

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Интеллектуальных систем и программирования

Кафедра «Программная инженерия» им. Л.П. Фельдмана

Направление подготовки 09.03.04 Программная инженерия

Направленность (профиль) Инженерия программного обеспечения

Допустить к защите

Заведующий кафедрой программной
инженерии им. Л.П. Фельдмана


Zori A.S.
« » июня 2024г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
БАКАЛАВРА**

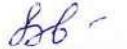
на тему:

Многопользовательский чат-мессенджер

Студент

Безуглый Виталий Витальевич

(фамилия, имя, отчество)



(подпись)

Руководитель

ст. препод. каф. ПИ Щедрин Сергей Валерьевич

(должность, ученая степень, ученое звание, фамилия, имя, отчество)



(подпись)

Консультанты

(раздел, должность, ученая степень, ученое звание, фамилия, имя, отчество)

(подпись)

Нормоконтролёр

старший преподаватель кафедры ПИ Коломойцева И.А

(должность, ученая степень, ученое звание, фамилия, имя, отчество)



(подпись)

ДОНЕЦК-2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Интеллектуальных систем и программирования
Кафедра «Программная инженерия» им. Л.П. Фельдмана
Направление подготовки 09.03.04 Программная инженерия
Направленность (профиль) Инженерия программного обеспечения

УТВЕРЖДАЮ:

Зав. кафедрой

С. А. Зори

(подпись) (ФИО)
« 15 » апреля 2024 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра
Студенту Безуглому Виталию Витальевичу

(фамилия, имя, отчество)

1. Тема ВКР: Многопользовательский чат-мессенджер

Название спецраздела Разработка серверной и клиентской части мессенджера
средствами Golang, HTML ,CSS, JS

утверждена приказом по университету от « 26 » марта 2024г. № 222-14

2. Исходные данные: результаты НИРС и преддипломной практики

3. Содержание ВКР (перечень, подлежащих разработке разделов): введение;
системный анализ предметной области; анализ требований к мессенджеру; проектирование архитектуры программного обеспечения; проектирование базы данных; разработка программных модулей сайта; описание алгоритмов; описание функций; описание программных модулей; тестирование ; заключение; список используемых источников

4. Перечень графического материала: диаграмма прецедентов; диаграмма состояний;
диаграмма компонентов; схема базы данных; схема движения запроса; скриншот окна чата;
скриншот окна авторизации.

5. Данные о консультантах, с указанием разделов пояснительной записки

Раздел ВКР	Консультант (должность, уч. степень, уч. звание, Ф.И.О.)	Задание выдал (подпись)	Задание принял (подпись, дата)
Нормоконтроль	Ст. преподаватель И.А. Коломойцева	<i>М.Ю.</i>	<i>М.Ю.</i> 10.06.24

Дата выдачи задания « 15 » апреля 2024 г.

Срок сдачи ВКР « 17 » июня 2024 г.

Руководитель *h* Щедрин Сергей Валерьевич
(Ф.И.О.)

Задание принял к исполнению *86-*
(подпись)

КАЛЕНДАРНЫЙ ПЛАН

№ п/п	Этапы выполнения ВКР	Срок выполнения этапов
1	Проведение системного анализа предметной области и общая постановка задачи, разработка требований к программной системе	15.04.2024
2	Проектирование архитектуры программной системы	20.04.2024
3	Разработка основных алгоритмов клиента и сервера	22.04.2024
4	Проектирование базы данных, сервера и клиента	24.04.2024
5	Разработка базы данных	26.04.2024
6	Разработка серверной программы	05.05.2024
7	Разработка клиентской программы	15.05.2024
8	Тестирование системы	25.05.2024
9	Написание основной части пояснительной записки	29.05.2024
10	Написание спец. части пояснительной записки	03.06.2024
11	Написание приложений для пояснительной записки	06.06.2024

Студент/слушатель *86-*
(подпись)

Руководитель ВКР *A*
(подпись)

РЕФЕРАТ

Пояснительная записка к дипломному проекту бакалавра: 112 страниц, 65 рисунков, 8 таблиц, 7 источников, 3 приложения.

Объект исследования – многопользовательский чат-мессенджер «Что случилось?»

Предмет исследования – Разработка серверной и клиентской части сайта-мессенджера средствами Golang ECHO, HTML, CSS, JS, Ajax, jQuery, MariaDB.

Цель данной работы заключается в освоении принципов разработки чат-мессенджеров приобретении теоретических знаний и практических навыков в разработке базы данных, разработке клиент-серверных приложений. В рамках исследования выполнен анализ аналогичных игровых систем, и выявление лучших качеств для чат-мессенджера..

В разработанном чате задействованы средства фреймворка ECHO, вёрстки на HTML, CSS, JS, JQUERY, а также AJAX инструмент для обмена данными с MariaDB. Основной акцент сделан на актуальности технологий разработки, создании удобного интерфейса пользователя, а также разработке программных модулей серверной части с целью дальнейшего развития проекта.

ЧАТ, МЕССЕНДЖЕР, СОЦИАЛЬНАЯ СЕТЬ, АРХИТЕКТУРА, БАЗА ДАННЫХ, GOLANG, JQUERY, ECHO, WEB, MARIADB

ABSTRACT

The object of the study - Multi-user chat messenger "What happened?"

The subject of the study is the development of the server and client side of the messenger site using Golang ECHO, HTML, CSS, JS, Ajax, jQuery, MariaDB.

The purpose of this work is to master the principles of chat messenger development, acquire theoretical knowledge and practical skills in database development, and develop client-server applications. As part of the study, an analysis of similar gaming systems was performed, and the identification of the best qualities for a chat messenger..

The developed chat uses the ECHO framework, HTML, CSS, JS, JQUERY layout tools, as well as an AJAX tool for data exchange with MariaDB. The main focus is on the relevance of development technologies, the creation of a user-friendly user interface, as well as the development of software modules for the server side in order to further develop the project.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 СИСТЕМНЫЙ АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	8
1.1 Общая характеристика задачи	8
1.2 Проблемы автоматизации	10
1.3 Анализ существующих систем	11
1.4 Анализ существующих программных средств реализации	17
1.5 Постановка задача	19
2 АНАЛИЗ ТРЕБОВАНИЙ.....	20
3 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ	22
4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ	28
5 ПРОЕКТИРОВАНИЕ ОБЪЕКТНЫХ МОДУЛЕЙ	36
6 ОПИСАНИЕ ПРОГРАММЫ	40
6.1 Алгоритмы	40
6.2 Описание функций	43
6.3 Описание программных модулей	50
7 ОПИСАНИЕ ИНТЕРФЕЙСА	57
8 ТЕСТИРОВАНИЕ	62
ЗАКЛЮЧЕНИЕ	76
СПИСОК ЛИТЕРАТУРЫ	77
ПРИЛОЖЕНИЕ А. Фрагменты программного кода	78
ПРИЛОЖЕНИЕ Б. Графическая часть.....	107
ПРИЛОЖЕНИЕ В. Перечень замечаний нормоконтролёра	112

ВВЕДЕНИЕ

Ни одна деятельность человека или компании не обходится без общения.

Каждой организации необходима платформа для безопасного общения между сотрудниками. Технологии позволяют оптимизировать, упростить и улучшить этот сложный процесс. Внедрение собственной платформы позволяет свести к минимуму возможность перехвата важных данных.

Преимущества уникального чат-мессенджера: экономия времени, более эффективное использование ресурсов, повышение прозрачности общения в компании, сохранение истории каждого взаимодействия, гибкость.

Концепция чат-мессенджера подразумевает особый способ общения между людьми по всему миру. Данная тема очень актуальна, так как чат решает многие проблемы любого учреждения.

При выполнении данной работы необходимо не только изучить принципы работы существующих систем, но и реализовать "каркас" чата и модули для работы с базой данных. При этом необходимо учитывать универсальность системы, так как данный продукт может использоваться как в некоммерческих целях, так и на предприятиях.

1 СИСТЕМНЫЙ АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Общая характеристика задачи

Чат-мессенджер – это многопользовательская система для формирования, передачи и хранения сообщений [1].

Задачи чат-мессенджера.

1. Функциональность чата. Основная цель сайта - обеспечить пользователей средствами обмена сообщениями в режиме реального времени. Это включает в себя возможность отправки текстовых сообщений, использование смайликов, возможность отправки файлов (изображений, аудио, видео и документов), а также функции, такие как групповые чаты и приватные сообщения.

2. Регистрация и аутентификация. Пользователи должны иметь возможность создания учетной записи и входа в систему. Для безопасности и конфиденциальности информации важно реализовать механизм аутентификации с использованием паролей или других методов (например, двухфакторной аутентификации).

3. Безопасность. С учетом важности конфиденциальности переписки, необходимо обеспечить безопасность передаваемых данных, в том числе с использованием шифрования сообщений.

4. Интерфейс пользователя. Веб-интерфейс должен быть интуитивно понятным и удобным для использования. Это включает в себя разработку удобного дизайна, легкость навигации и поддержку мобильных устройств.

5. Хранение сообщений. Важно иметь эффективную систему хранения сообщений для обеспечения доступности переписки пользователей и возможности просмотра истории сообщений.

6. Масштабируемость. Сайт должен быть способен обрабатывать большое количество пользователей и сообщений, поэтому важно учитывать масштабируемость при разработке архитектуры системы.

7. Мобильная совместимость. Учитывая популярность мобильных устройств, важно обеспечить полноценную работу чат-мессенджера на мобильных платформах, возможно, с помощью мобильных приложений или адаптивного веб-дизайна.

Основная цель коммуникации – облегчение общения пользователей в сети.

Основные этапы работы с сообщениями:

- обработка сообщения;
- сохранение сообщения в БД;
- отправка сообщения получателю.

Рассмотрим на примере диаграммы состояния жизненный цикл сообщения в системе (см. рис. 1.1).

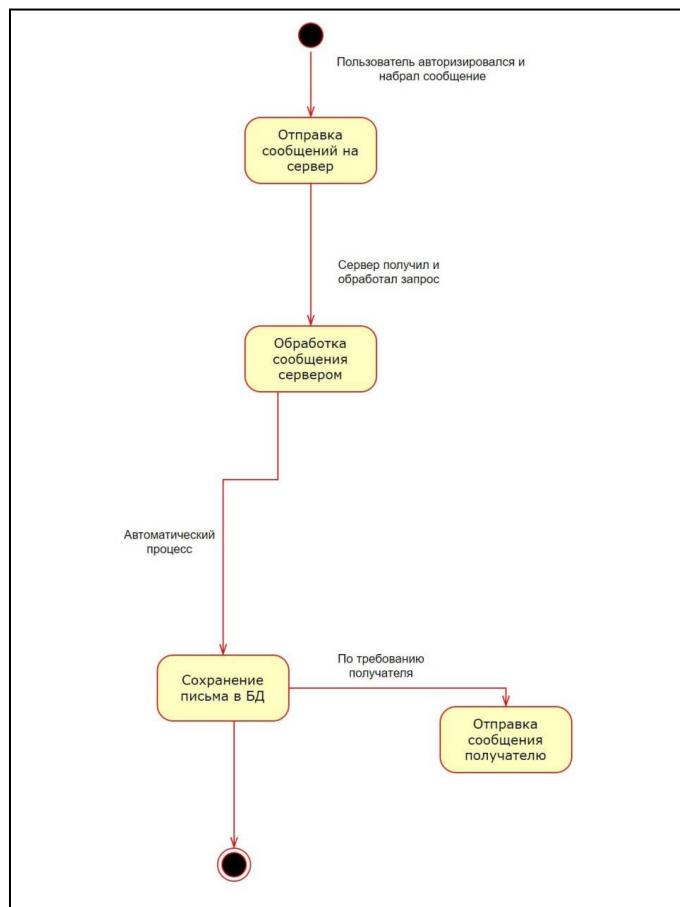


Рисунок 1.1 – Состояния сообщения на различных этапах работы коммуникации

В рамках дипломного проекта предусмотрена разработка модуля коммуникации. Принцип работы следующий:

Для отправки сообщения требуется:

- аккаунт отправителя;
- отправитель (по умолчанию);
- получатель (можно выбрать из графического интерфейса);
- текст сообщения;
- вложение (опционально).

На данном этапе проводится последовательное сохранение сообщения. Получатель может авторизироваться на сайте и получить новое сообщение.

1.2 Проблемы автоматизации

Мессенджер решает множество проблем учреждения, таких как экономии времени на коммуникацию, сохранность данных и т.д.. Достигается всё за счёт автоматизации многих действий пользователя.

Системы коммуникации помогают выстроить простую и безопасную цепочку действий пользователя, приводящую к ожидаемому результату. Автоматизация различных процессов избавляет от недочетов, слабых мест в чате.

Мессенджер повышает производительность работы, так как позволяет:

- коммуницировать с любого места;
- надёжно хранить файлы на сервере;
- фиксировать историю истории диалогов;
- снизить число ошибок из-за несогласованности действий.

1.3 Анализ существующих систем

1. WhatsApp (2009) на данный момент подразделение Facebook.

«WhatsApp» — это легендарный мессенджер, разработанный Ян Кумом и выпущенный в 2009 году. Эта самый обычный мессенджер в базовом его понимании [2].

Интерфейс мессенджера изображен на рисунке 1.2.

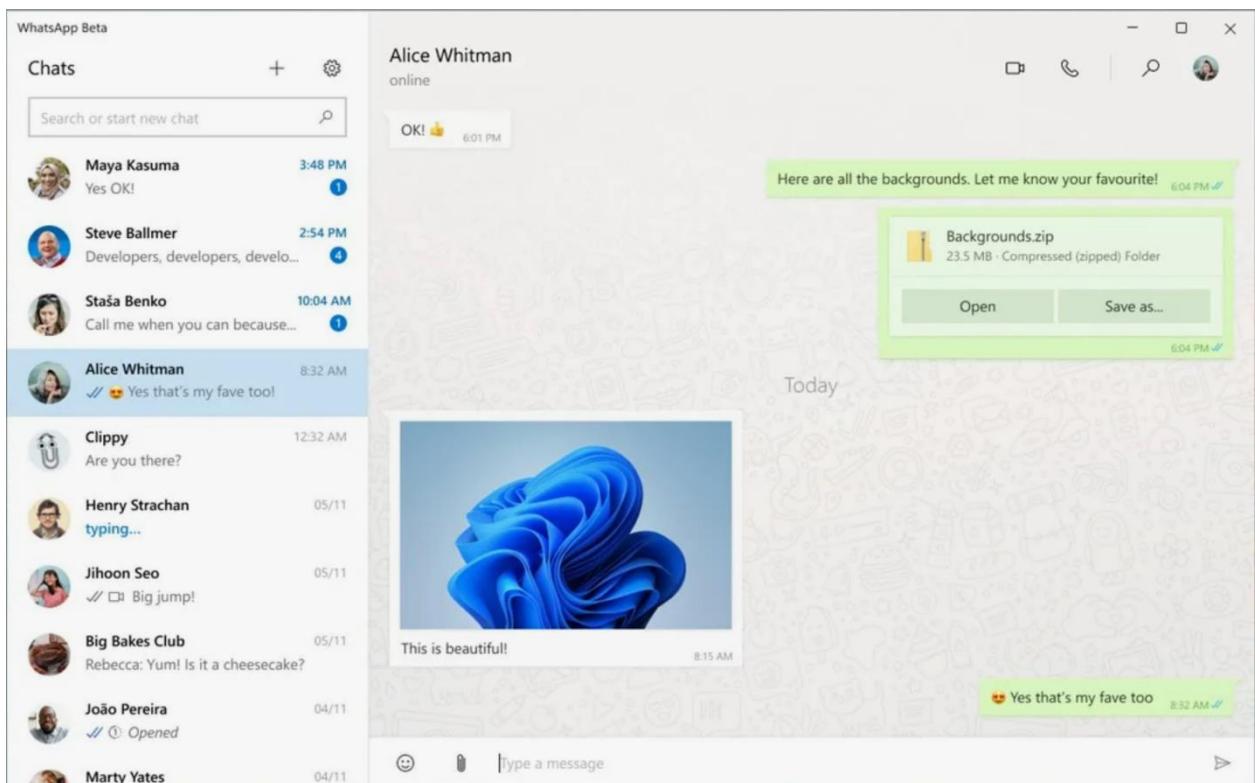


Рисунок 1.2 – Интерфейс WhatsApp

Мессенджер представляет собой мобильное, десктопное приложение для общения между пользователями в записной книжке смартфона. Дизайн приложения создан настолько простым, что с ним разберётся даже не самый продвинутый пользователь интернета.

Чат предлагает относительно скучный набор инструментов. Разработчики не торопятся добавлять современные функции для своих

пользователей. Так ,например, свет не увидели функции самоудаляющихся сообщений.

В свое время «WhatsApp» впечатляла тем, что был похож на приложение «Сообщения» на Android и IOS и можно было без оплаты оператору сотовой связи связываться с родственниками и друзьями в понятном интерфейсе.

Основатели компании внедрили очень простую и удобную систему регистрации в приложении, только по номеру телефона, без сбора конфиденциальных данных пользователей. Переписка является приватной, никто не сможет ее расшифровать.

Кум и Эктон говорили, что не планировали создавать компанию, а хотели лишь создать продукт, которым будет удобно пользоваться абсолютно каждому человеку. Важно отметить, что основатели WhatsApp были противниками рекламы, их девиз: «Никакой рекламы! Никаких игр! Никаких уловок!». Они считали, что маркетинг пускает пыль в глаза, не давая возможности сфокусироваться на самом продукте. Они не тратили средства на привлечение пользователей, полагаясь на органический рост.

2. «WeChat» (2010) работа исследовательского центра Tencent.

За каждым великим делом стоит великая личность. В случае с WeChat это программист Аллен Чжан , чье имя на родине при жизни обросло легендами. Чжан родился в 1969 году, в крестьянской семье. Его с детства тянуло к науке и он, помимо занятий в школе, постоянно учился сам, поглощая книгу за книгой. Это позволило мальчику из глухой деревни поступить в Хуачжунский университет науки и технологий (Ухань) и получить степень магистра телекоммуникационной инженерии в 1994-м году.

Изначально над проектом работала команда из семи человек во главе с Чжаном. К ноябрю 2010-го у них был готов прототип, носивший китайское название Weixin. В основном, приложение копировало Kik, мало

чем отличалось от конкурентов и было запущено в январе 2011-го без особого шума и рекламы.

В начале 2012-го в мессенджере появилась одна из важнейших составляющих будущей экосистемы – QR-код. Каждый пользователь получал уникальный код и мог, просканировав код на смартфоне другого человека, добавить его в друзья, посмотреть каталог товаров, меню, расписание. Через три месяца (14 после первоначального запуска) количество подписчиков выросло до 100 млн.

Интерфейс мессенджера изображен на рисунке 1.3.

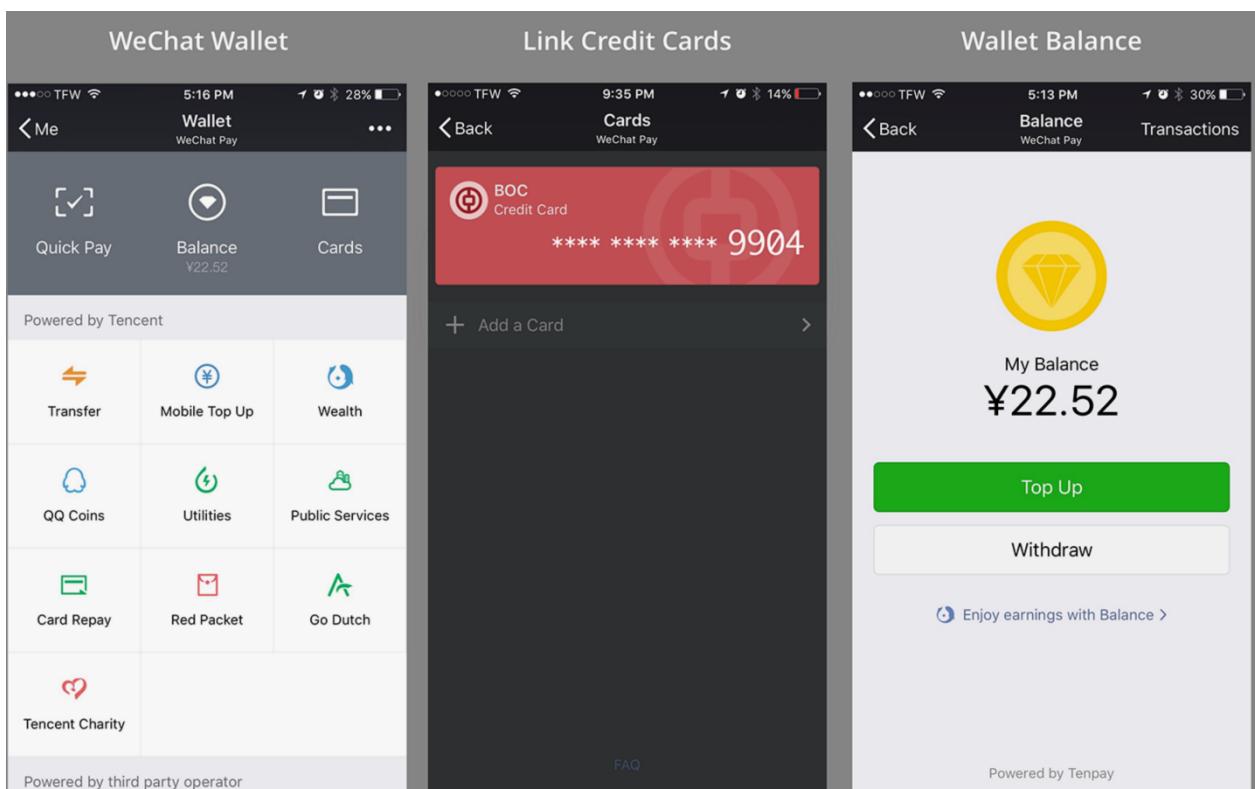


Рисунок 1.3 – Интерфейс WeChat

По данным за 2021-й год WeChat активно пользуются 1,2 миллиарда человек ежемесячно, каждый из которых тратит на него в среднем 77 минут в день, а 17% ежедневно проводит в приложении по 4 часа и больше.

3. «Telegram» (2013) Павел Дуров.

Telegram – это популярный чат-инструмент, доступный на 32 языках. За десять с лишним лет существования Telegram постоянно входит в пятерку самых скачиваемых приложений в мире, а к 2023 году число его пользователей достигнет 800 миллионов в месяц.

Со слов самого Павла Дурова, идея создания мессенджера пришла к нему еще в 2011 году. В тот момент на фоне проблем с правоохранительными органами он осознал, что у него нет безопасного способа коммуникации с близкими. Брат Павла, Николай Дуров, разработал криптографический протокол для шифрования переписки MTProto, который лег в основу Телеграм.

Интерфейс мессенджера изображен на рисунке 1.4.

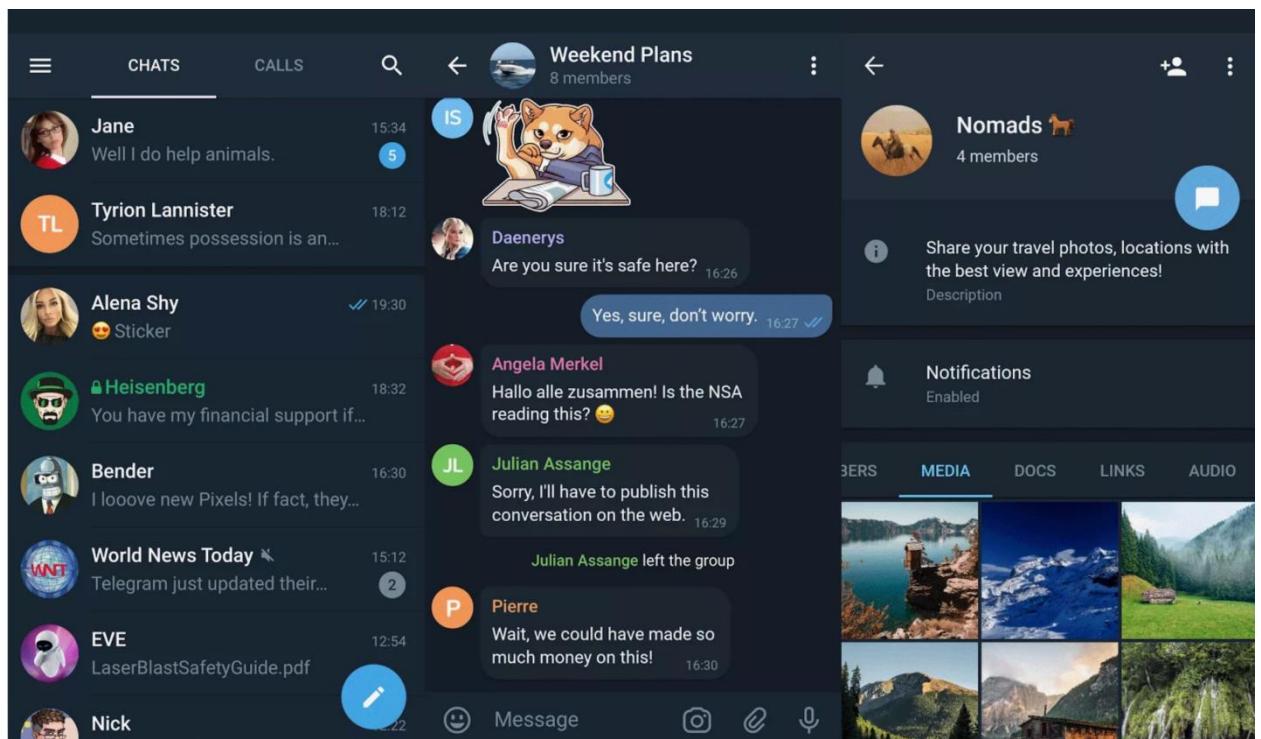


Рисунок 1.4 – Интерфейс Telegram

Одной из ключевых особенностей «Telegram» является защищенность. Разработанный протокол является безопасным способом

передавать сообщений. Это и легло в основу незаконной деятельности внутри мессенджера.

16 апреля 2018 года на территории России начался процесс ограничения доступа к мессенджеру Telegram.

Предпосылкой блокировки мессенджера стали поправки в законе, имеющие антитеррористическую направленность.

Это не отменяет факта популярности мессенджера на территории СНГ. Telegram является тем, что представляет собой безопасный мессенджер.

Сравнение основных характеристик аналогов систем по основным функциональным характеристикам представлено в таблице 1.1.

Таблица 1.1 — Анализ функциональных характеристик программных систем

Характеристики / Чаты	WhatsApp	WeChat	Telegram
Год выпуска	2009	2010	2013
Отправка сообщений	Есть	Есть	Есть
Отправка голосовых сообщений	Есть	Есть	Есть
Поддержка платформ	Windows, MacOS, Android, iOS, S40, KaiOS, WEB	Windows, MacOS, Android, iOS, S40/J2ME, Windows Phone, Symbian, BlackBerry	Windows, MacOS, Android, iOS, WEB

Продолжение таблицы 1.1

Функция самоуничтожения сообщений	Доступна для отдельных диалогов	Нет	Есть
Цифровые звонки и видеозвонки	Есть	Есть	Есть
Защита сообщений	Присутствует. Сообщения хранятся у пользователей.	Присутствует	Присутствует
Платежная система	Нет	Есть. Самая популярная в Китае платежная система.	Есть. Криптовалюты GRAM, TON.

На основе проведённого анализа мессенджеров следует отметить, что ни один из них не является идеальным, но у каждого есть свои преимущества, которые выделяют его на фоне других.

Например, WhatsApp, хоть и не хранит письма на сервере и в случае утери доступа к устройству теряются и переписки но является безопасным от взлома серверов WhatsApp. WeChat несмотря на лучшую платёжную систему, не поддерживает базовую на текущее время функцию самоудаляющихся сообщений. Telegram, хотя и имеет криптовалютные наработки но не раз попадались на обмане пользователей и привлекались к ответственности.

Telegram – для тех, кто ценит конфиденциальность и широкие возможности. WhatsApp – для массового общения с простым и интуитивно понятным интерфейсом. WeChat – для всесторонней интеграции в жизнь и пользования множеством дополнительных сервисов.

1.4 Анализ существующих программных средств реализации

За время существования чатов было создано множество проектов. Как следствие, существует множество вариантов решений, связанных с используемыми инструментами.

В большинстве случаев используется клиент-серверная система - основные модули управления сообщениями и данными располагаются на специальном сервере, а клиентская часть представляет собой интерфейс для взаимодействия пользователя с системой;

Клиент-серверный подход выбран для реализации чатов. Все данные хранятся и обрабатываются на сервере, а пользователь манипулирует ими с помощью клиентского приложения. Однако для создания клиентского приложения мы используем веб-технологии.

Преимущества такого подхода - быстрая обработка данных и надежность системы. Самое главное преимущество веб-технологий - их кроссплатформенность. Доступ к сайтам можно получить практически с любого устройства.

В качестве СУБД для создания серверной части рабочего процесса была выбрана MariaDB [3].

MariaDB – это объектно-реляционная система управления базами данных. Ключевой особенностью объектно-реляционной базы данных является поддержка определяемых пользователем объектов и их поведения, таких как типы данных, функции, операции, домены и индексы. Это делает выбранную СУБД очень гибкой и надежной.

Клиентская часть реализована с помощью веб-технологий, таких как HTML, CSS и JavaScript (jQuery). Рассмотрим, почему были выбраны именно эти технологии.

HTML – это язык гипертекстовой разметки, который используется для отображения веб-страниц. Все содержимое хранится в статических файлах.

Преимуществами HTML-приложений являются:

- легкий вес;
- экономичное потребление ресурсов;
- высокая стабильность (ошибка на одной странице не влияет на весь сайт);
- поддерживается широким спектром браузеров [4].

CSS – это язык для обозначения стиля сайта, предназначенный для улучшения процесса взаимодействия с веб-страницами. Каскадные таблицы стилей задают внешний вид страниц.

CSS также обладает рядом преимуществ:

- простота использования;
- экономия времени (один и тот же код можно применять к разным страницам);
- позволяет адаптировать страницы под разные устройства;
- глобальные веб-стандарты;
- приложения могут хранить CSS локально в автономном кэше;
- CSS не зависит от платформы клиента [5].

JavaScript присутствует на странице для обработки действий пользователя, и его положительными качествами являются:

- он быстр для конечного пользователя (действия выполняются локально, а не на сервере);
- он использует модель DOM и предоставляет множество функций для различных объектов на странице;
- универсальность.

Надстройка jQuery[6] упрощает написание кода и сокращает общий объем проекта, что положительно сказывается на читаемости кода.

Предполагается, что сервер будет написан на языке программирования Golang[7].

Golang обладает рядом неоспоримых преимуществ:

- высокая скорость и общая производительность;

- простой синтаксис (сходство с C++ и Python);
- многозадачность и широкие возможности разработки;
- язык разработан как серверный.

Поэтому выбор технологии рационален и дает возможность писать качественные, гибкие системы.

1.5 Постановка задача

В результате проведенного анализа существующих мессенджеров, выявленных недостатков и учтённых требований пользователей к подобным системам, принято решение о создании чата-мессенджера для демонстрации основных возможностей при разработке веб сайтов на Golang, с использованием базы данных MariaDB.

Цель работы – разработка веб приложения, представляющего собой мессенджер, демонстрирующую основные механики при создании веб сайта, такие как: дизайн вёрстки, вёрстка веб-сайта, разработка БД, интеграция бэкэнд-составляющей.

Задачи работы:

- разработка архитектуры системы (определение структуры приложения);
- создание интерфейса пользователя (проектирование простого и интуитивно понятного пользовательского интерфейса, включая страницы входа и страницы чатов);
- разработка программных модулей бэкэнд (реализация механики обмена сообщениями, хранения сообщений в базе данных);
- тестирование разработанной системы (проведение тестов, включающих проверку чата на работоспособность, корректность работы интерфейса и обработку сценариев использования);
- формулировка выводов и областей дальнейшего совершенствования системы.

2 АНАЛИЗ ТРЕБОВАНИЙ

Требование к программному обеспечению – это функциональная или нефункциональная потребность, которая должна быть реализована в системе.

Функциональная потребность подразумевает предоставление пользователю определённой услуги. Требование к программному обеспечению также может быть нефункциональным, это может быть требование к производительности и интерфейсу [7].

Рассмотрим бизнес-требования для программного обеспечения для системы обмена сообщениями:

- система должна обеспечить обмен информацией между пользователями;
- пользователь должен иметь возможность покинуть свой аккаунт;
- система должна снизить время на коммуникацию корпоративных пользователей;
- система должна оптимально использовать серверное время;
- система должна оптимально использовать ресурсы устройств пользователей.

Следующий этап анализа требований – формирование функциональных требований к системе:

- информация о сообщениях при получении в списке должна содержать информацию: время отправки, вложение (если есть), текст сообщения;
- должна быть удобная система поиска пользователей;
- разработать систему отправки файла с базы данных пользователю (на случай запроса пользователем вложения);
- должно быть предусмотрено примитивное шифрование всего контента в обязательном автоматическом порядке;

- в системе должно быть руководство пользователя;
- необходима возможность создания неограниченного количества сообщений;
- должна быть возможность регистрации и авторизации уже зарегистрированных пользователей;
- необходимо обеспечить сохранность данных при потере соединения с сервером;
- должна быть возможность сохранения активных диалогов на сервере.

Рассмотрим не функциональные требования к интерфейсу:

- система должна иметь простой и понятный интерфейс;
- форма авторизации должна содержать два текстовых поля для ввода логина и пароля, а также кнопку для входа;
- сообщение об ошибке выводить в виде отдельной страницы;
- на странице чата сделать меню в шапке, в котором реализовать выход из аккаунта и переключение цветовой темы.
- реализовать добавление смайликов к сообщению.

Важные аспекты к требованиям программного обеспечения:

- требования должны быть конкретными, измеримыми, достижимыми, релевантными и ограниченными по времени (SMART);
- документ с требованиями должен быть понятен всем участникам проекта.

Требования к программному обеспечению – это важный инструмент, который помогает создать качественный и эффективный программный продукт, соответствующий потребностям пользователей.

3 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Следующие вопросы оказывают существенное влияние на архитектуру разрабатываемого приложения

Параллелизм, включая вопросы организации процессов, подходы и методы.

Обработка ошибок (особые случаи) и отказоустойчивость. Проблему этой темы можно сформулировать очень просто: как предотвратить сбои или гарантировать дальнейшую функциональность системы в случае сбоя.

Взаимодействие и представление – взаимодействие между пользователем и системой, т.е. представление информации пользователю и реакция системы на действия пользователя [8].

Для создания приложения была выбрана архитектура клиент-сервер, поскольку она имеет множество преимуществ.

При создании приложения очень важно представить разрабатываемую архитектуру графически. Для реализации этого используются язык UML.

Унифицированный язык моделирования (UML) – это графический язык для визуализации, спецификации, построения и документирования систем, в которых основную роль играет программное обеспечение С помощью UML можно создавать детальные планы создаваемой системы, включая системные функции, бизнес-процессы и другие концептуальные элементы. детальные планы создаваемой системы, включая концептуальные элементы, такие как системные функции, бизнес-процессы и т. д. [9].

Для начала построим абстрактную модель разрабатываемого модуля – диаграмму вариантов использования. Основываясь на требованиях к системе, можем создать обзор приложения.

В системе предусмотрено один вида пользователей: собеседник. Собеседник получает весь функционал продукта (кроме конфиденциальной информации других пользователей).

Составим диаграмму вариантов использования чата на примере двух пользователей (см. рис. 3.1).

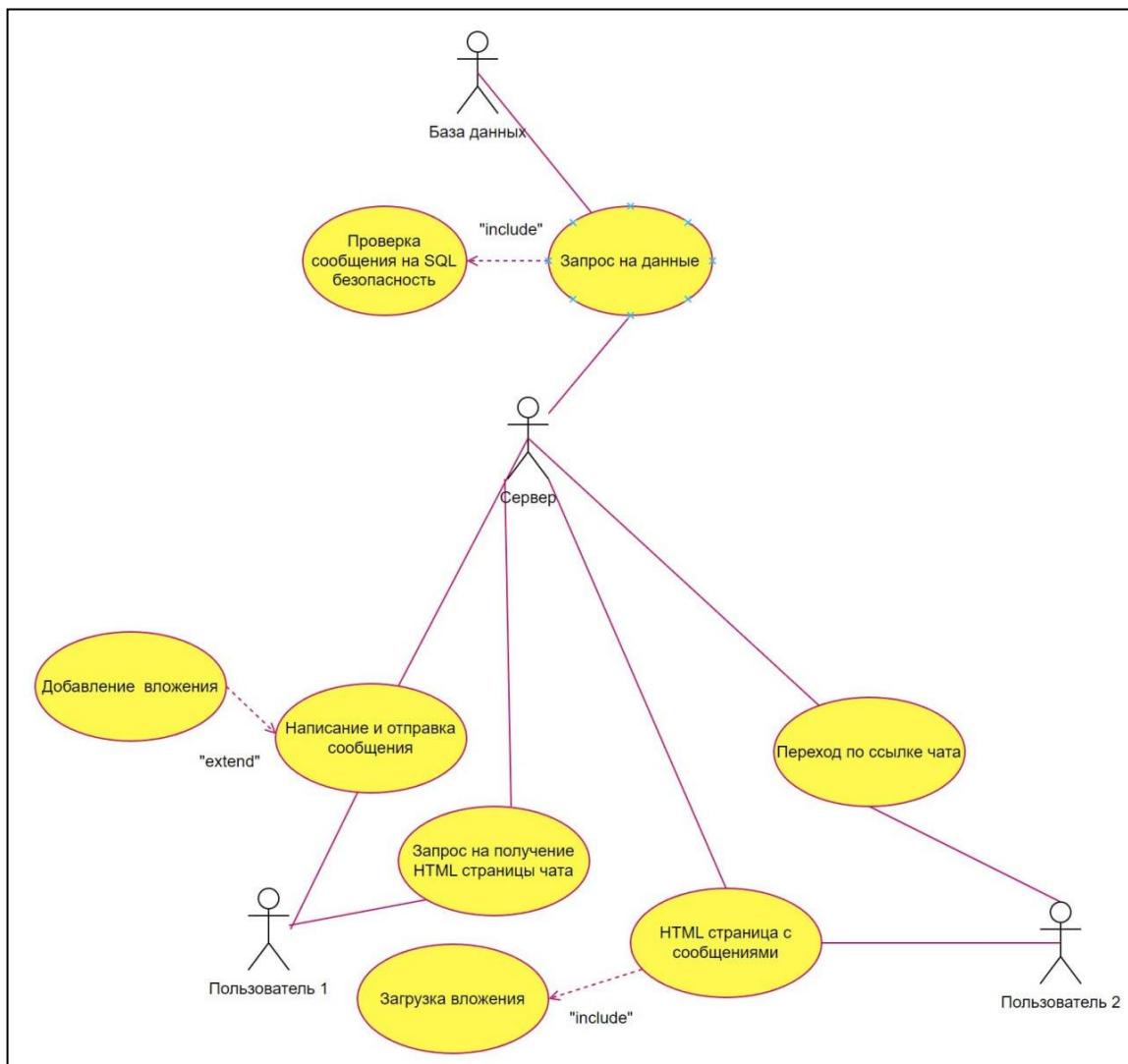


Рисунок 3.1 – Диаграмма прецедентов для двух пользователей чата

Стоит выделить какие участники предусмотрены и что им позволено делать:

- «Пользователь 1» – отправитель сообщения;
- «Пользователь 2» – получатель сообщения.

Рассмотрим диаграмму вариантов использования для пользователей (см. рис. 3.2).

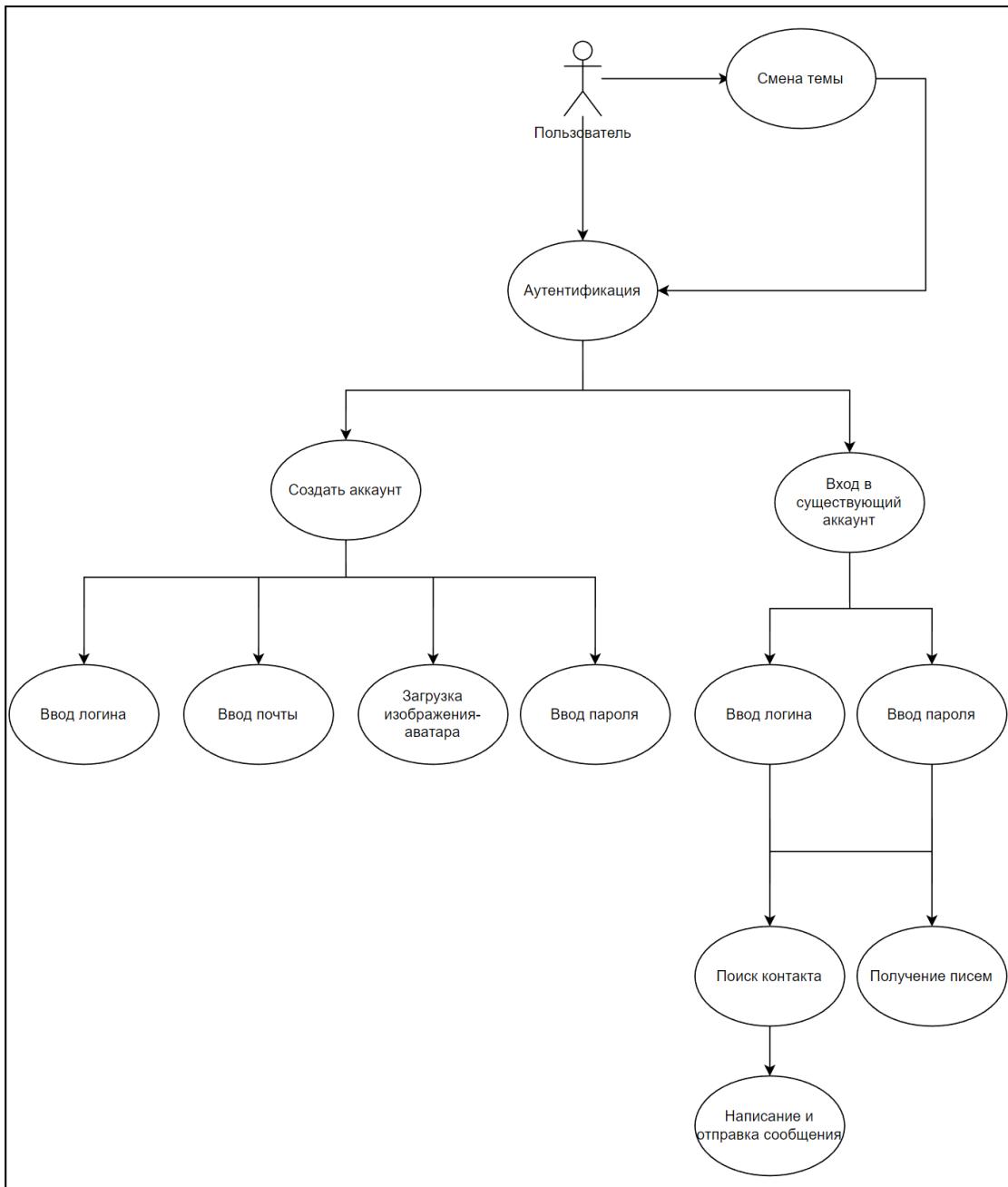


Рисунок 3.2 – Диаграмма вариантов использования для пользователя

Как видно на диаграмме, пользователь может участвовать как в отправке так и в получении сообщения (по сценарию рисунка 3.1).

Ниже представлена диаграмма состояний. На ней отображается жизненный цикл объекта, начиная с момента его создания и заканчивая разрушением. Таким объектом будет выступать сообщение (см. рис. 3.3).

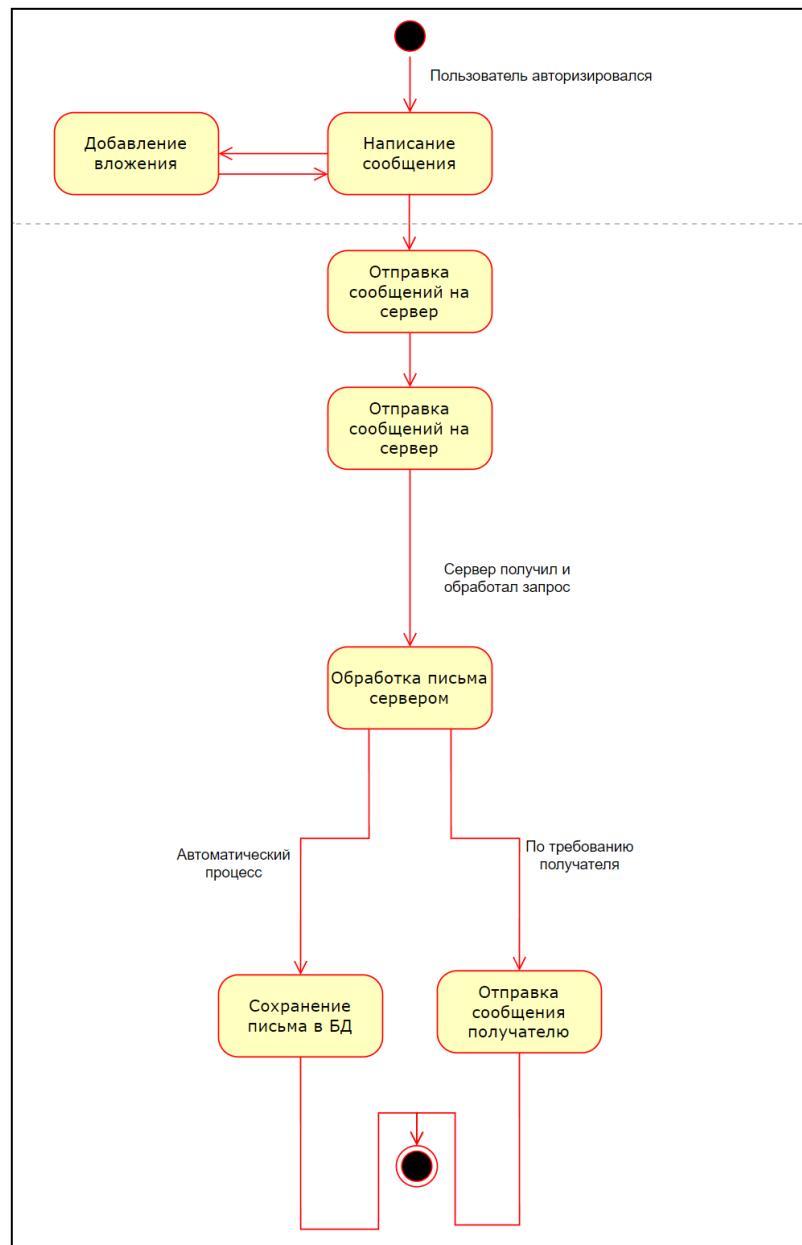


Рисунок 3.3 – Диаграмма состояний

Как и описывалось ранее, автор создаёт сообщение и отправляет сообщение серверу. Сервер отображает сообщение получателю на его странице чата.

Рассмотрим диаграмму последовательности (см. рис. 3.4), которая описывает взаимодействие объектов посредством приёма и передачи сообщений в определённой последовательности во времени.

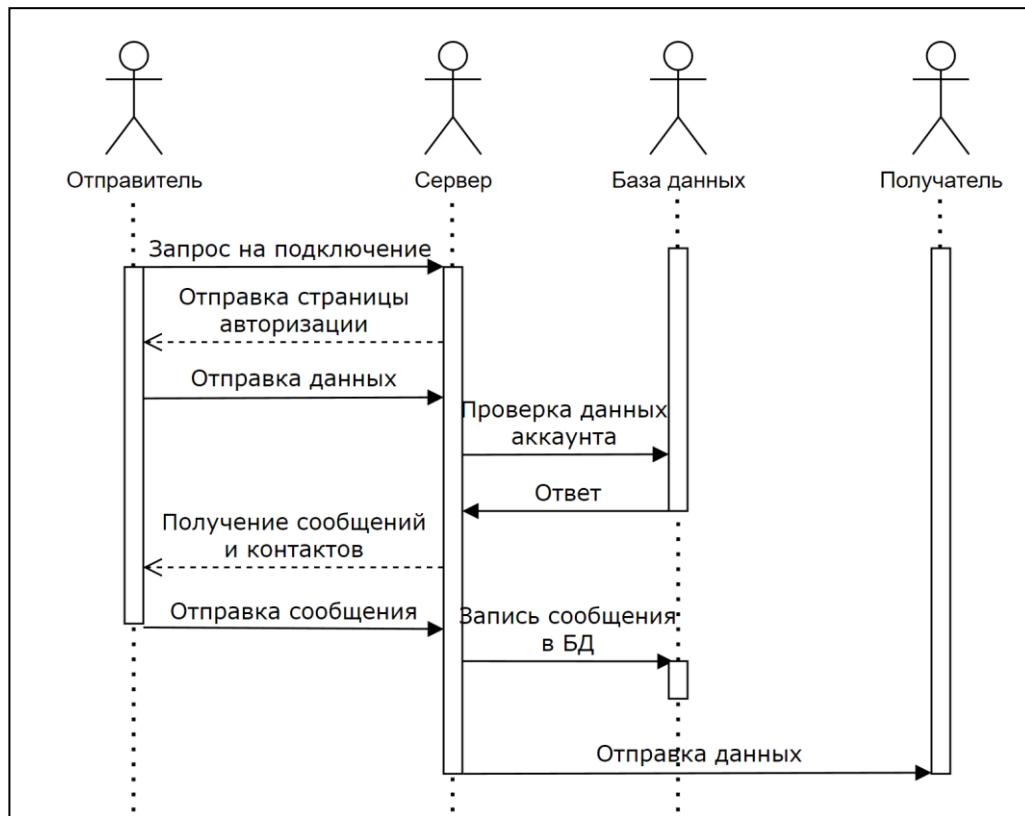


Рисунок 3.4 – Диаграмма последовательности

На данной диаграмме наглядно представлено, как разные участники ПО взаимодействуют друг с другом и чатом. Также показано, когда производится работа с базой данных.

Необходимость данной диаграммы заключается в следующем:

- визуализация взаимодействия: позволяет наглядно представить, как объекты взаимодействуют между собой в конкретном сценарии;
- анализ и проектирование: помогает разработчикам понять логику взаимодействия, выявить возможные проблемы и улучшить дизайн системы;

- документация: служит в качестве документации, объясняющей функционирование системы.

Диаграмма компонентов позволяет создать физическое отражение системы (см. рис. 3.5).

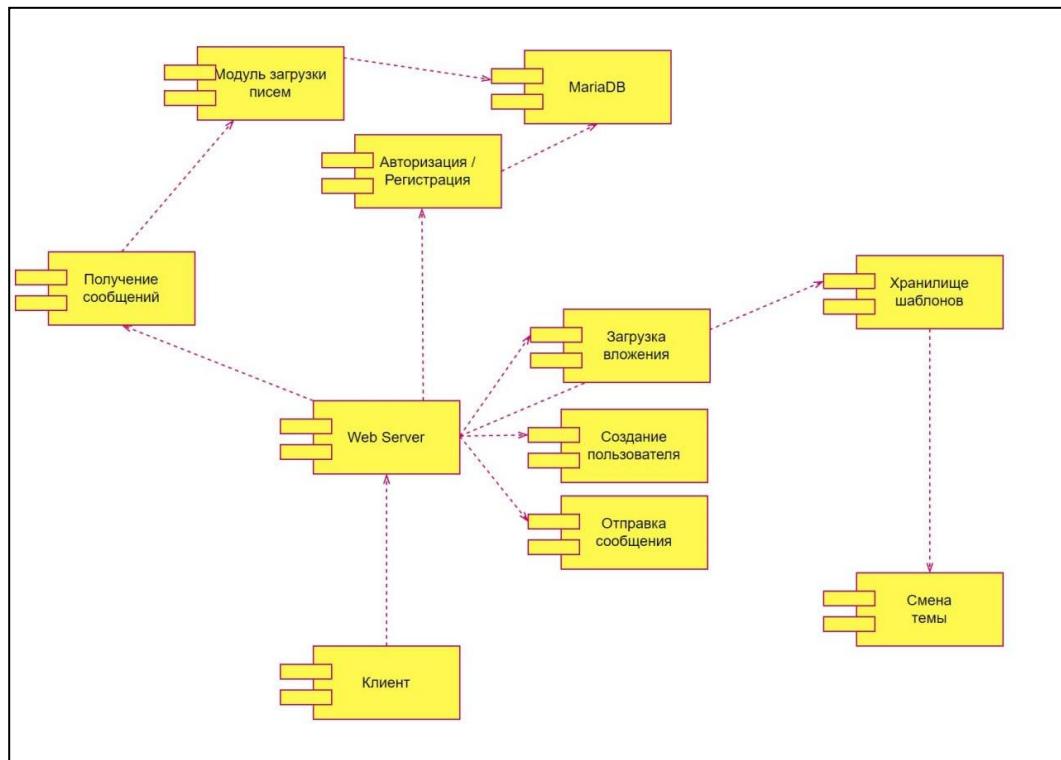


Рисунок 3.5 – Диаграмма компонентов

Продукт состоит из нескольких основных страниц: авторизация, регистрация, главная, чат. К большему числу этих файлов подключены автономные js скрипты для работы локально. Также на схеме видно взаимодействие с базой данных через «MariaDB».

На диаграмме не указан css-файл. Он универсален под каждую страницу.

Необходимость данной диаграммы заключается в следующем:

- моделирование структуры системы;
- анализ зависимостей;
- управление сложностью.

4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Основное назначение базы данных — управление пользовательским контентом и взаимодействиями между пользователями, такими как сообщения и избранные сообщения. Для чат-мессенджера была разработана база данных с учётом ограничений для обеспечения работоспособности логики сайта. Схема данных чата представлена на рисунке 4.1

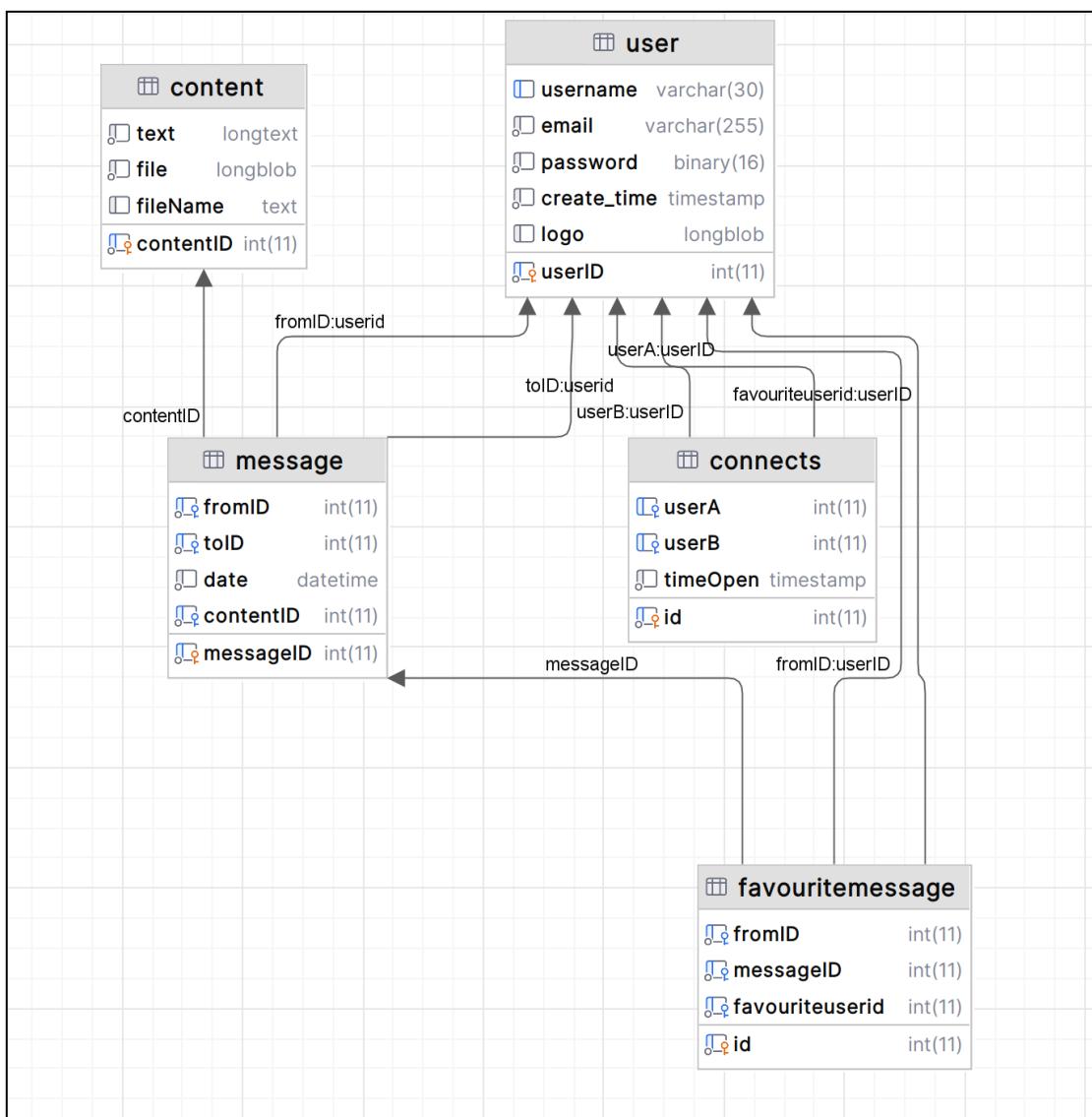


Рисунок 4.1 – Схема данных

Рассмотрим назначение каждой таблицы и ограничений более подробно.

Таблица «content» (см. таб. 4.1).

Таблица 4.1 – «content»

№	Name	Type	Key	Not null	Comment
1	contentId	int	Key	Not null	Идентификатор.
2	text	longtext		Not null	Текстовое содержание сообщения.
3	file	longblob		Not null	Содержимое файла в байтовой строке.
4	fileName	text		Null	Название файла.

Таблица хранит контент, загруженный пользователями, который может включать текст и файлы. Идентификатор на эту таблицу находится в таблице «message». Было предпринято решение вынести тяжёлые данные в эту таблицу, чтобы во время поиска сообщений было ниже. Для лучшей отзывчивости поля таблицы были проиндексированы и применено сжатие данных. Скрипт создания таблицы изображён на рисунке 4.2.

```
create table content
(
    contentID int auto_increment
        primary key,
    text      longtext not null,
    file     longblob not null,
    fileName  text      null
)
row_format = COMPRESSED;
```

Рисунок 4.2 – Таблица «content»

Таблица «user» (см.таб. 4.2)

Таблица 4.2 – «user»

№	Name	Type	Key	Not null	Comment
1	username	Varchar(30)		Not null	Уникальное имя пользователя.
2	email	Varchar(255)		Not null	Электронная почта пользователя.
3	password	Binary(16)		Not null	Хэш пароля.
4	create_time	Timestamp		Not null	Время регистрации.
5	userID	int	Key	Not null	Идентификатор.
6	logo	longblob		Null	Логотип пользователя в байтовом представлении.

Скрипт создания таблицы изображён на рисунке 4.2.

```
create table user
(
    username  varchar(30)          null,
    email     varchar(255)         not null,
    password  binary(16)          not null,
    create_time timestamp default CURRENT_TIMESTAMP not null,
    userID    int auto_increment primary key,
    logo      longblob           null,
    constraint user_pk
        unique (username),
    constraint user_username_uindex
        unique (username)
)
row_format = COMPRESSED;
```

Рисунок 4.2 – Таблица «user»

Таблица хранит информацию о всех пользователях мессенджера, включая их учётные данные и логотипы в байтовом представлении. Для лучшей отзывчивости поля таблицы были проиндексированы и применено сжатие данных.

Для обеспечения корректной работы логики базы данных был разработан триггер before_delete_user. Он удаляет сообщения, отправленные пользователем, перед удалением записи пользователя. Триггер необходим для поддержки целостности данных. Скрипт создания триггера изображён на рисунке 4.3.

```
create definer = mysql0`%` trigger before_delete_user
  before delete
  on user
  for each row
BEGIN
    delete from message where message.fromID=OLD.userID;
END;
```

Рисунок 4.3 – Триггер «before_delete_user»

Таблица «connects» (см. таб. 4.3).

Таблица 4.3 – «connects»

№	Name	Type	Key	Not null	Comment
1	userA	int		Null	Идентификатор пользователя.
2	userB	int		Null	Идентификатор пользователя.

Продолжение таблицы 4.3

3	id	int	Key	Not null	Уникальный идентификатор связи.
4	timeVisit	timestamp		Not null	Время последнего открытия диалога.

Таблица управления связями между пользователями. Эта таблица необходима для корректной работы последних диалогов пользователя. Для лучшей отзывчивости поля таблицы были проиндексированы и применено сжатие данных. Скрипт создания таблицы изображён на рисунке 4.4.

```

create table connects
(
    userA      int                      null,
    userB      int                      null,
    id         int auto_increment
        primary key,
    timeOpen timestamp default CURRENT_TIMESTAMP not null on update CURRENT_TIMESTAMP,
    constraint connects_pk
        unique (userA, userB),
    constraint connects_user(userID_fk
        foreign key (userA) references user (userID)
        on delete cascade,
    constraint connects_user(userID_fk2
        foreign key (userB) references user (userID)
        on delete cascade
)
row_format = COMPRESSED;

```

Рисунок 4.4 – Таблица «connects»

Таблица «message» (см. таб. 4.4).

Таблица 4.4 – «message»

№	Name	Type	Key	Not null	Comment
1	messageID	int	Key	Not null	Идентификатор сообщения

Продолжение таблицы 4.4

2	fromID	int		Not null	Идентификатор отправителя сообщения.
3	toID	int		Not null	Идентификатор получателя сообщения.
4	date	Timestamp		Not null	Дата и время отправки сообщения.
5	contentID	int		Not null	Идентификатор контента сообщения.

Скрипт создания таблицы изображён на рисунке 4.5.

```
create table message
(
    messageID int auto_increment
        primary key,
    fromID    int                      not null,
    toID      int                      not null,
    date      datetime default CURRENT_TIMESTAMP not null,
    contentID int                      not null,
    constraint content
        foreign key (contentID) references content (contentID)
            on update cascade on delete cascade,
    constraint `from`
        foreign key (fromID) references user (userid),
    constraint `to`
        foreign key (toID) references user (userid)
)
row_format = COMPRESSED;
```

Рисунок 4.5 – Таблица «messages»

Таблица хранит информацию о сообщениях. Эта таблица необходима для хранения сообщений пользователей. Весь контент сообщения хранится в таблице «content». Для лучшей отзывчивости поля таблицы были проиндексированы и применено сжатие данных.

Для обеспечения корректной работы логики базы данных был разработан триггер before_delete_message: удаляет запись из таблицы content перед удалением сообщения, чтобы избежать "висячих" данных. Он удаляет контент, отправленный пользователем, перед удалением упоминания сообщения. Скрипт создания триггера изображён на рисунке 4.6.

```
create definer = mysql@`%` trigger before_delete_message
before delete
on message
for each row
BEGIN
    delete from content where content.contentID=OLD.contentID;
END;
```

Рисунок 4.6 – Триггер «before_delete_message»

Таблица «favouritemessage» (см. таб. 4.5).

Таблица 4.5 – «favouritemessage»

№	Name	Type	Key	Not null	Comment
1	id	int	Key	Null	Идентификатор сообщения
2	fromID	int		Not null	Идентификатор отправителя сообщения.

Продолжение таблицы 4.5

3	messageID	int		Not null	Идентификатор сообщения.
4	favouriteus erid	int		Not null	Идентификатор пользователя.

Таблица хранит информацию об избранных сообщениях. Эта таблица необходима для хранения избранных сообщений пользователей. Представляет собой связь между пользователем и сообщением, которое он посчитал важным для него. Для лучшей отзывчивости поля таблицы были проиндексированы и применено сжатие данных. Скрипт создания таблицы изображён на рисунке 4.7.

```

create table favouritemessage
(
    id          int auto_increment
    primary key,
    fromID      int not null,
    messageID   int not null,
    favouriteuserid int not null,
    constraint favouriteMessage_message_messageID_fk
        foreign key (messageID) references message (messageID)
            on update cascade on delete cascade,
    constraint favouriteMessage_user(userID_fk
        foreign key (fromID) references user (userID),
    constraint favouriteMessage_user(userID_fk1
        foreign key (favouriteuserid) references user (userID)
);

```

Рисунок 4.7 – Таблица «favouritemessage»

Эта структура базы данных позволяет управлять пользователями, их контентом, взаимодействиями между ними и избранными сообщениями. База данных обеспечивает целостность и поддерживает каскадное обновление и удаление данных для поддержания согласованности.

5 ПРОЕКТИРОВАНИЕ ОБЪЕКТНЫХ МОДУЛЕЙ

Для разработки полноценного мессенджера в веб окружении требуется слаженная работа серверной и клиентской частей. Для этого необходимо разработать устойчивые модели данных.

Рассмотрим процесс авторизации. Для авторизации пользователь заполняет форму и отправляет её на сервер. Сервер опрашивает базу данных на предмет наличия такого пользователя. Если такого пользователя не было зарегистрировано то браузер получает cookie настройку с токеном доступа(в токене зашифрован никнейм пользователя) (см. рис. 5.1).

```
// structure of JWT claim(contains username)
type JwtCustomClaims struct { 2 usages
    Name string `json:"name"`
    jwt.RegisteredClaims
}
```

Рисунок 5.1 – Структура токена JWT

В дальнейшем сервер вместе с запросами получает и этот JWT токен. Так происходит аутентификация пользователя в системе.

Рассмотрим процесс загрузки сообщений. Предусмотрено две структуры сообщений (избранные и обычные). Для загрузки сообщений при открытии страницы чата браузер посылает запрос серверу. Сервер проверяет корректность токена доступа и делает запрос в базу данных.

С использованием запроса "select" база данных возвращает список сообщений в зависимости от поставленной цели. Полученные сообщения из базы данных записываются в структуру типа message(см рис где она там) или favouritemessage(см. Рис. Где она там). Это позволяет серверной части лучше понять запрос пользователя. После этого сообщения подставляются в HTML шаблон и возвращаются пользователю.

После получения ответа данные добавляются в структуру (см. рис. 5.2).

```
// struct of Message
type Message struct { 17 usages
    Text      string
    File     []byte
    FileName string
    Date     time.Time
    DateStr   string
    Pos      string
    MessageID int
    Favourite bool
}
```

Рисунок 5.2 – Структура сообщения

В структуре сообщения предусмотрены поля для текста сообщения, содержимого файла, названия файла, даты в формате time.Time , строкового представления даты отправки сообщения, позиции сообщения (входящие сообщения отображаются слева а исходящие справа), идентификатор сообщения, индикатор избранного сообщения.

Все эти данные подставляются в шаблон HTML (см. рис. 5.3).

```
{{range $key, $value := .}}
{{if $value.Favourite}}
{{if eq $value.Pos "r"}}
<div class="messageRight">
    <a>Ваше сообщение</a>
    {{ $value.Text }}
    <span>{{ $value.DateStr }}</span>
    <span style="cursor:pointer;" onclick="downloadBlob({{ $value.File }}, '{{ $value.FileName }}');">
        {{ $value.FileName }}</span>
    <div class="messageBottom">
        <a href="/chat?deletefavourite={{ $value.MessageID }}"></a>
    </div>
</div>
{{else}}
<div class="messageLeft">
    <a href="/chat?recipient={{ $value.UserName }}">{{ $value.UserName }}</a>
    {{ $value.Text }}
    <span>{{ $value.DateStr }}</span>
    <span style="cursor:pointer;" onclick="downloadBlob({{ $value.File }}, '{{ $value.FileName }}');">
        {{ $value.FileName }}</span>
    <div class="messageBottom">
        <a href="/chat?deletefavourite={{ $value.MessageID }}"></a>
    </div>
</div>
{{end}}
```

Рисунок 5.3 – Шаблон HTML для отображения исходящих сообщений

Для отображения избранных сообщений выполняется запрос на получение (нажатие на кнопку «Избранные»). Сервер обрабатывает запрос и подставляет данные из структуры (см. рис. 5.4).

```
// struct of favourite Message
type FavouriteMessages struct {
    Text      string
    File     []byte
    FileName string
    Date     time.Time
    DateStr  string
    Pos      string
    FromID   int
    MessageID int
    UserName  string
    Favourite bool
}
```

Рисунок 5.4 – Структура избранного сообщения

Как видно на рисунке, структура избранного сообщения отличается только наличием поля идентификатора отправителя.

Для автоматической загрузки сообщений используется AJAX (см. рис. 5.5).

```
function refresh(): void { 1 usage
$.ajax({
    url: "http://localhost:1323/refresh",
    method: "POST",
    data: {},
    success: function(data) : void {
        if (data != null){
            //If container contains light messages then new message must be in light
            let theme
            if ($("#.messages").find('.messageRight').length > 0) {
                theme = "";
            }else{
                theme = "dark";
            }
            //for every message
            for (let i:number = 0; i < data.length; i++) {
                if (data[i]["Pos"] == "r") {
                    //add LIGHT/DARK outgoing message to html
                    $(`<div class="messageRight"+theme+>` + data[i]["Text"] + '<span>' + data[i]["DateStr"] +
                    '</span><span style="cursor:pointer;" onclick="downloadBlob(' + data[i]["File"] + ', \'' + data[i]["FileName"] + '\')";>' + data[i]["FileName"] +
                    '</span><div class="messageBottom"><a href="/chat?delete=' + data[i]["ID"] +
                    '"></a><a href="/chat?favourite=' + data[i]["ID"] +
                    '"></a></div></div>').appendTo('.messages');

                }else{
                    //add LIGHT/DARK incoming message to html
                    $(`<div class="messageLeft"+theme+>` + data[i]["Text"] + '<span>' + data[i]["DateStr"] +
                    '</span><span style="cursor:pointer;" onclick="downloadBlob(' + data[i]["File"] + ', \'' + data[i]["FileName"] + '\')";>' + data[i]["FileName"] +
                    '</span><div class="messageBottom"><a href="/chat?delete=' + data[i]["ID"] +
                    '"></a><a href="/chat?favourite=' + data[i]["ID"] +
                    '"></a></div></div>').appendTo('.messages');
                }
            }
        },
    });
}
```

Рисунок 5.5 – AJAX алгоритм по добавлению новых сообщений

Каждые несколько секунд происходит фоновый запрос на сервер. В ответе браузер получает структуру сообщения в формате JSON. Новые сообщения уже отсортированы по дате отправки.

Чтобы браузер не получил уже полученные сообщения разработан алгоритм сохранения и обновления времени последнего отправленного сообщения (см. рис. 5.6). В cookie сохраняется самое позднее время из всех загруженных с базы данных сообщений.

```
lastMessageCookie := &http.Cookie{  
    Name: "LastMessage",  
    Value: newDateForCookie.Format("2006-01-02 15:04:05"),  
    SameSite: 3,  
    HttpOnly: true,  
    Secure: true,  
}  
c.SetCookie(lastMessageCookie)
```

Рисунок 5.6 – Сохранение времени последнего отправленного сообщения в cookie

Чтобы браузер получал только актуальные сообщения был разработан алгоритм сохранения и обновления времени последнего отправленного сообщения. Данного подхода позволяет хранить информацию о сообщениях только в рамках одной сессии, что не создаёт лишнюю нагрузку на сервер если клиент покинет сайт. В cookie сохраняется самое позднее время из всех, выгруженных с базы данных сообщений.

Программный код сохранения времени в cookie изображен на рисунке 5.6

6 ОПИСАНИЕ ПРОГРАММЫ

6.1 Алгоритмы

Программное обеспечение предполагает наличие сложных разветвлений алгоритмов для обеспечения работы мессенджера в полной мере. Рассмотрим основные алгоритмы внутри функций.

Функция сортировки дат. После преобразования всех сообщений из базы данных в структуры происходит сортировка сообщений по дате(см. рис. 6.1)

```
sort.Slice(favouriteArray, func(i, j int) bool { return favouriteArray[i].Date.Before(favouriteArray[j].Date) })
```

Рисунок 6.1 – Функция сортировки дат

В функции сортировке используется функция из пакета «sort». Встроенные функции языка Golang позволяют оптимально отсортировать слайсы и другие типы данных. На вход функция Slice получает interface {} и функцию сортировки(по какому признаку сортировать слайс. В данном случае сравниваются две даты и возвращается логическое значение (больше/меньше).

Функция сжатия изображения. Для уменьшения размера базы данных данные в неё помимо сжатия средствами MariaDB также сжимаются средствами Golang.

Для сжатия изображения используется Resize метод объекта image.
Изображение считывания из полученного файла побайтово.

На рисунке 6.2 представлен алгоритм сжатия изображений логотипов пользователей.

```
logo, err := c.FormFile("name: \"logo\"")
if err != nil {
    return err
}
src, err := logo.Open()
if err != nil {
    return err
}
defer src.Close()
var dst []byte
img, _ := jpeg.Decode(src)

compressedImage := imaging.Resize(img, width: 100, height: 100, imaging.Lanczos)
buf := new(bytes.Buffer)
if err != nil {
    return err
}
err = jpeg.Encode(buf, compressedImage, o: nil)
dst = buf.Bytes()
if err != nil {
    return err
}
```

Рисунок 6.2 – Функция сжатия изображения

Файл с логотипом извлекается из формы и читается побайтово. Создаётся объект изображения. Изображение масштабируется до размеров 100x100пикселей с использованием сглаживания методом Ланцоша. Такой подход позволяет с минимальными потерями качества самого изображения уменьшить разрешение. Такой алгоритм позволяет сжать размер изображение в размерах более чем в 1000 раз, что явно положительно скажется на скорости работы запросов к базе данных и снизит нагрузку на сервер.

Алгоритм хэширования пароля. Для обеспечения простого и безопасного способа хранения паролей в базе данных было принято решение использовать алгоритм MD5 для хэширования пароля. Преимуществом такого подхода является безопасность при передаче пароля через интернет и фиксированный размер хэш. Функция хэширования на рисунке 6.3

```
hashPassword := md5.Sum([]byte(password + variables.AdditionalString))
```

Рисунок 6.2 – Функция хэширования пароля

Для дополнительной безопасности к паролю пользователя добавляется секретная фраза-соль, что значительно усложняет подбор пароля методом Bruteforce. Во время авторизации при расшифровке пароля эта строка также добавляется к паролю пользователя.

Злоумышленники зачастую используют доступ напрямую к серверу без всех составляющих. И подобрать пароль+фразу-соль будет сложно из-за того, что эта фраза является сложной к подбору и содержит символы разных языков и регистров.

Алгоритм кэширования. Для ускорения доступа к серверу данные в нем кэшируются средствами фреймворка ECHO. Функция включения кэширования изображена на рисунке 6.3.

```
e.AutoTLSManager.Cache = autocert.DirCache("/var/www/.cache")
```

Рисунок 6.3 – Функция кэширования данных

Алгоритм валидации токена JWT. Валидация происходит путем расшифровки токена секретной фразой.

Функция валидации токена изображена на рисунке 6.4.

```
// func for validate JWT token
func ValidateToken(c echo.Context) string { 7 usages
    //retrieve cookie value
    cookie, _ := c.Cookie( name: "JWTToken")
    if cookie == nil : "Token error" ↴
    //try parse token
    token, _ := jwt.Parse(cookie.Value, func(token *jwt.Token) (interface{}, error) {
        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
            return nil, fmt.Errorf( format: "There was an error in parsing")
        }
        return variables.Secret, nil
    })
    //if token not valid
    if token == nil {
        return "Token error"
    }
    //try parse claim from token
    claims, ok := token.Claims.(jwt.MapClaims)
    //if token claims not valid
    if !ok {
        return "Token error"
    }
    //return username string from claim
    return claims["name"].(string)
}
```

Рисунок 6.4 – Функция валидации токена авторизации

После этого происходит извлечение из него структуры данных (см. рис. 5.1).

Если в этих процессах происходит ошибка то возвращается строка с ошибкой. Если всё прошло успешно то функция возвращает никнейм пользователя.

6.2 Описание функций

Рассмотрим функцию работы страницы чата на рисунке 6.5.

```
func ChatGET(c echo.Context) error { 1 usage
    logoutParam := c.QueryParam( name: "logout")
    if logoutParam != "" {
        return LogoutFunc(c)
    }
    var theme = ""
    toggleParam := c.QueryParam( name: "toggle")
    if toggleParam == "true" {
        theme = additional.Toggle(c)
    }
    if theme == "" {
        theme = additional.CheckTheme(c)
    }
    username := additional.ValidateToken(c)
    if username != "Token error" {
        usernameID := additional.CheckUser(username)
        if usernameID == "" : c.Redirect(http.StatusMovedPermanently, "/") ↴

        //search recipient MessageID from Cookie or GET request
        recipientID := additional.SearchRecipient(c)
        //if user press on favourite messages
        favouriteChatsParam := c.QueryParam( name: "favouritechats")
        if favouriteChatsParam != "" {
            FavouritePage(c)
            recipientID = ""
        }
        //if user press on delete message button
        deleteFavouriteID := c.QueryParam( name: "deletefavourite")
        if deleteFavouriteID != "" {
            additional.DeleteFavouriteMessage(usernameID, deleteFavouriteID)
        }
        deleteID := c.QueryParam( name: "delete")
        if deleteID != "" {
            additional.DeleteMessage(usernameID, deleteID)
        }
    }
}
```

Рисунок 6.5 – Функция GET обработчика сайта (часть 1)

Проверка GET запроса на наличие поля «logout». Если оно присутствует то вызывается функция по выходу из аккаунта. Алгоритм предполагает удаление токена доступа из cookie.

Проверка GET запроса на наличие поля «toggle». Если оно присутствует то алгоритм меняет текущую тему на другую (тёмную/светлую).

Проверка cookie пользователя с целью определения цветовой темы сайта (она записывается в cookie автоматически при первом входе на сайт).

Проверка токена авторизации (если он не действителен то происходит перенаправление на страницу авторизации).

Поиск собеседника в cookie (автоматически добавляется при переходе на страницу диалога).

Проверка GET запроса на наличие поля «favouritechats». Если поле присутствует значит пользователь желает перейти на страницу избранное. Алгоритм удаляет из cookie идентификатор текущего диалога.

Проверка GET запроса на наличие поля «deletefavourite». Если поле присутствует значит пользователь хочет удалить сообщение. Функция удаления представлен на рисунке 6.6.

```
func DeleteFavouriteMessage(usernameID string, deleteID string) { 1 usage
    _, err := variables.Db.Exec(query: "delete from favouritemessage where id = ? and favouriteuserid = ?",
        deleteID, usernameID)
    if err != nil {
        fmt.Println(err)
    }
}
```

Рисунок 6.6 – Функция удаления избранного сообщения

Проверка GET запроса на наличие поля «delete». Алгоритм проверяет принадлежность пользователя к этому сообщению и если пользователь действительно получал или отправлял это сообщение то сообщение будет удалено из базы данных. Функция удаления сообщения представлена на рисунке 6.7.

```
func DeleteMessage(usernameID string, deleteID string) { 1 usage
    _, err := variables.Db.Exec(query: "delete from message where messageId = ? and " +
        "(message.fromID = ? or message.toID = ?)", deleteID, usernameID, usernameID)
    if err != nil {
        fmt.Println(err)
    }
}
```

Рисунок 6.7 – Функция удаления сообщения

Вторая часть функции работы обработчика GET запросов представлена на рисунке 6.8.

```

//if user press on favourite message button
favouriteID := c.QueryParam("name: "favourite")
if favouriteID != "" {
    additional.FavouriteMessage(usernameID, favouriteID)
}

if recipientID != "" && recipientID != usernameID {
    messages := additional.ParseMessages(c, usernameID, recipientID)
    if theme == "dark" {
        return c.Render(http.StatusOK, name: "chatDARK.html", messages)
    } else {
        return c.Render(http.StatusOK, name: "chat.html", messages)
    }
} else {
    favouriteMessagesArray := additional.DownloadFavourite(c, usernameID)
    if theme == "dark" {
        return c.Render(http.StatusOK, name: "chatDARK.html", favouriteMessagesArray)
    } else {
        return c.Render(http.StatusOK, name: "chat.html", favouriteMessagesArray)
    }
}

```

Рисунок 6.8 – Функция GET обработчика сайта (часть 2)

Модуль добавления сообщения в избранный список. Если пользователь нажал на кнопку добавления в избранные сообщения то сервер получит GET запрос который обрабатывается и идентификатор добавится в базу данных.

Проверка на наличие собеседника в диалоге (если его нет значит пользователь желает попасть на страницу с избранными сообщениями).

Если собеседника нет то выполняется загрузка избранных сообщений из базы данных и отправка шаблона с нужной цветовой темой пользователю.

Алгоритм загрузки избранных сообщений представлен на рисунке 6.9

```

func DownloadFavourite(c echo.Context, favouriteID string) interface{} { 1 usage
    var favouriteArray []variables.FavouriteMessages
    result, err := variables.Db.Query( query: "select favouritemessage.id,username,favouritemessage.fromID,text,"+
        "cast(date as char),file,fileName from favouritemessage join message on "+
        "favouritemessage.messageID = message.messageID join chat.content on message.contentID = "+
        "content.contentID join user on favouritemessage.fromID = user.userID where favouritemessage.favouriteuserid "+
        "= ? and favouritemessage.favouriteuserid != favouritemessage.fromID", favouriteID)
    if err != nil {
        fmt.Println(err)
    }
    for result.Next() {
        var username, text, dateStr, fileName string
        var byteFile []byte
        var fromid, id int
        err = result.Scan(&id, &username, &fromid, &text, &dateStr, &byteFile, &fileName)
        date, err := time.Parse(variables.TimeFormat, dateStr)
        if err != nil {
            fmt.Println(err)
        }
        var elem variables.FavouriteMessages
        if bytes.Equal(byteFile, []byte{48}) == true {
            elem = variables.FavouriteMessages{MessageID: id, Favourite: true, UserName: username, FromID: fromid,
                Text: text, File: byteFile, FileName: "", Date: date,
                DateStr: date.Format( layout: "2006-01-02 15:04:05"), Pos: "l"}
        } else {
            elem = variables.FavouriteMessages{MessageID: id, Favourite: true, UserName: username, FromID: fromid,
                Text: text, File: byteFile, FileName: fileName, Date: date,
                DateStr: date.Format( layout: "2006-01-02 15:04:05"), Pos: "r"}
        }
        favouriteArray = append(favouriteArray, elem)
    }
}

```

Рисунок 6.9 – Функция получения избранных сообщения

В функцию получения избранных сообщений выполняется запрос в базу данных на получение сообщений. После выполняется чтение каждого сообщения отдельно и добавление нового сообщения в слайс favouriteArray. Предусмотрен случай когда пользователь не прикреплял файл. Его сообщение не имеет имени файла и содержимого в структуре. На рисунке 6.4 представлен только алгоритм получения исходящих сообщений. Функция получения входящих сообщений аналогичен. Отличия присутствуют в запросе к базе данных и полем «Pos» в структуре сообщения (l - входящие, r - исходящие).

Получение сообщений из диалогов происходит подобным образом за исключением другого запроса SQL.

Рассмотрим функцию определения собеседника диалога на рисунке 6.10.

```
func SearchRecipient(c echo.Context) string { 3 usages
    var recipientID string
    recipient := c.QueryParam("name: "recipient")
    recipientID = CheckUser(recipient)
    if recipientID != "" {
        newRecipientCookie := &http.Cookie{}
        newRecipientCookie.Name = "recipient"
        newRecipientCookie.Value = recipientID
        newRecipientCookie.SameSite = 3
        newRecipientCookie.HttpOnly = true
        newRecipientCookie.Secure = true
        c.SetCookie(newRecipientCookie)
    } else {
        var recipientCookie, err = c.Cookie("name: "recipient")
        if err != nil {
            fmt.Println(err)
        }
        if recipientCookie != nil {
            recipientID = recipientCookie.Value
        }
    }
    return recipientID
}
```

Рисунок 6.10 – Функцию определения идентификатора собеседника

В функции происходит поиск параметра собеседника в GET запросе. Если собеседника нет(не было совершено перехода на страницу с диалогом пользователя) то данных извлекаются из cookie и возвращаются в функцию chatGET с рисунка 6.1. Если собеседника нет и в cookie данных то возвращается пустая строка идентификатора.

Рассмотрим функцию смены цветовой темы сайта на рисунке 6.11.

```
// function for Toggle theme value on cookie
func Toggle(c echo.Context) string { 3 usages
    //make cookie
    theme, err := c.Cookie("theme")
    Cookie := &http.Cookie{}
    Cookie.Name = "theme"
    Cookie.SameSite = 3
    Cookie.HttpOnly = true
    Cookie.Secure = true
    //if theme cookie not found
    if err != nil {
        Cookie.Value = "light"
        c.SetCookie(Cookie)
        return Cookie.Value
    }
    //if last theme is dark
    if theme.Value == "dark" {
        Cookie.Value = "light"
    }
    //if last theme is light
    if theme.Value == "light" {
        Cookie.Value = "dark"
    }
    //set cookie
    c.SetCookie(Cookie)
    //return current theme
    return Cookie.Value
}
```

Рисунок 6.11 – Функция изменения цветовой темы сайта

В функции происходит извлечение cookie с именем «theme» и создается новый cookie с противоположным значением. Если cookie не найдено то задаётся светлая тема сайта по умолчанию.

Рассмотрим алгоритм поиска пользователя в базе данных на рисунке 6.12.

```
// func check if exist username in DB
func CheckUser(username string) string { 5 usages
    var ID string
    //query for DB
    result, _ := variables.Db.Query(query: "select userID from user where username = ?", username)
    if result == nil {
        //if username not found
        return ""
    }
    for result.Next() {
        //scan userID
        err := result.Scan(&ID)
        if err != nil {
            fmt.Println(err)
        }
    }
    //return username MessageID
    return ID
}
```

Рисунок 6.12 – Функция поиска пользователя в базе данных

Происходит запрос в базу данных для поиска идентификатора пользователя по нику. Возвращаемое значение является идентификатором пользователя. Он необходим для работы остальных запросов SQL связанных с мессенджером.

6.3 Описание программных модулей

Для удобства разработки и читабельности кода весь проект был разбит на связанные пакеты.

Серверная часть проекта изображена на рисунке 6.13.

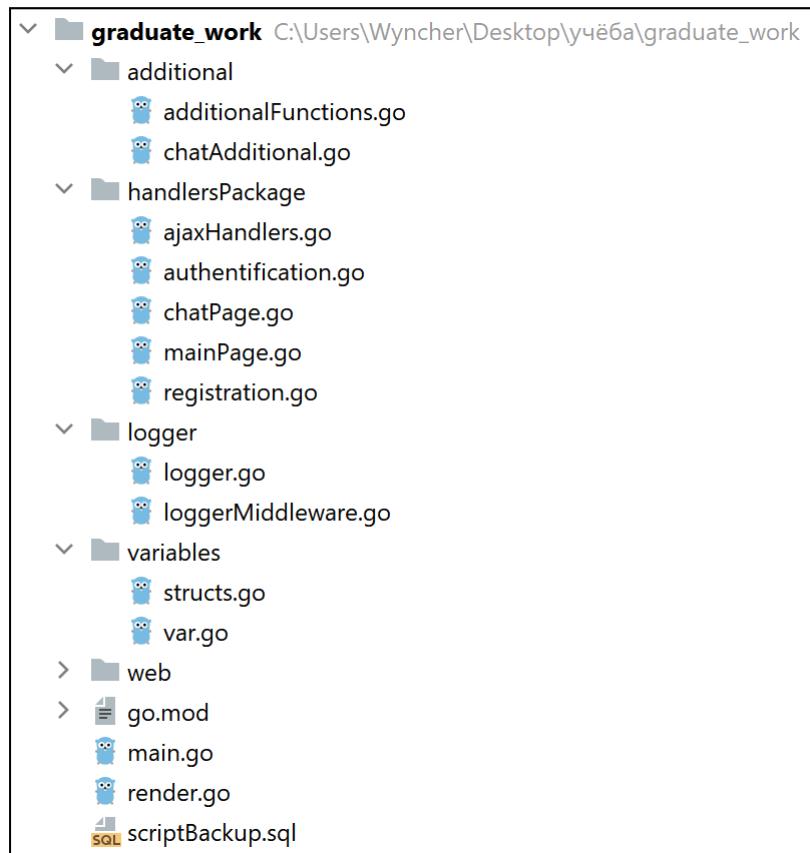


Рисунок 6.13 – Файловая структура серверной части мессенджера

Назначение файлов указано в таблице 6.1

Таблица 6.1 – Файловая структура серверной части

Название файла	Назначение файла	Пакет
additionalFunctions.go	Дополнительные функции для обеспечения работы всего сайта в целом	additional

Продолжение таблицы 6.1

chatAdditon	Дополнительные	additional
-------------	----------------	------------

al.go	функции для обеспечения работы страницы чата	
ajaxHandler.go	Функции обеспечивает ответы на запросы AJAX от клиента.	handlersPackage
authentication.	Функции обеспечения работы страницы авторизации.	handlersPackage
chatPage.go	Функции обеспечения работы страницы чата.	handlersPackage
mainPage.go	Функции обеспечения работы главной страницы авторизации.	handlersPackage
registration.go	Функции обеспечения работы страницы регистрации.	handlersPackage
logger.go	Общая структура логгера событий.	logger
loggerMiddleware.go	Реализация отображения событий в консоли сервера.	logger
structs.go	Файл для всех структур проекта.	variables
var.go	Важные глобальные переменные сервера.	variables
go.mod	Метаданные модуля	

Для обеспечения работы клиентской части присутствуют файлы шаблонов и стилей для отображения готовых страниц у пользователя в

браузере. Рассмотрим файловую структуру серверной части мессенджера на рисунке 6.14

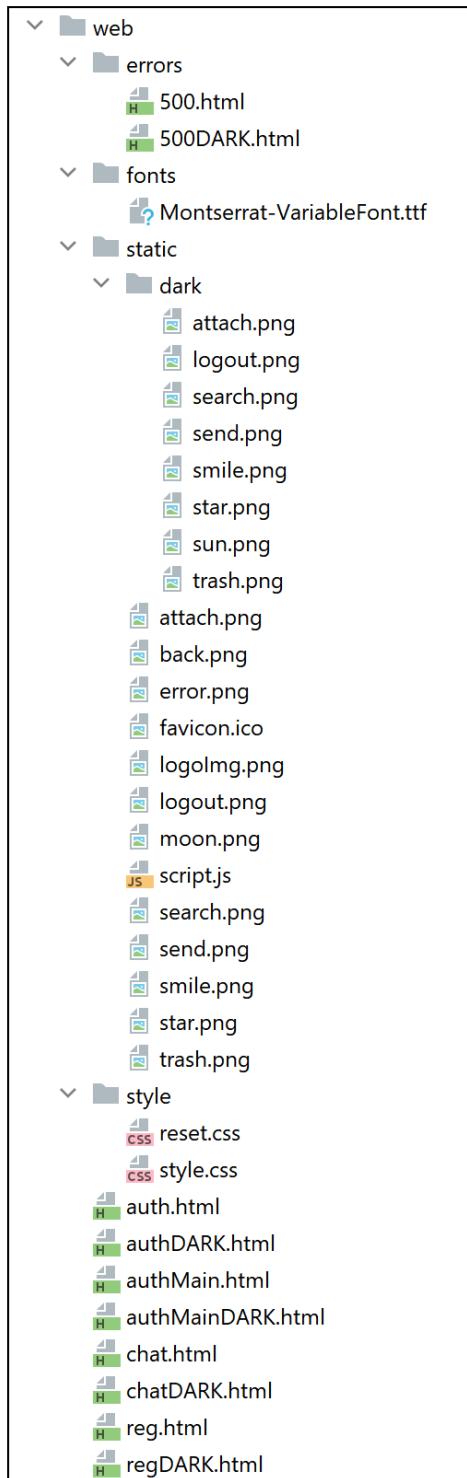


Рисунок 6.14 – Файловая структура клиентской части мессенджера
Назначение файлов указано в таблице 6.2

Таблица 6.2 – Файловая структура серверной части

Название файла	Назначение файла	Директория
500.html	Страница ошибки.	errors
500DARK.html	Страница ошибки.	errors
Montserrat-VariableFont.ttf	Шрифт сайта.	fonts
dark/attach.png	Кнопка вложения (тёмная тема)	static
dark/logout.png	Кнопка выхода (тёмная тема).	static
dark/search.png	Кнопка поиска пользователя (тёмная тема).	static
dark/smile.png	Кнопка открытия панели смайликов (тёмная тема).	static
dark/star.png	Кнопка «избранное» (тёмная тема).	static
dark/trash.png	Кнопка «удаление» (тёмная тема).	static
attach.png	Кнопка вложения (светлая тема)	static
back.png	Кнопка назад (светлая тема)	static

Продолжение таблицы 6.2

favicon.ico	Favicon сайта	static
error.png	Изображение на странице ошибки.	static
logoImg.png	Логотип сайта в шапке.	static
logout.png	Кнопка выхода (светлая тема).	static
moon.png	Кнопка смены темы на тёмную (светлая тема).	static
script.js	Файл JavaScript для сайта.	static
search.png	Кнопка поиска пользователя (светлая тема).	static
send.png	Кнопка отправки сообщения (светлая тема).	static
smile.png	Кнопка открытия панели смайликов (светлая тема).	static
star.png	Кнопка «избранное» (светлая тема).	static
trash.png	Кнопка «удаление» (светлая тема).	static
reset.css	Сброс встроенных стилей CSS.	style
style.css	Файл стилей CSS.	style

Продолжение таблицы 6.2

auth.html	Шаблон страницы авторизации (светлая тема).	/
authDARK.html	Шаблон страницы авторизации (тёмная тема).	/
authMain.html	Шаблон главной страницы авторизации (светлая тема).	/
authMainDARK.html	Шаблон главной страницы авторизации (тёмная тема).	/
chat.html	Шаблон страницы диалога (светлая тема).	/
chatDARK.html	Шаблон страницы диалога (тёмная тема).	/
reg.html	Шаблон страницы регистрации(светлая тема).	/
regDARK.html	Шаблон страницы регистрации(тёмная тема).	/

Структура клиентской части сайта была разработана с целью предоставления пользователям интуитивно понятного и удобного интерфейса. Все элементы навигации, страницы и функциональные возможности тщательно продуманы, для обеспечения максимально положительного пользовательского опыта.

7 ОПИСАНИЕ ИНТЕРФЕЙСА

Страница «Главная» (см. рис. 7.2) открывается при переходе на любую страницу неавторизированным пользователем.

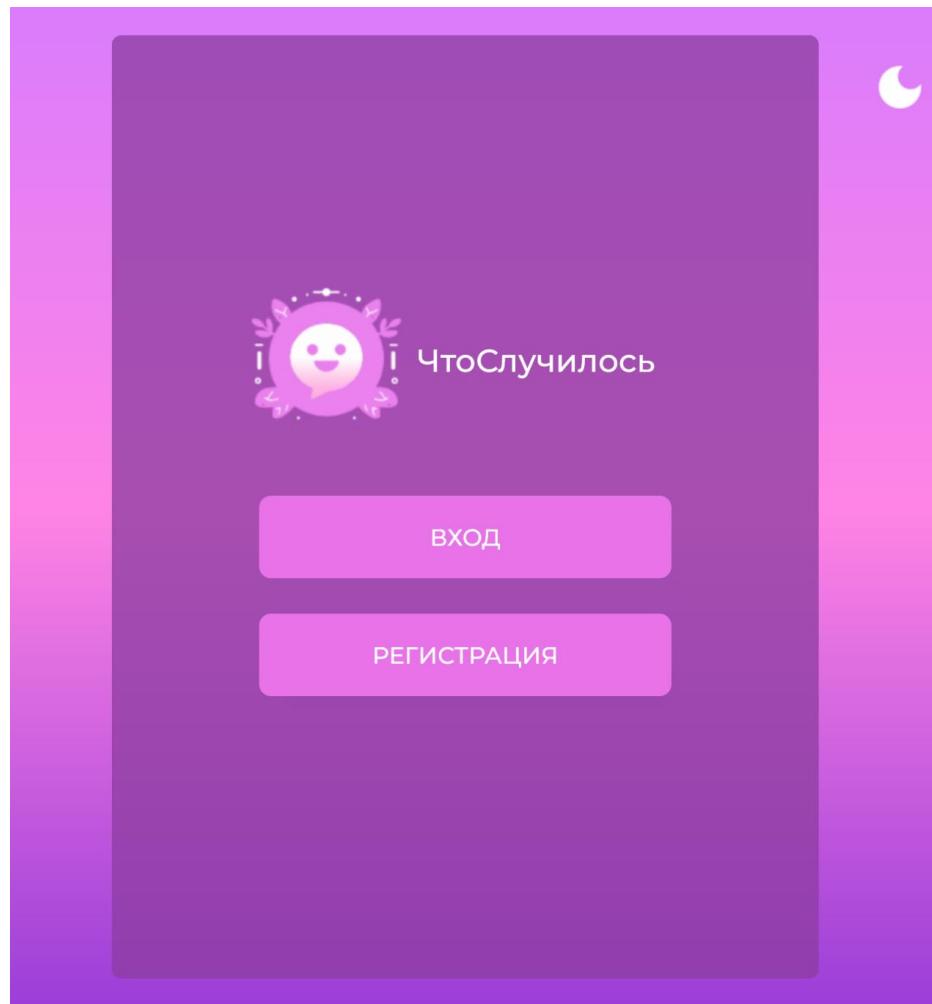


Рисунок 7.1 – Экранная форма страницы «Главная»

Элементы интерфейса:

- логотип;
- меню:
 - 1) «ВХОД»: переход на страницу авторизации для зарегистрированных пользователей;

2) «РЕГИСТРАЦИЯ»: переход на страницу регистрации для новых пользователей;

- луна/солнце : кнопка смены цветовой темы.

Страница «Авторизация» (см. рис.7.2) открывается по нажатию на кнопку «ВХОД» на странице «Главная».

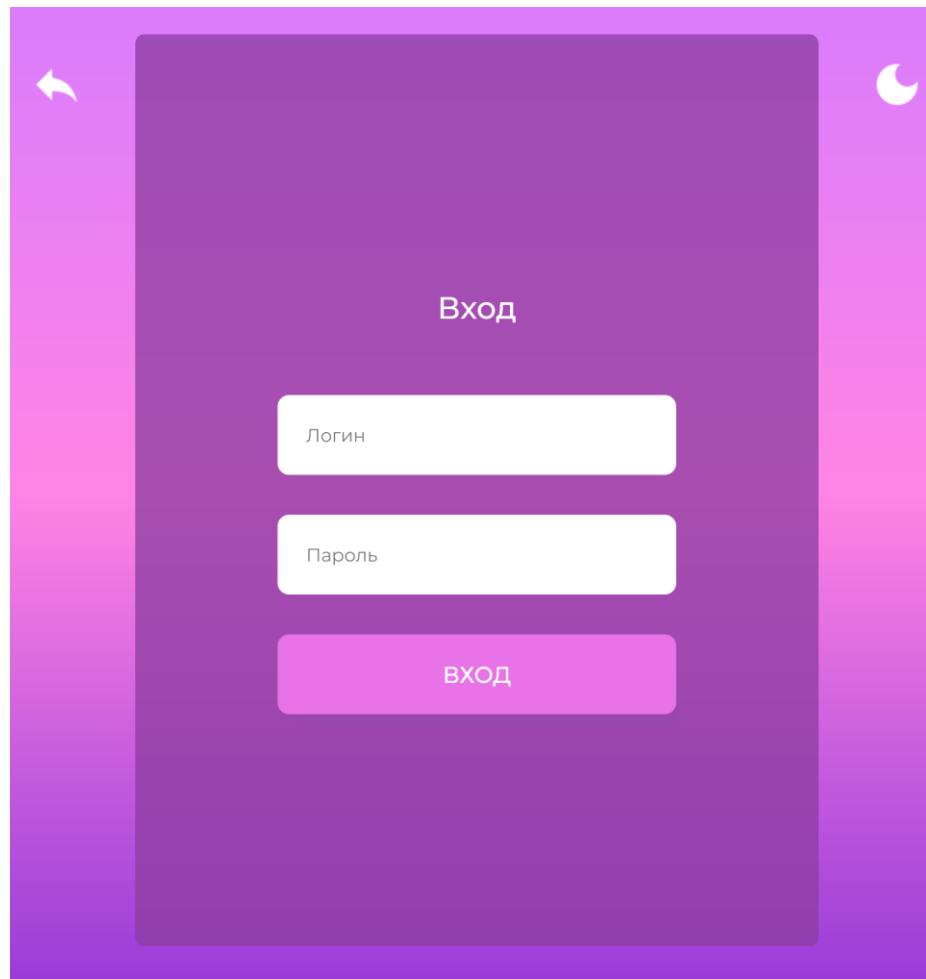


Рисунок 7.2 – Экранная форма страницы «Авторизация»

Элементы интерфейса:

- заголовок;
- форма авторизации:
 - 1) «Логин»: поле для ввода логина пользователя;
 - 2) «Пароль»: поле для ввода пароля пользователя;
 - 3) «ВХОД»: кнопка отправки формы авторизации.

- луна/солнце : кнопка смены цветовой темы;
- кнопка возврата на страницу «Главная».

Страница «Регистрация» (см. рис.7.3) открывается по нажатию на кнопку «РЕГИСТРАЦИЯ» на странице «Главная».

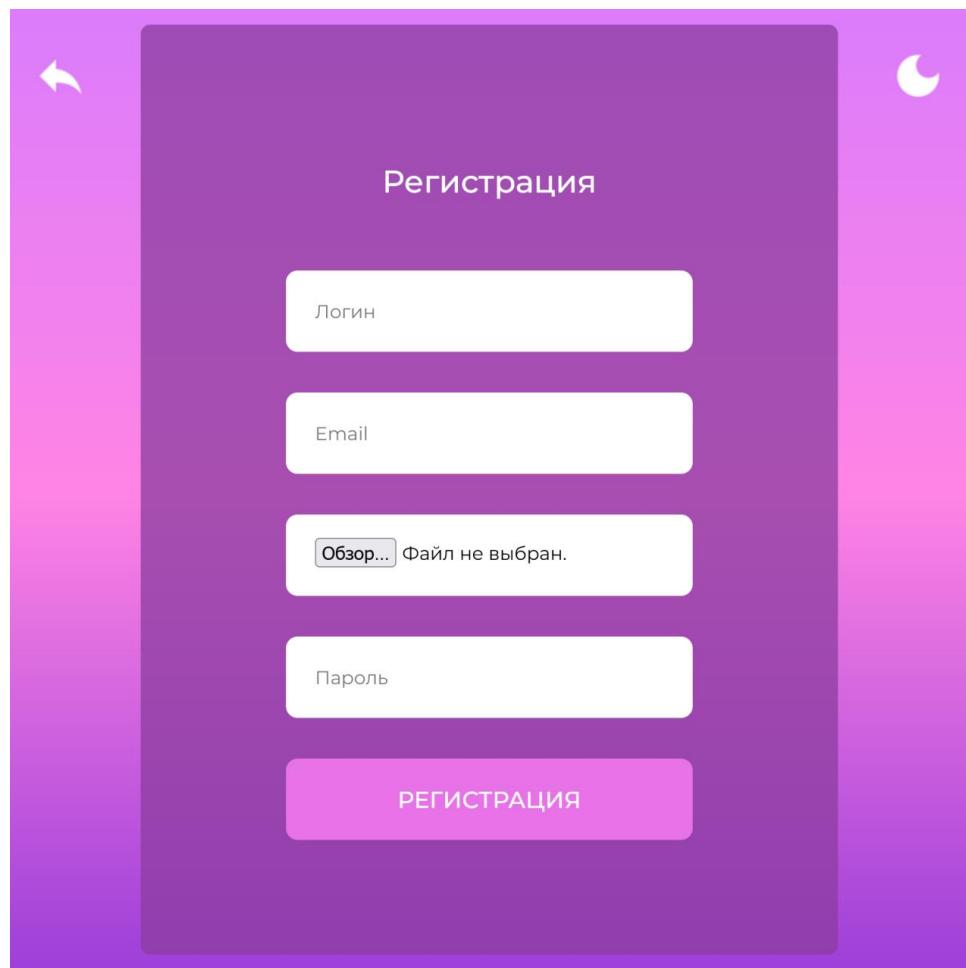


Рисунок 7.3 – Экранная форма страницы «Регистрация»

Элементы интерфейса:

- заголовок;
- форма авторизации:
 - 1) «Логин»: поле для ввода логина пользователя;
 - 2) «Email»: поле для ввода электронной почты пользователя;
 - 3) форма выбора файла логотипа;
 - 4) «Пароль»: поле для ввода пароля пользователя;

5) «РЕГИСТРАЦИЯ»: кнопка отправки формы авторизации.

- луна/солнце : кнопка смены цветовой темы;
- кнопка возврата на страницу «Главная».

Страница «Чат» (см. рис.7.4) открывается после успешной авторизации пользователя в системе.

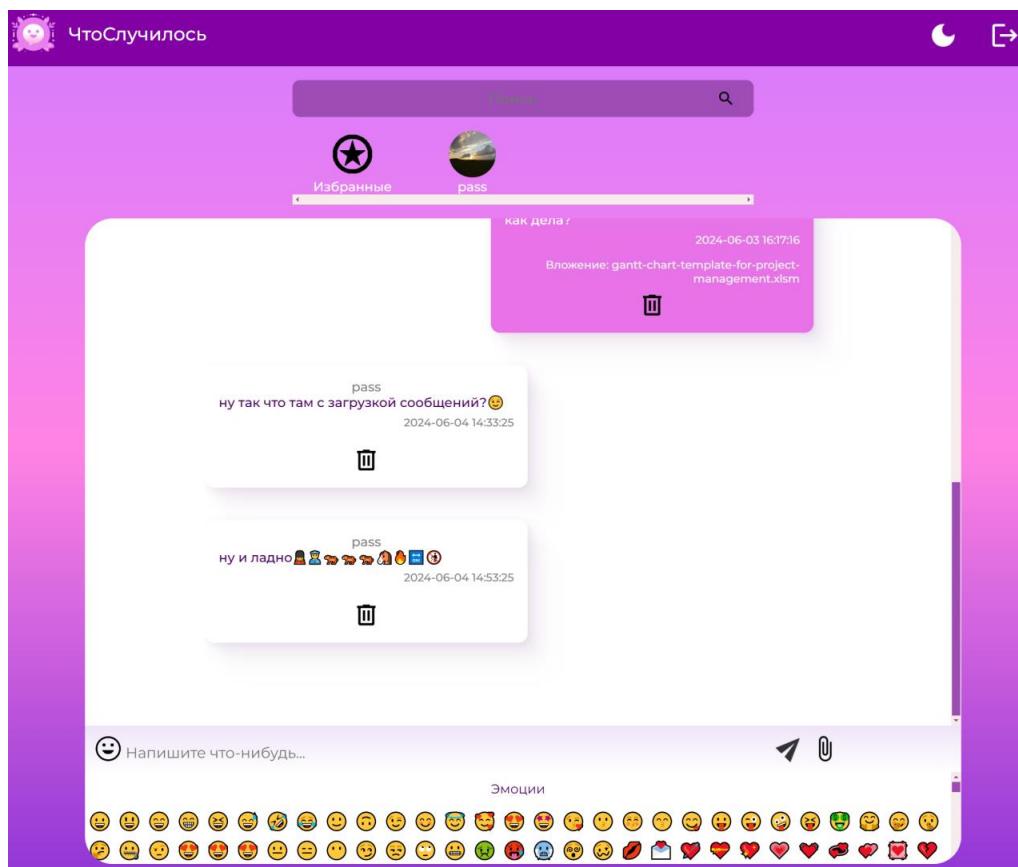


Рисунок 7.4 – Экранная форма страницы «Чат»

Элементы интерфейса:

- логотип;
- луна/солнце : кнопка смены темы сайта;
- кнопка выхода из аккаунта;
- форма поиска пользователя:
 - 1) «Поиск»: поле для ввода никнейма пользователя;
 - 2) лупа: поиск пользователей;

- 3) контейнер с пользователями;
- 4) звезда: переход в избранные сообщения;
- 5) логотип пользователя: переход в диалог с пользователем.

- форма чата:
 - 1) список сообщений диалога;
 - 2) «Напишите что-нибудь...»: поле для ввода текста сообщения;
 - 3) скрепка: форма выбора файла вложения;
 - 4) самолёт: кнопка отправки сообщения;
 - 5) смайлик: открытие списка смайликов;
 - 6) корзина: удаление сообщения;
 - 7) звезда: добавление в избранное сообщения.
- кнопка смены цветовой темы;
- кнопка возврата на страницу «Главная».

Описание функциональности.

Авторизация: пользователь вводит email и пароль, система проверяет данные и предоставляет доступ к чату.

Регистрация: новый пользователь вводит необходимые данные, система создает новый аккаунт.

Чат: пользователь может выбирать контакты, отправлять и получать сообщения в режиме реального времени, добавлять и просматривать избранные сообщения.

Таким образом этот интерфейс обеспечивает простой и интуитивно понятный пользовательский опыт, предоставляя основные функции мессенджера.

8 ТЕСТИРОВАНИЕ

Тестирование методом чёрного ящика - это основной метод, используемый специалистами по контролю качества для проверки цифровых продуктов, которые недоступны и внутренняя структура которых неизвестна.

Основная цель этой процедуры - последовательно исследовать, соответствует ли поведение программы текущим требованиям. Работоспособность изделия также проверяется в критических ситуациях, то есть при вводе неверных входных данных.

Тестирование страницы «Авторизация».

1. Проверка отображения страницы авторизации (см. рис. 8.1):
 - открыть страницу авторизации;
 - убедиться, что все элементы отображаются корректно;
 - предполагаемый результат: отображение страницы авторизации (поле email, поле ввода пароля, кнопка «Войти», ссылки «Назад» и «Тема»);
 - полученный результат: отображение страницы авторизации со всеми элементами.

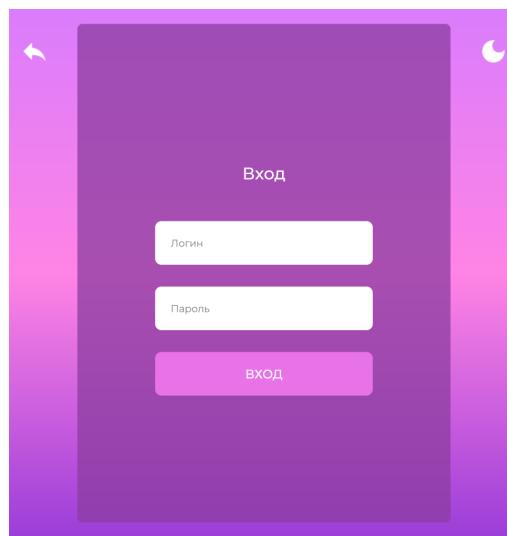


Рисунок 8.1 – Проверка отображения страницы авторизации

2. Валидация поля email (см. рис. 8.2):

- ввести некорректный email (например, «user» или «user@domain»);
 - проверить, что отображается сообщение об ошибке;
 - предполагаемый результат: отображение страницы ошибки или иного уведомления об ошибке или перезагрузка страницы;
 - полученный результат: отображение страницы ошибки.

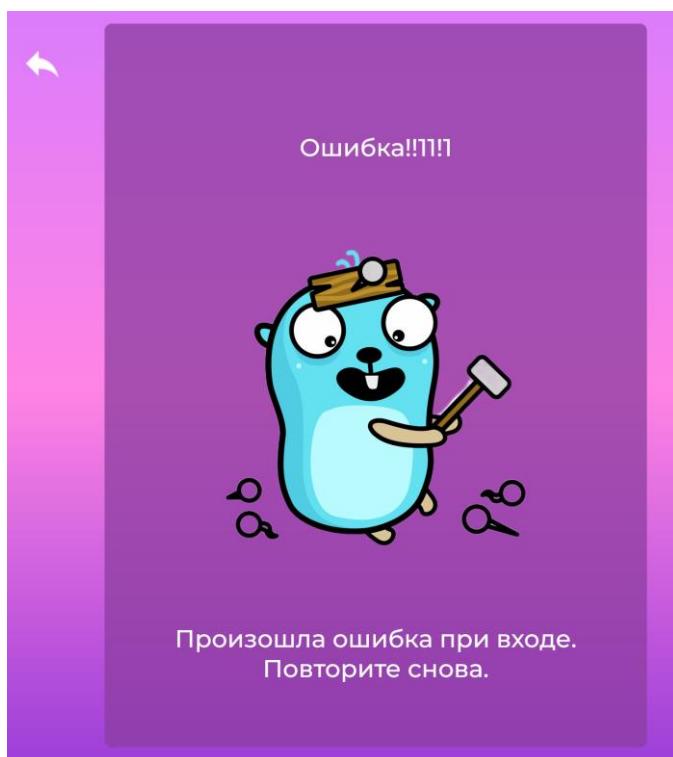


Рисунок 8.2 – Валидация поля email

3. Валидация поля пароля (см.рис. 8.3):

- оставить поле пароля пустым;
- предполагаемый результат: отображение страницы ошибки или иного уведомления об ошибке или перезагрузка страницы;
- полученный результат: отображение страницы ошибки.

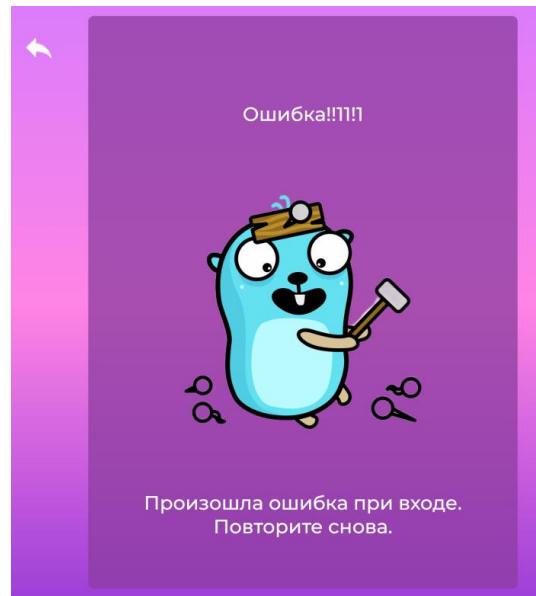


Рисунок 8.3 – Валидация поля пароля

4. Авторизация с неверными данными (см. рис. 8.4):

- ввести несуществующий email и любой пароль и нажать на кнопку «ВХОД»;
- предполагаемый результат: отображение страницы ошибки или иного уведомления об ошибке или перезагрузка страницы;
- полученный результат: отображение страницы ошибки.

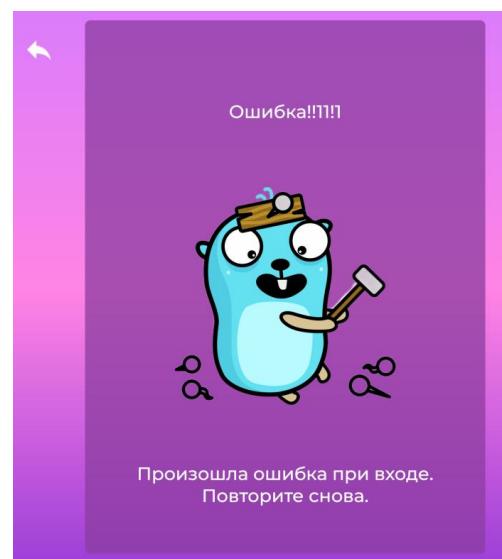


Рисунок 8.4 – Авторизация с неверными данными

5. Авторизация с корректными данными (см.рис. 8.5):

- ввести существующий email и правильный пароль и нажать на кнопку «ВХОД»;
- предполагаемый результат: отображение страницы чата с присутствующими элементами (список сообщений, список контактов, кнопка смены темы, кнопка выхода из аккаунта, кнопка открытия списка смайликов, кнопка отправки, кнопка добавления вложения);
- полученный результат: отображение страницы чата со всеми элементами.

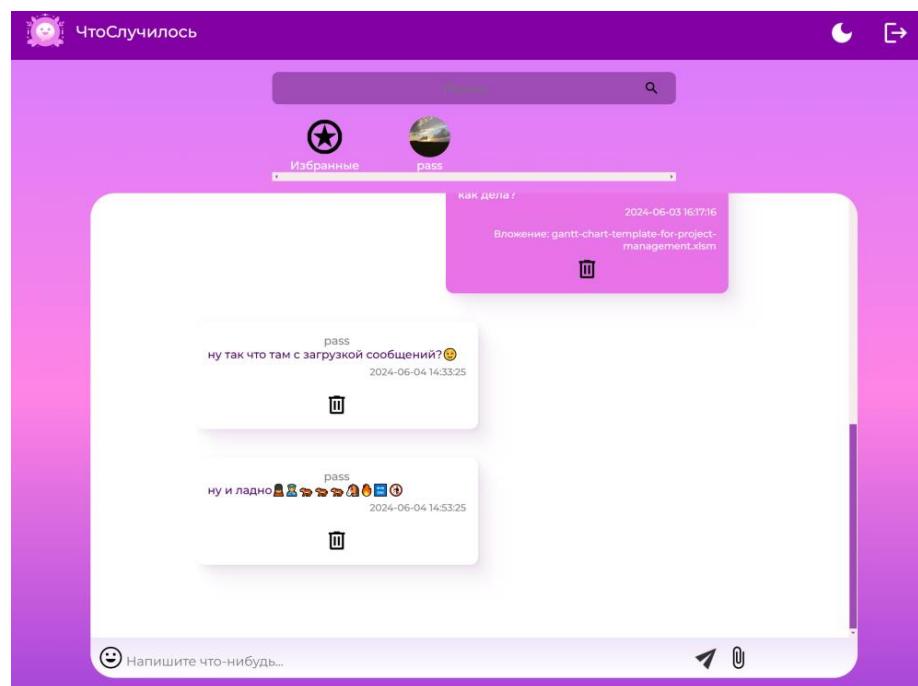


Рисунок 8.5 – Авторизация с корректными данными

6. Ссылка «Регистрация» (см. рис. 8.6):

- нажать на ссылку «Регистрация»;
- проверить, что происходит переход на страницу регистрации;
- полученный результат: отображение страницы регистрации со всеми элементами.

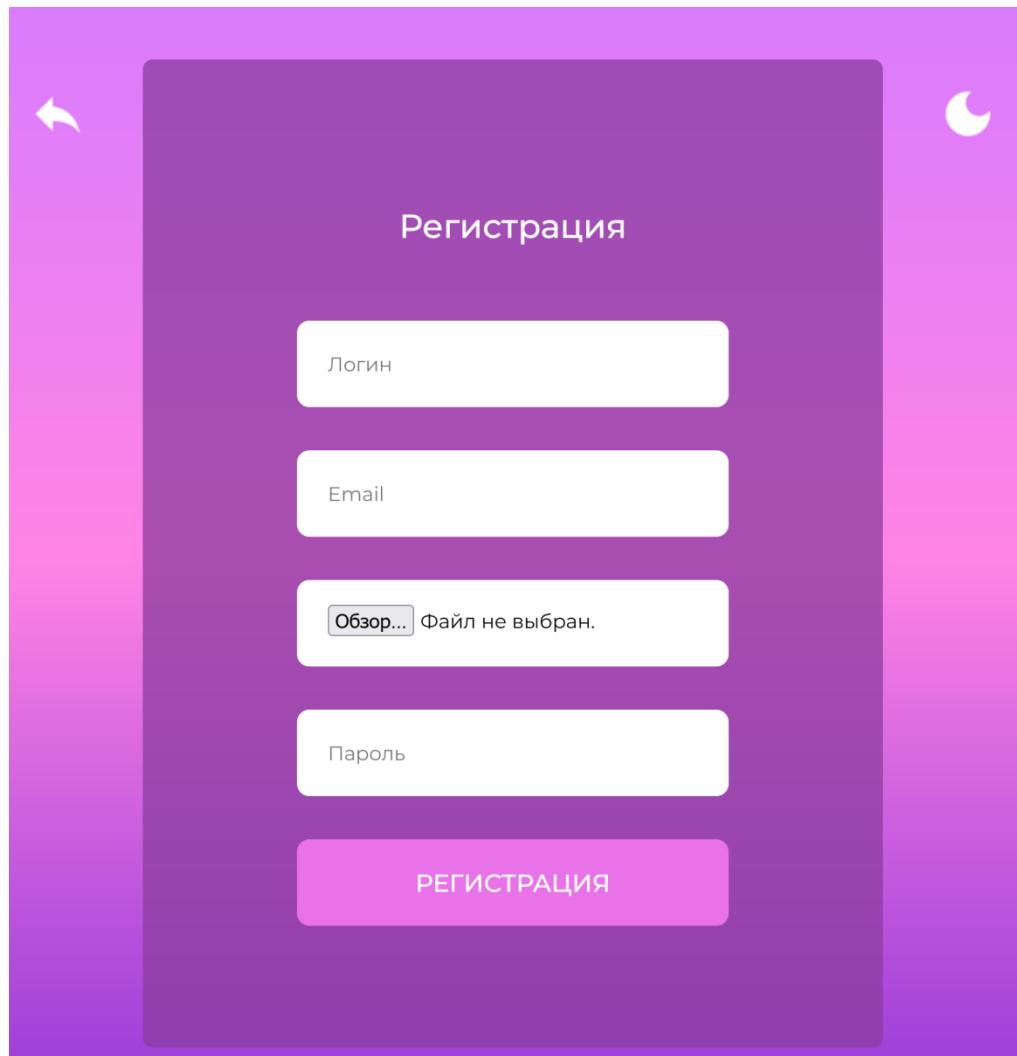


Рисунок 8.6 – Ссылка «Регистрация»

Тестирование страницы «Регистрация».

1. Проверка отображения страницы регистрации (см.рис. 8.7):
 - открыть страницу регистрации;
 - предполагаемый результат: отображение страницы авторизации (поле имени, поле email, поле пароля, кнопка смены темы, кнопка «назад»);
 - полученный результат: отображение страницы регистрации со всеми элементами.

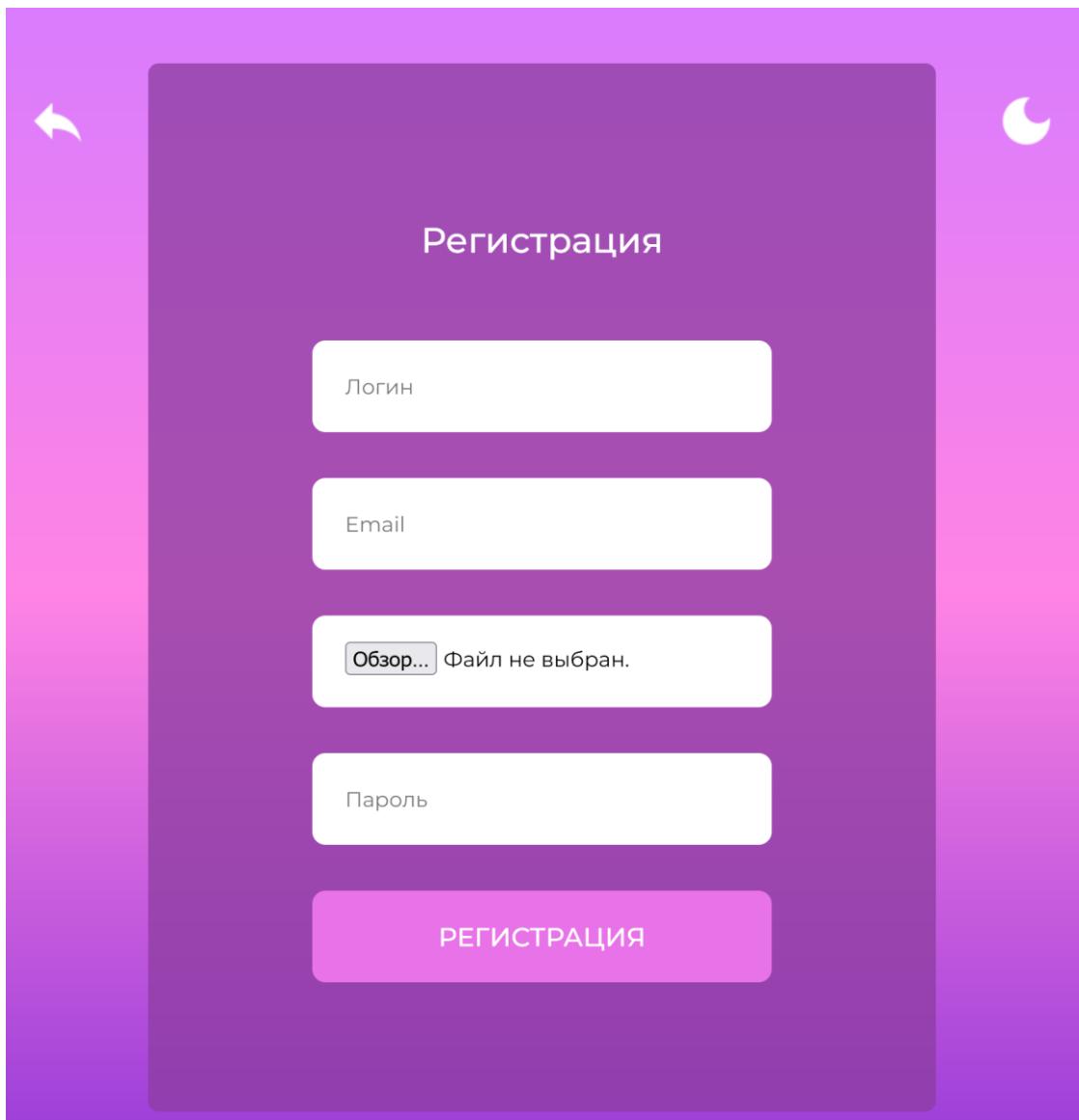


Рисунок 8.7 – Проверка отображения страницы регистрации

2. Валидация поля email (см.рис. 8.8):
 - ввести некорректный email;
 - предполагаемый результат: отображение страницы ошибки или иного уведомления об ошибке или перезагрузка страницы;
 - полученный результат: очистка полей для ввода.

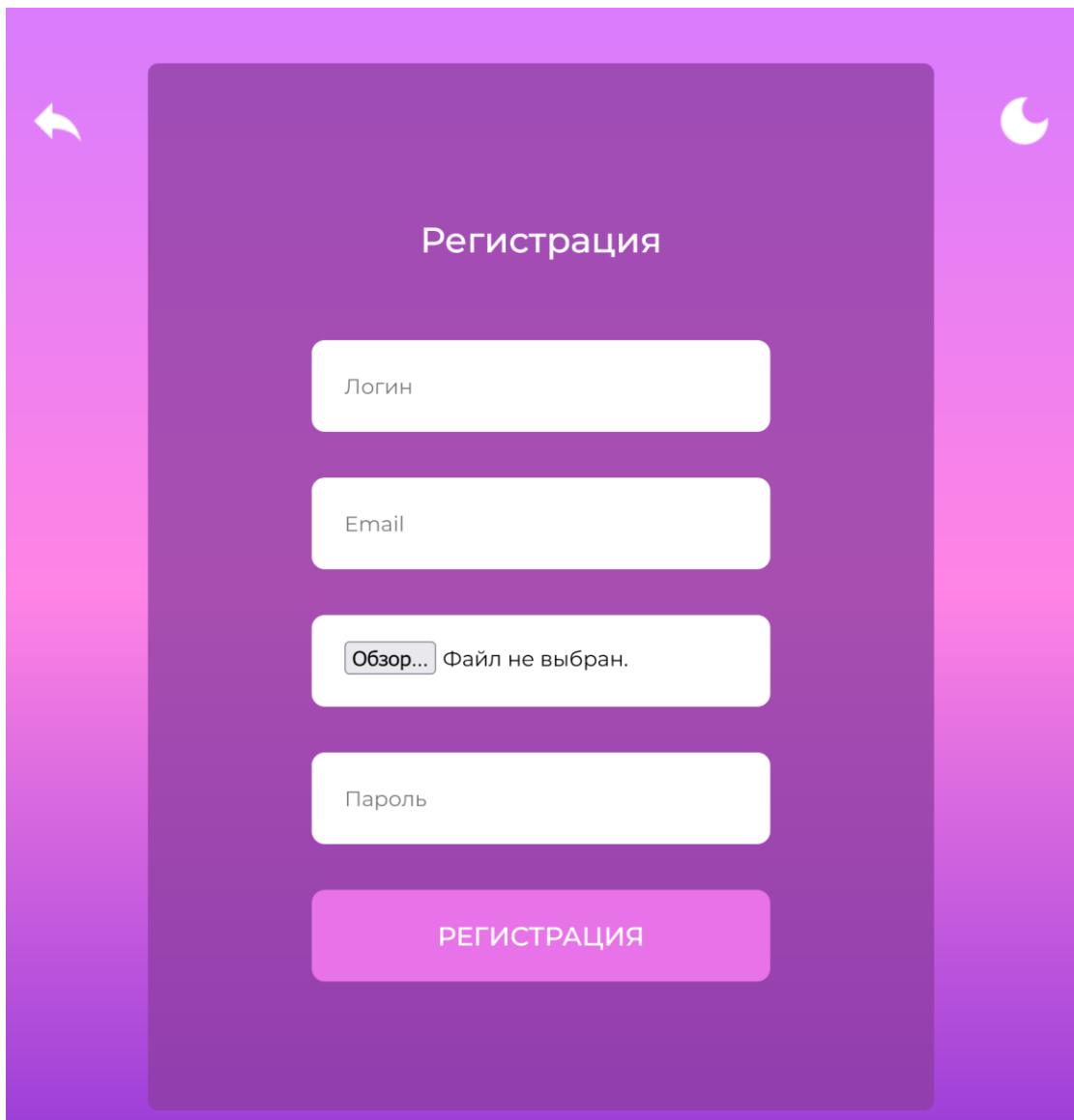


Рисунок 8.8 – Валидация поля email

3. Валидация поля пароля (см.рис. 8.9):

- ввести пустой пароль;
- предполагаемый результат: отображение страницы ошибки или иного уведомления об ошибке или перезагрузка страницы.
- полученный результат: очистка полей для ввода.

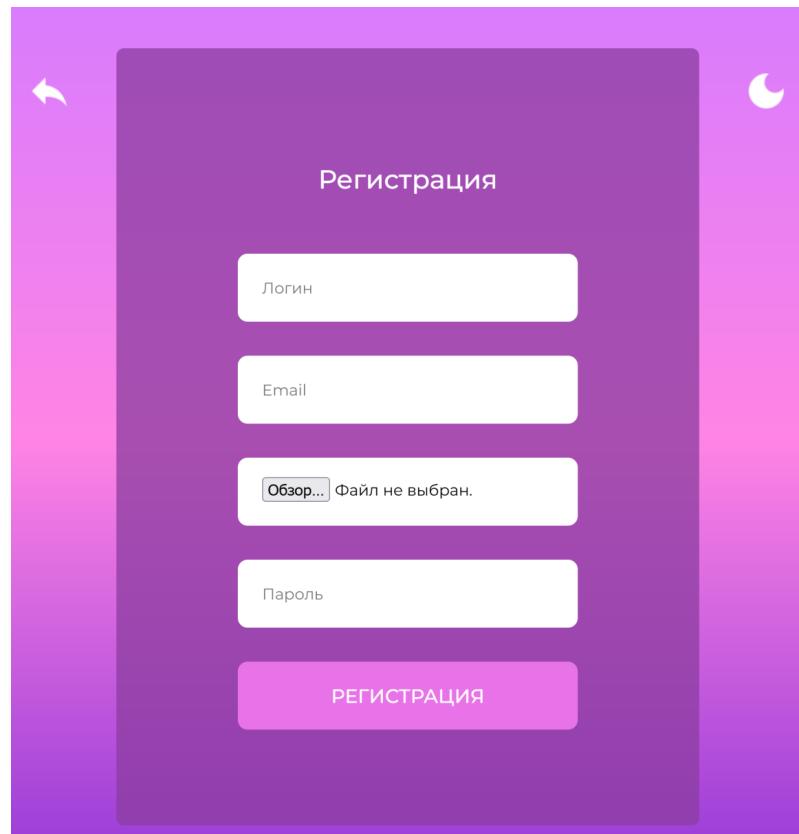


Рисунок 8.9 – Валидация поля пароля

4. Регистрация без логотипа (см.рис. 8.10):

- не добавляя файл нажать «РЕГИСТРАЦИЯ»;
- предполагаемый результат: отображение страницы ошибки или иного уведомления об ошибке или перезагрузка страницы;
- полученный результат: окно с ошибкой.

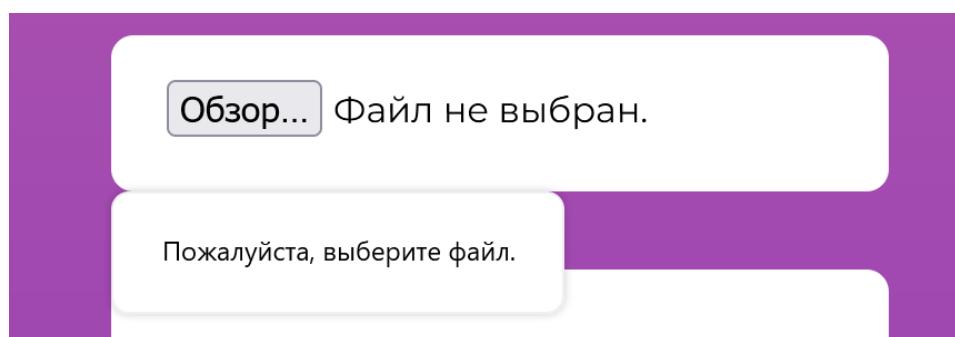


Рисунок 8.10 – Регистрация без логотипа

5. Регистрация с корректными данными (см.рис. 8.11):

- вести валидные данные;
- предполагаемый результат: отображение страницы чата с присутствующими элементами(список сообщений, список контактов, кнопка смены темы, кнопка выхода из аккаунта, кнопка открытия списка смайликов, кнопка отправки, кнопка добавления вложения);
- полученный результат: отображение страницы чата со всеми элементами.

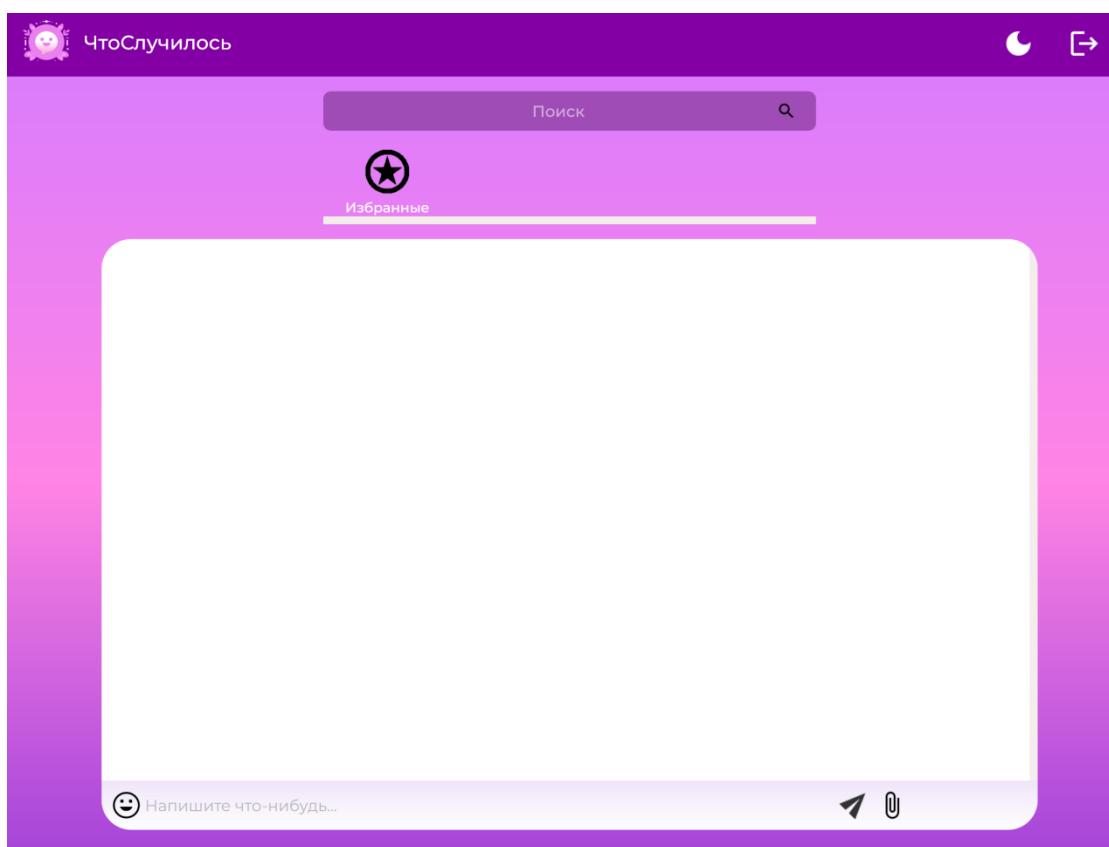


Рисунок 8.11 – Регистрация с корректными данными

Тестирование страницы «Чат».

1. Проверка отображения страницы чата (см.рис. 8.12):
- переход на страницу чата авторизированному пользователю;
 - предполагаемый результат: отображение страницы чата с присутствующими элементами (список сообщений, список контактов,

кнопка смены темы, кнопка выхода из аккаунта, кнопка открытия списка смайликов, кнопка отправки, кнопка добавления вложения);

- полученный результат: отображение страницы чата со всеми элементами.

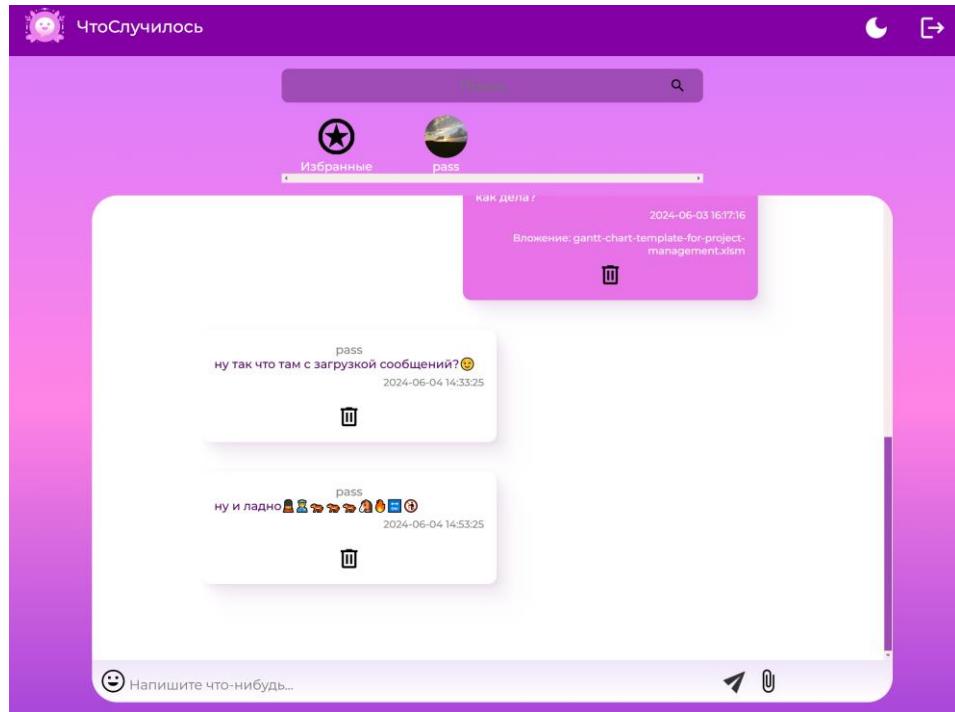


Рисунок 8.12 – Проверка отображения страницы чата

2. Отправка сообщения (см.рис. 8.13):

- ввести текст в поле ввода сообщения;
- нажать кнопку отправки сообщения;
- предполагаемый результат: отображение страницы чата с присутствующим новым сообщением (временем отправки, кнопок добавления в избранное, удаления сообщения, текста сообщения, ссылкой на вложение);
 - полученный результат: отображение нового сообщения со всеми элементами.

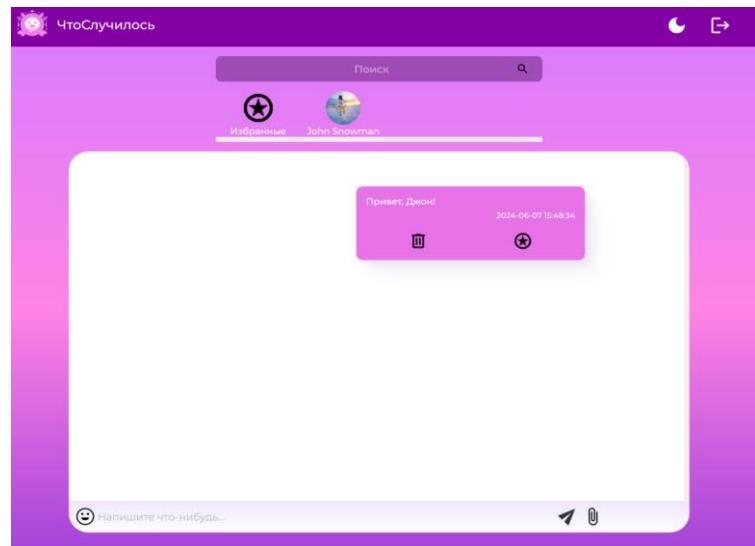


Рисунок 8.13 – Отправка сообщения

3. Получение сообщения (см.рис. 8.14):
- отправить сообщение из другого аккаунта;
 - предполагаемый результат: отображение страницы чата с присутствующим новым сообщением (временем отправки, кнопок добавления в избранное, удаления сообщения, текста сообщения, ссылкой на вложение) без перезагрузки;
 - полученный результат: отображение нового сообщения со всеми элементами без перезагрузки.

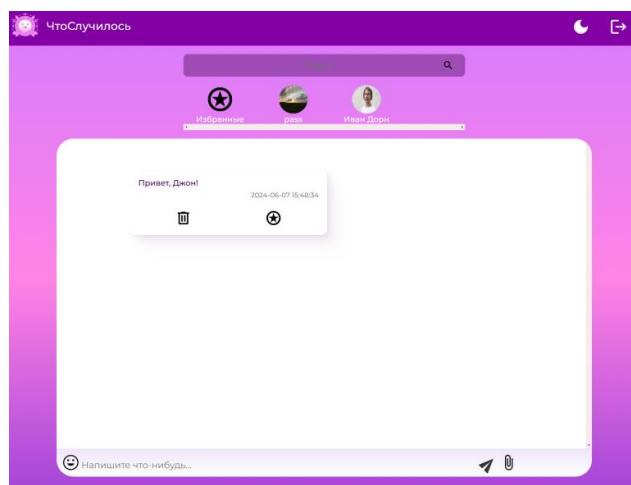


Рисунок 8.14 – Получение сообщения

4. Отправка пустого сообщения (см.рис. 8.15):

- оставить поле ввода пустым и нажать кнопку отправки;
- предполагаемый результат: отображение страницы чата без нового сообщения или сообщение об ошибке;
- полученный результат: ничего не изменилось.

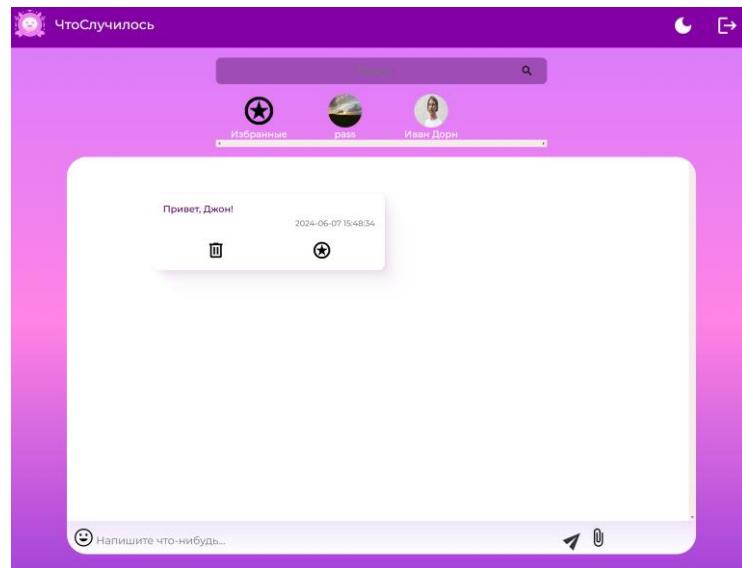


Рисунок 8.15 – Отправка пустого сообщения

5. Переключение между контактами (см.рис. 8.16):

- нажать на разные контакты в списке;
- предполагаемый результат: отображение страницы чата с другими сообщениями с диалога выбранных пользователей;
- полученный результат: после перезагрузки страницы сообщения в чате изменились на сообщения из диалога с другим пользователем.

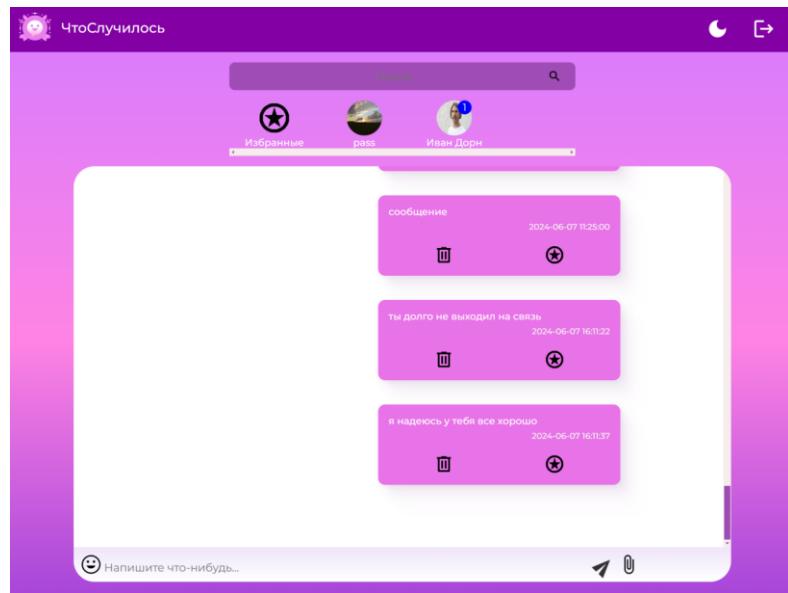


Рисунок 8.16 – Переключение между контактами

6. Проверка меток времени сообщений (см.рис. 8.17):

- предполагаемый результат: отображение времени отправки сообщений отображаются в полном объёме (год, день, час, минута);
- полученный результат: сообщения имеют подпись со временем отправки (год, день, час, минута, секунда).

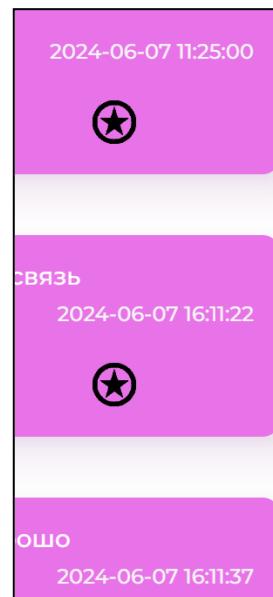


Рисунок 8.17 – Проверка меток времени сообщений

7. Проверка отображения непрочитанных сообщения (см.рис. 8.18):

- отправить сообщение из аккаунта А аккаунту Б: у аккаунта Б в диалоге с другим пользователем должно отобразится уведомление о непрочитанном сообщении;
- предполагаемый результат: отображение уведомления о новом сообщении;
- полученный результат: возле контакта появляется счётчик непрочитанных сообщений.



Рисунок 8.18 – Проверка уведомления о новых сообщениях

Тестирование методом чёрного ящика чат-мессенджера включает проверку функциональных элементов на каждой странице, а также валидацию ввода данных и корректность отображения сообщений. По результатам тестирования программное обеспечение обеспечивает должную стабильность, логика отображения экранных форм соблюдена.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была достигнута цель создания чат-мессенджера с использованием языка программирования Go и фреймворка Echo. Разработанное приложение демонстрирует высокую производительность, масштабируемость и надёжность, что подтверждается результатами тестирования.

Основные достижения проекта.

Разработка серверной части: создано серверное приложение на Go, обеспечивающее обработку запросов, управление соединениями пользователей и маршрутизацию сообщений. Использование фреймворка Echo позволило существенно ускорить процесс разработки благодаря его простоте и гибкости.

Обеспечение безопасности: реализованы механизмы аутентификации и авторизации пользователей, а также меры по защите данных и предотвращению возможных атак. Использование JWT-токенов и MD5 хеширования обеспечивает высокий уровень безопасности обмена данными.

Интеграция с базой данных: в проекте использована база данных MariaDB для хранения пользовательской информации и сообщений. Это решение обеспечило надёжное и эффективное управление данными.

Тестирование и отладка: проведены обширные тестирования и отладка приложения, включающие ручное тестирование исключительных ситуаций. Это позволило выявить и устранить возможные ошибки и узкие места в производительности и логике ПО.

Проект «Многопользовательский чат-мессенджер «Что случилось?»» является примером успешной реализации современных технологий и практик в области разработки веб-приложений. Полученные результаты подтверждают жизнеспособность выбранных решений и технологий.

СПИСОК ЛИТЕРАТУРЫ

1. Мессенджер – что это такое, для чего он нужен – URL:
<https://elama.ru/glossary/messendzher> (дата обращения: 29.05.2024).
2. WhatsApp: история самого популярного мессенджера в мире – URL:
<https://dzen.ru/a/ZEDARmuMHklH3UfL> (дата обращения: 31.05.2024).
3. MariaDB: что это, как устроена, чем хороша и как начать с ней работать – URL: <https://skillbox.ru/media/code/MariaDB-chto-eto-kak-ustroena-chem-khorosha-i-kak-nachat-s-ney-rabotat/> (дата обращения: 31.05.2024).
4. HTML или CMS – что лучше для коммерческого сайта – URL:
<https://vzh.ru/article/html-ili-cms-chto-luchshe-dlya-kommercheskogo-sajta/> (дата обращения: 01.06.2024).
5. CSS – Что такое, история создания и преимущества – URL:
<http://proglang.su/css/introduction> (дата обращения: 01.06.2024).
6. Русская документация по API jQuery – URL: <https://jquery-docs.ru/> (дата обращения: 03.06.2024).
7. Руководство по языку Go – URL: <https://metanit.com/go/tutorial/> (дата обращения: 05.06.2024).

ПРИЛОЖЕНИЕ А

Фрагменты программного кода

```
//main.go
package main

import (
    "chat/additional"
    "chat/handlersPackage"
    "chat/logger"
    "chat/variables"
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
    "github.com/golang-jwt/jwt/v5"
    "github.com/labstack/echo-jwt/v4"
    "github.com/labstack/echo/v4"
    "html/template"
)

// Handlers

func main() {
    variables.Db, _ = sql.Open("mysql", "mysql:mysql@/chat")
    e := echo.New()
    logger.NewLogger()          // new
    e.Use(logger.LoggingMiddleware) // ne
    e.HTTPErrorHandler = additional.CustomHTTPErrorHandler
    //e.AutoTLSManager.Cache = autocert.DirCache("/var/www/.cache")
    e.Renderer = &Template{
        templates: template.Must(template.ParseGlob("web/*.html")),
    }
    e.Static("/", "web/")
    // Routes
    e.GET("/login", handlersPackage.Auth)
    e.POST("/login", handlersPackage.AuthPOST)
    e.GET("/authMain", handlersPackage.AuthMain)
    e.GET("/registration", handlersPackage.RegGET)
    e.POST("/registration", handlersPackage.RegPOST)
    e.GET("/chat", handlersPackage.ChatGET)
    e.POST("/chat", handlersPackage.ChatPOST)
    e.GET("/", handlersPackage.MainPage)
    e.POST("/refresh", handlersPackage.GetMessages)
    e.POST("/loaduserconnects", handlersPackage.LoaderUser)
    e.POST("/loadnewmessage", handlersPackage.LoadNewMessageCounter)
    // Restricted group
    r := e.Group("/")
}

// Configure middleware with the custom claims type
config := echojwt.Config{
    NewClaimsFunc: func(c echo.Context) jwt.Claims {
        return new(variables.JwtCustomClaims)
    },
    SigningKey: variables.Secret,
    TokenLookup: "cookie:JWTToken",
}
r.Use(echojwt.WithConfig(config))

logger.Logger.LogInfo().Msg(e.Start(":1323").Error())
//e.Logger.Fatal(e.StartAutoTLS(":443"))
//e.Start(":1323")
//if l, ok := e.Logger.(*log.Logger); ok {
//    l.SetHeader("${time_rfc3339} ${level}")
//}
}
```

```

//render.go

package main

import (
    "github.com/labstack/echo/v4"
    "html/template"
    "io"
)

// struct for custom template
type Template struct {
    templates *template.Template
}

// custom render func
func (t *Template) Render(w io.Writer, name string, data interface{}, c echo.Context) error {
    return t.templates.ExecuteTemplate(w, name, data)
}

//web/errors/500.html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ошибка</title>
    <link rel="stylesheet" href="style/style.css">
    <link rel="stylesheet" href="style/reset.css">
</head>
<body>
<div class="authContainer">
    <a class="backButton" href="/"></a>
    <div class="authInnerContainer">
        <div class="authLogo">
            <h1>Ошибка!!1!1</h1>
        </div>
        
        <h2 class="errorMessage">Произошла ошибка при входе. Повторите снова.</h2>
    </div>
</div>

</body>
</html>

//web/errors/500DARK

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ошибка</title>
    <link rel="stylesheet" href="style/style.css">
    <link rel="stylesheet" href="style/reset.css">
</head>
<body class="bodyDark">
<div class="authContainer">
    <a class="backButton" href="/"></a>
    <div class="authInnerContainerDARK">
        <div class="authLogo">
            <h1>Ошибка!!1!1</h1>
        </div>
        
    </div>
</div>

```

```

<h2 class="errorMessage">Произошла ошибка при входе. Повторите снова.</h2>

</div>

</div>

</body>
</html>

//web/style/reset.css

*{padding:0;margin:0;border:none}*,:after,:before{box-sizing:border-box}a,a:hover,a:link,a:visited{text-decoration:none}aside,footer,header,legend,main,nav,section{display:block}h1,h2,h3,h4,h5,h6,p{font-size:inherit;font-weight:inherit}ul,ul li{list-style:none}img{vertical-align:top}img,svg{max-width:100%;height:auto}address{font-style:normal}input::-ms-clear{display:none}button,input[type=submit]{display:inline-block;box-shadow:none;background:0 0;cursor:pointer}button:active,button:focus,input:active,input:focus{outline:0}button::-moz-focus-inner{padding:0;border:0}label{cursor:pointer}

//web/style/style.css

.bodyDark,body{height:100vh;width:100vw;display:flex}.bodyDark,.logo,body{display:flex}.errorMessage,.logo h1{color:#fff;font-family:Montserrat;font-size:27px;font-weight:500;line-height:33px;margin-top:50px}.authContainer form input,.authContainerDARK form input,select{margin-bottom:35px;padding:20px 25px;font-weight:400;line-height:20px*}.userCard h1{overflow:hidden}.smileSlide a:hover,.userCard img:hover,img:hover{transform:scale(1.2);transition-duration:.4s}@font-face{font-family:Montserrat;src:url(/fonts/Montserrat-VariableFont.ttf)}body{background:linear-gradient(180deg,#d67bff,#ff84e5,#8e34d6);flex-direction:column}.bodyDark{background:linear-gradient(180deg,#671A8FA6,#c954D0FF,#e34FC3FF);flex-direction:column}.logo{margin-bottom:60px;width:440px}.authInnerContainer,.authInnerContainerDARK{width:600px;height:800px;border-radius:8px;display:flex}.logo h1{letter-spacing:0;text-align:center;margin-left:-25px}.errorMessage{letter-spacing:0;text-align:center}.authButton,.authButtonDARK{box-shadow:0 0 0 rgba(142,70,175,,1),2px 2px 6px 0 rgba(142,70,175,,1),9px 8px 12px 0 rgba(142,70,175,,09),19px 18px 16px 0 rgba(142,70,175,,05),34px 32px 19px 0 rgba(142,70,175,,01),53px 50px 20px 0 rgba(142,70,175,,0);margin-bottom:30px;font-size:20px;line-height:24px;color:#fff;letter-spacing:0;font-weight:500;text-align:center}.authInnerContainer{background-color:rgba(139,60,159,,75);align-items:center;flex-direction:column;justify-content:center}.authInnerContainerDARK{background:rgba(113,3,143,,5);align-items:center;flex-direction:column;justify-content:center}.authButton,.authButtonDARK,.authContainer form input,.authContainerDARK form input,select{width:350px;height:70px;border-radius:10px;font-family:Montserrat}.authContainer{margin-top:75px;display:flex;flex-direction:row;justify-content:center}.authButton{align-items:center;background:#e872e7}.authButtonDARK{align-items:center;background:#71038f}.authContainer lightMode{margin:20px 0 0 45px}.authContainer .backButton{margin:20px 45px 0 0}.authLogo{color:#fff;font-family:Montserrat;font-size:27px;font-weight:500;line-height:33px;letter-spacing:0;text-align:center;margin:50px 0 60px}.header,.headerDARK,.searchContainer{margin-bottom:15px;display:flex}.authContainer form{display:flex;flex-direction:column}.authContainer form input,select{background-color:#fff;font-size:16px}.authContainerDARK form input,select{background:rgba(71,68,68,,99);font-size:16px}.header{justify-content:space-around;flex-direction:row;height:65px;background-color:#8200a4}.headerDARK{justify-content:space-around;flex-direction:row;height:65px;background-color:#71038f}.body,.userCard{flex-direction:column}.chatLogo{display:flex;color:#fff;font-family:Montserrat;font-size:20px;font-weight:500;line-height:24px;letter-spacing:0;text-align:center}.chatLogo img{height:45px;margin-top:6px}.chatLogo h1{margin-top:18px}.blockButton img{margin:15px 30px 0 0;width:33px;height:33px}.body{display:flex;justify-content:center;align-items:center}.searchContainer{width:500px;height:40px;background-color:#8b3C9FBF;border-radius:8px}.searchContainerDARK{display:flex;width:500px;height:40px;background-color:#cc5fcf;border-radius:8px;margin-bottom:15px}.userListContainer,.userListContainerDARK{width:500px;height:100px;overflow-x:scroll;margin-bottom:15px;scrollbar-width:thin;border-radius:8px;display:flex}.searchContainer button,.searchContainerDARK button{width:20px}.searchContainer input,.searchContainerDARK input{width:460px;height:40px;background-color:transparent;color:#fff;font-family:Montserrat;font-size:16px;font-weight:500;line-height:20px;letter-spacing:0;text-align:center}.userListContainer{flex-direction:row;scrollbar-color:rgb(158,75,179) rgb(245,236,236)}.userListContainerDARK{flex-direction:row;scrollbar-color:#FD60FF #B27CB2}.messages,.smileSlideOuter{overflow-y:scroll;scrollbar-width:thin;scrollbar-color:rgb(158,75,179) rgb(245,236,236)}.userCard img:hover{border-radius:35% !important;margin-top:10px}.userCard{display:flex;justify-content:center;align-items:center;flex:0 0 130px}.userCard a{display:flex;flex-direction:column;align-items:center}.userCard img{border-radius:50%;height:55px}.userCard h1{color:#fff;font-family:Montserrat;font-size:14px;font-weight:500;line-height:14px;letter-spacing:0;text-align:center}.newMessage{font-family:Montserrat;line-height:20px;text-align:center;position:absolute;color:#fff;font-size:15px;margin-left:30px;background-color:#00f;border-radius:50%;width:20px;height:20px}.chatBody,.chatBodyDARK{width:950px;border-radius:30px}.chatBody{background-color:#fff}.chatBodyDARK{background-color:#474444FC}.messages{height:550px;letter-spacing:0;font-weight:500;margin-left:130px;padding:55px 0}.messageLeft,.messageLeftDARK{width:350px;border-radius:10px;font-family:Montserrat;font-size:14px;line-height:17px;text-align:left;display:flex;min-height:70px;padding:15px;margin-bottom:35px}.messageBottom{display:flex;justify-content:space-around}.messages a img{width:30px}.messages a{text-align:center}.messages span{text-align:right;font-family:Montserrat;font-size:12px;line-height:15px;margin:6px 0}.messageLeft{box-shadow:0 0 0 rgba(148,99,161,,1),2px 2px 7px 0 rgba(148,99,161,,1),8px 9px 12px 0 rgba(148,99,161,,09),17px 21px 16px 0 rgba(148,99,161,,05),31px 37px 19px 0 rgba(148,99,161,,01),48px 58px 21px 0 rgba(148,99,161,0);background:#fff;color:#d0161;flex-direction:column;justify-content:space-around}.messageLeftDARK{box-shadow:0 0 0 rgba(203,119,204,,1),1px 1px 4px 0 rgba(203,119,204,,1),5px 5px 7px 0 rgba(203,119,204,,09),10px 12px 9px 0 rgba(203,119,204,,05),19px 21px 11px 0 rgba(203,119,204,,01),29px 32px 12px 0 rgba(203,119,204,0);background:#4a494a;color:#fd60ff;flex-direction:column;justify-content:space-around}.messageRight,.messageRightDARK{width:350px;border-radius:10px;color:#fff;font-size:14px;min-

```

```

height:70px;padding:15px;margin:0 0 35px 310px;text-align:left;display:flex;font-family:Montserrat;line-height:17px}.messageLeft
a,.messageLeft span,.messageRight a{color:#848484}.messageRight{box-shadow:0 0 0 rgba(148,99,161,1),2px 2px 7px 0
rgba(148,99,161,1),8px 9px 12px 0 rgba(148,99,161,09),17px 21px 16px 0 rgba(148,99,161,05),31px 37px 19px 0
rgba(148,99,161,01),48px 58px 21px 0 rgba(148,99,161,0);background:#e872e7;flex-direction:column;justify-content:space-around}.messageRightDARK{box-shadow:0 0 0 rgba(203,119,204,1),1px 1px 4px 0 rgba(203,119,204,1),4px 6px 7px 0
rgba(203,119,204,,09),9px 13px 9px 0 rgba(203,119,204,05),15px 23px 11px 0 rgba(203,119,204,01),24px 36px 12px 0
rgba(203,119,204,0);background:#71038f;flex-direction:column;justify-content:space-around}.messageRight
span{color:#f6f6f6}.messageLeftDARK a,.messageRightDARK a,p.smileDarkText{color:#fff}.sendPanel{height:50px;background:linear-gradient(180deg,#f0e5fa,rgba(255,255,255,0) 100%)}.sendPanelDARK{height:50px;background:linear-gradient(180deg,#696868,#484445 100%)}.sendPanel
.textInput{width:700px;height:50px;background-color:transparent;color:#848484;font-family:Montserrat;font-size:16px;line-height:20px;text-align:left}.sendPanelDARK
.textInput{width:700px;height:50px;background-color:transparent;color:#b2b0b0;font-family:Montserrat;font-size:16px;line-height:20px;text-align:left}.sendPanel button,.sendPanelDARK
button{height:35px;width:35px;opacity:.8}.sendPanel img,.sendPanelDARK img{max-width:30px;max-height:30px}.sendBtn{margin-top:-10px;transform:rotate(-45.6deg)}input#file-input{display:none}.attachImg{margin-top:10px}.smileBtn{margin-top:9px;margin-left:10px;cursor:pointer}.smileSlide{display:flex;flex-direction:row;flex-wrap:wrap}.smileSlideOuter{height:100px}.smileSlide
a{display:block;margin:3px;cursor:pointer;font-size:19px}.smileSlide
p{color:#4d0161;font-family:Montserrat;font-size:14px;line-height:17px;margin:10px}

```

//web/auth.html

```

<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Вход в аккаунт</title>
<link rel="stylesheet" href="/style/style.css">
<link rel="stylesheet" href="/style/reset.css">
<link rel="icon" href="/static/favicon.ico">
</head>
<body>
<div class="authContainer">
<a class="backButton" href="/authMain"></a>
<div class="authInnerContainer">
<div class="authLogo">
<h1>Вход</h1>
</div>

<form method="post">
<input type="text" id="login" name="username" placeholder="Логин" required>
<input type="password" id="password" name="password" placeholder="Пароль" required>
<button class="authButton" type="submit">ВХОД</button>
</form>
</div>
<a class="lightMode" href="/login?toggle=true"></a>
</div>

```

```

</body>
</html>

```

//web/authDARK.html

```

<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Вход в аккаунт</title>
<link rel="stylesheet" href="/style/style.css">
<link rel="stylesheet" href="/style/reset.css">
<link rel="icon" href="/static/favicon.ico">
</head>
<body class="bodyDark">
<div class="authContainer">
<a class="backButton" href="/authMain"></a>
<div class="authInnerContainerDARK">
<div class="authLogo">

```

```

        <h1>Вход</h1>
    </div>

    <form method="post">
        <input type="text" id="login" name="username" placeholder="Логин" required>
        <input type="password" id="password" name="password" placeholder="Пароль" required>
        <button class="authButtonDARK" type="submit">ВХОД</button>
    </form>
    </div>
    <a class="lightMode" href="/login?toggle=true"></a>
</div>

</body>
</html>

//web/authMain.html

<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Вход в аккаунт</title>
    <link rel="stylesheet" href="/style/style.css">
    <link rel="stylesheet" href="/style/reset.css">
    <link rel="icon" href="/static/favicon.ico">
</head>
<body>
    <div class="authContainer">
        <div class="authInnerContainer" style="margin-left: 93px">
            <div class="logo">
                
                <h1>ЧтоСлучилось</h1>
            </div>
            <a href="/login"><button class="authButton">ВХОД</button></a>
            <a href="/registration"><button class="authButton">РЕГИСТРАЦИЯ</button></a>
        </div>
        <a class="lightMode" href="/authMain?toggle=true"></a>
    </div>

    </body>
</html>

//web/authMainDARK.html

<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Вход в аккаунт</title>
    <link rel="stylesheet" href="/style/style.css">
    <link rel="stylesheet" href="/style/reset.css">
    <link rel="icon" href="/static/favicon.ico">
</head>
<body class="bodyDark">
    <div class="authContainer">
        <div class="authInnerContainerDARK" style="margin-left: 93px">
            <div class="logo">
                
                <h1>ЧтоСлучилось</h1>
            </div>
            <a href="/login"><button class="authButtonDARK">ВХОД</button></a>
            <a href="/registration"><button class="authButtonDARK">РЕГИСТРАЦИЯ</button></a>
        </div>
        <a class="lightMode" href="/authMain?toggle=true"></a>
    </div>

```

```

</body>
</html>

//web/chat.html

<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Чат</title>
    <link rel="stylesheet" href="style/style.css">
    <link rel="stylesheet" href="style/reset.css">
    <link rel="icon" href="/static/favicon.ico">
    <script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNg4=" crossorigin="anonymous"></script>
    <script src="static/script.js"></script>
</head>
<body>
    <div class="header">
        <div class="chatLogo">
            
            <h1>Что Случилось</h1>
        </div>
        <div class="blockButton">
            <a class="lightMode" href="/chat?toggle=true"></a>
            <a class="lightMode" href="/chat?logout=true"></a>
        </div>
    </div>

    <div class="body">
        <div class="searchContainer">
            <input id="searchInput" type="text" placeholder="Поиск">
            <button type="submit" id="send"></button>
        </div>
        <div class="userListContainer" id="userListContainer">
            <div class="userCard"><a href="/chat?favouritechats=true"><h1>Избранные</h1></a></div>
        </div>
        <div class="chatBody">
            <div class="messages">
                {{range $key, $value := .}}
                {{if $value.Favourite}}
                {{if eq $value.Pos "r"}}
                <div class="messageRight">
                    <a>Ваше сообщение</a>
                    {{ $value.Text }}
                    <span>{{ $value.DateStr }}</span>
                    <span style="cursor:pointer;" onclick="downloadBlob('{{ $value.File }}', '{{ $value.FileName }}');">
                        {{ $value.FileName }}
                    </span>
                <div class="messageBottom">
                    <a href="/chat?deletefavourite={{ $value.MessageID }}"></a>
                </div>
                {{else}}
                <div class="messageLeft">
                    <a href="/chat?recipient={{ $value.UserName }}">{{ $value.UserName }}</a>
                    {{ $value.Text }}
                    <span>{{ $value.DateStr }}</span>
                    <span style="cursor:pointer;" onclick="downloadBlob('{{ $value.File }}', '{{ $value.FileName }}');">
                        {{ $value.FileName }}
                    </span>
                <div class="messageBottom">

```



```

<div class="userListContainerDARK" id="userListContainer">
    <div class="userCard"><a href="/chat?favouritechats=true"><h1>Избранные</h1></a></div>
    </div>
    <div class="chatBodyDARK">
        <div class="messages">
            {{ range $key, $value := .}}
            {{if $value.Favourite}}
            {{if eq $value.Pos "r"}}
            <div class="messageRightDARK">
                <a>Ваше сообщение</a>
                {{ $value.Text }}
                <span>{{ $value.DateStr }}</span>
                <span style="cursor:pointer;" onclick="downloadBlob('{{ $value.File }}', '{{ $value.FileName }}');">{{ $value.FileName }}</span>
            </div>
            <div class="messageBottom">
                <a href="/chat?deletefavourite={{ $value.MessageID }}"></a>
            </div>
        </div>
        {{else}}
        <div class="messageLeftDARK">
            <a href="/chat?recipient={{ $value.UserName }}">{{ $value.UserName }}</a>
            {{ $value.Text }}
            <span>{{ $value.DateStr }}</span>
            <span style="cursor:pointer;" onclick="downloadBlob('{{ $value.File }}', '{{ $value.FileName }}');">{{ $value.FileName }}</span>
        </div>
        <div class="messageBottom">
            <a href="/chat?deletefavourite={{ $value.MessageID }}"></a>
        </div>
        {{end}}
        {{else}}
        {{if eq $value.Pos "r"}}
        <div class="messageRightDARK">
            {{ $value.Text }}
            <span>{{ $value.DateStr }}</span>
            <span style="cursor:pointer;" onclick="downloadBlob('{{ $value.File }}', '{{ $value.FileName }}');">{{ $value.FileName }}</span>
        </div>
        <div class="messageBottom">
            <a href="/chat?delete={{ $value.MessageID }}"></a>
            <a href="/chat?favourite={{ $value.MessageID }}"></a>
        </div>
        </div>
        {{else}}
        <div class="messageLeftDARK">
            {{ $value.Text }}
            <span>{{ $value.DateStr }}</span>
            <span style="cursor:pointer;" onclick="downloadBlob('{{ $value.File }}', '{{ $value.FileName }}');">{{ $value.FileName }}</span>
        </div>
        <div class="messageBottom">
            <a href="/chat?delete={{ $value.MessageID }}"></a>
            <a href="/chat?favourite={{ $value.MessageID }}"></a>
        </div>
        {{end}}
        {{end}}
        {{end}}
    </div>
    <div class="sendPanelDARK">
        <form method="POST" enctype="multipart/form-data">
            <a></a>
            <input class="textInput" type="text" placeholder="Напишите что-нибудь..." name="message">

```



```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Регистрация нового аккаунта</title>
<link rel="stylesheet" href="style/style.css">
<link rel="stylesheet" href="style/reset.css">
<link rel="icon" href="/static/favicon.ico">
</head>
<body>
<div class="authContainer">
<a class="backButton" href="/authMain"></a>
<div class="authInnerContainer">
<div class="authLogo">
<h1>Регистрация</h1>
</div>

<form method="post" enctype="multipart/form-data">
<input type="text" id="login" name="username" placeholder="Логин" required>
<input type="text" id="email" name="email" placeholder="Email" required>

<input type="file" name="logo" required>
<input type="password" id="password" name="password" placeholder="Пароль" required>
<button class="authButton" type="submit">РЕГИСТРАЦИЯ</button>
</form>
</div>
<a class="lightMode" href="/registration?toggle=true"></a>
</div>

</body>
</html>
```

```
//web/regDARK.html
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Регистрация нового аккаунта</title>
<link rel="stylesheet" href="style/style.css">
<link rel="stylesheet" href="style/reset.css">
<link rel="icon" href="/static/favicon.ico">
</head>
<body class="bodyDark">
<div class="authContainer">
<a class="backButton" href="/authMain"></a>
<div class="authInnerContainerDARK">
<div class="authLogo">
<h1>Регистрация</h1>
</div>

<form method="post" enctype="multipart/form-data">
<input type="text" id="login" name="username" placeholder="Логин" required>
<input type="text" id="email" name="email" placeholder="Email" required>

<input type="file" name="logo" required>
<input type="password" id="password" name="password" placeholder="Пароль" required>
<button class="authButtonDARK" type="submit">РЕГИСТРАЦИЯ</button>
</form>
</div>
<a class="lightMode" href="/registration?toggle=true"></a>
</div>
```

```
</body>
</html>
```

```
//scriptBackup.sql
```

```

create table content
(
    contentID int auto_increment
        primary key,
    text longtext not null,
    file longblob not null,
    fileName text null
)
row_format = COMPRESSED;

create table user
(
    username varchar(30)         null,
    email    varchar(255)        not null,
    password binary(16)         not null,
    create_time timestamp default CURRENT_TIMESTAMP not null,
    userID   int auto_increment
        primary key,
    logo    longblob            null,
constraint user_pk
    unique (username),
constraint user_username_uindex
    unique (username)
)
row_format = COMPRESSED;

create table connects
(
    userA  int                  null,
    userB  int                  null,
    id     int auto_increment
        primary key,
    timeOpen timestamp default CURRENT_TIMESTAMP not null on update CURRENT_TIMESTAMP,
constraint connects_pk
    unique (userA, userB),
constraint connects_user(userID_fk
    foreign key (userA) references user (userID)
        on delete cascade,
constraint connects_user(userID_fk2
    foreign key (userB) references user (userID)
        on delete cascade
)
row_format = COMPRESSED;

create table message
(
    messageID int auto_increment
        primary key,
    fromID    int                 not null,
    toID     int                 not null,
    date     datetime default CURRENT_TIMESTAMP not null,
    contentID int                 not null,
constraint message_date_uindex
    unique (date),
constraint content
    foreign key (contentID) references content (contentID)
        on update cascade on delete cascade,
constraint `from`
    foreign key (fromID) references user (userid),
constraint `to`
    foreign key (toID) references user (userid)
)
row_format = COMPRESSED;

create table favouritemessage
(
    id      int auto_increment
        primary key,

```

```

fromID      int not null,
messageID   int not null,
favouriteuserid int not null,
constraint favouriteMessage_message_messageID_fk
    foreign key (messageID) references message (messageID)
        on update cascade on delete cascade,
constraint favouriteMessage_user(userID)_fk
    foreign key (fromID) references user (userID),
constraint favouriteMessage_user(userID)_fk1
    foreign key (favouriteuserid) references user (userID)
);

create definer = mysql@`%` trigger before_delete_message
before delete
on message
for each row
BEGIN
    delete from content where content.contentID=OLD.contentID;
END;

create definer = mysql@`%` trigger before_delete_user
before delete
on user
for each row
BEGIN
    delete from message where message.fromID=OLD.userID;
END;

//additional/additionalFunctions.go

package additional

import (
"chat/variables"
"fmt"
"github.com/golang-jwt/jwt"
"github.com/labstack/echo/v4"
"net/http"
)

// fone for validate JWT token
func ValidateToken(c echo.Context) string {
//retrieve cookie value
cookie, _ := c.Cookie("JWTToken")
if cookie == nil {
    return "Token error"
}
//try parse token
token, _ := jwt.Parse(cookie.Value, func(token *jwt.Token) (interface{}, error) {
    if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
        return nil, fmt.Errorf("There was an error in parsing")
    }
    return variables.Secret, nil
})
//if token not valid
if token == nil {
    return "Token error"
}
//try parse claim from token
claims, ok := token.Claims.(jwt.MapClaims)
//if token claims not valid
if !ok {
    return "Token error"
}
//return username string from claim
return claims["name"].(string)
}

```

```

// func retrieve Theme from cookie
func CheckTheme(c echo.Context) string {
    //retrieve cookie value
    theme, err := c.Cookie("theme")
    if err != nil {
        //if cookie not found
        return "ERR"
    }
    //return theme cookie value
    return theme.Value
}

// func check if exist username in DB
func CheckUser(username string) string {
    var ID string
    //query for DB
    result, _ := variables.Db.Query("select userID from user where username = ? or userID = ?", username, username)
    if result == nil {
        //if username not found
        return ""
    }
    for result.Next() {
        //scan userID
        err := result.Scan(&ID)
        if err != nil {
            fmt.Println(err)
        }
    }
    //return username MessageID
    return ID
}

// function for Toggle theme value on cookie
func Toggle(c echo.Context) string {
    //make cookie
    theme, err := c.Cookie("theme")
    Cookie := &http.Cookie{}
    Cookie.Name = "theme"
    Cookie.SameSite = 3
    Cookie.HttpOnly = true
    Cookie.Secure = true
    //if theme cookie not found
    if err != nil {
        Cookie.Value = "light"
        c.SetCookie(Cookie)
        return Cookie.Value
    }
    //if last theme is dark
    if theme.Value == "dark" {
        Cookie.Value = "light"
    }
    //if last theme is light
    if theme.Value == "light" {
        Cookie.Value = "dark"
    }
    //set cookie
    c.SetCookie(Cookie)
    //return current theme
    return Cookie.Value
}

// handler for HTTP errors
func CustomHTTPErrorHandler(err error, c echo.Context) {
    //retrieve code error
    code := http.StatusInternalServerError
    //if he, ok := err.(*echo.HTTPError); ok {

```

```

//      code = he.Code
//}
//c.Logger().Error(err)
theme := CheckTheme(c)
errorPage := ""
//make error page lin
if theme == "dark" {
    errorPage = fmt.Sprintf("web/errors/%dDARK.html", code)
} else {
    errorPage = fmt.Sprintf("web/errors/%d.html", code)
}

if err := c.File(errorPage); err != nil {
    c.Logger().Error(err)
}
}

//additional/chatAdditional.go

package additional

import (
"bytes"
"chat/variables"
"fmt"
"github.com/labstack/echo/v4"
"net/http"
"sort"
"time"
)

// func analyze auth status and return response view
func AuthRegView(c echo.Context, page string) error {
var theme = ""
toggleParam := c.QueryParam("toggle")
if toggleParam == "true" {
    theme = Toggle(c)
}
if ValidateToken(c) == "Token error" {
    if theme == "" {
        theme = CheckTheme(c)
    }
    if theme == "dark" {
        return c.Render(http.StatusOK, page+"DARK.html", nil)
    } else {
        return c.Render(http.StatusOK, page+".html", nil)
    }
} else {
    return c.Redirect(http.StatusMovedPermanently, "/chat")
}
}

// func for chat views
func ParseMessages(c echo.Context, usernameID string, recipientID string) []variables.Message {
_, err := variables.Db.Exec("insert into connects(userA, userB) values (?,?)", usernameID, recipientID)
if err != nil {
    fmt.Println(err)
}

var messages []variables.Message
result, err := variables.Db.Query("select messageID,text,file,fileName,cast(date as char) from message join content on content.contentID = message.contentID where message.fromID = ? and message.toID = ?", usernameID, recipientID)
if err != nil {
    fmt.Println(err)
}
for result.Next() {

```

```

var text, dateStr, fileName string
var byteFile []byte
var id int
err = result.Scan(&id, &text, &byteFile, &fileName, &dateStr)
date, err := time.Parse(variables.TimeFormat, dateStr)
if err != nil {
    fmt.Println(err)
}
var elem variables.Message
if bytes.Equal(byteFile, []byte{48}) == true {
    elem = variables.Message{Favourite: false, MessageID: id, Text: text, File: byteFile, FileName: "", Date: date, DateStr:
date.Format("2006-01-02 15:04:05"), Pos: "r"}
} else {
    elem = variables.Message{Favourite: false, MessageID: id, Text: text, File: byteFile, FileName: fileName, Date: date,
DateStr: date.Format("2006-01-02 15:04:05"), Pos: "r"}
}
messages = append(messages, elem)
}

result, err = variables.Db.Query("select messageID,text,file,fileName,cast(date as char) from message join content on content.contentID =
message.contentID where message.fromID = ? and message.toID = ?", recipientID, usernameID)
if err != nil {
    fmt.Println(err)
}

for result.Next() {
    var text, dateStr, fileName string
    var byteFile []byte
    var id int
    err = result.Scan(&id, &text, &byteFile, &fileName, &dateStr)
    date, err := time.Parse(variables.TimeFormat, dateStr)
    if err != nil {
        fmt.Println(err)
    }
    var elem variables.Message
    if bytes.Equal(byteFile, []byte{48}) == true {
        elem = variables.Message{Favourite: false, MessageID: id, Text: text, File: byteFile, FileName: "", Date: date, DateStr:
date.Format("2006-01-02 15:04:05"), Pos: "l"}
    } else {
        elem = variables.Message{Favourite: false, MessageID: id, Text: text, File: byteFile, FileName: fileName, Date: date,
DateStr: date.Format("2006-01-02 15:04:05"), Pos: "l"}
    }
    messages = append(messages, elem)
}
sort.Slice(messages, func(i, j int) bool { return messages[i].Date.Before(messages[j].Date) })
lastMessageCookie := &http.Cookie{
    Name: "lastMessage"
}
if len(messages) > 0 {
    lastMessageCookie.Value = messages[len(messages)-1:][0].DateStr
} else {
    now := time.Now()
    lastMessageCookie.Value = now.Format("2006-01-02 15:04:05")
}
lastMessageCookie.SameSite = 3
lastMessageCookie.HttpOnly = true
lastMessageCookie.Secure = true
c.SetCookie(lastMessageCookie)
return messages
}
func UpdateTimeVisit(userA string, userB string) {
now := time.Now()
lastVisitTime := now.Format("2006-01-02 15:04:05")
_, err := variables.Db.Exec("update connects set timeOpen = ? where userA = ? and userB = ?",
    lastVisitTime, userA, userB)
if err != nil {
    fmt.Println(err)
}
}
}

```

```

func SearchRecipient(c echo.Context) string {
    var recipientID string
    recipient := c.QueryParam("recipient")
    recipientID = CheckUser(recipient)
    if recipientID != "" {
        newRecipientCookie := &http.Cookie{ }
        newRecipientCookie.Name = "recipient"
        newRecipientCookie.Value = recipientID
        newRecipientCookie.SameSite = 3
        newRecipientCookie.HttpOnly = true
        newRecipientCookie.Secure = true
        c.SetCookie(newRecipientCookie)
    } else {
        var recipientCookie, err = c.Cookie("recipient")
        if err != nil {
            fmt.Println(err)
        }
        if recipientCookie != nil {
            recipientID = recipientCookie.Value
        }
    }
    return recipientID
}
func DeleteMessage(usernameID string, deleteID string) {
    _, err := variables.Db.Exec("delete from message where messageID = ? and "+
        "(message.fromID = ? or message.toID = ?)", deleteID, usernameID, usernameID)
    if err != nil {
        fmt.Println(err)
    }
}

func DeleteFavouriteMessage(usernameID string, deleteID string) {
    _, err := variables.Db.Exec("delete from favouritemessage where id = ? and favouriteuserid = ?",
        deleteID, usernameID)
    if err != nil {
        fmt.Println(err)
    }
}

func FavouriteMessage(usernameID string, favouriteID string) {
    result, err := variables.Db.Query("select fromID from message where(message.fromID = ? or message.toID = ?) and message.messageID = ?",
        usernameID, usernameID, favouriteID)
    if err != nil {
        fmt.Println(err)
    }
    var fromID string
    for result.Next() {
        err = result.Scan(&fromID)
    }
    _, err = variables.Db.Exec("insert into favouritemessage(fromID, messageID, favouriteuserid) values (?, ?, ?)", fromID, favouriteID,
        usernameID)
    if err != nil {
        fmt.Println(err)
    }
}

func DownloadFavourite(c echo.Context, favouriteID string) []variables.FavouriteMessages {
    var favouriteArray []variables.FavouriteMessages
    result, err := variables.Db.Query("select favouritemessage.id,username,favouritemessage.fromID,text,"+
        "cast(date as char),file,fileName from favouritemessage join message on "+
        "favouritemessage.messageID = message.messageID join chat.content on message.contentID = "+
        "content.contentID join user on favouritemessage.fromID = user.userID where favouritemessage.favouriteuserid "+
        "= ? and favouritemessage.favouriteuserid != favouritemessage.fromID", favouriteID)
    if err != nil {
        fmt.Println(err)
    }
    for result.Next() {

```

```

var username, text, dateStr, fileName string
var byteFile []byte
var fromid, id int
err = result.Scan(&id, &username, &fromid, &text, &dateStr, &byteFile, &fileName)
date, err := time.Parse(variables.TimeFormat, dateStr)
if err != nil {
    fmt.Println(err)
}
var elem variables.FavouriteMessages
if bytes.Equal(byteFile, []byte{48}) == true {
    elem = variables.FavouriteMessages{MessageID: id, Favourite: true, UserName: username, FromID: fromid,
        Text: text, File: byteFile, FileName: "", Date: date,
        DateStr: date.Format("2006-01-02 15:04:05"), Pos: "l"}
} else {
    elem = variables.FavouriteMessages{MessageID: id, Favourite: true, UserName: username, FromID: fromid,
        Text: text, File: byteFile, FileName: fileName, Date: date,
        DateStr: date.Format("2006-01-02 15:04:05"), Pos: "l"}
}
favouriteArray = append(favouriteArray, elem)
}

result, err = variables.Db.Query("select favouritemessage.id,username,favouritemessage.fromID,text"+
    ",cast(date as char),file,fileName from favouritemessage join message "+
    "on favouritemessage.messageID = message.messageID join chat.content on "+
    "message.contentID = content.contentID join user on favouritemessage.fromID "+
    "= user.userID where favouritemessage.favouriteuserid = ? and favouritemessage.favouriteuserid "+
    "= favouritemessage.fromID", favouriteID)
if err != nil {
    fmt.Println(err)
}

for result.Next() {
    var username, text, dateStr, fileName string
    var byteFile []byte
    var fromid, id int
    err = result.Scan(&id, &username, &fromid, &text, &dateStr, &byteFile, &fileName)
    if err != nil {
        fmt.Println(err)
    }
    date, err := time.Parse(variables.TimeFormat, dateStr)
    if err != nil {
        fmt.Println(err)
    }
    var elem variables.FavouriteMessages
    if bytes.Equal(byteFile, []byte{48}) == true {
        elem = variables.FavouriteMessages{MessageID: id, Favourite: true, UserName: username,
            FromID: fromid, Text: text, File: byteFile, FileName: "", Date: date,
            DateStr: date.Format("2006-01-02 15:04:05"), Pos: "r"}
    } else {
        elem = variables.FavouriteMessages{MessageID: id, Favourite: true, UserName: username,
            FromID: fromid, Text: text, File: byteFile, FileName: fileName, Date: date,
            DateStr: date.Format("2006-01-02 15:04:05"), Pos: "r"}
    }
    favouriteArray = append(favouriteArray, elem)
}
sort.Slice(favouriteArray, func(i, j int) bool { return favouriteArray[i].Date.Before(favouriteArray[j].Date) })
return favouriteArray
}

//handlersPackage/ajaxHandlers.go

package handlersPackage

import (
    "bytes"
    "chat/additional"
    "chat/variables"

```

```

"database/sql"
"encoding/base64"
"fmt"
"github.com/labstack/echo/v4"
"github.com/labstack/gommon/log"
"net/http"
"sort"
"time"
)

func LoadNewMessageCounter(c echo.Context) error {
var newDate time.Time
var messages []variables.MessagesCounter
username := additional.ValidateToken(c)
if username != "Token error" {
    usernameID := additional.CheckUser(username)
    result, err := variables.Db.Query("select id,userA,userB,timeOpen from connects where userA = ?", usernameID)
    if err != nil {
        fmt.Println(err)
    }
    for result.Next() {
        var dateStr string
        var id, userA, userB int
        err = result.Scan(&id, &userA, &userB, &dateStr)
        date, err := time.Parse(variables.TimeFormat, dateStr)
        if err != nil {
            fmt.Println(err)
        }
        resMessage, err := variables.Db.Query("select cast(date as char) from message where message.fromID = ? and message.toID = ? and date > ?", userB, userA, newDate.Format("2006-01-02 15:04:05"))
        var counter = 0
        for resMessage.Next() {
            var strDateMessage string
            err = resMessage.Scan(&strDateMessage)
            dateMessage, _ := time.Parse(variables.TimeFormat, strDateMessage)
            if dateMessage.After(date) {
                counter++
                date = dateMessage
            }
        }
        if counter != 0 {
            var elem variables.MessagesCounter
            elem = variables.MessagesCounter{UserID: userB, MCounter: counter}
            messages = append(messages, elem)
        }
    }
}

return c.JSON(http.StatusOK, messages)
}

func LoaderUser(c echo.Context) error {
var result *sql.Rows
var err error
username := additional.ValidateToken(c)
usernameID := additional.CheckUser(username)
if usernameID == "" {
    return c.Redirect(http.StatusMovedPermanently, "/")
} else {
    result, err = variables.Db.Query("select user.UserID,username,logo from connects join user on connects.userB = user.userID and connects.userA = ? limit 10", usernameID)
    if err != nil {
        fmt.Println(err)
    }
}

```

```

}

if c.FormValue("search") != "" {
    // retrieve search string
    queryString := c.FormValue("search")

    //ВНИМАЕНИЕ

    result, err = variables.Db.Query("select user.UserID,username,logo from user where username like '%" + queryString + "%' limit
10 ")

    //ВНИМАНИЕ
}

if err != nil {
    fmt.Println(err)
}

var users []variables.User
for result.Next() {
    var uid int
    var username string
    var logo []byte
    err = result.Scan(&uid, &username, &logo)
    strLogo := base64.StdEncoding.EncodeToString(logo)

    users = append(users, variables.User{Id: uid, Username: username, Logo: strLogo})
}
if err != nil {
    log.Error(err)
}
return c.JSON(http.StatusOK, users)
}

func GetMessages(c echo.Context) error {
lastMessage, err := c.Cookie("lastMessage")
var newDate time.Time
var newDateForCookie time.Time
if err != nil {
    newDateForCookie, err = time.Parse(variables.TimeFormat, "2000-01-01 00:00:00")
    if err != nil {
        fmt.Println(err)
    }
} else {
    if lastMessage.Value != "" {
        newDateForCookie, err = time.Parse(variables.TimeFormat, lastMessage.Value)
        if err != nil {
            fmt.Println(err)
        }
    }
}
var messages []variables.Message
newDate = newDateForCookie
username := additional.ValidateToken(c)
if username != "Token error" {
    recipientID := additional.SearchRecipient(c)
    userID := additional.CheckUser(username)
    result, err := variables.Db.Query("select messageID,text,file,fileName,cast(date as char) from message "+
        "join content on content.contentID = message.contentID join user on user.userID = message.fromID "+
        "where message.fromID = ? and message.toID = ? and date >?", userID, recipientID,
        newDate.Format("2006-01-02 15:04:05"))
    if err != nil {
        fmt.Println(err)
    }
    for result.Next() {
        var text, dateStr, fileName string
        var byteFile []byte
        var id int
        err = result.Scan(&id, &text, &byteFile, &fileName, &dateStr)
    }
}
}

```

```

date, err := time.Parse(variables.TimeFormat, dateStr)
if err != nil {
    fmt.Println(err)
}
if newDate.After(date) {

} else {
    newDateForCookie = date
    var elem variables.Message
    if bytes.Equal(byteFile, []byte{48}) == true {
        elem = variables.Message{Favourite: false, MessageID: id, Text: text, File: byteFile, FileName:
"", Date: date, DateStr: date.Format("2006-01-02 15:04:05"), Pos: "r"}
    } else {
        elem = variables.Message{Favourite: false, MessageID: id, Text: text, File: byteFile, FileName:
fileName, Date: date, DateStr: date.Format("2006-01-02 15:04:05"), Pos: "r"}
    }
    messages = append(messages, elem)
}
}
result = nil
result, err = variables.Db.Query("select messageID,text,file,fileName,cast(date as char) from message join content on
content.contentID = message.contentID join user on user.userID = message.fromID where message.fromID = ? and message.toID = ? and
date >?", recipientID, userID, newDate.Format("2006-01-02 15:04:05"))
if err != nil {
    fmt.Println(err)
}

for result.Next() {
    var text, dateStr, fileName string
    var byteFile []byte
    var id int
    err = result.Scan(&id, &text, &byteFile, &fileName, &dateStr)
    date, err := time.Parse(variables.TimeFormat, dateStr)
    if err != nil {
        fmt.Println(err)
    }
    if newDate.After(date) {

} else {
    newDateForCookie = date
    var elem variables.Message
    if bytes.Equal(byteFile, []byte{48}) == true {
        elem = variables.Message{Favourite: false, MessageID: id, Text: text, File: byteFile, FileName:
"", Date: date, DateStr: date.Format("2006-01-02 15:04:05"), Pos: "l"}
    } else {
        elem = variables.Message{Favourite: false, MessageID: id, Text: text, File: byteFile, FileName:
fileName, Date: date, DateStr: date.Format("2006-01-02 15:04:05"), Pos: "l"}
    }
    messages = append(messages, elem)
}
}
sort.Slice(messages, func(i, j int) bool { return messages[i].Date.Before(messages[j].Date) })
lastMessageCookie := &http.Cookie{}
lastMessageCookie.Name = "lastMessage"
lastMessageCookie.Value = newDateForCookie.Format("2006-01-02 15:04:05")
lastMessageCookie.SameSite = 3
lastMessageCookie.HttpOnly = true
lastMessageCookie.Secure = true
c.SetCookie(lastMessageCookie)
}

return c.JSON(http.StatusOK, messages)
}

//handlersPackage/authentification.go

package handlersPackage

```

```

import (
    "chat/additional"
    "chat/variables"
    "crypto/md5"
    "github.com/golang-jwt/jwt/v5"
    "github.com/labstack/echo/v4"
    "net/http"
    "time"
)

func Auth(c echo.Context) error {
    return additional.AuthRegView(c, "auth")
}

func AuthPOST(c echo.Context) error {
    if additional.ValidateToken(c) == "Token error" {
        username := c.FormValue("username")
        password := c.FormValue("password")

        hash := md5.Sum([]byte(password + variables.AdditionalString))
        result, err := variables.Db.Query("select password from user where username = ?", username)
        if err != nil {
            return err
        }

        for result.Next() {
            var u []byte
            err = result.Scan(&u)
            var fixedSizePassword [16]byte
            copy(fixedSizePassword[:], u)

            // Throws unauthorized error
            if fixedSizePassword == hash {

                // Set custom claims
                claims := &variables.JwtCustomClaims{
                    Name: username,
                    RegisteredClaims: jwt.RegisteredClaims{
                        ExpiresAt: jwt.NewNumericDate(time.Now().Add(time.Hour * 72)),
                    },
                }

                // Create token with claims
                token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)

                // Generate encoded token and send it as response.
                t, err := token.SignedString(variables.Secret)
                if err != nil {
                    return err
                }
                JWTCookie := &http.Cookie{}
                JWTCookie.Name = "JWTToken"
                JWTCookie.Expires = time.Now().Add(time.Hour * 72)
                JWTCookie.Value = t
                JWTCookie.SameSite = 3
                JWTCookie.HttpOnly = true
                JWTCookie.Secure = true
                c.SetCookie(JWTCookie)

            }
        }
    }
    return c.Redirect(http.StatusMovedPermanently, "/chat")
}

//handlersPackage/chatPage.go

```

```

package handlersPackage

import (
    "chat/additional"
    "chat/variables"
    "fmt"
    "github.com/labstack/echo/v4"
    "github.com/labstack/gommon/log"
    "io"
    "net/http"
)

func LogoutFunc(c echo.Context) error {
    Cookie := &http.Cookie{}
    Cookie.Name = "JWTToken"
    Cookie.Value = ""
    Cookie.SameSite = 3
    Cookie.HttpOnly = true
    Cookie.Secure = true
    c.SetCookie(Cookie)
    Cookie.Name = "recipient"
    c.SetCookie(Cookie)
    Cookie.Name = "lastMessage"
    c.SetCookie(Cookie)
    return c.Redirect(http.StatusMovedPermanently, "/")
}

func FavouritePage(c echo.Context) {
    recipientCookie := &http.Cookie{}
    recipientCookie.Name = "recipient"
    recipientCookie.Value = ""
    recipientCookie.SameSite = 3
    recipientCookie.HttpOnly = true
    recipientCookie.Secure = true
    c.SetCookie(recipientCookie)
}

func ChatGET(c echo.Context) error {
    logoutParam := c.QueryParam("logout")
    if logoutParam != "" {
        return LogoutFunc(c)
    }
    var theme = ""
    toggleParam := c.QueryParam("toggle")
    if toggleParam == "true" {
        theme = additional.Toggle(c)
    }
    if theme == "" {
        theme = additional.CheckTheme(c)
    }
    username := additional.ValidateToken(c)
    if username != "Token error" {
        userID := additional.CheckUser(username)
        if userID == "" {
            return c.Redirect(http.StatusMovedPermanently, "/")
        }

        //search recipient MessageID from Cookie or GET request
        recipientID := additional.SearchRecipient(c)
        //if user press on favourite messages
        favouriteChatsParam := c.QueryParam("favouritechats")
        if favouriteChatsParam != "" {
            FavouritePage(c)
            recipientID = ""
        }
        //if user press on delete message button
        deleteFavouriteID := c.QueryParam("deletefavourite")
    }
}

```

```

        if deleteFavouriteID != "" {
            additional.DeleteFavouriteMessage(usernameID, deleteFavouriteID)
        }
        deleteID := c.QueryParam("delete")
        if deleteID != "" {
            additional.DeleteMessage(usernameID, deleteID)
        }
        //if user press on favourite message button
        favouriteID := c.QueryParam("favourite")
        if favouriteID != "" {
            additional.FavouriteMessage(usernameID, favouriteID)
        }

        if recipientID != "" && recipientID != usernameID {
            additional.UpdateTimeVisit(usernameID, recipientID)
            messages := additional.ParseMessages(c, usernameID, recipientID)
            if theme == "dark" {
                return c.Render(http.StatusOK, "chatDARK.html", messages)
            } else {
                return c.Render(http.StatusOK, "chat.html", messages)
            }
        } else {
            favouriteMessagesArray := additional.DownloadFavourite(c, usernameID)
            if theme == "dark" {
                return c.Render(http.StatusOK, "chatDARK.html", favouriteMessagesArray)
            } else {
                return c.Render(http.StatusOK, "chat.html", favouriteMessagesArray)
            }
        }

    } else {
        return c.Redirect(http.StatusMovedPermanently, "/")
    }
}

func ChatPOST(c echo.Context) error {
    username := additional.ValidateToken(c)
    userID := additional.CheckUser(username)
    if userID == "" {
        return c.Redirect(http.StatusMovedPermanently, "/")
    }
    if c.FormValue("search") != "" {
        return LoaderUser(c)
    }
    recipientID := additional.SearchRecipient(c)
    //read text from form
    text := c.FormValue("message")
    // read file
    file, err := c.FormFile("file")
    if err != nil {
        fmt.Println("no file")
    }
    var dst = []byte{48}
    var fName string
    if file == nil {
        //if file not appended to message
        fName = ""
    } else {
        src, err := file.Open()
        if err != nil {
            return err
        }
        defer src.Close()

        dst, _ = io.ReadAll(src)
        fName = "Вложение: " + file.Filename
    }
}

```

```

//insert content of new message to DB
_, err = variables.Db.Exec("insert into chat.content (text, file, fileName) values (?, ?, ?)", text, dst, fName)
if err != nil {
    log.Error(err)
}
//insert new message to DB
_, err = variables.Db.Exec("insert into message (fromID, toID, contentID) values (?, ?, LAST_INSERT_ID());", usernameID, recipientID)
if err != nil {
    log.Error(err)
}
return c.Redirect(http.StatusMovedPermanently, "/chat")
}

//handlersPackage/mainPage.go

package handlersPackage

import (
    "chat/additional"
    "github.com/labstack/echo/v4"
    "net/http"
)

func MainPage(c echo.Context) error {
    //make theme cookie
    Cookie := &http.Cookie{}
    Cookie.Name = "theme"
    Cookie.Value = "light" // light theme default
    Cookie.SameSite = 3
    Cookie.HttpOnly = true
    Cookie.Secure = true
    c.SetCookie(Cookie)
    //if authorize
    if additional.ValidateToken(c) == "Token error" {
        return c.Redirect(http.StatusMovedPermanently, "/authMain")
    } else {
        return c.Redirect(http.StatusMovedPermanently, "/chat")
    }
}
func AuthMain(c echo.Context) error {
    return additional.AuthRegView(c, "authMain")
}

//handlersPackage/registration.go

package handlersPackage

import (
    "bytes"
    "chat/additional"
    "chat/variables"
    "crypto/md5"
    "github.com/disintegration/imaging"
    "github.com/golang-jwt/jwt/v5"
    "github.com/labstack/echo/v4"
    "image/jpeg"
    "net/http"
    "time"
)

func RegGET(c echo.Context) error {
    return additional.AuthRegView(c, "reg")
}

func RegPOST(c echo.Context) error {
    username := c.FormValue("username")

```

```

result, err := variables.Db.Query("select username from user where username = ?", username)
if err != nil {
    return err
}
if result.Next() != false {
    return err
}
email := c.FormValue("email")
resultEmail, err := variables.Db.Query("select email from user where email = ?", email)
if err != nil {
    return err
}
if resultEmail.Next() != false {
    return err
}
password := c.FormValue("password")
logo, err := c.FormFile("logo")
if err != nil {
    return err
}
src, err := logo.Open()
if err != nil {
    return err
}
defer src.Close()
var dst []byte
img, _ := jpeg.Decode(src)

compressedImage := imaging.Resize(img, 100, 100, imaging.Lanczos)
buf := new(bytes.Buffer)
if err != nil {
    return err
}
err = jpeg.Encode(buf, compressedImage, nil)
dst = buf.Bytes()
if err != nil {
    return err
}
hashPassword := md5.Sum([]byte(password + variables.AdditionalString))
_, err = variables.Db.Exec("insert into user (username, email, password, logo) values (?, ?, ?, ?)",
    username, email, string(hashPassword[:]), dst)
if err != nil {
    return err
}

// Set custom claims
claims := &variables.JwtCustomClaims{
    Name: username,
    RegisteredClaims: jwt.RegisteredClaims{
        ExpiresAt: jwt.NewNumericDate(time.Now().Add(time.Hour * 72)),
    },
}

// Create token with claims
token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)

// Generate encoded token and send it as response.
t, err := token.SignedString(variables.Secret)
if err != nil {
    return err
}
JWTCookie := &http.Cookie{}
JWTCookie.Name = "JWTToken"
JWTCookie.Expires = time.Now().Add(time.Hour * 72)
JWTCookie.Value = t
JWTCookie.SameSite = 3
JWTCookie.HttpOnly = true
JWTCookie.Secure = true

```

```

c.SetCookie(JWTCookie)

return c.Redirect(http.StatusMovedPermanently, "/chat")
}

//logger/logger.go

package logger

import (
"fmt"
"os"
"strings"
"time"

"github.com/rs/zerolog"
)

type MyLogger struct {
zerolog.Logger
}

var Logger MyLogger

func NewLogger() MyLogger {
// create output configuration
output := zerolog.ConsoleWriter{Out: os.Stdout, TimeFormat: time.RFC3339}

// Format level: fatal, error, debug, info, warn
output.FormatLevel = func(i interface{}) string {
    return strings.ToUpper(fmt.Sprintf("| % -6s|", i))
}
output.FormatFieldName = func(i interface{}) string {
    return fmt.Sprintf("%s:", i)
}
output.FormatFieldValue = func(i interface{}) string {
    return fmt.Sprintf("%s", i)
}

// format error
output.FormatErrFieldName = func(i interface{}) string {
    return fmt.Sprintf("%s: ", i)
}

zerolog := zerolog.New(output).With().Caller().Timestamp().Logger()
Logger = MyLogger{zerolog}
return Logger
}

func (l *MyLogger) LogInfo() *zerolog.Event {
return l.Logger.Info()
}

func (l *MyLogger) LogError() *zerolog.Event {
return l.Logger.Error()
}

func (l *MyLogger) LogDebug() *zerolog.Event {
return l.Logger.Debug()
}

func (l *MyLogger) LogWarn() *zerolog.Event {
return l.Logger.Warn()
}

func (l *MyLogger) LogFatal() *zerolog.Event {
return l.Logger.Fatal()
}

```

```

}

//logger/loggetMiddlerware.go

package logger

import (
"github.com/labstack/echo/v4"
)

func LoggingMiddleware(next echo.HandlerFunc) echo.HandlerFunc {
return func(c echo.Context) error {
    // log the request
    Logger.LogInfo().Fields(map[string]interface{}{
        "method": c.Request().Method,
        "uri":   c.Request().URL.Path,
        "query": c.Request().URL.RawQuery,
    }).Msg("Request")

    // call the next middleware/handler
    err := next(c)
    if err != nil {
        Logger.LogError().Fields(map[string]interface{}{
            "error": err.Error(),
        }).Msg("Response")
        return err
    }

    return nil
}
}

```

```

//variables/structs.go

package variables

import (
"github.com/golang-jwt/jwt/v5"
"time"
)

// struct of newMessageCounter
type MessagesCounter struct {
MCounter int
UserID  int
}

// struct of Message
type Message struct {
Text  string
File  []byte
FileName string
Date  time.Time
DateStr string
Pos   string
MessageID int
Favourite bool
}

// struct of favourite Message
type FavouriteMessages struct {
Text  string
File  []byte
FileName string
Date  time.Time
DateStr string
}
```

```

Pos    string
FromID int
MessageID int
UserName string
Favourite bool
}

// struct of user connects
type User struct {
Id    int
Username string
Logo   string
}

// structure of JWT claim(contains username)
type JwtCustomClaims struct {
Name string `json:"name"`
jwt.RegisteredClaims
}

//var.go

package variables

import "database/sql"

// const variables
var TimeFormat = "2006-01-02 15:04:05"
var Db *sql.DB

// Secret phrase for encryption
var Secret = []byte("secret")
var AdditionalString = "additionalS 97treng.6123"

//go.mod

module chat

go 1.21.5

require (
github.com/disintegration/imaging v1.6.2
github.com/go-sql-driver/mysql v1.8.1
github.com/golang-jwt/jwt/v5 v5.2.1
github.com/labstack/echo-jwt/v4 v4.2.0
github.com/labstack/echo/v4 v4.11.4
github.com/labstack/gommon v0.4.2
github.com/rs/zerolog v1.33.0
)

require (
filippo.io/edwards25519 v1.1.0 // indirect
github.com/golang-jwt/jwt v3.2.2+incompatible // indirect
github.com/matttn/go-colorable v0.1.13 // indirect
github.com/matttn/go-isatty v0.0.20 // indirect
github.com/valyala/bytebufferpool v1.0.0 // indirect
github.com/valyala/fasttemplate v1.2.2 // indirect
golang.org/x/crypto v0.17.0 // indirect
golang.org/x/image v0.14.0 // indirect
golang.org/x/net v0.19.0 // indirect
golang.org/x/sys v0.15.0 // indirect
golang.org/x/text v0.14.0 // indirect
golang.org/x/time v0.5.0 // indirect
)

```

ПРИЛОЖЕНИЕ Б

Графическая часть

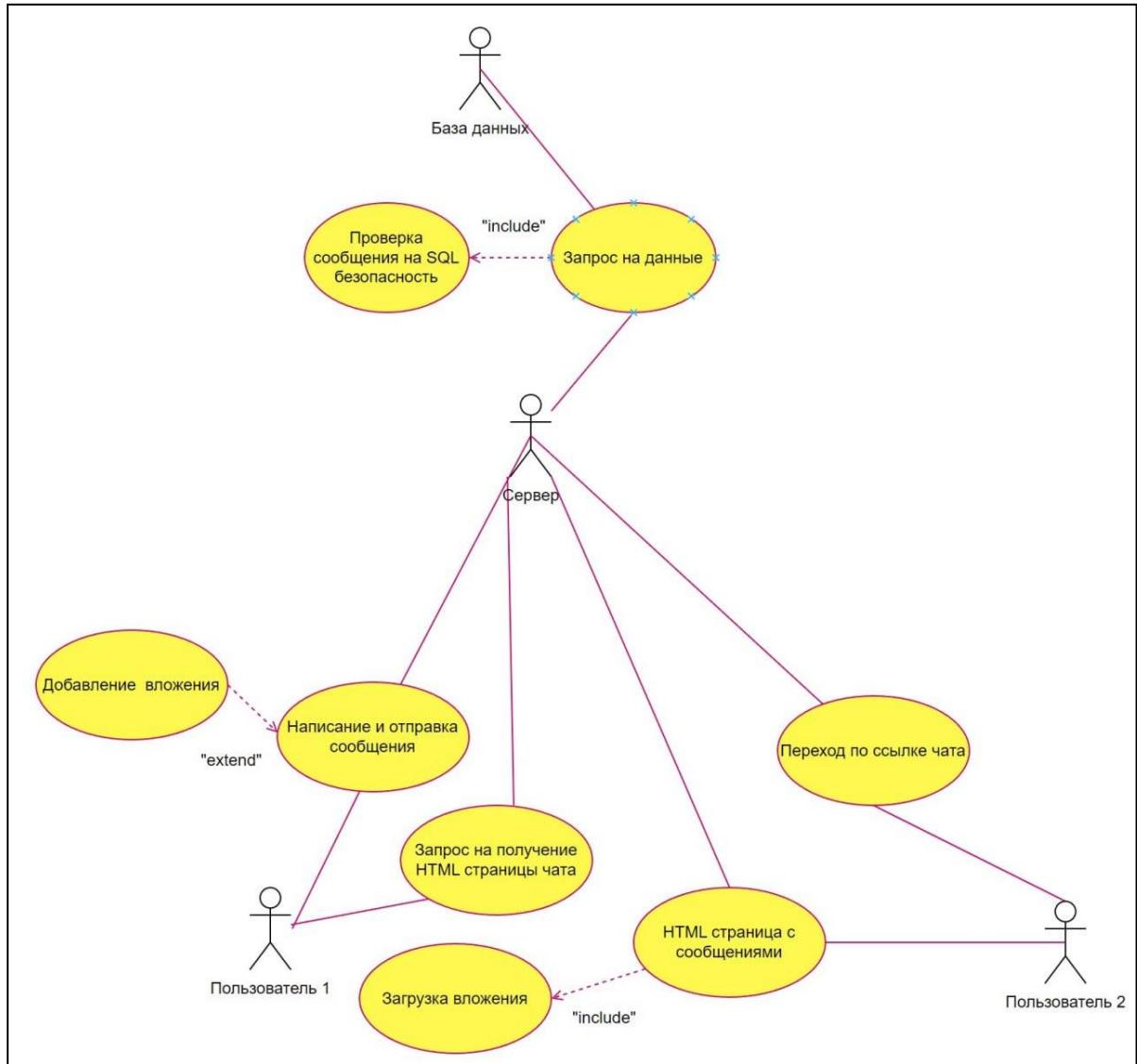


Рисунок Б1 – Диаграмма прецедентов

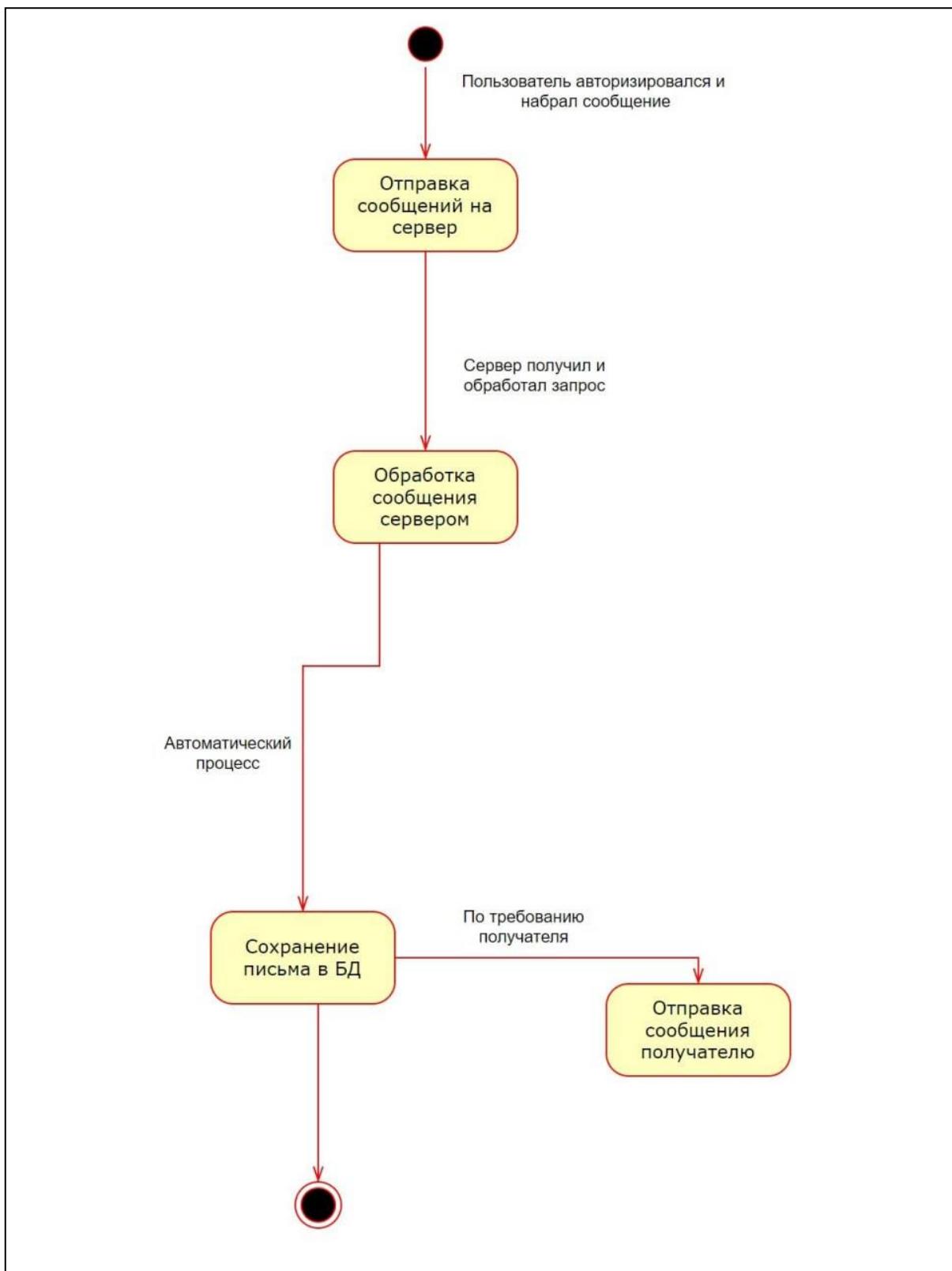


Рисунок Б2 – Диаграмма состояний

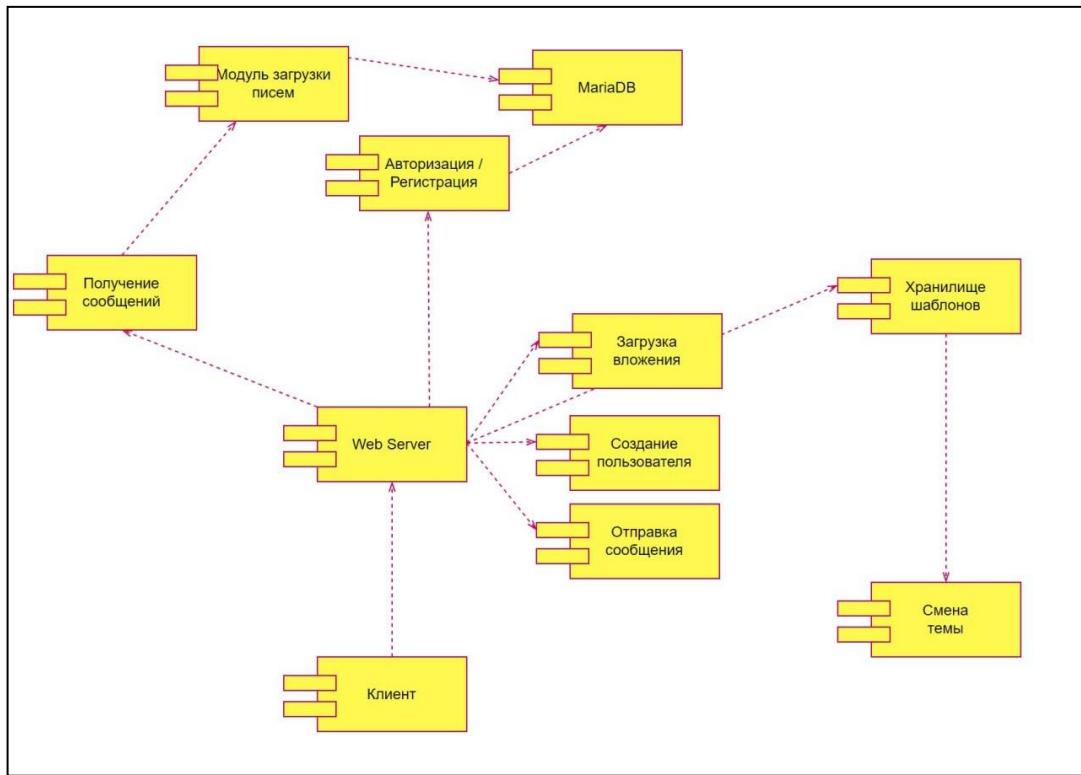


Рисунок Б3 – Диаграмма компонентов

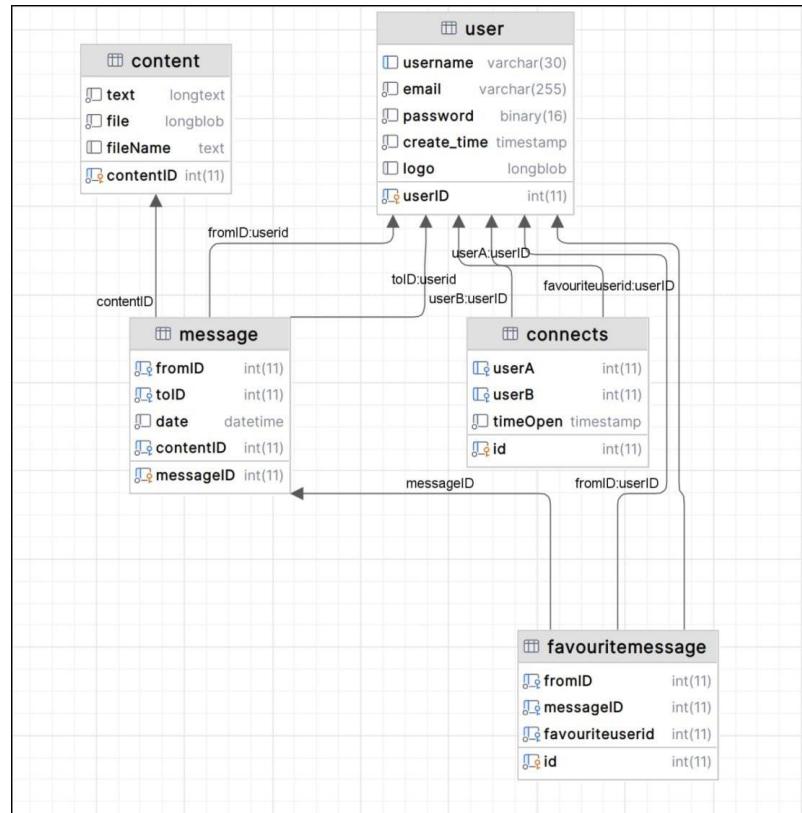


Рисунок Б4 – Схема базы данных

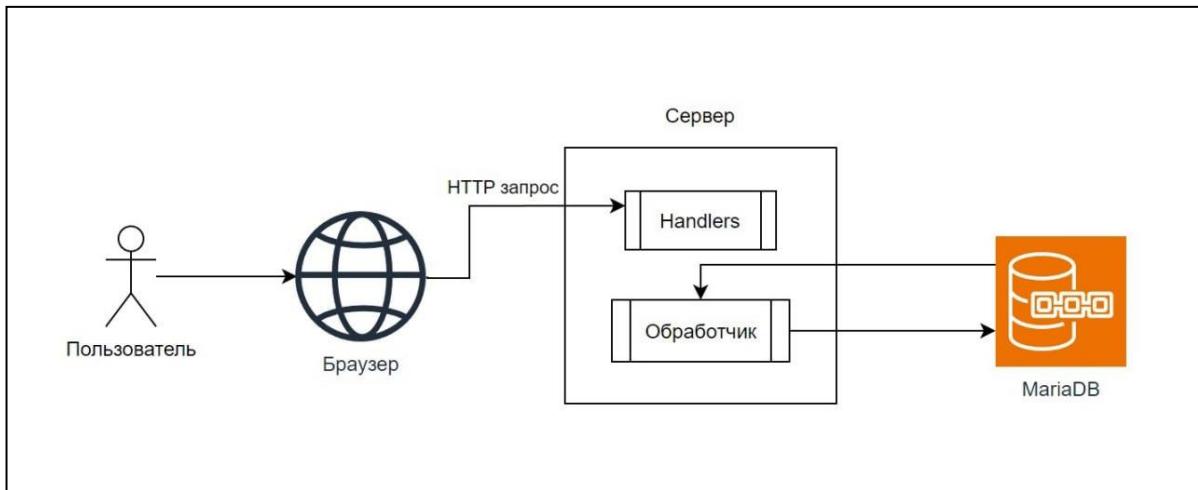


Рисунок Б5 – Схема движения запроса

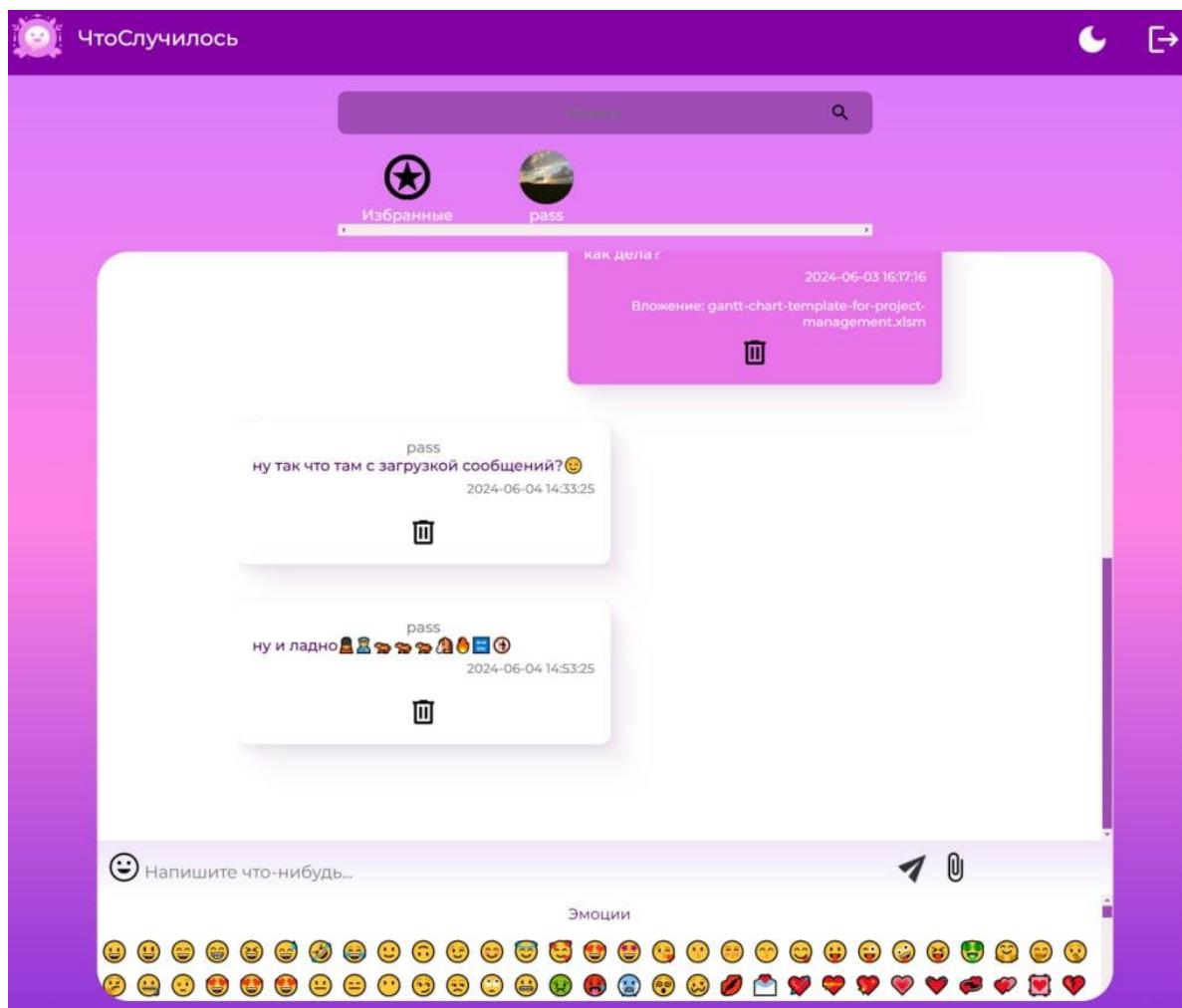


Рисунок Б6 – Скриншот окна чата

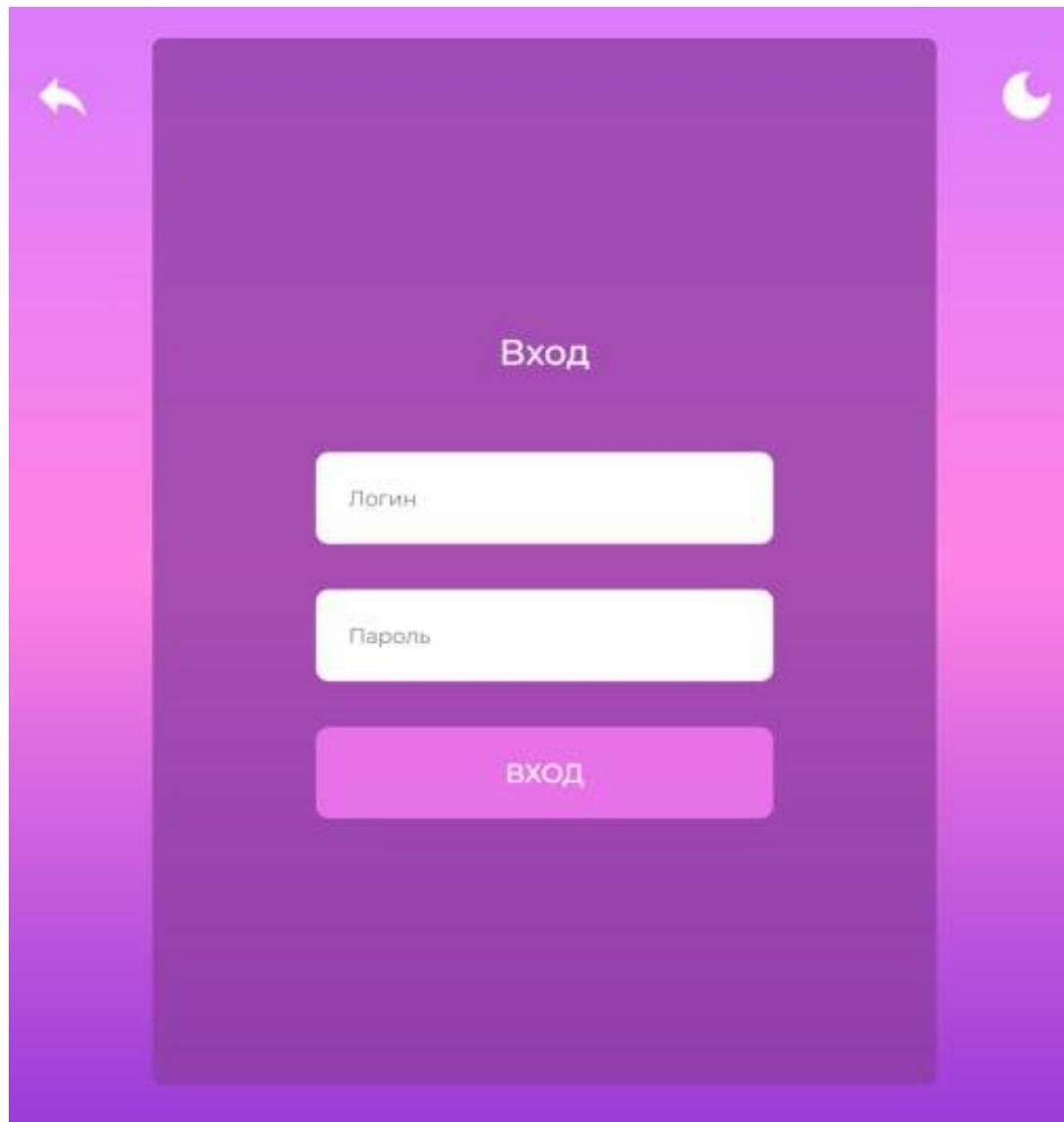


Рисунок Б7 – Скриншот окна авторизации

ПРИЛОЖЕНИЕ В
Перечень замечаний нормоконтролёра

Перечень замечаний нормоконтролёра к дипломному проекту

Студента Безуглого Виталия Витальевича группы ПИ-206

Обозначение документа	Документ	Условное обозначение	Содержание замечания

Дата _____

Подпись _____