Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Реализация мессенджера в веб-сервисе для помощи бездомным животным

Карасева Елизавета Олеговна

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
старший преподаватель кафедры системного программирования, к.т.н., Литвинов Ю. В.

Консультант:
инженер-разработчик ООО «КНС Групп», Шеремет И. Д.

Санкт-Петербург
2024

Оглавление

Введение

В нашей стране как никогда актуальна проблема бездомных животных. Волонтерское движение в сфере зоозащиты объединяет неравнодушных людей, но систематизация информации и координация действий между ними часто затруднена. А ведь при поиске и оказании помощи животным важно поддерживать связь не только между волонтерами, например, решившими организовать отлов стаи собак, но и между специалистами из разных областей: нормативно-правовой или ветеринарной.

Веб-сервис, разрабатываемый компанией YADRO, призван объединить информацию по вопросам пропажи и находки животных, их отлова, лечения и последующего пристройства. Одним из связующих элементов сервиса должен стать мессенджер: личные чаты помогут пользователям обращаться напрямую к нужным специалистам, а групповые — организовать командные действия для спасения животных или совместной работы с ветеринарами и передержками. Коммуникация должна быть доступна для всех потребителей продукта, вне зависимости от их возраста и уровня цифровой грамотности, поэтому организация общения пользователей путем перехода в мессенджеры из других социальных сетей исключается.

На данный момент в проекте реализованы статические страницы чатов, соответствующие дизайну, но функциональность к ним отсутствует. В перспективе отсутствие данного раздела в веб-сервисе приведет к необходимости использовать сторонние приложения для коммуникации, что замедлит координацию и может стать критичным для оперативной помощи животным. Для обеспечения непрерывной коммуникации между пользователями было решено самостоятельно реализовать логику чатов внутри платформы, так как найти и использовать готовый браузерный чат, написанный с помощью такого же стека технологий, как и используемый в проекте, в open source не удалось.

В рамках учебной практики предполагается разработка мессенджера с поддержкой как личных, так и групповых чатов и интеграция его

в веб-сервис. Также будут решаться задачи добавления логики к статическим страницам и синхронизации сообщений в реальном времени.

1. Постановка задачи

Целью работы является разработка основной функциональности мессенджера в веб-сервисе для помощи бездомным животным. Для её выполнения были поставлены следующие задачи:

1. Провести обзор технологий для моментальной отправки сообщений и выбрать подходящие для проекта библиотеки.
2. Добавить поддержку мессенджера на стороне базы данных, а именно:
    - спроектировать и добавить необходимые для работы мессенджера таблицы;
    - настроить миграции для корректного обновления структуры проекта.
3. Спроектировать и реализовать программный интерфейс для мессенджера.
4. Поддержать чаты на фронтенде.
5. Обновить документацию проекта.
6. Протестировать добавленную функциональность.
7. Провести демонстрацию и получить отзыв владельца продукта.

2. Обзор

Перед тем, как приступить к реализации мессенджера, необходимо было выделить требования, определить стек технологий и выбрать способ реализации на основе обзора существующих инструментов для организации обмена сообщениями.

2.1. Требования к мессенджеру

После обсуждения задачи с владельцем продукта были выявлены основные требования к мессенджеру.

1. Возможность общения между пользователями платформы путем моментальной отправки и принятия сообщений.
2. Возможность создания чатов с несколькими пользователями.
3. Поддержка разделения чатов по типам (ролям пользователей в системе).

2.1.1. Моментальная отправка и принятие сообщений

Основной задачей является организация общения между пользователями без излишней нагрузки на сервер и базу данных. При этом важно минимизировать задержки в процессе отправки и получения сообщений, а также отслеживать актуальное состояние чатов, чтобы вовремя узнавать о новых сообщениях.

2.1.2. Чаты с несколькими пользователями

В дополнение к индивидуальным беседам, необходимо предусмотреть возможность создания групповых чатов, что улучшит координацию действий между участниками. Необходимо отличать сообщения разных участников друг от друга.

2.1.3. Разделение чатов по типам

Учитывая различные роли пользователей в системе, необходимо организовать разделение чатов в зависимости от типа специалиста: отловщик, человек, предоставляющий передержку, или ветеринар. Также следует учесть возможность создания чата как отклика на объявления о пропаже или нахождении животного.

2.2. Используемые технологии

Реализация мессенджера выполнялась на основе технологий, используемых в проекте по требованию заказчика.

1. Клиентская часть написана на языке программирования TypeScript [15] с использованием фреймворка для проектирования приложений Angular [28]. TypeScript обеспечивает статическую типизацию, что помогает отлавливать ошибки на этапе написания кода. Angular, в свою очередь, позволяет создавать сложные веб-приложения, предоставляя инструменты для создания динамических пользовательских интерфейсов.

2. Для серверной части использовался язык Go [2] с использованием библиотеки Fiber [3]. Go был выбран из-за способности легко обрабатывать большое количество одновременных запросов, что идеально подходит для веб-сервисов. Fiber же изначально построен поверх горутин — легковесных потоков, управляемых средой управления Go, что делает его нативно асинхронным и дает возможность обрабатывать множество запросов параллельно.

3. Для хранения и манипуляций с данными использовалась система управления базами данных PostgreSQL [10]. Она известна своей надежностью, а также предоставляет большой набор функции для работы с данными.

2.3. Обзор методов мгновенной передачи сообщений

В приложениях реального времени использования обычных HTTP-методов [30] становится недостаточно. Данные методы работают по модели запрос-ответ, поэтому для получения новых сообщений клиенту придется раз в несколько секунд опрашивать сервер. Такие запросы создают избыточный сетевой трафик, а также задержки по времени, связанные с обработкой получаемых данных и ответов на них.

Для организации обмена сообщениями в реальном времени существует несколько популярных подходов.

- Long Polling [29]. Клиент посылает запрос на сервер, а тот держит соединение открытым до тех пор, пока не появятся новые данные. Как только это происходит, сервер отправляет их клиенту и закрывает соединение, а клиент тут же отправляет новый запрос, и процесс повторяется. Подход прост в реализации, но не так эффективен, как более современные технологии.

- WebSockets [18]. Это протокол связи поверх HTTP. Данная технология обеспечивает полнодуплексный канал связи, то есть участники соединения могут одновременно отправлять данные друг другу. Соединение между клиентом и сервером устанавливается один раз и поддерживается открытым (рис. 1), а данные передаются только при необходимости, что снижает объем передаваемого трафика. Также данный протокол позволяет отправлять сообщения подмножеству клиентов без задержек и дополнительных запросов. Бывает сложно определить, открыто ли соединение в данный момент времени, или уже нет, но для обработки таких случаев есть свои способы. Веб-сокеты может быть сложно встроить в существующую систему, так как часто в системах стоят прокси-сервера, блокирующие соединения, идущие не по HTTP.

- Server-Sent-Events (SSE) [12]. Это технология, позволяющая серверу отправлять свои обновления клиенту после инициализации им соединения через однонаправленное соединение по протоколу

HTTP. SSE предназначена для однонаправленной коммуникации и идеально подходит для ситуаций, где клиент должен обновляться в реальном времени, но не отправлять при этом данные на сервер, например, при обновлении ленты новостей в приложении.

- WebTransport [19]. Этот API использует протокол HTTP/3 QUIC [8] и поддерживает различные варианты передачи данных, в том числе полнодуплексное соединение. Он подходит для приложений с высокопроизводительными сетевыми взаимодействиями, однако по состоянию на октябрь 2024 года WebTransport не поддерживается в некоторых браузерах. И сам API, и библиотека для работы с ним в языке Go находятся в состоянии working draft.

- WebRTC [16] — стандарт, описывающий передачу потоковых аудиоданных, видеоданных и контента между браузерами в режиме реального времени. Имеет открытый исходный код. Изначально, цель WebRTC — создание видеоконференций без использования какого-либо дополнительного сервера. WebRTC не может использоваться в качестве замены вышеперечисленных технологий, ведь для его непосредственной реализации понадобится использовать сигнальный сервер на SSE, WebTransport или веб-сокетах.

Для создания чата было решено использовать соединения WebSocket, так как они подходят для сценариев, где предполагаются частые обновления, низкая задержка и поддержка одновременной отправки данных, в отличие от SSE, более эффективны по сравнению с long polling и не требуют настройки отдельного сервиса, как WebRTC. Выбор в пользу веб-сокетов подтверждает и сравнение технологий по разным критериям, в том числе по производительности (рис. 2), проведенное авторами статьи [31].

2.4. Обзор инструментов для работы с вебсокетами

Серверная часть веб-сервиса написана на Go и Fiber, а клиентская — на Angular и TypeScript. Так как веб-сокеты необходимо поддерживать

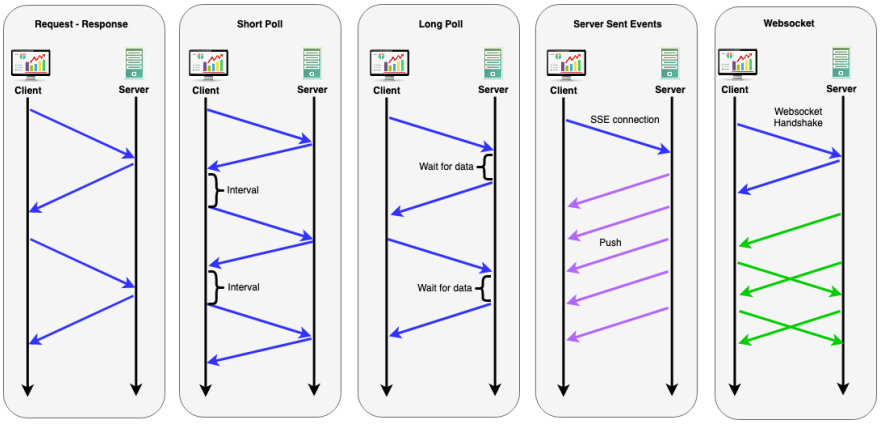

Рис. 1: Сравнение методов мгновенной передачи сообщений, https://dzone.com/articles/real-time-stock-data-updates-with-websockets-using

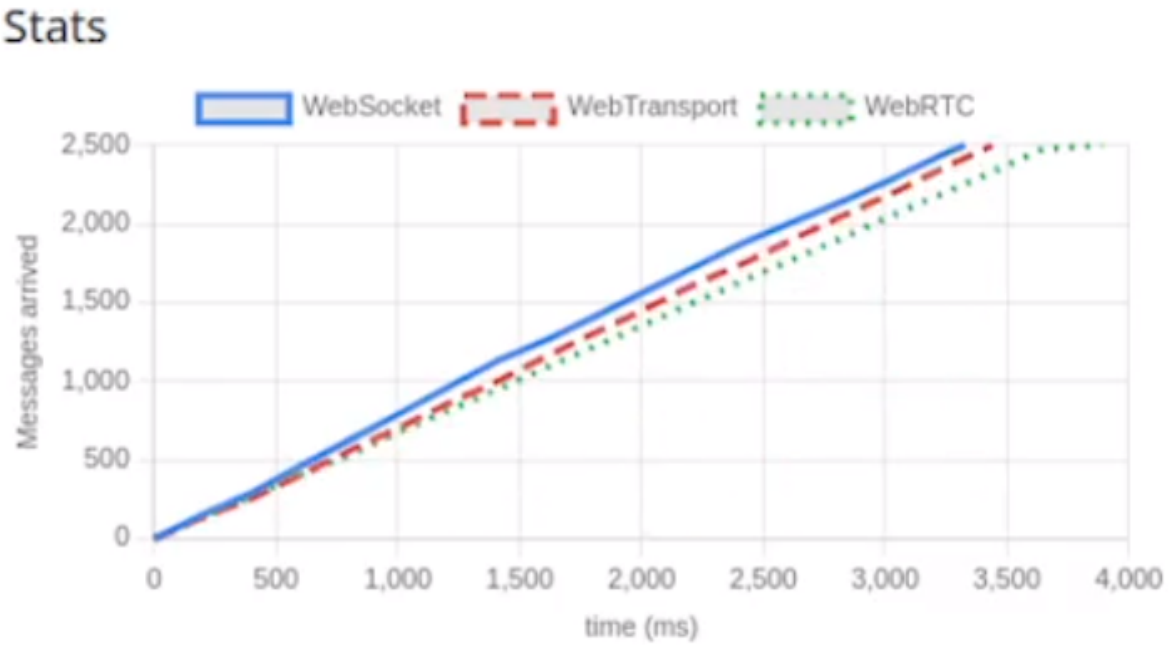

Рис. 2: Сравнение показателей производительности WebSockets, WebRTC и WebTransport, https://rxdb.info/articles/websockets-sse-polling-webrtc-webtransport.html

и на клиенте, и на сервере, были рассмотрены способы взаимодействия с ними для обеих частей проекта.

2.4.1. Пакеты для Go

При написании приложения на языке программирования Go предполагается подключение в него пакетов, обладающих необходимой функциональностью. Благодаря встроенным в язык инструментам для поддержки конкурентности, Go предоставляет много возможностей для работы с WebSocket. Ниже представлены некоторые из самых популярных пакетов:

- `Gorilla WebSocket` [25] — один из самых распространенных пакетов для работы с веб-сокетами на языке Go. Предоставляет полный набор функций для установки соединений, чтения и записи сообщений, поддерживает как клиентскую, так и серверную часть. Используется в множестве проектов, так как надежен и прост в использовании.

- `x/net/websocket` [26] — часть стандартного набора пакетов Go, не получила широкого распространения из-за ограниченных возможностей по сравнению с другими библиотеками. По состоянию на октябрь 2024 года можно считать его deprecated[1]. Предполагаемая альтернатива — `coder/websocket` [24], позволяет контролировать таймауты, сроки жизни соединений и обработку ошибок через контексты Go.

- `gobwas/ws` [27] — особенность этого пакета — низкоуровневая поддержка веб-сокетов. Производительный, но сложен в настройке.

- `centrifugal/centrifuge` [4] — еще один пакет для обмена сообщениями в реальном времени, построенный на основе веб-сокетов. Подходит для масштабируемых систем с большим количеством соединений. Предоставляет официальные клиентские SDK на базе

[1] x/net/websocket: remedy neglect; merge with gorilla websocket? — URL: https://github.com/golang/go/issues/18152 — Дата обращения: 08.10.24.

Centrifuge для обеспечения двунаправленной связи между клиентом и сервером, однако Angular среди них не представлен.

- `go-socket.io` [5] — библиотека для работы с Socket.IO [13] на Go, реализующая многие особенности этого фреймворка (например, комнаты и пространства имен).

- `fiber/websocket` [20] — это расширение фреймворка Fiber с простым и удобным API. Позволяет работать с веб-сокетами без потерь в производительности. Работает в паре с клиентской частью, которая может быть реализована через любую из клиентский библиотек.

Веб-сервис, в рамках которого реализуется чат, написан с использованием Fiber, поэтому выбор `fiber/websocket` является естественным:

- использование интегрированного пакета снижает сложность проекта, поскольку не требуется подключение внешних библиотек;

- он позволяет комбинировать Websocket-соединения с другими промежуточными слоями Fiber;

- `fiber/websocket` упрощает настройку маршрутов для обработки веб-сокетов.

Что касается остальных рассмотренных библиотек, обычно они работают в связке с `net/http` — еще одним популярным пакетом для работы с сетью в Go. Для использования их вместе с Fiber потребуется довольно сложная дополнительная настройка, например, в Gorilla придется переписать обработку маршрутов и контексты.

2.4.2. Библиотеки для Angular

На TypeScript и Angular также существует несколько библиотек для работы с вебсокетами, каждая из которых имеет свои преимущества.

- `RxJS + WebSocketSubject`. Для управления потоками данных в Angular часто используется RxJS [1] — библиотека для работы с

асинхронным кодом. WebSocketSubject [11] из RxJS предоставляет обертку для работы с веб-сокетами, но не обладает многими возможностями, реализованными в других библиотеках.

- `ngx-socket-io` [23]. Использует в качестве основы Socket.IO, что дает много дополнительных возможностей, например, автоматическое переподключение при разрыве связи. Но из-за разнообразной функциональности она достаточно тяжеловесна. Socket.IO использует веб-сокеты, но добавляет дополнительные метаданные к каждому пакету, поэтому для осуществления взаимодействий с серверной частью она, также, как и клиент, должна быть реализована на Socket.IO.

- `WebSocket API (MDN)` [17]. Он позволяет клиентским приложениям устанавливать двухстороннее соединение с сервером через протокол WebSocket, прост в использовании и уже встроен в современные браузеры. Имеет не так много возможностей для работы с событиями, чем другие библиотеки.

Так как в данном веб-сервисе вся логика работы с веб-сокетами выполнена в серверной части, было решено не усложнять клиентскую часть и использовать нативный WebSocket API, ведь он обеспечивает двустороннее соединение без необходимости внедрять дополнительные зависимости.

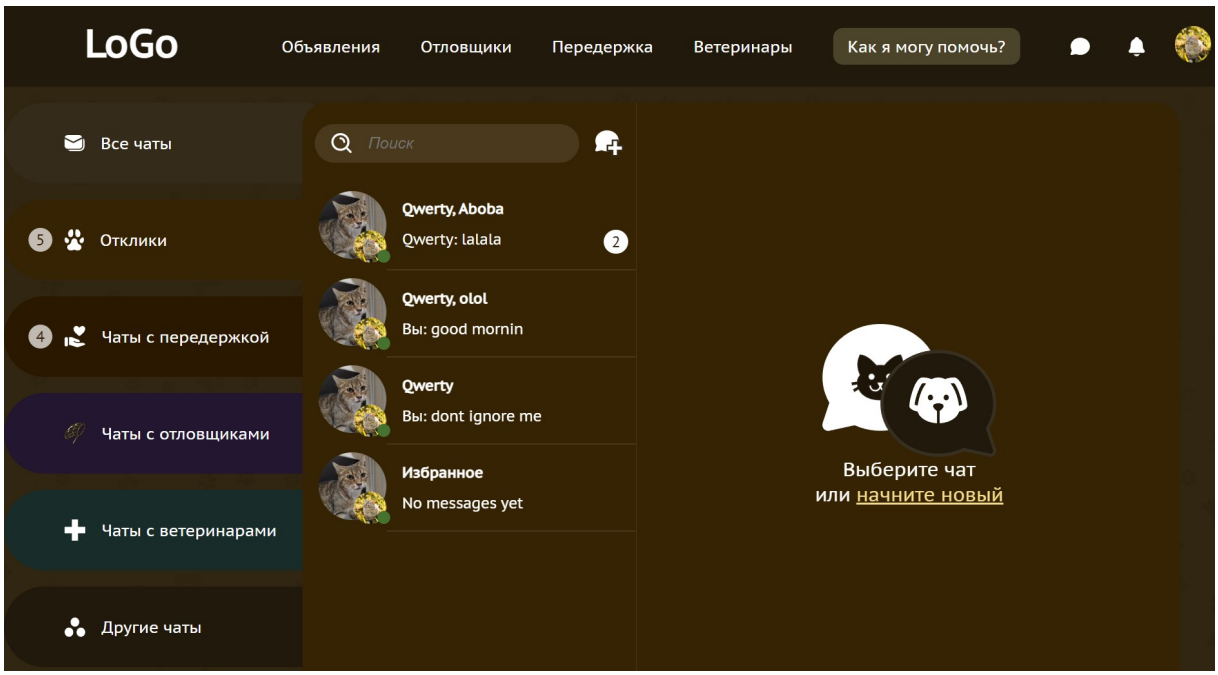

Рис. 3: Макет общей страницы чатов

3. Реализация

Работа проходила в несколько этапов: изучение дизайна, создание таблиц для базы данных, продумывание и реализация методов API, интеграция их с другими частями сервиса, создание тестов, поддержка чатов на фронтенде.

3.1. Дизайн интерфейса

Основной дизайн страниц был предложен и согласован еще до начала работы. Мессенджер представлен страницей `chats-page` (рис. 3), слева находится панель разделения чатов по типам, далее идет список всех чатов. После нажатия на какой-либо чат из списка он открывается в поле справа. Основная верстка страницы была сделана заранее, однако в рамках данной работы были исправлены недочеты, связанные с отображением текста сообщений, а также было добавлено отображение имен над сообщениями в чатах. В связи с отсутствием UI для части методов, необходимых для работы с мессенджером, они были реализованы только на бекенде.

3.2. Макет базы данных

Для взаимодействий с чатами были продуманы и добавлены в базу данных приложения следующие таблицы (рис. 4):

- `chats` отвечает за информацию о чате, а именно, содержит его тип, время создания и ссылку на пост, если чат является откликом на чье-то объявление;
- `chat_members` содержит информацию об участниках чата: когда и куда человек был добавлен, какая у него роль в этом чате;
- `messages` помимо текста сообщения показывает, кто именно отправил сообщение, в какой чат, в какое время это было сделано, когда сообщение последний раз изменили;
- `message_read` хранит данные о том, во сколько и кем прочитано определенное сообщение;
- также в рамках работы велось взаимодействие с таблицами `users` и `posts`, которые уже находились в базе данных ранее.

Актуальная документация базы данных создана с использованием инструмента dbdocs [22] и представлена по ссылке ниже[2].

Были созданы миграции для добавления в базу данных необходимых таблиц. Пулл-реквесты с изменениями базы данных отправлены в репозиторий веб-сервиса и слиты в ветку master[3].

3.3. API

В проект были добавлены сервисы чатов, сообщений и участников чата. В основу написанных методов API легли операции CRUD [21], только вместо безвозвратного удаления данных из базы было решено помечать их как deleted.

[2]https://dbdocs.io/kleo-53/SOS-kotopes

[3]https://github.com/kotopesp/sos-kotopes/pull/30 и https://github.com/kotopesp/sos-kotopes/pull/47

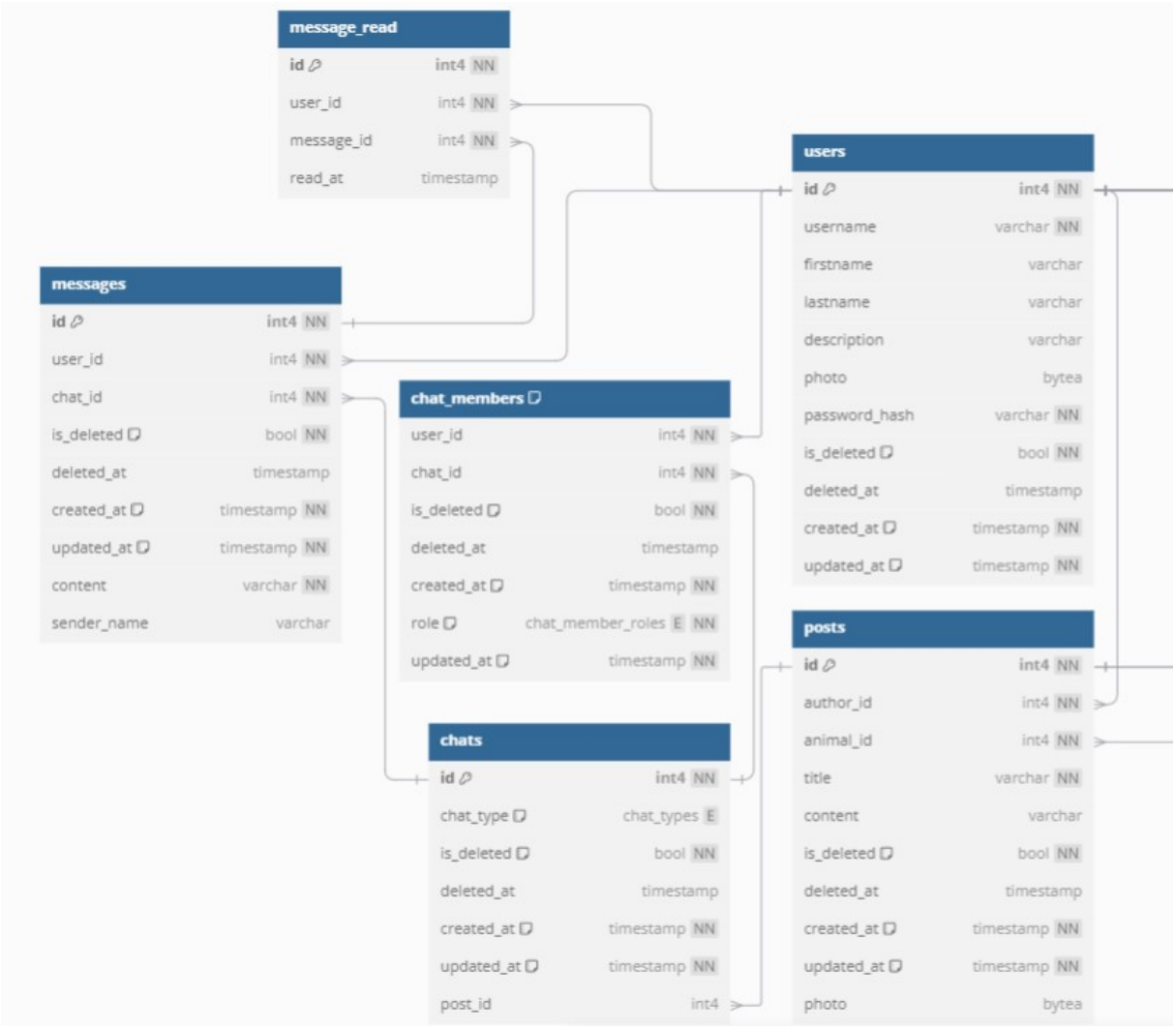

Рис. 4: Схема задействованных таблиц из базы данных

Так как чаты доступны только авторизованным пользователям, в каждом методе проверяется наличие токена авторизации в запросе и также ведется работа с id текущего пользователя, получаемого из токена. Методы возвращают ответ успеха (`200 OK` или `201 Created`), если запланированное действие выполнено, либо одну из следующих ошибок в случае сбоев:

- `400 Bad Request`, если у какого-либо значения в запросе указан неверный тип данных;
- `401 Unauthorized`, если заголовки запроса не содержат токен авторизации, либо срок действия токена закончился;
- `500 Internal Server Error`, если при обработке данных произошла ошибка сервера.

3.3.1. Сервис чатов

Данный сервис отвечает за модификацию чатов. Были добавлены методы:

- `GET /chats/{chat_id}` — возвращает информацию о конкретном чате, включая список пользователей, связанных с этим чатом, и последнее сообщение, отправленное в этот чат.
- `GET /chats` — возвращает список всех чатов, доступных для текущего пользователя. Чаты можно фильтровать по типу, передавая query-параметр chat_type.
- `POST /chats` — создает новый чат с выбранными пользователями, принимая в теле запроса список идентификаторов пользователей, которые должны быть добавлены в чат. Если чат с таким набором пользователей уже существует, возвращается ошибка `409 Conflict` и информация о существующем чате.
- `DELETE /chats/{chat_id}` — удаляет чат по идентификатору для всех его участников. Удалить чат может только его участник.

3.3.2. Сервис сообщений

Сервис работает с сущностями сообщений и количеством прочитанных сообщений. Идентификатор текущего пользователя берем из токена, переданного в запросе.

- `GET /chats/{chat_id}/unread` — возвращает количество непрочитанных сообщений для текущего пользователя в указанном чате.
- `PATCH /chats/{chat_id}/unread` — отмечает все сообщения в указанном чате как прочитанные для текущего пользователя.
- `GET /chats/{chat_id}/messages` — возвращает все сообщения в указанном чате для текущего пользователя. Есть возможность вывести сообщения в обратном по времени создания порядке, а также найти сообщения, содержащие определенный (переданный в запросе) текст.
- `POST /chats/{chat_id}/messages` — создает новое сообщение от лица текущего пользователя в указанном чате.
- `PATCH /chats/{chat_id}/messages/{message_id}` — редактирует сообщение. В теле запроса передается новый текст. Редактировать можно только свои сообщения.
- `DELETE /chats/{chat_id}/messages/{message_id}` — удаляет указанное сообщение из чата. Удалять можно только свои сообщения.

3.3.3. Сервис участников чатов

Этот сервис отвечает за манипуляции с участниками чатов. Получать информацию о пользователях можно только тем, кто сам состоит в этом же чате.

- `GET /chats/{chat_id}/members` — возвращает список участников чата, которые на данный момент времени не удалены из него.

- `POST /chats/{chat_id}/members/{user_id}` — добавляет участника в беседу и возвращает информацию о нем.
- `PATCH /chats/{chat_id}/members/{user_id}` — меняет роль участника в чате на новую, переданную в теле запроса. Возвращает обновленные данные об участнике.
- `DELETE /chats/{chat_id}/members/{user_id}` — удаляет участника из чата. Удалить участника может только другой участник чата.

Документацию с подробным описанием методов и примерами ответов сервера можно найти по ссылке[4]. Информация по вышеописанным методам представлена в разделе Conversations.

3.4. WebSocket сервис

В основе мгновенной передачи сообщений лежат веб-сокет-соединения. В Go такие соединения можно асинхронно, без блокировок и сильных накладных расходов обрабатывать в горутинах, а данные между ними передавать через каналы.

В проекте это было реализовано в виде типа WebSocketManager, который хранит ассоциативный массив идентификаторов чатов, где для каждого чата хранится список активных соединений, и мьютекс для синхронизации. Экземпляр такого типа должен реализовывать методы добавления и удаления соединения из словаря, отправки сообщения по всем активным соединениям чата, а также эндпоинты управления логикой для каждого соединения.

То есть, при создании сервера на отдельные маршруты вида `/ws/{chat_id}` добавляется middleware для инициализации и регистрации веб-сокет соединения в конкретном чате, а также эндпоинт, предназначенный для обработки веб-сокет сообщений (листинг 1). При этом нужно вовремя сохранять и подгружать в чаты данные из базы.

[4]https://mocokbelcr.apidog.io/

Листинг 1: Упрощенный пример взаимодействия с WebSocket соединениями при инициализации приложения Fiber

```
    func main() {
    app := fiber.New() // Create app

    // Create webSocket manager
    wsManager := NewWebSocketManager()

    // Endpoint for connections
    app.Use("/ws/:chatID", wsManager.HandleWebSocket)

    // Processing websocket requests
    app.Get("/ws/:chatID",
        websocket.New(wsManager.WebSocketEndpoint))

    log.Fatal(app.Listen(":3000")) // Start app
}
```

Когда пользователь заходит в чат, туда подгружаются сообщения, сохраненные в базе, и открывается веб-сокет соединение. Теперь мы отправляем, получаем и сразу же отрисовываем на экране сообщения до тех пор, пока соединение не закроется. Это произойдет, если пользователь выйдет из чата или закроет страницу. При следующем заходе в чат старые сообщения отобразятся уже из базы, а веб-сокет соединение откроется заново.

Сообщения моментально отображаются у всех пользователей, с кем в этом чате есть активное соединение. Если же пользователь выйдет из чата, о новых сообщениях он узнает чуть позже, когда они подгрузятся в базу и отобразятся в виде непрочитанных после обновления панели чатов слева.

3.5. Поддержка чатов на фронтенде

Полностью статические страницы на фронтенде, отвечающие за клиента, были модифицированы в соответствии с логикой серверной части приложения.

3.5.1. Сервисы

Приложения на Angular структурированы таким образом, что в отдельной папке собраны файлы, предоставляющие логику для взаимодействия с данными, в том числе с API (с помощью HttpClient [9]) и WebSocket. В ходе работы были добавлены сервисы:

1. `websocket-service` подключается к чату, инициируя на клиенте веб-сокет соединение, которое ловит и обрабатывает сервер. Также сервис использует это соединение для передачи данных серверу. После отправки сообщения через веб-сокет сервис отправляет уже обычный http-запрос на добавление сообщения в базу данных.

2. `chat-service` собирает данные со страницы чатов и отправляет запросы к серверу в зависимости от задачи:

    - получение всех чатов, они отображаются на панели слева вместе с последним сообщением, именем его отправителя, а также количеством непрочитанных сообщений;
    - получение чата по идентификатору при выборе пользователем одного из чатов в панели слева, при этом все сообщения отображаются на экране, а внизу появляется поле ввода нового сообщения;
    - получение и отрисовка всех сообщений чата, вид отображения зависит от конкретного пользователя: свои сообщения справа и без имени пользователя, чужие — слева, с именами отправителей;
    - генерация названия чата для конкретного пользователя по находящимся в чате участникам;

- создание чата с выбранными пользователями, либо перенаправление на существующий с таким же набором пользователей;
- обновление информации о последнем сообщении чата при получении в него сообщения, чтобы чат в панели слева обновлялся без задержек;
- получение списка пользователей, с которыми можно начать чат, в перспективе это те люди, кого текущий пользователь добавил в избранное;
- получение числа непрочитанных сообщений для конкретного чата;
- прочтение сообщений, если пользователь заходит в чат.

3.5.2. Компоненты

К UI чатов была добавлена логика для отображения информации, полученной от запросов, отправленных сервисами к серверу. Страница чатов содержит в себе все остальные перечисленные ниже компоненты.

1. На страницу чатов были добавлены поля, отвечающие за все доступные чаты, активный чат, полученные сообщения и выбранных для создания чата пользователей. Также были добавлены функции для вызова методов сервисов по нажатиям на соответствующие кнопки и элементы.
2. В компоненту чата была добавлена настройка отображения последнего сообщения.
3. В компоненту сообщения было добавлено отображение имени отправителя, а также текста и времени отправки сообщения.
4. В компоненте добавления пользователя в чат была усовершенствованна логика добавления и удаления пользователей в выбранные.

Листинг 2: Интерфейс сообщения на фронтенде

```
export interface Message {
    id?: number,
    user_id: number,
    chat_id: number,
    message_content: string,
    created_at: Date,
    is_user_message: boolean,
    time: string,
    sender_name: string,
}
```

3.5.3. Интерфейсы

В приложение были добавлены интерфейсы, соответствующие структурам, отправляемым и приходящим с бекенда: `ResponseUser` для выбора пользователей при добавления в чат, а также `Chat`, `ChatMember` и `Message` (листинг 2).

3.6. Структура проекта

3.6.1. Backend

Получение и валидация данных, вызов методов сервисов и обработка ошибок находится в папке `backend/internal/controller/http`. Модели для обработки данных в формате JSON можно найти в `backend/internal/controller/http/model`. Также в папку `backend/internal/core` были добавлены типы, соответствующие таблицам из базы данных, чтобы корректно взаимодействовать со значениями, получаемыми или отправляемыми в базу. В `backend/internal/service` содержится работа с данными и передача их на более низкий уровень обработки. В `backend/internal/store` описывается взаимодействие с базой данных.

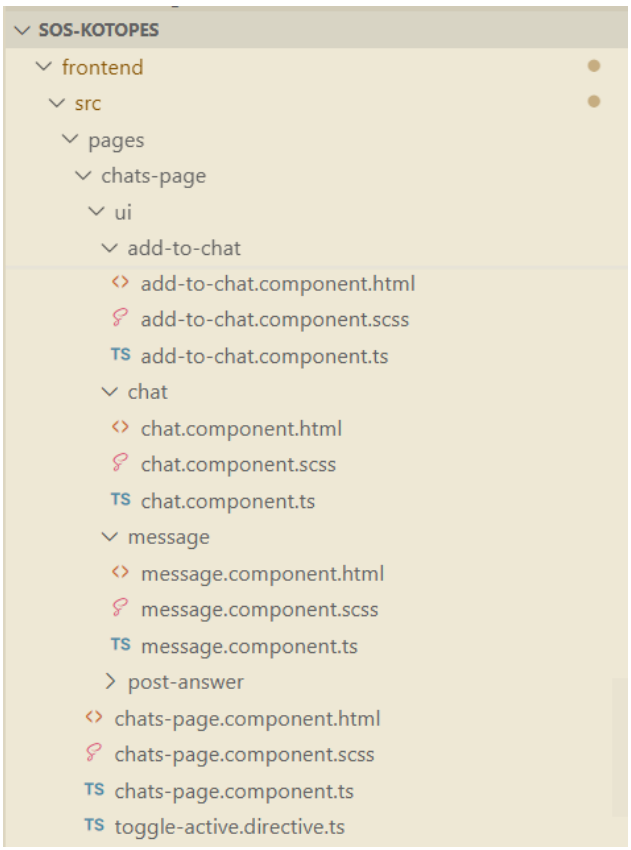

Рис. 5: Структура проекта на фронтенде

3.6.2. Frontend

Интерфейсы находятся в папке `frontend/src/model`. Сервисы лежат по адресу `frontend/src/services`. Компоненты страниц находятся в разделе `frontend/src/pages/chats-page`. Каждая компонента представляет независимый блок UI со своим HTML-шаблоном, стилями, заданными в файлах CSS/SCSS, а также логикой, представленной в файлах .ts (рис. 5).

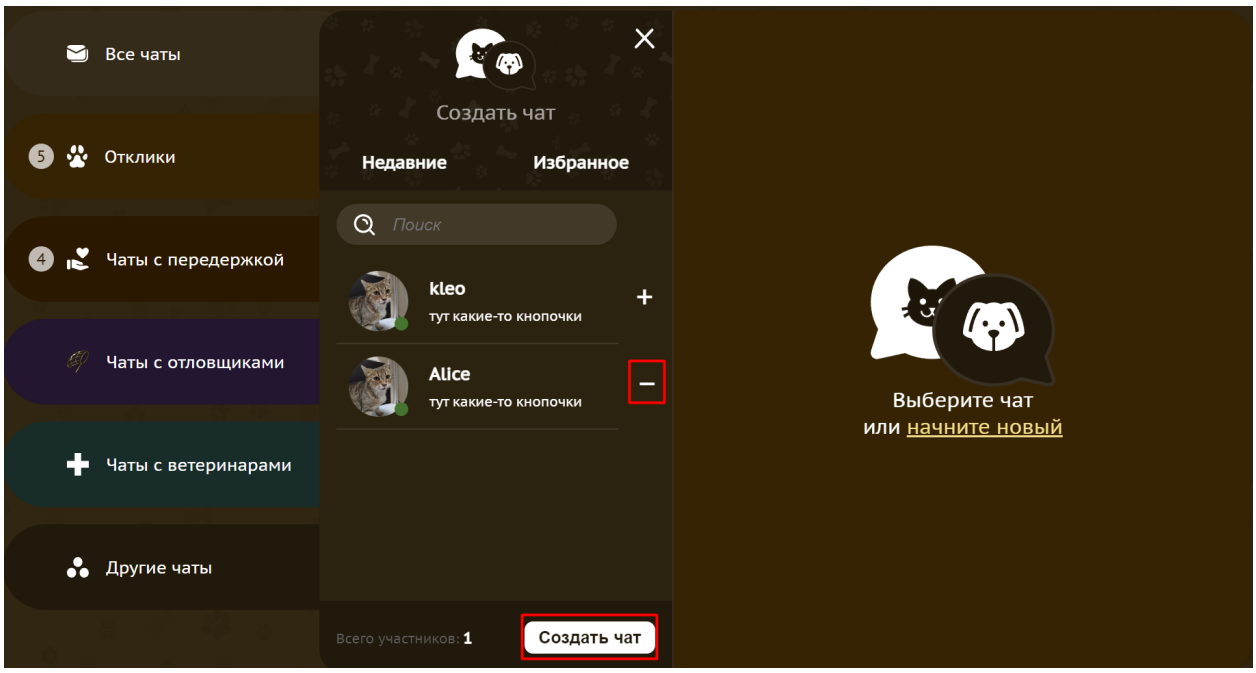

Рис. 6: Выбор пользователей для создания чата

4. Описание работы

Чтобы перейти на страницу чатов, пользователь должен быть авторизирован. После авторизации пользователь попадает на главную страницу, откуда по нажатию на иконку сообщений происходит перенаправление на страницу чатов. Теперь можно либо создать новый чат, либо открыть существующий. Для создания чата требуется нажать на соответствующую кнопку. После этого высветится окошко с пользователями для выбора. При нажатии на пользователя минус справа от имени заменяется на плюс, а в счетчик количества участников чата добавляется единица (рис. 6). Если пользователь выбран, а мы передумали добавлять его в чат, нажимаем на плюс, он меняется на минус, из числа участников чата вычитается единица. Можно создать чат с самим собой, он будет отмечаться как «Избранное».

Создадим чат с пользователем Alice и отправим пару сообщений. То, как чаты выглядят со стороны Alice, можно увидеть на рис. 7, зайдем в чат, прочитаем сообщения и что-нибудь ответим (рис. 8).

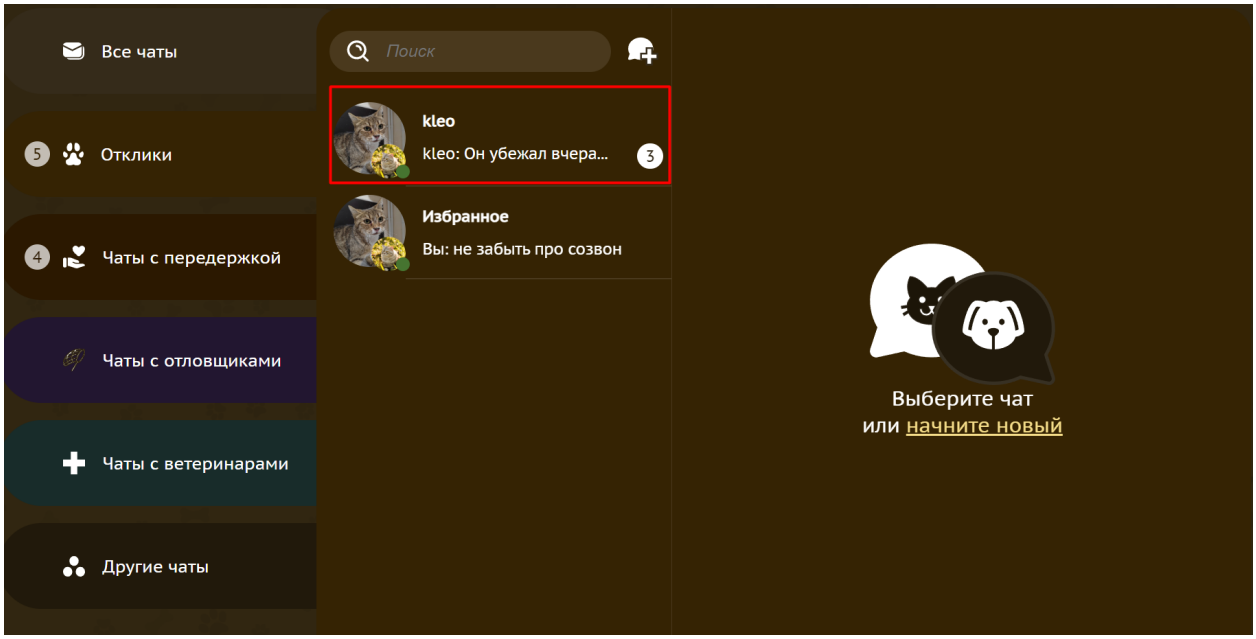

Рис. 7: Непрочитанные сообщения от лица Alice

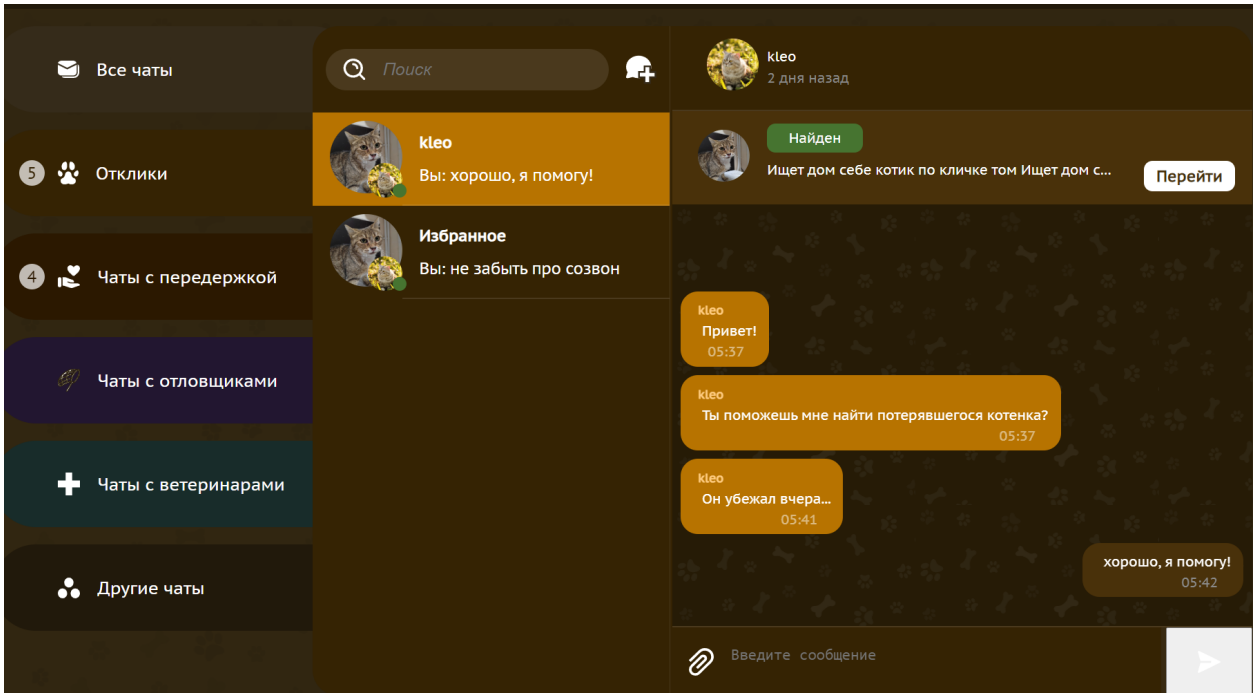

Рис. 8: Прочитали и ответили на сообщения в чате

5. Тестирование и апробация

Для тестирования бекенда приложения были использованы пакеты для тестов в Go: `testify` [6] и `mockery` [7]: сгенерированные mock-объекты использовались в качестве заглушек для тестирования экземпляра приложения.

Пулл-реквест[5] с реализацией мессенджера был предложен в репозиторий проекта, на момент 07.10.2024 тесты и линтер в CI проходят.

Была проведена демонстрация добавленной функциональности владельцу продукта, получен положительный отзыв. Также среди команды разработчиков веб-сервиса для помощи бездомным животным была проведена апробация с использованием методики System Usability Scale (SUS) [14]. Шкала SUS помогает оценить субъективные удобства использования системы, оценка по ней лежит в диапазоне от 0 до 100 баллов, среднее значение — 68 баллов. Средняя оценка мессенджера по SUS составила 91,3. Этот результат свидетельствует о высоком уровне удобства использования мессенджера.

[5]https://github.com/kotopesp/sos-kotopes/pull/8

Заключение

В результате работы над практикой была разработана основная функциональность мессенджера в веб-сервисе для помощи бездомным животным. В ходе выполнения были выполнены следующие задачи:

1. Проведен обзор технологий для моментальной отправки сообщений, выбраны подходящие для проекта библиотеки.
2. Добавлена поддержка мессенджера на стороне базы данных.
3. Спроектирован и реализован программный интерфейс для мессенджера.
4. Добавлена поддержка чатов на фронтенде.
5. Обновлена документация проекта.
6. Добавленная функциональность протестирована.
7. Демонстрация проведена, отзыв продукт овнера получен.

С реализацией можно ознакомиться в Github-репозитории https://github.com/kotopesp/sos-kotopes в ветке `feature/chats`, разработка ведется под аккаунтом kleo-53. Пулл-реквесты с реализацией:

- основная функциональность мессенджера:
  - https://github.com/kotopesp/sos-kotopes/pull/8
- миграции для базы данных:
  - https://github.com/kotopesp/sos-kotopes/pull/30
  - https://github.com/kotopesp/sos-kotopes/pull/47
- обновленная документация находится по ссылкам ниже:
  - API: https://mocokbelcr.apidog.io/
  - база данных: https://dbdocs.io/kleo-53/SOS-kotopes

Список литературы

[1] Angular и RxJS. — URL: https://angdev.ru/archive/angular9/angular-and-rxjs (дата обращения: 4 октября 2024 г.).

[2] Documentation - The Go Programming Language. — URL: https://go.dev/doc/ (дата обращения: 4 октября 2024 г.).

[3] Fiber. — URL: https://docs.gofiber.io/ (дата обращения: 4 октября 2024 г.).

[4] GitHub - centrifugal/centrifuge: Real-time messaging library for Go. — URL: https://github.com/centrifugal/centrifuge (дата обращения: 4 октября 2024 г.).

[5] GitHub - feederco/go-socket.io: socket.io library for golang, a real-time application framework. — URL: https://github.com/feederco/go-socket.io (дата обращения: 7 октября 2024 г.).

[6] GitHub - stretchr/testify: A toolkit with common assertions and mocks that plays nicely with the standard library. — URL: https://github.com/stretchr/testify?tab=readme-ov-file (дата обращения: 7 октября 2024 г.).

[7] GitHub - vektra/mockery: A mock code autogenerator for Go. — URL: https://github.com/vektra/mockery (дата обращения: 7 октября 2024 г.).

[8] HTTP/3 - Wikipedia. — URL: https://en.wikipedia.org/wiki/HTTP/3 (дата обращения: 4 октября 2024 г.).

[9] HttpClient • Angular. — URL: https://angular.dev/api/common/http/HttpClient (дата обращения: 7 октября 2024 г.).

[10] PostgreSQL 17.0 Documentation. — URL: https://www.postgresql.org/docs/17/index.html (дата обращения: 4 октября 2024 г.).

[11] RxJS - WebSocketSubject. — URL: https://rxjs.dev/api/webSocket/WebSocketSubject (дата обращения: 4 октября 2024 г.).

[12] Server Sent Events. — URL: https://learn.javascript.ru/server-sent-events (дата обращения: 4 октября 2024 г.).

[13] Socket.IO. — URL: https://socket.io/ (дата обращения: 4 октября 2024 г.).

[14] System usability scale - Wikipedia. — URL: https://en.wikipedia.org/wiki/System_usability_scale (дата обращения: 7 октября 2024 г.).

[15] TypeScript: JavaScript With Syntax For Types. — URL: https://www.typescriptlang.org/ (дата обращения: 4 октября 2024 г.).

[16] WebRTC. — URL: https://webrtc.org/ (дата обращения: 4 октября 2024 г.).

[17] The WebSocket API (WebSockets) - Web APIs | MDN. — URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата обращения: 4 октября 2024 г.).

[18] WebSockets for fun and profit - Stack Overflow. — URL: https://stackoverflow.blog/2019/12/18/websockets-for-fun-and-profit/ (дата обращения: 4 октября 2024 г.).

[19] WebTransport - Web APIs | MDN. — URL: https://developer.mozilla.org/en-US/docs/Web/API/WebTransport (дата обращения: 4 октября 2024 г.).

[20] Websocket | Fiber. — URL: https://docs.gofiber.io/contrib/websocket/ (дата обращения: 7 октября 2024 г.).

[21] What is CRUD. — URL: https://www.codecademy.com/article/what-is-crud (дата обращения: 4 октября 2024 г.).

[22] dbdocs.io - Database Documentation and Catalog Tool. — URL: https://dbdocs.io/ (дата обращения: 4 октября 2024 г.).

[23] ngx-socket-io - npm. — URL: https://www.npmjs.com/package/ngx-socket-io (дата обращения: 4 октября 2024 г.).

[24] websocket package - github.com/coder/websocket - Go Packages. — URL: https://pkg.go.dev/github.com/coder/websocket (дата обращения: 4 октября 2024 г.).

[25] websocket package - github.com/gorilla/websocket - Go Packages. — URL: https://pkg.go.dev/github.com/gorilla/websocket (дата обращения: 4 октября 2024 г.).

[26] websocket package - golang.org/x/net/websocket - Go Packages. — URL: https://pkg.go.dev/golang.org/x/net/websocket#section-documentation (дата обращения: 4 октября 2024 г.).

[27] ws package - github.com/gobwas/ws - Go Packages. — URL: https://pkg.go.dev/github.com/gobwas/ws (дата обращения: 4 октября 2024 г.).

[28] Введение в документацию по Angular. — URL: https://angdev.ru/angular/ (дата обращения: 4 октября 2024 г.).

[29] Длинные опросы. — URL: https://learn.javascript.ru/long-polling (дата обращения: 4 октября 2024 г.).

[30] Методы HTTP запроса. — URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Methods (дата обращения: 4 октября 2024 г.).

[31] Сравнение технологий WebSockets, Server-Sent-Events, Long-Polling, WebRTC и WebTransport. — URL: https://rxdb.info/articles/websockets-sse-polling-webrtc-webtransport.html (дата обращения: 4 октября 2024 г.).