

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

КУРСОВАЯ РАБОТА (ПРОЕКТ)

по дисциплине «Проектирование и архитектура программных систем»  
наименование дисциплины

на тему Веб мессенджер

Направление подготовки (специальность) 09.03.04  
(код, наименование)

Программная инженерия

Автор работы (проекта) Б.Р. Якубов  
(инициалы, фамилия) (подпись, дата)

Группа ПО-126

Руководитель работы (проекта) А. А. Чаплыгин  
(инициалы, фамилия) (подпись, дата)

Работа (проект) защищена  
(дата)

Оценка

Члены комиссии

подпись, дата	фамилия и. о.
подпись, дата	фамилия и. о.
подпись, дата	фамилия и. о.

Курск 2024 г.

# **Минобрнауки России**

## **Юго-Западный государственный университет**

Кафедра программной инженерии

### **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ (ПРОЕКТ)**

Студента Якубова Б.Р., шифр 21-06-0052, группа ПО-126

1. Тема «Веб мессенджер » .
2. Срок предоставления работы к защите «11» января 2024 г.
3. Исходные данные для создания программной системы:

#### 3.1. Перечень решаемых задач:

- 1) ознакомиться с обработкой веб запросов;
- 2) разработать серверную часть приложения;
- 3) разработать клиентскую часть приложения;
- 4) Доработать мессенджер и провести тестирование.

#### 3.2. Входные данные и требуемые результаты для программы:

1) Входными данными у мессенджера является возможность писать а так же принимать сообщения так же необходима система регистрации и входа, дабы отличать пользователей по их именам или же никнеймам.

2) выходными данными является тот факт что пользователь взаимодействует с приложением, предоставляя данные для регистрации, входа и отправки сообщений, и ожидает соответствующих результатов в виде успешных операций.

#### 4. Содержание работы (по разделам):

##### 4.1. Введение

##### 4.2. Анализ предметной области

4.3. Техническое задание: основание для разработки, назначение разработки, требования к программной системе, требования к оформлению документации.

4.4. Технический проект: общие сведения о программной системе, проект данных программной системы, проектирование архитектуры программной системы, проектирование пользовательского интерфейса программной системы.

4.5. Рабочий проект: спецификация компонентов и классов программной системы, тестирование программной системы, сборка компонентов программной системы.

4.6. Заключение

4.7. Список использованных источников

Руководитель работы (проекта)	_____	А. А. Чаплыгин
	(подпись, дата)	(инициалы, фамилия)
Задание принял к исполнению	_____	Б.Р. Якубов
	(подпись, дата)	(инициалы, фамилия)

## РЕФЕРАТ

Объем работы равен 56 страницам. Работа содержит 22 иллюстрации, 5 таблиц, 13 библиографических источников. Количество приложений – 1. Фрагменты исходного кода представлены в приложении А.

Перечень ключевых слов: Веб-приложение, мессенджер, JavaScript, HTML, CSS, frontend, регистрация пользователя, вход в систему, чат, отправка сообщений, AJAX, Fetch API, асинхронное программирование, WebSockets, серверная часть, Python, SQLite, Waitress, web-разработка, RESTful API, аутентификация, безопасность, клиент-серверное взаимодействие, URL-маршрутизация, реальное время.

Объектом разработки является веб-приложение мессенджера, предоставляющее современные возможности обмена сообщениями и взаимодействия пользователей в режиме реального времени.

Целью проекта является создание удобного и функционального мессенджера, способного обеспечить эффективное общение пользователей, а также привлечь новых пользователей своей удобной и привлекательной функциональностью.

В процессе разработки был создан веб-приложение а так же сервер с использованием современных технологий, таких как JavaScript, HTML, CSS и Python. Реализованы основные элементы мессенджера, включая чат, отправку сообщений, систему регистрации и входа в систему.

При разработке использовались технологии веб-разработки, обеспечивающие асинхронное взаимодействие и удобный пользовательский интерфейс.

Разработанное веб-приложение успешно прошло этапы тестирования и полностью готова к полноценной эксплуатации.

## ABSTRACT

The volume of work is 56 pages. The work contains 22 illustrations, 5 tables, 13 bibliographic sources. The number of applications is 1. The layout of the site, including the connection of components, is presented in annex A.

List of keywords: Web application, messenger, JavaScript, HTML, CSS, frontend, user registration, login, chat, sending messages, AJAX, Fetch API, asynchronous programming, WebSockets, server side, Python, SQLite, Waitress, web development, RESTful API, authentication, security, client-server interaction, URL routing, real time.

The object of development is a web-based messenger application that provides modern messaging and user interaction in real time.

The goal of the project is to create a convenient and functional messenger capable of providing effective user communication, as well as attracting new users with its convenient and attractive functionality.

In the process of development was created a web application and a server using modern technologies such as JavaScript, HTML, CSS and Python. The main elements of the messenger were realized, including chat, sending messages, registration and login system.

Web development technologies were used during the development, providing asynchronous interaction and a convenient user interface.

The developed web application has successfully passed the testing stages and is fully ready for full-scale operation.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ	9
1 Анализ предметной области	11
1.1 История развития веб-мессенджеров	11
1.2 Веб-мессенджер: особенности и тенденции	12
2 Техническое задание	14
2.1 Основание для разработки	14
2.2 Цель и назначение разработки	14
2.3 Описание веб-мессенджера	15
2.3.1 Иконки и элементы интерфейса	15
2.3.2 Функциональные элементы	18
2.4 Требования пользователя к интерфейсу веб-мессенджера	19
2.5 Моделирование вариантов использования	20
2.5.1 Диаграмма прецедентов	20
2.5.2 Сценарии прецедентов программы	20
2.6 Требования к оформлению документации	21
3 Технический проект	22
3.1 Общая характеристика организации решения задачи	22
3.2 Описание используемых библиотек и языков программирования	22
3.2.1 Серверная часть: Python	22
3.2.2 Веб-клиент: HTML, CSS, JavaScript	22
3.2.3 База данных: SQLite 3	22
3.3 Основные функции мессенджера	23
3.4 Технические детали проекта	23
3.4.1 Обновление сообщений в реальном времени	23
3.4.2 Авторизация пользователей	23
3.4.3 Хранение данных	23
3.4.4 Интерфейс веб-клиента	24
3.5 Структура проекта	24
3.5.1 Структура серверной части	24

3.5.2 Структура веб-клиента	25
3.5.3 Диаграмма классов	26
4 Рабочий проект	27
4.1 Классы, используемые при разработке веб-мессенджера	27
4.2 Системное тестирование веб-мессенджера	29
ЗАКЛЮЧЕНИЕ	38
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	38
ПРИЛОЖЕНИЕ А Фрагменты исходного кода программы	41

## ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ПО – программное обеспечение.

РП – рабочий проект.

ТЗ – техническое задание.

ТП – технический проект.

UML (Unified Modelling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.



## ВВЕДЕНИЕ

В современном мире информационных технологий веб-мессенджеры стали неотъемлемой частью нашей повседневной жизни. За последние десятилетия создание мессенджеров стало одним из самых востребованных направлений в программировании. Искусство разработки мессенджеров представляет уникальную возможность соединить техническое мастерство с креативностью и фантазией. В рамках данного курсового проекта мы погрузимся в увлекательный мир разработки веб-мессенджера и рассмотрим процесс создания современного приложения для обмена сообщениями с использованием технологий веб-разработки.

Наш веб-мессенджер предназначен для обеспечения эффективного и удобного общения пользователей в режиме реального времени, а также привлечения новых пользователей своей функциональностью и привлекательным дизайном.

В процессе разработки было создано веб-приложение, использующее современные веб-технологии, включая JavaScript, HTML и CSS. Реализованы основные элементы мессенджера, такие как чат, отправка сообщений, система регистрации и входа в систему.

При разработке использовались современные технологии веб-разработки, а также библиотеки и фреймворки, обеспечивающие удобный пользовательский интерфейс и безопасное взаимодействие.

*Цель настоящей работы* – создание современного веб-мессенджера с использованием технологий веб-разработки. Для достижения этой цели необходимо решить *следующие задачи*:

- провести анализ современных веб-мессенджеров;
- разработать концепцию и функциональные требования к мессенджеру;
- спроектировать веб-приложение с учетом требований безопасности и удобства использования;
- реализовать функционал мессенджера.

*Структура и объем проекта.* Проект включает в себя введение, 4 раздела основной части, заключение, список использованных технологий, 2 приложения. Общий объем проекта равен 9 страницам.

*Во введении* сформулирована цель проекта, поставлены задачи разработки, описана структура проекта, приведено краткое содержание каждого из разделов.

*В первом разделе* производится анализ существующих веб-мессенджеров и их технических характеристик.

*Во втором разделе* формулируются требования к создаваемому веб-мессенджеру на этапе технического задания.

*В третьем разделе* представлен детальный технический проект, включая выбор используемых технологий, архитектуру приложения и основные функциональные блоки.

*В четвертом разделе* представлен исходный код разработанного мессенджера, проведено тестирование и оценка его производительности.

В заключении изложены основные результаты проекта, полученные в ходе разработки.

В приложении А представлен исходный код.

## **1 Анализ предметной области**

### **1.1 История развития веб-мессенджеров**

История веб-мессенджеров тесно связана с развитием интернет-технологий и появлением средств онлайн-коммуникации. Этот жанр программного обеспечения стал неотъемлемой частью современного общения в виртуальном пространстве. Вот краткая история развития веб-мессенджеров:

**1. IRC и ICQ (1990-е годы):** Первыми формами онлайн-чата были интернет-ретрансляции (IRC), позволяющие пользователям общаться в реальном времени. В середине 1990-х годов появилась ICQ, первая широко распространенная программа мгновенного обмена сообщениями, что сделало общение в сети более доступным.

**2. AIM, MSN Messenger, Yahoo Messenger (2000-е годы):** В 2000-х годах появились AIM (AOL Instant Messenger), MSN Messenger и Yahoo Messenger. Они предложили дополнительные функции, такие как передача файлов, видеозвонки и персонализированные статусы.

**3. Skype и Google Talk (2000-е годы):** Skype и Google Talk (позднее ставший частью Google Hangouts) предоставили пользователям возможность совершения голосовых и видеозвонков, расширяя возможности коммуникации.

**4. WhatsApp, Telegram, и Viber (2010-е годы):** С массовым распространением смартфонов в 2010-х годах, приложения мессенджеров на основе мобильных платформ, такие как WhatsApp, Telegram и Viber, стали популярными, предлагая шифрование сообщений и другие расширенные функции.

**5. Современные веб-мессенджеры (2020-е годы):** С появлением современных веб-мессенджеров, таких как Slack, Microsoft Teams, и Discord, область коммуникаций в корпоративном и геймерском сегментах значительно разнообразилась. Эти приложения интегрируют функциональности чата с коллективной работой, обменом файлами и другими возможностями.

История веб-мессенджеров демонстрирует постоянное развитие средств онлайн-коммуникации, от простых текстовых чатов до много-

функциональных приложений, объединяющих в себе различные аспекты виртуального общения.

## **1.2 Веб-мессенджер: особенности и тенденции**

Веб-мессенджер – это приложение, предназначенное для обмена мгновенными сообщениями в режиме реального времени через сеть интернет. Основные особенности и тенденции веб-мессенджеров включают:

**1. Многоплатформенность:** Современные веб-мессенджеры обеспечивают поддержку различных платформ, включая веб-версии, мобильные приложения и настольные приложения.

**2. Безопасность и шифрование:** В ответ на повышенный интерес к безопасности в сети, веб-мессенджеры внедряют технологии шифрования сообщений для защиты конфиденциальности пользователей.

**3. Интеграция с другими сервисами:** Современные мессенджеры предоставляют интеграцию с другими сервисами и приложениями, такими как облачные хранилища, календари, видеоконференции, что обогащает опыт пользователей.

**4. Коллективная работа:** Веб-мессенджеры для бизнеса (например, Slack, Microsoft Teams) акцентируют внимание на коллективной работе, предоставляя инструменты для обмена файлами, обсуждения проектов и интеграции с рабочими инструментами.

**5. Использование искусственного интеллекта:** Некоторые веб-мессенджеры начинают использовать искусственный интеллект для улучшения опыта пользователей, предоставляя персонализированные рекомендации и автоматизированные задачи.

Тенденции веб-мессенджеров продолжают эволюцию в ответ на изменяющиеся потребности пользователей и бизнеса, что подчеркивает их важность в современной коммуникационной экосистеме.

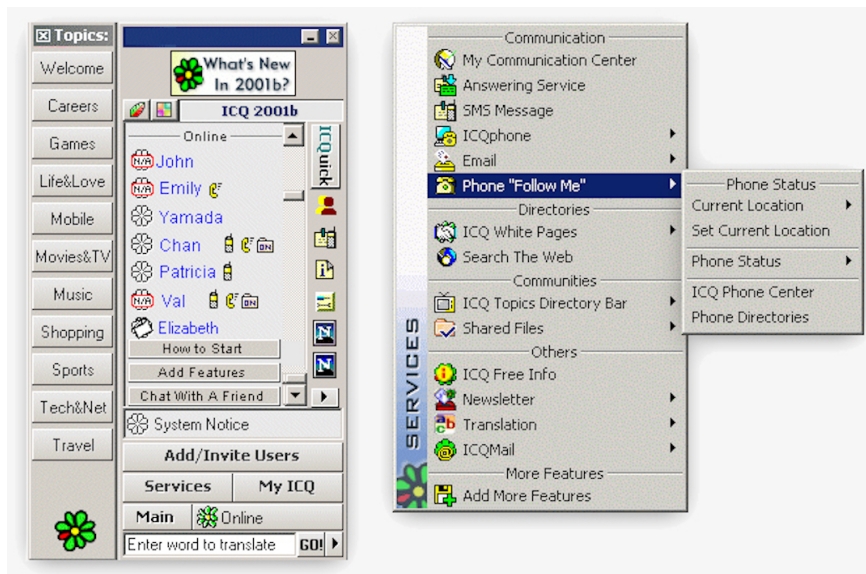


Рисунок 1.1 – Первый мессенджер ICQ

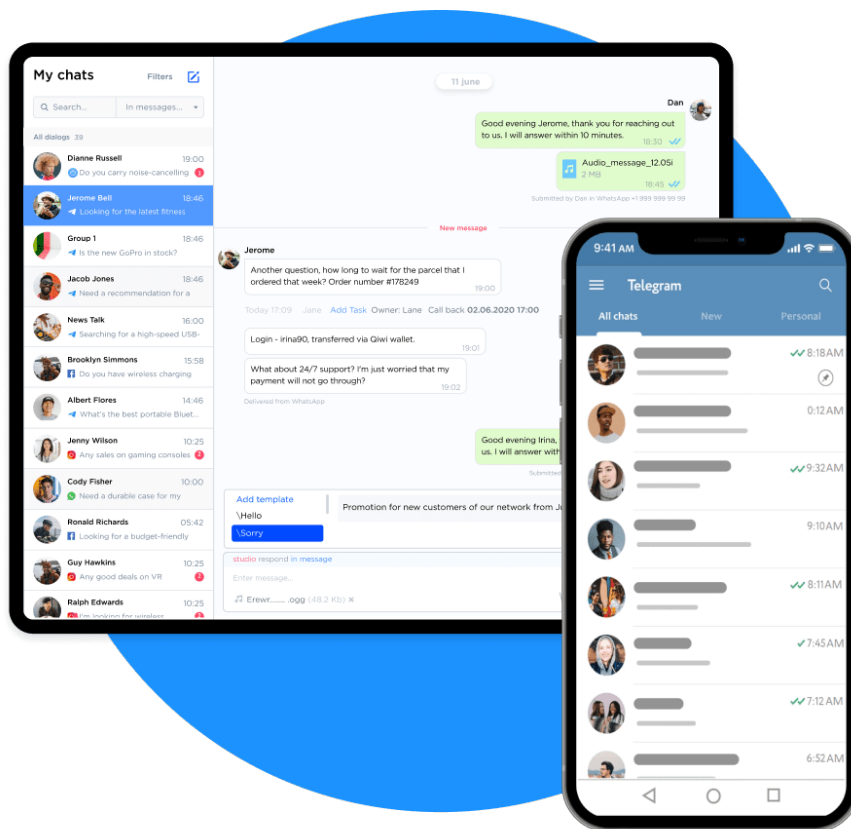


Рисунок 1.2 – Популярный мессенджер Telegram

## **2 Техническое задание**

### **2.1 Основание для разработки**

Основанием для разработки является потребность в создании веб-мессенджера в рамках проекта по предмету "Проектирование и разработка программных систем".

### **2.2 Цель и назначение разработки**

Основной целью данного проекта является разработка веб-мессенджера с использованием технологий Python для серверной части, а также JavaScript, HTML и CSS для клиентской части.

Целью разработки веб-мессенджера является предоставление эффективного инструмента для обмена мгновенными сообщениями в веб-пространстве, обеспечивая удобство использования и современные функциональные возможности.

Задачи данной разработки включают:

- Создание серверной части мессенджера на языке Python, обеспечивающей обработку сообщений, управление пользователями и хранение данных.
- Разработка клиентской части веб-мессенджера, используя JavaScript, HTML и CSS для обеспечения удобного пользовательского интерфейса.
- Реализация механизмов обмена сообщениями в режиме реального времени.
- Обеспечение безопасности обмена данными и внедрение технологий шифрования для защиты конфиденциальности.
- Тестирование и отладка системы для выявления и устранения возможных ошибок и недоразумений.

## 2.3 Описание веб-мессенджера

Веб-мессенджер представляет собой платформу для обмена мгновенными сообщениями между пользователями. Он обладает следующими особенностями:

1. **Многоплатформенность:** Поддержка различных платформ, включая веб-версию, мобильные приложения и настольные приложения.
2. **Безопасность и шифрование:** Внедрение технологий шифрования для обеспечения безопасности обмена сообщениями и конфиденциальности данных пользователей.
3. **Интеграция с другими сервисами:** Возможность интеграции с другими сервисами и приложениями для расширения функциональности мессенджера.
4. **Групповые чаты и коллективная работа:** Поддержка групповых чатов и инструментов для коллективной работы и обмена файлами.
5. **Использование искусственного интеллекта:** Возможное использование искусственного интеллекта для улучшения пользовательского опыта и предоставления персонализированных рекомендаций.

### 2.3.1 Иконки и элементы интерфейса

Сообщение - основной элемент взаимодействия пользователей в чате. Отображает текстовое сообщение и информацию об отправителе.



Рисунок 2.1 – Сообщение

Пользователь онлайн - индикатор онлайн пользователя. Отображает, что пользователь в данный момент находится в сети.



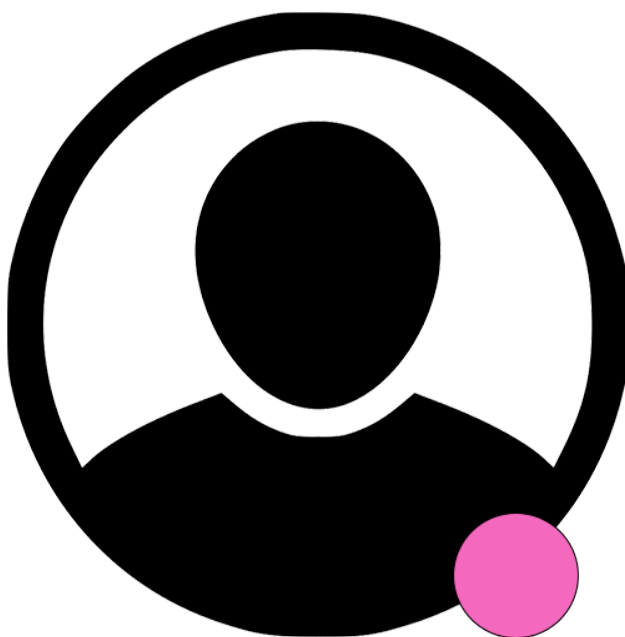


Рисунок 2.2 – Пользователь онлайн

”Смайлик иконка, указывающая на отправку ”смайликов” в сообщении.  
Позволяет пользователям отправлять ”смайлики”.



Рисунок 2.3 – Кнопка отправки смайликов

### 2.3.2 Функциональные элементы

Интерфейс должен предоставлять следующие элементы и функции:

- Возможность отправки текстовых сообщений.
- Возможность прикрепления и отправки файлов.
- Отображение онлайн статуса пользователей.
- Возможность создания и управления групповыми чатами.
- Индикация новых сообщений и уведомлений.

Композиция интерфейса мессенджера представлена на рисунке ??.



Рисунок 2.4 – Композиция мессенджера

## 2.4 Требования пользователя к интерфейсу веб-мессенджера

Веб-мессенджер должен обеспечивать удобное взаимодействие пользователей и предоставлять следующие возможности:

- Отправка текстовых сообщений: Пользователи должны иметь возможность обмениваться текстовыми сообщениями для эффективного общения.
- Отправка смайликов: Возможность отправлять смайлики, обеспечивая обмен различными эмоциями.
- Полная авторизация для дальнейшего отображения имени в чате.

## **2.5 Моделирование вариантов использования**

### **2.5.1 Диаграмма прецедентов**

Для разрабатываемого веб-мессенджера была реализована модель, которая обеспечивает наглядное представление вариантов использования мессенджера.

Она помогает в физической разработке и детальном анализе взаимосвязей объектов. При построении диаграммы вариантов использования применяется унифицированный язык визуального моделирования UML.

На основании анализа предметной области в программе должны быть реализованы следующие прецеденты:

- (a) Регистрация нового пользователя.
- (b) Авторизация пользователя.
- (c) Отправка текстового сообщения.
- (d) Прикрепление и отправка смайликов.

Рисунок 2.5 – Диаграмма прецедентов

### **2.5.2 Сценарии прецедентов программы**

- (a) Сценарий для прецедента «Регистрация нового пользователя»:
  - основной исполнитель: пользователь;
  - заинтересованные лица и их требования: пользователю необходимо предоставить уникальный логин и пароль;
  - предусловие: пользователь открыл веб-мессенджер в браузере;
  - основной успешный сценарий: пользователь вводит уникальный логин и пароль, система регистрирует нового пользователя.
- (b) Сценарий для прецедента «Авторизация пользователя»:
  - основной исполнитель: пользователь;
  - заинтересованные лица и их требования: пользователю необходимо предоставить зарегистрированный логин и пароль;

- предусловие: пользователь открыл веб-мессенджер в браузере;
- основной успешный сценарий: пользователь вводит зарегистрированный логин и пароль, система производит авторизацию.

(с) Сценарий для прецедента «Отправка текстового сообщения»:

- основной исполнитель: пользователь;
- заинтересованные лица и их требования: пользователю необходимо написать текст сообщения;
- предусловие: пользователь авторизован в системе;
- основной успешный сценарий: пользователь вводит текст сообщения, отправляет его.

(d) Сценарий для прецедента «Прикрепление и отправка смайликов»:

- основной исполнитель: пользователь;
- заинтересованные лица и их требования: пользователю необходимо выбрать смайлик и отправить его;
- предусловие: пользователь авторизован в системе;
- основной успешный сценарий: пользователь выбирает смайлик, отправляет его.

## **2.6 Требования к оформлению документации**

Разработка программной документации и программного изделия должна производиться согласно ГОСТ 19.102-77 и ГОСТ 34.601-90. Единая система программной документации.

## **3 Технический проект**

### **3.1 Общая характеристика организации решения задачи**

Необходимо спроектировать и разработать веб-мессенджер в ретро-стиле для обмена текстовыми сообщениями между пользователями. Все пользователи общаются в общем групповом чате, и нет личных контактов. Серверная часть реализована на языке программирования Python с использованием SQLite 3 для хранения сообщений и информации о пользователях. Веб-клиент реализован с использованием HTML, CSS и JavaScript.

### **3.2 Описание используемых библиотек и языков программирования**

Проект реализован с использованием следующих языков программирования и технологий:

#### **3.2.1 Серверная часть: Python**

Python используется для реализации серверной части мессенджера. Он обеспечивает взаимодействие с базой данных SQLite 3, обработку запросов от клиентов, отправку и прием сообщений.

#### **3.2.2 Веб-клиент: HTML, CSS, JavaScript**

Для создания веб-клиента используются технологии веб-разработки. HTML используется для структуры веб-страницы, CSS - для стилей и внешнего вида, JavaScript - для взаимодействия с сервером, обновления данных на странице и реализации интерактивности.

#### **3.2.3 База данных: SQLite 3**

SQLite 3 используется для хранения сообщений и информации о пользователях. Это легковесная база данных, которая интегрируется в проект и обеспечивает эффективное хранение и извлечение данных.

### **3.3 Основные функции мессенджера**

Проект реализует следующие основные функции мессенджера:

- Отправка сообщений: Пользователи могут отправлять текстовые сообщения в общий групповой чат.
- Отображение сообщений: Веб-интерфейс отображает сообщения, отправленные всеми пользователями, с указанием отправителя.
- Обновление сообщений в реальном времени: Новые сообщения отображаются на веб-странице без необходимости перезагрузки.
- Хранение сообщений и информации о пользователях: Сервер хранит сообщения и информацию о зарегистрированных пользователях в базе данных SQLite 3.

### **3.4 Технические детали проекта**

#### **3.4.1 Обновление сообщений в реальном времени**

Для обновления сообщений в реальном времени используется технология WebSocket. Когда пользователь отправляет новое сообщение, сервер немедленно уведомляет всех подключенных пользователей о появлении нового сообщения, и их веб-страницы обновляются динамически.

#### **3.4.2 Авторизация пользователей**

Для обеспечения безопасности и отделения пользователей мессенджера, каждый пользователь проходит процедуру авторизации. Пользователь вводит свое имя при запуске мессенджера, и это имя используется как его идентификатор при отправке сообщений и взаимодействии с сервером.

#### **3.4.3 Хранение данных**

Информация о пользователях и сообщения хранятся в базе данных SQLite 3. Каждое сообщение содержит информацию об отправителе, тексте сообщения и времени отправки. Таблица пользователей содержит уникальные имена пользователей и их идентификаторы.

#### **3.4.4 Интерфейс веб-клиента**

Интерфейс веб-клиента состоит из области отображения сообщений и формы для отправки новых сообщений. Сообщения отображаются в хронологическом порядке с указанием имени отправителя и времени отправки. Форма отправки нового сообщения включает поле ввода текста и кнопку для отправки.

### **3.5 Структура проекта**

#### **3.5.1 Структура серверной части**

Структура серверной части мессенджера представлена следующим образом:



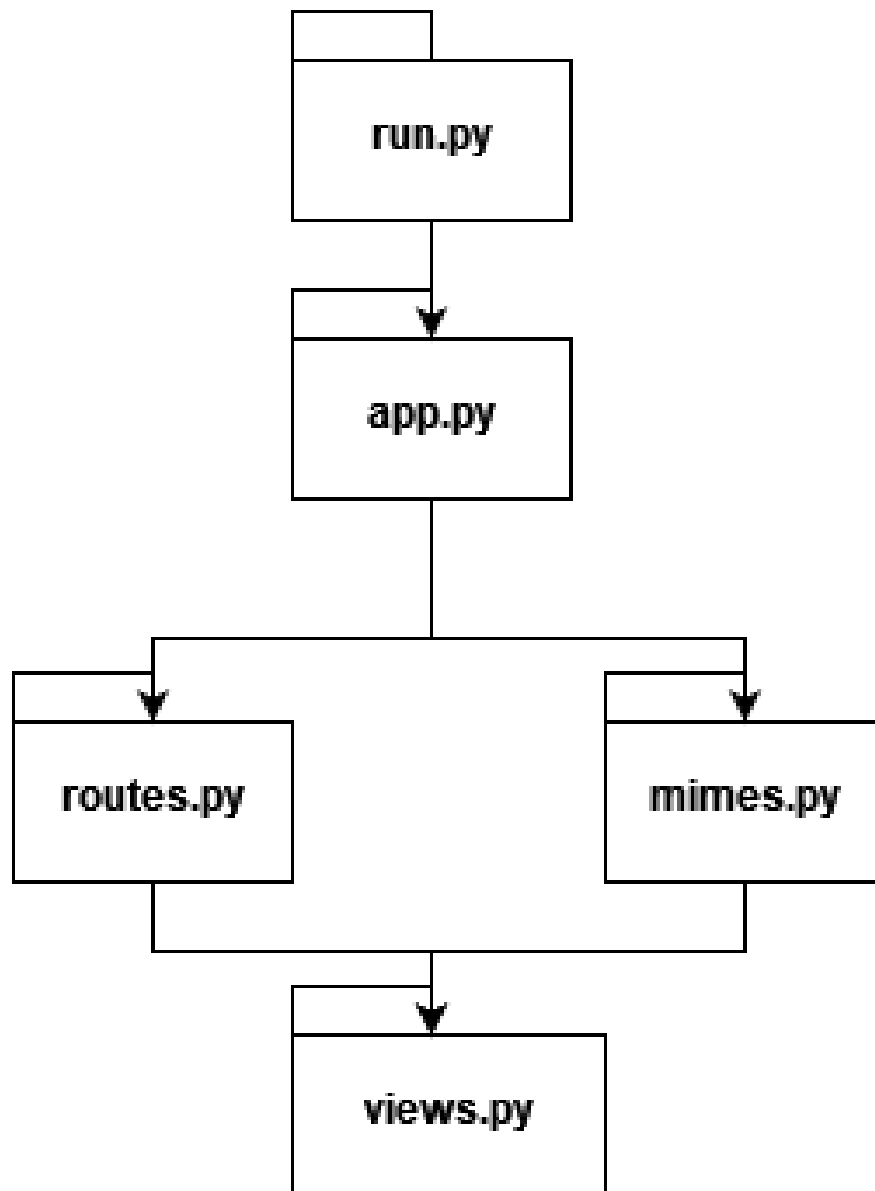


Рисунок 3.1 – Диаграмма компонентов сервера

### 3.5.2 Структура веб-клиента

Структура веб-клиента мессенджера представлена следующим образом:

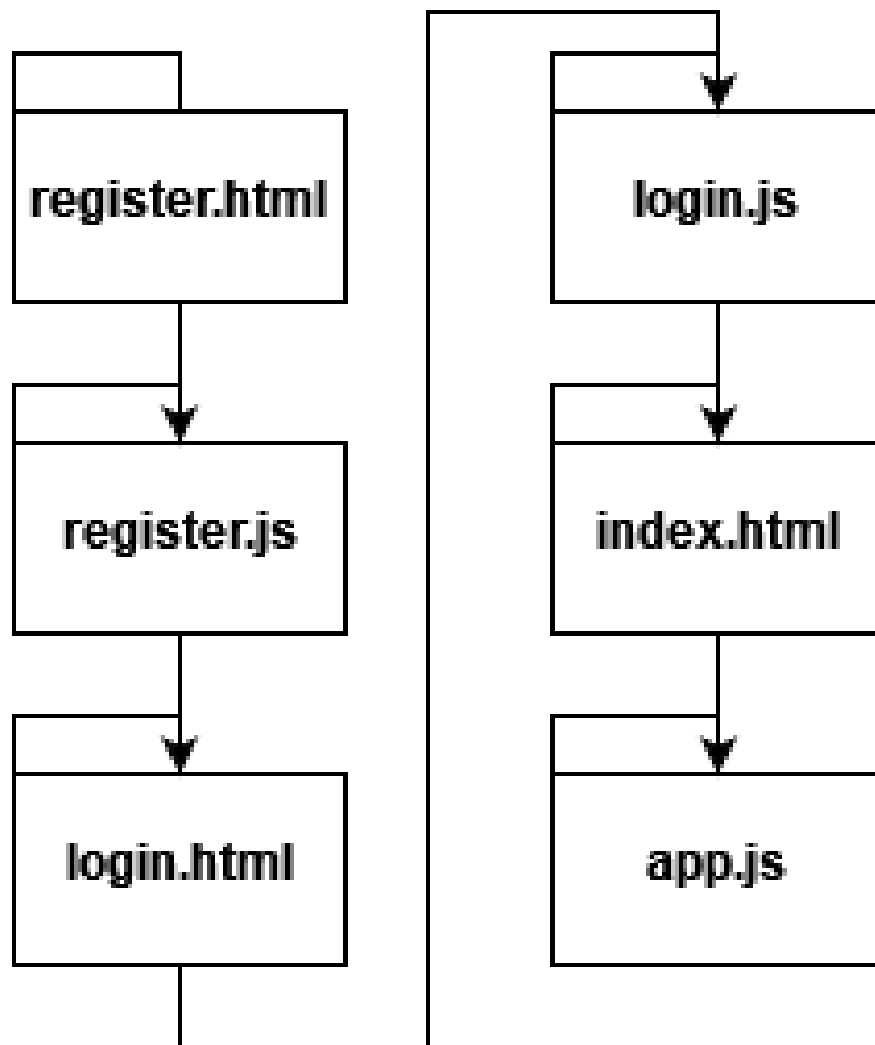


Рисунок 3.2 – Диаграмма компонентов веб-клиента

### 3.5.3 Диаграмма классов

На рисунке 3.3 изображена диаграмма классов для моего проекта веб-мессенджера. Данная диаграмма визуализирует взаимодействие между различными классами, представляющими пользователей, сообщения, чаты, элементы управления и другие основные компоненты системы.

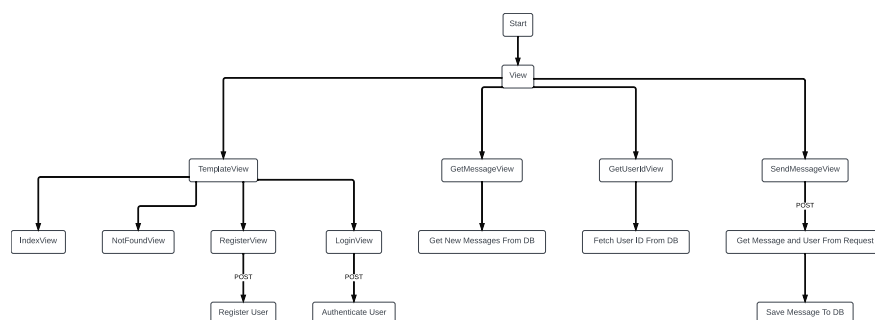


Рисунок 3.3 – Диаграмма классов

## 4 Рабочий проект

### 4.1 Классы, используемые при разработке веб-мессенджера

Можно выделить следующий список классов и их методов, использованных при разработке веб-мессенджера (таблица 4.1). Пример таблицы с уменьшенным межстрочным интервалом.

Таблица 4.1 – Описание классов, используемых в игре Montezuma

Название класса	Модуль, к которому относится класс	Описание класса	Методы и состояния
1	2	3	4
App	app.py	Главный класс приложения. Отвечает за обработку HTTP-запросов и вызов соответствующих представлений.	app, load, run
View	views.py	Базовый класс представления. Отвечает за формирование HTTP-ответа на основе URL и запроса.	response, read_file

Продолжение таблицы 4.1

1	2	3	4
TemplateView	views.py	Класс представления с использованием шаблона. Расширяет базовый класс View для работы с HTML-шаблонами.	response, read_file
IndexView	views.py	Класс представления для главной страницы. Использует HTML-шаблон.	response, read_file
NotFoundView	views.py	Класс представления для страницы 404 Not Found. Использует HTML-шаблон.	response, read_file
GetMessageView	views.py	Класс представления для получения новых сообщений. Обработывает запросы и возвращает JSON с новыми сообщениями.	response, get_new_messages_from_db
GetUserIdView	views.py	Класс представления для получения идентификатора пользователя. Обработывает запросы и возвращает JSON с идентификатором.	response, fetch_user_id_from_database
SendMessageView	views.py	Класс представления для отправки сообщений. Обработывает запросы и сохраняет новые сообщения в базу данных.	response, save_message_to_db, get_message_and_user_from_re

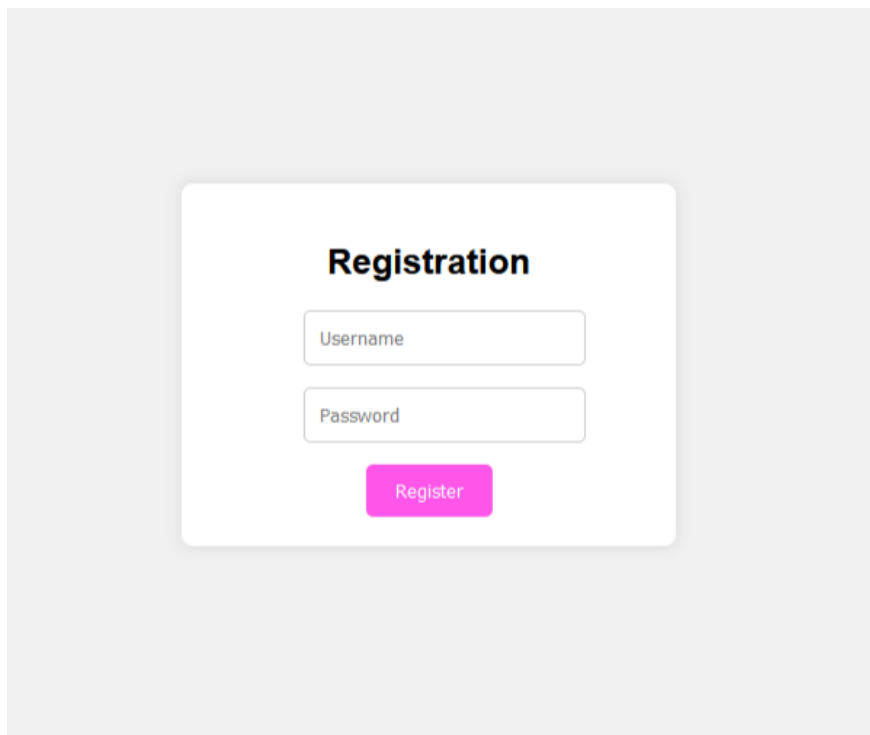
Продолжение таблицы 4.1

1	2	3	4
RegisterView	views.py	Класс представления для регистрации новых пользователей. Обработывает запросы и регистрирует новых пользователей в базе данных.	response, register_user, get_post_data
LoginView	views.py	Класс представления для аутентификации пользователей. Обработывает запросы и возвращает JSON с результатом аутентификации.	response, authenticate_user, get_post_data

## 4.2 Системное тестирование веб-мессенджера

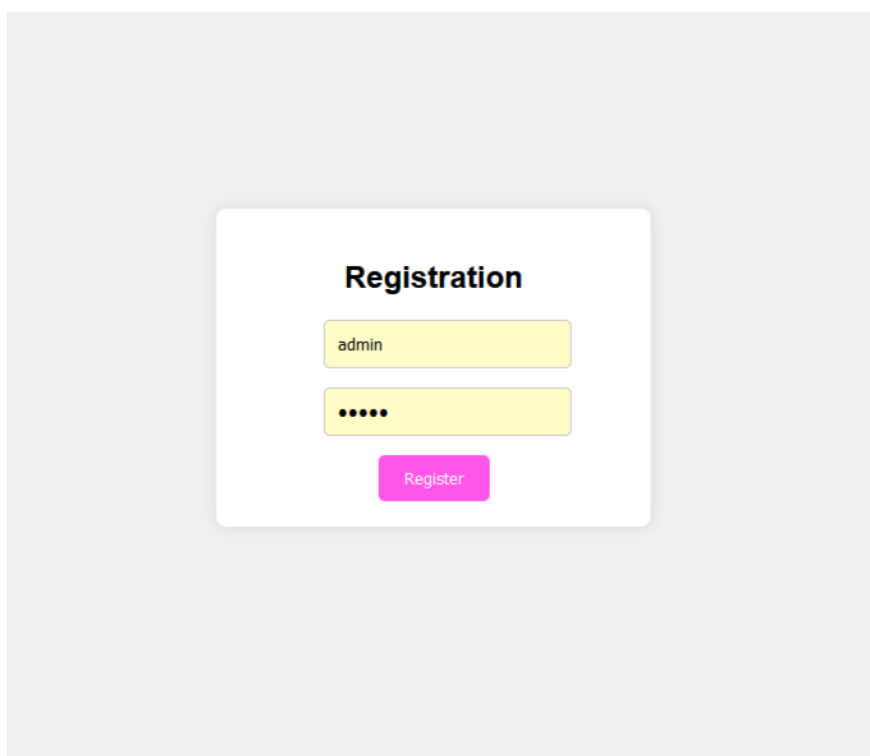
Для отладки работы веб-мессенджера разработаны следующие тестовые сценарии:

- (а) Случай использования: регистрация нового пользователя
  - Предусловие: веб-мессенджер доступен по адресу.
  - Тестовый случай: пользователь запускает веб-мессенджер, переходя по URL с припиской /register.
  - Ожидаемый результат: открывается страница регистрации с полем ввода логина и пароля (после успешной регистрации пользователя перекидывает на страницу /login).
  - Результат представлен на рисунках 4.1 и 4.2 4.3.



The image shows a registration form titled "Registration" centered on a light gray background. The form is a white rounded rectangle containing two input fields: "Username" and "Password", both of which are empty. Below the fields is a pink "Register" button.

Рисунок 4.1 – Страница регистрации (поля пустые)



The image shows the same registration form as in Figure 4.1, but with the fields filled. The "Username" field contains the text "admin" and the "Password" field contains five dots, indicating a masked password. The pink "Register" button remains below the fields.

Рисунок 4.2 – Страница регистрации (заполненные поля admin admin)

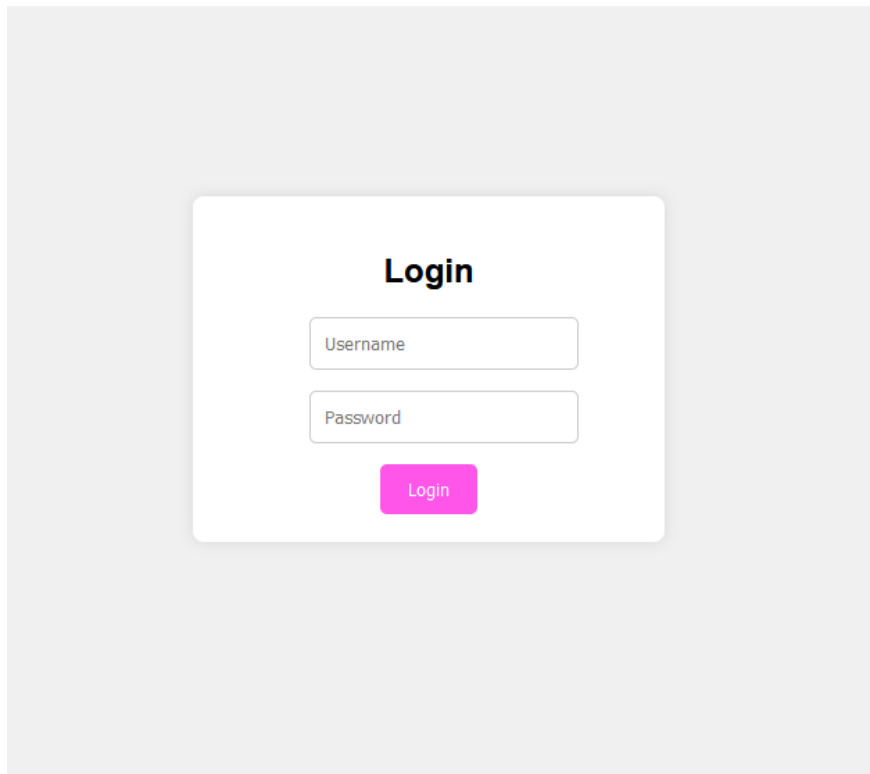


Рисунок 4.4 – Страница логина (поля пустые)

	id INTEGER PRIMARY KEY AUTOINCREMENT	username TEXT NOT NULL UNIQUE	password TEXT NOT NULL	user_id INTEGER
1	5	admin	admin	7e69e907a7804d478a...

Рисунок 4.3 – База данных после регистрации

(b) Случай использования: логин пользователя

- Предусловие: пользователь не авторизован в системе, веб-мессенджер доступен по адресу.
- Тестовый случай: пользователь переходит по URL ссылке с припиской /login.
- Ожидаемый результат: открывается страница логина с полями для ввода логина и пароля (после успешного логина пользователя перекидывает на страницу с мессенджером).
- Результаты представлены на рисунках 4.4 и 4.5.

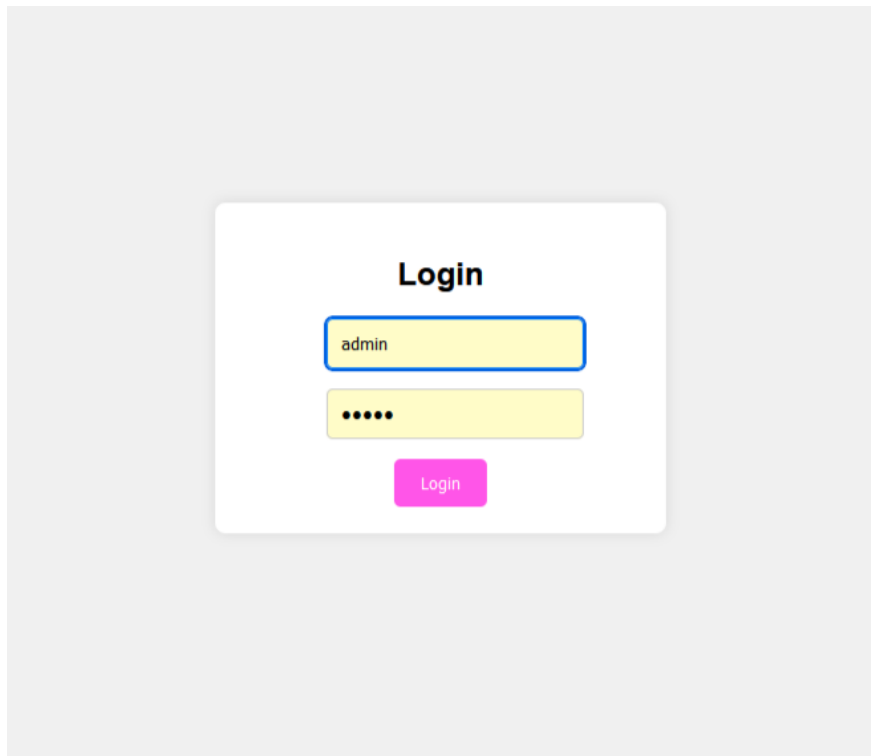


Рисунок 4.5 – Страница логина(заполненные поля admin admin)

(с) Случай использования: отправка сообщения (без логина)

- Предусловие: пользователь не авторизован в системе.
- Тестовый случай: пользователь вводит текст сообщения и нажимает кнопку "Отправить".
- Ожидаемый результат: сообщение отправляется, отображаемое имя Guest.
- Результаты представлены на рисунках 4.6 и 4.7.



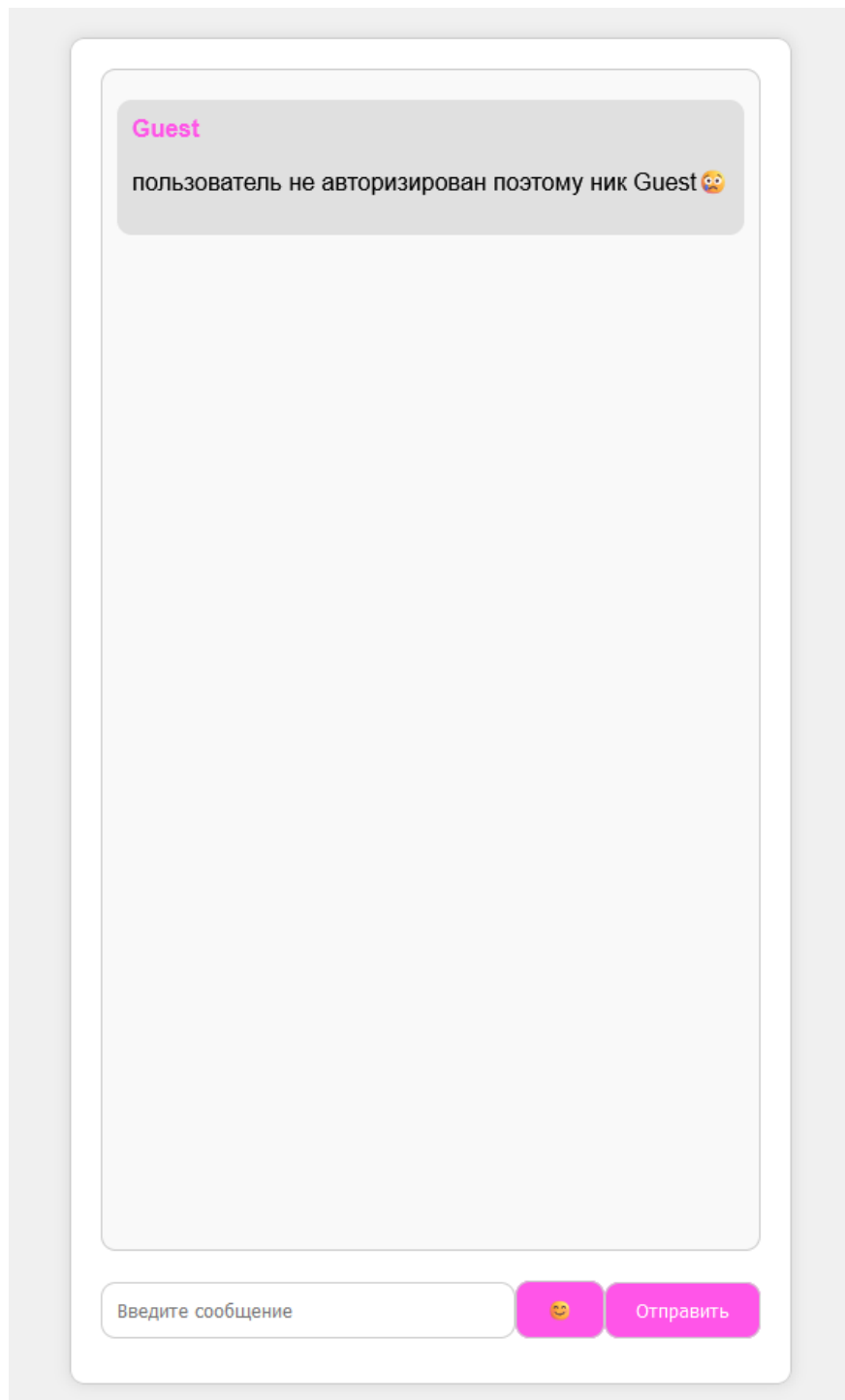


Рисунок 4.6 – Пользователь не вошел и отправил сообщение

[illegible]

Рисунок 4.7 – Пустая база данных (зарегистрированные пользователи отсутствуют)

(d) Случай использования: отправка сообщения (логин subemalevich)

- Предусловие: пользователь авторизован в системе.
- Тестовый случай: пользователь вводит текст сообщения и нажимает кнопку "Отправить".
- Ожидаемый результат: сообщение отправляется, отображаемое имя cubemalevich.
- Результаты представлены на рисунках 4.8 и 4.9.

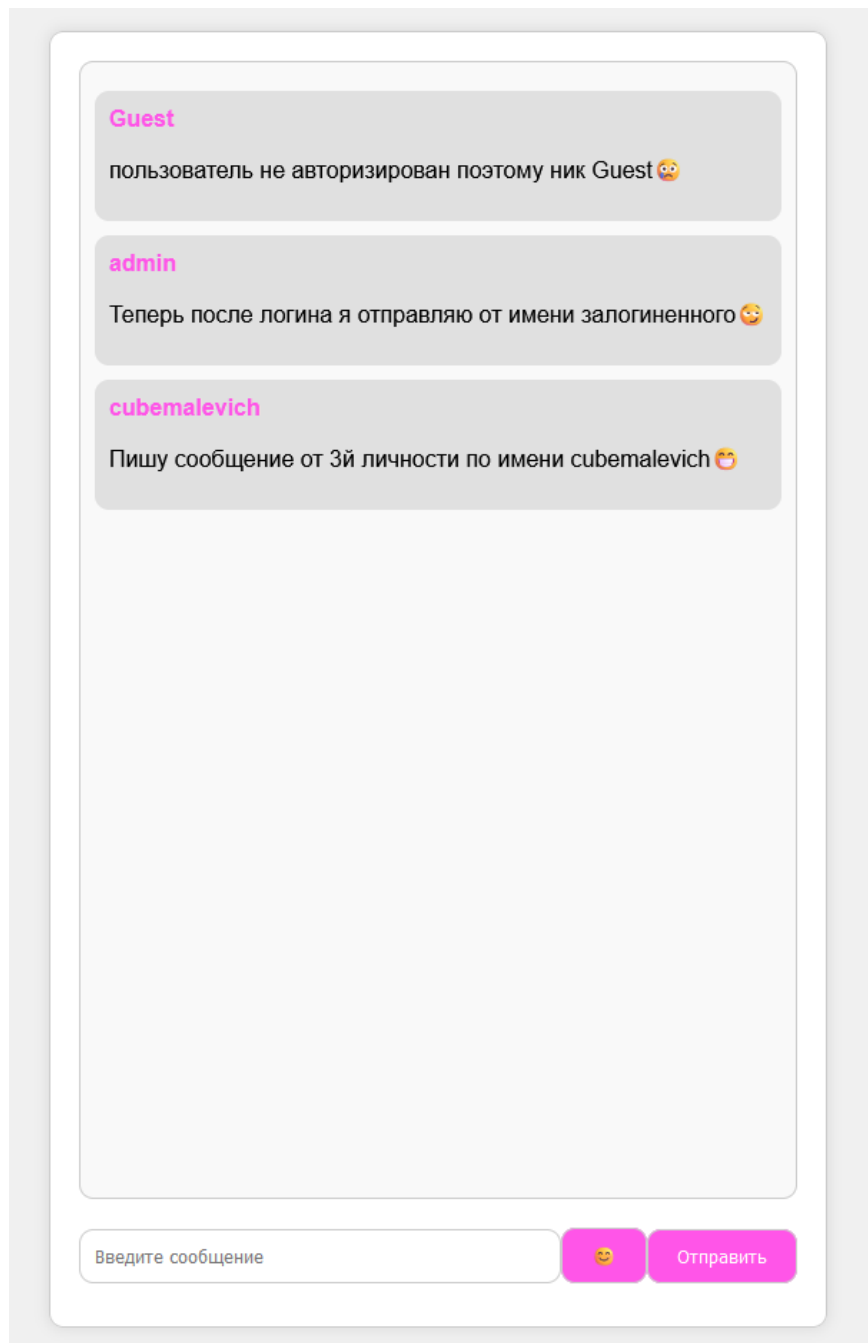


Рисунок 4.8 – Пользователь cubemalevich отправил сообщение

The screenshot shows a database query result in a tool like DBeaver. The table is named 'messages' and has the following columns: message\_id (INTEGER, PRIMARY KEY, AUTOINCREMENT), user\_id (INTEGER), sender (TEXT, NOT NULL), message\_text (TEXT, NOT NULL), and timestamp (INTEGER). The data is as follows:

	message_id	user_id	sender	message_text	timestamp
1	11	NULL	Guest	Guest: пользователь не авторизован поэтому ник Guest 😬	1785957125
2	12	admin	admin	admin: Теперь после логина я отправляю от имени залогиненного 😬	1785957194
3	13	cubemalevich	cubemalevich	cubemalevich: Пишу сообщение от 3й личности по имени cubemalevich 😬	1785957257

Рисунок 4.9 – База данных с сообщениями

(е) Случай использования: отправка смайликов

- Предусловие: пользователь авторизован в системе.

- Тестовый случай: пользователь нажимает кнопку отправки смайликов.
- Ожидаемый результат: открывается ”пикер”смайликов, пользователь выбирает смайлик и отправляет.
- Результаты представлены на рисунках 4.10, 4.11 и 4.12.

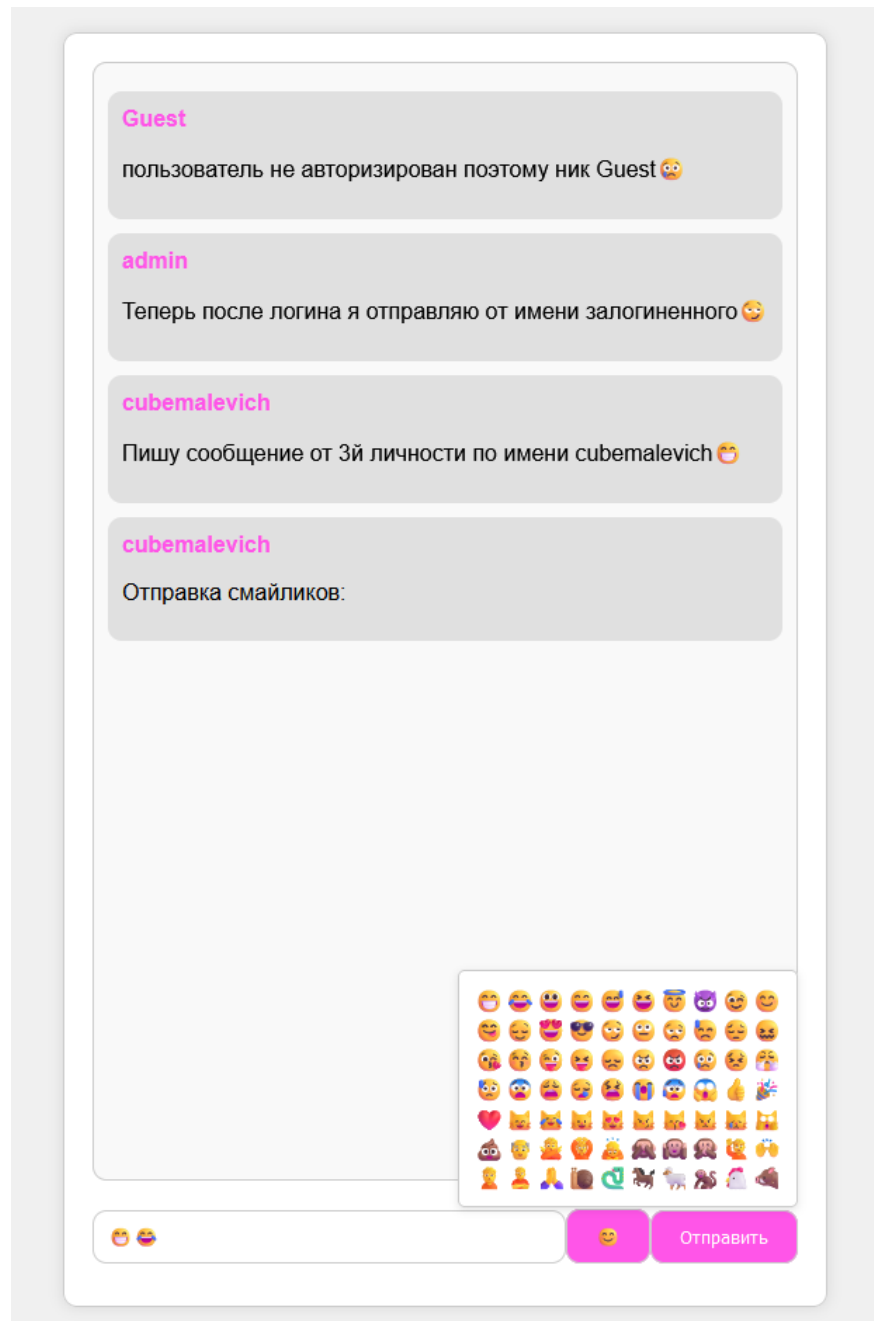


Рисунок 4.10 – Пикер смайликов



## ЗАКЛЮЧЕНИЕ

Веб-мессенджер, разработанный с использованием современных веб-технологий, представляет собой реализацию эффективного средства обмена сообщениями. Процесс разработки данного проекта позволил глубже понять принципы работы веб-приложений, взаимодействие с базой данных и создание динамического пользовательского интерфейса.

Основные результаты данного проекта включают в себя:

(a) Проведен анализ существующих веб-мессенджеров, выделены ключевые функциональности и принципы их реализации, которые были учтены при разработке данного проекта.

(b) Создано веб-приложение, включающее в себя возможность регистрации и авторизации пользователей, отправки и получения сообщений.

(c) Проведено тестирование веб-мессенджера, результаты которого подтвердили корректную работу основных функций приложения.

Все изначально поставленные задачи были успешно выполнены в ходе разработки. Созданный веб-мессенджер предоставляет пользователям удобный инструмент для общения и обмена информацией.

Разработанный проект успешно реализует основные функциональности веб-мессенджера, делая его полезным и эффективным средством коммуникации.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мэтиз Э. Изучаем Python: программирование игр, визуализация данных, веб-приложения. 3-е изд. / Мэтиз Эрик 2022 ISBN 978-5-4461-1528-0. – Текст : непосредственный.
2. МакГрат М. JavaScript для начинающих. 6-е издание / МакГрат Майк 2023, ISBN 978-5-04-121621-4. – Текст : непосредственный.
3. Джейкс, Дж. Python Crash Course / Дж. Джейкс. – Санкт-Петербург : Питер, 2022. – 560 с. – ISBN 978-5-4461-2206-7.
4. Любанович Б. Простой Python. Современный стиль программирования/ Б. Любанович – Санкт-Петербург : Питер, 2022. – 592 с. – ISBN 978-5-4461-1639-3. – Текст : непосредственный.
5. Бейдер Д. Знакомство с Python / Д. Бейдер. – Санкт-Петербург : Питер, 2023. – 512 с. – ISBN 978-5-4461-1924-0. – Текст : непосредственный.
6. Джонс Б, Бизли Д. Python. Книга рецептов /Б. Джонс, Д. Бизли – Москва : ДМК Пресс, 2019. – 898 с. – ISBN 978-5-97060-751-0. – Текст : непосредственный.
7. Гринберг М. Разработка веб-приложений с использованием Flask на языке Python / Гринберг Мигель 2016. ISBN - 978-5-97060-206-5, 978-1-449-37262-0
8. Вигерс, К. Разработка требований к программному обеспечению : [практические приёмы сбора требований и управления ими при разработке программных продуктов : пер. с англ.] / К. Вигерс, Д. Битти. – 3-е изд., доп. – Санкт-Петербург, 2020. – 736 с. – ISBN: 978-5-9775-3348-5.
9. Гамма Э., Хелм Р. Паттерны объективно–ориентированного программирования/ БЭ. Гамма, Р. Хелм – Санкт-Петербург : Питер, 2021 – 448 с. – ISBN 978-5-4461-1595-2. – Текст : непосредственный.
10. Рамбо Дж. UML 2.0. Объектно-ориентированное моделирование и разработка/ Дж. Рамбо – Санкт-Петербург : Питер, 2015 – 545 с. – ISBN 5-469-00814-2. – Текст : непосредственный.

11. Скотт Адам Д., Пауэрс Ш. JavaScript. Рецепты для разработчиков. 3-е изд / Скотт Адам Д., Пауэрс Шелли 2023. ISBN - 978-5-4461-2001-7 – Текст : непосредственный.

12. Кириченко А.;Никольский А. Web на практике. CSS, HTML, JavaScript, MySQL, PHP для fullstack-разработчиков 2023. ISBN - 978-5-94387-271-6 – Текст : непосредственный.

13. Веб-технологии для разработчиков [Электронный ресурс]  
<https://developer.mozilla.org/ru/docs/Web>



## ПРИЛОЖЕНИЕ А

### Фрагменты исходного кода программы

app.js

```
1 document.addEventListener("DOMContentLoaded", function () {
2     const chatBox = document.getElementById("chat-box");
3     const messageForm = document.getElementById("message-form");
4     const messageInput = document.getElementById("message-input");
5     const emojiButton = document.getElementById("emoji-button");
6     const emojiPicker = document.getElementById("emoji-picker");
7
8     const emojiList = [
9         '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐',
10        '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐',
11        '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐',
12        '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐',
13        '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐',
14        '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐',
15        '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐', '☐'
16    ];
17
18    const emojisPerRow = 10;
19
20    emojiList.forEach((emoji, index) => {
21        const emojiSpan = document.createElement('span');
22        emojiSpan.innerText = emoji;
23        emojiPicker.appendChild(emojiSpan);
24
25        if ((index + 1) % emojisPerRow === 0) {
26            const breakElement = document.createElement('br');
27            emojiPicker.appendChild(breakElement);
28        }
29    });
30
31    emojiButton.addEventListener("click", function () {
32        emojiPicker.classList.toggle("show");
33    });
34
35    emojiPicker.addEventListener("click", function (event) {
36        if (event.target.tagName === "SPAN") {
37            const emoji = event.target.innerText;
38            messageInput.value += emoji;
39        }
40        emojiPicker.classList.remove("show");
41    });
42
43    let lastTimestamp = 0;
44
45    function getMessages() {
46        fetch(`/get_messages?timestamp=${lastTimestamp}`)
47            .then((response) => response.json())
48            .then((data) => {
49                console.log("Received data from server:", data);
```

```

50         if (data && data.messages) {
51             data.messages.forEach((message) => {
52                 if (message.message_text && message.sender) {
53                     addMessage(message, message.sender);
54                 }
55             });
56         }
57         lastTimestamp = data.timestamp || lastTimestamp;
58     })
59     .catch((error) => {
60         console.error("Error getting messages:", error);
61     });
62 }
63
64 function addMessage(message, sender) {
65     console.log("Adding message:", message, "from sender:", sender);
66
67     if (message && message.message_text && sender) {
68         const messageDiv = document.createElement("div");
69         messageDiv.className = "message";
70         const messageText = document.createElement("p");
71
72         const displaySender = sender || 'Guest';
73         const displayMessage = message.message_text || '';
74
75         console.log("Displaying:", displaySender, displayMessage);
76
77         const senderDiv = document.createElement("div");
78         senderDiv.className = "sender";
79         senderDiv.innerText = displaySender;
80         messageDiv.appendChild(senderDiv);
81
82         messageText.innerText = displayMessage;
83         messageDiv.appendChild(messageText);
84
85         chatBox.appendChild(messageDiv);
86     }
87 }
88
89 messageForm.addEventListener("submit", function (e) {
90     e.preventDefault();
91     const message = messageInput.value;
92
93     if (message) {
94         const username = sessionStorage.getItem('username');
95         addMessage(message, username);
96         messageInput.value = "";
97
98         const requestBody = { message, username };
99
100         fetch("/send_message", {
101             method: "POST",
102             body: JSON.stringify({ message, username }),
103             headers: { "Content-Type": "application/json" },

```

```

104     })
105     .then((response) => response.json())
106     .then((data) => {
107         if (data && data.message) {
108             console.log("Server response after sending message:",
109                 data.message);
110         } else {
111             console.error("Invalid server response:", data);
112         }
113     })
114     .catch((error) => {
115         console.error("There was a problem with the fetch operation:"
116             , error);
117     });
118 }
119 });
120
121 setInterval(getMessages, 1000);
122 });

```

### login.js

```

1 document.addEventListener("DOMContentLoaded", function () {
2     const loginForm = document.getElementById("login-form");
3
4     loginForm.addEventListener("submit", function (e) {
5         e.preventDefault();
6
7         console.log("Form submitted");
8
9         const usernameInput = document.getElementById("username");
10        const passwordInput = document.getElementById("password");
11
12        const username = usernameInput.value;
13        const password = passwordInput.value;
14
15        console.log("Username input (login.html):", username);
16        console.log("Password input (login.html):", password);
17
18        const requestBody = new URLSearchParams({ username, password });
19        console.log("Request body:", requestBody);
20
21        if (username && password) {
22            console.log("Sending login request with username:", username);
23
24            fetch("/login", {
25                method: "POST",
26                headers: {
27                    'Content-Type': 'application/x-www-form-urlencoded'
28                },
29                body: requestBody
30            })
31            .then(response => {
32                console.log("Received response from server:", response);
33                if (!response.ok) {

```

```

34         throw new Error("Network response was not ok");
35     }
36     return response.json();
37 })
38 .then(data => {
39     console.log("Server response:", data);
40     sessionStorage.setItem('username', username);
41
42     if (data.redirect) {
43         console.log("Redirecting to:", data.redirect);
44         window.location.replace(data.redirect);
45     } else if (data.error) {
46         sessionStorage.setItem('username', username);
47         console.log("Username saved to sessionStorage:", username
48             );
49         console.log("Current sessionStorage:", sessionStorage);
50     } else {
51         if (data.user_id && data.user_id !== 'null') {
52             sessionStorage.setItem('username', username);
53             getUserId(username);
54         } else {
55             console.error("User ID is undefined, empty, or 'null
56                 '. Cannot proceed.");
57             sessionStorage.setItem('username', username);
58         }
59     }
60 })
61 .catch(error => {
62     console.error("There was a problem with the fetch operation:"
63         , error);
64 });
65
66
67 function getUserId(username) {
68     fetch('/get_user_id')
69     .then(response => response.json())
70     .then(data => {
71         sessionStorage.setItem('username', username);
72     })
73     .catch(error => {
74         console.error("Error fetching user_id:", error);
75     });
76 }
77
78
79 });

```

### register.js

```

1 document.addEventListener("DOMContentLoaded", function () {
2     const registerForm = document.getElementById("register-form");
3

```

```

4   registerForm.addEventListener("submit", function (e) {
5       e.preventDefault();
6       console.log("Form submitted");
7
8       const usernameInput = document.getElementById("username");
9       const passwordInput = document.getElementById("password");
10
11      const username = usernameInput.value;
12      const password = passwordInput.value;
13
14      const requestBody = new URLSearchParams({ username, password });
15      console.log("Request body:", requestBody); // Проверка содержимого
16                                                  запроса
17
18      fetch('/register', {
19          method: 'POST',
20          headers: {
21              'Content-Type': 'application/x-www-form-urlencoded'
22          },
23          body: requestBody
24      }).then(response => {
25          console.log("Server response:", response); // Проверка ответа от
26                                                  сервера
27
28          if (!response.ok) {
29              throw new Error('Network response was not ok');
30          }
31          window.location.replace("/login");
32          //return response.json();
33      }).then(data => {
34          console.log(data.message); // Вывод сообщения об успешной
35                                  регистрации
36      }).catch(error => {
37          console.error('There was a problem with the fetch operation:',
38                      error);
39      });
40 });

```

## index.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <link rel="stylesheet" type="text/css" href="static/style.css">
7      <title>Messenger</title>
8      <script src="static/app.js"></script>
9  </head>
10 <body>
11     <div class="chat-container">
12         <div class="chat-box" id="chat-box">
13             <!-- Здесь будут отображаться сообщения -->
14         </div>

```

```

15     <form id="message-form">
16         <div id="input-container">
17             <input type="text" id="message-input" placeholder="
18                 Введите сообщение" autocomplete="off">
19             <button type="button" id="emoji-button">☺</button>
20             <button type="submit">Отправить</button>
21             <!-- Обновленный пикер смайликов -->
22             <div id="emoji-picker" class="emoji-picker">
23                 <div class="emoji-row" id="emoji-row-1"></div>
24                 <div class="emoji-row" id="emoji-row-2"></div>
25                 <div class="emoji-row" id="emoji-row-3"></div>
26                 <div class="emoji-row" id="emoji-row-4"></div>
27                 <div class="emoji-row" id="emoji-row-5"></div>
28             </div>
29         </div>
30     </form>
31 </div>
32 </body>
</html>

```

## login.html

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <link rel="stylesheet" type="text/css" href="/static/login.css">
7     <title>Login</title>
8 </head>
9 <body>
10     <div class="container">
11         <form id="login-form">
12             <h2>Login</h2>
13             <div class="input-container">
14                 <input type="text" id="username" name="username" placeholder="
15                     Username" autocomplete="off" required>
16             </div>
17             <div class="input-container">
18                 <input type="password" id="password" name="password"
19                     placeholder="Password" autocomplete="off" required>
20             </div>
21             <input type="hidden" id="user-id" name="user-id">
22             <button type="submit">Login</button>
23         </form>
24     </div>
25     <script src="/static/login.js"></script>
26 </body>
27 </html>

```

## register.html

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">

```

```

5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <link rel="stylesheet" type="text/css" href="/static/register.css">
7     <title>Registration</title>
8 </head>
9 <body>
10     <div class="container">
11         <form id="register-form">
12             <h2>Registration</h2>
13             <div class="input-container">
14                 <input type="text" id="username" name="username" placeholder=
                    "Username" autocomplete="off" required>
15             </div>
16             <div class="input-container">
17                 <input type="password" id="password" name="password"
                    placeholder="Password" autocomplete="off" required>
18             </div>
19             <button type="submit">Register</button>
20         </form>
21     </div>
22     <script src="/static/register.js"></script>
23 </body>
24 </html>

```

#### app.py

```

1 import os
2 import re
3 import sqlite3
4
5 from routes import routes
6 from mimes import get_mime
7 from views import NotFoundView
8
9 conn = sqlite3.connect('data.db')
10 cursor = conn.cursor()
11
12 # Таблица учетных данных пользователей
13 cursor.execute('''
14     CREATE TABLE IF NOT EXISTS users (
15         id INTEGER PRIMARY KEY AUTOINCREMENT,
16         username TEXT NOT NULL,
17         password TEXT NOT NULL,
18         user_id INTEGER,
19         UNIQUE (username)
20     )
21 ''')
22 # Таблица сообщений
23 cursor.execute('''
24     CREATE TABLE IF NOT EXISTS messages (
25         message_id INTEGER PRIMARY KEY AUTOINCREMENT,
26         user_id INTEGER,
27         sender TEXT NOT NULL,
28         message_text TEXT NOT NULL,
29         timestamp INTEGER
30     )

```

```

31 '''
32
33 conn.close()
34
35 def load(file_name):
36     f = open(file_name, encoding='utf-8')
37     data = f.read()
38     f.close()
39     return data
40
41 def app(envIRON, start_response):
42     """
43     (dict, callable( status: str,
44                     headers: list[(header_name: str, header_value: str)]))
45     -> body: iterable of strings_
46     """
47     url = envIRON['REQUEST_URI']
48     view = None
49
50     for key in routes.keys():
51         if re.match(key, url) is not None:
52             view = routes[key](url)
53             break
54
55     if view is None:
56         view = NotFoundView(url)
57
58     resp = view.response(envIRON, start_response)
59     # Возвращаем HTTP-ответ с сгенерированной страницей
60     return resp

```

#### views.py

```

1 from collections import namedtuple
2 import sqlite3
3 import json
4 import time
5 import uuid
6
7 from urllib.parse import parse_qs
8 from mimetypes import get_mime
9 from webob import Request
10
11 Response = namedtuple("Response", "status headers data")
12
13 class View:
14     path = ''
15
16     def __init__(self, url) -> None:
17         self.url = url
18
19     def response(self, envIRON, start_response):
20         file_name = self.path + self.url
21         headers = [('Content-type', get_mime(file_name))]
22

```



```

23         try:
24             data = self.read_file(file_name[1:])
25             status = '200 OK'
26         except FileNotFoundError:
27             data = ''
28             status = '404 Not found'
29
30         start_response(status, headers)
31         return [data.encode('utf-8')]
32
33     def read_file(self, file_name):
34         print(file_name)
35         with open(file_name, 'r', encoding='utf-8') as file:
36             return file.read()
37
38 class TemplateView(View):
39     template = ''
40
41     def __init__(self, url) -> None:
42         super().__init__(url)
43         self.url = '/' + self.template
44
45     def response(self, environ, start_response):
46         file_name = self.path + self.url
47         headers = [('Content-type', get_mime(file_name))]
48         try:
49             data = self.read_file(file_name[1:])
50             status = '200 OK'
51         except FileNotFoundError:
52             data = ''
53             status = '404 Not found'
54         start_response(status, headers)
55         return [data.encode('utf-8')]
56
57     def read_file(self, file_name):
58         print(file_name)
59         with open(file_name, 'r', encoding='utf-8') as file:
60             return file.read()
61
62 class IndexView(TemplateView):
63     template = 'templates/index.html'
64
65 class NotFoundView(TemplateView):
66     pass
67
68 class GetMessageView(View):
69     def __init__(self, url) -> None:
70         super().__init__(url)
71         self.last_timestamp = 0
72
73     def response(self, environ, start_response):
74         headers = [('Content-type', 'application/json')]
75         status = '200 OK'
76

```

```

77     query_params = parse_qs(envIRON.get('QUERY_STRING', ''))
78     timestamp = int(query_params.get('timestamp', [0])[0])
79
80     messages, timestamp = self.get_new_messages_from_db(timestamp)
81     data = json.dumps({'messages': messages, 'timestamp': timestamp})
82
83     #print(f"Sending messages: {data}")
84
85     start_response(status, headers)
86     return [data.encode('utf-8')]
87
88     def get_new_messages_from_db(self, timestamp):
89         conn = sqlite3.connect('data.db')
90         cursor = conn.cursor()
91         cursor.execute('SELECT sender, message_text, timestamp FROM messages
92                        WHERE timestamp > ?', (timestamp,))
93         messages = cursor.fetchall()
94         conn.close()
95
96         if messages:
97             timestamp = max(msg[2] for msg in messages)
98         else:
99             timestamp = timestamp
100
101         formatted_messages = [
102             {'sender': msg[0], 'message_text': msg[1].split(':', 1)[1]} if
103             msg[0] else {'sender': 'Guest', 'message_text': msg[1].split(
104                 ': ', 1)[1]} for msg in messages
105         ]
106
107         return formatted_messages, timestamp
108
109     class GetUserIdView(View):
110         def fetch_user_id_from_database(self, username):
111             connection = sqlite3.connect('data.db')
112             cursor = connection.cursor()
113
114             cursor.execute("SELECT user_id FROM users WHERE username=?", (
115                 username,))
116
117             user_id = cursor.fetchone()
118
119             cursor.close()
120             connection.close()
121
122             return user_id[0] if user_id else None
123
124     def response(self, environ, start_response):
125         headers = [
126             ('Content-type', 'application/json'),
127             ('Access-Control-Allow-Origin', 'http://localhost:8000'),
128             ('Access-Control-Allow-Credentials', 'true'),
129         ]

```

```

127 request = Request(envIRON)
128
129 user_id_cookie = request.cookies.get('user_id')
130
131 if user_id_cookie:
132     user_id = user_id_cookie
133 else:
134     user_id = None
135
136 print("Response from /get_user_id:", {"user_id": user_id})
137
138 if user_id is None:
139     print("User ID is None. Cannot fetch messages.")
140     status = '200 OK'
141     data = json.dumps({'user_id': None})
142     start_response(status, headers)
143     return [data.encode('utf-8')]
144 else:
145     print(f"Fetching messages for user_id: {user_id}")
146
147     fetched_user_id = self.fetch_user_id_from_database(user_id)
148
149     if fetched_user_id is not None:
150         user_id = fetched_user_id
151
152     print("Fetched user_id:", user_id)
153
154     status = '200 OK'
155     data = json.dumps({'user_id': str(user_id)})
156     start_response(status, headers + [('Set-Cookie', f'user_id={
157         user_id}; Path=/')])
158     return [data.encode('utf-8')]
159
160 class SendMessageView(View):
161     def response(self, environ, start_response):
162         message, username = self.get_message_and_user_from_request(environ)
163         #print("Data before sending:", message, username)
164
165         if message and username:
166             timestamp = int(time.time())
167             username = self.get_nickname_from_database(username) or 'Guest'
168             full_message = f"{username}: {message}"
169             self.save_message_to_db(full_message, username, timestamp)
170
171             status = '200 OK'
172             headers = [('Content-type', 'application/json')]
173             data = json.dumps({'message': 'Сообщение успешно получено и
174                 сохранено'})
175             #print("Data after sending:", message, username)
176         else:
177             status = '400 Bad Request'
178             headers = [('Content-type', 'application/json')]
179             data = json.dumps({'error': 'Неверное сообщение или пользователь'
180                 })

```

```

178         #print("Data after sending:", message, username)
179
180     start_response(status, headers)
181     return [data.encode('utf-8')]
182
183     def save_message_to_db(self, message, username, timestamp):
184         conn = sqlite3.connect('data.db')
185         cursor = conn.cursor()
186         user_id = self.get_nickname_from_database(username)
187         cursor.execute('INSERT INTO messages (user_id, sender, message_text,
188                                timestamp) VALUES (?, ?, ?, ?)', (user_id, username, message,
189                                timestamp))
188         conn.commit()
189         conn.close()
190
191     def get_message_and_user_from_request(self, environ):
192         try:
193             request_body_size = int(environ.get('CONTENT_LENGTH', 0))
194             request_body = environ['wsgi.input'].read(request_body_size).
195                 decode('utf-8')
196
197             parsed_body = json.loads(request_body)
198
199             message = parsed_body.get('message', '')
200             username = parsed_body.get('username', '') or 'Guest'
201
202             if not message or not username:
203                 raise ValueError("Invalid message or username")
204
205             print(f"Received message: {message}, username: {username}")
206
207             return message, username
208         except ValueError as ve:
209             print(f"ValueError: {ve}")
210             return None, None
211         except Exception as e:
212             print(f"Error while extracting message, username, and user_id: {e
213                 }")
214             return None, None
215
216     def get_nickname_from_database(self, username):
217         conn = sqlite3.connect('data.db')
218         cursor = conn.cursor()
219         cursor.execute('SELECT username FROM users WHERE username=?', (
220             username,))
221         nickname = cursor.fetchone()
222         conn.close()
223
224         if nickname:
225             return nickname[0]
226         else:
227             return None

```

```

227 class RegisterView(TemplateView):
228     template = 'templates/register.html'
229
230     def response(self, environ, start_response):
231         request = Request(environ)
232         if request.method == 'POST':
233             post_data = request.POST
234             username = post_data.get('username', '')
235             password = post_data.get('password', '')
236
237             if username and password:
238                 success = self.register_user(username, password)
239                 if success:
240                     status = '200 OK'
241                     headers = [('Content-type', 'application/json')]
242                     data = json.dumps({'message': 'User registered successfully'})
243                 else:
244                     status = '400 Bad Request'
245                     headers = [('Content-type', 'application/json')]
246                     data = json.dumps({'error': 'Username already exists'})
247             else:
248                 status = '400 Bad Request'
249                 headers = [('Content-type', 'application/json')]
250                 data = json.dumps({'error': 'Invalid username or password'})
251
252             start_response(status, headers)
253             return [data.encode('utf-8')]
254
255     return super().response(environ, start_response)
256
257     def register_user(self, username, password):
258         print(f"Received username reg_us: {username}, password: {password}")
259         conn = sqlite3.connect('data.db')
260         cursor = conn.cursor()
261
262         user_id = uuid.uuid4().hex # Генерируем уникальный идентификатор
263                                   # пользователя
264
265         cursor.execute('SELECT * FROM users WHERE username=?', (username,))
266         existing_user = cursor.fetchone()
267
268         if existing_user:
269             conn.close()
270             return False # Пользователь с таким именем уже существует
271
272         cursor.execute('INSERT INTO users (username, password, user_id)
273                        VALUES (?, ?, ?)', (username, password, user_id))
274         conn.commit()
275         conn.close()
276         return True # Пользователь успешно зарегистрирован
277
278     def get_post_data(self, request, key):
279         try:

```

```

278         data = request.POST.get(key, '')
279         #print(f"Received data for {key}: {data}")
280         return data
281     except Exception as e:
282         print(f"Error while extracting {key}: {e}")
283         return None
284
285 class LoginView(TemplateView):
286     template = 'templates/login.html'
287
288     def response(self, environ, start_response):
289         if environ['REQUEST_METHOD'] == 'POST':
290             request = Request(environ)
291             post_data = parse_qs(request.body.decode('utf-8'))
292             username = post_data.get('username', [''])[0]
293             password = post_data.get('password', [''])[0]
294
295             if username and password:
296                 user_id = self.authenticate_user(username, password)
297                 if user_id:
298                     status = '200 OK'
299                     headers = [('Content-type', 'application/json')]
300                     data = json.dumps({'redirect': '/', 'user_id': str(
301                         user_id[0])})
302                     start_response(status, headers)
303                     return [data.encode('utf-8')]
304                 else:
305                     status = '401 Unauthorized'
306                     headers = [('Content-type', 'application/json')]
307                     data = json.dumps({'error': 'Invalid username or password'})
308                     start_response(status, headers)
309                     return [data.encode('utf-8')]
310             else:
311                 status = '400 Bad Request'
312                 headers = [('Content-type', 'application/json')]
313                 data = json.dumps({'error': 'Invalid username or password'})
314                 start_response(status, headers)
315                 return [data.encode('utf-8')]
316         else:
317             return super().response(environ, start_response)
318
319     def authenticate_user(self, username, password):
320         conn = sqlite3.connect('data.db')
321         cursor = conn.cursor()
322
323         #print(f"Received username: {username}, password: {password}")
324
325         cursor.execute('SELECT user_id FROM users WHERE username=? AND
326                        password=?', (username, password))
327         user_id = cursor.fetchone()
328
329         conn.close()

```

```

329         #print("Authenticated user_id:", user_id)
330
331         return str(user_id[0]) if user_id else None
332
333
334
335     def get_post_data(self, request):
336         try:
337             data = request.POST
338             return data
339         except Exception as e:
340             print(f"Error while extracting POST data: {e}")
341             return None

```

#### mimes.py

```

1 mimes = {
2     '.html': 'text/html',
3     '.css': 'text/css',
4     '.js': 'text/javascript'
5 }
6
7 def get_mime(file_name):
8     """
9     Возвращает тип содержимого по имени файла
10    """
11    for key in mimes.keys():
12        if file_name.find(key) > 0:
13            return mimes[key]
14    return 'text/plain'

```

#### routes.py

```

1 from views import *
2
3 routes = {
4     '/static/': View,
5     '/$': IndexView,
6     '/get_user_id': GetUserIdView,
7     '/get_messages': GetMessageView,
8     '/send_message': SendMessageView,
9     '/register': RegisterView,
10    '/login': LoginView
11 }
12
13 def route(url):
14     """
15     Преобразовывает URL в путь к файлу в соответствии с определенными
16     маршрутами.
17    """
18    for key in routes.keys():
19        if url.startswith(key):
20            return routes[key] + url[len(key):]
21    return url

```

#### run.py

```
1 from app import app
2 from waitress import serve
3
4 if __name__ == '__main__':
5
6     from waitress import serve
7     serve(app, host='0.0.0.0', port=8000)
```