

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

WEB–приложение социальной сети по
интересам для совместного общения и развития

Студент _____ Платонов Егор Валерьевич
(подпись, дата) (ФИО полностью)

Руководитель _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____
(подпись, дата) Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов «25»
декабря 2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Платонов Егор Валерьевич гр. 20ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Web–приложение социальной сети по интересам для совместного общения и развития
- Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____
2. Руководитель работы Попова Наталия Александровна
3. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение: предоставление платформы для оптимизации социального взаимодействия где пользователи могут связываться с другими людьми и развиваться вместе с ними на основе общих интересов и увлечений
Основные функции системы:
 1. Формирование личного кабинета пользователя на основе его предпочтений;
 2. Фильтрация контента на платформе исходя из активности и интересов пользователя;
 3. Система сообществ для взаимодействия людей с общими интересами;
 4. Взаимодействие пользователей в рамках проведения различных мероприятий;
 5. Поиск людей основанный на различных предпочтениях пользователя;
 6. Система мессенджера для общения людей в рамках платформы;Технология разработки: предметно–ориентированное программирование.
Язык программирования: Next JS, Nest JS, TypeScript
Среда программирования: Visual Studio Code
4. Перечень подлежащих разработке вопросов
 1. анализ предметной области и постановка задачи;

2. проектирование программных средств;
3. реализация программных средств;
4. тестирование программных средств;
5. оформление ВКР.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 162 листа, 16 таблиц, 10 рисунков, 5 использованных источников, 25 листингов.

СОЦИАЛЬНАЯ СЕТЬ, ИНТЕРЕСЫ, ПРИЛОЖЕНИЕ, WEB, ОБЩЕНИЕ, РАЗВИТИЕ, АКТИВНОСТЬ, МЕРОПРИЯТИЯ.

Объект разработки: Web–приложение социальной сети по интересам для совместного общения и развития.

Цель работы: спроектировать, разработать, протестировать и отладить web–приложение социальной сети по интересам для совместного общения и развития.

Предмет разработки: разработка программного обеспечения для web–приложения социальной сети по интересам для совместного общения и развития

Результаты работы: в процессе работы было спроектировано и впоследствии разработано и протестировано приложение социальной сети по интересам для совместного общения и развития, в ходе чего было выявлено что приложение работает нормально.

Средства разработки: Visual Studio Code, Nest JS, Next JS, Type Script, Mongo DB.

					ПГУ 09.03.02 – 08ВР201.12 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Платонов Е.В.			Web–приложение социальной сети по интересам для совместного общения и развития.	Лит.	Лист	Листов
Провер.		Попова Н.А.					4	162
Реценз.						группа 20ВИ1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение	7
1 Постановка задачи и анализ требований	8
1.1 Постановка задачи	8
1.2 Анализ программного обеспечения аналогичного назначения	9
1.3 Формулировка и анализ требований к разрабатываемому программному обеспечению	11
2 Проектирование web-приложения социальной сети по интересам для совместного общения и развития	14
2.1 Проектирование бизнес-процессов системы	14
2.2 Проектирование базы данных.....	16
2.3 Проектирование интерфейса.....	20
3 Программная реализация web-приложения социальной сети по интересам для совместного общения и развития	29
3.1 Реализация серверного приложения.....	29
3.2 Реализация клиентского приложения.....	42
3.3 Реализация базы данных	51
4 Тестирование web-приложения социальной сети по интересам для совместного общения и развития	56
4.1 Тестирование белым ящиком.....	56
4.1.1 Покрытие операторов	56
4.1.2 Покрытие решений	59
4.1.3 Покрытие решений и условий.....	60
4.2 Метрики кода	61
4.2.1 Метрика SLOC	61

4.2.2 Цикломатическая сложность.....	62
4.2.3 Метрика Холстеда.....	63
4.2.4 Метрика Чепина.....	66
4.3 Тестирование интерфейса Мессенджера.....	67
4.3.1 Таблица тестирования	67
5 Планирование и расчёт бюджета.....	69
Заключение	72
Список использованных источников.....	73
Приложение А (рекомендуемое) Проектирование интерфейса.....	75
Приложение Б (рекомендуемое) Программный код методов серверной части приложения.....	92
Приложение В (рекомендуемое) Программный код методов клиентской части приложения.....	112
Приложение Г (рекомендуемое) Результаты тестирования мессенджера.....	152
Приложение Д (рекомендуемое) Скриншоты тестирования мессенджера.....	155

Введение

С развитием технологий и доступностью интернета, социальные сети стали неотъемлемой частью повседневной жизни многих людей. Они предоставляют возможность общения, обмена информацией и установления новых контактов. Однако существующие социальные платформы не всегда способствуют развитию и обогащению интересов пользователей, поскольку они чаще ориентированы на повседневные коммуникации.

Целью данной дипломной работы является создание социальной сети по интересам, пользователи которой смогут взаимодействовать с другими людьми, которые подбираются исходя из интересов пользователя, его активности в различных мероприятиях, а также встречах, нацеленных на получение новых знаний, навыков и знакомств.

1 Постановка задачи и анализ требований

1.1 Постановка задачи

Для создания социальной сети по интересам для совместного общения и развития, была выбрана среда web, поскольку на текущий момент это самый популярный и востребованный способ организации различных приложений. В качестве архитектуры разрабатываемого приложения была выбрана архитектура клиент-сервер.

На стороне клиента будет использоваться фреймворк Next JS, а также Type Script. Использование данных технологий обусловлено тем, что Next JS хорошо подходит для разработки приложений, нуждающихся в том, чтобы они хорошо индексировались различными поисковиками в браузерах.

На стороне сервера будет использоваться фреймворк Nest JS, а также Type Script. В качестве базы данных была выбрана Mongo DB, потому что она имеет хорошие показатели скорости работы, при частой модификации данных, что безусловно подходит для этого проекта.

Для того, чтобы создать социальную сеть по интересам, нужно выделить ряд особенностей и требований которыми должна обладать система, а именно:

- авторизация пользователя в системе, основанная на технологии jwt-токенов, для обеспечения защищенности аккаунта пользователя;
- личный кабинет пользователя, с возможностью его редактирования, а также наполнения различного рода контентом;
- темизация интерфейса по выбору пользователя;
- интернационализация элементов интерфейса;
- мессенджер, используемый для общения пользователей, с возможностями отправки различных медиа-файлов, а также текстовых сообщений;
- модуль для организации мероприятий и встреч для пользователей;
- организация взаимодействия пользователей посредством создания

сообществ по интересам;

- поиск пользователей по интересующим критериям;
- интеграция с агрегатором официальных мероприятий, с привязкой к городу в котором находится пользователь;
- новостная лента, которая содержит интересующий конкретного пользователя контент, подбираемый на основе указанных им интересов, активности и закладок (например, другие пользователи или сообщества).

Важной особенностью разрабатываемого приложения является возможность пользователей принимать участие в организуемых официально (через агрегатор) мероприятиях, а также приглашать других пользователей на эти мероприятия.

Также каждый пользователь имеет возможность создавать свои встречи, указывая интересы, цели конкретной встречи, а также делиться любыми медиа-данными о ходе конкретной встречи.

1.2 Анализ программного обеспечения аналогичного назначения

В качестве систем аналогов для проведения сравнительного анализа были выбраны ВКонтакте [1] и Last.fm [2], потому что на текущий момент они являются наиболее популярными приложениями данного типа. Результаты сравнения приведены в таблице 1. Сравнительный анализ, а также выбор параметров для сравнения проводился на основе опыта работы с каждой из систем аналогов.

Таблица 1 – Результаты сравнения систем аналогов

Возможности	ВКонтакте	Last.fm	Разрабатываемая система
Личный кабинет пользователя	Да	Да	Да
Темизация интерфейса	Да	Нет	Да
Интернационализация интерфейса	Нет	Нет	Да
Взаимодействия пользователей через сообщества по интересам	Да	Нет	Да

Продолжение таблицы 1

Изменение настроек личного кабинета	Да	Да	Да
Организация собственных мероприятий и встреч пользователей	Нет	Нет	Да
Приглашения пользователей на мероприятия и встречи, проводимые другими пользователями или официальным агрегатором	Нет	Нет	Да
Страница музыки, с возможностью публиковать свои композиции и вести статистику внутри приложения, слушать композиции других пользователей	Да	Да	Да
Мессенджер с возможностями отправлять сообщения любого типа	Да	Нет	Да
Новостная лента, с подбором контента по интересам, активности и закладкам	Да	Да	Да
Возможность публиковать различного рода контент в личном кабинете пользователя	Да	Да	Да
Возможность публиковать различного рода контент в сообществах по интересам	Да	Нет	Да
Управления фильтрами контента	Да	Да	Да
Поиск людей по интересам, встречам и мероприятиям	Нет	Да	Да

В ходе проведенного анализа существующих аналогов разрабатываемой системы, было выявлено что, разрабатываемая система не уступает по общим критериям и возможностям, а во многом даже превосходит существующие проекты.

1.3 Формулировка и анализ требований к разрабатываемому программному обеспечению

Для того, чтобы четко сформулировать требования к разрабатываемому программному обеспечению, были созданы две диаграммы вариантов использования (для авторизованного и неавторизованного пользователя). Диаграммы вариантов использования хорошо подходят для использования при проектировании системы, с целью обозначить необходимый функционал которым должна обладать система или ее отдельные составляющие [3].

Диаграмма вариантов использования для неавторизованного пользователя приведена на рисунке 1.

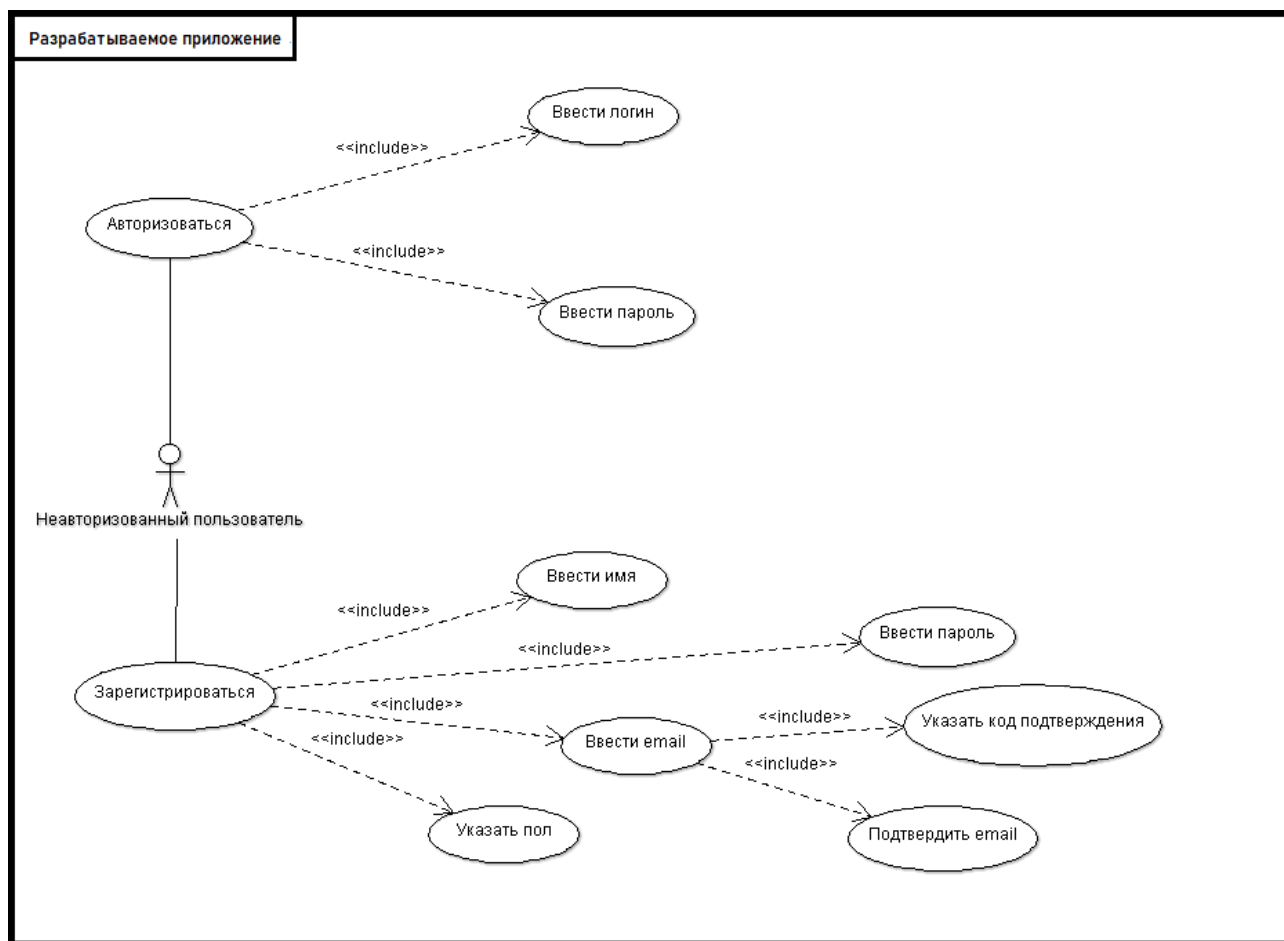


Рисунок 1 – Диаграмма вариантов использования «Неавторизованный пользователь»

Каждый вариант использования на диаграмме может быть описан с помощью спецификации, которая содержит подробное описание составляющих вариант использования элементов [4]. Спецификации вариантов использования для неавторизованного пользователя приведены в таблице 2.

Таблица 2 – Спецификации вариантов использования для неавторизованного пользователя

Вариант использования	Описание
Авторизоваться	Вариант использования «Авторизоваться» содержит несколько обязательных вложенных вариантов использования, а именно: «Ввести логин» и «Ввести пароль»
Зарегистрироваться	Вариант использования «Зарегистрироваться» содержит несколько обязательных вложенных вариантов использования, а именно: «Ввести email» и «Ввести пароль», «Ввести имя» и «Указать пол». Вложенный вариант использования «Ввести email» содержит два обязательных варианта использования, а именно: «Указать код подтверждения» и «Подтвердить email».

Диаграмма вариантов использования для авторизованного пользователя приведена на рисунке 2.

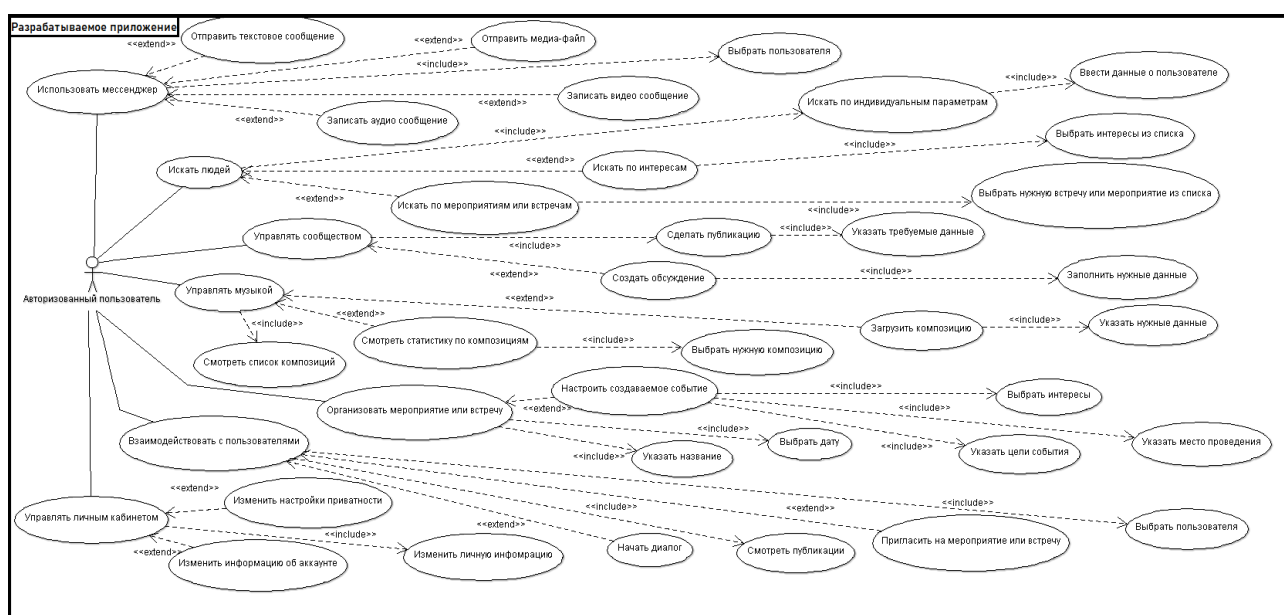


Рисунок 2 – Диаграмма вариантов использования для авторизованного пользователя

Диаграмма вариантов использования для авторизованного пользователя содержит 7 основных вариантов использования, а именно: «Использовать мессенджер», «Искать людей», «Управлять сообществом», «Организовать

мероприятие или встречу», «Взаимодействовать с пользователями», «Управлять личным кабинетом».

Вариант использования «Использовать мессенджер» содержит следующие обязательные варианты использования: «Выбрать пользователя», а также необязательные варианты использования: «Отправить текстовое сообщение», «Отправить медиа-файл», «Отправить аудиосообщение», «Отправить видеосообщение».

Вариант использования «Искать людей» содержит следующие обязательные варианты использования: «Искать по индивидуальным параметрам», а также необязательные варианты использования: «Искать по интересам», «Искать по мероприятиям или встречам».

Вариант использования «Управлять сообществом» содержит следующие варианты использования: «Сделать публикацию», «Создать обсуждение», а также «Смотреть новостную ленту сообщества».

Вариант использования «Организовать мероприятие или встречу» содержит следующие обязательные варианты использования: «Выбрать дату», «Указать место проведения», «Указать название», а также необязательный: «Настроить создаваемое событие», который состоит из «Указать цели события», «Указать интересы».

Вариант использования «Взаимодействовать с пользователями» содержит следующие обязательные варианты использования: «Выбрать пользователя», «Смотреть публикации», а также необязательные: «Начать диалог», «Пригласить на мероприятие или встречу».

Вариант использования «Управлять личным кабинетом» содержит следующие варианты использования: «Изменить личную информацию», «Изменить информацию об аккаунте», «Изменить настройки приватности».

2 Проектирование web-приложения социальной сети по интересам для совместного общения и развития

2.1 Проектирование бизнес-процессов системы

Для проектирования бизнес-процессов системы, был выбран функционал мессенджера в приложении, алгоритм работы которого описан на диаграмме деятельности (рисунок 3).

Получившаяся диаграмма деятельности приведена на рисунке 3.

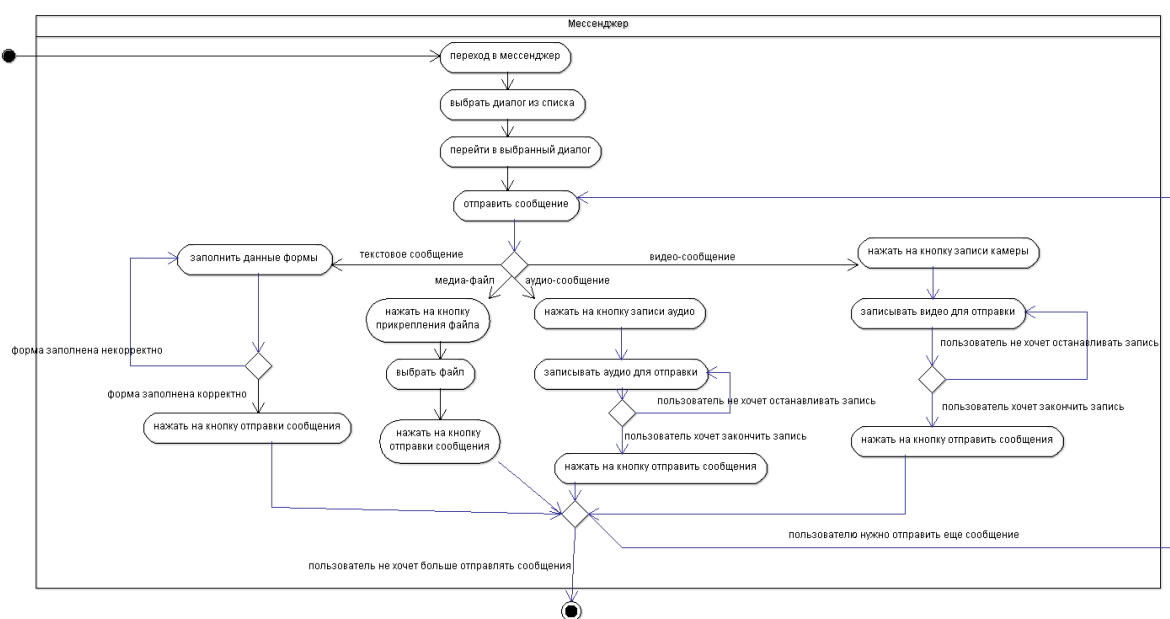


Рисунок 3 – Диаграмма деятельности «мессенджер»

Данная схема описывает алгоритм поведения пользователя при работе с мессенджером, а также все возможные опции и действия, которые пользователь может совершать при работе в мессенджере. Так, у пользователя есть выбор из нескольких способов отправки сообщения, а именно:

- видеосообщение;
- аудиосообщение;
- текстовое сообщение;
- медиа-файл.

Отправка видеосообщения подразумевает необходимость со стороны пользователя записать видео, а также прервать запись, когда это необходимо и отправить полученное видеосообщения другому пользователю.

Отправка аудиосообщения подразумевает необходимость со стороны пользователя записать аудио, а также прервать запись, когда это необходимо и отправить полученное аудиосообщения другому пользователю.

Отправка текстового сообщения подразумевает необходимость со стороны пользователя написать определенный текст, нажать на кнопку отправки сообщения, чтобы переслать его другому пользователю.

Отправка медиа-файла подразумевает необходимость со стороны пользователя выбрать нужный файл, имеющийся на его текущем устройстве, а также, в момент, когда нужный файл выбран, нажать на кнопку отправки сообщения, чтобы переслать файл другому пользователю.

Для более подробного описания структуры мессенджера, будет использоваться диаграмма компонентов. Получившаяся диаграмма приведена на рисунке 4.

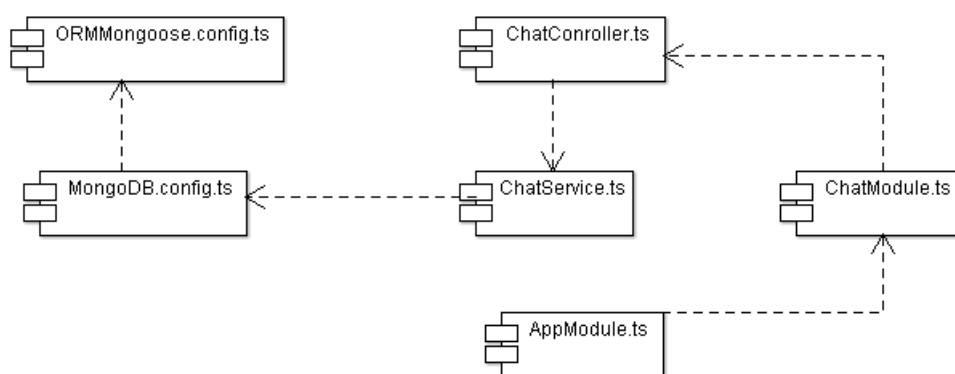


Рисунок 4 – Диаграмма компонентов мессенджера

На данной диаграмме присутствует несколько компонентов системы, описывающих модуль мессенджера, а именно:

Компонент ChatModule – основной модуль, осуществляющий настройку взаимодействия между остальными компонентами системы.

Компонент ChatController – компонент системы, отвечающий за приём запросов пользователя на добавление или изменение сообщений в мессенджере.

Компонент ChatService – компонент системы, отвечающий за выполнение действий, предусмотренных определенным запросом и делегирующий их выполнение компоненту базы данных.

Компонент MongoDB.config.ts – компонент системы, осуществляющий работы с базой данных (подключение, обращение с запросами).

Компонент OrmMongoose.config.ts – компонент системы, осуществляющий управление базой данных с использованием встроенных наборов команд и запросов.

2.2 Проектирование базы данных

В качестве базы данных в проекте используется Mongo DB. Mongo DB – это СУБД, которая является документно-ориентированной. Особенностью СУБД данного класса является отсутствие необходимости описания схемы таблиц. Данные в Mongo DB хранятся в виде коллекций объектов, каждая из которых может иметь разную структуру [5].

В ходе проектирования базы данных будут описаны все классы объектов предметной области, эти же классы будут представлены в виде объектов с набором свойств, каждое из которых будет типизировано и для каждого свойства будут разработаны ограничения.

Важно отметить, что в Mongo DB, свойства объектов могут быть типизированы как пользовательскими типами, так и встроенными в СУБД (например, string, string[] и так далее) [6].

В случае, если свойство типизировано только встроенным типом, на него действует стандартные ограничения СУБД (например, тип строка имеет вместимость до 16Мб) [7].

В случае если наложены дополнительные пользовательские ограничения, они дополняют стандартные ограничения таким образом, чтобы не вызывать противоречий.

Перед началом проектирования базы данных, необходимо выделить основные классы объектов, которые присутствует в предметной области «социальная сеть по интересам для совместного общения и развития». Важно описывать классы объектов таким образом, чтобы по данному описанию можно было напрямую создать необходимые коллекции в базе данных, а также предусмотреть все описанные в них ограничения.

Описание коллекции «Подтвержденные пользователи» приведено в таблице 3.

Таблица 3 – Коллекция «Подтвержденные пользователи»

Наименование	Тип	Ограничения
userId	number	0–999999
contentFilters	boolean	boolean
markedUsers	number[]	number[]
name	string	0–30, [a–яA–Я][a–zA–Z]
status	string	0–100, [a–яA–Я][a–zA–Z]
phone	string	11, [0–9, +]
email	string	6–100, [a–яA–Я][a–zA–Z][0–9, @]
password	string	6–30, [a–яA–Я][a–zA–Z][0–9]
gender	string	1, [М, Ж]
avatar	string	0–100, [a–яA–Я][a–zA–Z][0–9]
dateRegister	date	date
login	number	0–999999
birthDate	date	date
city	string	0–100, [a–яA–Я][a–zA–Z][0–9]
events	string[]	string[]
interests	number[]	number[]
outerInvites	{ invitedUsers: number[], eventId: string, dateOfSending: date }[]	{ invitedUsers: number[], eventId: string, dateOfSending: date }[]
innerInvites	{ fromUserId: number, eventId: string, dateOfSending: date, status: boolean }[]	{ fromUserId: number, eventId: string, dateOfSending: date, status: boolean }[]
tag	{color: string,title: string}	{color: string,title: string}

Описание коллекции «Сообщества» приведено в таблице 4.

Таблица 4 – Коллекция «Сообщества»

Наименование	Тип	Ограничения
groupId	number	0–999999
name	string	6–30, [a–яA–Я][a–zA–Z][0–9, @]
description	string	0–200, [a–яA–Я][a–zA–Z][0–9, @]
interestsId	number[]	number[]
creatorId	number	0–999999
membersId	number[]	number[]
photos	string[]	string[]
videos	string[]	string[]
attachments	string[]	string[]
posts	{ id: number, title: string, files: { src: string, type: string }[], date: date, description: string, likes: number, views: number, comments: { userId: number, text: string, date: date }[] }	{ id: 0–999999, title: 6–30, [a–яA–Я][a–zA–Z][0–9, @], files: { src: string, type: string date: date, description: 0–200, [a–яA–Я][a–zA–Z][0–9, @], likes: 0–999999, views: 0–999999, comments: { userId: 0–999999, text: 0–200, [a–яA–Я][a–zA–Z][0–9, @], date: date } }
talks	{ id: number, title: string, dateOfCreation: date, messages: { userId: number, text: string, date: date }[] }	{ id: 0–999999, title: 6–30, [a–яA–Я][a–zA–Z][0–9, @], dateOfCreation: date, messages: { userId: 0–999999, text: 0–200, [a–яA–Я][a–zA–Z][0–9, @], date: date } }

Описание коллекции «Интересы» приведено в таблице 5.

Таблица 5 – Коллекция «Интересы»

Наименование	Тип	Ограничения
interestId	number	0–999999
title	string	0–50, [a–zA–Z][a–яA–Я]

Описание коллекции «Встречи» приведено в таблице 6.

Таблица 6 – Коллекция Встречи

Наименование	Тип	Ограничения
meetingId	number	0–999999
participantsId	number[]	number[]
date	date	date
description	string	0–100, [a–zA–Z][a–яА–Я][0–9]
goal	string	0–100, [a–zA–Z][a–яА–Я][0–9]
maxParticipants	number	0–10000
preview	string	string
title	string	0–100, [a–zA–Z][a–яА–Я][0–9]
creatorId	number	0–999999
address	string	0–200, [a–zA–Z][a–яА–Я][0–9]
media	{ src: string, type: string }[]	{ src: string, type: string }[]
comments	{ userId: number, text: string, date: date }[]	{ userId: 0–999999, text: 0–100, [a–zA–Z][a–яА–Я][0–9], date: date }[]

Описание коллекции «Неподтвержденные пользователи» приведено в таблице 7.

Таблица 7 – Коллекция Неподтвержденные пользователи

Наименование	Структура объектов	Ограничения
name	string	0–30, [a–яА–Я][a–zA–Z]
email	string	6–30, [a–яА–Я][a–zA–Z][0–9, @]
password	string	6–30, [a–яА–Я][a–zA–Z][0–9]
gender	string	1, [М/Ж]
city	string	6–100
actualConfirmationCode	number	100000–999999

Описание коллекции «Авторы музыки», приведено на таблице 8.

Таблица 8 – Коллекция Авторы музыки

Наименование	Структура объектов	Ограничения
userId	number	0–999999
name	string	6–30, [a–яА–Я][a–zA–Z][0–9]
compositions	{ id: number, audioSrc: string, imageSrc: string, title: string, description: string, playsNumber: number }[]	{ id: 0–999999, audioSrc: string, imageSrc: string, title: 6–100, [a–яА–Я][a–zA–Z][0–9], description: 6–200, [a–яА–Я][a–zA–Z][0–9], playsNumber: 0–999999 }[]

В ходе выполнения данной части работы, были описаны все необходимые классы объектов предметной области «социальная сеть по интересам для совместного общения и развития», описана структура всех объектов для каждого класса, а также разработаны ограничения на каждое свойство для каждого объекта.

2.3 Проектирование интерфейса

Перед проектированием интерфейса, следует выделить несколько основных модулей программы (страниц web приложения), к которым относятся:

- страница авторизации;
- страница мессенджера;
- страница личного кабинета пользователя;
- страница настроек;
- страница для поиска людей;
- страница мероприятий;
- страница организации встреч;
- страница просмотра приглашений;
- страница закладок пользователя;
- страница музыки;
- страница сообществ;
- страница новостей.

Для создания качественного и полного интерфейса приложения, нужно определить какие именно элементы должны находиться на каждой из страниц, а также определить возможности пользователя, которые должны быть ему доступны на каждой странице. Описание страниц «авторизация», «личный кабинет», «мессенджер», «настройки», «поиск людей», «закладки пользователя» приведено в таблице 9.

Таблица 9 – Описание страниц в интерфейсе

Название страницы	Требуемые возможности пользователя	Составляющие элементы
Страница авторизации	В случае, если пользователь уже имеет аккаунт, требуется показать форму для входа, иначе предложить перейти на форму регистрации	Элементы ввода на каждой форме в соответствии с данными, нужными для регистрации \ входа, а также кнопки подтверждения действий
Страница личного кабинета	Пользователь должен иметь возможность смотреть свои данные, просматривать статистику профиля, а также просматривать и добавлять свои публикации. Если пользователь просматривает чужой личный кабинет, должны быть элементы для перехода в диалог с пользователем или для приглашения на мероприятие	Элементы для вывода пользовательской информации и его публикаций, элементы ввода для создания новых публикаций, кнопки для перехода в диалог с пользователем или для приглашения на мероприятие (в случае если личный кабинет другого пользователя)
Страница мессенджера	Пользователь должен видеть имеющиеся диалоги, а также переходить в конкретный диалог, где будет появляться история сообщений	Элементы для вывода информации о диалогах, а также истории сообщений в конкретном диалоге
Страница настроек	Пользователь должен видеть текущие настройки аккаунта, а также иметь возможность менять основную информацию об аккаунте или профиле	Элементы для вывода информации о текущих настройках профиля, а также элементы формы для изменения настроек аккаунта или профиля
Страница поиска людей	Пользователь должен иметь возможность настраивать фильтры для поиска, просматривать список найденных людей, а также переходить в личный кабинет к конкретному пользователю	Элементы вывода списков для отображения списка людей, кнопки для применения конкретного фильтра, а также элементы навигации для перехода в личный кабинет конкретного пользователя
Страница закладок пользователя	Пользователь должен иметь возможность смотреть список людей и мероприятий, добавленных в закладки	Элементы вывода списков для просмотра списка людей и мероприятий, навигационные элементы для перехода на страницу пользователя или мероприятия

Описание для страниц «мероприятия», «встречи», «сообщества», «новости», «музыка» и «приглашения», а также описание элементов, которые должны быть доступны пользователю для взаимодействия на этих страницах приведено в таблице 10.

Таблица 10 – Описание страниц в интерфейсе

Название страницы	Требуемые возможности пользователя	Составляющие элементы
Страница музыки	Пользователь должен видеть список имеющейся музыки, управлять каждой конкретной композицией, а также видеть статистику авторов и композиций	Элементы вывода списка композиций, элементы управления для каждой конкретной композиции, элементы отображения статистических данных

Продолжение таблицы 10

Страница встречи	Пользователь должен видеть списки собственных встреч, будущих и прошедших встреч, а также должен иметь возможность перейти на страницу просмотра конкретной встречи	Элементы вывода списка встреч, навигационные элементы для перехода на страницу встречи
Страница новости	Пользователь должен видеть список новостных публикаций, а также иметь возможность перейти к источнику публикации	Элементы вывода списка публикаций, навигационные элементы для перехода к источнику публикации
Страница приглашения	Пользователь должен видеть список отправленных им приглашений, а также входящих приглашений	Элементы вывода списка приглашений, элементы для подтверждения действий над приглашениями
Страница сообщества	Пользователь должен видеть список сообществ, а также иметь возможность перейти в конкретное сообщество для дальнейшего взаимодействия	Элементы вывода списка сообществ, навигационные элементы для перехода на страницу конкретного сообщества
Страница мероприятия	Пользователь должен видеть список категорий проводимых мероприятий, а список мероприятий для каждой категории, а также перейти на страницу конкретного мероприятия	Элементы вывода списка мероприятий и категорий, навигационные элементы для перехода на страницу конкретного мероприятия

Когда требования к возможностям интерфейса сформулированы, можно составить макеты для каждой из страниц приложения.

Структура основного навигационного меню в приложении, появляющееся сразу после авторизации приведена на рисунке 5.

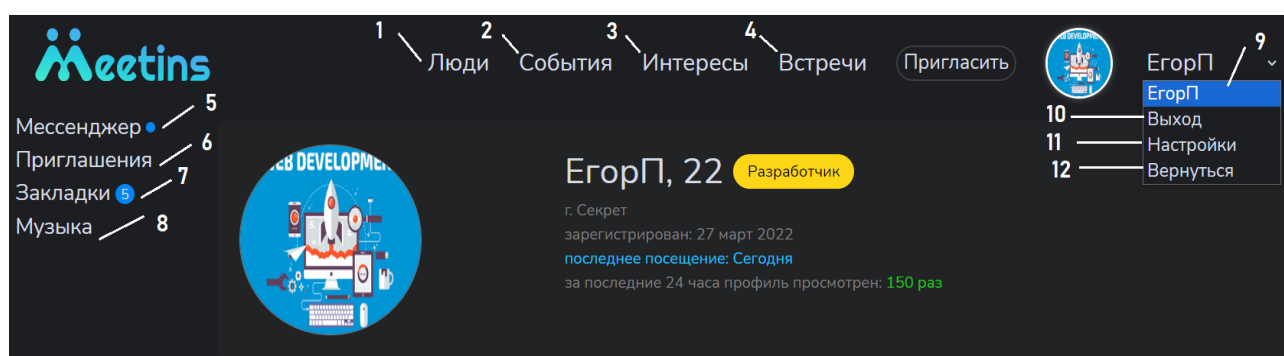


Рисунок 5 – Основное навигационное меню

На рисунке 5 отображены основные навигационные элементы, служащие для перехода между разными страницами веб приложения, а именно:

- пункт 1 – ссылка для перехода на страницу «Люди»;
- пункт 2 – ссылка для перехода на страницу «События»;
- пункт 3 – ссылка для перехода на страницу «Интересы»;
- пункт 4 – ссылка для перехода на страницу «Встречи»;
- пункт 5 – ссылка для перехода на страницу «Мессенджер»;
- пункт 6 – ссылка для перехода на страницу «Приглашения»;
- пункт 7 – ссылка для перехода на страницу «Закладки»;
- пункт 8 – ссылка для перехода на страницу «Музыка»;
- пункт 9 – ссылка для перехода на страницу «Личный кабинет»;
- пункт 10 – ссылка для перехода на страницу «Авторизация»;
- пункт 11 – ссылка для перехода на страницу «Настройки»;
- пункт 12 – ссылка для перехода на предыдущую активную страницу.

На рисунке А.1 приведены основные элементы, входящие в состав формы входа, а именно: 1 – заголовок формы, 2 – блок формы, 3 – текстовые поля ввода, 4 – кнопка подтверждения действия для входа, 5 – навигационный элемент, для перехода на форму регистрации.

На рисунке А.2 приведены элементы формы регистрации, а именно: 1 – заголовок формы, 2 – текстовые поля для ввода, 3 – кнопка подтверждения регистрации, 4 – навигационный элемент для перехода на форму входа.

На рисунке А.3 приведены основные элементы, входящие в состав личного кабинета, а именно: 1 – элементы для вывода статистики кабинета, 2 – блоки для вывода информации о пользователе, 3 – кнопка для открытия формы добавления публикации, 4 – блок для вывода созданной публикации.

На рисунке А.4 приведены основные элементы, служащие для редактирования информации в личном кабинете, а именно: 1 – поле для ввода нового статуса, 2 – кнопки для подтверждения обновления статуса или отмены действия, 3 – текущие выбранные интересы пользователя, 4 – интересы,

доступные к выбору пользователя, 5 – кнопка подтверждения обновления интересов пользователя.

На рисунке А.5 приведена форма для настройки пользовательского тега, она содержит следующие элементы: 1 – текстовое поле для обновления надписи в теге, 2 – кнопка для сохранения изменений.

На рисунке А.7 приведены дополнительные элементы, появляющиеся в личном кабинете, если пользователь просматривает чужой личный кабинет, а именно: 1 – кнопка открытия списка дополнительных действий, 2 – кнопки для перехода в диалог с пользователем и для приглашения пользователя на мероприятие соответственно.

На рисунке А.8 приведена страница настроек, которая содержит в себе ряд следующих элементов: 1 – элементы для управления фильтрами контента, интернационализации интерфейса приложения и темизации интерфейса соответственно, 2 – элементы ввода для изменения информации о профиле пользователя.

На рисунке А.9 приведена форма для настройки аккаунта пользователя, она содержит следующие элементы: 1 – название формы, 2 – элементы для изменения основной информации об аккаунте, 3 – кнопка подтверждения изменения обновленных данных.

На рисунке А.10 приведена страница мессенджера, которая содержит следующие элементы: 1 – список доступных к выбору диалогов пользователя, 2 – список сообщений, имеющих в выбранном диалоге, а также информация о пользователе, с которым ведётся диалог, 3 – элементы ввода текстового сообщения и отправки медиа-сообщений (таких как файл, аудиосообщение или видеосообщение).

На рисунке А.11 приведена страница мероприятий, где пользователь видит шесть стандартных категорий. При клике на каждую из них появляется список доступных событий в данной категории.

На рисунке A.12 приведен список событий в выбранной пользователем категории, состоящий из следующих элементов: 1 – название выбранной категории, 2 – краткая информация о конкретном мероприятии.

На рисунке A.13 приведена страница с информацией о выбранном мероприятии, которая содержит следующие элементы: 1 – расширенная информация о мероприятии, 2 – кнопка для добавления мероприятия в список закладок, и кнопка для оценки мероприятия.

На рисунке A.14 приведена страница с приглашениями, которая содержит следующие элементы: 1 – описание события на которое было отправлено или получено приглашение, 2 – кнопки действия над конкретным приглашением.

На рисунке A.15 приведена страница с музыкой, которая содержит следующие элементы: 1 – список текущей музыки пользователя, включая элементы для управления каждой из композиций в списке, 2 – статистика исполнителей, 3 – список людей с которыми у пользователя совпадает музыкальный вкус, 4 – кнопка для добавления новой композиции, 5 – кнопка для просмотра статистики пользователя.

На рисунке A.16 приведена форма для добавления новой композиции, содержащая следующие элементы: 1 – текстовые поля для ввода информации о композиции, 2 – заголовки текстовых полей, 3 – кнопка для добавления новой композиции.

На рисунке A.17 приведено модальное окно, содержащее элементы для отображения статистики для каждой опубликованной пользователем композиции.

На рисунке A.18 приведена страница встречи, которая содержит следующие элементы: 1 – навигационное меню для переключения по видам встреч, 2 – кнопка для перехода на форму для добавления новой встречи, 3 – блок с информацией о текущих встречах, содержащий информацию о заголовках встреч, организаторе, планируемой дате встречи и о ее целях, 4 – кнопки для перехода к конкретной встрече и для связи с организатором встречи.

На рисунке A.19 приведена форма для добавления новой встречи, которая содержит следующие элементы: 1 – заголовок формы, 2 – заголовки для полей ввода информации, 3 – текстовые поля для ввода информации о планируемой встрече, 4 – кнопка подтверждения создания новой встречи.

На рисунке A.20 приведена страница для просмотра информации о встрече, содержащая следующие элементы: 1 – кнопка для присоединения к выбранной встрече, 2 – список пользователей, записавшихся на встречу и информация о них, 3 – элементы вывода информации о целях встречи и ее описание, 4 – список медиа файлов, а именно фотографий или видео, 5 – список комментариев от участников встречи, 6 – кнопка для добавления нового комментария, 7 – кнопка для загрузки нового файла.

На рисунке A.21 приведена страница для просмотра сообществ, содержащая следующие элементы: 1 – список сообществ с краткой информацией о каждом из них, 2 – элементы для перехода к конкретному сообществу из списка, 3 – навигационное меню, 4 – ссылка для создания нового сообщества.

На рисунке A.22 приведена форма для создания нового сообщества, содержащая следующие элементы: 1 – заголовок формы, 2 – текстовые поля для указания информации о новом сообществе, 3 – кнопка подтверждения создания нового сообщества.

На рисунке A.23 приведена главная страница сообщества, которая содержит следующие элементы: 1 – название сообщества, 2 – шапка сообщества, 3 – описание сообщества, 4 – навигационные элементы для открытия соответствующих вкладок внутри сообщества, 5 – кнопка для добавления новой публикации, 6 – список всех участников сообщества.

На рисунке A.24 приведена форма для добавления новой публикации в сообществе, содержащая следующие элементы: 1 – название формы, 2 – заголовки текстовых полей ввода, 3 – поля ввода информации о сообществе, 4 – кнопка для загрузки данных в публикацию, 5 – кнопка подтверждения создания новой публикации.

На рисунке А.25 список публикаций в сообществе, содержащий следующие элементы: 1 – название публикации, 2 – описание публикации, 3 – кнопка для просмотра комментариев, 4 – кнопка для оценки публикации, 5 – дата появления публикации в сообществе, 6 – количество просмотров публикации, 7 – слайдер с отображением информации, прикрепленной к публикации.

На рисунке А.26 приведено модальное окно для просмотра списка комментариев публикации, содержащее следующие элементы: 1 – список комментариев, 2 – текстовое поле для создания нового комментария, 3 – кнопка для добавления нового комментария.

На рисунке А.27 приведено модальное окно, использующееся для отображения списка обсуждения, содержащее следующие элементы: 1 – заголовок окна, 2 – название конкретного обсуждения, 3 – дата создания обсуждения, 4 – информация об отправителе последнего сообщения в обсуждении, 5 – кнопка для создания нового обсуждения, 6 – количество сообщений в обсуждении.

На рисунке А.28 приведено модальное окно со списком сообщений в обсуждении, содержащее следующие элементы: 1 – заголовок окна, 2 – аватар пользователя, оставившего сообщение в обсуждении, 3 – идентификатор пользователя, 4 – содержимое сообщения, 5 – дата написания сообщения, 6 – текстовое поле для написания нового сообщения, 7 – кнопка подтверждения отправки сообщения.

На рисунке А.29 приведено модальное окно с отображением фотографий, которые были опубликованы в сообществе, состоящее из следующих элементов: 1 – список фотографий, 2 – заголовок модального окна.

На рисунке А.30 приведено модальное окно с отображением видеозаписей, которые были опубликованы в сообществе, состоящее из следующих элементов: 1 – список видеозаписей, 2 – заголовок модального окна.

На рисунке А.31 приведены дополнительные элементы взаимодействия с пользователем, появляющиеся только для создателя сообщества, а именно: 1 – кнопка для управления информацией о сообществе.

На рисунке А.32 приведено модальное окно, содержащее информацию о настройках сообщества, которое состоит из следующих элементов: 1 – заголовок модального окна, 2 – текстовые заголовки для каждого из полей ввода, 3 – текстовые поля для изменения информации о сообществе, 4 – кнопка для изменения шапки сообщества, 5 – кнопка для сохранения обновленных настроек.

На рисунке А.33 приведена страница новостей, состоящая из следующих элементов: 1 – название публикации, 2 – аватар автора данной публикации, 3 – описание публикации, 4 – количество оценивших публикацию пользователей, 5 – дата появления публикации в источнике, 6 – слайдер с информацией о файлах, прикрепленных к публикации, 7 – меню для просмотра списка дополнительных действий.

3 Программная реализация web-приложения социальной сети по интересам для совместного общения и развития

3.1 Реализация серверного приложения

Для реализации серверной части приложения был выбран фреймворк Nest JS, который является одним из самых популярных средств разработки веб-серверов [8]. Для взаимодействия с базой данных будет использоваться ORM Mongoose. Данная ORM предоставляет множество встроенных средств для подключения и работы с базой данных mongo DB [9].

В Nest JS для создания простого веб-сервера используется класс NestFactory. Код программы для инициализации сервера приведен в листинге 1.

Листинг 1 – Инициализация сервера Nest JS

```
const start = async () => {
  try {
    const PORT = process.env.PORT || 5000;
    const app = await
NestFactory.create<NestExpressApplication>(AppModule);
    app.useStaticAssets(join(__dirname, '../src/static'));
    app.use(cookieParser());
    app.enableCors({
      origin: true,
      methods: 'GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS',
      credentials: true,
    });

    const config = new DocumentBuilder()
      .setTitle('Meetins Server')
      .setDescription('Meetins Documentation')
      .setVersion('1.0.0')
      .addTag('Egor Platonov')
      .build()
    const swagger = SwaggerModule.createDocument(app, config)
    SwaggerModule.setup('/api/docs', app, swagger);

    await app.listen(PORT, () => {
      console.log('Server started');
    })
  }
  catch(error) {
    console.log(error);
  }
}
start();
```

В Nest JS для обеспечения доступа к серверу с использованием сокетов используется класс `AppGateway`, который содержит реализацию двух методов, а именно `OnGatewayConnection`, `OnGatewayDisconnect`, служащих для обработки событий подключения клиента к сокету и отключения соответственно.

Программный код класса `AppGateway` приведен в листинге 2.

Листинг 2 – Программный код класса `AppGateway`

```
import { Injectable } from '@nestjs/common';
import { OnGatewayConnection, OnGatewayDisconnect,
SubscribeMessage, WebSocketGateway, WebSocketServer } from
'@nestjs/websockets';
import { Server } from 'socket.io';
import { HelpJwtService } from '../help/token.service';
import { Cron } from '@nestjs/schedule';

@Injectable()
@WebSocketGateway({ cors: true, transports: ['websocket'] })
export class AppGateway implements OnGatewayConnection,
OnGatewayDisconnect {

  @WebSocketServer()
  server: Server;

  public activeUsersList: string[] = [];
  public activeFullUsersList: any[] = [];

  constructor(private jwtHelpService: HelpJwtService) { }

  handleDisconnect(client: any) {
    const accessToken = client.handshake.headers.cookie?.split(';')
      .find((cookie: string) =>
        cookie.startsWith('access')).split('=')[1];
    const decodeToken =
      this.jwtHelpService.decodeJwtFromString(accessToken);
    this.activeUsersList = this.activeUsersList.filter(el => el !==
      decodeToken?.email);
    this.activeFullUsersList = this.activeFullUsersList.filter(el
      => el.email !== decodeToken?.email);
  }

  handleConnection(client: any, ...args: any[]) {
    const accessToken = client.handshake.headers.cookie?.split(';')
      .find((cookie: string) =>
        cookie.startsWith('access')).split('=')[1];
    const decodeToken =
      this.jwtHelpService.decodeJwtFromString(accessToken);
    this.activeUsersList = [...this.activeUsersList,
      decodeToken?.email];
    const fullClient = {
      email: decodeToken?.email,
      socketId: client.id,
      userId: decodeToken?.userId
    }
  }
}
```

```

    };
    this.activeFullUsersList = [...this.activeFullUsersList,
fullClient];
    }

    @Cron('45 * * * * *')
    handleUpdateUserList() {
        this.server.emit('updateUsers', { users:
this.activeFullUsersList });
    }
}

```

Класс `AppGateway` содержит несколько методов, а именно `handleConnection` и `handleDisconnect`, содержащих логику сохранения информации о каждом новом клиенте, а также удаления информации о клиенте, в момент, когда он отключается.

Также, этот класс содержит метод `handleUpdateUserList`, который используется для получения списка подключенных клиентов.

Для реализации алгоритма авторизации пользователя, создан класс `AuthController`, который содержит в себе набор методов, каждый из которых отвечает за различные этапы авторизации пользователя. Программный код данного класса приведен в листинге 3.

Листинг 3 – Программный код класса `AuthController`

```

@ApiTags('Авторизация')
@Controller('auth')
export class AuthController {

    constructor(private authService: AuthService) {}

    @Post('/login')
    login(@Body() userDto: UserDto, @Res() res: Response) {
        return this.authService.login(userDto, res);
    }
    @Get('/logout')
    logout(@Res() res: Response, @Req() req: Request) {
        return this.authService.logout(res, req);
    }
    @Post('/registration')
    registration(@Body() userDto: CreateUserDto) {
        return this.authService.registration(userDto);
    }
    @Post('/registrationWithConfirmation')
    registrationWithConfirmation(@Body() userDto: CreateUserDto) {
        return
this.authService.registrationWithConfirmation(userDto);
    }
}

```

```

        @Post('/acceptUserAccount')
        acceptUserAccount(@Body() userDto: AcceptUserDto, @Res() res:
Response) {
            return this.authService.acceptUserAccount(userDto, res);
        }
    }

```

Данный класс содержит несколько методов, а именно:

- `post /login` – метод, который реализует логику для входа пользователя на сайт;
- `get /logout` – метод, который реализует логику удаления сессии пользователя;
- `post /registration` – метод, который реализует логику регистрации пользователя на сайте, а также создания новой сессии;
- `post /registrationWithConfirmation` – метод, который нужен для регистрации пользователя с подтверждением электронной почты;
- `post /acceptUserAccount` – метод, который реализует логику проверки правильности кода подтверждения.

Программный код полной реализации данных методов приведен в приложении Б.

Для управления личным кабинетом пользователя создан класс `UserController`, который содержит набор методов, реализующих логику работы с пользователями. Программный код данного класса приведен в листинге 4.

Листинг 4 – Программный код класса `UserController`

```

@ApiTags('Пользователи')
@Controller('/users')
@UseGuards(JwtAuthGuard)
export class UserController {

    constructor(private userService: UserService) {}
    @ApiResponse({status: 200, type: User})
    @Post()
    addUser(@Body() userDto: CreateUserDto) {
        return this.userService.addUser(userDto);
    }
    @ApiResponse({status: 200, type: User})
    @Post('/getUserByLogin')
    getUserByLogin(@Body() login: string) {
        return this.userService.getUserByLogin(login);
    }
    @ApiResponse({status: 200, type: User})

```

```

@Delete('/removeUserInterest')
removeUserInterest(@Req() req: Request) {
    return this.userService.removeUserInterest(req);
}
@ApiResponse({status: 200, type: User})
@Update('/updateUserInterest')
updateUserInterest(@Req() req: Request) {
    return this.userService.updateUserInterest(req);
}
@ApiResponse({status: 200, type: User})
@Post('/addUserIntoMarkedList')
addUserIntoMarkedList(@Req() request: Request) {
    return this.userService.addUserIntoMarkedList(request);
}
@ApiResponse({status: 200, type: User})
@Delete('/removeUserFromMarkedList/:userId')
removeUserFromMarkedList(@Param() params: any, @Req() request:
Request) {
    return this.userService.removeUserFromMarkedList(request,
params.userId);
}
@ApiResponse({status: 200, type: User})
@Get('/getMarkedUsersInfo')
getMarkedUsersInfo(@Req() request: Request) {
    return this.userService.getMarkedUsersInfo(request);
}
@ApiResponse({status: 200, type: User})
@Post('/addUserEvent')
addUserEvent(@Req() request: Request) {
    return this.userService.addUserEvent(request);
}
@ApiResponse({status: 200, type: User})
@Delete('/deleteUserEvent')
deleteUserEvent(@Req() request: Request) {
    return this.userService.deleteUserEvent(request);
}
@ApiResponse({status: 200, type: User})
@Put('/updateUserTag')
updateUserTag(@Req() request: Request) {
    return this.userService.updateUserTag(request);
}
@Post('/addUserPost')
@UseInterceptors(FilesInterceptor('media', 5,
FinallMulterOptions))
addUserPost(@UploadedFiles() files, @Req() request: Request) {
    return this.userService.addUserPost(files, request);
}
}

```

Данный класс содержит методы, для работы с пользователем и данными о нём, а именно:

- `get /getUserByLogin` – метод для получения информации о пользователе;

- `post /addUserPost` – метод для добавления новой публикации в личном кабинете пользователя;
- `put /updateUserTag` – метод для обновления информации о пользователе;
- `delete /deleteUserInterest` – метод для удаления пользователя;
- `post /addUserInterest` – метод для добавления нового интереса пользователя;
- `get /getMarkedUsersInfo` – метод для получения списка пользователей;
- `delete /removeUserFromMarkedList/:userId` – метод удаления пользователя из списка закладок;
- `post /addUserIntoMarkedList` – метод для добавления пользователя в список закладок;
- `delete /removeUserInterest` – метод для удаления интереса из списка интересов пользователя;
- `update /updateUserInterest` – метод для обновления списка интересов пользователя.

Программный код полной реализации данных методов приведен в приложении Б.

Для реализации логики работы с модулем событий был создан класс `EventController`, программный код которого приведен в листинге 5.

Листинг 5 – Программный код класса `EventController`

```
@Controller('event')
@UseGuards(JwtAuthGuard)
export class EventController {

    constructor(private eventService: EventService) {}

    @Get('/getEventsCategory')
    getEventsByCategory(@Request() request) {
        return this.eventService.getEventsByCategory(request.body)
    }
    @Get('/getEventInfoById')
    getEventInfoById(@Request() request) {
```

```

        return
        this.eventService.getEventInfoById(request.body.eventId)
    }
    @Get('/getCommentsForEventById')
    getCommentsForEventById(@Request() request) {
        return
        this.eventService.getCommentsForEventById(request.body.eventId)
    }
    @Post('/sendInviteToUser')
    sendInviteToUser(@Request() request) {
        return this.eventService.sendInviteToUser(request)
    }
    @Get('/getUserEventsInfo')
    getUserEventsInfo(@Request() request) {
        return this.eventService.getUserEventsInfo(request)
    }
    @Get('/getUserInnerInvitesEventInfo')
    getUserInnerInvitesEventInfo(@Request() request) {
        return
        this.eventService.getUserInnerInvitesEventInfo(request)
    }
    @Get('/getUserOuterInvitesEventInfo')
    getUserOuterInvitesEventInfo(@Request() request) {
        return
        this.eventService.getUserOuterInvitesEventInfo(request)
    }
    @Put('/declineInnerInvite')
    declineInnerInvite(@Request() request) {
        return this.eventService.declineInnerInvite(request)
    }
}

```

Данный класс содержит в себе несколько методов, а именно:

- `get /getEventsCategory/:id` – метод для получения списка актуальных мероприятий в заданной категории;
- `get /getEventInfoById/:id` – метод для получения информации о конкретном мероприятии;
- `get /getCommentsForEventById/:id` – метод для получения списка отзывов о конкретном мероприятии;
- `post /sendInviteToUser` – метод для отправки приглашения на событие или мероприятие определенному пользователю;
- `get /getUserEventsInfo` – метод для получения информации о событиях или мероприятиях пользователя;
- `get /getUserInnerInvitesEventInfo` – метод для получения списка входящих приглашений на мероприятия для конкретного пользователя;

- `get /getUserOuterInvitesEventInfo` – метод для получения списка исходящих приглашений на мероприятия для конкретного пользователя;
- `put /declineInnerInvite` – метод, реализующий логику отклонения входящего приглашения.

Программный код данных методов приведен в приложении Б.

Для реализации логики работы мессенджера был создан класс `ChatController`, программный код которого приведен в листинге 6.

Листинг 6 – Программный код класса `ChatController`

```
@Controller('chat')
@UseGuards(JwtAuthGuard)
export class ChatController {
    constructor(private chatService: ChatService) {}
    @Post('/send-file-to-chat')
    @UseInterceptors(FileInterceptor('uploadedFile',
    FinallMulterOptions))
    sendFileToChat(@UploadedFile() file, @Req() request: Request) {
        return this.chatService.sendFileToChat(file, request);
    }
    @Post('/send-message')
    sendNewMessage(@Req() request: Request) {
        return this.chatService.sendNewMessage(request);
    }
    @Get('/messages')
    getDialogMessages(@Req() request: Request) {
        return this.chatService.getDialogMessages(request);
    }
    @Post('/start-dialog')
    startNewDialog(@Req() request: Request) {
        return this.chatService.startNewDialog(request);
    }
    @Get('/my-dialogs')
    getUserDialogs(@Req() request: Request) {
        return this.chatService.getUserDialogs(request);
    }
    @Post('/check-dialog')
    checkDialog(@Req() request: Request) {
        return this.chatService.checkDialog(request);
    }
    @Post('/mark-messages-as-readed')
    markDialogMessagesAsReaded(@Req() request: Request) {
        return
        this.chatService.markDialogMessagesAsReaded(request);
    }
}
```

Данный класс содержит набор методов для работы мессенджера на сайте, а именно:

- `post /send-file-to-chat` – метод для отправки файла любого формата;

- `post /send-message` – метод для отправки текстового сообщения;
- `get /messages` – метод для получения истории сообщений;
- `post /start-dialog` – метод для создания нового диалога;
- `get /my-dialogs` – метод для получения списка диалогов;
- `post /check-dialog` – метод для проверки наличия диалога между двумя пользователями.

Для реализации модуля встреч, был создан класс `MeetingsController`, программный код которого приведен в листинге 7.

Листинг 7 – Программный код класса `MeetingController`

```
@Controller('meetings')
@ApiTags('Встречи')
@UseGuards(JwtAuthGuard)
export class MeetingsController {
    constructor(private meetingsService: MeetingsService) {}
    @Get('/get-all-meetings')
    getAllMeetings() {
        return this.meetingsService.getAllMeetings();
    }
    @Post('/create-meeting')
    createNewMeeting(@Req() request: Request) {
        return this.meetingsService.createNewMeeting(request);
    }
    @Post('/upload-media')
    @UseInterceptors(FileInterceptor('uploadedFile', MulterOptions))
    uploadMedia(@Req() request: Request, @UploadedFile() file:
Express.Multer.File) {
        return this.meetingsService.uploadMedia(request, file);
    }
    @Put('/reg-in-meeting/:meetingId')
    regInMeetings(@Req() request: Request, @Param('meetingId')
meetingId) {
        return this.meetingsService.regInMeetings(request,
meetingId);
    }
    @Post('/send-comment')
    sendCommentAboutMeeting(@Req() request: Request) {
        return
this.meetingsService.sendCommentAboutMeeting(request);
    }
    @Get('/places/:userId/')
    getUserPlaces(@Param() params: any, @Req() request: Request) {
        return this.meetingsService.getUserPlaces(request,
params.userId);
    }
}
```

Данный класс содержит набор методов для работы с модулем встреч, а именно:

- `get /get-all-meetings` – метод для получения списка всех встреч;
- `post /create-meeting` – метод для добавления новой встречи;
- `post /upload-media` – метод для загрузки медиа-данных со встречи;
- `put /reg-in-meeting/:meetingId` – метод для добавления участника встречи;
- `post /send-comment` – метод для добавления нового комментария;
- `get /places/:userId/` – метод для получения списка мест, где был пользователь.

Программный код данных методов приведен в приложении Б.

Для реализации модуля настроек, был создан класс `SettingsController`, программный код которого приведен в листинге 8.

Листинг 8 – Программный код класса `SettingsController`

```
@Controller('settings')
@UseGuards(JwtAuthGuard)
export class SettingsController {
  constructor(private settingsService: SettingsService) {}
  @Post('/update-avatar')
  @UseInterceptors(FileInterceptor('uploadedFile', {
    storage: diskStorage(
      {
        destination: './src/static',
        filename: (req, file, cb) => {
          const fileNameSplit = file.originalname.split('.');
          const fileExt = fileNameSplit[fileNameSplit.length
- 1];

          cb(null, `${Date.now()}.${fileExt}`);
        }
      }
    )
  }))
  updateUserAvatar(@UploadedFile() file, @Req() request: Request)
  {
    return this.settingsService.updateUserAvatar(file,
request);
  }
  @Post('/update-status')
  updateUserStatus(@Req() request: Request) {
    return this.settingsService.updateUserStatus(request);
  }
  @Post('/update-filter-status')
  updateFilterStatus(@Req() request: Request) {
```

```

        return this.settingsService.updateFilterStatus(request);
    }
    @Post('/update-account')
    updateUserAccount(@Req() request: Request) {
        return this.settingsService.updateUserAccount(request);
    }
    @Post('/update-profile')
    updateUserProfile(@Req() request: Request) {
        return this.settingsService.updateUserProfile(request);
    }
}

```

Данный класс содержит набор методов для реализации модуля настроек, а именно:

- post /update-avatar – метод для обновления аватара пользователя;
- post /update-status – метод для обновления статуса пользователя;
- post /update-filter-status – метод для обновления статуса фильтров контента;
- post /update-account – метод для обновления информации об аккаунте;
- post /update-profile – метод для обновления информации профиля.

Программный код данных методов приведен в приложении Б.

Для реализации логики работы модуля сообществ, был создан класс GroupsController, программный код которого приведен в листинге 9.

Листинг 9 – Программный код класса GroupsController

```

@Controller('groups')
@ApiTags('Группы')
@UseGuards(JwtAuthGuard)
export class GroupsController {
    constructor(private groupsService: GroupsService) {}
    @Get('/get-all-groups')
    getAllGroups(@Req() request: Request) {
        return this.groupsService.getAllGroups(request);
    }
    @Post('/manage-group-by-id')
    @UseInterceptors(FileInterceptor('headImage',
    FinallMulterOptions))
    manageGroupById(@UploadedFile() file, @Req() request: Request)
    {
        return this.groupsService.manageGroupById(file, request);
    }
    @Post('/create-new-group')
    createNewGroup(@Req() request: Request) {
        return this.groupsService.createNewGroup(request);
    }
}

```

```

        @Post('/create-new-post-in-group')
        @UseInterceptors(FilesInterceptor('media', 5,
        FinallMulterOptions))
        createNewPostInGroup(@UploadedFiles() files, @Req()
        request:Request) {
            return this.groupsService.createNewPostInGroup(files,
        request);
        }
        @Post('/create-new-talk-in-group')
        createNewTalkInGroup(@Req() request: Request) {
            return this.groupsService.createNewTalkInGroup(request);
        }
        @Post('/create-new-message-in-group-talk')
        createNewMessageInGroupTalk(@Req() request: Request) {
            return
        this.groupsService.createNewMessageInGroupTalk(request);
        }
        @Post('/get-group-talks-by-id')
        getGroupTalksList(@Req() request: Request) {
            return this.groupsService.getGroupTalksList(request);
        }
        @Post('/get-group-talk-messages-by-id')
        getTalkMessagesInGroupById(@Req() request: Request) {
            return
        this.groupsService.getTalkMessagesInGroupById(request);
        }
        @Post('/add-new-comment-into-post/:groupId/:postId')
        addNewCommentIntoPost(@Param() params: any, @Req() request:
        Request) {
            return
        this.groupsService.addNewCommentIntoPost(request,params);
        }
        @Put('/like/:groupId/:postId')
        likePostInGroup(@Param() params: any, @Req() request: Request)
        {
            return this.groupsService.likePostInGroup(request,params);
        }
        @Put('/unlike/:groupId/:postId')
        unlikePostInGroup(@Param() params: any, @Req() request:
        Request) {
            return
        this.groupsService.unlikePostInGroup(request,params);
        }
    }

```

Данный класс содержит набор методов для работы с модулем сообществ,

а именно:

- get /get-all-groups – метод для получения списка всех сообществ;
- get /get-group-by-id – метод для получения информации о сообществе;
- post /join-to-group – метод для добавления пользователя в список

участников;

- `post /manage-group-by-id` – метод для обновления информации о сообществе;
- `post /create-new-group` – метод для добавления нового сообщества;
- `post /create-new-post-in-group` – метод для добавления новой публикации;
- `post /create-new-talk-in-group` – метод для создания нового обсуждения;
- `post /create-new-message-in-group-talk` – метод для добавления нового сообщения в обсуждение;
- `get /get-group-talks-by-id` – метод для получения списка обсуждений;
- `post /add-new-comment-into-post/:groupId/:postId` – метод для добавления нового комментария;
- `put /like/:groupId/:postId` – метод для добавления реакции на публикацию;
- `put /unlike/:groupId/:postId` – для удаления реакции на публикацию.

Программный код данных методов приведен в приложении Б.

Важно отметить, что каждый из разработанных классов, помечен декоратором `@UseGuards(JwtAuthGuard)`, который используется для того, чтобы запретить доступ к данным методам неавторизованным пользователям. В качестве механизма авторизации используется модель JWT-токенов, которая подразумевает создание токена в момент авторизации или регистрации пользователя на сайте. Данный токен содержит информацию о пользователе, которому он принадлежит. Токен сохраняется в cookie-файлах пользователя после его авторизации и хранится там до истечения срока хранения.

3.2 Реализация клиентского приложения

Для реализации клиентского приложения используется фреймворк Next JS [8]. Он необходим для проектов, которым требуется высокий уровень индексации поисковыми роботами. В число таких проектов входят и социальные сети по интересам.

Программный код страницы авторизации приведен в листинге 10.

Листинг 10 – Программный код страницы авторизации

```
export default function Login(): JSX.Element {
  const sendLoginData = (data: {login: string, password: string})
=> {
    const login = data.login;
    const pass = data.password;
    sendLogData({email: login,password: pass});
  }
  return (
    <div className={s.cardWrapper}>
      <div className={s.card}>
        <Head>
          <title>{t('Вход')}

```

```

        </div>
      </form>
    </div>
  </div>
)

```

Данный код содержит метод `sendLoginData`, который отвечает за отправку данных на сервер, а также переадресацию пользователя в личный кабинет в случае успешной проверки данных на сервере.

Программный код страницы личного кабинета пользователя приведен в листинге 11. Полный код всех функций страницы авторизации приведен в приложении В.

Листинг 11 – Программный код страницы личного кабинета

```

function Profile(): JSX.Element {
  const [choosedEventForInvite, setChoosedEventForInvite] = useState<number>();
  const [isEditTagOpen, setIsEditTagOpen] =
    useState<boolean>(false);
  const changeAddingImageStatus = (status: boolean) => {
    if (currentUser.login === authedUser?.login) {
      setAddingImageStatus(() => status);
    }
  };
  const handleSendInvite = () => {
    sendInviteToUser({ userId: currentUser.userId, eventId:
      choosedEventForInvite });
    setIsInviteModal(false);
  };
  const handleOpenEditTag = () => {
    if (authedUser.userId ===
      currentUser.userId) setIsEditTagOpen(true);
    return;
  };
  useEffect(() => {
    setLoginLoading(false); getMyDialogs(true);
    return () => setCurrentProfileUser({} as User); }, []);
  useEffect(() => {
    if (currentUser.userId) {
      getUserPlaces({ userId: currentUser.userId });
    }
  }, [currentUser]);
  if (currentUserLoaded) {
    return (
      <PageContainer>
        <ProfileView
          setIsInviteModal={setIsInviteModal}
          setIsEditTagOpen={setIsEditTagOpen}
          handleStartDialog={handleStartDialog}
          onChangeInputImage={onChangeInputImage}
          handleSaveNewStatus={handleSaveNewStatus}
          changeAddingImageStatus={changeAddingImageStatus}
          onAddingModalClick={onAddingModalClick}
          handleAddUserIntoMarked={handleAddUserIntoMarked}
          handleOpenEditTag={handleOpenEditTag}
          currentUserPlaces={currentUserPlaces}

```

```

        />
      </PageContainer>
    )
  } else return <CustomLoader />
}

```

Данный код отвечает за создание HTML разметки, необходимой для отображения интерфейса личного кабинета пользователя, а также всех элементов с которыми пользователь может взаимодействовать (поля ввода, кнопки, выпадающие списки). Также в коде объявлены вспомогательные функции, которые являются необходимыми для корректной работы каждого элемента интерфейса.

Например, метод `handleAddUserIntoMarkedList` для добавления пользователя в список закладок.

Метод `handleSendInvite` используется для отправки приглашения на мероприятие.

Метод `handleStartDialog` используется для создания нового диалога с пользователем.

Метод `handleSaveNewStatus` используется для обновления статуса пользователя.

Полный программный код страницы личного кабинета пользователя приведён в приложении В.

Программный код страницы мессенджера приведён в листинге 12.

Листинг 12 – Программный код интерфейса для страницы мессенджера

```

export default function Messenger(): JSX.Element {
  const activeChat$ = useStore(activeChat);
  const isMobile$ = useStore(isMobile);
  const isChatNeededToShow = Boolean(activeChat$.dialogId);
  const sendForm = (inputValue: string) => {
    if(inputValue.length > 0)
    {createdSendMessageAndUploadActiveChat(inputValue)};};
  return(
    <PageContainer>
      { !isMobile$
        ?
          <div className={s.messengerContent}>
            <div className={` ${s.chatList} ${s.block}`}>
              <ChatList/>
            </div>
            <div className={` ${s.chatZone}`}>

```

```

        {isChatNeededToShow ? <ChatZone
activeChat$={activeChat$} /> :
        <div className={s.noChat}>
            <h3 className={s.alert}>Выберите
диалог...</h3>
        </div>
        }
    </div>
</div>:
<div className={s.mobileMessengerContent}>
    {
        !isChatNeededToShow
        ? <MobileChatList />
        :
        <div className={` ${s.mobileChatZone}`}>
            <div className={s.mobileChatZoneContent}>
                <ChatZone activeChat$={activeChat$} />
            </div>
        </div>
    }
</div>
}
</PageContainer>
)
}

```

Данный код содержит HTML разметку страницы мессенджера, а также объявленные состояния и методы для их изменения, а также метод для отправки нового сообщения.

Состояние `activeChat$` содержит информацию об активном диалоге, а также историю сообщений, информацию о собеседнике.

Метод `handleSendForm` используется для отправки запроса на сервер с целью зарегистрировать пользователя в системе.

Полный программный код страницы мессенджера приведен в приложении В.

Программный код страницы встреч приведен в листинге 13.

Листинг 13 – Программный код страницы встреч

```

export default function Meetings() {
    const [currentMenu, setCurrentMenu] = useState<number>(1);
    const currentMeetings$ = useStore(currentMeetings);
    const authedUser$ = useStore($user);
    const router = useRouter();
    const handleGoToMeetingRoom = (meeting: IMeeting) => {
        setSelectedMeeting(meeting);
        router.push(`/meetingsRoom/${meeting.meetingId}`);
    };
    useEffect(() => {

```

```

        getAllMeetings();
    }, []);
    return (
        <PageContainer>
            <MeetingsPageView
                handleGoToMeetingRoom={handleGoToMeetingRoom}
                currentMeetings={currentMeetings$}
                currentMenu={currentMenu}
                setCurrentMenu={setCurrentMenu}
                authedUser={authedUser$}
            />
        </PageContainer>
    )
}

```

В данном коде содержится набор состояний и методов для управления этими состояниями.

Метод `handleGoToMeetingRoom` используется для того, чтобы пользователь мог перейти к просмотру информации о встрече.

Программный код страницы для просмотра встреч приведен в листинге 14.

Листинг 14 – Программный код страницы для просмотра встреч

```

export default function MeetingsRoom() {
    const handleFileUpload = (event: ChangeEvent<HTMLInputElement>) =>
    {uploadFileToMediaMeeting({event,meetingId:selectedMeeting$.meeting
    Id});}
    const handleRegInCurrentMeeting = (meetingId: string) => {
        if (!isUserInParticipants) { handleRegInMeeting(meetingId);
        } else {
            addNotification({
                text: "Вы уже в списке участников.",textColor: "white",
                time: 3000,color: 'orange'}});
        const handleAddComment = () => {
            if (isUserInParticipants) { setIsAddCommentModal(true);
            } else {
                addNotification({
                    text: "Сначала запишитесь на встречу.",textColor:
            return (
                <PageContainer>
                    <>
                        <MeetingsRoomPageView
                            selectedMeeting={selectedMeeting$}
                            handleOpenAllFiles={handleOpenAllFiles}
                            isUserInParticipants={isUserInParticipants}
                        />
                        <CustomModal
                            isDisplay={isAddCommentModal}
                            changeModal={setIsAddCommentModal}
                            actionConfirmed={setIsAddCommentModal}
                            typeOfActions="none"
                            title="Добавить комментарий"
                        >
                            <AddCommentIntoMeeting />
                    </>
                </PageContainer>
            )
        }
    }
}

```

```

        </CustomModal>
        <CustomModal
          isDisplay={isOpenAllFilesModal}
          changeModal={setIsOpenAllFilesModal}
          actionConfirmed={setIsOpenAllFilesModal}
          typeOfActions="none"
          title="Все медиа файлы со встречи"
        >
          <MeetingsFilesList
            files={selectedMeeting$?.files}
          />
        </CustomModal>
      </>
    </PageContainer>
  )
}

```

Данный программный код содержит элементы для создания HTML разметки для страницы подробного просмотра информации о встрече, а также обработчики событий и методы для организации взаимодействия пользователей в рамках текущей встречи.

Например, метод `handleRefInCurrentMeeting` используется для добавления текущего пользователя в список участников встречи.

Метод `addNotification` используется для отображения всплывающего уведомления чтобы пользователь мог отслеживать успешность выполнения своих действий.

Программный код страницы для просмотра подробной информации о встрече приведен в приложении В.

Программный код страницы поиска людей приведен в листинге 15.

Листинг 15 – Программный код страницы поиска людей

```

export default function Peoples(): JSX.Element {
  const [currentChapter, setCurrentChapter]
    =useState<string>('searching');
  const [isHintOpen, setIsHintOpen] = useState<boolean>(false);
  const currentWall = useStore(currentWall$);const router =
    useRouter();
  const handleGoToLink = (isGroup: boolean, linkId: number) => {
    if (isGroup) {router.push(`/groups/${linkId}`);} else {
      router.push(`/profile/${linkId}`)};};
  useEffect(() => {getCurrentWall();getInterests();}, []);
  return(
    <PageContainer>
      <div className={s.peoples}><Head><title>Meetins - Люди</title>
      <meta name="description" content="Ищите новые знакомства!"
      key="desc" />
    </div>
  )
}

```

```

        <meta property="og:title" content="Только у нас
Вы сможете найти себе хороших друзей и пообщаться!" />
<metaname="keywords"content="meetins, meetin-s, Meetins, Meetin-s,
знакомства, meetings, meet, люди, peoples, meetins seven peoples"
/><meta property="og:description"content="Присоединяйтесь прямо
сейчас, ищите людей."/></Head><div className={s.chapters}>
  <button onClick={() => setCurrentChapter('searching')}
    className={currentChapter === 'searching' ?
s.selectedChapter : s.notSelectedChapter}>Поиск людей</button>
  <FaRegCircleQuestion className={s.hint} onClick={() =>
setIsHintOpen(true)}></button><button onClick={() =>
setCurrentChapter('groups')}
  className={currentChapter === 'groups' ? s.selectedChapter :
s.notSelectedChapter}>Сообщества</button><button onClick={() =>
setCurrentChapter('list')}className={currentChapter === 'list' ?
s.selectedChapter : s.notSelectedChapter}>Новости</button></div>
<div className={s.activeChapter}>{
currentChapter === 'searching' ? <SearchingPeople />
:currentChapter === 'list' ? <LentaList
wallPosts={currentWall}handleGoToLink={handleGoToLink}/> :
currentChapter === 'groups' && <GroupsList />}
  </div>
</div>
</PageContainer> )}

```

В данном коде содержится разметка для страницы поиска людей на сайте, а также соответствующие методы, обеспечивающие работоспособности каждого элемента интерфейса.

Например, метод `handleGoToProfileLink` содержит логику для переадресации пользователя в личный кабинет пользователя, выбранного из списка найденных пользователей.

Состояние `currentWall` содержит информацию о текущем списке найденных пользователей после применения фильтров.

Полный программный код реализации интерфейса страницы для поиска людей на сайте приведен в приложении В.

Программный код реализации интерфейса для страницы событий приведен в листинге 16.

Листинг 16 – Программный код интерфейса для страницы событий

```

export default function EventInfo(): JSX.Element {
  const currentEventById$ = useStore(currentEventById);
  const currentEventCommentsById$ =
useStore(currentEventCommentsById);
  const authedUser$ = useStore($user);
  const { query } = useRouter();
  const addUserEventHandler = (id: number) => {

```

```

        if
        (authedUser$.events.includes(String(currentEventById$.id))) {
            addNotification({ text: "Уже есть в вашем списке",
time: 3000, color: 'orange', textColor: "black" });
            return;
        } addUserEvent(id);
    }
    useEffect(() => {
        if (query.id) {
            getEventById(String(query.id));
            getCommentsForEventById(String(query.id));
        }
    }, [query]);
    useEffect(() => {
        return () => {
            setCurrentEventById(null);
        }
    }, []);
    return(
        <PageContainer>
            {currentEventById$ ?
                <EventBlock
                    addUserEvent={addUserEventHandler}
                    currentEventById={currentEventById$}
                    commentsEvent={currentEventCommentsById$}
                /> : <Loader />}
        </PageContainer>
    );
};

```

Данный код содержит элементы создания HTML разметки необходимой для отображения интерфейса для просмотра информации о текущем, выбранном пользователем событии, а также обработчики событий, состояния и методы необходимые для обеспечения работоспособности интерфейса.

Например, метод `addUserEventHandler` содержит логику для добавления текущего события в список закладок пользователя.

Метод `addNotification` содержит логику для отображения всплывающего уведомления пользователю, с целью проинформировать о статусе завершения той или иной операции (например, успешно или неуспешно событие было добавлено в список закладок).

Состояние `currentEventInfo$` содержит информацию о текущем событии, его организаторе и других необходимых аспектах события.

Полный код реализации интерфейса и всех методов необходимых для страницы событий приведены в приложении В.

Программный код реализации интерфейса для страницы сообществ приведен в листинге 17.

Листинг 17 – Программный код интерфейса для страницы сообществ

```
const router = useRouter();const groupInfo$ = useStore(groupInfo);
const groupMembersInfo$ = useStore(groupMembersInfo);
const authedUser$ = useStore($user);
const isAutherUserCreator = authedUser$?.userId
==groupInfo$?.creatorId;
    const [selectedTalkId, setSelectedTalkId] = useState<number>();
    const [activePostId, setActivePostId] = useState<number>();
const videoPhotoAttachmentsInfo
=destrucutreFilesInGroupPost(groupInfo$);
    const handleOpenGroupSettings = () => {
        setModals({...modals,isSettingsGroupOpen: true});};
const handleOpenComments = (postId: number) => {
setActivePostId(postId);setModals({modals,isCommentsModalOpen:
true});};
    const handleOpenAddingPost = () => {
        setModals({...modals,isAddingPostModalOpen: true});};
    const handleOpenTalks = () => {
        setModals({...modals,isTalksOpen: true});};
        setModals({...modals,isTalkCreationOpen: true});};
    const handleSuccessSubmitTalkCreation = () => {
        setModals({...modals,isTalkCreationOpen: false,isTalksOpen:
true});};
    const handleAddNewMessage = (message: IGroupTalkMessage) => {
        addActiveGroupTalkMessage(message);};
    useEffect(() => {
        if(router.query?.id) {
            getGroupById(+router.query.id);getGroupMembersInfo(+router.query
.id);}
        }, [router.query.id]);
    return (
        <PageContainer><>
<GroupInfoPageView authedUser={authedUser$}groupInfo={groupInfo$}
isAutherUserCreator={isAutherUserCreator}
handleOpenGroupSettings={handleOpenGroupSettings}
isSettingsGroupOpen={modals.isSettingsGroupOpen}
groupMembersInfo={groupMembersInfo$}
videoPhotoAttachmentsInfo={videoPhotoAttachmentsInfo}/><CustomModal
    isDisplay={modals.isSettingsGroupOpen}changeModal={(status) =>
setModals({ ...modals,isSettingsGroupOpen:status })}
actionConfirmed={(status) => setModals({ ...modals,
isSettingsGroupOpen: status
})}typeOfActions="none"title="Управление сообществом">
        <ManageGroup
    /></CustomModal></></PageContainer>)}

```

В данном коде приведены элементы для создания HTML разметки интерфейса для страницы сообществ, а также методы для обеспечения работоспособности каждого элемента данного интерфейса.

Например, метод `handleOpenGroupSettings` содержит логику для отображения модального окна с возможностями управления сообществом, которое появляется в том случае если авторизованный пользователь является непосредственно создателем данного сообщества.

Метод `handleOpenEditingPist` содержит логику для добавления новой публикации в данном сообществе.

Полный программный код реализации элементов интерфейса и всех методов необходимых для страницы сообществ приведены в приложении В.

Программный код вспомогательных методов, обработчиков событий и компонентов также приведены в приложении В.

3.3 Реализация базы данных

В качестве базы данных для реализации приложения была выбрана Mongo DB. Данный выбор объясняется высокой скоростью работы СУБД при частых операциях вставки или модификации данных, что безусловно соответствует разрабатываемой системе [10]. В ходе работы были разработаны все описанные ранее на этапе проектирования коллекции документов и ограничения для них.

Создание коллекций базы данных осуществляется путём использования ОРМ `mongoose`, которая предоставляет встроенные средства для управления базой данных. Каждая коллекция будет представлена в виде схемы с полным описанием свойств каждого документа в коллекции, а также их типов.

Программный код отвечающий за создание схемы для коллекции пользователей приведен в листинге 18.

Листинг 18 – Программный код схемы коллекции пользователей

```
export class User {
  @Prop()
  userId: string;    @Prop({default: false})
  isFilter: boolean;
```

```

@Prop({default: []})
markedUsers: string[];
@Prop()
name: string;
status: string;
@Prop({ default: '' })
phoneNumber: string;
@Prop()
email: string;
@Prop({ default: 'password' })
password: string;
@Prop()
gender: string;
@Prop({ default: 'no-avatar.jpg' })
avatar: string;
@Prop({ default: new Date() })
dateRegister: Date;
login: number;
@Prop({ default: new Date() })
birthDate: Date;
@Prop({ default: 'Секрет' })
city: string;
@Prop({ default: 18 })
age: number;
@Prop({ default: [] })
events: string[];
@Prop({ default: [] })
posts: IPost[];
@Prop({ default: [] })
interests: IInterests[];
@Prop({ default: [] })
outerInvites: IOuterInvites[]; @Prop({ default: [] })
innerInvites: IInnerInvites[]; @Prop({ default: [] })
purchasedOpportunities: number[];
@Prop({
  type: {},
  default: { color: "rgba(42, 132, 251, 0.878)", title:
"Гость" }
})
tag: IUserTag;
};

```

Программный код для создания схемы для коллекции диалогов пользователей приведена в листинге 19. В нём описаны все свойства необходимые для хранения информации о диалоге, а именно идентификатор диалога, массив сообщений и массив пользователей, которые принимают участие в диалоге.

Листинг 19 – Программный код схемы для коллекции диалогов

```

@Schema()
export class Chat {
  @ApiProperty({example: '1', description: 'Уникальный ид'})
  @Prop()

```

```

    dialogId: string;
    @ApiProperty({example: [], description: 'Массив сообщений'})
    @Prop()
    messages: IMessage[];
    @ApiProperty({example: [], description: 'Массив пользователей
userId'})
    @Prop()
    members: string[];
}
export const ChatSchema = SchemaFactory.createClass(Chat);

```

Программный код создания схемы для коллекции сообществ приведен в листинге 20. В нём описаны все свойства документов в коллекции сообществ, такие как идентификатор, название, описание сообщества, массив интересов для данного сообщества, информация о заглавной картинке для сообщества, массив участников сообщества, идентификатор создателя сообщества, список обсуждений в данном сообществе, а также массивы фотографий, видео и других вложений.

Листинг 20 – Программный код схемы для коллекции сообществ

```

@Schema()
export class Group {
    @Prop()
    groupId: number;
    @Prop()
    name: string;
    @Prop()
    description: string;
    @Prop()
    interestsId: string[];
    @Prop({ default: "no-avatar.jpg" })
    mainAvatar: string;
    @Prop({ default: null })
    headAvatar: string;
    @Prop()
    membersId: string[];
    @Prop()
    creatorId: string;
    @Prop({ default: [] })
    posts: IGroupPost[];
    @Prop({ default: [] })
    talks: IGroupTalk[];
    @Prop({ default: [] })
    photos: string[];
    @Prop({ default: [] })
    video: string[];
    @Prop({ default: [] })
    attachments: string[];
}

```

Программный код создания схемы для коллекции встреч, приведен в листинге 21. В нём содержатся все поля документов в данной коллекции, а именно идентификатор встречи, планируемая дата встречи, цели встречи, максимальное количество участников, комментарии для данной встречи, главное изображение для встречи, название встречи, адрес для встречи, массив медиа-файлов со встречи (фотографии, видео).

Листинг 21 – Программный код создания схемы для коллекции встреч

```
@Schema()
export class Meeting {
  @Prop({ default: "meeting" +
    `${Math.floor(Math.random()*9999999)}` })
  meetingId: string;
  @Prop()
  participants: string[];
  @Prop({ default: new Date() })
  date: Date;
  @Prop()
  description: string;
  goal: string;
  @Prop({ default: 25 })
  maxParticipants: string;
  @Prop({ default: [] })
  comments: MeetingComment[];
  @Prop({ default: "no-preview.png" })
  preview: string;
  title: string;
  @Prop()
  address: string;
  @Prop()
  creatorId: string;
  @Prop({ default: [] })
  files: {
    src: string, type: string
  }[];
}
export const MeetingSchema = SchemaFactory.createClass(Meeting);
```

Программный код создания схемы для коллекции, которая хранит информацию об авторах музыки и их композициях на сайте, приведен в листинге 22. В нём содержатся все свойства для документов в данной коллекции, а именно идентификатор автора, псевдоним автора, массив композиций, принадлежащих конкретному автору.

Листинг 22 – Программный код схемы для коллекции музыки

```
@Schema()
export class MusicAuthor {
  @ApiProperty({example: 1, description: 'Уникальный ид юзера'})
```

```

    @Prop()
    authorId: string;

    @ApiProperty({example: "Имя исполнителя", description: 'Имя
исполнителя'})
    @Prop()
    name: string;

    @ApiProperty({example: [], description: 'Массив композиций
автора'})
    @Prop()
    compositions: {
      id: number,
      audioSrc: string,
      imageSrc: string,
      title: string,
      description: string,
      playsNumber: number
    }[];
  }
}
export const MusicAuthorSchema =
SchemaFactory.createClass(MusicAuthor);

```

Программный код схемы для коллекции неподтвержденных пользователей приведен в листинге 23.

Листинг 23 – Программный код схемы для коллекции неподтвержденных пользователей

```

@Schema()
export class UnConfirmedUser {
  @ApiProperty({example: 'Имя', description: 'Имя пользователя'})
  @Prop()
  name: string;
  @ApiProperty({example: 'user@gmail.com', description: 'Email
пользователя'})
  @Prop()
  email: string;
  @ApiProperty({example: 'sdfdf3r4341', description: 'Пароль'})
  @Prop({ default: 'password' })
  password: string;
  @ApiProperty({example: 'М', description: 'Пол пользователя'})
  @Prop()
  gender: string;
}

```

В нём содержатся все свойства документов в данной коллекции, а именно имя пользователя, электронная почта, пароль, пол, город регистрации, а также код подтверждения необходимый для подтверждения учётной записи пользователя.

4 Тестирование web-приложения социальной сети по интересам для совместного общения и развития

4.1 Тестирование белым ящиком

4.1.1 Покрытие операторов

Тестирование белого ящика, также тестирование стеклянного ящика, структурное тестирование – тестирование, которое учитывает внутренние механизмы системы или компонента. Обычно включает тестирование ветвей, маршрутов, операторов [11].

Для проведения тестирования был выбран один из основных модулей реализуемой программы, а именно – мессенджер. Данный модуль нужен для обеспечения коммуникации пользователей на сайте в режиме реального времени, и именно поэтому очень важно, чтобы он работал без ошибок, а также элементы его реализации были максимально эффективными.

Критерий покрытия операторов подразумевает выполнение каждого оператора программы по крайней мере один раз. Для тестирования была выбрана функция отправки нового сообщения в мессенджер. Код метода приведен в листинге 24.

Листинг 24 – Код программы для отправки сообщения

```
async addNewMessage(
    inithiatorJwtData: DecodedJwt,
    dialogId: string,
    content: string,
    type: string
) {
    const message: IMessage = {
        dialogId: dialogId,
        content: content,
        messageId: String(Math.floor(Math.random() * 5000000)),
        sendAt: new Date(),
        senderId: inithiatorJwtData.userId,
        isRead: false,
        avatar: inithiatorJwtData.avatar,
        senderName: inithiatorJwtData.name,
        status: false,
        type: type
    }
}
```

```

    await this.chatModel.updateOne({ dialogId: message.dialogId
}, { $push: { messages: message } } );
    const currentChatState = await this.chatModel.findOne({
dialogId: message.dialogId });

    if (currentChatState) {
        const userSessionForSendingMessage =
this.socketServer.activeFullUsersList
        .filter(el =>
currentChatState.members.includes(el.userId))
        .map(el => el.socketId);

        if (userSessionForSendingMessage) {
            this.socketServer.server.to([...userSessionForSendi
ngMessage]).emit('message', message);
        }
    }
    return
currentChatState.messages[currentChatState.messages.length - 1];
}

```

Блок схема, отражающая работу метода для отправки сообщения, приведена на рисунке 6.

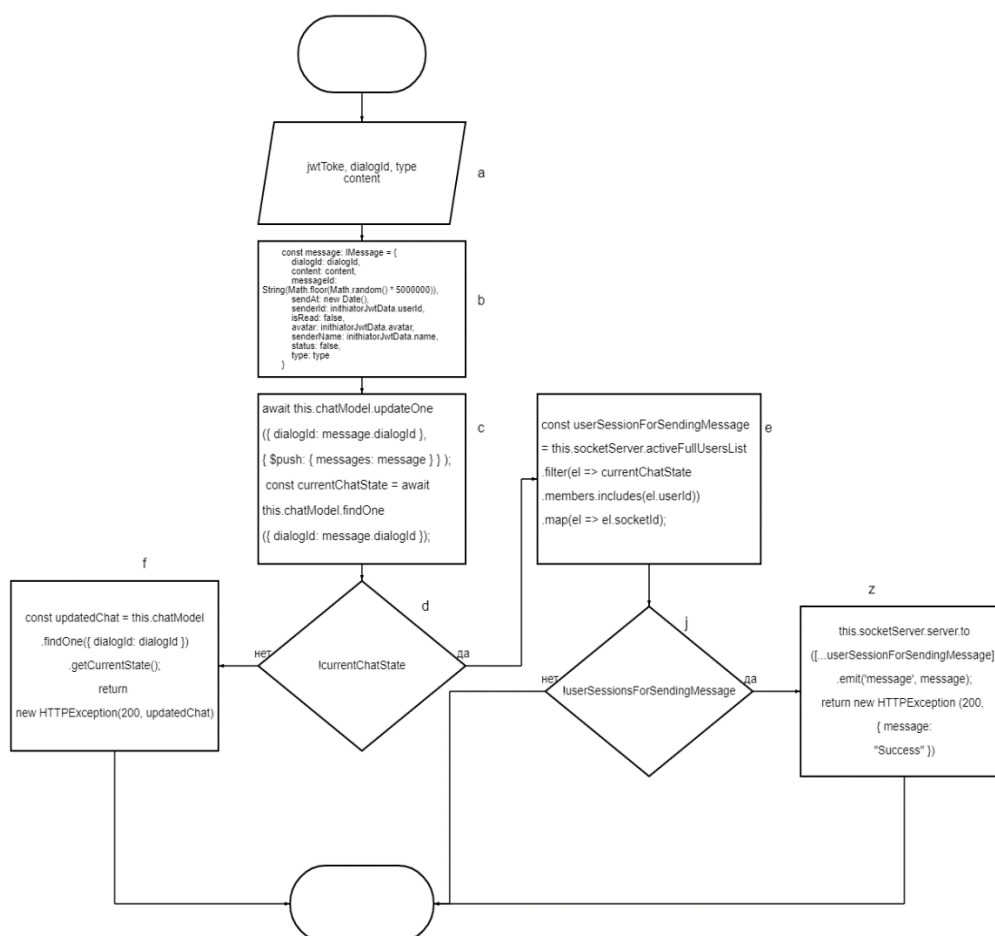


Рисунок 6 – Блок схема работы программы

Чтобы выполнить каждый оператор не менее одного раза, нужно составить несколько тестов со следующими значениями. Все операторы, входные данные и их значения, а также промежуточные значения переменных приведены в таблице 11.

Таблица 11 – Тестирование операторов

№	Операторы	Входные данные и промежуточные значения	Выходные данные
1	b, c, e, z	<pre> jwtToken = «eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImRhbmtpbHBsYXRvbm92MjAwMkRnbWFpbC5jb20iLCJuYW1lIjoia0JTQtdC90LjRgSIsImNpdHkiOiJQZW56YSIsImF2YXRhciI6IjE3MDc4MTQzNTc5MDMuanBnIiwidXNlcklkIjoiaWQ5NzY3NyIsImVudCI6MTcwODUwMjUyOSwiZXhwIjozA4NTA2MTI5fQ.2Nh5birgh7A3iCkw-rbCYFkGWUY95g3xV6ce7iJw8fM» dialogId = «id3875925» content = «Hello» type = «Text» currentChatState = true userSessionForSendingMessage = ['userId398402'] message = { dialogId: dialogId, content: content, sendAt: Date.now(), senderId: «userId29024», isRead: false, status: false } </pre>	<pre> message = { dialogId: dialogId, content: content, sendAt: Date.now(), senderId: «userId29024», isRead: false, status: false } </pre>
2	b, c, f	<pre> jwtToken = «eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImRhbmtpbHBsYXRvbm92MjAwMkRnbWFpbC5jb20iLCJuYW1lIjoia0JTQtdC90LjRgSIsImNpdHkiOiJQZW56YSIsImF2YXRhciI6IjE3MDc4MTQzNTc5MDMuanBnIiwidXNlcklkIjoiaWQ5NzY3NyIsImVudCI6MTcwODUwMjUyOSwiZXhwIjozA4NTA2MTI5fQ.2Nh5birgh7A3iCkw-rbCYFkGWUY95g3xV6ce7iJw8fM» dialogId = «id3875925» content = «Hello» type = «Text» currentChatState = false message = { dialogId: dialogId, content: content, sendAt: Date.now(), } </pre>	false

Исходя из проведенных тестов, можно выяснить, что метод для отправки сообщения пользователю работает корректно при любых входных данных.

4.1.2 Покрытие решений

В соответствии с этим критерием необходимо составить такое число тестов, при которых каждое условие в программе примет как истинное значение, так и ложное значение. Составленные тесты приведены в таблице 12.

Таблица 12 – Покрытие решений

№	Решения	Входные данные	Выходные данные
1	d – да j – да	<pre> jwtToken = «eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImRhbmtpbHBsYXRvbm92MjAwMkNnbWFpbC5jb20iLCJuYW 1lIjo0JTQtdC90LjRgSIsImNpdHkiOiJQZW56YSIsImF2YXRh ciI6IjE3MDc4MTQzNTc5MDMuanBnIiwidXNlcklkIjoiaWQ» dialogId = «id3875925» content = «Hello» type = «Text» currentChatState = true userSessionForSendingMessage = ['userId038952'] message = { dialogId: dialogId, content: content, sendAt: Date.now(), senderId: «userId29024», isRead: false, status: false } </pre>	<pre> message = { dialogId: dialogId, content: content, sendAt: Date.now(), senderId: «userId29024», isRead: false, status: false } </pre>
2	d – нет	<pre> jwtToken = «eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImRhbmtpbHBsYXRvbm92MjAwMkNnbWFpbC5jb20iLCJuYW 1lIjo0JTQtdC90LjRgSIsImNpdHkiOiJQZW56YSIsImF2YXRh ciI6IjE3MDc4MTQzNTc5MDMuanBnIiwidXNlcklkIjoiaWQ» dialogId = «id3875925»content = «Hello» type = «Text» currentChatState = false message = { dialogId: dialogId, content: content,sendAt: Date.now()} </pre>	false
3	d – да j – нет	<pre> jwtToken = «eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImRhbmtpbHBsYXRvbm92MjAwMkNnbWFpbC5jb20iLCJuYW 1lIjo0JTQtdC90LjRgSIsImNpdHkiOiJQZW56YSIsImF2YXRh ciI6IjE3MDc4MTQzNTc5MDMuanBnIiwidXNlcklkIjoiaWQ» dialogId = «id3875925» content = «Hello» type = «Text» currentChatState = true userSessionsForSendingMessages = false message = { dialogId: dialogId,content: content, sendAt: Date.now()} </pre>	<pre> message = { dialogId: dialogId, content: content, sendAt: Date.now(), senderId: «userId29024», isRead: false, status: false } </pre>

В ходе работы было успешно составлено необходимое количество тестов, для того чтобы каждое условие принимало как истинное значение, так и ложное.

4.1.3 Покрытие решений и условий

В соответствии с этим критерием необходимо составить тесты так, чтобы результаты каждого условия выполнялись хотя бы один раз, результаты каждого решения так же выполнялись хотя бы один раз, и каждый оператор должен быть выполнен хотя бы один раз.

Если после составления тестов останутся не покрытые операторы, то нужно дополнить набор тестов таким образом, чтобы каждый оператор выполнялся не менее одного раза [12]. Составленные тесты приведены в таблице 13.

Таблица 13 – Покрытие решений и условий

№	Условие	Входные данные	Выходные данные
1	a,b,c,d (да), e, j (да), z	jwtToken=«eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnB1bGpHbS5YXRvbm92MjAwMkNbnWFpbC5jb20iLCJuYW1lIjo0JTQtdC90LjRgSIsImNpdHkiOiJQZW56YSIsImF2YXRhcil6IjE3MDc4MTQzNTc5MDMuanBnliwidXNlcklkIj» dialogId = «id3875925» currentChatState = true userSessionForSendingMessage = ['userId038952'] message = {dialogId: dialogId, content: content, sendAt: Date.now(),senderId: «userId29024», isRead: false, status: false}	message = { dialogId: dialogId, content: content, sendAt: Date.now(), senderId: «userId29024», isRead: false, status: false}
2	a,b,c,d (нет), f	jwtToken=«eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnB1bGpHbS5YXRvbm92MjAwMkNbnWFpbC5jb20iLCJuYW1lIjo0JTQtdC90LjRgSIsImNpdHkiOiJQZW56YSIsImF2YXRhcil6IjE3MDc4MTQzNTc5MDMuanBnliwidXNlcklkIj» dialogId = «id3875925» content = «Hello» type = «Text» currentChatState = true, message = {dialogId: dialogId, userSessionForSendingMessage = ['userId038952'] content:content,sendAt:Date.now(),senderId: «userId29024», isRead: false, status: false}	false
3	a,b,c,d (да), e, j (нет)	jwtToken=«eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnB1bGpHbS5YXRvbm92MjAwMkNbnWFpbC5jb20iLCJuYW1lIjo0JTQtdC90LjRgSIsImNpdHkiOiJQZW56YSIsImF2YXRhcil6IjE3MDc4MTQzNTc5MDMuanBnliwidXNlcklkIj» dialogId = «id3875925» content = «Hello» type = «Text» currentChatState = true userSessionForSendingMessage = ['userId038952'] message = {dialogId: dialogId, content: content, sendAt: Date.now(),senderId: «userId29024», isRead: false, status: false}	message = { dialogId: dialogId, content: content, sendAt: Date.now(), senderId: «userId29024», isRead: false, status: false}

Составленные тесты полностью покрывают все операторы, что исключает необходимость составления дополнительных тестов, которые бы покрыли недостающие операторы.

4.2 Метрики кода

4.2.1 Метрика SLOC

Количество строк кода (англ. Source Lines of Code – SLOC) – это метрика программного обеспечения, используемая для измерения его объёма с помощью подсчёта количества строк в тексте исходного кода [13]. Проведем подсчет данной метрики в функциях `addNewMessage`, `sendFileToChat`, `getUserDialogs`, служащие для отправки текстового сообщения пользователю в чат, отправки вложения в виде файла в чат и получения списка всех чатов в который участвует пользователь. Функция приведена в листинге 25.

Листинг 25 – Методы `addNewMessage`, `sendFileToChat`, `getUserDialogs`

```
async addNewMessage(
    initiatorJwtData: DecodedJwt,
    dialogId: string,
    content: string,
    type: string
) {
    await this.chatModel.updateOne({ dialogId: message.dialogId }, { $push: { messages: message } });
    const currentChatState = await this.chatModel.findOne({
        dialogId: message.dialogId });

    if (currentChatState) {
        const userSessionForSendingMessage =
            this.socketServer.activeFullUsersList
                .filter(el =>
                    currentChatState.members.includes(el.userId))
                .map(el => el.socketId);

        if (userSessionForSendingMessage) {
            this.socketServer.server.to([...userSessionForSendingMessage]).emit('message', message);
        }
    }
    return
    currentChatState.messages[currentChatState.messages.length - 1];
}

async sendFileToChat(file: any, request: Request) {
    const decodedJwt = await
        this.helpJwtService.decodeJwt(request);
    if (file.size > 5300000) {
        throw new HttpException({ message: «Слишком большой
        размер файла»}, 400);
    } else {
```

```

        const addedMessages = await
this.addNewMessage(decodedJwt, request.body.dialogId,
file.filename, file.mimetype);
        throw new HttpException(addedMessages, 200);
    }
}

    const shortDialogsForUser = await
Promise.all(findedDialogs.map(async (eachDialog) => {
        const userChatId = eachDialog.members.filter(memberId
=> memberId !== requestedUser.userId);
        const userChatIdInfo: User = await
this.userService.getUserByUserId(userChatId[userChatId.length-1]);
        return {
            dialogId: eachDialog.dialogId,
            userName: userChatIdInfo.name,
            userAvatar: userChatIdInfo.avatar,
            isRead: true,
            content: eachDialog.messages[0].content,
            messages: eachDialog.messages
        }
    })))
    throw new HttpException(shortDialogsForUser, 200);
}

```

Физическими строками считаются все непустые строки текстового файла с кодом программы.

Измеряя логические строки кода, предпринимается попытка посчитать количество собственно операторов в рассматриваемой программе.

4.2.2 Цикломатическая сложность

При вычислении цикломатической сложности используется граф потока управления программы. Узлы графа соответствуют неделимым группам команд программы, они соединены ориентированными рёбрами, если группа команд, соответствующая второму узлу, может быть выполнена непосредственно после группы команд первого узла [14].

Граф потока управления программы для рассматриваемой функции приведен на рисунке 7.

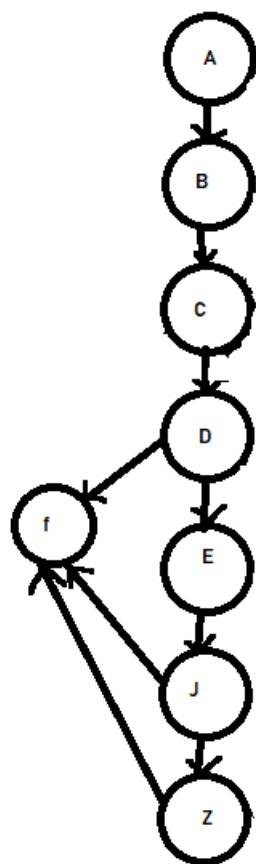


Рисунок 7 – Граф потока управления

Для вычисления цикломатической сложности $V(G)$ применяется следующая формула:

$$V(G) = e - n + 2p,$$

где e – количество дуг;

n – количество вершин;

p – число компонент связности.

Таким образом значение цикломатической сложности для данной программы равно 3.

4.2.3 Метрика Холстеда

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы [15].

Основу метрики Холстеда составляют четыре измеряемые характеристики:

- $nuoprtr (n1)$ – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций;
- $nuoprnd (n2)$ – число уникальных операндов программы;
- $porptr (N1)$ – общее число операторов в программе;
- $porrnd (N2)$ – общее число операндов в программе.

Операторы и операнды приведены в таблицах 14 и 15 соответственно.

Таблица 14 – Операторы

Операторы	Число вхождений
=	10
!==	1
await	9
.	54
if	3
–	2
*	1
findOne	1
includes	1
to	1
emit	1
decodeJwt	2
httpException	3
getUserByUserId	2
$n1 = 17$	$N2 = 119$
return	2
,	25

Таблица 15 – Операнды

Операнды	Число вхождений
message	4
currentChatState	5

Продолжение таблицы 15

userSessionForSendingMessage	3
decodedJwt	5
addedMessages	2
findedDialogs	2
requestedUser	2
shortDialogsForUser	2
request	5
userChatId	5
userChatIdInfo	3
file	4
inithiatorJwtData	4
dialogId	9
content	3
type	2
n2 = 16	N2 = 60

Длина программы, значение которой равно 179, рассчитывается по формуле:

$$N = N1 + N2.$$

Словарь программы, значение которого равно 33, рассчитывается по формуле:

$$n = n1 + n2.$$

Объем программы, значение которого равно 904.45, рассчитывается по формуле:

$$HV = N \times \log_2(n).$$

Потенциальный объем программы, значение которого равно 11.6, рассчитывается по формуле:

$$V^* = (2 + n') \times \log_2(2 + n').$$

Теоретический уровень программы, значение которого равно 0.012, рассчитывается по формуле:

$$L' = V \times HV.$$

Сложность программы, значение которой равно 31.875, рассчитывается по формуле:

$$HD = \frac{n1}{2} \times \frac{N2}{n2}.$$

В данном случае, сложность программы равняется 31.875, что является хорошим значением, так как исследования показывают, что программа является сложной в понимании и поддержке если значение сложности превышает 100.

Данное значение показывает примерное количество тестов, которое необходимо для полного покрытия кода.

4.2.4 Метрика Чепина

Суть метода состоит в оценке информационной прочности отдельно взятого программного модуля с помощью анализа характера использования переменных из списка ввода-вывода [16].

Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы:

- группа Р – вводимые переменные для расчетов и для обеспечения вывода;
- группа М – модифицируемые, или создаваемые внутри программы переменные;
- группа С – переменные, участвующие в управлении работой программного модуля (управляющие переменные);
- группа Т – не используемые в программе («паразитные») переменные.

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе.

Метрика Чепина рассчитывается по следующей формуле:

$$Q = a1 \times P + a2 \times M + a3 \times C + a4 \times T,$$

где a_1 – весовой коэффициент;
 a_2 – весовой коэффициент;
 a_3 – весовой коэффициент;
 a_4 – весовые коэффициент.

Весовые коэффициенты использованы для отражения различного влияния на сложность программы каждой функциональной группы. По мнению автора метрики, наибольший вес, имеет функциональная группа С, так как она влияет на поток управления программы. Весовые коэффициенты остальных групп распределяются следующим образом: $a_1 = 1$, $a_2 = 2$, $a_4 = 0.5$. Весовой коэффициент группы Т не нулевой, поскольку «паразитные» переменные не увеличивают сложность потока данных программы, но иногда затрудняют ее понимание.

Значение метрики SLOC равняется 60 физическим строкам кода и 39 логическим. Это нормальные значения для одной функции.

Значение цикломатическая сложности равняется 3. Это значение не превышает 10, следовательно, функция не является структурно ложной.

Значение уровня программы согласно метрике Холстеда равняется 0.012. Уровень программы имеет значение ниже среднего по данным измерений из книги Холстеда.

Значение метрики Чепина равняется 61. Если в результате изменения рассматриваемых функций значение метрики значительно увеличиться, это будет означать низкое качество изменений.

4.3 Тестирование интерфейса Мессенджера

4.3.1 Таблица тестирования

Для проведения тестирования интерфейса был выбран функционал мессенджера. Тестирование интерфейса подразумевает проверку корректности отображения различных элементов (кнопки, поля ввода, текст), а также

корректность расположения элементов при изменении расширения экрана пользователя.

Для тестирования интерфейса были выделены следующие критерии:

- корректное позиционирование элементов при изменении расширения с на десктопный и мобильный форматы (1);
- читабельность текстового содержимого страниц при изменении расширения на десктопный и мобильный форматы (2);
- отзывчивость интерфейса (3).

Результаты тестирования мессенджера, приведены в таблице Г.1.

5 Планирование и расчёт бюджета

Для планирования и расчёта графика работ по проекту будет использоваться диаграмма Ганта. Данная диаграмма учитывает все этапы реализации проекта, а также сроки, в которые проводится работа над каждым из них. Разработанная диаграмма ганта приведена на рисунке 8.

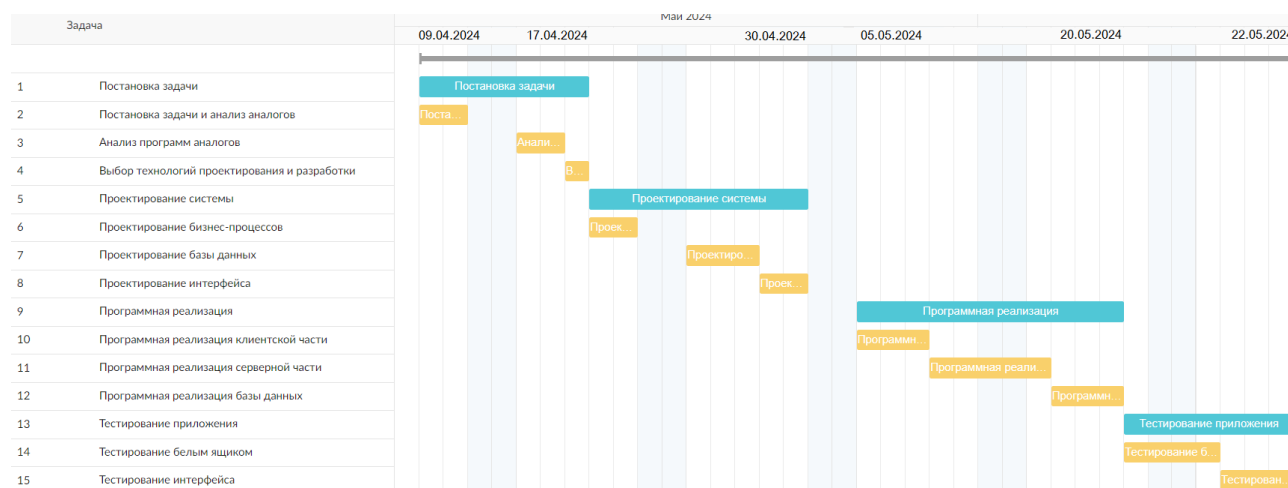


Рисунок 8 – Диаграмма Ганта

На данной диаграмме представлены следующие этапы:

Постановка задачи – данный этап состоит из следующих подзадач, а именно: постановка задачи и анализ аналогов, анализ программ аналогов, выбор технологий проектирования и разработки.

Проектирование системы – данный этап состоит из следующих подзадач, а именно: проектирование бизнес-процессов, проектирование базы данных, проектирование интерфейса.

Программная реализация – данный этап состоит из следующих подзадач, а именно: программная реализация клиентской части, программная реализация серверной части и программная реализация базы данных.

Тестирование приложения – данный этап состоит из следующих подзадач, а именно: тестирование белым ящиком и тестирование интерфейса.

Также, был рассчитан бюджет на разработку, с учетом оплаты каждому из специалистов, принимающих участие в разработке данной системы. По итогам

расчетов, общий бюджет на разработку составил 70.100 рублей. Более детализированные расчеты приведены на рисунке 9.

Задача	Исполнитель	Статус	Затраченное время	Фактические затраты
Постановка задачи и анализ аналогов		● Выполнено	40ч	12.000р
Постановка задачи	● Аналитик	● Выполнено	16ч	4.800р
Анализ программ аналогов	● Аналитик	● Выполнено	16ч	4.800р
Выбор технологий проектирования и разра	● Аналитик	● Выполнено	8ч	2.400р
Проектирование системы		● Выполнено	72ч	21.600р
Проектирование бизнес-процессов	● Архитектор	● Выполнено	20ч	6.000р
Проектирование базы данных	● Архитектор	● Выполнено	32ч	9.600р
Проектирование интерфейса	● Архитектор	● Выполнено	20ч	6.000р
Программная реализация		● Выполнено	94ч	26.900р
Программная реализация клиентской части	● Frontend-разработчик	● Выполнено	26ч	6.500р
Программная реализация серверной части	● Backend-разработчик	● Выполнено	44ч	13.200р
Программная реализация базы данных	● Backend-разработчик	● Выполнено	24ч	7.200р
Тестирование приложения		● Выполнено	48ч	9.600р
Тестирование белым ящиком	● Тестировщик	● Выполнено	24ч	4.800р
Тестирование интерфейса	● Тестировщик	● Выполнено	24ч	4.800р

Рисунок 9 – Расчет бюджета на разработку

Для каждого исполнителя, была определена ставка, которую он получает за отработанные часы, данные об этих ставках приведены на рисунке 10.

Исполнитель	Стоимость
● Аналитик	в час ▼ 300р
● Архитектор	в час ▼ 300р
● Frontend-разработчик	в час ▼ 250р
● Backend-разработчик	в час ▼ 300р
● Тестировщик	в час ▼ 200р

Рисунок 10 – Список ставок для исполнителей

На рисунке 10 приведены несколько исполнителей, а именно: аналитик, архитектор, frontend-разработчик, backend-разработчик, тестировщик.

Для каждого из исполнителей определена ставка за каждый час работы: 300 рублей, 300 рублей, 250 рублей, 300 рублей и 200 рублей соответственно.

Заключение

В ходе работы было спроектировано и впоследствии разработано Web-приложение социальной сети по интересам для совместного общения и развития. Была разработана диаграмма Ганта, а также осуществлено планирование бюджета необходимого для разработки приложения. Данное приложение было отлажено и протестировано, в ходе чего было выявлено, что получившийся проект отвечает современным стандартам, а также работает эффективно, без ошибок и сбоев.

Список использованных источников

1. Социальная сеть ВКонтакте // [Электронный ресурс] / VK. URL: vk.com (дата обращения: 10.04.2024)
2. Социальная сеть для любителей музыки // [Электронный ресурс] / FM. URL: last.fm (дата обращения: 10.04.2024)
3. Мартин Фаулер. Основы UML. – 3-е изд. – СПб: Символ Плюс, 2004. – 181 с.
4. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // [Электронный ресурс] / Хабр. URL: <https://habr.com/ru/post/566218/> (дата обращения: 10.04.2024)
5. Кайл Бэнкер MongoDB в действии. – 2-е изд. – СПб: ДМК Пресс, 2012. – 395 с.
6. Беликов Д.А. MongoDB полное руководство. – 1-е изд. – СПб: ДМК Пресс, 2020. – 540 с.
7. Основы Mongo DB // [Электронный ресурс] / Metanit. URL: <https://metanit.com/nosql/mongodb>. (дата обращения: 10.04.2024)
8. Документация Nest JS // [Электронный ресурс] / NestJsDoc. URL <https://docs.nestjs.com>. (дата обращения: 10.04.2024)
9. Документация ORM Mongoose // [Электронный ресурс] / MongooseDoc. URL <https://mongoosejs.com/docs>. (дата обращения: 10.04.2024)
10. Документация Next JS // [Электронный ресурс] / NextDoc. URL <https://nextjs.org/docs>. (дата обращения: 10.04.2024)
11. Продвинутый Mongo DB // [Электронный ресурс] / MongodbDoc. URL <https://www.mongodb.com/developer/expertise-levels/advanced>. (дата обращения: 10.04.2024)
12. Тестирование ПО – Белый ящик // [Электронный ресурс] / Zaptest. URL: <https://www.zaptest.com/ru>. (дата обращения: 10.04.2024)

13. Тестирование ПО – Методы // [Электронный ресурс] / Unetway. URL: <https://unetway.com/tutorial/testing-software-methods>. (дата обращения: 10.04.2024)
14. Тестирование ПО – Метрика SLOC // [Электронный ресурс] / Studref. URL: https://studref.com/388370/informatika/metriki_slozhnosti_programmnogo_obespecheniya_informatsionnyh_sistem. (дата обращения: 10.04.2024)
15. Тестирование ПО – Цикломатическая сложность // [Электронный ресурс] / TestEng. URL: <https://testengineer.ru/>. (дата обращения: 10.04.2024)
16. Тестирование ПО – Метрика Холстеда // [Электронный ресурс] / Elibrary. URL: <https://www.elibrary.ru>. (дата обращения: 10.04.2024)
17. Тестирование ПО – Метрика Чепина // [Электронный ресурс] / NaJournal. URL: <https://na-journal.ru/8-2023-informacionnye-tehnologii>. (дата обращения: 10.04.2024)

Приложение А
(рекомендуемое)
Проектирование интерфейса

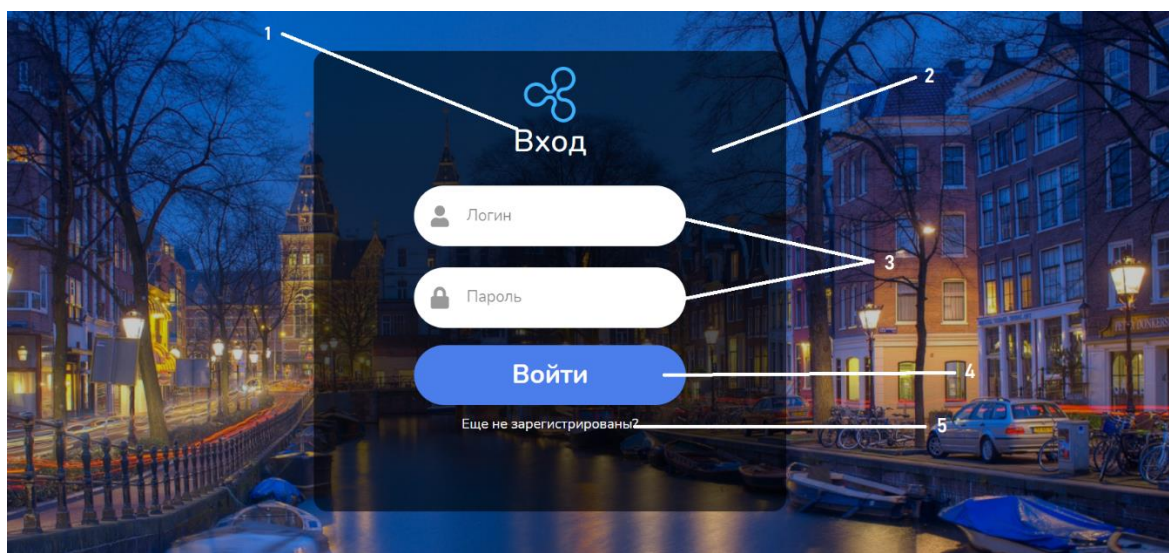


Рисунок А.1 – Страница входа

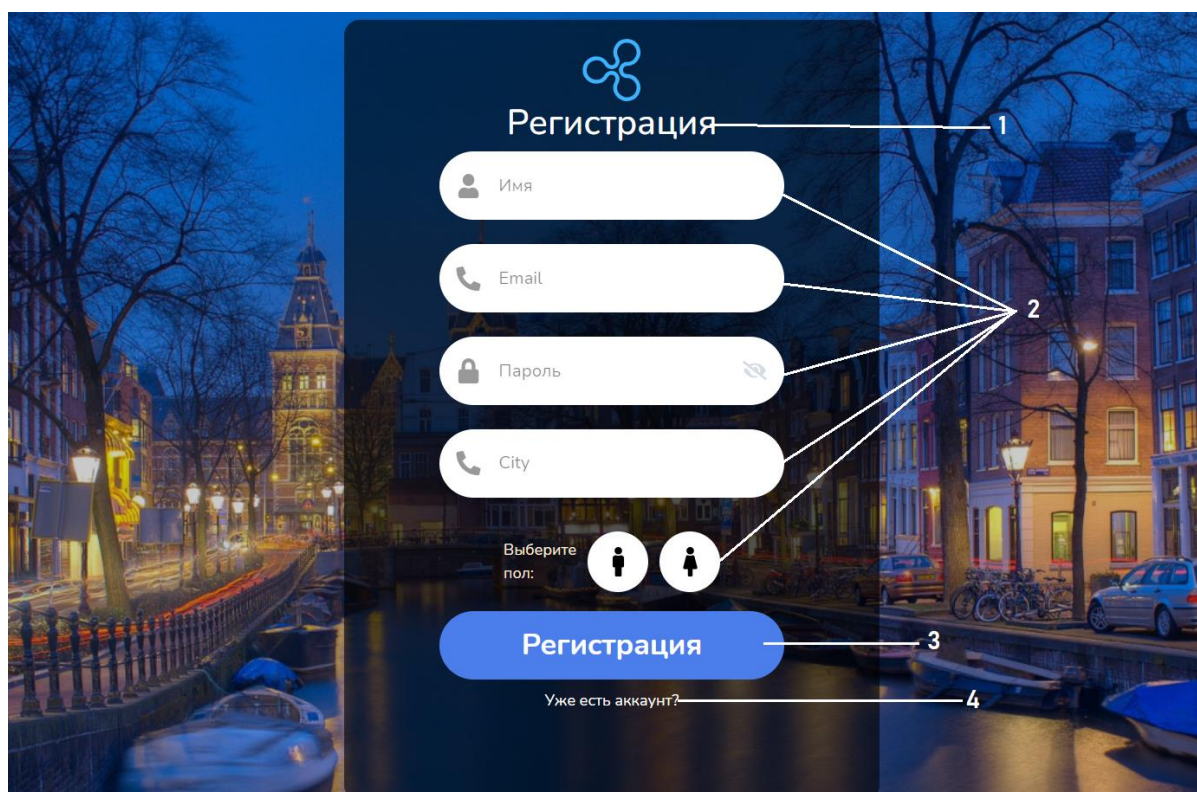


Рисунок А.2 – Страница регистрации

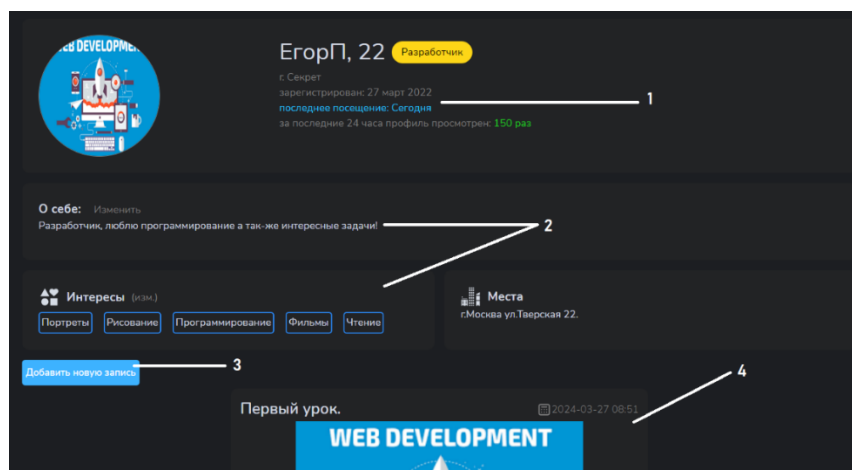


Рисунок А.3 – Страница профиля

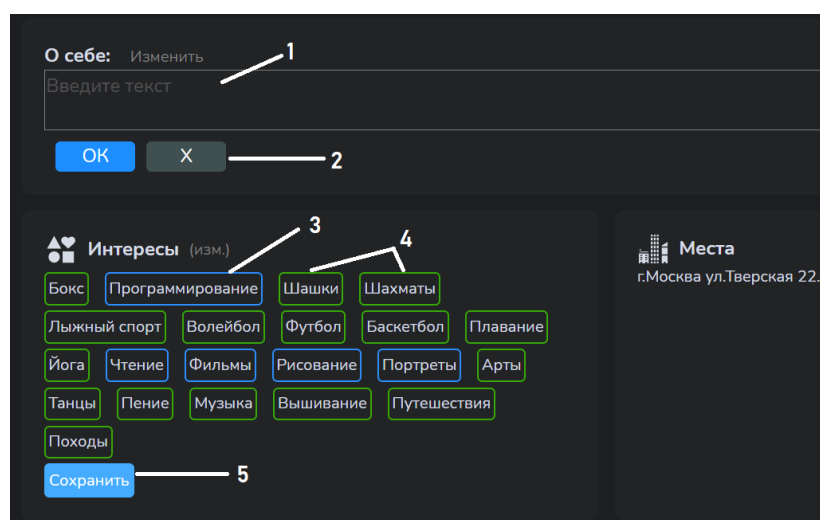


Рисунок А.4 – Интерфейс настроек для интересов

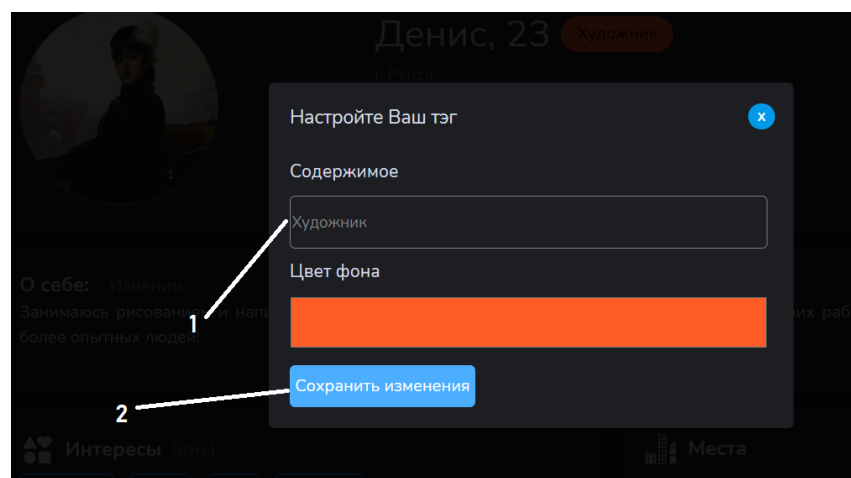


Рисунок А.5 – Форма настройки тега



Рисунок А.6 – Список постов пользователя

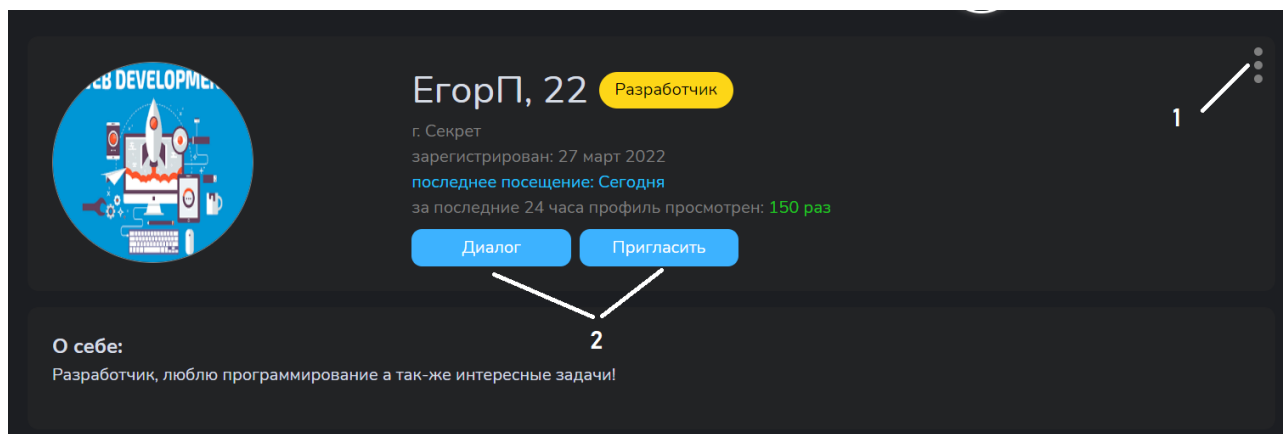


Рисунок А.7 – Страница профиля пользователя

Управление фильтрами контента

Выбрать язык

Темы

Черная Белая

Информация профиля

Имя

Егор П

Дата рождения

10.06.2002

Мобильный телефон

+79693463619

Сохранить

Управление аккаунтом

Рисунок А.8 – Страница настроек информации профиля

Дата рождения

10.06.2002

Мобильный телефон

+79693463619

Сохранить

Управление аккаунтом

Ваш Email

egorplatonov2002@gmail.com

Пароль

Адрес аккаунта

333

Сохранить

Вы можете изменить адрес аккаунта на более привлекательный.
Другие пользователи смогут найти Вас по адресу meetins.ru/ВашАдрес

Отключение

Рисунок А.9 – Страница настроек информации аккаунта

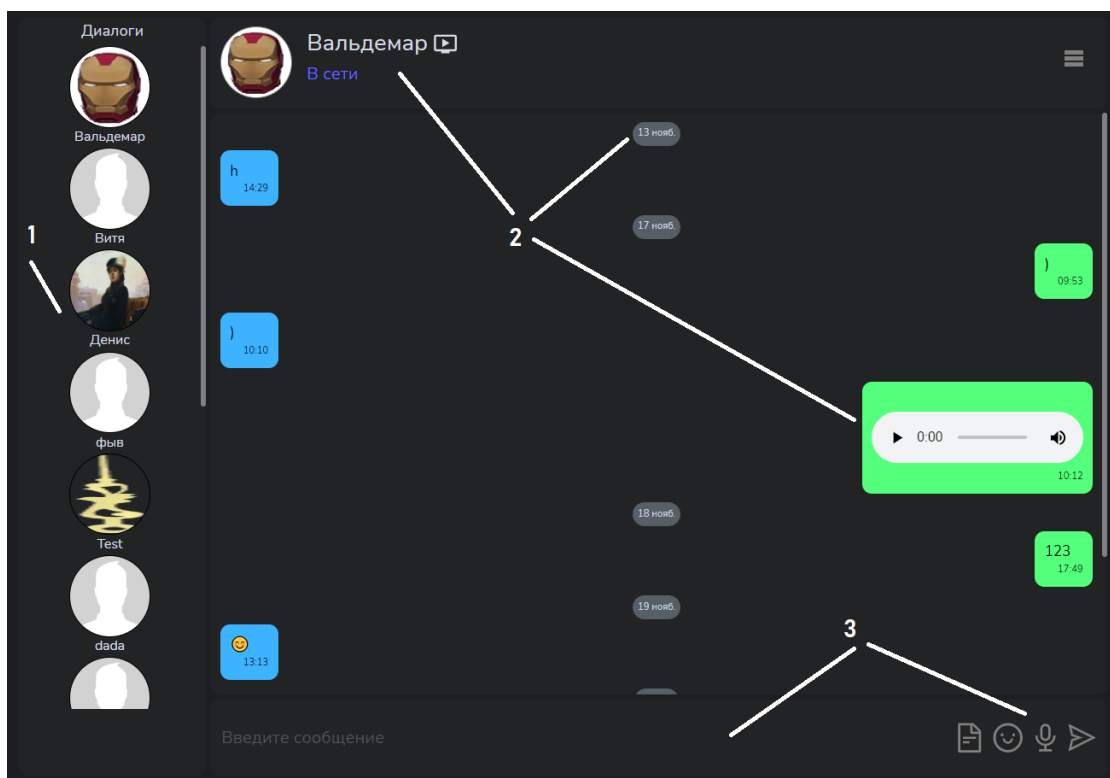


Рисунок А.10 – Страница мессенджера

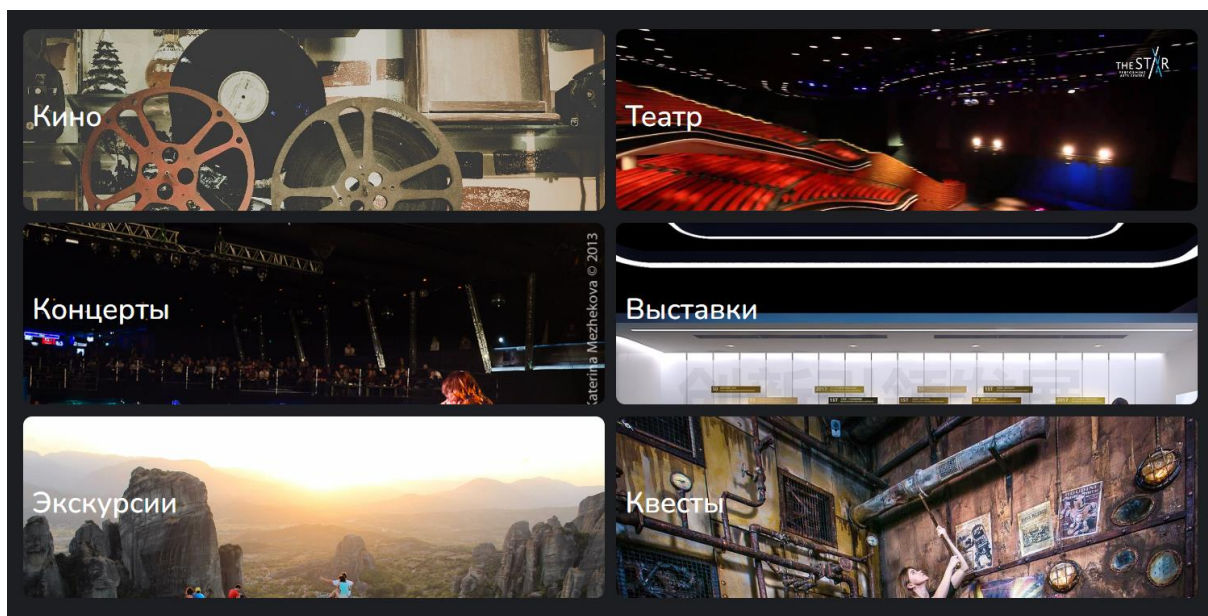


Рисунок А.11 – Страница категорий мероприятий

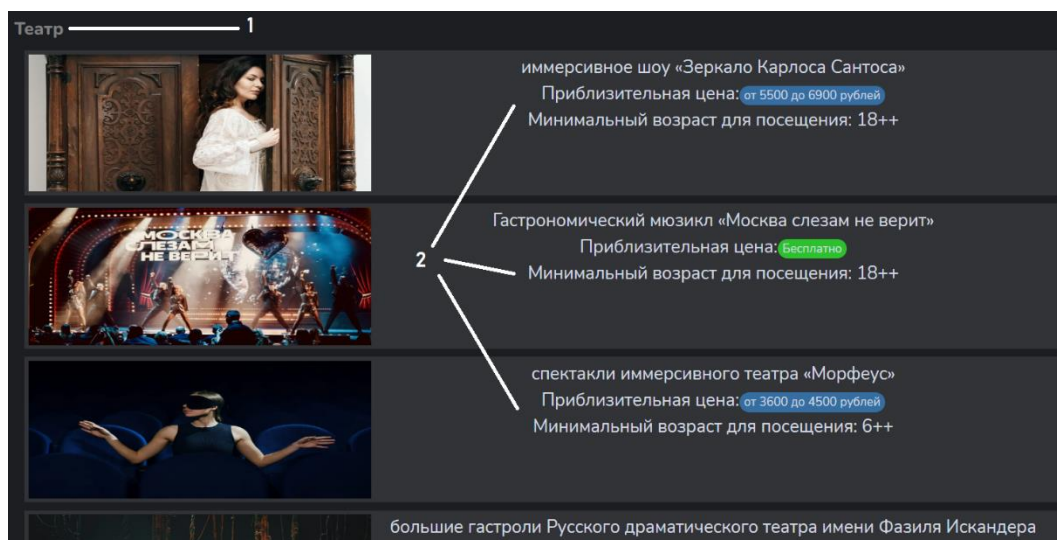


Рисунок А.12 – Страница со списком мероприятий в выбранной категории

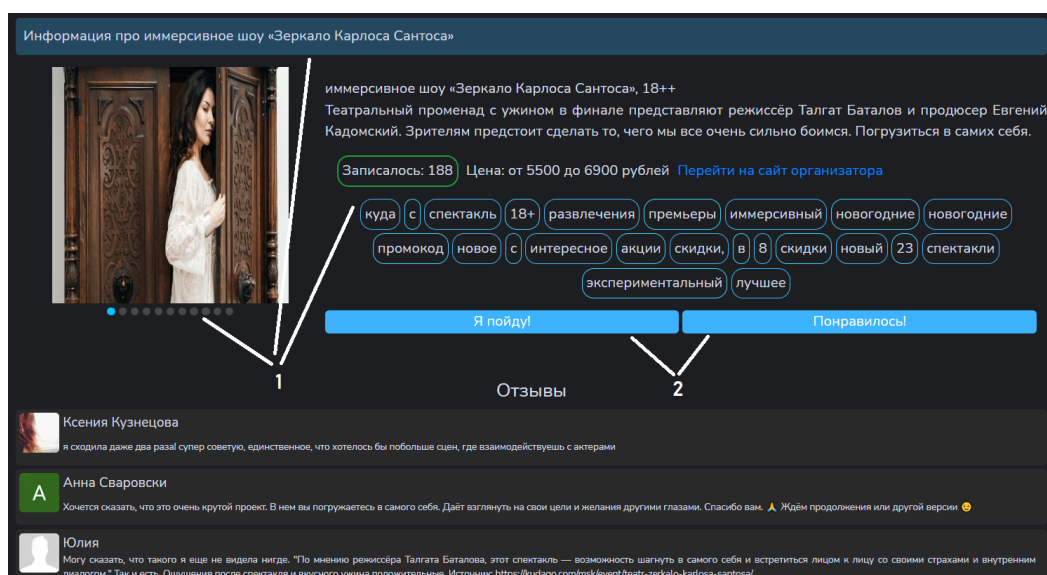


Рисунок А.13 – Страница просмотра информации о событии

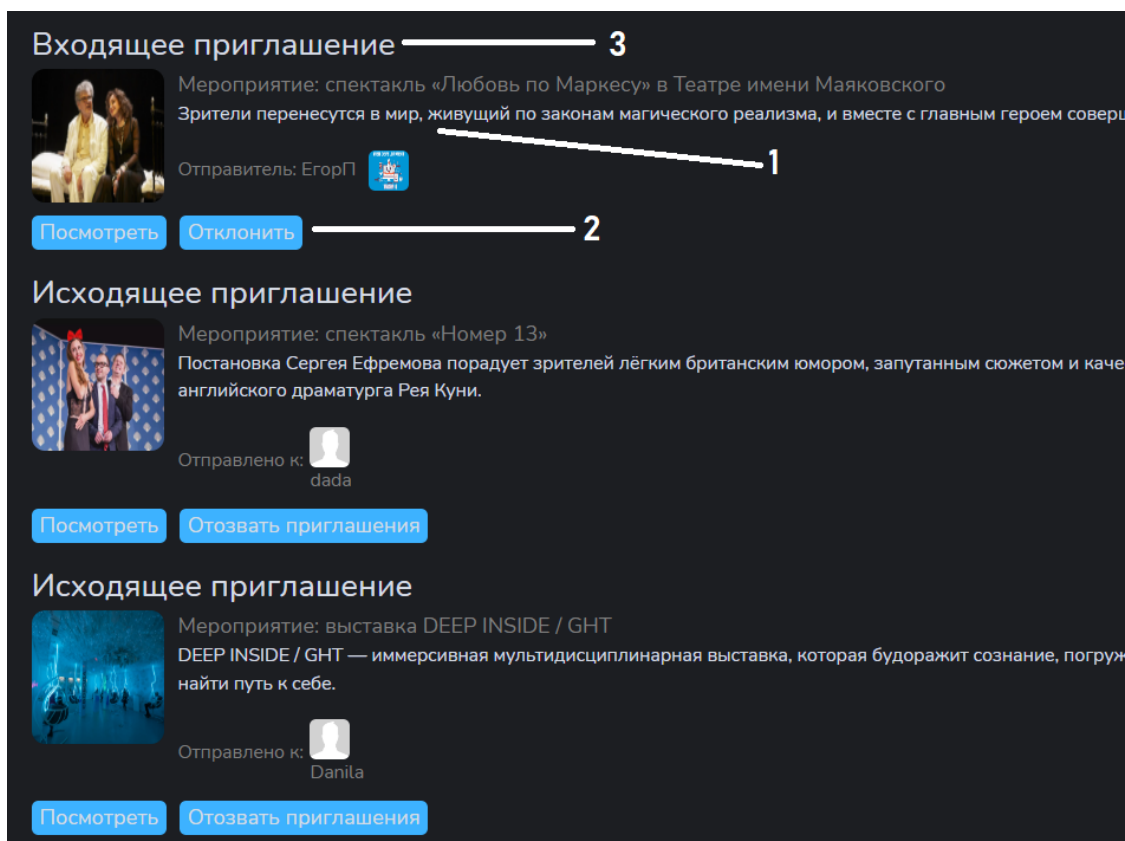


Рисунок А.14 – Страница регистрации

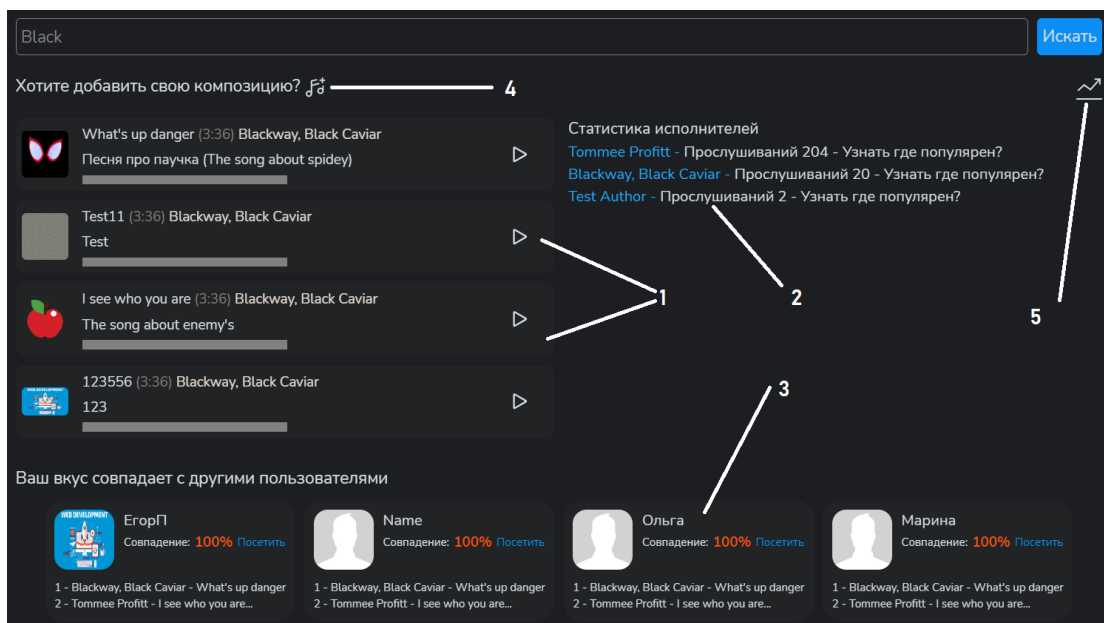


Рисунок А.15 – Страница музыки

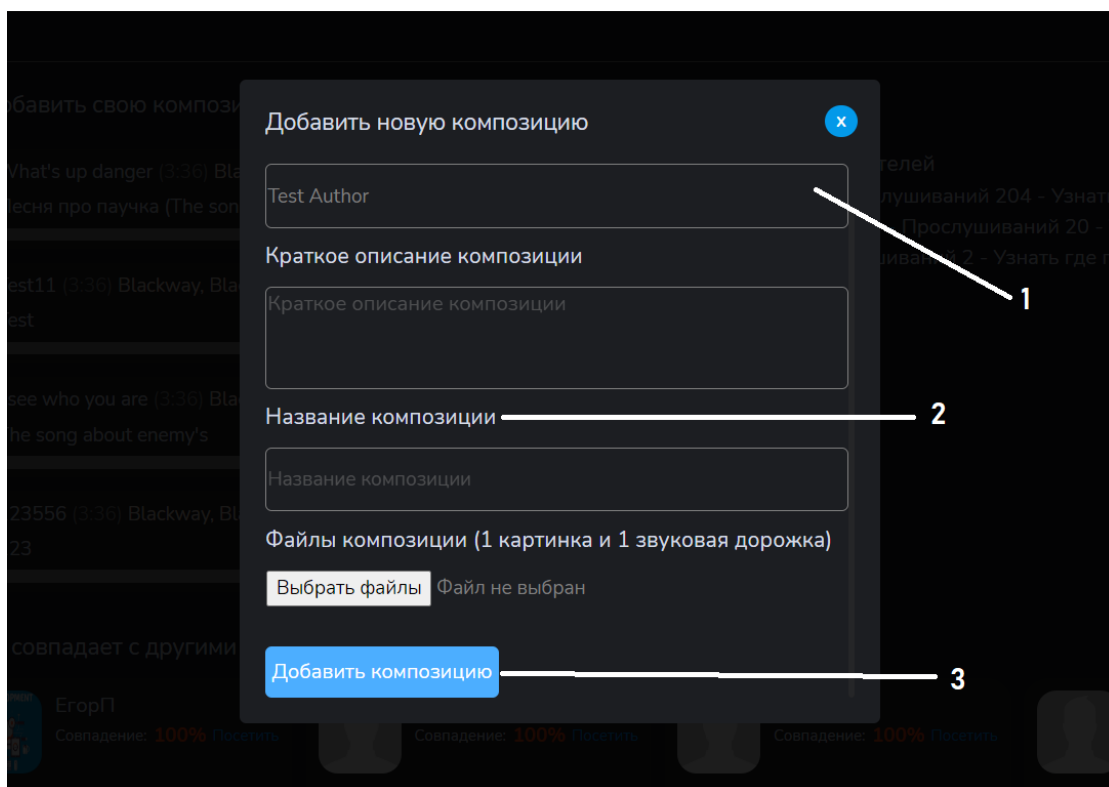


Рисунок А.16 – Форма добавления композиции

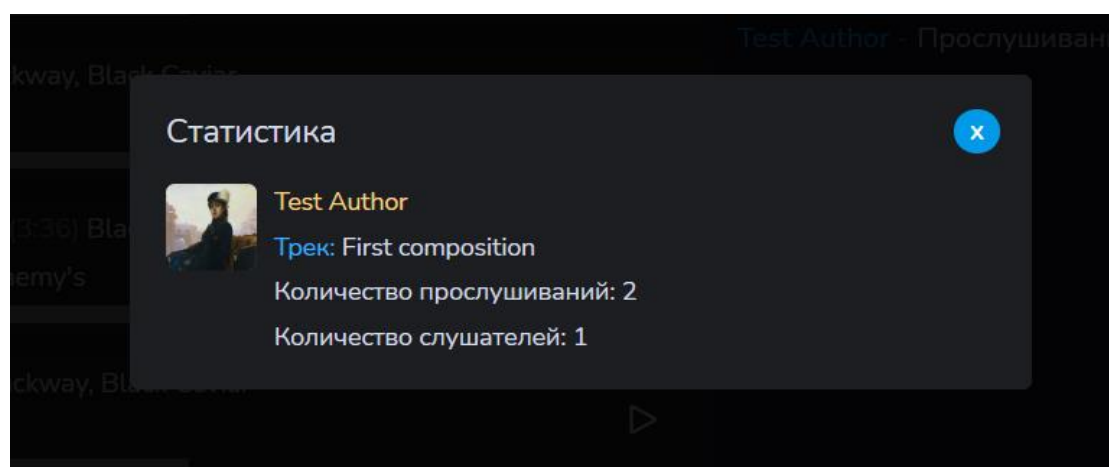


Рисунок А.17 – Форма просмотра статистики композиций

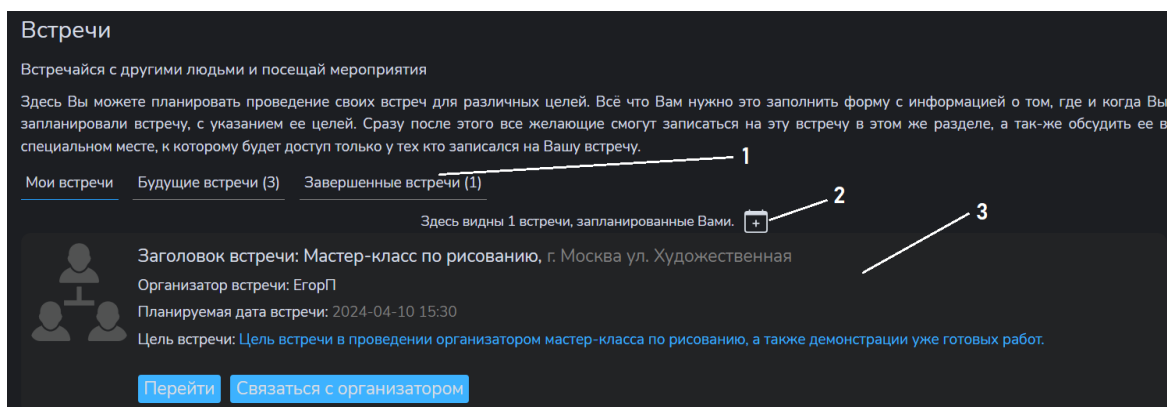


Рисунок А.18 – Страница просмотра встреч

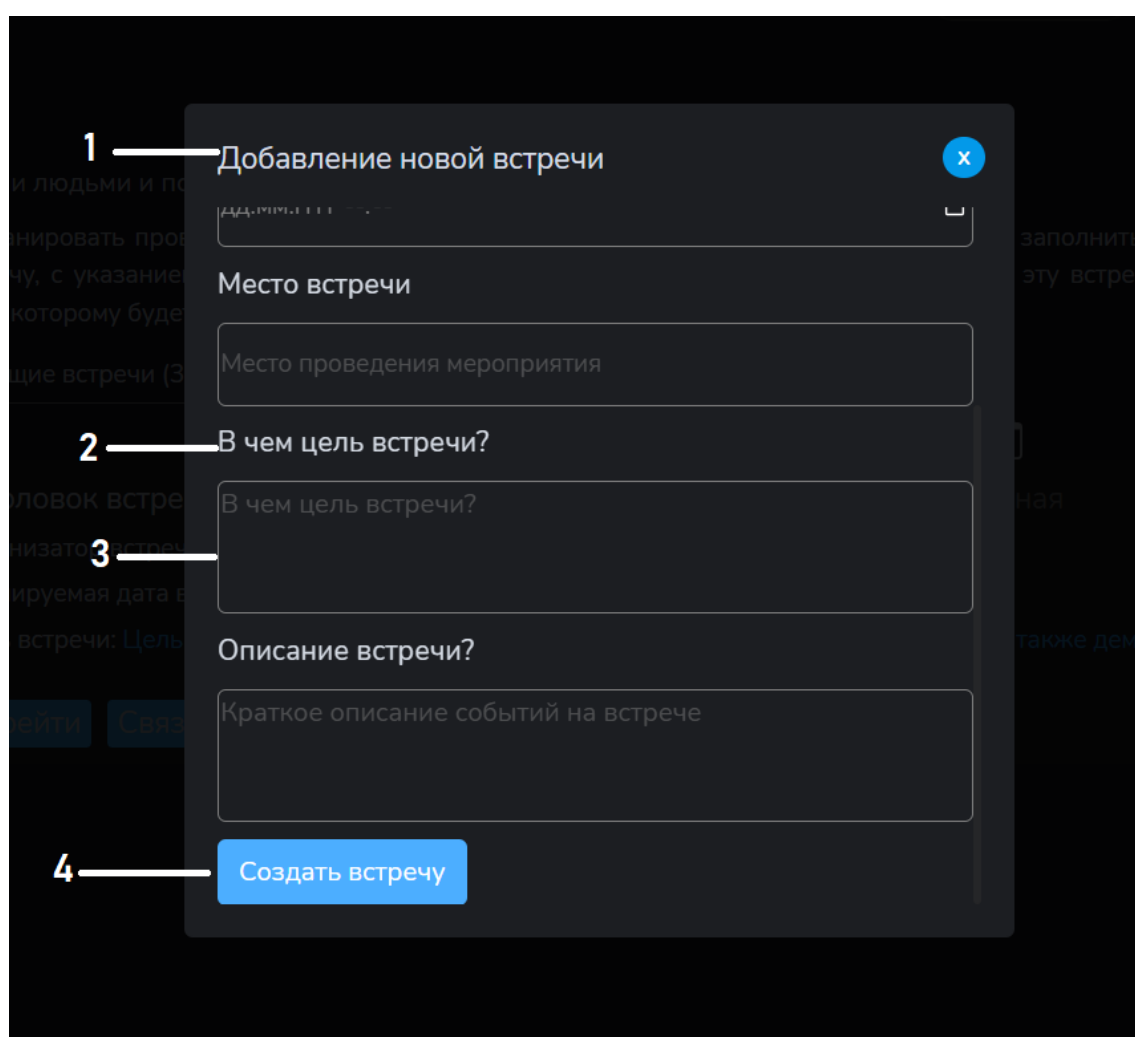


Рисунок А.19 – Форма добавления новой встречи



Рисунок А.20 – Страница просмотра информации о встрече

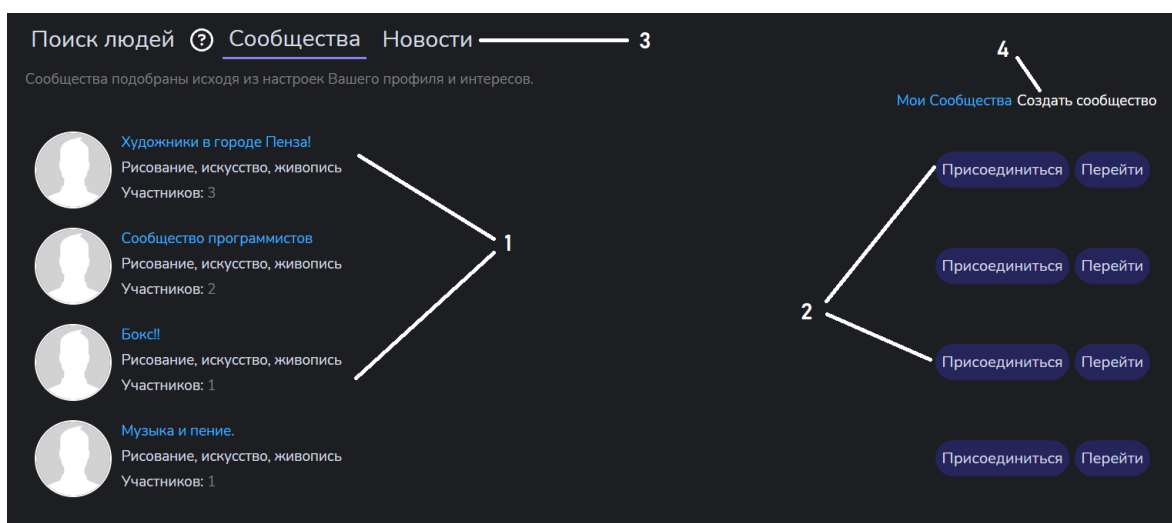


Рисунок А.21 – Страница просмотра списка сообществ

1 Создать новую группу

2 Название сообщества

Введите название сообщества

Описание для сообщества

Краткое описание Вашего сообщества

Выберите интересы сообщества

Бокс
Программирование
Шашки
Шахматы

3 Создать сообщество

Рисунок А.22 – Форма добавления сообщества

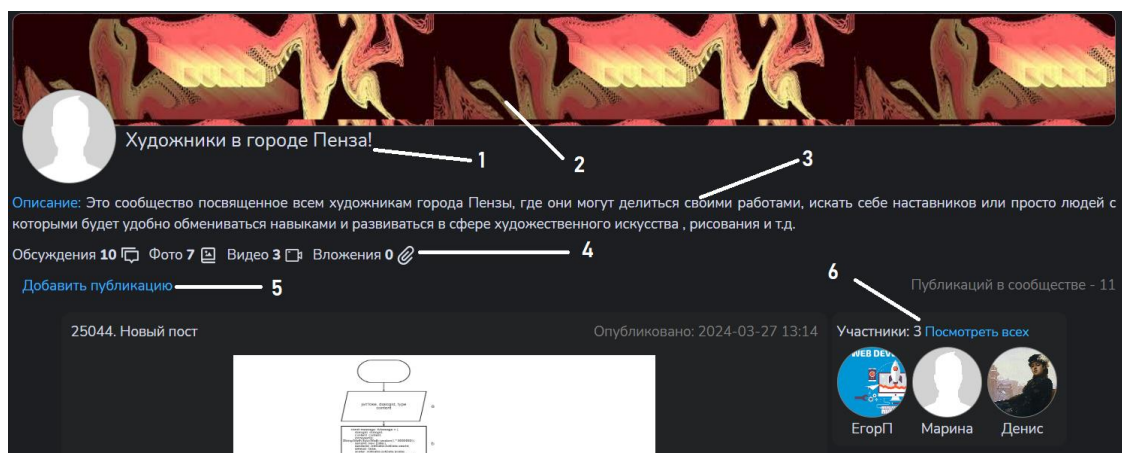


Рисунок А.23 – Страница просмотра информации о сообществе

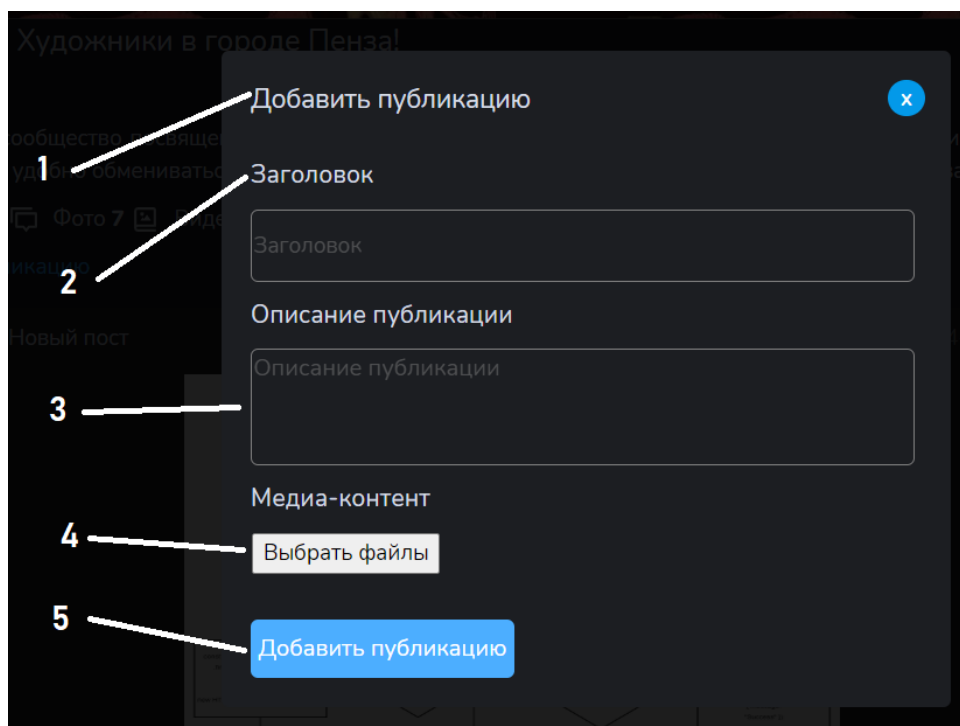


Рисунок А.24 – Форма добавления публикации в сообществе

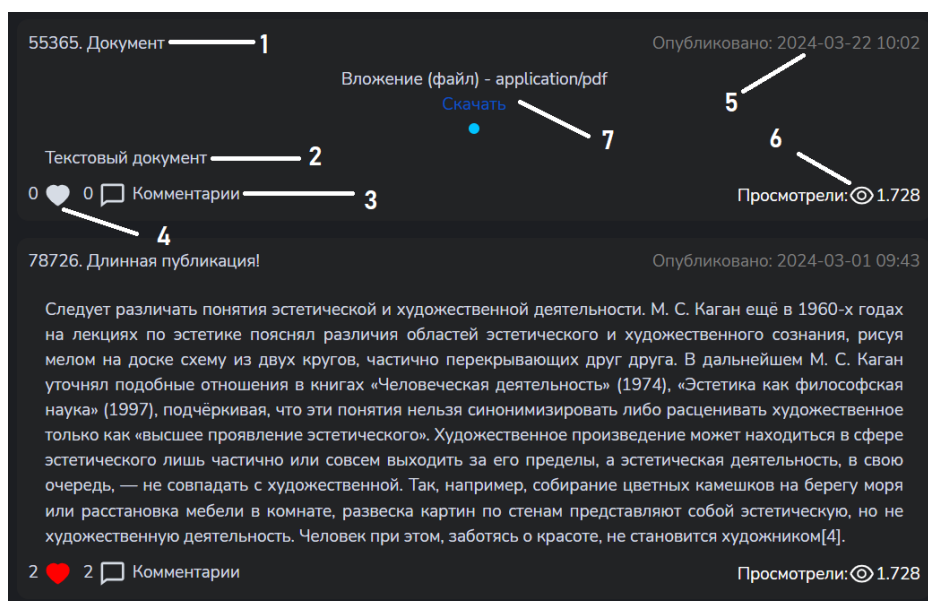


Рисунок А.25 – Страница просмотра списка публикаций сообщества

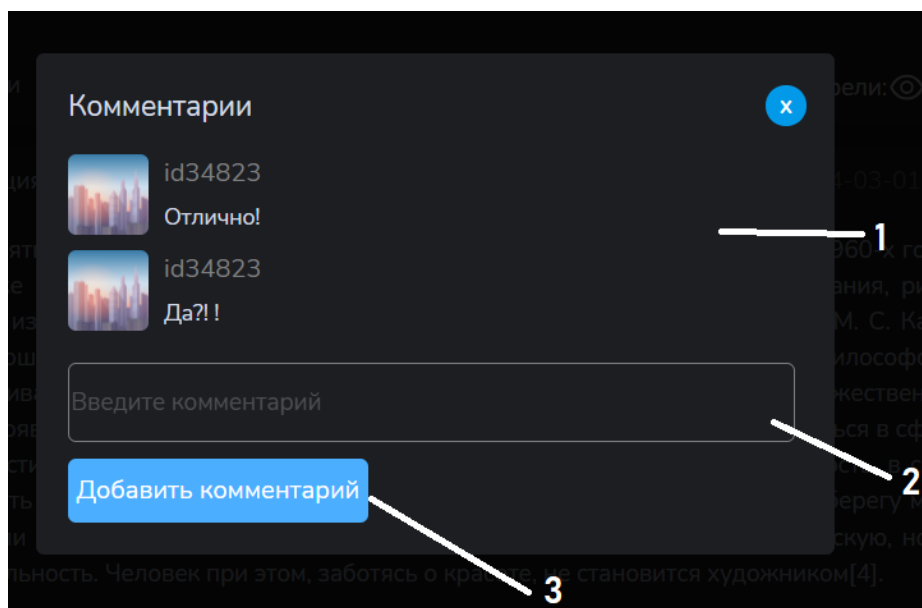


Рисунок А.26 – Форма добавления комментария

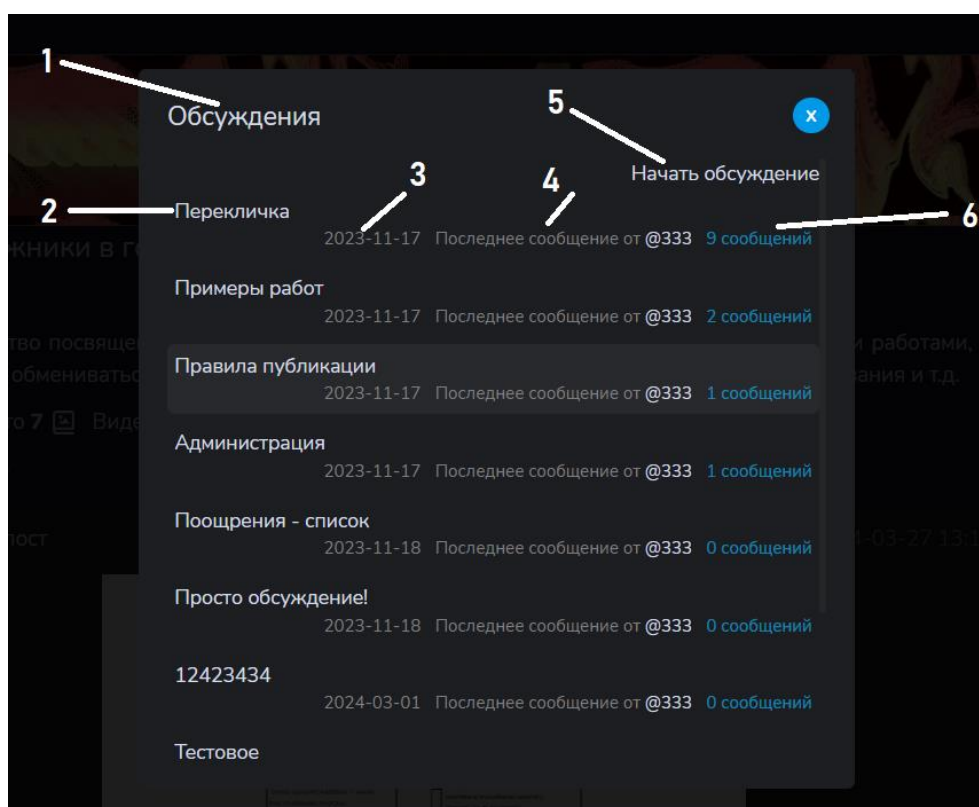


Рисунок А.27 – Форма просмотра списка обсуждений

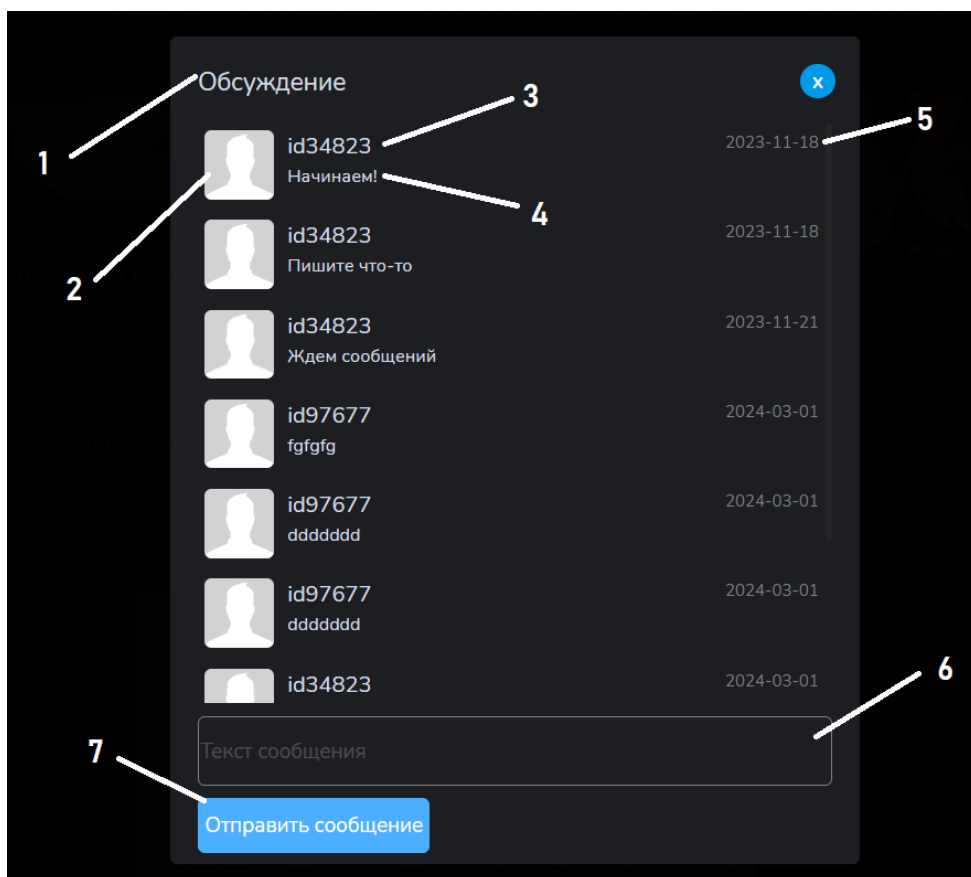


Рисунок А.28 – Форма для добавления комментария в обсуждение

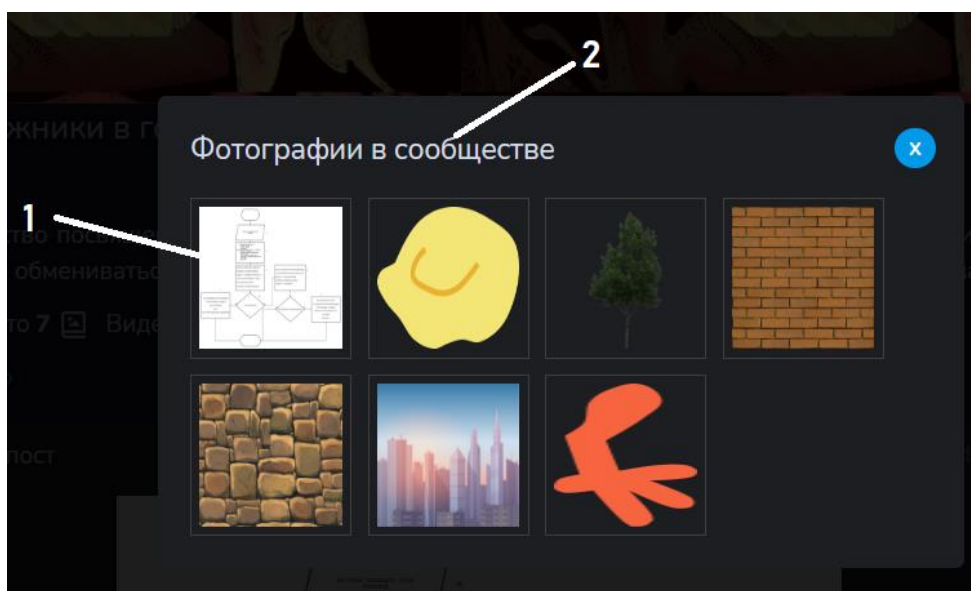


Рисунок А.29 – Форма просмотра списка картинок в сообществе

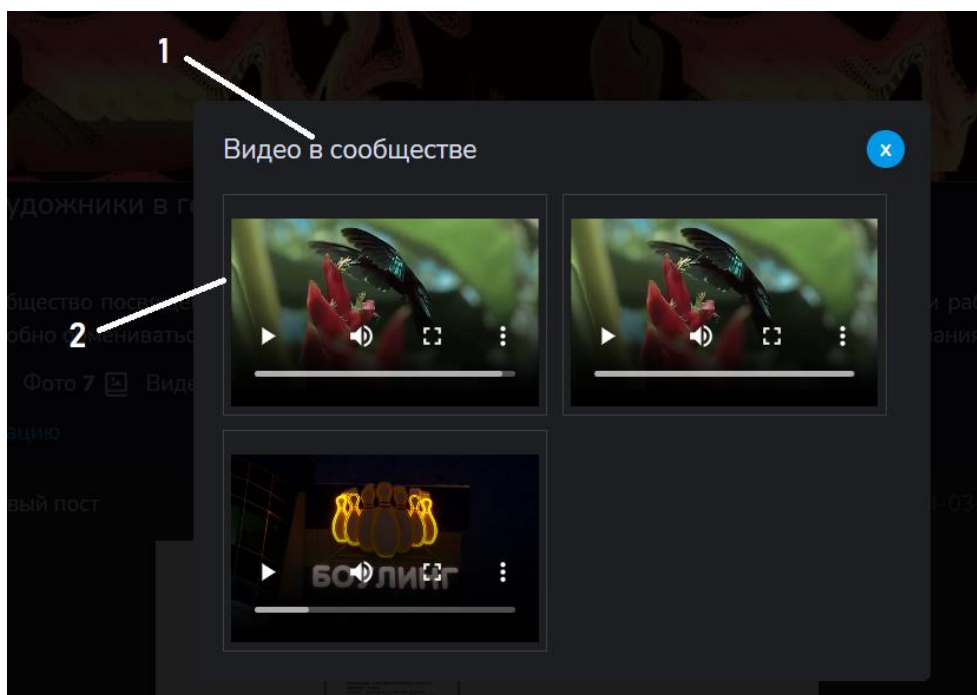


Рисунок А.30 – Форма просмотра списка видео в сообществе

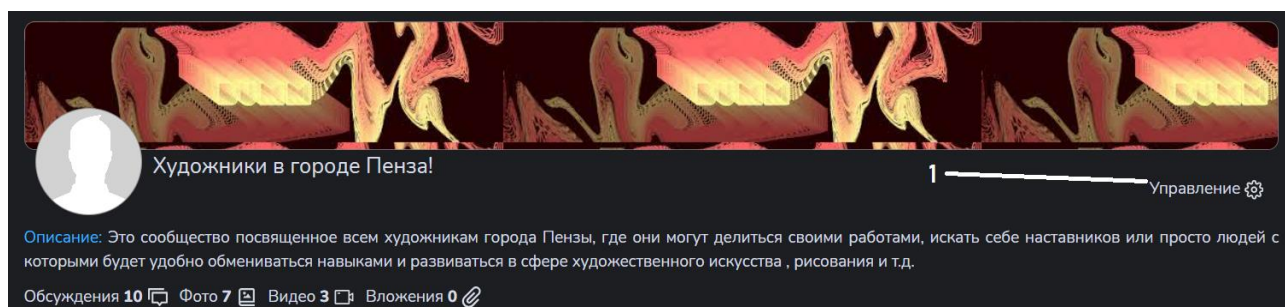


Рисунок А.31 – Страница просмотра информации о сообществе

Вложение (файл) - application/pdf

Управление сообществом

1 Название группы

2 Художники в городе Пенза!

3 Описание группы

4 Это сообщество посвященное всем художникам города Пензы, где они могут делиться своими работами, искать себе наставников или просто людей с которыми будет

5 Шапка сообщества

Выберите файл Файл не выбран

Сохранить настройки

Рисунок А.32 – Форма редактирование информации о сообществе

Давайте обсудим: на какую тему мне сделать следующую публикацию. Пишите идеи прямо здесь.

Понравилось: 1 Опубликовано: 2024-03-27 09:32

ЕгорП - Урок 1

Не получать уведомления
Перейти к источнику

нет	2 ↔ 6	2 ↔ 7	2 ↔ 9	2 ↔ 4
4	4	4	4	2
9	9	9	2	4
7	7	2	9	9
6	2	7	7	7
2	6	6	6	6
3	3	3	3	3

Нулевой проход, сравниваемые пары выделены

Урок 1. Тема: "Сортировка пузырьком в Python".

Понравилось: 0 Опубликовано: 2024-03-31 16:57

Денис - Первая работа.

Рисунок А.33 – Страница просмотра списка публикаций в новостной ленте

Приложение Б
(рекомендуемое)

Программный код методов серверной части приложения

```

    }

    async getGroupTalksList(request: Request) {
        const { groupId } = request.body;

        const findedTalks = await this.groupsModel.find({ groupId:
groupId }, {
            talks: true
        })

        if (findedTalks) {
            return findedTalks[0]?.talks;
        } else {
            throw new HttpException({ errorMessage: "Внутренняя ошибка
сервера." }, 500);
        }
    }

    async createNewMessageInGroupTalk(request: Request) {
        const { groupId, talkId, text } = request.body;
        const { userId } = this.jwtHelpService.decodeJwt(request);

        const message = { userId: userId, date: new Date(), text: text
    }

        const findedGroup = await this.groupsModel.updateOne(
            { "talks.id": talkId, groupId: groupId },
            {
                $push: {
                    "talks.$.messages": message
                }
            }
        );

        if (findedGroup) {
            return message;
        } else {
            throw new HttpException({ errorMessage: "Внутренняя ошибка
сервера." }, 500);
        }
    }

    async getTalkMessagesInGroupById(request: Request) {
        const { groupId, talkId } = request.body;

        const findedGroup = await this.groupsModel.findOne(
            { "talks.id": talkId, groupId: groupId },
        );

        if (findedGroup) {
            return findedGroup.talks.filter(talk => talk.id ===
talkId)[0]?.messages;

```

```

    } else {
        throw new HttpException({ errorMessage: "Внутренняя ошибка сервера." }, 500);
    }
}

async addNewCommentIntoPost(request: Request, groupId: number, postId: number) {
    const { userId } = this.jwtHelpService.decodeJwt(request);
    const inithiator = await this.usersModel.findOne({ userId: userId }, { avatar: true });
    const date = new Date();

    const newComment = {
        userId: userId,
        date: date,
        avatar: inithiator.avatar,
        text: request.body.text
    }

    const updatedGroup = await this.groupsModel.findOneAndUpdate({
        groupId: +groupId, "posts.id": +postId }, {
        $push: { "posts.$.comments": newComment }
    }, { returnDocument: 'after' })

    return updatedGroup;
}

async likePostInGroup(request: any, groupId: string, postId: string) {
    const decodedToken = this.jwtHelpService.decodeJwt(request);
    const updatedGroup = this.groupsModel.findOneAndUpdate({
        groupId: +groupId, "posts.id": +postId }, {
        $addToSet: { "posts.$.likes": decodedToken.userId }
    }, { returnDocument: "after" });

    if (updatedGroup) return updatedGroup;
    throw new HttpException({ message: "Попробуйте снова, произошла неизвестная ошибка"}, 400);
}

async unlikePostInGroup(request: any, groupId: string, postId: string) {
    const decodedToken = this.jwtHelpService.decodeJwt(request);
    const updatedUser = this.groupsModel.findOneAndUpdate({ groupId: +groupId, "posts.id": +postId }, {
        $pull: { "posts.$.likes": decodedToken.userId }
    }, { returnDocument: "after" });

    if (updatedUser) return updatedUser;
}

```

```

    throw new HttpException({ message: "Попробуйте снова, произошла
неизвестная ошибка"}, 400);
  }
}
@Inject()
export class InterestsService {

  constructor(
    private jwtService: JwtService,
    private userService: UserService,
    private socketServer: AppGateway,
    @InjectModel(Interest.name) private interestsModel:
Model<InterestDocument>
  ) {}

  async getInterests(request: Request) {
    const allInterests = await this.interestsModel.find();
    if (allInterests) {
      return allInterests;
    } else {
      throw new HttpException('Ничего не найдено.', 400);
    }
  }

  async getInterestsById(request: Request) {
    let findedInterests = [];
    const { interests } = request.body;

    if (interests) {
      await Promise.all(interests.map(async (el) => {
        const eachInterest = await
this.interestsModel.findOne({ interestId: el});
        if (eachInterest) {
          findedInterests = [...findedInterests,
eachInterest];
        }
      }));
    }

    if (findedInterests) {
      return findedInterests;
    }
  }

  async addInterests(request: Request) {
    const { title } = request.body;
    const newId = Math.floor(Math.random()*500000);

    const newInterest = {
      title: title,
      interestId: newId
    };
  }
}

```

```

                                const      interest      =      await
this.interestsModel.create(newInterest);
    if(interest) {
        return interest;
    }
}
}
@Inject()
export class MeetingsService {

    constructor(
        private jwtHelpService: HelpJwtService,
        @InjectModel(Meeting.name) private meetingModel:
Model<MeetingDocument>,
        @InjectModel(User.name) private userModel: Model<UserDocument>
    ) {}

    async getAllMeetings() {
        const meetings = await this.meetingModel.find();
        const fullMeetingsInfo = await Promise.all(await
meetings.map(async (meeting) => {
            const participantsFullInfo = await this.userModel.find({
userId: { $in: meeting.participants }}, {
            login: true, _id: false, avatar: true, name: true, userId:
true
            });
        return {
            meetingId: meeting.meetingId,
            date: meeting.date,
            description: meeting.description,
            goal: meeting.goal,
            maxParticipants: meeting.maxParticipants,
            comments: meeting.comments,
            preview: meeting.preview,
            title: meeting.title,
            address: meeting.address,
            files: meeting.files,
            participants: participantsFullInfo,
            creatorId: meeting.creatorId
        }
    })).then(res => {
        return res
    })
    return fullMeetingsInfo;
}

    async createNewMeeting(request: Request) {
        const { date, description, goal, title, address } = request.body;
        const user = this.jwtHelpService.decodeJwt(request);

        const newMeeting = {

```

```

        participants: [user.userId],
        date,
        description,
        goal,
        title,
        address,
        meetingId: `meeting${Math.floor(Math.random()*1000000)}\`,
        creatorId: user.userId
    };

    const createdMeeting = await
this.meetingModel.create(newMeeting);
    if (createdMeeting) {
        return createdMeeting;
    } else {
        throw new HttpException({ errorMessage: "Пожалуйста попробуйте
снова" }, 500);
    }
}

async uploadMedia(request: Request, file: Express.Multer.File) {
    const { meetingId } = request.body;

    const updatedMeeting = await this.meetingModel.findOneAndUpdate(
        { meetingId: meetingId },
        { $push: { files: { src: file.filename, type: file.mimetype
}} },
        { returnDocument: 'after' }
    );

    const participantsFullInfo = await this.userModel.find({ userId:
{ $in: updatedMeeting.participants }}, {
        login: true, _id: false, avatar: true, name: true, userId:
true
    });
    return {
        meetingId: updatedMeeting.meetingId,
        date: updatedMeeting.date,
        description: updatedMeeting.description,
        goal: updatedMeeting.goal,
        maxParticipants: updatedMeeting.maxParticipants,
        comments: updatedMeeting.comments,
        preview: updatedMeeting.preview,
        title: updatedMeeting.title,
        address: updatedMeeting.address,
        files: updatedMeeting.files,
        participants: participantsFullInfo
    }
}

async regInMeetings(request: Request, meetingId: string) {
    try {
        const { userId } = this.jwtHelpService.decodeJwt(request);

```

```

                                const      updatedMeeting      =      await
this.meetingModel.findOneAndUpdate(
    { meetingId: meetingId, $expr: { $lt: [{ $size:
"$participants" }, "$maxParticipants"] } },
    { $push: { participants: userId } },
    { returnDocument: 'after' }
);
    const participantsFullInfo = await this.userModel.find({
userId: { $in: updatedMeeting.participants }}, {
    login: true, _id: false, avatar: true, name: true, userId:
true
});
    return {
        meetingId: updatedMeeting.meetingId,
        date: updatedMeeting.date,
        description: updatedMeeting.description,
        goal: updatedMeeting.goal,
        maxParticipants: updatedMeeting.maxParticipants,
        comments: updatedMeeting.comments,
        preview: updatedMeeting.preview,
        title: updatedMeeting.title,
        address: updatedMeeting.address,
        files: updatedMeeting.files,
        participants: participantsFullInfo
    }
} catch(error) {
    throw new HttpException({ errorMessage: "Что-то пошло не так,
попробуйте снова" }, 400);
}
}

async sendCommentAboutMeeting(request: Request) {
    const currentDate = new Date();
    const { userId } = this.jwtHelpService.decodeJwt(request);

    try {
        const newComment: MeetingComment = {
            userId: userId,
            text: request.body.text,
            date: currentDate,
            meetingId: request.body.meetingId
        }

                                const      updatedMeeting      =      await
this.meetingModel.findOneAndUpdate({
newComment.meetingId }, {
                                meetingId:
        $push: { comments: newComment}
        }, {returnDocument: 'after'}));

        return updatedMeeting;
    } catch(error) {

```

```

        throw new HttpException({ errorMessage: "Что-то пошло не так,
попробуйте снова" }, 400);
    }
}

async getUserPlaces(request: Request, userId: string) {
    const prevMeetings = await this.meetingModel.find({
        date: { $lt: new Date() },
        participants: { $elemMatch: { $eq: userId } }
    });
    const places = prevMeetings.map(el => el.address);
    return places;
}
}
@Injectable()
export class MusicService {

    constructor(
        private jwtHelpService: HelpJwtService,
        @InjectModel(MusicAuthor.name) private musicAuthorModel:
Model<MusicAuthorDocument>,
        @InjectModel(User.name) private userModel: Model<UserDocument>
    ) {}

    async getAllMusic() {
        const authors = await this.musicAuthorModel.find({});
        return authors;
    };

    async getAuthorsStatistics() {
        const authors = await this.musicAuthorModel.aggregate([
            {
                $unwind: "$compositions"
            },
            {
                $group: {
                    _id: "$authorId",
                    playsNumber: { $sum: "$compositions.playsNumber" },
                    name: { $first: "$name" }
                }
            },
            {
                $project: {
                    _id: 0,
                    id: "$_id",
                    playsNumber: 1,
                    name: 1
                }
            }
        ])
        return authors;
    };
}

```

0

```

async getAuthorStatistics(request: Request) {
  const { userId } = this.jwtHelpService.decodeJwt(request);
  const statistic = await this.musicAuthorModel.aggregate([
    { $match: { authorId: userId } },
    {
      $unwind: "$compositions"
    },
    {
      $project: {
        _id: false,
        author: "$name",
        title: "$compositions.title",
        playsNumber: "$compositions.playsNumber",
        image: "$compositions.imageSrc"
      }
    }
  ]);
  if (statistic) {
    return statistic;
  } else {
    throw new HttpException({ errorMessage: "Вы еще не публиковали
музыку." }, 400 );
  }
}

async addMusic(files: any[], request: Request) {
  const { userId } = this.jwtHelpService.decodeJwt(request);
  const findedAuthor = await this.musicAuthorModel.findOne({
authorId: userId });
  let imageSrc = "";
  let audioSrc = "";

  files.map(file => {
    if (file.mimetype.includes('image')) imageSrc = file.filename;
    if (file.mimetype.includes('audio')) audioSrc = file.filename;
  });

  const newComposition = {
    audioSrc: audioSrc,
    imageSrc: imageSrc,
    title: request.body.title,
    description: request.body.description,
    playsNumber: 0,
    id: Math.floor(Math.random()*1000000)
  };

  if (findedAuthor) {
    await this.musicAuthorModel.updateOne({ authorId: userId }, {
      $set: {

```

1

```

        compositions:    [...findedAuthor.compositions,
newComposition]
    }
    });
} else {
    await this.musicAuthorModel.create({
        authorId: userId,
        name: request.body.name,
        compositions: [newComposition]
    });
}
}

async addPlaysNumber(request: Request, authorId: string, trackId:
string) {
    const data = await this.musicAuthorModel.findOneAndUpdate(
        { authorId: authorId, "compositions.id": +trackId },
        {
            $inc: { "compositions.$.playsNumber": 1 }
        },
        {returnDocument: "after"}
    );
    return data;
}

async getMatchesList(request: Request) {
    const { userId } = this.jwtHelpService.decodeJwt(request);

    const data = await this.userModel.aggregate([
        { $limit: 4 },
        {
            $project: {
                name: 1,
                login: 1,
                avatar: 1
            }
        }
    ]);
    return data;
}

async getAuthorCurrentName(request: Request) {
    const { userId } = this.jwtHelpService.decodeJwt(request);
    const authorData = await this.musicAuthorModel.findOne({
authorId: userId }, { name: 1 });

    if (authorData) {
        return authorData.name;
    } else {
        return "";
    }
}

```

2

```

    }

}
@Inject()
export class UserService {

    constructor(
        @InjectModel(User.name) private userModel:
Model<UserDocument>,
        private helpJwtService: HelpJwtService
    ) {}

    async updateUsersData() {
        await this.userModel.updateMany({}, { $set: { tag: { color:
"rgba(42, 132, 251, 0.878)", title: "Гость" } } });
        await this.userModel.updateMany({}, { $set: {
purchasedOpportunities: [] } });
    }
    async getUsers() {
        const users = await this.userModel.find({}, {
            password: false,
            _id: false,
            __v: false
        });
        return users;
    }

    async addUser(dto: CreateUserDto) {
        let candidate = {
            ...dto,
            login: Math.floor(Math.random()*20000),
            userId: "id" + String(Math.floor(Math.random()*100000))
        }
        const user = await this.userModel.create(candidate);
        if(user) {
            return user;
        }
    }

    async getUserByLogin(login: string) {
        const user = await this.userModel.findOne({login: login}, {
            password: false,
            _id: false,
            __v: false
        })
        return user;
    }

    async getUserByLoginMainInfoOnly(login: string) {
        const user = await this.userModel.findOne({login: login}, {
            password: false,
            _id: false,

```

```

        __v: false,
        innerInvites: false,
        outerInvites: false,
        isFilter: false,
        markedUsers: false,
        purchasedOpportunities: false
    })
    return user;
}

async getUserList() {
    const user = await this.userModel.find({}, {
        password: false,
        __id: false,
        __v: false
    })
    const peoples: IPeople[] = user.map(user => {
        return {
            email: user.email,
            login: user.login,
            userName: user.name,
            userAvatar: user.avatar,
            status: user.status,
            age: user.age,
            city: user.city,
            gender: user.gender
        }
    })
    return peoples;
}

async addUserIntoMarkedList(request: Request) {
    const { userId } = this.helpJwtService.decodeJwt(request);
    const { neededUserId } = request.body;

    const updatedUser = this.userModel.findOneAndUpdate({
        userId: userId }, { $push: {
            markedUsers: neededUserId
        } }, { returnDocument: 'after' });

    if (updatedUser) return updatedUser;
    throw new HttpException({ errorMessage: "Что-то пошло не
так."}, 400);
}

async formMarkedUsersInfoByIds(userId: string) {
    const usersInfo = await this.userModel.aggregate([
        {
            $match: { userId: userId }
        },
        {

```

4

```

        $unwind: "$markedUsers"
      },
      {
        $lookup: {
          from: "users",
          localField: "markedUsers",
          foreignField: "userId",
          as: "markedUserInfo"
        }
      },
      {
        $unwind: "$markedUserInfo"
      },
      {
        $project: {
          _id: false,
          "markedUserInfo.name": true,
          "markedUserInfo.avatar": true,
          "markedUserInfo.login": true,
          "markedUserInfo.userId": true
        }
      }
    ]));
    if (usersInfo) return usersInfo.map(el => { return
    el.markedUserInfo });
  }

  async updateUserTag(request: Request) {
    const { userId } = this.helpJwtService.decodeJwt(request);
    const { title, color } = request.body;

    const updatedUser = await this.userModel.findOneAndUpdate({
    userId: userId }, {
      $set: { tag: { title, color } }
    }, { returnDocument: 'after' });

    if (updatedUser) return updatedUser;
    throw new HttpException({ errorMessage: "Что-то пошло не
    так."}, 400);
  }

  async getMarkedUsersInfo(request: Request) {
    const { userId } = this.helpJwtService.decodeJwt(request);

    const markedUsersInfo = await
    this.formMarkedUsersInfoByIds(userId);
    if (markedUsersInfo) {
      return markedUsersInfo;
    }
  }

```

5

```

        throw new HttpException({ errorMessage: "Что-то пошло не
так."}, 400);
    }

    async removeUserFromMarkedList(request: Request, removedUserId:
string) {
        const { userId } = this.helpJwtService.decodeJwt(request);

        await this.userModel.updateOne({ userId: userId }, { $pull:
{
            markedUsers: removedUserId
        }});

        const markedUsersInfo = await
this.formMarkedUsersInfoByIds(userId);
        if (markedUsersInfo) {
            return markedUsersInfo;
        }
        throw new HttpException({ errorMessage: "Что-то пошло не
так."}, 400);
    }

    async getUserListByPageNumber(request: Request) {
        const { userId } = this.helpJwtService.decodeJwt(request);
        const inithiator = await this.userModel.findOne({ userId:
userId });
        const { pageNumber, pageSize, filters } = request.body;

        let users = [];
        if (inithiator.isFilter) {
            users = await this.userModel.find({ interests: {
$elemMatch: { $in: inithiator.interests } } }, {
                password: false,
                _id: false,
                __v: false
            });
        } else {
            if (filters.event) {
                users = await this.userModel.find({
                    events: { $elemMatch: { $eq: String(filters.event)
} }
                }, {
                    password: false,
                    _id: false,
                    __v: false
                });
            } else {
                users = await this.userModel.find({}, {
                    password: false,
                    _id: false,
                    __v: false
                });
            }
        }
    }

```

```

        }
    }
    let peoples: IPeople[] = users.map((user, index) => {
        return {
            email: user.email,
            login: user.login,
            userName: user.name,
            userAvatar: user.avatar,
            status: user.status,
            age: user.age,
            city: user.city,
            gender: user.gender
        }
    });
    if (filters.age !== 0) {
        peoples = peoples.filter(peoples => peoples.age ===
filters.age);
    }
    if (filters.gender !== 'all') {
        peoples = peoples.filter(peoples => peoples.gender ===
filters.gender)
    }
    const maxPage = Math.ceil(peoples.length / pageSize);
    peoples = peoples.slice(pageNumber * pageSize - pageSize,
pageNumber * pageSize);

    return { peoples: peoples, maxPage: maxPage };
}
async getSortedPeoples(sortParams: ISortParams) {
    let peoples = await this.getUserList();
    if(sortParams.age !== 50) {
        peoples = peoples.filter((user) => user.age ==
sortParams.age);
    }
    if(sortParams.gender !== "") {
        peoples = peoples.filter((user) => user.gender ==
sortParams.gender);
    }
    throw new HttpException(peoples, 200);
}
async getUpdatedUserByEmail(email: string) {
    const user = await this.userModel.findOne({email: email}, {
        password: false,
        _id: false,
        __v: false
    });
    return user;
}
async getUserByEmail(email: string) {
    const user = await this.userModel.findOne({email: email}, {
        _id: false,

```

7

```

        __v: false
    }); __v: false
    return user;
}
async getUserById(userId: string) {
    const user = await this.userModel.findOne({userId: userId},
{
    _id: false,
    __v: false
    });
    return user;
}
async updateUserStatus(decodedToken: any, status: string) {
    await this.userModel.updateOne({email : decodedToken.email},
{$set: {status : status}});

    const updatedUser: User = await
this.getUpdatedUserByEmail(decodedToken.email);
    if(updatedUser) {
        return updatedUser;
    }
}
async updateUserFilterStatus(decodedToken: any, filterStatus:
string) {
    const updatedUser = await this.userModel.findOneAndUpdate(
        {userId : decodedToken.userId}, {$set: {isFilter :
filterStatus}}, { returnDocument: 'after' }
    );
    return updatedUser;
}
async updateUserAccount(decodedToken: any, accountData:
IAccount) {
    const users = await this.userModel.find({
        $or: [
            { email: accountData.email },
            { login: accountData.login }
        ]
    });

    if (users.length === 0) {
        await this.userModel.updateOne({email :
decodedToken.email}, {$set: {
            email : accountData.email,
            password: accountData.password,
            login: accountData.login
        }});

        const updatedUser: User = await
this.getUpdatedUserByEmail(decodedToken.email);

```

```

        if(updatedUser) {
            return updatedUser;
        }
    } else {
        throw new HttpException({ message: "Занят login или
email"}, 400);
    }
}

async updateUserProfile(decodedToken: any, accountData:
IProfile) {

    const users = await this.userModel.find({ email:
accountData.phoneNumber });

    if (users.length === 0) {
        await this.userModel.updateOne({email :
decodedToken.email}, {$set: {
            phoneNumber : accountData.phoneNumber,
            name: accountData.name,
            birthDate: accountData.birthDate
        }});

        await this.updateUserBirthDate(decodedToken.email, new
Date(accountData.birthDate));

        const updatedUser: User = await
this.getUpdatedUserByEmail(decodedToken.email);
        if(updatedUser) {
            return updatedUser;
        }
    } else {
        throw new HttpException({ message: "Занят login или
email"}, 400);
    }
}

async updateUserAvatar(file: any, user: User) {

    const updatedUser: User = await
this.userModel.findOneAndUpdate({email : user.email}, {$set: {
        avatar : file.filename,
    }}, { returnDocument: "after" });

    if(updatedUser) {
        return updatedUser;
    }
}

async updateUserBirthDate(userEmail: string, date: Date) {
    const actualYear = new Date().getFullYear();
    const userBirthDateYear = date.getFullYear();

```

9

```

        await this.userModel.updateOne({email : userEmail}, {$set:
{
    age: actualYear - userBirthDateYear,
    }}});
    }

    async addUserEvent(request: Request) {
        const { body } = request;
        const decodedToken = this.helpJwtService.decodeJwt(request);
        const prevUserState = await this.userModel.findOne({email :
decodedToken.email});

        if (!prevUserState.events.includes(body.eventId)) {
            await this.userModel.updateOne({email :
decodedToken.email}, {$set: {
                events: [...prevUserState.events, body.eventId],
            }});
            const updatedUser: User = await
this.getUpdatedUserByEmail(decodedToken.email);
            if (updatedUser) {
                return updatedUser;
            }
        } else {
            throw new HttpException({ message: "Данное событие уже
добавлено" }, 400);
        }
    }

    async deleteUserEvent(request: Request) {
        const { body } = request;
        const decodedToken = this.helpJwtService.decodeJwt(request);
        const prevUserState = await this.userModel.findOne({email :
decodedToken.email});

        await this.userModel.updateOne({email : decodedToken.email},
{$set: {
            events: prevUserState.events.filter(el => el !==
body.eventId),
        }});
        const updatedUser: User = await
this.getUpdatedUserByEmail(decodedToken.email);
        if (updatedUser) {
            return updatedUser;
        }
    }

    async updateUserInterest(request: Request) {
        const { body } = request;
        const decodedToken = this.helpJwtService.decodeJwt(request);
        const prevUserState = await this.userModel.findOne({email :
decodedToken.email});

```

0

```

        await this.userModel.updateOne({email : decodedToken.email},
{$set: {
            interests: body.interests,
        }});
        const updatedUser: User = await
this.getUpdatedUserByEmail(decodedToken.email);
        if (updatedUser) {
            return updatedUser;
        }
    }
    async removeUserInterest(request: Request) {
        const { body } = request;
        const decodedToken = this.helpJwtService.decodeJwt(request);
        const prevUserState = await this.userModel.findOne({email :
decodedToken.email});

        await this.userModel.updateOne({email : decodedToken.email},
{$set: {
            interests: prevUserState?.interests.filter(el => el !==
body.interestId),
        }});
        const updatedUser: User = await
this.getUpdatedUserByEmail(decodedToken.email);
        if (updatedUser) {
            return updatedUser;
        }
    }

    async addUserPost(files: any, request: Request) {
        const { body } = request;
        const decodedToken = this.helpJwtService.decodeJwt(request);

        const newPost: Post = {
            id: String(Math.floor(Math.random()*150000)),
            title: body.title,
            description: body.description,
            files: files.map(el => {
                return {
                    src: el.filename,
                    type: el.mimetype
                }
            }),
            date: String(new Date()),
            likes: [],
        };

        const updatedUser = await
this.userModel.findOneAndUpdate({email : decodedToken.email},
{$push: {
            posts: newPost

```

1

```
    }}, {returnDocument: "after"}));

    if (updatedUser) {
        throw new HttpException(updatedUser, 200);
    } else {
        throw new HttpException({ message: "Попробуйте снова,
        произошла неизвестная ошибка"}, 400);
    }
}

    async addLikeToUserPost(request: Request, postId: string,
    userId: string) {
        const decodedToken = this.helpJwtService.decodeJwt(request);
        const updatedUser = this.userModel.findOneAndUpdate({
            userId: userId, "posts.id": postId }, {
            $addToSet: { "posts.$.likes": decodedToken.userId }
        }, { returnDocument: "after" });

        if (updatedUser) return updatedUser;
        throw new HttpException({ message: "Попробуйте снова,
        произошла неизвестная ошибка"}, 400);
    }

    async removeLikeFromUserPost(request: Request, postId: string,
    userId: string) {
        const decodedToken = this.helpJwtService.decodeJwt(request);
        const updatedUser = this.userModel.findOneAndUpdate({
            userId: userId, "posts.id": postId }, {
            $pull: { "posts.$.likes": decodedToken.userId }
        }, { returnDocument: "after" });

        if (updatedUser) return updatedUser;
        throw new HttpException({ message: "Попробуйте снова,
        произошла неизвестная ошибка"}, 400);
    }
}
```

Приложение В
(рекомендуемое)

Программный код методов клиентской части приложения

```

export default function Peoples(): JSX.Element {

    const [currentChapter, setCurrentChapter] =
useState<string>('searching');
    const [isHintOpen, setIsHintOpen] = useState<boolean>(false);

    const currentWall = useStore(currentWall$);
    const router = useRouter();

    const handleGoToLink = (isGroup: boolean, linkId: number) => {
        if (isGroup) {
            router.push(`/groups/${linkId}`);
        } else {
            router.push(`/profile/${linkId}`);
        }
    };

    useEffect(() => {
        getCurrentWall();
        getInterests();
    }, []);

    return(
        <PageContainer>
            <div className={s.peoples}>
                <Head>
                    <title>Meetins - Люди</title>
                    <meta name="description" content="Ищите новые
знакомства!" key="desc" />
                    <meta property="og:title" content="Только у нас
Вы сможете найти себе хороших друзей и пообщаться!" />
                    <meta
                        name="keywords"
                        content="meetins, meetin-s, Meetins, Meetin-
s, знакомства, meetings, meet, люди, peoples, meetins seven peoples"
                    />
                    <meta
                        property="og:description"
                        content="Присоединяйтесь прямо сейчас, ищите
людей."
                    />
                </Head>
                <div className={s.chapters}>
                    <button
                        onClick={() => setCurrentChapter('searching')}
                        className={currentChapter === 'searching' ?
s.selectedChapter : s.notSelectedChapter}
                    >Поиск людей</button>
                    <FaRegCircleQuestion
                        className={s.hint}
                        onClick={() => setIsHintOpen(true)}

```

```

        />
        <button
            onClick={() => setCurrentChapter('groups')}
            className={currentChapter === 'groups' ?
s.selectedChapter : s.notSelectedChapter}
        >Сообщества</button>
        <button
            onClick={() => setCurrentChapter('list')}
            className={currentChapter === 'list' ?
s.selectedChapter : s.notSelectedChapter}
        >Новости</button>
    </div>
    <div className={s.activeChapter}>
        {
            currentChapter === 'searching' ?
<SearchingPeople /> :
            currentChapter === 'list' ?
                <LentaList
                    wallPosts={currentWall}
                    handleGoToLink={handleGoToLink}
                /> :
            currentChapter === 'groups' && <GroupsList
/>
        }
    </div>
    <CustomModal
        isDisplay={isHintOpen}
        title="Информация"
        actionConfirmed={setIsHintOpen}
        changeModal={setIsHintOpen}
        typeOfActions="none"
    >
        <div className={s.hintText}>
            Внимание, данные фильтры лучше использовать
при выключенной функции
            <span style={{ color: "var(--default-
color)"}}> "Фильтры контента"</span>.
            <div>*</div>
            <div className={s.block}>
                Если данная функция включена, применение
дополнительных фильтров может
                не дать желаемого результата.
            </div>
        </div>
    </CustomModal>
</div>
</PageContainer>
)
}
export default function Music() {

```

5

```

        const [addMusicModal, setAddMusicModal] =
useState<boolean>(false);
        if (!route.isReady) return;
        getDataForProfilePage(String(route.query.id));
    }, [route.isReady, route.asPath]);

    useEffect(() => {
        if (currentUser.userId) {
            getUserPlaces({ userId: currentUser.userId });
        }
    }, [currentUser])

    if (currentUserLoaded) {
        return(
            <PageContainer>
                <ProfileView
                    asyncLoaded={userLoaded}
                    addingImageStatus={addingImageStatus}
                    currentUser={currentUser}
                    authedUser={authedUser}
                    isAddPostModal={isAddPostModal}
                    isInviteModal={isInviteModal}
                    isEditTagOpen={isEditTagOpen}
                    handleSendInvite={handleSendInvite}
                    setChoosedEventForInvite={setChoosedEventForIn
vite}

                    setIsAddPostModal={setIsAddPostModal}
                    setIsInviteModal={setIsInviteModal}
                    setIsEditTagOpen={setIsEditTagOpen}
                    handleStartDialog={handleStartDialog}
                    onChangeInputImage={onChangeInputImage}
                    handleSaveNewStatus={handleSaveNewStatus}
                    changeAddingImageStatus={changeAddingImageStatus}
                    onAddingModalClick={onAddingModalClick}
                    handleAddUserIntoMarked={handleAddUserIntoMarked}
                    handleOpenEditTag={handleOpenEditTag}
                    currentUserPlaces={currentUserPlaces}
                />
            </PageContainer>
        )
    } else {
        return (
            <CustomLoader />
        )
    }
}

export default Profile;
export default function Login(): JSX.Element {
    const {
        register,

```

6

```

        handleSubmit,
        formState: { errors },
      } = useForm()
      const [gender, setGender] = useState<string>('')
    />
    <label
      onClick={() => setGender('female')}
      className={gender === 'female' ?
` ${s.checked}` : ''}
      htmlFor='gender_female'>
      <Image
        src={femaleIcon}
        alt='female icon'
        width={40}
        height={40}
      />
    </label>
    <input
      {...register('gender', { required: true
    }}}
      id='gender_female'
      value='female'
      type='radio'
    />
  </div>
  {errors.gender && (
    <span className={s.errorSpan}>{t('Пожалуйста
укажите пол')}

```



```

        onChange={handleChangeStatus}
      />
      <ButtonWithHint
        fontSize={14}
        title="?"
        hintTitle={
          `*Фильтры контента - это функция,
которая дает Вам возможность активировать
алгоритм подбора контента на
сайте (сообщества, люди, встречи, лента)
исходя из Ваших настроек профиля
(например интересов, закладок, активности).
*Включая данную функцию важно
понимать что количество контента на сайте может
уменьшиться, при этом вероятность
того что Вам попадется что-то не нужное или Вам не интересное
крайне мала.
`
        }
      />
    </div>
  </div>
  <div className={s.locale}>
    <h4>{t("Выбрать язык")}</h4>
    <div className={s.locales}>
      <div className={s.localeImg}>
        <Image src={us} width="62px"
height="50px" onClick={() => handleChangeLocale('en')} />
      </div>
      <div className={s.localeImg}>
        <Image src={ru} width="62px"
height="50px" onClick={() => handleChangeLocale('ru')} />
      </div>
    </div>
  </div>
  <div className={s.themes}>
    <h4>{t("Темы")}</h4>
    <button className={s.swipeTheme} onClick={() => handleSwipeTheme('black')}>Черная</button>
    <button className={s.swipeTheme} onClick={() => handleSwipeTheme('white')}>Белая</button>
  </div>
  <div className={s.info}>
    <h4>{t("Информация профиля")}</h4>
    { isLoading ? <div className={s.profileForm}>
      <ProfileInfoForm />
    </div> : <CustomLoader /> }
  </div>
  <div className={s.management}>
    <h4>{t("Управление аккаунтом")}</h4>
    <div className={s.manageAcc}>

```

```

        {isLoading ? <div className={s.accountForm}>
          <ManageAccountForm />
        </div> : <CustomLoader />}
      </div>
      <div className={` ${s.infoText}`}>
        <div className={` ${s.description}`>
          {t('Вы можете изменить адрес аккаунта
на более привлекательный')}.<br />
          {t('Другие пользователи смогут найти
Вас по адресу')} <b>meetins.ru/ВашАдрес</b><br />
          <b>Отключение</b><br/>
          {t('Вы можете удалить свой аккаунт,
тогда другие пользователи больше не смогут')}<br />
          {t('Вас найти на нашем сайте')}.
        </div>
      </div>
    </div>
  </div>
  <div>
    <button className={s.deleteBtn} onClick={()
=> changeModal(true)}>{t("Удалить аккаунт")}</button>
  </div>
</div>
</div>
  <Modal isDisplay={isModal} changeModal={changeModal}
actionConfirmed={deleteAccount}>
    <h6>{t('Подвердите действие - Удаление
аккаунта')}.</h6>
  </Modal>
</div>
)
}
export default function Messenger(): JSX.Element {
  const activeChat$ = useStore(activeChat);
  const isMobile$ = useStore(isMobile);

  const isChatNeededToShow = Boolean(activeChat$.dialogId);

  return(
    <PageContainer>
      { !isMobile$
        ?
        <div className={s.messengerContent}>
          <div className={` ${s.chatList} ${s.block}`}>
            <ChatList/>
          </div>
          <div className={` ${s.chatZone}`>
            {isChatNeededToShow ? <ChatZone
activeChat$={activeChat$} /> :
            <div className={s.noChat}>

```

0

```

                                <h3 className={s.alert}>Выберите
диалог...</h3>
                                </div>
                                }
                                </div>
                                </div>
                                :
                                <div className={s.mobileMessengerContent}>
                                {
                                    !isChatNeededToShow
                                    ? <MobileChatList />
                                    :
                                    <div className={` ${s.mobileChatZone}`>
                                        <div className={s.mobileChatZoneContent}>
                                            <ChatZone activeChat$={activeChat$} />
                                        </div>
                                    </div>
                                }
                                </div>
                                }
                                </PageContainer>
                                )
                                }
export default function MeetingsRoom() {

    const router = useRouter();
    const selectedMeeting$ = useStore(selectedMeeting);
    const [isAddCommentModal, setIsAddCommentModal] =
useState<boolean>(false);
    const [isOpenAllFilesModal, setIsOpenAllFilesModal] =
useState<boolean>(false);
    const user$ = useStore($user);
    const isUserInParticipants =
selectedMeeting$.participants.filter(el => el.userId ===
user$.userId).length !== 0;

    const handleFileUpload = (event: ChangeEvent<HTMLInputElement>)
=> {
        uploadFileToMediaMeeting({
            event,
            meetingId: selectedMeeting$.meetingId
        });
    };

    const handleRegInCurrentMeeting = (meetingId: string) => {
        if (!isUserInParticipants) {
            handleRegInMeeting(meetingId);
        } else {
            addNotification({
                text: "Вы уже в списке участников.",
                textColor: "white",
            });
        }
    };

```

1

```

        time: 3000,
        color: 'orange'
      ))
    }
  };

  const handleAddComment = () => {
    if (isUserInParticipants) {
      setIsAddCommentModal(true);
    } else {
      addNotification({
        text: "Сначала запишитесь на встречу.",
        textColor: "white",
        time: 3000,
        color: 'orange'
      })
    }
  };

  const handleOpenAllFiles = () => {
    setIsOpenAllFilesModal(true);
  };

  return (
    <PageContainer>
      <>
        <MeetingsRoomPageView
          selectedMeeting={selectedMeeting$}
          handleFileUpload={handleFileUpload}
          handleAddComment={handleAddComment}
          handleRegInMeeting={handleRegInCurrentMeeting}
          handleOpenAllFiles={handleOpenAllFiles}
          isUserInParticipants={isUserInParticipants}
        />
        <CustomModal
          isDisplay={isAddCommentModal}
          changeModal={setIsAddCommentModal}
          actionConfirmed={setIsAddCommentModal}
          typeOfActions="none"
          title="Добавить комментарий"
        >
          <AddCommentIntoMeeting />
        </CustomModal>
        <CustomModal
          isDisplay={isOpenAllFilesModal}
          changeModal={setIsOpenAllFilesModal}
          actionConfirmed={setIsOpenAllFilesModal}
          typeOfActions="none"
          title="Все медиа файлы со встречи"
        >
          <MeetingsFilesList

```

2

```

        files={selectedMeeting$?.files}
      />
    </CustomModal>
  </>
</PageContainer>
)
}
export default function Meetings() {

  const [currentMenu, setCurrentMenu] = useState<number>(1);
  const currentMeetings$ = useStore(currentMeetings);
  const authedUser$ = useStore($user);
  const router = useRouter();

  const handleGoToMeetingRoom = (meeting: IMeeting) => {
    setSelectedMeeting(meeting);
    router.push(`/meetingsRoom/${meeting.meetingId}`);
  };

  useEffect(() => {
    getAllMeetings();
  }, []);

  return (
    <PageContainer>
      <MeetingsPageView
        handleGoToMeetingRoom={handleGoToMeetingRoom}
        currentMeetings={currentMeetings$}
        currentMenu={currentMenu}
        setCurrentMenu={setCurrentMenu}
        authedUser={authedUser$}
      />
    </PageContainer>
  )
}
export default function Marks(): JSX.Element {

  const [currentMark, setCurrentMark] =
  useState<string>('events');
  const user$ = useStore($user);
  const markedUsersInfo$ = useStore($markedUsersInfo);

  useEffect(() => {
    getMarkedUsersInfo();
  }, []);

  return(
    <PageContainer>
      <div className={s.content}>
        <div className={s.types}>
          <button

```

3

```

        className={currentMark === 'peoples' ? s.active
: s.inactive}
        onClick={() => setCurrentMark('peoples')}>Люди
      </button>
      <button
        className={currentMark === 'events' ? s.active
: s.inactive}
        onClick={() => setCurrentMark('events')}>События
      </button>
    </div>
    <div className={s.result}>
      { currentMark === 'events' && <MarkedEvents
user={user$} /> }
      { currentMark === 'peoples' && <MarkedUsers
markedUsers={markedUsersInfo$} /> }
    </div>
  </div>
</PageContainer>
)
}
export default function Invites() {

  const authedUser$ = useStore($user);
  const unitedEventsInfo$ = useStore(unitedInviteEvents);
  const loadedStatus$ = useStore(loadedStatus);
  const router = useRouter();

  const handleWatch = (eventId: number) => {
    router.push(`/eventInfo/${eventId}`);
  };

  const handleDecline = (event: IInnerInvites) => {
    declineInnerInvite(event);
  };

  const handleVisit = (login: number) => {
    router.push(`/profile/${login}`);
  }

  useEffect(() => {
    getUserInnerInvitesEventInfo();
    getUserOuterInvitesEventInfo();
  }, []);

  return (
    <PageContainer>
      <InvitesPageView
        handleWatch={handleWatch}
        handleDecline={handleDecline}
        handleVisit={handleVisit}
        authedUser={authedUser$}

```

4

```

        unitedEventsInfo={unitedEventsInfo$}
        loadedStatus={loadedStatus$}
      />
    </PageContainer>
  )
}
export default function Interests() {

  const currentInterests$ = useStore($currentInterestsList);
  const titleRef = useRef<HTMLInputElement>(null);

  const handleSendForm = () => {
    if (titleRef.current && titleRef.current.value.length >= 3)
    {
      addInterest(titleRef.current.value);
      addNotification({
        time: 3000,
        color: "green",
        textColor: "white",
        text: "Ваша заявка успешно отправлена!"
      })
    } else {
      addNotification({
        time: 3000,
        color: "orange",
        textColor: "white",
        text: "Название интереса не менее 3 симв."
      })
    }
  };

  useEffect(() => {
    getInterests();
  }, []);

  return (
    <PageContainer>
      <InterestsPageView
        currentInterests={currentInterests$}
        handleSendForm={handleSendForm}
        titleRef={titleRef}
      />
    </PageContainer>
  )
}
export default function Groups() {

  const router = useRouter();
  const groupInfo$ = useStore(groupInfo);
  const groupMembersInfo$ = useStore(groupMembersInfo);
  const authedUser$ = useStore($user);

```

5

```

        const isAutherUserCreator = authedUser$?.userId ===
groupInfo$?.creatorId;
    const [modals, setModals] = useState({
        isSettingsGroupOpen: false,
        isCommentsModalOpen: false,
        isAddingPostModalOpen: false,
        isTalksOpen: false,
        isTalkMessagesOpen: false,
        isTalkCreationOpen: false,
        isPhotosOpen: false,
        isVideosOpen: false
    });
    const [selectedTalkId, setSelectedTalkId] = useState<number>();
    const [activePostId, setActivePostId] = useState<number>();
    const videoPhotoAttachmentsInfo =
destrucutreFilesInGroupPost(groupInfo$);

    const handleOpenGroupSettings = () => {
        setModals({
            ...modals,
            isSettingsGroupOpen: true
        });
    };
    const handleOpenComments = (postId: number) => {
        setActivePostId(postId);
        setModals({
            ...modals,
            isCommentsModalOpen: true
        });
    };
    const handleOpenAddingPost = () => {
        setModals({
            ...modals,
            isAddingPostModalOpen: true
        });
    };
    const handleOpenTalks = () => {
        setModals({
            ...modals,
            isTalksOpen: true
        });
    };
    const handleOpenTalkMessages = (talkId: number) => {
        setSelectedTalkId(talkId)
        setModals({
            ...modals,
            isTalkMessagesOpen: true
        });
    };
    const handeOpenTalkCreation = () => {
        setModals({

```

```

        ...modals,
        isTalkCreationOpen: true
    });
};
const handleSuccessSubmitTalkCreation = () => {
    setModals({
        ...modals,
        isTalkCreationOpen: false,
        isTalksOpen: true
    });
};
const handleAddNewMessage = (message: IGroupTalkMessage) => {
    addActiveGroupTalkMessage(message);
};
const handleOpenPhotos = () => {
    setModals({
        ...modals,
        isPhotosOpen: true
    });
};
const handleOpenVideos = () => {
    setModals({
        ...modals,
        isVideosOpen: true
    });
};
const handleLikePost = (post: IGroupPost, groupId: number) => {
    if (post.likes.includes(authedUser$.userId)) {
        unlikePostInGroup({ groupId: groupId, postId: post.id
    });
    } else {
        likePostInGroup({ groupId: groupId, postId: post.id });
    }
}

useEffect(() => {
    if(router.query?.id) {
        getGroupById(+router.query.id);
        getGroupMembersInfo(+router.query.id);
    }
}, [router.query.id]);

return (
    <PageContainer>
        <>
            <GroupInfoPageView
                authedUser={authedUser$}
                groupInfo={groupInfo$}
                isAutherUserCreator={isAutherUserCreator}
                handleOpenGroupSettings={handleOpenGroupSettings}
                isSettingsGroupOpen={modals.isSettingsGroupOpen}

```

```

        groupMembersInfo={groupMembersInfo$}
        handleOpenComments={handleOpenComments}
        handleOpenAddingPost={handleOpenAddingPost}
        handleOpenTalks={handleOpenTalks}
        handleOpenPhotos={handleOpenPhotos}
        handleLikePost={handleLikePost}
        handleOpenVideos={handleOpenVideos}
        videoPhotoAttachmentsInfo={videoPhotoAttachmen
tsInfo}
    />
    <CustomModal
        isDisplay={modals.isSettingsGroupOpen}
        changeModal={(status) => setModals({ ...modals,
isSettingsGroupOpen: status })}
        actionConfirmed={(status) => setModals({
...modals, isSettingsGroupOpen: status })}
        typeOfActions="none"
        title="Управление сообществом"
    >
        <ManageGroup />
    </CustomModal>
    <CustomModal
        isDisplay={modals.isCommentsModalOpen}
        changeModal={(status) => setModals({ ...modals,
isCommentsModalOpen: status })}
        actionConfirmed={(status) => setModals({
...modals, isCommentsModalOpen: status })}
        typeOfActions="none"
        title="Комментарии"
    >
        <GroupCommentsView
            groupInfo={groupInfo$}
            activePostId={activePostId}
        />
    </CustomModal>
    <CustomModal
        isDisplay={modals.isAddingPostModalOpen}
        changeModal={(status) => setModals({ ...modals,
isAddingPostModalOpen: status })}
        actionConfirmed={(status) => setModals({
...modals, isAddingPostModalOpen: status })}
        typeOfActions="none"
        title="Добавить публикацию"
    >
        <AddNewPostIntoGroupForm
groupId={String(router.query.id)} />
    </CustomModal>
    <CustomModal
        isDisplay={modals.isTalksOpen}
        changeModal={(status) => setModals({ ...modals,
isTalksOpen: status })}

```

```

        actionConfirmed={(status) => setModals({
...modals, isTalksOpen: status })}
        typeOfActions="none"
        title="Обсуждения"
    >
        <GroupTalks
            handleOpenTalkMessages={handleOpenTalkMess
ages}

            handeOpenTalkCreation={handeOpenTalkCreation}
            groupId={groupInfo$.groupId}
            groupTalks={groupInfo$.talks}
        />
    </CustomModal>
    <CustomModal
        isDisplay={modals.isTalkMessagesOpen}
        changeModal={(status) => setModals({ ...modals,
isTalkMessagesOpen: status })}
        actionConfirmed={(status) => setModals({
...modals, isTalkMessagesOpen: status })}
        typeOfActions="custom"
        actionsComponent={
            <AddNewMessageIntoGroupTalk
                groupId={groupInfo$.groupId}
                talkId={selectedTalkId}
                onSaveMessage={handleAddNewMessage}
            />
        }
        title="Обсуждение"
    >
        <GroupTalksMessagesView
            talkId={selectedTalkId}
            groupId={groupInfo$.groupId}
        />
    </CustomModal>
    <CustomModal
        isDisplay={modals.isTalkCreationOpen}
        changeModal={(status) => setModals({ ...modals,
isTalkCreationOpen: status })}
        actionConfirmed={(status) => setModals({
...modals, isTalkCreationOpen: status })}
        typeOfActions="none"
        title="Создать обсуждение"
    >
        <AddNewTalkInGroup
            groupId={String(router.query.id)}
            successSubmit={handleSuccessSubmitTalkCrea
tion}
        />
    </CustomModal>
    <CustomModal
        isDisplay={modals.isPhotosOpen}

```

```

        changeModal={({status}) => setModals({ ...modals,
isPhotosOpen: status })}
        actionConfirmed={({status}) => setModals({
...modals, isPhotosOpen: status })}
        typeOfActions="none"
        title="Фотографии в сообществе"
      >
        <GroupAttachments
images={videoPhotoAttachmentsInfo.images} />
      </CustomModal>
      <CustomModal
        isDisplay={modals.isVideosOpen}
        changeModal={({status}) => setModals({ ...modals,
isVideosOpen: status })}
        actionConfirmed={({status}) => setModals({
...modals, isVideosOpen: status })}
        typeOfActions="none"
        title="Видео в сообществе"
      >
        <GroupAttachments
videos={videoPhotoAttachmentsInfo.videos} />
      </CustomModal>
    </>
  </PageContainer>
)
}
export default function CategoryEventInfo(): JSX.Element {
  const router = useRouter();
  const { nameCategory } = router.query;
  const { t } = useTranslation();

  return(
    <PageContainer>
      <div className={s.content}>
        <div className={` ${s.list} ${s.block}`}>
          {
            nameCategory && typeof nameCategory ===
"string"
              ? <EventsList category={nameCategory}/> :
null
          }
        </div>
      </div>
    </PageContainer>
  )
}
export default function Events(): JSX.Element {
  const { t } = useTranslation();

```

0

```

return (
  <PageContainer>
    <div className={s.events}>
      <Head>
        <title>Meetins-{t('Events')}</title>
        <meta name="description" content="Meetins -
события" key="desc" />
        <meta property="og:title" content="Ищите события
и приглашайте на них своих друзей!" />
        <meta
          property="og:description"
          content="Приглашайте своих новых друзей куда
угодно!"
        />
      </Head>
      <div className={s.mainContent}>
        <Card
          img_src = "images/cinema.jpg"
          name = {t("Кино")}
          categoryRoute='cinema'
        />
        <Card
          img_src = "images/theatre.jpg"
          name = {t("Театр")}
          categoryRoute='theater'
        />
        <Card
          img_src = "images/concert.jpg"
          name = {t("Концерты")}
          categoryRoute='concert'
        />
        <Card
          img_src = "images/exhibition.jpg"
          name = {t("Выставки")}
          categoryRoute='exhibition'
        />
        <Card
          img_src = "images/tour.jpg"
          name = {t("Экскурсии")}
          categoryRoute='tour'
        />
        <Card
          img_src = "images/quest.jpg"
          name = {t("Квесты")}
          categoryRoute='quest'
        />
      </div>
    </div>
  </PageContainer>
)
}

```

1

```

export default function EventInfo(): JSX.Element {

    const currentEventById$ = useStore(currentEventById);
    const currentEventCommentsById$ = useStore(currentEventCommentsById);
    const authedUser$ = useStore($user);

    const { query } = useRouter();

    const addUserEventHandler = (id: number) => {
        if (authedUser$.events.includes(String(currentEventById$.id))) {
            addNotification({ text: "Уже есть в вашем списке", time: 3000, color: 'orange', textColor: "black" });
            return;
        }
        addUserEvent(id);
    }

    useEffect(() => {
        if (query.id) {
            getEventById(String(query.id));
            getCommentsForEventById(String(query.id));
        }
    }, [query]);

    useEffect(() => {
        return () => {
            setCurrentEventById(null);
        }
    }, []);

    return(
        <PageContainer>
            {currentEventById$
            ?
                <EventBlock
                    addUserEvent={addUserEventHandler}
                    currentEventById={currentEventById$}
                    commentsEvent={currentEventCommentsById$}
                />
                : <Loader />
            }
        </PageContainer>
    )
};

export default function Confirmation() {

    const emailForConfirmation$ = useStore($emailForConfirmation);
    const router = useRouter();

```

2

```

    const sendConfirmationCode = (code: number) => {
        sendConfirmationCodeForAccept({ email:
emailForConfirmation$, code }).then(res => {
            if (res.status <= 217) {
                router.push(`/profile/${res.data.profile.user.logi
n}$`);
            }
        });
    }

    return (
        <ConfirmationView
sendConfirmationCode={sendConfirmationCode} />
    )
}
import axios, { AxiosError, AxiosRequestConfig } from 'axios'
import { createEffect, createEvent, createStore, sample } from
'effector'
import { IMarkedUserInfo, IUserTag, User } from '../interfaces';
import { instanseRouter } from './router_model';
import { handleLogOut } from './login_model';
import { addNotification } from './notifications_model';
import { IInterest } from '../interfaces/interest';

export const baseURL = 'http://localhost:5000/';
//export const baseURL = 'https://meetins-egorplat.amvera.io/';

export const instance = axios.create({
    baseURL: baseURL,
    withCredentials: true,
})
instance.interceptors.request.use((config: AxiosRequestConfig) => {
    config.headers = {
        'Content-Type': 'application/json',
    }
    return config;
}, (errors: AxiosError) => {
    return Promise.reject(errors);
})
instance.interceptors.response.use((response) => {
    return response;
}, (error: AxiosError) => {
    const ec: AxiosRequestConfig = error.config;
    const ers: number | undefined = error.response?.status;
    if (ers === 401) {
        handleLogOut();
        setIsAsyncLoaded(true);
    }
    if (ers >= 400 && ers <= 499) {
        const { message } = error.response.data;
    }
}

```

3

```

        if (message) {
            addNotification({
                text: message,
                color: "orange",
                textColor: "black",
                time: 3000
            });
        }
    }
    return error;
})

export const setIsMobile = createEvent<boolean>();
export const isMobile = createStore<boolean>(false).on(setIsMobile,
(_, isMobile) => {
    return isMobile;
})
export const setIsUserLoaded = createEvent<boolean>();
export const isUserLoaded =
createStore<boolean>(false).on(setIsUserLoaded, (_, status) => {
    return status;
})
export const setIsCurrentUserLoaded = createEvent<boolean>();
export const isCurrentUserLoaded =
createStore<boolean>(false).on(setIsCurrentUserLoaded, (_, status)
=> {
    return status;
})
export const setIsAsyncLoaded = createEvent<boolean>();
export const isAsyncLoaded =
createStore<boolean>(false).on(setIsAsyncLoaded, (_, tokenUpdated)
=> {
    return tokenUpdated;
})
export const setUser = createEvent<User | null>()
export const $user = createStore<User | null>(null).on(setUser, (_,
userDetails) => {
    return userDetails;
})
export const setOnlineUsers = createEvent<any[]>()
export const $onlineUsers =
createStore<any[]>([]).on(setOnlineUsers, (_, onlineUserList) => {
    return onlineUserList;
})
export const setCurrentProfileUser = createEvent<User>();
export const $currentProfileUser = createStore<User>({} as
User).on(setCurrentProfileUser, (_, currentUser) => {
    return currentUser;
})
export const setCurrentPage = createEvent<string>()
export const $currentPage = createStore<string>('').on(

```

4

```

    setCurrentPage,
    (_, currPage) => {
      return currPage;
    }
  )

export const setIsScrollPageBlocked = createEvent<boolean>()
export const $scrollPageBlocked = createStore<boolean>(false).on(
  setIsScrollPageBlocked,
  (_, scrollPageBlocked) => {
    return scrollPageBlocked;
  }
)

export const setCurrentInterestsList = createEvent<IInterest[]>()
export const addNewInterest = createEvent<IInterest>()
export const $currentInterestsList = createStore<IInterest[]>([])
.on(setCurrentInterestsList, (_, interests) => {
  return interests;
})
.on(addNewInterest, (interests, interest) => {
  return [...interests, interest];
})

export const setMarkedUsersInfo = createEvent<IMarkedUserInfo[]>()
export const $markedUsersInfo =
createStore<IMarkedUserInfo[]>([]).on(setMarkedUsersInfo, (_,
usersInfo) => {
  return usersInfo;
})

export const getUserInterests = createEffect(async (userInterests)
=> {
  const response = await instance.post('interests/get-interests-
by-id', JSON.stringify({ interests: userInterests}));
  if(response.status <= 217) {
    return response.data;
  }
})

export const addUserIntoMarkedList = createEffect(async (userId:
string) => {
  const response =
instance.post('users/addUserIntoMarkedList', JSON.stringify({
neededUserId: userId }));
  return response;
})

export const removeUserFromMarkedList = createEffect(async (userId:
string) => {

```

5

```

        const response = await
instance.delete(`users/removeUserFromMarkedList/${userId}`);
    return response;
})

export const updateInterests = createEffect(async (interests:
string[]) => {
    const response = await instance.post('users/updateUserInterest',
JSON.stringify({ interests: interests }));
    if(response.status <= 217) {
        setUser(response.data);
        setCurrentProfileUser(response.data);
        return response;
    }
})

export const getMarkedUsersInfo = createEffect(async () => {
    const response = await instance.get('users/getMarkedUsersInfo');
    return response;
})

export const getInterests = createEffect(async () => {
    const response = await instance.get('interests/get-interests');
    return response;
})

export const addInterest = createEffect(async (title: string) => {
    const response = await instance.post('interests/add-interest',
{ title });
    return response;
})

export const getUserData = createEffect(async () => {
    setIsUserLoaded(false);
    const response = await instance.get('profile/my-profile');
    if(response.status === 200) {
        setUser(response.data);
        setIsUserLoaded(true);
    }
    return response;
})

export const getInitialUserDataAndCheckAuth = createEffect(() => {
    const instanseRouter$ = instanseRouter.getState();
    const savedRoute = localStorage.getItem("previousPage");
    setIsUserLoaded(false);
    getUserData().then( (res) => {
        if(res.status === 200) {
            setIsUserLoaded(true);
            instanseRouter$.push(savedRoute);
        } else {

```

6

```

        instanseRouter$?.push('/login');
    }
  })
});

export const getUserDataByUserId = createEffect(async (userId:
string | number) => {
  const response = await instance.post(`users/getUserByUserId`);
  if(response.data) {
    return response.data;
  }
})

export const getUserDataByLoginUrl = createEffect(async (loginUrl:
string | number) => {
  setIsUserLoaded(false);
  const response = await instance.get(`profile/by-
login/${loginUrl}`);
  if(response.status <= 202) {
    setIsUserLoaded(true);
  }
  return response;
})

export const getDataForProfilePage = createEffect((userId: string)
=> {
  setIsCurrentUserLoaded(false);
  getUserDataByLoginUrl(userId).then( (res) => {
    if(res.status === 200) {
      setCurrentProfileUser(res.data);
      setIsCurrentUserLoaded(true);
    }
  })
})

export const updateTokens = createEffect(async () => {
  const response = await instance.post('user/refresh-token',
localStorage.getItem('refrash-token'));
  if(response.status === 200) {
    localStorage.setItem('access-token',
response.data.accessToken);
    localStorage.setItem('refrash-token',
response.data.refreshToken);
  }
  return response;
})

export const sendNewUserPost = createEffect(async (formData:
FormData) => {
  const response = await instance.post('users/addUserPost',
formData);
  if(response.status === 200) {
    setUser(response.data);
  }
})

```

7

```

        addNotification({ text: 'Пост успешно создан на вашей
странице!', color: 'green', time: 3000, textColor: "white" })
    }
    return response;
})

export const updateUserTag = createEffect(async (data: IUserTag) => {
    {
        const response = await instance.put('users/updateUserTag',
data);
        return response;
    }
})

export const likeUserPost = createEffect(async (data: { userId:
string, postId: string }) => {
    {
        const response = await
instance.put(`users/like/${data.userId}/${data.postId}`);
        return response;
    }
})

export const unlikeUserPost = createEffect(async (data: { userId:
string, postId: string }) => {
    {
        const response = await instance.put(`users/remove-
like/${data.userId}/${data.postId}`);
        return response;
    }
})

sample({
    clock: [likeUserPost.doneData, unlikeUserPost.doneData],
    filter: (res) => res.status <= 217,
    fn: (res) => res.data,
    target: setCurrentProfileUser
})

sample({
    clock: [addInterest.doneData],
    filter: (res) => res.status <= 217,
    fn: (res) => res.data,
    target: addNewInterest
})

sample({
    clock: getInterests.doneData,
    filter: (res) => res.status <= 217,
    fn: (res) => res.data,
    target: setCurrentInterestsList
})

sample({
    clock: [addUserIntoMarkedList.doneData,
updateUserTag.doneData],

```

8

```

    filter: (res) => res.status <= 217,
    fn: (res) => res.data,
    target: setUser
  })

sample({
  clock: [getMarkedUsersInfo.doneData,
removeUserFromMarkedList.doneData],
  filter: (res) => res.status <= 217,
  fn: (res) => res.data,
  target: setMarkedUsersInfo
})
import { createEffect, createEvent, createStore } from "effector";
import { IShortEventInfo, IEventInfoCard, IUnitedInvitesEvent,
IInnerInviteEvent, IOuterInviteEvent, IEventComments } from
"../interfaces/events";
import { instance, setUser } from "../store";
import { sample } from 'effector';
import { addNotification } from "../notifications_model";
import { IInnerInvites } from "../interfaces";

export const setCurrentEvents = createEvent<IShortEventInfo[]>();
export const currentEvents =
createStore<IShortEventInfo[]>([]).on(setCurrentEvents, (_,
newEvents) => {
  return newEvents;
});
export const setUserEvents = createEvent<IShortEventInfo[]>();
export const userEvents =
createStore<IShortEventInfo[]>([]).on(setUserEvents, (_,
newUserEvents) => {
  return newUserEvents;
});
export const setCurrentEventById = createEvent<IEventInfoCard>();
export const currentEventById = createStore<IEventInfoCard |
null>(null).on(setCurrentEventById, (_, currentEvent) => {
  return currentEvent;
});
export const setCurrentEventCommentsById =
createEvent<IEventComments[]>();
export const currentEventCommentsById =
createStore<IEventComments[]>([])
.on(setCurrentEventCommentsById, (_, currentEventComments) => {
  return currentEventComments;
});
export const setLoadedStatus = createEvent<boolean>();
export const loadedStatus =
createStore<boolean>(false).on(setLoadedStatus, (_, newStatus) => {
  return newStatus;
});

```

```

export const setCurrentEventsInfoLoaded = createEvent<boolean>();
export      const      currentEventsInfoLoaded      =
createStore<boolean>(false).on(setCurrentEventsInfoLoaded,      (_,
newStatus) => {
    return newStatus;
});

export      const      setUnitedInnerInviteEvents      =
createEvent<IInnerInviteEvent[]>();
export      const      setUnitedOuterInviteEvents      =
createEvent<IOuterInviteEvent[]>();
export const removeFromUnitedInnerInvite = createEvent<number>();
export      const      unitedInviteEvents      =
createStore<IUnitedInvitesEvent>({
    innerInvites: [],
    outerInvites: []
});

unitedInviteEvents.on(setUnitedInnerInviteEvents,      (prev,
innerInvites) => {
    return {...prev, innerInvites: innerInvites}
});
unitedInviteEvents.on(setUnitedOuterInviteEvents,      (prev,
outerInvites) => {
    return {...prev, outerInvites: outerInvites}
})
unitedInviteEvents.on(removeFromUnitedInnerInvite,      (prev,
removingInviteId) => {
    return {...prev, innerInvites: prev.innerInvites.filter(el =>
el.id !== removingInviteId)}
})

export const sendInviteToUser = createEffect(async (data: {userId:
string | number, eventId: string | number}) => {
    const response = await instance.post(
        'event/sendInviteToUser', {userIdTo: data.userId, eventId:
data.eventId}
    );
    return response;
})

export const getEvents = createEffect(async (info: {categoryName:
string, page: number}) => {
    const response = await instance.post(
        'event/getEventsCategory', {nameCategory: info.categoryName,
page: info.page}
    );
    return response;
})

export const getEventById = createEffect(async (id: string) => {

```

0

```

    const response = await instance.post(
      'event/getEventInfoById', {eventId: id}
    );
    return response;
  })

export const getUserEventsInfo = createEffect(async () => {
  const response = await instance.get(
    'event/getUserEventsInfo'
  );
  return response;
})

export const getUserInnerInvitesEventInfo = createEffect(async () =>
{
  const response = await instance.get(
    'event/getUserInnerInvitesEventInfo'
  );
  return response;
})

export const getCommentsForEventById = createEffect(async (eventId:
string | number) => {
  const response = await instance.post(
    'event/getCommentsForEventById', { eventId }
  );
  return response;
})

export const getUserOuterInvitesEventInfo = createEffect(async () =>
{
  const response = await instance.get(
    'event/getUserOuterInvitesEventInfo'
  );
  return response;
})

export const addUserEvent = createEffect(async (id: number) => {
  const response = await instance.post(
    'users/addUserEvent', {eventId: String(id)}
  );
  return response;
})

export const deleteUserEvent = createEffect(async (id: number) => {
  const response = await instance.post(
    'users/deleteUserEvent', {eventId: String(id)}
  );
  return response;
})

```

1

```

export const declineInnerInvite = createEffect(async (event:
IInnerInvites) => {
  const response = await instance.put(
    'event/declineInnerInvite', { fromUserId: event.fromUserId,
eventId: event.eventId }
  );
  return response;
})

sample({
  clock: getUserEventsInfo.doneData,
  filter: response => response.status <= 201,
  fn: response => response.data,
  target: setUserEvents
})

sample({
  clock: getUserEventsInfo.doneData,
  filter: response => response.status <= 201,
  fn: response => true,
  target: setCurrentEventsInfoLoaded
})

sample({
  clock: declineInnerInvite.doneData,
  filter: response => response.status <= 201,
  fn: response => response.data,
  target: removeFromUnitedInnerInvite
})

sample({
  clock: addUserEvent.doneData,
  filter: response => response.status === 201,
  fn: response => response.data,
  target: setUser
})
sample({
  clock: deleteUserEvent.doneData,
  filter: response => response.status === 201,
  fn: response => response.data,
  target: setUser
})
sample({
  clock: getEventById.doneData,
  filter: response => response.status === 201,
  fn: response => response.data,
  target: currentEventById
})
sample({
  clock: getEvents.doneData,
  filter: response => response.status === 201,

```

2

```

        fn: response => response.data,
        target: currentEvents
    })

    sample({
        clock: getUserOuterInvitesEventInfo.doneData,
        filter: response => response.status <= 217,
        fn: response => response.data,
        target: setUnitedOuterInviteEvents
    })
    sample({
        clock: [getUserInnerInvitesEventInfo.doneData],
        filter: response => response.status <= 217,
        fn: response => response.data,
        target: setUnitedInnerInviteEvents
    })
    sample({
        clock: getCommentsForEventById.doneData,
        filter: response => response.status <= 217,
        fn: response => response.data,
        target: setCurrentEventCommentsById
    })
    sample({
        clock: addUserEvent.doneData,
        filter: response => response.status <= 217,
        fn: (response) => {
            return { text: "Успешно добавлено в 'Закладки'", time: 3000,
color: "green", textColor: "white" }
        },
        target: addNotification
    })

    sample({
        clock: getUserInnerInvitesEventInfo.doneData,
        filter: response => response.status <= 217 &&
response.data.length === 0,
        fn: (response) => {
            return { text: "Возможно у вас пока нет приглашений!", time:
3000, color: "blue", textColor: "white" }
        },
        target: addNotification
    })

    sample({
        clock: getUserInnerInvitesEventInfo.pending,
        fn: () => false,
        target: loadedStatus
    })
    sample({
        clock: getUserInnerInvitesEventInfo.doneData,
        fn: () => true,

```

3

```

    target: loadedStatus
  })
  sample({
    clock: getEvents.pending,
    fn: () => false,
    target: loadedStatus
  })
  sample({
    clock: getEvents.doneData,
    fn: () => true,
    target: loadedStatus
  })
  sample({
    clock: getUserEventsInfo.pending,
    fn: () => false,
    target: loadedStatus
  })
  sample({
    clock: getUserEventsInfo.doneData,
    fn: () => true,
    target: loadedStatus
  })
import { attach, createEffect, createEvent, createStore, sample }
from "effector";
import { IDialogMessage, IMyDialog, INewDialog, User } from
"../interfaces";
import { defaultDialog } from "../mock/defaultDialog";
import { $user, instance } from "../store";

export const setActiveChat = createEvent<IMyDialog>();
export const activeChat = createStore<IMyDialog>(defaultDialog).on(setActiveChat, (_,
newActiveChat) => {
  return newActiveChat;
})
export const setMyDialogs = createEvent<IMyDialog[]>();
export const myDialogs = createStore<IMyDialog[] |
null>(null).on(setMyDialogs, (_, newMyDialogs) => {
  return newMyDialogs;
})
const handleIsUserUnReadMessagesExists = createEffect(({ params: {
myDialogs: IMyDialog[], authedUser: User }}) => {
  setIsUserUnReadMessagesExists(false);
  params.myDialogs.map(dialog => {
    dialog.messages.map(message => {
      if (!message.isRead && params.authedUser.userId !==
message.senderId) {
        setIsUserUnReadMessagesExists(true);
        return;
      }
    })
  })
})

```

4

```

    ))
  })
  sample({
    source: { myDialogs: myDialogs, authedUser: $user },
    target: handleIsUserUnReadMessagesExists
  })

  export const setIsUserUnReadMessagesExists =
  createEvent<boolean>();
  export const isUserUnReadMessagesExists =
  createStore<boolean>(false).on(setIsUserUnReadMessagesExists, (_,
  status) => {
    return status;
  })

  export const setIsMyDialogsLoaded = createEvent<boolean>();
  export const isMyDialogsLoaded =
  createStore<boolean>(false).on(setIsMyDialogsLoaded, (_,
  newMyDialogs) => {
    return newMyDialogs;
  })
  export const setIsMessageWithFileLoaded = createEvent<boolean>();
  export const isMessageWithFileLoaded =
  createStore<boolean>(true).on(setIsMessageWithFileLoaded, (_,
  isMessageLoaded) => {
    return isMessageLoaded;
  })

  export const sendFileAndUploadActiveChat = createEffect((params: {
  file: any, datastore: {activeChat: IMyDialog } }) => {
    setIsMessageWithFileLoaded(false);
    const actualActiveChat = params.dataStore.activeChat;
    if(actualActiveChat) {
      if(actualActiveChat.userId == undefined) {
        const file = params.file;
        const formData = new FormData();
        formData.append('uploadedFile', file);
        formData.append('dialogId', actualActiveChat.dialogId);

        sendFileInDialog(
          formData
        ).then(res => {
          if (res.status <= 217) {
            setActiveChat({...actualActiveChat, messages:
[...actualActiveChat.messages,      res.data],      content:
res.data.content});
          }
          setIsMessageWithFileLoaded(true);
        })
      }
    }
  })
}

```

5

```

    })
    export const sendMessageAndUploadActiveChat = createEffect((params:
    { message: string, dataStore: {activeChat: IMyDialog} }) => {
        const actualActiveChat = params.dataStore.activeChat;
        if(actualActiveChat) {
            if(actualActiveChat.dialogId !== 'none') {
                sendMessageInDialog(
                    {dialogId: actualActiveChat.dialogId, content:
params.message}
                ).then((response) => {
                    setActiveChat({...actualActiveChat, messages:
[...actualActiveChat.messages, response?.data], content:
params.message});
                })
            } else {
                startNewDialog({userId: actualActiveChat.userId,
messageContent: params.message}).then((response) => {
                    setActiveChat({
                        dialogId: response?.data.dialogId,
                        userName: actualActiveChat.userName,
                        userAvatar: actualActiveChat.userAvatar,
                        isRead: true,
                        content: params.message,
                        messages: [response?.data],
                        status: true
                    });
                });
            }
        }
    });
    export const createdSendMessageAndUploadActiveChat = attach({
        effect: sendMessageAndUploadActiveChat,
        source: {activeChat: activeChat},
        mapParams: (message: string, dataStore) => {
            return {message: message, dataStore: dataStore}
        },
    })

    export const createdSendFileAndUploadActiveChat = attach({
        effect: sendFileAndUploadActiveChat,
        source: {activeChat: activeChat},
        mapParams: (file: any, dataStore) => {
            return {file: file, dataStore: dataStore}
        },
    })

```

6

```

export const getMyDialogs = createEffect(async (isFirstGetDialogs:
boolean) => {
  if (isFirstGetDialogs) setIsMyDialogsLoaded(false);
  try {
    const response = await instance.get('chat/my-dialogs');
    if (isFirstGetDialogs) setIsMyDialogsLoaded(true);
    return response;
  }
  catch(error) {
    console.log(error);
  }
});

export const checkDialog = createEffect(async (user: User) => {
  try {
    const response = await instance.post('chat/check-dialog',
{userId: user.userId});
    if(response.data.length !== 0) {
      setActiveChat({
        ...defaultDialog,
        dialogId: response.data[0].dialogId,
        userAvatar: user.avatar,
        userName: user.name,
        messages: response.data[0].messages
      });
      return response;
    } else {
      setActiveChat({...defaultDialog, userName: user.name,
userAvatar: user.avatar, userId: user.userId, dialogId: "none"});
    }
  }
  catch(error) {
    console.log(error);
  }
});

export const getDialogMessages = createEffect(async (chosedDialog:
IMyDialog) => {
  try {
    const response = await instance.post('chat/messages',
JSON.stringify({dialogId: chosedDialog.dialogId}));
    if(response.status === 200) {
      const activeDialogWithMessages: IMyDialog =
{...chosedDialog, messages: response.data};
      setActiveChat(activeDialogWithMessages);
      return response;
    }
  }
  catch(error) {
    console.log(error);
  }
});

```

7

```

    }
  })

export const updatedIsReadMessagesInActiveDialog =
createEffect(async (dialogId: string) => {
  const response = await instance.post('chat/mark-messages-as-
readed', { dialogId: dialogId });
  return response;
})

export const sendFileInDialog = createEffect(async (message:
FormData) => {
  const response = await instance.post('chat/send-file-to-chat',
message);
  return response;
})

export const sendMessageInDialog = createEffect(async (message:
IDialogMessage) => {
  try {
    const response = await instance.post('chat/send-message',
message);
    if(response.status === 200) {
      return response;
    }
  }
  catch(error) {
    console.log(error);
  }
})

export const startNewDialog = createEffect(async (newDialog:
INewDialog) => {
  try {
    const response = await instance.post('chat/start-dialog',
newDialog);
    if(response.status === 200) {
      return response;
    }
  }
  catch(error) {
    console.log(error);
  }
})

sample({
  clock: getDialogMessages.doneData,
  filter: response => response.status === 200,
  fn: response => response.data[0].dialogId,
  target: updatedIsReadMessagesInActiveDialog
})

sample({
  clock: updatedIsReadMessagesInActiveDialog.doneData,

```

8

```

    filter: response => response.status <= 217,
    fn: response => response.data,
    target: setMyDialogs
  })
  sample({
    clock: getMyDialogs.doneData,
    filter: response => response.status === 200,
    fn: response => response.data,
    target: setMyDialogs
  })

import { createEffect, createEvent, createStore, sample } from
"effector";
import { ICreateMeeting, IMeeting, ISplitedMeetings, MeetingComment
} from "../interfaces/meetings";
import { instance } from "./store";
import { ChangeEvent } from "react";

export const setCurrentMeetings = createEvent<ISplitedMeetings>();
export const currentMeetings = createStore<ISplitedMeetings>({
  previousMeetings: [],
  furtherMeetings: []
}).on(setCurrentMeetings, (_, meetings) => {
  return meetings;
});

export const setUserPlaces = createEvent<string[]>();
export const currentUserPlaces$ =
createStore<string[]>([]).on(setUserPlaces, (_, meetings) => {
  return meetings;
});

export const setSelectedMeeting = createEvent<IMeeting>();
export const addCommentToSelectedMeeting =
createEvent<MeetingComment>();
export const selectedMeeting = createStore<IMeeting>(null)
.on(setSelectedMeeting, (_, meeting) => {
  return meeting;
})
selectedMeeting.on(addCommentToSelectedMeeting, (meeting, comment)
=> {
  return {
    ...meeting,
    comments: [...meeting.comments, comment]
  }
})

export const getAllMeetings = createEffect(async () => {
  const meetings = await instance.get('meetings/get-all-
meetings');
  return meetings;
});

```

9

```

    ))

export const uploadFileToMediaMeeting = createEffect(async (params: {
  event: ChangeEvent<HTMLInputElement>,
  meetingId: string
}) => {
  const file = params.event.target.files[0];
  const formData = new FormData();
  formData.append('uploadedFile', file);
  formData.append('meetingId', params.meetingId)

  const response = await
instance.post<IMeeting>('/meetings/upload-media', formData);
  return response;
});

export const handleCreateNewMeeting = createEffect(async (meeting:
ICreateMeeting) => {
  const response = await
instance.post<IMeeting>('/meetings/create-meeting', meeting);
  return response;
});

export const handleSendCommentIntoMeeting = createEffect(async
(params: {
  meetingId: string, text: string
}) => {
  const response = await instance.post<IMeeting>('/meetings/send-
comment', {
    meetingId: params.meetingId,
    text: params.text
  });
  return response;
});

export const handleRegInMeeting = createEffect(async (meetingId:
string) => {
  const response = await instance.put<IMeeting>(`/meetings/reg-
in-meeting/${meetingId}`);
  return response;
});

export const getUserPlaces = createEffect(async (data: { userId:
string }) => {
  const response = await
instance.get(`/meetings/places/${data.userId}`);
  return response;
})

```

0

```

sample({
  clock: getUserPlaces.doneData,
  filter: (res) => res.status <= 217,
  fn: (res) => res.data,
  target: setUserPlaces
});

sample({
  clock: uploadFileToMediaMeeting.doneData,
  filter: (res) => res.status <= 217,
  fn: (res) => res.data,
  target: setSelectedMeeting
});

sample({
  clock: handleCreateNewMeeting.doneData,
  filter: (res) => res.status <= 217,
  fn: (res) => res,
  target: getAllMeetings
});

sample({
  clock: handleRegInMeeting.doneData,
  filter: (res) => res.status <= 217,
  fn: (res) => res.data,
  target: setSelectedMeeting
});

sample({
  clock: handleSendCommentIntoMeeting.doneData,
  filter: (res) => res.status <= 217,
  fn: (res) => res.data.comments[res.data.comments.length - 1],
  target: addCommentToSelectedMeeting
});

sample({
  clock: getAllMeetings.doneData,
  filter: (res) => res.status === 200,
  fn: (res) => {
    let splittedMeetings = { previousMeetings: [],
furtherMeetings: [] };
    res.data.map(el => {
      if (new Date(el.date).getTime() > new
Date(Date.now()).getTime()) {
        splittedMeetings = {
          ...splittedMeetings,
furtherMeetings:
[...splittedMeetings.furtherMeetings, el]
        }
      } else {
        splittedMeetings = {

```

1

```
        ...splitedMeetings,
previousMeetings:
[...splitedMeetings.previousMeetings, el]
    }
    });
    return splitedMeetings;
},
target: setCurrentMeetings
})
```

Приложение Г
(рекомендуемое)
Результаты тестирования мессенджера

Таблица Г.1 – Результаты тестирования мессенджера

Описание	Ожидаемый результат	Фактический результат	Итог	Критерий
Пользователь перешел в несуществующий диалог	Пользователь видит сообщение о том, что диалог с выбранным пользователем еще не начат (менее 500мс)	Пользователь видит сообщение о том, что диалог с выбранным пользователем еще не начат (110мс)	Тест пройден. Рисунок Д.1	3
Пользователь перешел в существующий диалог	Пользователь видит историю переписки с выбранным пользователем (менее 500мс)	Пользователь видит историю переписки с выбранным пользователем (160мс)	Тест пройден. Рисунок Д.2	3
Пользователь (отправитель) отправил текстовое сообщение корректной длины	Пользователь (отправитель) увидел свое сообщение в истории переписки (менее 500мс)	Пользователь (отправитель) увидел свое сообщение в истории переписки (155мс)	Тест пройден. Рисунок Д.4	3
Пользователь (отправитель) отправил текстовое сообщение не корректной длины (пустое)	Пользователь (отправитель) увидел сообщение об ошибке «Некорректная длина сообщения» (менее 500мс)	Пользователь (отправитель) увидел сообщение об ошибке «Некорректная длина сообщения» (170мс)	Тест пройден. Рисунок Д.3	3
Пользователь (отправитель) отправил аудиосообщение	Пользователь (отправитель) увидел аудиосообщение в истории переписки и может его прослушать (менее 500мс)	Пользователь (отправитель) увидел аудиосообщение в истории переписки и может его прослушать (160мс)	Тест пройден. Рисунок Д.5	3
Пользователь (отправитель) отправил сообщение в виде изображения	Пользователь (отправитель) увидел отправленное изображение в истории переписки и может посмотреть его (менее 500мс)	Пользователь (отправитель) увидел отправленное изображение в истории переписки и может посмотреть его (140мс)	Тест пройден. Рисунок Д.6	3
Пользователь (отправитель) отправил сообщение в виде видео файла	Пользователь (отправитель) увидел отправленное видео и может посмотреть его (менее 500мс)	Пользователь (отправитель) увидел отправленное видео и может посмотреть его (170мс)	Тест пройден. Рисунок Д.7	3
Пользователь (отправитель) отправил сообщение в виде любого файла кроме видео и изображения	Пользователь (отправитель) увидел отправленный файл в истории переписки и может скачать его (менее 500мс)	Пользователь (отправитель) увидел отправленный файл в истории переписки и может скачать его (120мс)	Тест пройден. Рисунок Д.8	3
Пользователь (отправитель) нажал на значок видеосообщения	Пользователь (отправитель) увидел окно с вопросом хочет ли он на самом деле записать	Пользователь (отправитель) увидел окно с вопросом хочет ли он записать	Тест пройден. Рисунок Д.9	3

	видеосообщение (менее 500мс)	видеосообщение (170мс)		
--	------------------------------	------------------------	--	--

Продолжение таблицы Г.4

Пользователь (отправитель) нажал кнопку «закрыть» для отмены отправки видеосообщения	Окно с вопросом закрылось, пользователь опять увидел историю переписки (менее 500мс)	Окно с вопросом закрылось, пользователь опять увидел историю переписки (170мс)	Тест пройден. Рисунок Д.10	3
Пользователь (отправитель) отправил видеосообщение	Пользователь (отправитель) увидел видеосообщение в истории переписки и может посмотреть его (менее 500мс)	Пользователь (отправитель) увидел видеосообщение в истории переписки и может посмотреть его (200мс)	Тест пройден. Рисунок Д.11	3
Пользователь (получатель) зашел в переписку с отправителем	Пользователь (получатель) получил все отправленные ему сообщения (менее 500мс)	Пользователь (получатель) получил все отправленные ему сообщения (170мс)	Тест пройден. Рисунок Д.12	3
Корректное отображение десктопного интерфейса	Все элементы на странице имеют корректное позиционирование	Все элементы на странице имеют корректное позиционирование	Тест пройден. Рисунок Д.13	1
Корректное отображение мобильного интерфейса	Все элементы имеют корректное позиционирование	Все элементы на странице имеют корректное позиционирование	Тест пройден. Рисунок Д.14	1

Приложение Д
(рекомендуемое)
Скриншоты тестирования мессенджера

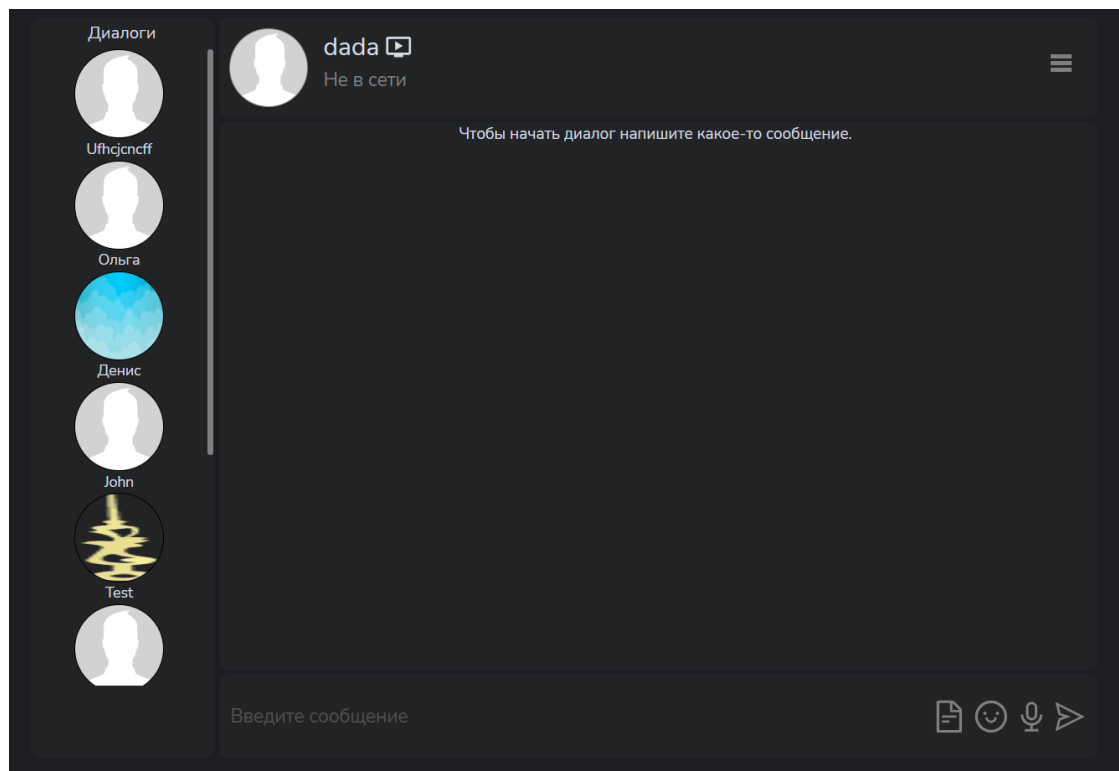


Рисунок Д.1 – Результат выполнения теста «Пользователь перешел в несуществующий диалог»

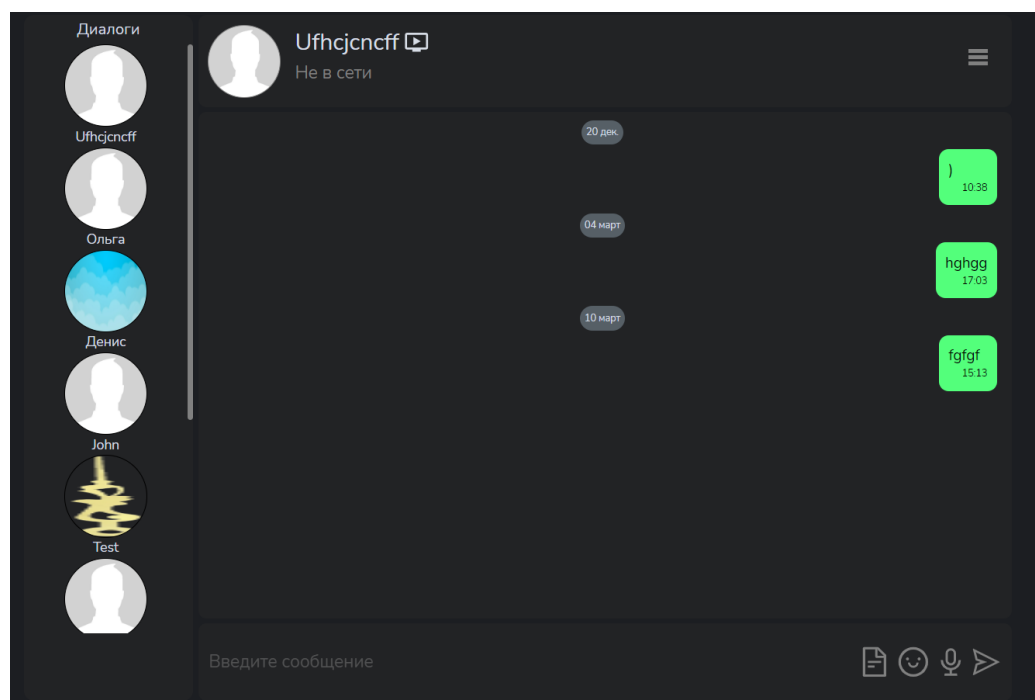


Рисунок Д.2 – Результат выполнения теста «Пользователь перешел в существующий диалог»

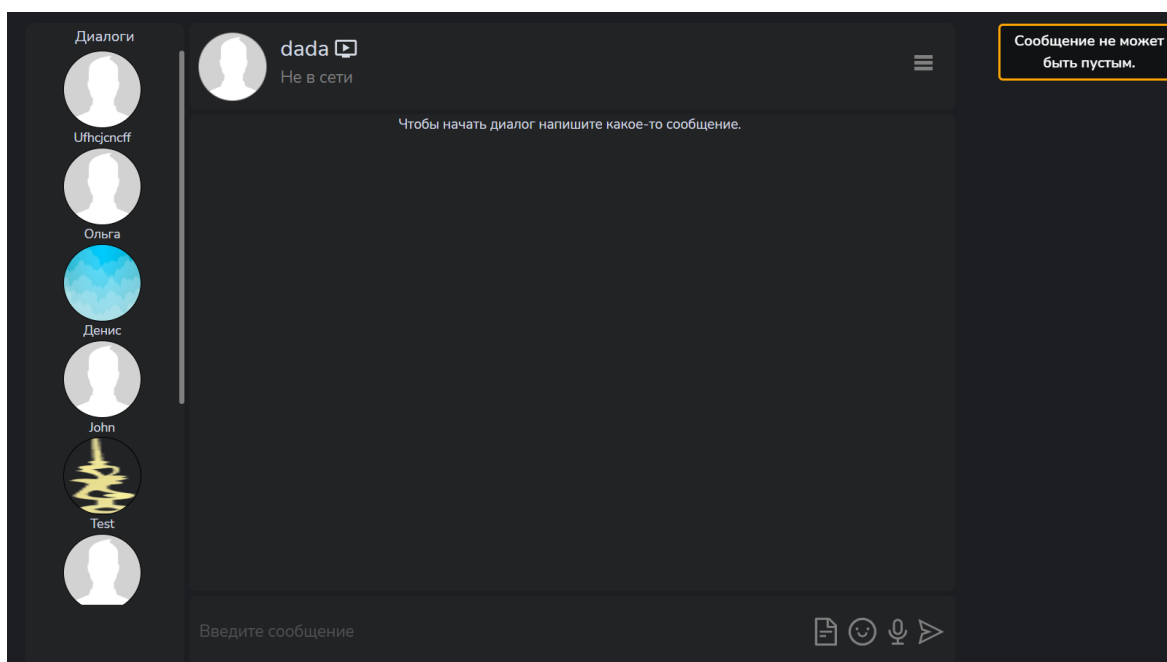


Рисунок Д.3 – Результат выполнения теста «Пользователь (отправитель) отправил текстовое сообщение некорректной длины»

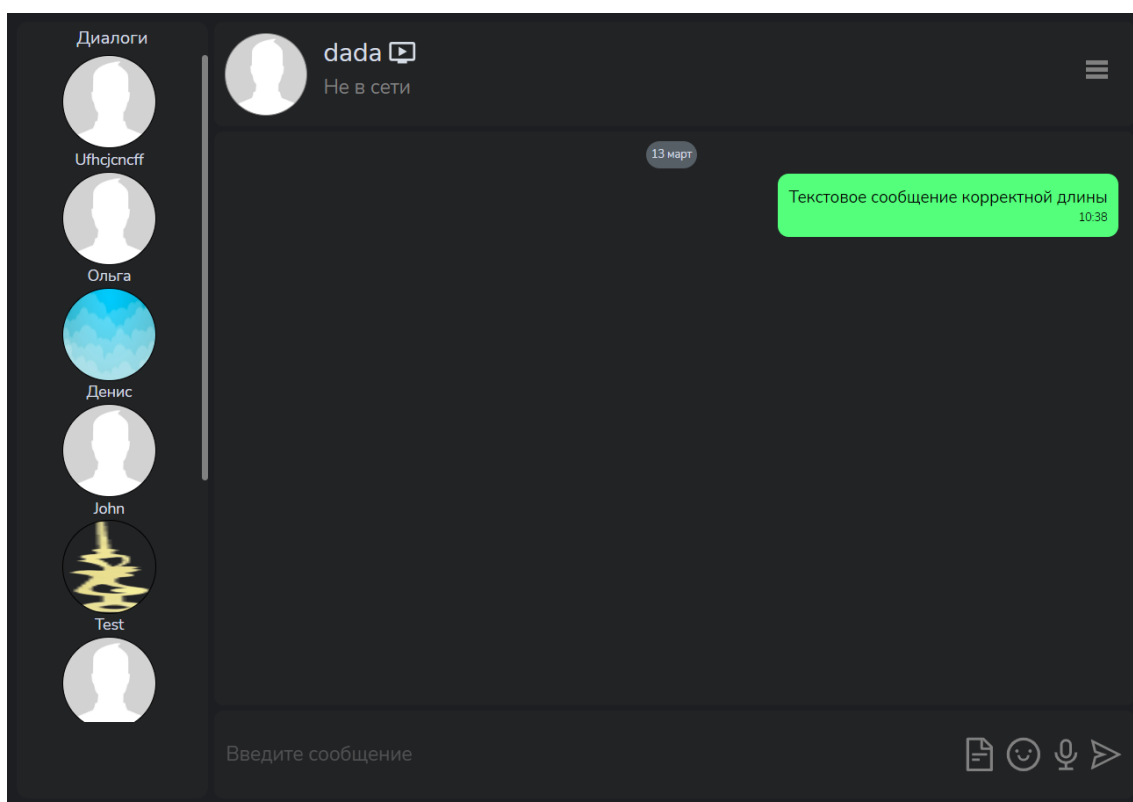


Рисунок Д.4 – Результат выполнения теста «Пользователь (отправитель) отправил текстовое сообщение корректной длины»

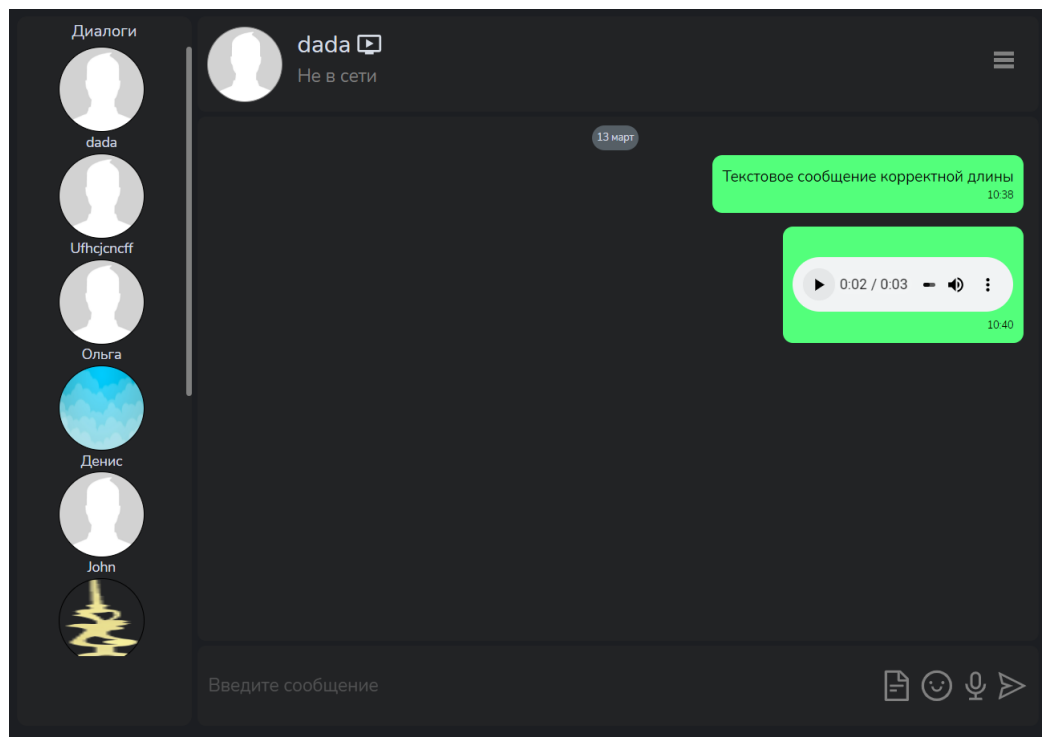


Рисунок Д.5 – Результат выполнения теста «Пользователь (отправитель) отправил аудиосообщение»

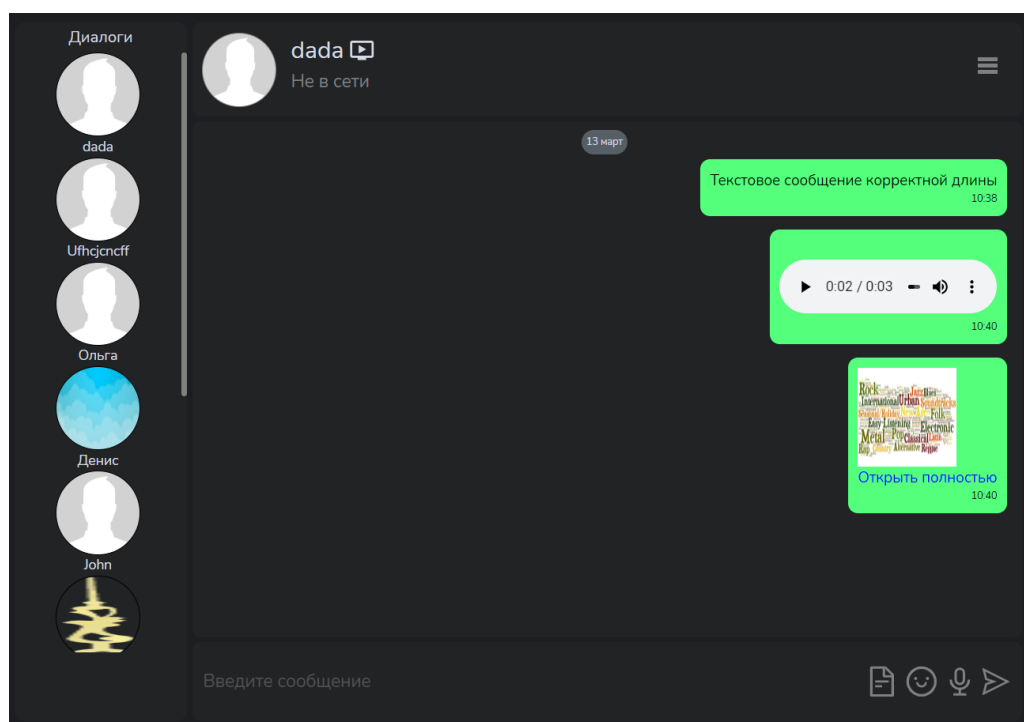


Рисунок Д.6 – Результат выполнения теста «Пользователь (отправитель) отправил сообщение в виде изображения»

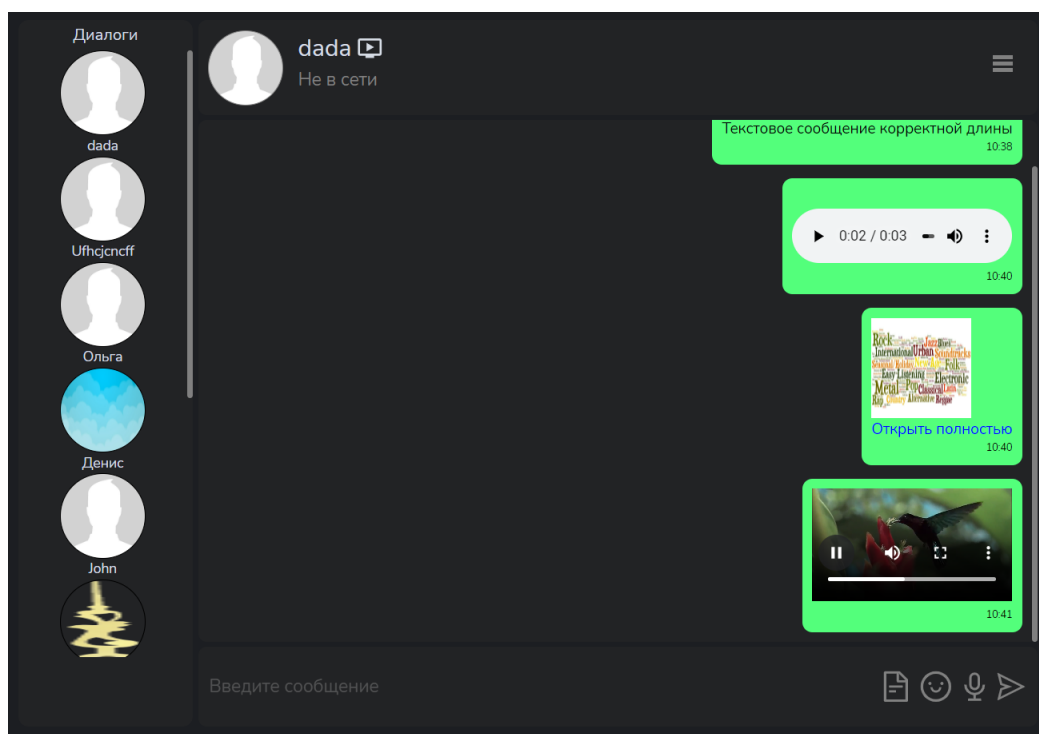


Рисунок Д.7 – Результат выполнения теста «Пользователь (отправитель) отправил сообщение в виде видео файла»

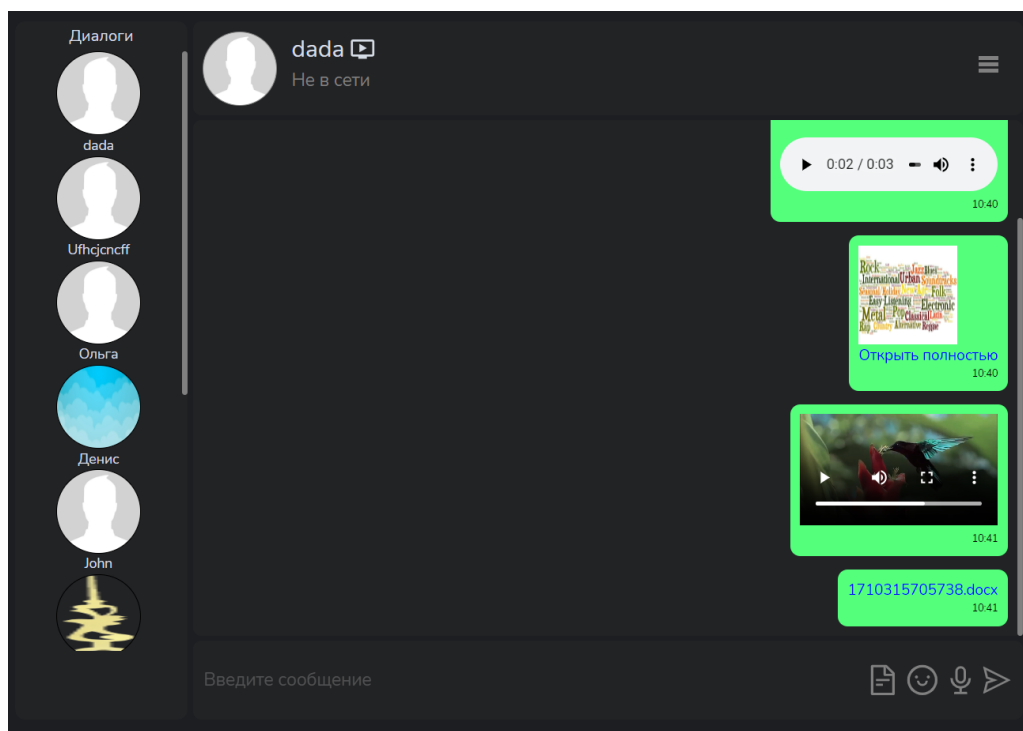


Рисунок Д.8 – Результат выполнения теста «Пользователь (отправитель) отправил сообщение в виде любого файла кроме видео и изображения»

0

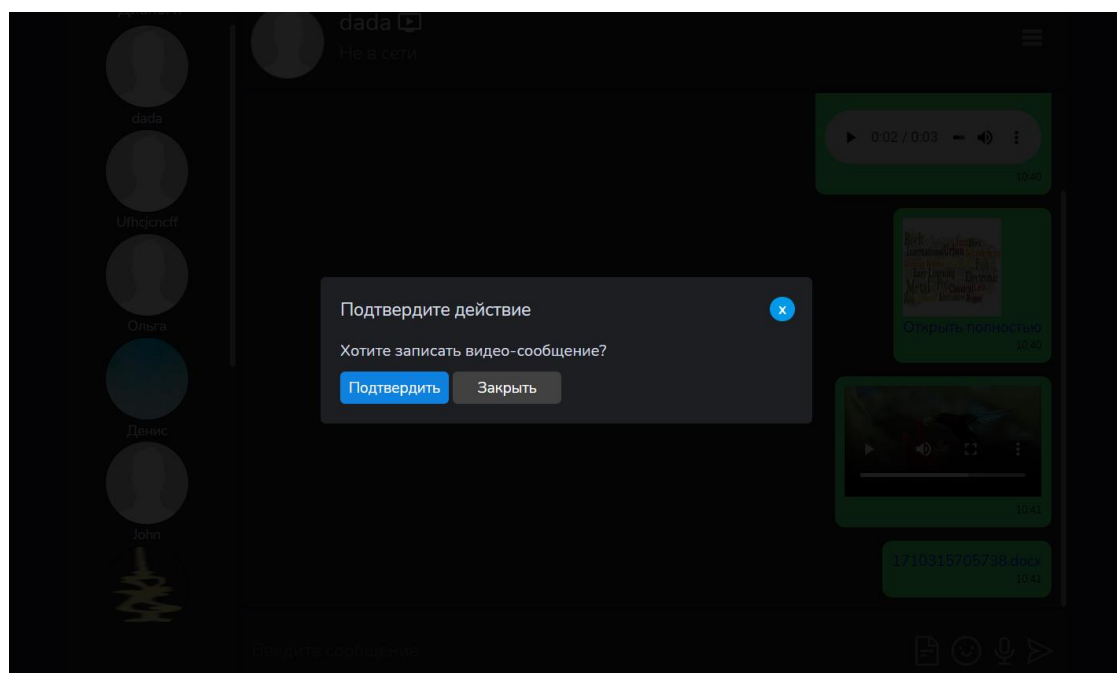


Рисунок Д.9 – Результат выполнения теста «Пользователь (отправитель) нажал на значок видеосообщения»

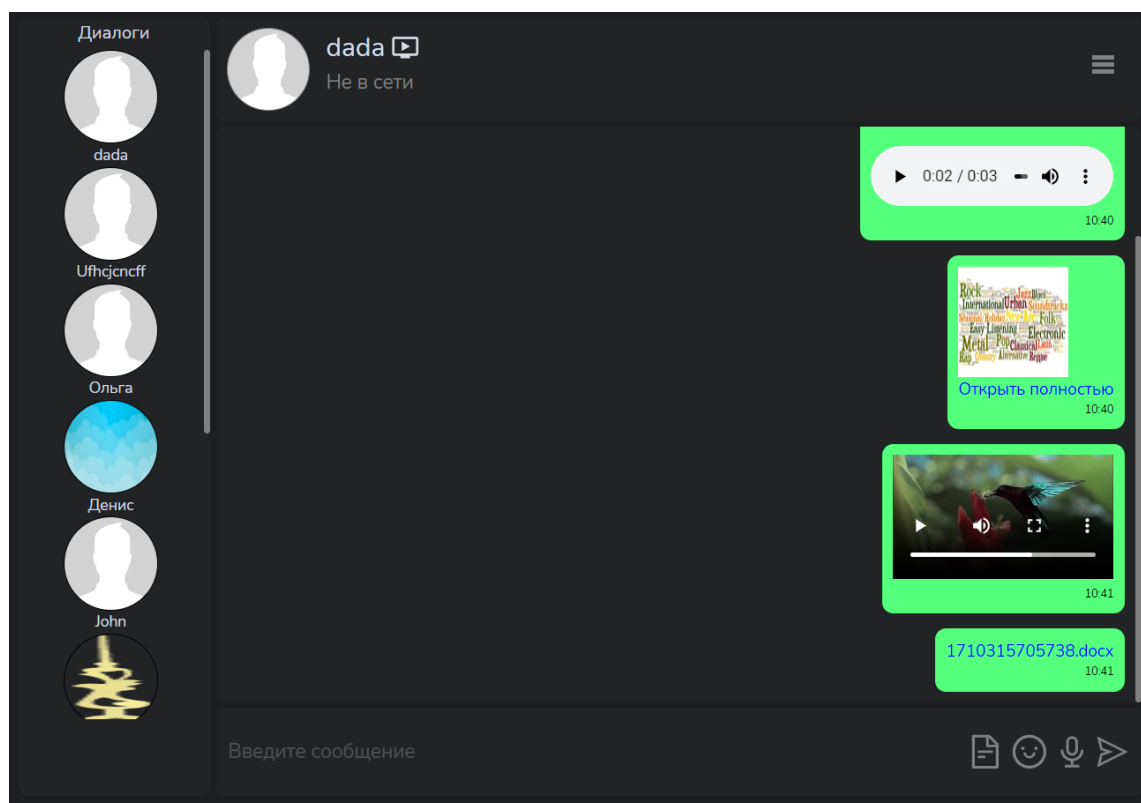


Рисунок Д.10 – Результат выполнения теста «Пользователь (отправитель) нажал кнопку «закреть» для отмены отправки видеосообщения»

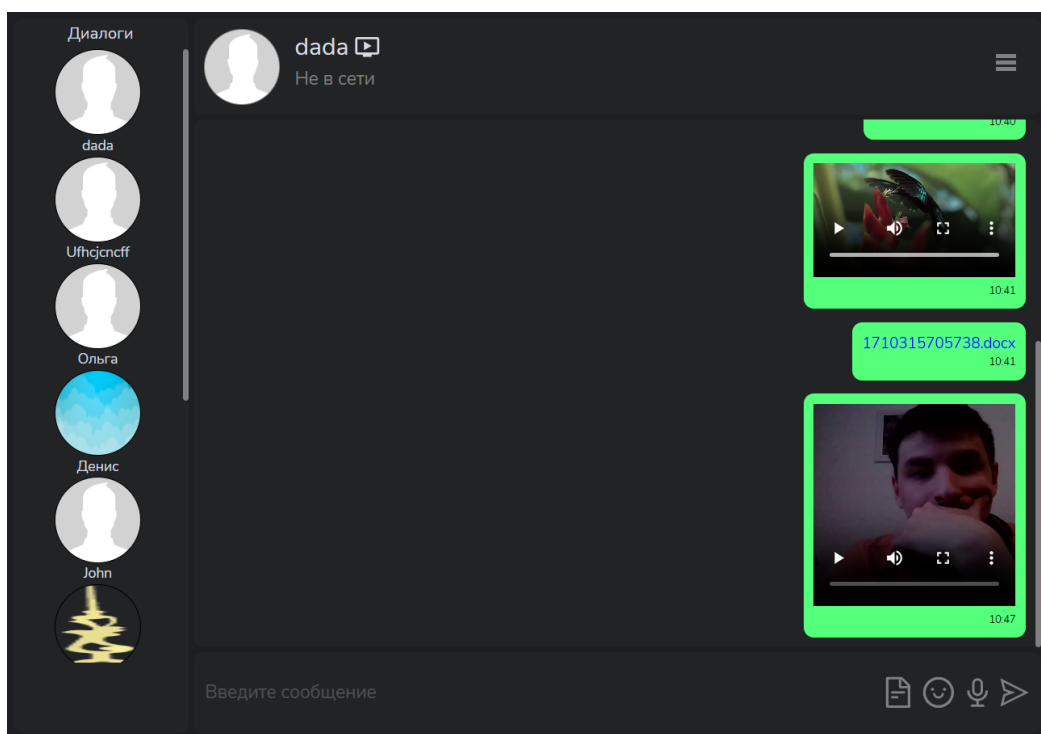


Рисунок Д.11 – Результат выполнения теста «Пользователь (отправитель) отправил видеосообщение»

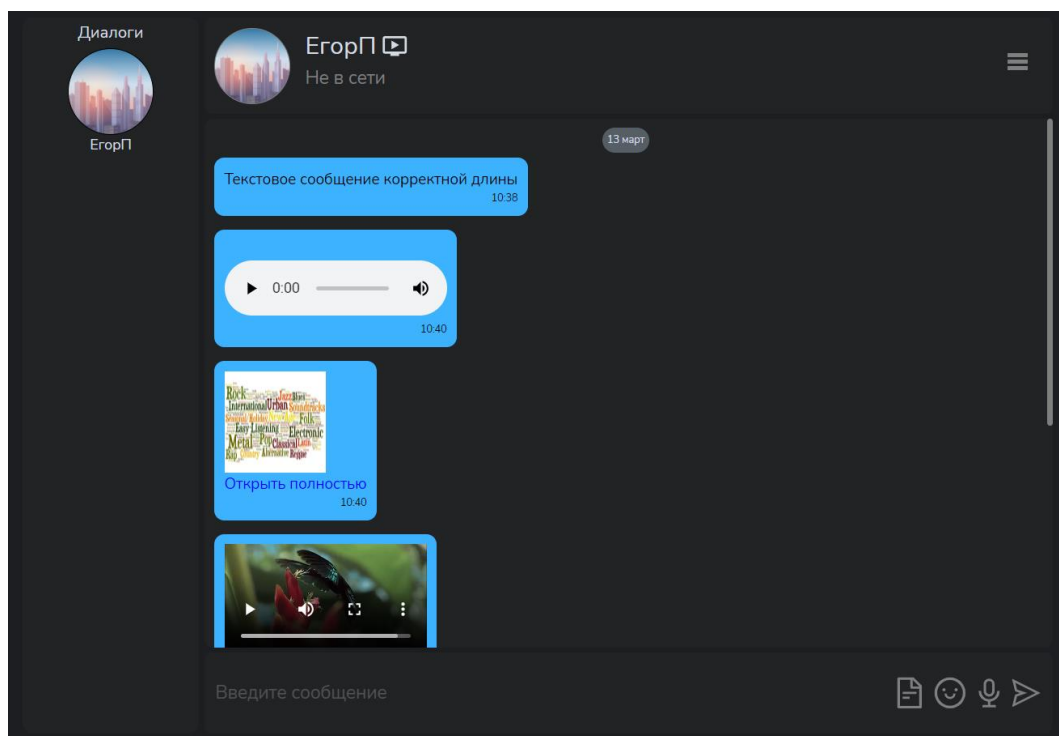


Рисунок Д.12 – Результат выполнения теста «Пользователь (получатель) зашел в переписку с отправителем»

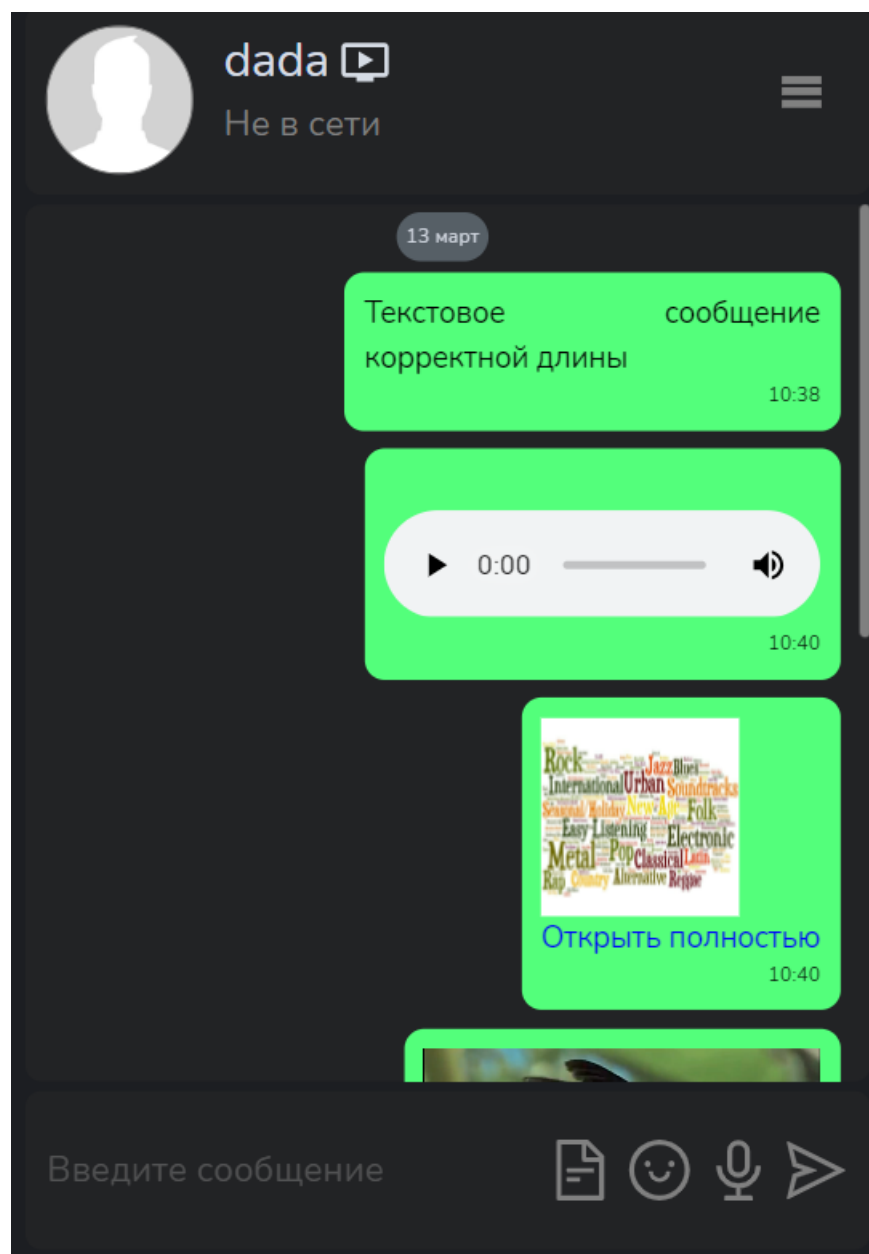


Рисунок Д.13 – Отображение текстового содержимого в мобильном интерфейсе

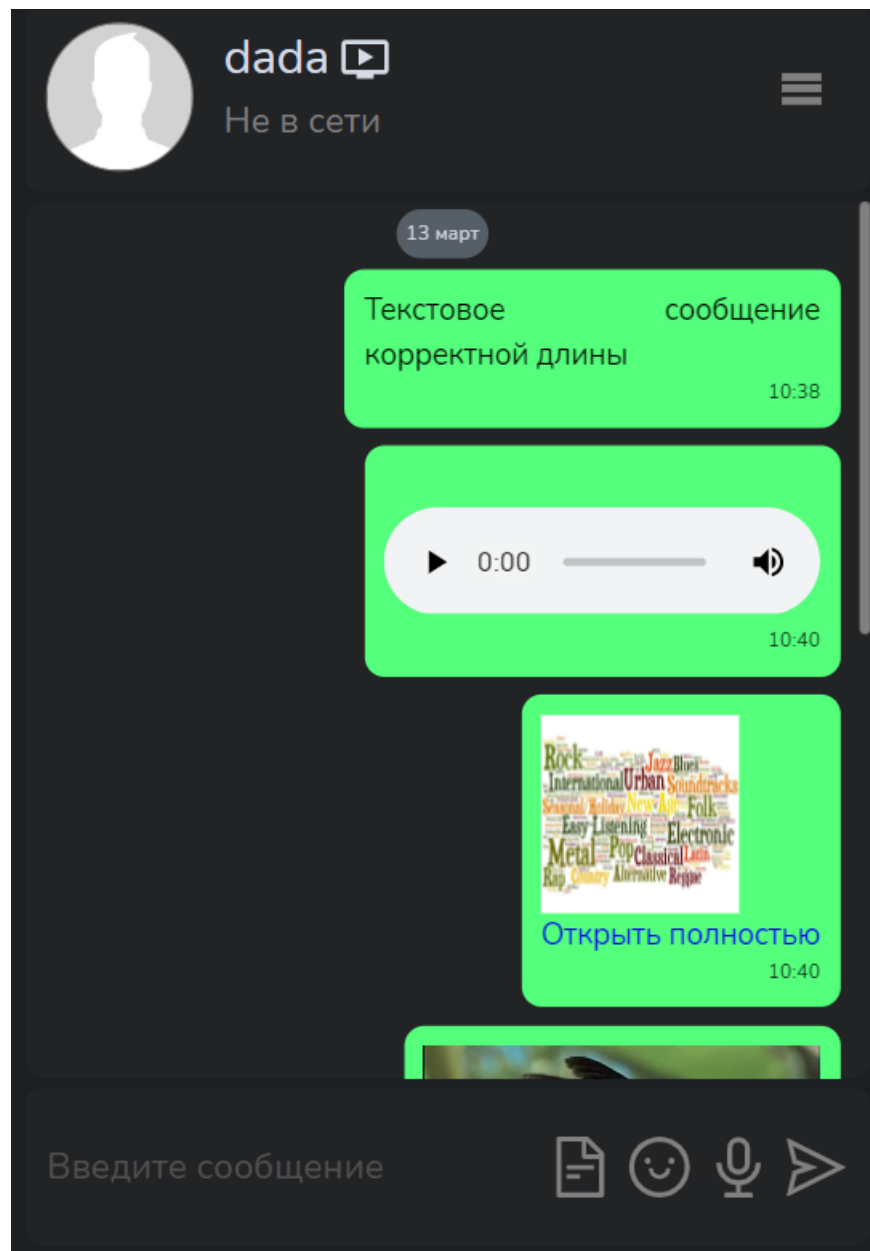


Рисунок Д.14 – Отображение мобильного интерфейса