вызывает API удаление', () => {

const handleDelete = jest.fn();

render(<DeleteChatButton onDelete={handleDelete} />);

const button = screen.getByRole('button', { name: /удалить/i });

fireEvent.click(button);

expect(handleDelete).toHaveBeenCalledTimes(1);

});

// Тест для валидации формы регистрации

test('Ошибка отображается при некорректном вводе email', () => {

render(<RegistrationForm />);

const emailInput = screen.getByLabelText(/email/i);

const submitButton = screen.getByRole('button', { name: /зарегистрироваться/i });

fireEvent.change(emailInput, { target: { value: 'invalid–email' } });

fireEvent.click(submitButton);

const errorMessage = screen.getByText(/некорректный email/i);

expect(errorMessage).toBeInTheDocument();

Тест для проверки работоспособности приложения

describe('AppController (e2e)', () => {

    let app: INestApplication;

    beforeEach(async () => {

        const moduleFixture: TestingModule = await Test.createTestingModule({

            imports: [AppModule],

        }).compile();

        app = moduleFixture.createNestApplication();

        await app.init();

    });

    it('/ (GET)', () => {

        return request(app.getHttpServer())

            .get('/')

            .expect(200)

            .expect('Hello World!');

    });

});