

**ВСЕРОССИЙСКИЙ КОНКУРС ДОСТИЖЕНИЙ ТАЛАНТЛИВОЙ
МОЛОДЁЖИ
«НАЦИОНАЛЬНОЕ ДОСТОЯНИЕ РОССИИ»**

Секция: информационные технологии

Тема: Разработка веб-мессенджера Utail на Node.JS

Соискатель: Кузнецов Дмитрий Эдуардович
Серебренников Александр Михайлович

Научный руководитель: Заикина Наталья Ивановна

Место выполнения работы: МБОУ СШ №5, Республика Адыгея, Тахтамукайский
р-н, пгт. Яблоновский

2025-2026

АННОТАЦИЯ

В проекте разработан веб-мессенджер Utail на базе Node.JS, предназначенный для обмена сообщениями между преподавателями и учащимися школы с акцентом на безопасность и шифрование данных. Актуальность разработки обусловлена ростом использования мессенджеров в повседневной коммуникации, особенно в условиях удаленного обучения и e-commerce, где требуется быстрый, удобный и защищенный канал связи.

Цель: разработка и реализация веб-мессенджера для обмена сообщениями с использованием современных технологий.

Задачи: анализ литературы и существующих мессенджеров; изучение алгоритмов безопасности; выбор технологий; проектирование серверной и клиентской частей; разработка интерфейса и логики приложения.

Объект исследования: сервис обмена сообщениями в школьной среде. Предмет: технологии шифрования данных.

Методы: анализ рынка мессенджеров (Viber, WhatsApp, Skype, Telegram); выбор Node.JS как основного языка за динамичность и масштабируемость; использование Express.JS, Nginx, Docker Compose, PostgreSQL для серверной части; React.JS, HTML, CSS для клиентской; внедрение WebSocket для реального времени, bcrypt для хэширования паролей, UUIDv4 для идентификаторов.

Результаты: создан мессенджер с функциями регистрации, авторизации, обмена сообщениями, групповыми чатами, историей сообщений; интерфейс в Figma с монохромной пурпурной палитрой, шрифтом Montserrat; система безопасности «User Gate» для предотвращения атак (SQL-инъекции, XSS, CSRF); базы данных с таблицами для аккаунтов, сессий, сообщений; развертывание в Docker.

Практическая значимость: автоматизация общения в школе, сокращение времени на коммуникацию, обеспечение конфиденциальности данных.

Ключевые слова: веб-мессенджер, Node.JS, безопасность, шифрование, React.JS, PostgreSQL.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ.....	5
1.1. Анализ существующих разработок.....	5
1.2. Обзор технологий для создания мессенджеров.....	7
1.3. Выбор языка программирования.....	8
2. ПРАКТИЧЕСКАЯ ЧАСТЬ.....	10
2.1. Разработка дизайна интерфейса мессенджера.....	10
2.2. Дизайн-система мессенджера.....	10
2.3. Структура проекта.....	16
2.4. Инструменты для разработки.....	17
2.5. Разработка backend-приложения.....	19
2.6. Создание таблиц БД.....	25
2.7. Создание основной (серверной) логики backend-приложения.....	27
2.8. Разработка Web-приложения.....	29
ЗАКЛЮЧЕНИЕ.....	39
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ.....	40
Приложение	

ВВЕДЕНИЕ

В современном мире мессенджеры стали неотъемлемой частью повседневной жизни, заменив традиционные формы коммуникации, такие как электронная почта. Популярность мессенджеров в России и за рубежом обусловлена их удобством, скоростью и многофункциональностью.

Особенно в 2024-2025 годах приложения для общения стали огромной частью жизни миллиардов людей. Мы обмениваемся сообщениями, пересылаем файлы, обсуждаем проекты, совершаем звонки — всё это происходит в удобных чатах намного быстрее, чем по электронной почте. Переход на «удаленку», бум e-commerce, рост числа мобильных пользователей — всё это сильно изменило наши коммуникационные привычки. Люди больше не хотят ждать ответа по email или пробиваться через формы обратной связи: им нужен быстрый и безопасный канал связи — и мессенджер идеально решает эту задачу.

Мессенджер - это интегрированная платформа, объединяющая в себе набор инструментов для обеспечения эффективной и удобной коммуникации.

Современный мессенджер — это продукт, объединяющий в себе множество функций: мгновенный обмен сообщениями, передача файлов, групповые чаты, голосовые и видеозвонки, интеграции с другими сервисами. Но главное — это должно быть интуитивное, надежное и защищенное приложение. В условиях, когда пользователи всё острее воспринимают вопросы безопасности и конфиденциальности, важную роль в разработке играет создание таких возможностей, как сквозное шифрование, исчезающие сообщения и двухфакторная авторизация.

На фоне высокой вовлеченности аудитории идея создания мессенджера становится **актуальной** как никогда. Спрос на удобные и функциональные решения для общения продолжает расти по всему миру. Пользователи активно пробуют новые приложения, подбирая для себя наиболее удобные варианты.

Целью проекта является разработка и реализация веб-мессенджера на Node.JS для обмена сообщениями.

Для достижения поставленной цели необходимо решить следующие **задачи**:

- 1) провести анализ теоретической и научно-методической литературы;
- 2) рассмотреть алгоритмы и протоколы для организации обеспечения безопасности личной переписки;
- 3) разработать концепцию системы: изучить возможные средства реализации и выбрать наиболее эффективное;
- 4) проанализировать некоторые существующие мессенджеры; разработать и спроектировать элементы системы;
- 5) систематизировать процессы, разработать алгоритмы веб-сервера(серверной части мессенджера, и клиентского приложения на JavaScript и Node.JS с использованием html и css).

Объект исследования: сервис по обмену сообщения между преподавателем и учащимися школы.

Предмет исследования: технология шифрования данных.

Практическая значимость: состоит в автоматизации системы, которая позволит сократить временные затраты на общение преподавателей и школьников с помощью шифрования данных.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Анализ существующих разработок

На данный момент программы для сетевого общения представлены в широком ассортименте, разнообразны по функциональным возможностям и целям, для которых применяются. При этом на сегодняшний день на рынке программного обеспечения не представлено качественных и удобных программ для сетевого общения, которые бы удовлетворяли требованиям большинства пользователей. В тоже время многие из представленных программных продуктов, не дают качественных возможностей шифрования личной переписки. Рассмотрим наиболее распространенные, популярные продукты, которые предоставляют возможность коллективного общения, в том числе возможность не только текстового, но и визуального (графического) обмена информацией.

Сегодня на рынке мессенджеров серьезная конкуренция. Почти у каждой социальной сети есть свой мессенджер: Facebook Messenger, Вконтакте, Hangouts (в недавнем прошлом GoogleTalk). Так же свои «родные» мессенджеры есть и у мобильных ОС: iMessage для iOS, Messenger для WindowsPhone, BlackBerryMessenger для BlackBerry. Многие слышали про Viber, WhatsApp, Max, Skure и прочие приложения.

Рассмотрим таблицу 1.1 где приведен краткий анализ популярных мессенджеров.

Таблица 1.1 – Анализ мессенджеров по функциональности

Название	Кросс платформенность	Достоинства	Недостатки
Telegram	+	защита от несанкционированного чтения осуществляется благодаря применению протокола связи MTProto; – наличие опции переписки в «секретном чате» — Secret Chat. Благодаря ней вы можете общаться при помощи зашифрованных сообщений; – наличие таймера для автоматического уничтожения сообщений – спустя заданное время (например, 1 час, а не через максимум всего 10 секунд, как в Viber); – возможность передачи файлов большого размера (до 1 Гб); – высочайшая скорость работы, в т.ч. доставки ваших файлов адресату. Нет задержек, характерных другим мессенджер-ам; – синхронизация между пользователями.	– не поддерживает в меню интерфейса русский язык; – отсутствие возможности голосовых звонков.

WhatsApp	+	<p>- низкий расход батареи;</p> <p>- простой интерфейс;</p> <p>- синхронизация с контактами из адресной книги;</p> <p>- поддержка групповых чатов;</p> <p>- работы на таких платфор-мах, как iOS, Android, Windows Phone, BlackBerry, Nokia Symbian и Nokia S40, у WhatsApp есть ком-пьютерная версия клиента, а также web-версия, т.е. можно общаться в обычном браузере;</p> <p>-бесплатные аудио- и видеозвонки, которые можно делать посредством Интернет-соединения (3G или Wi-Fi);</p> <p>-отправка фотографии и видео, можно делиться PDF-файлами (книгами, журнала-ми), слайд-шоу и другими документами.</p> <p>Единственное ограничение - размер файла не должен превышать 100 Мб.</p> <p>-фото, аудио и видео-мате-риалы вначале отправляются на специаль-ный HTTP-сер-вер, а потом уже передаются в уменьшенном варианте конечному получателю. Это позволяет экономить интернет-трафик;</p> <p>- программа с 2016-го года полностью бесплатна. В данный момент у приложения нет никаких встроенных покупок.</p>	<p>- программа не сильна в вопросах шифро-вания;</p> <p>- в WhatsApp нет стикеров, таких как в Telegram или Viber.</p>
Skype	+	<p>-возможность общаться посредством видеозвонка;</p> <p>-нет привязки к какому-то определенному устройству;</p> <p>-программа может устанавливаться везде: от смартфона до «умного» телевизора;</p> <p>-возможность совершать звонки на стационарные телефоны.</p>	<p>- большой объем файла;</p> <p>- необходим хороший сигнал Интернета</p>
Viber	+	<p>-бесплатные звонки и сообщения;</p> <p>-отсутствие рекламы;</p> <p>-стабильная связь даже на 2G -</p>	<p>- отсутствие видеозвонков;</p> <p>- на Symbian</p>

		<p>работает в фоне и «ест» очень мало оперативной памяти;</p> <p>-обилие смайликов, в том числе анимированные;</p> <p>-голосовые сообщения.</p>	<p>доступен лишь обмен текстовыми сообщениями;</p> <p>- без интернета Viber бесполезен.</p>
--	--	---	---

Лидером по итоговой оценке сегодня стал Viber. Это приложение является наиболее универсальным представителем своего сегмента – чаты, аудио- и видеосвязь, конференции. Тут есть даже звонки на обычные телефоны. Чуть позади расположился WhatsApp. Он более популярен, но не позволяет делать вызовы на мобильные номера.

Третье место делят Skype и Telegram. Они также весьма популярны, но имеют определенные недостатки. Skype больше ориентирован на стационарные ПК, а в Telegram нет голосовой связи.

Если подробно рассматривать таблицу по функциональным особенностям, то можно убедиться, что практически все мессенджеры имеют версии для нескольких платформ. Аудиосвязь есть во всех приложениях кроме Telegram, видеозвонки отсутствуют еще и в WhatsApp.

Звонки на обычные телефоны поддерживают лишь 2 приложения – Viber, Skype. Основным идентификатором пользователя практически везде служит номер телефона, хотя в Hangouts, Skype и Facebook Messenger применяются Google Аккаунт, e-mail или учетная запись Facebook соответственно. Для обычных вызовов и переписок хорошо подойдут WhatsApp или Viber, для звонков на стационарные номера – Skype, если нужны только чаты, то можно остановиться на Telegram.

1.2. Обзор технологий для создания мессенджеров

Сегодня одной из наиболее перспективных современных технологий интерактивного общения становится Instant Messanging (англ. «мгновенное сообщение»). Instant Messanging это сетевой сервис обмена сообщениями в «реальном» времени, т.е. с фактически мгновенной доставкой, когда задержка не ощутима. Актуальным вопросом обмена сообщениями является, как для корпоративных компаний, так и для школьников и студентов.

Работа системы, направленной на поддержку обменом сообщений выглядит следующим образом. Основным лицом является пользователь (пользователь системы), для которого определены следующие функциональные требования деятельности, изображенные на рис. 1.



Рисунок 1. Функциональные требования деятельности пользователя

Популярные мессенджеры, как правило, предлагают мобильное и десктопное приложения, а также веб-версию. Для каждой из этих платформ используется определенный набор технологий. При разработке своего мессенджера необходимо заранее определить формат будущего приложения.

Кроме того, выбор языков программирования и фреймворков должен соответствовать следующим требованиям:

- работа с большим объемом данных;
- высокая производительность;
- кроссплатформенность (возможность работы на разных устройствах и ОС);
- надежность и безопасность с целью защиты персональных данных пользователей;
- наличие активного сообщества разработчиков для решения технических задач.

Наиболее востребованным языком для нативной разработки под Android остаётся Java (Kotlin также активно используется и рекомендуется Google как современная альтернатива). Для iOS основным языком является Swift. В кроссплатформенной разработке широко применяются React Native и Flutter.

Классическое веб-приложение, включая мессенджеры, состоит из клиентской и серверной частей. Сервер может быть реализован на Java, C#, Go или Python, а в некоторых случаях для выполнения низкоуровневых задач применяется C++. Для клиентской части стандартом являются JavaScript и TypeScript в сочетании с React, Vue, Angular.



Рисунок 2. Технологии для создания мессенджеров

1.3. Выбор языка программирования

Языки программирования имеют ограниченный запас слов (операторов) и строгие правила написания, а правила грамматики и семантики, как и для любого формального языка, явно однозначно и четко сформулированы.

Языки программирования классифицируют по разным признакам, например:

1. По уровню абстракции. Бывают низкоуровневые (близки к машинному коду) и высокоуровневые (более близки к естественному языку и абстрактным концепциям). Примеры:

- низкоуровневые — C, C++, Assembly, обеспечивают прямой контроль над аппаратным обеспечением;
- высокоуровневые — Python, Java, Ruby, упрощают разработку, позволяя программисту сосредоточиться на логике приложения, а не на деталях реализации.

2. По назначению. Бывают универсальные (подходят для широкого круга задач) и специализированные (разработаны для решения узконаправленных задач в конкретной области). Например:

- SQL — для работы с базами данных;
- HTML — для разметки веб-страниц;
- Lua;
- PHP;
- JavaScript/Type Script (Node.JS).

Языки программирования, которые менее понятны человеку, ориентированные на команды процессора, называют «низкоуровневыми». Обычно ими пользуются для достижения большей скорости или меньшего энергопотребления, при оптимизации команд процессора. Например NASA использует C и C++ для разработки систем управления космическими аппаратами, бортового программного обеспечения и систем реального времени. Ассемблер является низкоуровневым языком программирования. Программа, написанная на нем, представляет последовательность машинных команд. Языки программирования высокого уровня ориентированы на решение сложных задач, а не на максимальную оптимизацию работы оборудования. Более того, эти языки легко масштабируются, однако, нуждаются в соответствующих программах, которые будут переводить данный язык в машинный код в зависимости от архитектуры процессора.

При выборе языка программирования для разработки нашего мессенджера ключевыми критериями стали производительность в операциях ввода и вывода, и возможность удобно масштабироваться. Поэтому мы приняли решение выбрать Node.JS, как основной язык программирования. Характерные особенности этого языка - *динамичность* и *объектно-ориентированность*.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Разработка дизайна интерфейса мессенджера

Прежде чем приступить к разработке дизайна мессенджера, мы проанализировали цели и задачи проекта, определили целевую аудиторию и изучили существующие решения.

Основная цель – создать удобный, понятный и визуально сбалансированный мессенджер, предназначенный для общения внутри школьного сообщества: между учениками, учителями и администрацией.

Главные требования – простота использования, современный внешний вид, безопасность и логичная структура элементов.

Задачи, поставленные перед нами на этапе проектирования:

- разработать интуитивно понятный всем категориям пользователей интерфейс, который не требует обучения;
- обеспечить легкую навигацию и логическую взаимосвязь между экранами;
- создать современный визуальный стиль, немного молодёжный, но не перегруженный;
- реализовать интерфейс, одинаково удобный на всех устройствах.

Для разработки интерфейса был выбран Figma, так как это современный онлайн-инструмент, который позволяет быстро создавать интерфейсы, использовать компоненты, прототипировать переходы между экранами и проводить тестирование.

Работа в Figma удобна тем, что все изменения сохраняются автоматически, проект можно демонстрировать прямо в браузере, а компоненты и стили помогают выдерживать единый визуальный язык во всех макетах.

Этапы разработки дизайна мессенджера:

1. *Анализ.* Были изучены популярные мессенджеры – Telegram и Discord, чтобы понять, какие интерфейсные решения считаются самыми удобными. От Telegram позаимствовали минимализм, аккуратность и простую логику чатов, а от Discord – структуру серверов и групп, которая удобно ложится на школьные классы и предметы.

2. *Создание вайрфреймов.* На этом этапе была определена архитектура экранов и последовательность действий пользователя (вход в систему – выбор школы – просмотр чатов – общение – переход к профилю). Были созданы наброски будущих страниц без оформления – только композиция и логика навигации.

3. *Создание UI-дизайна.* Подобрали цветовую палитру, шрифты, иконки и построили визуальную систему: карточки, кнопки, поля ввода и состояния элементов. Все экраны объединены единой логикой сетки и пропорций.

4. *Прототипирование и тестирование.* На основе макетов создали кликабельный прототип в Figma. Протестировали интерфейс на одноклассниках и получили обратную связь. Доработали отдельные элементы – поля ввода, кнопки, контраст текста. После правок интерфейс стал более сбалансированным.

2.2. Дизайн-система мессенджера

Основные принципы, которым мы следовали:

- Единый визуальный стиль. Все элементы – одинаковая логика отступов, закругления, шрифты и иконки. Это даёт ощущение цельности.
- Контраст и читаемость. Текст и важные кнопки должны быть читаемы и заметны.
- Информативность контента. Изображения/аватары помогают быстрее ориентироваться,

превью прикрепленных файлов – сразу видно, что отправлено.

- Повторяемость визуала. Один набор цветов и шрифтов во всех экранах для узнаваемости.
- Адаптивность. Макеты рассчитаны на мобильную и десктопную версии.
- Доступность. Контраст, размеры шрифтов и кликабельные зоны – учитывал минимальные требования удобства.

Цветовая палитра. Для дизайна выбрали монохромную палитру бордово-фиолетового (пурпурного) оттенка, который сочетает в себе энергию красного и стойкость синего цветов. Оттенок ассоциируется со знанием и мудростью, стимулирует творческое мышление и воображение. Палитра благодаря «мягкой», приглушённой тональности удобна для длительного восприятия – пользователи могут проводить в приложении значительное время без дискомфорта для глаз.

С точки зрения функциональности и удобства использования, пурпурный оттенок обеспечивает хорошую контрастность для текста белого цвета. Интерфейсные элементы (кнопки, иконки) выполнены в монохромной гамме (от белого до тёмно-пурпурного), но за счёт разной насыщенности оттенка привлекают внимание и помогают выделять действия, не перегружая интерфейс и создавая единый стиль. Фирменный градиентный фон – выглядит нейтрально, что важно для удобства чтения.

Основной цветовой акцент – на аватарах и статусах. Аватары круглые, статусы цветные: зелёный – онлайн, красный – не беспокоить, жёлтый – не в сети. Это привычные ассоциации, которые люди интуитивно понимают.

Также, выбранная палитра сразу же выделяет Utail на фоне подавляющего большинства мессенджеров, использующих классическую синюю палитру, делая его визуально уникальным и запоминающимся.

Шрифты. Шрифт Montserrat – нейтральный, современный и хорошо читаемый даже при малом размере. Заголовки выделены чуть крупнее и полужирным, основной текст – обычным начертанием, а служебные подписи и время сообщений – имеют более мелкий размер и приглушённый цвет.

Иконки, формы и декоративные элементы. Все иконки – контурные, из одного набора, без лишней графики. Кнопки и карточки имеют одинаковое скругление углов (12–16 px), что создаёт единую визуальную пластичность интерфейса. Мягкие, округлые декоративные элементы на главном экране и подложки под текст выглядят спокойно и доброжелательно.

Композиция. Во всех экранах выдержана вертикальная иерархия – элементы расположены сверху вниз, по логике взаимодействия. Интерфейс «дышит»: достаточно отступов, ровный визуальный ритм, мягкие углы и минимальные тени. Основной принцип – *минимум визуального шума, максимум ясности.*

Экран регистрации и выбор школы – экраны первичной настройки.

Путь:

1. Пользователь открывает приложение впервые.
2. Видит минималистичный экран с приветствием и поле для регистрации с возможностью выбора типа пользователя (ученик/учитель).
3. После ввода данных появляется дополнительное поле с поисковой строкой для выбора школы.
4. Пользователь вводит название школы или выбирает из списка предложенных.
5. После выбора – система автоматически перенаправляет его обратно на экран регистрации для завершения действия.

Нет длинных форм и непонятных шагов. Пользователь сразу видит, что мессенджер «школьный», а не общий. На экране минимум элементов: фирменный паттерн на фоне, поле ввода данных и акценты – кнопки действия. Дизайн лаконичный: монохромная фирменная палитра, базовый пурпурный оттенок, затемнённые акценты, округлые формы. Такой подход создаёт ощущение лёгкости и технологичности.

Экран авторизации (рис.3) – повторный вход по почте или через соцсети (Яндекс, ВК, Google).

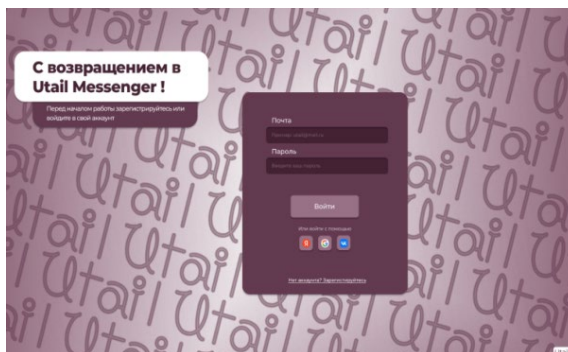


Рисунок 3. Экран авторизации

Этот экран служит входной точкой в систему. Его основная цель – дать уже зарегистрированному пользователю осуществить быстрый и безопасный вход, без лишних полей.

Путь:

1. Пользователь видит форму с 2 вкладками: вход по e-mail и паролю/быстрый вход через соцсети (VK, Google, Яндекс – поддержка популярных способов входа особенно важна для подростков).

2. При успешном входе – перенаправление на главный экран (чаты).

3. Если пользователь новый – есть подсказка «Регистрация» или «Создать аккаунт».

Здесь заложен принцип минимального барьера: меньше шагов – быстрее вход – выше шанс, что пользователь не закроет приложение.

Фон – тот же, что и на экране регистрации/выбора школы – фирменный паттерн, чтобы сохранить визуальное единство. Основная кнопка выделена оттенком светлее контрастного фона – акцентирует внимание, но благодаря монохромной палитре не выбивается из общего стиля.

Всё расположено по вертикальному центру, с равными отступами – визуальный баланс и фокус на середине. Элементы не отвлекают внимание. Поля ввода визуально крупные, с закруглёнными углами и лёгкой тенью – «дружелюбный» стиль, соответствующий школьной теме. Кнопки соцсетей используют брендовые цвета платформ, что повышает узнаваемость. Структура выстроена по вертикали, что помогает удерживать фокус. Дизайн вдохновлён Telegram и Discord: минимум лишнего, только логика действия.

Главный экран (список чатов) (рис.4) – центральный интерфейс мессенджера, от которого пользователь переходит ко всем остальным функциям. Он объединяет:

- список всех доступных чатов (личных, групповых, предметных);
- доступ к профилю, настройкам и поиску;
- навигацию по разделам приложения (например, чаты, пользователи «В сети», уведомления).

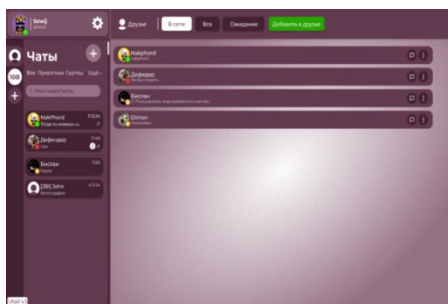


Рисунок 4. Главный экран

Именно здесь пользователь проводит наибольшее количество времени, поэтому дизайн и структура этого экрана максимально продуман под ежедневное использование.

Путь:

1. После авторизации пользователь попадает на главный экран сразу.
2. Он видит список чатов с превью последних сообщений, временем и индикатором непрочитанных сообщений и может:

- выбрать конкретный чат и открыть диалог;
- перейти к созданию нового чата;
- воспользоваться поиском по контактам или сообщениям;
- открыть свой профиль.

Основная цель – обеспечить быструю ориентацию и минимум кликов до нужного действия. Композиция построена по принципу «один взгляд – одно действие»: пользователь сразу видит активные чаты и может перейти в любой из них. Каждая карточка чата – это аватар, имя, последнее сообщение и время.

Активная вкладка подсвечена контрастным фирменным цветом для визуального акцента. Между чатами – чёткие разделители для визуального ритма. Фон в фирменном градиенте. Паттерн взят из Telegram и Discord: лаконичные карточки, максимум информации в одну строку, минимум визуального шума.

Экран создания чата/группы (рис.5) – это важный интерфейс, который расширяет функциональность мессенджера и позволяет пользователям взаимодействовать более гибко.

Пользователь может:

- создать личный чат (с одним собеседником);
- создать групповой чат (например, для класса или проекта);
- задать название, аватар и список участников.

Это один из ключевых процессов – он напрямую влияет на то, насколько легко пользователю формировать коммуникации внутри школьной системы.



Рисунок 5. Экран создания чата/группы

Путь:

1. Пользователь нажимает на кнопку «+» на главном экране.
2. Появляется экран создания чата, где:
 - вводится название (например, «11Б – Алгебра»);
 - выбирается тип чата (личный / групповой);
 - выбираются участники из списка контактов или поиска;
 - окно добавить иконку/аватар чата.

После подтверждения – чат появляется в общем списке на главном экране.

Основная идея: всё происходит на одном экране, интуитивно, без лишних переходов. Композиция вертикальная: сверху – выбор участников, ниже – название и тип чата. Цветовая и шрифтовая согласованность делает переход от главного экрана плавным и естественным. Поля ввода и чекбоксы выполнены в едином стиле, активные элементы выделены затемнением. Референсы – Telegram (простота формы) и Discord (логика выбора участников).

Экран чата (рис.6) – это центр экосистемы мессенджера, основное рабочее пространство пользователя. Здесь он общается с другими участниками, обменивается сообщениями, файлами, реагирует на контент.

Для школьного мессенджера этот экран особенно важен, потому что он должен:

- быть максимально простым (чтобы школьники быстро освоили);
- быть визуально чистым (чтобы не утомлять глаза при длительном общении);
- выглядеть современно, но не как игровой Discord – ближе к Telegram по стилю.

Пользователь должен видеть, где он находится (какой чат), понимать, кто пишет, и не отвлекаться на визуальный шум.



Рисунок 6. Экран чата

Путь:

1. Пользователь выбирает чат на главном экране.
2. Переходит в экран «Чат».
3. Перед пользователем появляется:
 - верхняя панель (название, участники, кнопки действий);
 - история сообщений;
 - поле ввода.
 - Может написать сообщение, прикрепить файл или отправить эмодзи.
4. По клику на аватар – открывается окно «Информация о пользователе».
5. Возврат на главный экран – по стрелке.

Цветовая палитра – продолжение общей концепции: интерфейс в монохромной гамме фирменной палитры. Композиционно весь экран строится по правилу вертикального ритма: каждый блок (верх, чат, низ) имеет свою визуальную «массу». Сообщения выровнены по левому краю. Отступы сверху и снизу уравновешены, создавая ощущение симметрии. Высота «пузырьков» сообщений зависит от объёма текста – интерфейс адаптивен.

Нейтральный фон в фирменном градиенте и аккуратная типографика позволяют глазу отдыхать. Поле ввода фиксировано внизу, имеет округлые углы и лёгкую тень. Сообщения отделены отступами, между днями – тонкие разделители. Такая структура позволяет легко читать длинные диалоги и не перегружает экран. Такое распределение предотвращает визуальную усталость – важно для учащихся, которые проводят в чате много времени.

Экран с информацией о пользователе (рис.7). Этот экран появляется при клике на карточку участника. Его цель – дать краткое, но визуально структурированное представление о пользователе: кто он, его статус, роль в системе и возможные действия. Поэтому необходимо показать максимум информации на одном экране, не перегружая пользователя текстом.

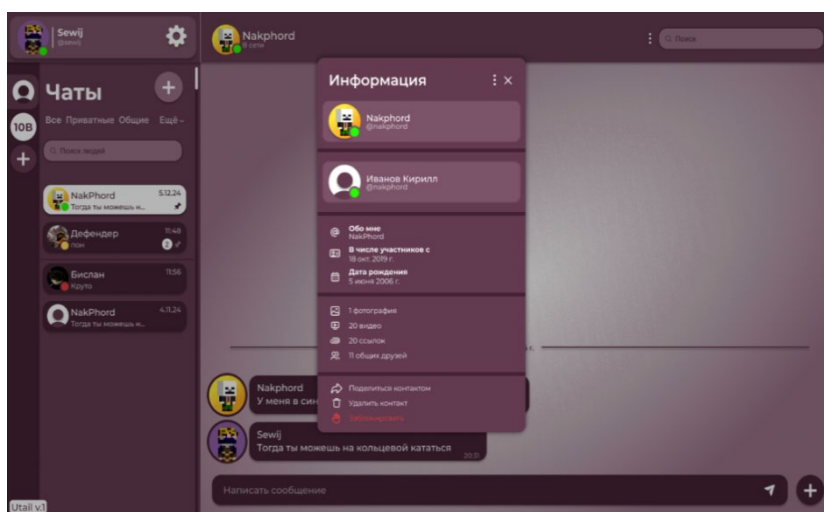


Рисунок 7. Экран с информацией о пользователе

Путь:

- пользователь открывает модальное окно профиля, где крупно отображается аватар, имя, статус активности и основная информация.

Вся система мессенджера выдержана в фирменной монохромной палитре, и этот экран – не исключение. Он использует контраст тёмного фона, «мягких» подложек по текст и акцента в виде аватара для выделения личности. Потому что этот экран – не эмоция, а информация. Его цель – не привлечь внимание, а удерживать фокус на человеке.

Экран строится по вертикальной иерархии сверху вниз – всё подчинено логике чтения и восприятия: Такая структура основана на принципе, когда глаз пользователя сначала видит лицо, потом имя, потом действие.

Используется один шрифт – Montserrat – современный, ровный, читаемый, но разная толщина, что делает интерфейс визуально «чистым». Модальное окно центрируется и имеет мягкие закругления углов – это создаёт ощущение «карточки профиля», а не диалогового окна. Тень под окном добавляет глубину. Композиция построена по центру, элементы выровнены и визуально сбалансированы.

2.3. Структура проекта

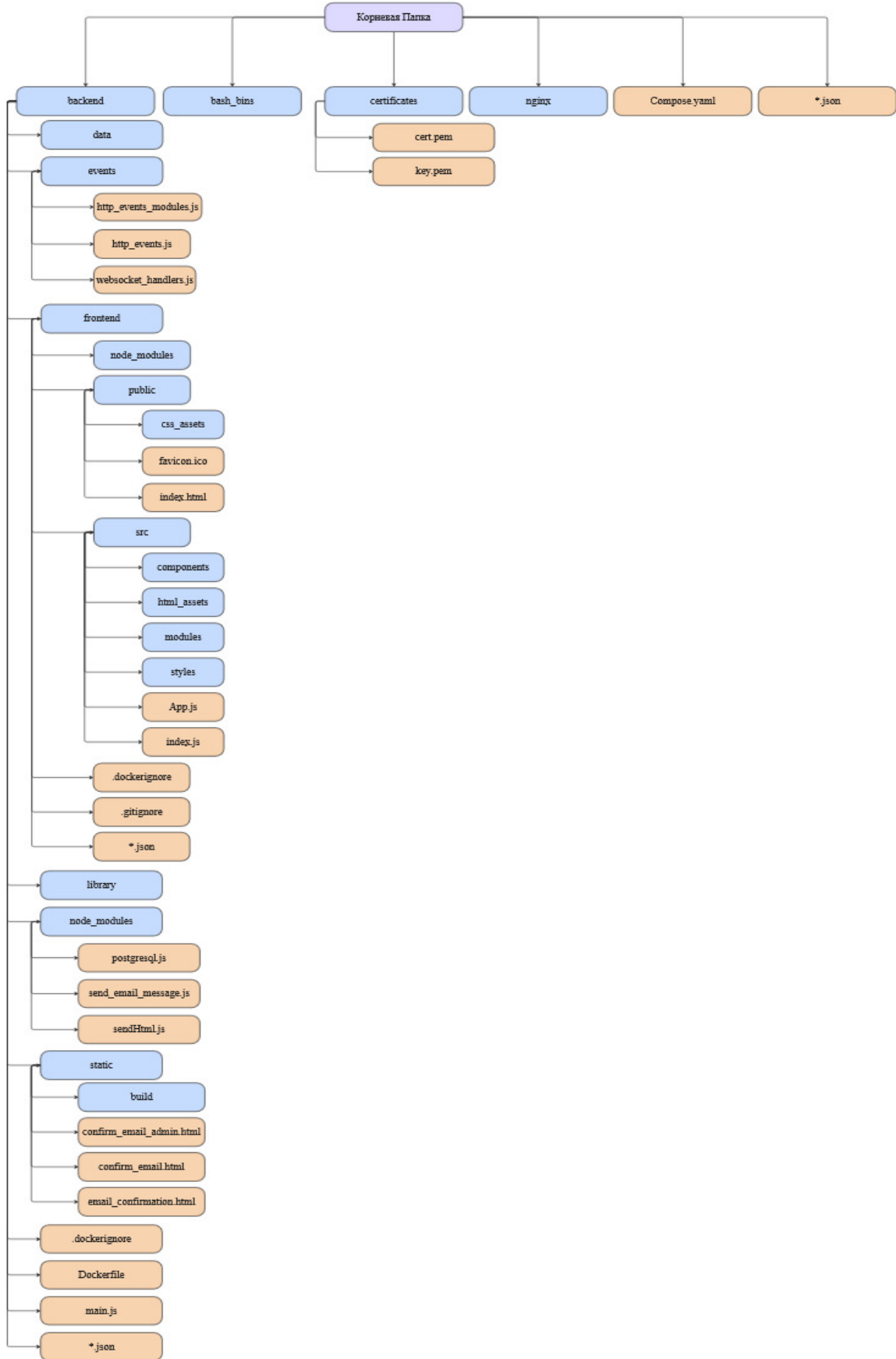


Рисунок 8. Структура проекта

Структура проекта представлена на рисунке 8. В корневой папке проекта находится файл `Compose.yaml`, предназначенный для запуска системы проекта. В директории `./backend` расположены основные файлы и библиотеки веб-сервера. Сборка веб-приложения осуществляется из папки `./backend/frontend`. Данные базы данных хранятся в `./backend/data`. Бизнес-логика проекта, включая такие функции, как `send_message`, `accept_friend_request` и `decline_friend_request`, реализована в модулях, размещённых в `./backend/events`.

Статические файлы, которые сервер обслуживает и предоставляет как часть веб-приложения, находятся в директории `./backend/static`. К ним относятся:

- `./backend/static/build` — собранное React.js приложение;
- `./backend/static/confirm_email_admin.html` — страница для подтверждения почты, отправляемая пользователям, подходящим под требования для создания аккаунта администратора;
- `./backend/static/confirm_email.html` — страница, отправляемая обычным пользователям для подтверждения регистрации; для создания аккаунта пользователь должен активировать его, нажав на кнопку в этом письме.

Сертификаты проекта находятся в `./certificates`. Скрипты для установки и настройки Docker (файлы формата `.sh`) расположены в `./bash_bins`. Для сборки образов используются файлы `Dockerfile`, которые присутствуют как в `./backend`, так и в `./backend/frontend`.

В директории `./backend/frontend` находится клиентская часть проекта, реализованная в виде React-приложения. Её структура организована следующим образом:

- `node_modules` — папка с внешними зависимостями проекта, устанавливаемыми через `npm`.
- `public` — каталог со статическими ресурсами, которые напрямую копируются при сборке:
- `css_assets` — содержит дополнительные CSS-файлы и стилевые ресурсы.
- `favicon.ico` — иконка сайта.
- `index.html` — основной HTML-файл, являющийся точкой входа для React-приложения.
- `src` — каталог с исходным кодом приложения:
- `components` — содержит React-компоненты, из которых строится интерфейс.
- `html_assets` — включает вспомогательные HTML-файлы и шаблоны.
- `modules` — располагает модулями с бизнес-логикой, утилитами и вспомогательными функциями.
- `styles` — хранит файлы стилей (CSS, Stylus и др.).
- `App.js` — корневой компонент приложения.
- `index.js` — точка входа, которая рендерит приложение в DOM.

В процессе сборки с помощью скриптов, определённых в `package.json`, это приложение компилируется в оптимизированные статические файлы, которые помещаются в папку `./backend/static/build` для последующего обслуживания веб-сервером.

2.4. Инструменты для разработки

В качестве инструмента для развертывания мы выбрали *Docker*. Это позволит обеспечить кросс-платформенность приложения. *Docker* предоставляет формальный язык, используемый для описания инструкций по сборке и запуску приложения. Мы создали файл `Compose.yaml` в корневой папке (рис. 8), в котором будут определены все необходимые параметры для запуска, включая настройки контейнеров, проверку доступности базы данных и порядок запуска различных компонентов приложения.

Для начала определили *контейнеры* с настройками, объединили их через внутреннюю сеть *Docker*, выдали нужные права доступа, определили последовательность запуска (рис. 9-13).

```

services:
  > Run Service
  backend:
    build:
      context: ./backend
    args:
      REACT_APP_BACKEND_URL : "https://192.168.1.194"
      REACT_APP_BACKEND_WS_URL : "wss://192.168.1.194"
      BACKEND_PORT : 3000 #устанавливаем внутренний сервер, на каком порту будет запускаться, а Nginx будет перенаправлять запросы с порта 443(https) на 3000(http)
    ports:
      - "3001:3001"
    networks:
      - my_network
    volumes:
      - ./backend/data/global/admin_config.json:/app/data/global/admin_config.json
    depends_on:
      db:
        condition: service_healthy
    user: root

```

Рисунок 9. Backend контейнер

```

nginx:
  image: nginx:latest
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx/conf.d:/etc/nginx/conf.d
    - ./certificates:/etc/nginx/certs
  depends_on:
    - backend # Nginx запускается после бэкенда
  networks:
    - my_network
  user: root

```

Рисунок 10. Nginx контейнер

```

db:
  container_name: postgresql_container
  image: postgres:15
  restart: always
  environment:
    POSTGRES_USER: root
    POSTGRES_PASSWORD: root
    POSTGRES_DB: utail_db
  # ports:
  #   - "5432:5432"
  volumes:
    - ./backend/data/global/postgresql_data:/var/lib/postgresql/data # Том для хранения данных PostgreSQL
  networks:
    - my_network
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U root -d utail_db"]
    interval: 10s
    timeout: 5s
    retries: 5
  user: root

```

Рисунок 11. Db контейнер, СУБД

```

pgadmin:
  container_name: pgsqladmin4_container
  image: dpape/pgadmin4:latest
  restart: always
  environment:
    PGADMIN_DEFAULT_EMAIL: dimonkuznecov2007@gmail.com
    PGADMIN_DEFAULT_PASSWORD: root
  ports:
    - "5050:80"
  volumes:
    - ./backend/data/global/pgadmin_data:/var/lib/pgadmin
  networks:
    - my_network
  depends_on:
    - db
  user: root

```

Рисунок 12. Pgadmin контейнер

```

networks:
  my_network:
    driver: bridge

```

Рисунок 13. Общая сеть контейнеров

Ключевое слово *context* указывает на папку, где располагаются файлы Dockerfile, которые будут выполняться при создании или запуске контейнера.

Контейнеры и их функции:

- **Backend.** Является веб-сервером сайта Utail, отвечающий на запросы клиентов. Он выполняет основные функции: доступность сайта, перенаправление запросов, защита данных, аутентификация пользователей, коммуникация, отображение сайта у пользователей.

- **Nginx.** Также немаловажный контейнер, который отвечает за шифрование протоколов.

- **Db.** Контейнер, который запускает СУБД PostgreSQL, к которой можно подключиться внутри сети для управления.

- **Pgadmin.** Является инструментом администрирования СУБД PostgreSQL, например: для отладки данных пользователей, решения проблем, связанных с ошибками данных, внесение корректив.

При построении *контейнера, Docker* выполняет все инструкции разово, при этом создавая так называемый *образ*, который может выполнить только сам *Docker*. Во время построения *Docker* делит команды на разные части: *build-time* команды и *runtime* команды. *Build-time* команды выполняются один раз во время построения, такие команды не будут запускаться вновь при повторном запуске образа. *Runtime* команды будут выполняться каждый раз, при запуске контейнера. Dockerfile контейнера Backend представлен на рисунке 14.

```
backend > Dockerfile > ...
1 FROM node:23-alpine AS frontend
2 WORKDIR /app/frontend
3
4 COPY ./frontend/package*.json ./
5 COPY ./frontend .
6
7 RUN npm install
8
9 ARG REACT_APP_BACKEND_URL
10 ENV REACT_APP_BACKEND_URL=$REACT_APP_BACKEND_URL
11 ARG REACT_APP_BACKEND_WS_URL
12 ENV REACT_APP_BACKEND_WS_URL=$REACT_APP_BACKEND_WS_URL
13
14 ENV LANG=C.UTF-8
15 ENV LC_ALL=C.UTF-8
16 RUN npm run build
17
18 FROM node:23-alpine AS backend
19 WORKDIR /app
20
21 COPY package*.json ./
22 RUN npm install
23
24 COPY . .
25
26 ARG REACT_APP_BACKEND_URL
27 ENV REACT_APP_BACKEND_URL=$REACT_APP_BACKEND_URL
28 ARG REACT_APP_BACKEND_WS_URL
29 ENV REACT_APP_BACKEND_WS_URL=$REACT_APP_BACKEND_WS_URL
30
31 ARG BACKEND_WS_PORT
32 ENV BACKEND_WS_PORT=$BACKEND_WS_PORT
33 ARG BACKEND_PORT
34 ENV BACKEND_PORT=$BACKEND_PORT
35
36 COPY --from=frontend /app/frontend/build ./static/build
37
38 EXPOSE $BACKEND_PORT
39
40 ENV LANG=C.UTF-8
41 ENV LC_ALL=C.UTF-8
42
43 CMD ["npm", "run", "start"]
```

Рисунок 14. Dockerfile

Такой *образ* будет запускать процесс сборки React.JS проекта в *frontend*, копируя его в образ *backend*. Самая последняя инструкция в этом файле запускает проект под runtime-командой, что означает – каждый запуск контейнера запустит последнюю команду.

2.5. Создание backend-приложения

На начальном этапе мы объявляем переменные (рис.16) и импортируем различные фреймворки – Express.JS, вспомогательный модуль Cors, Path: для управления файловой системой, cookie-parser – вспомогательный модуль Express, cookie – модуль для работы с необработанными cookie файлами (рис.17), node:url – модуль для работы с конечными точками.

```

const express = require('express');
const cors = require('cors');
const path = require('path')
const cookieParser = require('cookie-parser');
const cookie = require('cookie');
const url = require('node:url');
const WebSocket = require('ws');

```

Рисунок 16. Объявление переменных

```

utail_account_uid=a0e83ac9-8997-4e4c-b0e6-abdcaaf79e8f-dd2bfaff-
4d59-49c4-bdd9-2362ed6a66f6-4d1af79e-6eda-412b-99aa-
833890af8a3b; utail_session_uid=561c49e5-f8e5-4f08-a159-
ab0c253828eb-733d6cf8-05f3-433a-a132-958a165a2408-83c7c17f-4bda-
4a4c-87de-ad1a886bd7a6

```

Рисунок 17. Пример необработанных cookie файлов, представленные в виде строки

Далее создаём веб-сокеты веб-сервера и express, которые будут поддерживать процесс рукопожатия (рис. 18).

```

const websocket_server = new WebSocket.Server({ noServer: true });
const app = express();
const {EventsGet, EventsPost, createTables} = require('./events/http_events.js');
const {PostFunctions} = require('./events/http_events_modules.js')

const port = process.env.BACKEND_PORT;

app.use(cors({origin: process.env.REACT_APP_BACKEND_URL }));
app.use(express.json());
app.use(cookieParser());
app.use(express.static(path.resolve('static', 'build')));

```

Рисунок 18. Создание веб-сервера веб-сокета и express

Затем установим настройки CORS, для предотвращения атак: XSS, CSRF. Далее мы будем «парсить» (с англ. Parse - конечные точки). Вот как выглядит парсинг у Utail на Рис. 19.1 и Рис 19.2: код находит событие, связанное со своим методом, а затем вызывает это событие с нужными аргументами для дальнейшей работы с ответом на запрос клиента. Сами же события определены в переменных EventsGet, EventsPost.

```

// http:// && https://
app.post('/api/post', async (req, res) => {
  const action = req.body.action;

  console.log(`[backend-server] получил POST запрос: ${action}`);

  if (EventsPost.eventNames().includes(action)) {
    try {
      EventsPost.emit(action, req, res);
    } catch (error) {
      console.error(`Ошибка при эмитинге действия ${action}:`, error);
      res.status(500).json({ error: "Internal server error" });
    }
  } else {
    console.log(`Событие ${action} не зарегистрировано.`);
    res.status(400).json({ error: `Событие "${action}" не зарегистрировано.` });
  }
});

app.get('/api/get', async (req, res) => {
  const scheme = req.protocol || 'http';
  const host = req.get('host');
  const path = req.originalUrl;

  const baseUrl = `${scheme}://${host}${path}`;
  const parsedURL = url.parse(baseUrl, true);

  const action = parsedURL.query.action
  console.log(`[backend-server] получил GET запрос: ${action}`);
  if (EventsGet.eventNames().includes(action)) {
    try {
      EventsGet.emit(action, req, res, parsedURL);
    } catch (error) {
      console.error(`Ошибка при эмитинге действия ${action}:`, error);
      res.status(500).json({ error: "Internal server error" });
    }
  } else {
    console.log(`Событие ${action} не зарегистрировано.`);
    res.status(400).json({ error: `Событие "${action}" не зарегистрировано.` });
  }
});

app.get('/*name', (req, res) => {
  res.sendFile(path.resolve('static', 'build', 'index.html'));
});

```

Рисунок 19.1. Определение конечных точек

```

app.post('/api/beacon_drain', express.json(), express.text(), async (req, res) => {
  if (typeof req.body === "string"){
    req.body = JSON.parse(req.body)

    //gate
    const action = req.body.action;
    console.log(`[backend-server] получил POST-beacon запрос: ${action}`);

    if (EventsPost.eventNames().includes(action)) {
      try {
        EventsPost.emit(action, req, res);
      } catch (error) {
        console.error(`Ошибка при эмитинге действия ${action}:`, error);
        res.status(500).json({ error: "Internal server error" });
      }
    } else {
      console.log(`Событие ${action} не зарегистрировано.`);
      res.status(400).json({ error: `Событие "${action}" не зарегистрировано.` });
    }
  } else {
    res.status(500).json({error: 'данный запрос не соответствует требованиям'})
  };
});

const httpServer = app.listen(port, () => {
  createTables();
  console.log(`Backend listening on port: ${port}`);
});

```

Рисунок 19.2. Определение конечных точек и инициализация таблиц СУБД

Инструкция обработки запроса на upgrade протокола HTTP(S) до WebSocket представлена на рисунке 20.

```

// вебсокеты (ws://) - является обратной, непрерывной связью
const {executeHandler} = require('./events/websocket_handlers.js');

httpServer.on('upgrade', (req, socket, head) => {
  websocket_server.handleUpgrade(req, socket, head, (ws) => {
    websocket_server.emit('connection', ws, req)
  })
})

```

Рисунок 20. Инструкция обработки запроса на upgrade протокола HTTP(S) до WebSocket

Теперь, имея обработчики *GET* и *POST* методов, нужно разобраться, как будут передаваться сообщения в реальном времени. Хотя *HTTP* (и методы *GET/POST*) позволяют повторно использовать соединения (например, с помощью *HTTP* Keep-Alive или *HTTP/2*), они не обеспечивают полноценный двусторонний канал реального времени, необходимый для передачи сообщений в реальном времени. Для этой задачи лучше всего подходят веб-сокеты. Веб-сокеты немного сложнее, чем *GET* и *POST*. Этот протокол обеспечивает постоянное соединение между клиентом и сервером, позволяя обеим сторонам отправлять данные в любое время, пока соединение не будет разорвано. Для установки такого соединения, браузер клиента должен осуществить *Upgrade HTTPS* протокола на *WSS*. Таким образом, другой протокол получит все заголовки *HTTP* запроса. Для начала разберёмся: а как работает Upgrade? Браузер отправляет специальный *HTTP*-запрос с «Upgrade» заголовком на сервер, чтобы «попросить» сервер перейти на протокол *WebSocket*.

Если сервер поддерживает *WebSocket*, он отвечает специальным *HTTP*-ответом, и протокол передачи данных меняется на *WebSocket*. Это означает, что для установки *WebSocket*-соединения сначала требуется протокол *HTTP/HTTPS*.

Теперь клиент может осуществить upgrade до *websocket*. Вот как выглядит обработчик соединения на рисунке 21.

```

let connections = {};
websocket_server.on('connection', async (ws_client_stream, incoming_request) => {
  console.log("Новый клиент подключился");

  let utail_session_uid;
  let utail_account_uid;

  // Получение куки файлов
  if (incoming_request.headers.cookie) {
    const cookies = cookie.parse(incoming_request.headers.cookie);
    utail_session_uid = cookies.utail_session_uid;
    utail_account_uid = cookies.utail_account_uid;
  }

  // Проверка наличия куки файлов
  if (!utail_session_uid || !utail_account_uid) {
    console.warn("Клиент подключился без utail_session_uid или utail_account_uid. Закрываем соединение.");
    ws_client_stream.close(1000, "Требуется аутентификация."); //Закрываем с кодом ошибки
    return;
  }

  // Валидация формата куки файлов
  if (typeof utail_session_uid !== 'string' || utail_session_uid.trim() === '' ||
    typeof utail_account_uid !== 'string' || utail_account_uid.trim() === '') {
    console.warn("Обнаружена попытка взлома. Неверный формат utail_session_uid или utail_account_uid. Закрываем соединение.");
    ws_client_stream.close(1008, "Попытка взлома.");
    return;
  }
}

```

Рисунок 21. Обработчик нового соединения WebSocket

Сначала мы проверяем на существование заголовков файлов cookie. Затем мы их проверяем на формат, что с таким файлами действительно сервер готов работать и аутентифицировать пользователя (рис. 22).

```

try {
  //Получение и проверка сессии (ПРИ ПОДКЛЮЧЕНИИ)
  let session_obj = await PostFunctions.getSession_obj({ session_uid: utail_session_uid });
  if (!session_obj) throw new Error("Ошибка формата сессии. Пожалуйста, попробуйте в другой раз.");
  if ((new Date()).getTime() > session_obj.metadata.expires || session_obj.metadata.status !== "auth" || session_obj.metadata.owner_uid !== utail_account_uid)
    throw new Error("Попытка вызвать несанкционированный запрос! Неверная сессия.");

  //Получение и проверка аккаунта (ПРИ ПОДКЛЮЧЕНИИ)
  let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
  if (!account_obj) throw new Error("Ошибка. Такого аккаунта не существует!");
}

```

Рисунок 22. Аутентификация пользователя

Далее мы сохраняем соединения в хэш-таблицу, так как объект данных нельзя сохранять в базу данных (рис 23).

```

try {
  //Получение и проверка сессии (ПРИ ПОДКЛЮЧЕНИИ)
  let session_obj = await PostFunctions.getSession_obj({ session_uid: utail_session_uid });
  if (!session_obj) throw new Error("Ошибка формата сессии. Пожалуйста, попробуйте в другой раз.");
  if ((new Date()).getTime() > session_obj.metadata.expires || session_obj.metadata.status !== "auth" || session_obj.metadata.owner_uid !== utail_account_uid)
    throw new Error("Попытка вызвать несанкционированный запрос! Неверная сессия.");

  //Получение и проверка аккаунта (ПРИ ПОДКЛЮЧЕНИИ)
  let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
  if (!account_obj) throw new Error("Ошибка. Такого аккаунта не существует!");

  if (connections[utail_account_uid] == null || connections[utail_account_uid] == undefined) {
    connections[utail_account_uid] = []
  }
  connections[utail_account_uid].push({
    ws: ws_client_stream,
    session_uid: utail_session_uid,
    account_uid: utail_account_uid,
  });

  let last_index = connections[utail_account_uid].length - 1
  connections[utail_account_uid][last_index].ws.on('message', async message => {
    const buffered = Buffer.from(message)
    const string_message = buffered.toString();
    const json_message = JSON.parse(string_message);
    console.log("Получено сообщение сервером: ${string_message}");

    let result = []
    let handlers_state = {
      clients: {
        first_client: {
          states: []
        },
        second_client: {
          states: []
        }
      }
    }
  }
}

```

Рисунок 23. Сохранение соединения на сервере где обработчик, которого слушает входящие запросы

На рисунке 24 показано, как работает обработчик сообщений, который будет выполнять заданные процедуры. Например: сохранение сообщения в базу данных, сохранение запроса дружбы перед оповещением конечного пользователя. Такой механизм позволяет либо оповестить конечного пользователя, сначала выполнив нужную процедуру, либо оповестить его, не выполняя

никаких процедур.

```
// Проверка наличия нужных процедур в сообщении
if (json_message.settings && json_message.settings.handlers) {
  for (const handlerConfig of json_message.settings.handlers) {
    if (handlerConfig.condition === "before the sending message") {
      try {
        let res = await executeHandler(handlerConfig.handler, {
          utail_session_uid: utail_session_uid,
          utail_account_uid: utail_account_uid,
          body: json_message.body
        }, handlers_state)
        result.push(res);
      } catch (handlerError) {
        console.error(`Ошибка при выполнении обработчика ${handlerConfig.handler}:`, handlerError);
        result.push(-1);
      }
    }
  }
}
```

Рисунок 24. Обработчик запросов, вызов заданных событий

В целях безопасности, чтобы оповестить пользователя, нужно выполнить хотя бы одну процедуру. Кроме того, необходимо, чтобы сервер находил нужную процедуру, в противном же случае, сервер не будет оповещать (рис. 25).

```
else if(json_message.body.endpoint_public_uid != undefined){
  let correct_endpoint_public_uid = (json_message.body.endpoint_public_uid).replace("@", "");
  let resolved_endpoint_uid = (await PostFunctions.getAccount_obj({ public_uid: correct_endpoint_public_uid })).account_uid
  let sender_account = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid })
  if(connections[resolved_endpoint_uid] != null && connections[resolved_endpoint_uid] != undefined){
    connections[resolved_endpoint_uid].forEach(function (connection, index) {
      if (connection.ws.readyState === WebSocket.OPEN) {
        connection.ws.send(JSON.stringify({
          status: {
            type: "успех",
            message: "рукопожатие и передача данных прошли успешно!"
          },
          sender_public_info: {
            public_name: sender_account.metadata.first_name + ' ' + sender_account.metadata.last_name,
            public_uid: sender_account.public_uid,
            account_uid: sender_account.account_uid,
            social_role: sender_account.metadata.social_role,
          },
          handlers_state: handlers_state.clients.second_client,
          message: json_message.body.message,
          initiated: "user",
        }));
      }
    });
  }
}
```

Рисунок 25. Обработчик оповещений пользователя или пользователей

На следующем этапе создаём механизм Ping-Pong, ещё его называют *Heartbeat* (рис. 28).

```
//обеспечиваем heartbeat 0x9 - ping фрейм, а 0xA - pong
else if(parseInt(string_message) === 0x9){
  if(connections[utail_account_uid]){
    if(connections[utail_account_uid][last_index].ws.readyState === WebSocket.OPEN){
      connections[utail_account_uid][last_index].ws.send(0xA);
    }
  }
}
if(parseInt(string_message) !== 0x9 && connections[utail_account_uid]){
  if(connections[utail_account_uid][last_index].ws.readyState === WebSocket.OPEN){
    connections[utail_account_uid][last_index].ws.send(JSON.stringify({
      status: {
        type: "успех",
        message: "рукопожатие и передача данных прошли успешно!"
      },
      handlers_state: handlers_state.clients.first_client,
      message: "handlers_results",
    }));
  }
}
```

Рисунок 26. Heartbeat механизм

Соединение WebSocket нуждается в таком механизме, так как при его отсутствии соединение будет разорвано автоматически. Такой протокол websocket *RFC 6455* был установлен *IETF* «Internet Engineering Task Force». Для осуществления данного протокола нужно знать две вещи: клиент и сервер должны обмениваться байтами периодически для поддержания соединения, и что

у таких байтов есть стандарт. 0xA – байт ответа, 0x9 – байт, на который сервер должен ответить.

Также мы приняли решение добавить обработчик исключений на случай, когда процедура выполнится с ошибкой (рис. 27).

```
connections[utail_account_uid].forEach(function (connection, index) {
  connection.ws.send(JSON.stringify({status: {
    type: "успех",
    message: "клиент успешно подключился!"
  }}));
})
connections[utail_account_uid].forEach(function (connection, index) {
  connection.ws.on('close', async () => {
    console.log(`Клиент ${utail_account_uid} отключился (websocket)`);
    delete connections[utail_account_uid][index];
  });
})
}
catch(error){
  console.error("Ошибка в обработке WebSocket:", error);
  ws_client_stream.send(JSON.stringify({ status: { type: "ошибка", message: error.toString() } })); // Отправляем ошибку клиенту
  ws_client_stream.close(1011, "Внутренняя ошибка сервера."); // Закрываем соединение с кодом ошибки
}
```

Рисунок 27. Оповещение всех подключённых устройств пользователя мессенджера

Далее перейдём к обработчикам конечных точек. Utail представляет собой SPA «Single Page Application». В начале 2010-х годов появилась архитектура Single Page Application (SPA), или одностраничного приложения, которая предложила нечто среднее между этими двумя подходами. Такие сервисы, как VK.com или Facebook.com, являются примерами SPA, где большая часть контента загружается единой загрузкой, а последующие обновления происходят динамически, создавая плавный и непрерывный пользовательский опыт, схожий с мобильным приложением. При посещении SPA сервер всегда отдаёт один и тот же HTML-файл. Дальнейшая работа происходит динамически, с использованием JavaScript, который запрашивает данные у сервера через API «application programming interface» (рис. 19.1). Что такое API? Как работает API в мессенджере Utail?

Важную роль играет функция user_gate (рис. 28).

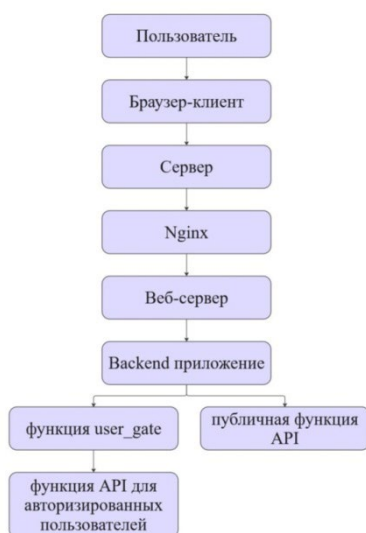


Рисунок 28. Этапы обработки запроса пользователя

При поступлении запроса от браузера-клиента к API (конечная точка /api/post с параметром action: «user_gate»), вызывается функция user_gate, которая выполняет аутентификацию и авторизацию. Если проверка пройдена успешно, user_gate вызывает нужную функцию API, передавая ей необходимые аргументы, и возвращает результат клиенту. Эту функцию можно

сравнить с «охранником и пограничником». Когда браузер-клиент отправляет запрос на /api/post с намерением вызвать определенную функцию API, user_gate перехватывает этот запрос, аутентифицирует пользователя и валидирует (с англ. validate) аргументы. Только после успешной проверки user_gate вызывает нужную функцию API, как бы передавая полномочия проверенному пользователю.

Вот как выглядит структура обработки запроса пользователя: сначала клиент отправляет запрос на сервер, Nginx получает входящий запрос, после чего шифрует его: происходит рукопожатие под TLS «Transport Layer Security». Затем Nginx расшифровывает каждый запрос и передаёт готовый входящий запрос веб-серверу внутри сети – обработчик запросов реагирует на запрос и вызывает либо user_gate, либо сразу функцию API (рис. 28).

Разделение процессов аутентификации и выполнения бизнес-логики в отдельные компоненты (user_gate и непосредственно функции API) является фундаментальным принципом архитектуры безопасности. Это разделение позволяет централизовать критически важную логику безопасности, избегая её дублирования в каждой функции API.

Вот так выглядит user_gate в мессенджере (рис. 29). В данном случае user_gate проверяет «куки», выполняет валидацию и аутентификацию. Затем выдаёт разрешенные функции, согласно роли. Планировалось, что администратор сможет просматривать подозрительные чаты людей.

```
EventsPost.on("user_gate", async (req, res) => {
  try {
    const { utail_session_uid, utail_account_uid } = req.cookies;
    const user_action = req.body.arguments.user_action;

    if (!utail_session_uid || !utail_account_uid) throw new Error("Требуется аутентификация. Попытка вызвать несанкционированный запрос!");
    if (!utail_session_uid || typeof utail_session_uid !== 'string' || utail_session_uid.trim() === '' ||
        !utail_account_uid || typeof utail_account_uid !== 'string' || utail_account_uid.trim() === '') throw new Error("Попытка вызвать несанкционированный запрос! (обнаружена попытка взлома)");
    let session_obj = await PostFunctions.getSession_obj({ session_uid: utail_session_uid });
    if (session_obj === undefined || session_obj === null) throw new Error("Ошибка формата. Пожалуйста, попробуйте в другой раз.");
    if ((new Date()).getTime() > session_obj.metadata.expires || session_obj.metadata.status !== "auth" || session_obj.metadata.owner_uid !== utail_account_uid) throw new Error("Попытка вызвать несанкционированный запрос!");

    let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
    if (account_obj === undefined || account_obj === null) throw new Error("Ошибка. Такого аккаунта не существует!");

    let allowedActions = [
      "get_account_metadata",
      "get_contacts",
      "get_resolved_open_contacts",
      "get_resolved_contacts_requests",
      "add_open_contact",
      "accept_contact_request",
      "decline_contact_request",
      "get_conversation_history",
      "add_message_to_conversation",
      "get_resolved_servers",
      "create_server",
      "get_invite_metadata",
      "create_invite",
      "join_server_with_invite",
      "get_server_members",
      "ban_member",
      "unban_member",
      "get_banned_members",
    ];
    if (account_obj.role === "администратор") { allowedActions.push("admin_action1"); allowedActions.push("admin_action2"); }

    //вызываем тут функцию пользователя
    if (allowedActions.includes(user_action) && ActionEvents.eventNames().includes(user_action)) {
      ActionEvents.emit(user_action, req, res);
    }
    else {
      throw new Error("Попытка вызвать несанкционированный запрос!");
    }
  } catch (error) {
    console.error("Ошибка в user_gate:", error);
    res.status(200).json({ status: { type: "ошибка", message: error.toString() } })
  }
}
```

Рисунок 29. Блок кода user_gate

2.6. Создание таблиц в БД

Для создания базы данных мы выбрали PostgreSQL в качестве СУБД. Определившись с системой управления базами данных (СУБД), создали таблицы с помощью формального языка запросов — SQL. Порядок создания таблиц был следующим:

1. Blacklist(blacklisted_uid; key) - для временной блокировки пользователей до завершения

сервером необходимых вычислений.

2. Accounts(account_uid; key; password; metadata; public_uid; counter_uid)- хранит данные аккаунтов пользователей мессенджера.

3. Sessions(session_uid; key; metadata) - обеспечивает безопасность, каждая сессия действительна 7 дней.

4. Owners_of_messages_uids(account_uid1; account_uid2; messages_uid; metadata; global_uid) - служит ключом для истории сообщений между пользователями или с сервером.

5. Chunks_messages(messages_uid; metadata; messages_limit; chunk_uid; global_uid) - содержит сообщения, организованные по разным историям для корректной загрузки.

6. Servers(server_uid; metadata; public_uid; counter_uid)- предназначена для хранения данных о группах или серверах.

Таким образом, все таблицы были созданы с использованием SQL-запросов, что позволяет системе эффективно управлять данными и обеспечивать работу мессенджера (рис. 30).

```
const database_db = require('../library/postgresql');
async function createTables() {
  try {
    await database_db.query(`
      CREATE TABLE IF NOT EXISTS sessions (
        session_uid TEXT PRIMARY KEY,
        key TEXT,
        metadata JSONB
      );
    `);
    console.log('Таблица sessions была создана или уже существует!');

    await database_db.query(`
      CREATE TABLE IF NOT EXISTS accounts (
        account_uid TEXT PRIMARY KEY,
        key TEXT,
        password TEXT,
        metadata JSONB,
        public_uid TEXT,
        counter_uid SERIAL
      );
    `);
    console.log('Таблица accounts была создана или уже существует!');

    await database_db.query(`
      CREATE TABLE IF NOT EXISTS blacklist (
        blacklisted_uid TEXT PRIMARY KEY,
        key TEXT
      );
    `);
    console.log('Таблица blacklist была создана или уже существует!');

    //сами чанки сообщений
    await database_db.query(`
      CREATE TABLE IF NOT EXISTS chunks_messages (
        messages_uid TEXT NOT NULL,
        metadata JSONB,
        messages_limit INTEGER,
        chunk_uid BIGINT NOT NULL,
        global_uid SERIAL PRIMARY KEY
      );
    `);
    console.log('Таблица chunks_messages была создана или уже существует!');

    //предназначена для того, чтобы сервер понимал, что пользователь один *уже* имеет
    await database_db.query(`
      CREATE TABLE IF NOT EXISTS owners_of_messages_uids (
        account_uid1 TEXT,
        account_uid2 TEXT,
        messages_uid TEXT,
        metadata JSONB,
        global_uid SERIAL PRIMARY KEY
      );
    `);
    console.log('Таблица owners_of_messages_uids была создана или уже существует!');

    await database_db.query(`
      CREATE TABLE IF NOT EXISTS servers (
        server_uid TEXT PRIMARY KEY,
        metadata JSONB,
        public_uid TEXT,
        counter_uid SERIAL
      );
    `);
  }
}
```

Рисунок 30. Создание таблиц базы данных

2.7. Создание основной (серверной) логики backend-приложения

Дальше создали публичные функции, с которыми неавторизованные пользователи могут взаимодействовать (рис. 31- 33).

```
//controller/api/post
EventsPost.on("authenticate_by_session_uid", async (req, res) => {
  const session_uid = req.cookies.utail_session_uid;
  const account_uid = req.cookies.utail_account_uid;
  if(session_uid == '' || session_uid == undefined || account_uid == '' || account_uid == undefined)
  {
    res.status(200).json({ status:
      {
        type: "ошибка",
        message: "Ошибка формата. Пожалуйста попробуйте в другой раз"
      }
    });
    return;
  }
  try{
    let db_table = (await database_db.query("SELECT * FROM sessions WHERE session_uid = '$1', [session_uid])).rows[0];
    if(db_table != undefined){
      let json_table = db_table.metadata;
      if(json_table.status != 'auth')
        throw new Error("Сессия не подходит под требования входа!");
    }
    else throw new Error("Сессия не найдена!");
    let account_table = (await database_db.query("SELECT * FROM accounts WHERE account_uid = '$1", [account_uid])).rows[0];
    if(account_table == undefined)
      throw new Error("Сессия не подходит под требования входа!");
  }
  catch(error){
    res.status(200).json({ status:
      {
        type: "ошибка",
        message: error.toString(),
      }
    });
    return;
  }
  res.status(200).json({ status:
    {
      type: "успех",
      message: "вы успешно вошли в аккаунт!"
    }
  });
});
})
```

Рисунок 31. Публичная функция аутентификации по идентификатору сессии

```
EventsPost.on("sign_in", async (req, res) => {
  const email = req.body.arguments.email;
  const password = req.body.arguments.password;
  const social_role = req.body.arguments.social_role;
  try{
    if(email == '' || password == '' || social_role == '' || email == undefined || password == undefined || social_role == undefined)
      throw new Error("Ошибка формата. Пожалуйста попробуйте в другой раз");
    if(!validator.isEmail(email) == false)
      throw new Error("Ошибка формата. Неправильный e-mail");
    let db_table_account = (await database_db.query("SELECT * FROM accounts WHERE key = '$1", [email])).rows[0];
    const accountCheckResult = await PostFunctions.checkExistingAccount(email);
    if(!accountCheckResult) throw new Error("Ошибка. Такого аккаунта не существует!");
    const blacklistCheckResult = await PostFunctions.checkBlacklist(email);
    if (blacklistCheckResult) throw new Error("Пожалуйста подождите!");
    await database_db.query("INSERT INTO blacklist VALUES ($1, $2)", [uidv4(), email]);
    const match = await bcrypt.compare(password, db_table_account.password);
    if(match){
      let created = new Date();
      let expires = new Date(created.getTime() + (7 * 24 * 60 * 60 * 1000)); //7 дней
      let generated_uid = uidv4() + '-' + uidv4() + '-' + uidv4();
      while((await database_db.query("SELECT * FROM sessions WHERE session_uid = '$1", [generated_uid])).rows[0] != undefined){
        generated_uid = uidv4() + '-' + uidv4() + '-' + uidv4();
      };
      await database_db.query("INSERT INTO sessions VALUES ($1, $2, $3)", [generated_uid, email, {
        "status": "auth",
        "created": created.getTime(),
        "expires": expires.getTime(),
        "owner_uid": db_table_account.account_uid,
      }]);
      res.cookie('utail_session_uid', generated_uid, {
        maxAge: 7 * 24 * 60 * 60 * 1000, //7 дней
        secure: true,
        sameSite: 'strict',
        httpOnly: true,
      });
      res.cookie('utail_account_uid', db_table_account.account_uid, {
        maxAge: 7 * 24 * 60 * 60 * 1000, //7 дней
        secure: true,
        sameSite: 'strict',
        httpOnly: true,
      });
      res.status(200).json({ status:
        {
          type: "успех",
          message: "вы успешно вошли в аккаунт!"
        }
      });
      await database_db.query("DELETE FROM blacklist WHERE key = '$1", [email]);
    }
    else{
      await database_db.query("DELETE FROM blacklist WHERE key = '$1", [email]);
      throw new Error("Упс! Что-то пошло не так!");
    }
  }
  catch(error){
    console.log(error.toString());
    res.status(200).json({ status:
      {
        type: "ошибка",
        message: "Ошибка: ${error.toString()}",
      }
    });
    await database_db.query("DELETE FROM blacklist WHERE key = '$1", [email]);
  }
});
```

Рисунок 32. Публичная функция входа в аккаунт

```

EventsPost.on("sign_up", async (req, res) => {
  const { email, password, social_role, first_name, last_name } = req.body.arguments;

  const rules = [
    {
      condition: email == '' || password == '' || social_role == ''
        // || school_uid == ''
        || email == undefined || password == undefined || social_role == undefined,
      status: { type: "ошибка", message: "Ошибка формата. Пожалуйста, попробуйте в другой раз" }
    },
    {
      condition: !validator.isEmail(email),
      status: { type: "ошибка", message: "Ошибка формата. Неправильный e-mail" }
    },
    {
      condition: first_name.indexOf(' ') >= 0 || last_name.indexOf(' ') >= 0 || first_name == '' || last_name == '' || first_name == "" || last_name == "",
      status: { type: "ошибка", message: "Ошибка формата. Пожалуйста, попробуйте в другой раз" }
    }
  ];
  for (const rule of rules) {
    if (rule.condition) {
      return res.status(200).json({ status: rule.status });
    }
  }

  const sessionCheckResult = await PostFunctions.checkExistingSession(email);
  if (sessionCheckResult) return res.status(200).json(sessionCheckResult);

  const blacklistCheckResult = await PostFunctions.checkBlacklist(email);
  if (blacklistCheckResult) return res.status(200).json(blacklistCheckResult);

  const accountCheckResult = await PostFunctions.checkExistingAccount(email);
  if (accountCheckResult) return res.status(200).json(accountCheckResult);

  await database_db.query("INSERT INTO blacklist VALUES ($1, $2)", [uuidv4(), email]);

  let generated_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
  let created = new Date();
  let expires = new Date(created.getTime() + (5 * 60 * 1000)); //5 минут

  while((await database_db.query("SELECT * FROM sessions WHERE session_uid = $1", [generated_uid])).rows[0] != undefined){
    generated_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
  }

  const htmlPath = path.join(__dirname, '..', 'static', 'email_confirmation.html');
  try{
    bcrypt.hash(password, salt, async (err, hash) => {
      if (err) {
        return;
      }
      switch (social_role) {
        case 'ученик':
          await database_db.query("INSERT INTO sessions VALUES ($1, $2, $3)", [generated_uid, email, {
            "status": "unauth",
            "first_name": (first_name).charAt(0).toUpperCase() + (first_name).slice(1).toLowerCase(),
            "last_name": (last_name).charAt(0).toUpperCase() + (last_name).slice(1).toLowerCase(),
            "created": created.getTime(),
            "expires": expires.getTime(),
            "social_role": social_role,
            // "school_uid": school_uid,
            "password": hash,
          }]);
          break;
        default:
          await database_db.query("INSERT INTO sessions VALUES ($1, $2, $3)", [generated_uid, email, {
            "status": "unauth",
            "first_name": (first_name).charAt(0).toUpperCase() + (first_name).slice(1).toLowerCase(),
            "last_name": (last_name).charAt(0).toUpperCase() + (last_name).slice(1).toLowerCase(),
            "created": created.getTime(),
            "expires": expires.getTime(),
            "social_role": social_role,
            "password": hash,
          }]);
          break;
      }
    });
    await sendConfirmationEmail(req, transporter, {
      PathToHtml: htmlPath,
      subject: "регистрация и подтверждение почты",
      changes: {
        href: "#" : `href="${process.env.REACT_APP_BACKEND_URL}/api/get?action=confirm_email_to_create&session_uid-${generated_uid}"`
      }
    });
    res.status(200).json({ status:
      {
        type: "успех",
        message: "подтвердите, перейдя по ссылке на почте!"
      }
    });
    await database_db.query("DELETE FROM blacklist WHERE key = $1", [email]);
  });
} catch(error){
  res.status(200).json({ status:
    {
      type: "ошибка",
      message: `Что-то случилось! Упс! Ошибка: ${error.toString()}`
    }
  });
  return;
}
})

```

Рисунок 33. Публичная функция регистрации

Непосредственное подключение к СУБД осуществляется в самом начале (рис. 34): когда запускается проект, класс подключения хранится в файле postgresql.js (рис. 8).

```
const {Pool} = require('pg')
const pool = new Pool({
  user: "root",
  password: "root",
  host: "postgresql_container",
  port: 5432,
  database: "utail_db",
})
module.exports = pool
```

Рисунок 34. Блок кода для подключения к СУБД и взаимодействия с ней

2.8. Разработка веб-приложения

На этом этапе перейдём к frontend разработке с использованием языков HTML и CSS. Структура frontend приложения представлена на рисунке 8. Точкой входа для React.JS – является файл index.js. На первом этапе разработки мы импортируем сам React и его DOM, иногда называют его Virtual (с англ. означает виртуальный) DOM (рис. 35).

```
import React from 'react';
import ReactDOM from 'react-dom/client';

//компоненты
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <App />
);
```

Рисунок 35. Index.JS

DOM (Document Object Model) — это программный интерфейс для HTML- и XML-документов. Он представляет структуру документа в виде дерева объектов, где каждый узел — это отдельный элемент (тег, атрибут, текст и т.д.). Браузер создаёт это дерево, когда загружает HTML-страницу.

Virtual DOM (Виртуальный DOM) — это легковесная копия реального DOM, которая существует только в памяти JavaScript. Это просто JavaScript-объект, который описывает, как должен выглядеть интерфейс. React использует Virtual DOM как стратегию для повышения производительности. Такой подход абстрагирует разработчика от ручной работы с DOM, делая код более предсказуемым и удобным для сопровождения.

Созданный нами контейнер Virtual DOM, будет отслеживать любые изменения. Затем импортируем и обновляем компонент App.js через Virtual DOM (рис. 36).

```
import React, { useState, useEffect, useRef } from 'react';
import { BrowserRouter, Navigate, Route, Routes, useLocation } from "react-router-dom";
import './styles/Main.css';

//страницы
import Messenger from './components/Messenger';
import About from './components/About';
import NotFound from './components/NotFound';
import Example from './components/Example';

function App() {
  const location = useLocation();

  useEffect(() => {
    // console.log('Location changed!', location.pathname);
  }, [location]);

  return (
    <div className="App">
      <Routes>
        <Route path="/" element={ <About /> } />
        <Route path="/about" element={ <About /> } />
        <Route path="/messenger" element={ <Messenger /> } />
        <Route path="/invite/*" element={ <Messenger /> } />
        <Route path="/example" element={ <Example /> } />
        <Route path="*" element={ <NotFound /> } />
      </Routes>
    </div>
  );
}

export default function AppWrapper() {
  return (
    <BrowserRouter>
      <App />
    </BrowserRouter>
  );
}
```

Рисунок 36. App.js

Если возникнет ошибка, то появится сообщение: «Страница не найдена». Вот так выглядит страница ошибки 404(рис.37)

```
import React from 'react';
import { Navigate } from 'react-router-dom';

function NotFound() {
  return (
    <div>
      <Navigate to="/error?code=404" />
      <h1>404 - Страница не найдена</h1>
      <p>Похоже вы не туда попали!</p>
    </div>
  );
}

export default NotFound;
```

Рисунок 37. NotFound.js

Далее создали компонент, который поможет отображать важные системные уведомления, готовый к использованию в других компонентах (рис.38).

```
import React, { useState, useEffect } from 'react';
import '../styles/Notification.css';

const Notification = ({ message, position = 'bottom', onClose, color }) => {
  const [isVisible, setIsVisible] = useState(false);

  useEffect(() => {
    if (message) {
      setIsVisible(true);

      const timer = setTimeout(() => {
        setIsVisible(false);
      }, 5000); // Время отображения 5 секунды

      return () => clearTimeout(timer); // Очистка таймера при размонтировании
    }
  }, [message]);

  const handleTransitionEnd = () => {
    if (isVisible && onClose) {
      onClose(); // Вызов onClose только после завершения анимации скрытия
    }
  };

  return (
    <div
      className={`notification ${position} ${isVisible ? 'show' : ''}`}
      id={color}
      onTransitionEnd={handleTransitionEnd}
    >
      <p <b className="message">{message}</b></p>
    </div>
  );
};

export default Notification;
```

Рисунок 38. Notification.js

Далее создали компонент переключателя для страницы регистрации, чтобы определить статус «учитель» или «ученик» (рис. 39).

```
import React, { useState } from 'react';
import '../styles/Switcher.css';

function Switcher({onToggle, toggle}) {
  const [toggle, setToggle] = useState(false);

  const handleClick = () => {
    setToggle(prevState => !prevState);
    onToggle(!toggle);
  };

  return (
    <div className="Switcher">
      <button
        className={`toggle-btn ${toggle ? "active" : ""}`}
        onClick={handleClick}
      >
        <div className={`thumb ${toggle ? "active" : ""}></div>
      </button>
    </div>
  );
}

export default Switcher;
```

Рисунок 39. Переключатель, предназначенный для страницы регистрации

Импортируем функции для взаимодействия с состояниями, чтобы изменять Virtual DOM, а `useRef` – для получения DOM элемента через React (рис. 40).

```
import React, {useState, useEffect, useRef} from 'react';
import { throwRequest } from '../modules/main_functions.js';

import '../styles/CreateServerPanel.css'

const useOutsideClick = (callback) => {
  const ref = useRef();

  useEffect(() => {
    const handleClick = (event) => {
      if (ref.current && !event.target.closest('.white-window') && event.target.closest('.opened') && event.target.closest("#server_panel_group")) {
        callback(event);
      }
    };

    document.addEventListener('click', handleClick);

    return () => {
      document.removeEventListener('click', handleClick);
    };
  }, [ref, callback]);

  return ref;
};
```

Рисунок 40. CreateServerPanel.js, компонент окна создания сервера, 1 часть кода

Далее в точке входа функции создаём параметры, которые понадобятся для взаимодействия с данными родительского компонента. После чего создаём переменные для инкапсуляции. Используем `useOutsideClick` и передаём `callback` функцию для закрытия всего окна (рис. 40). `GotoPage`, `resetEverything` - функции для работы с окнами для инкапсуляции компонента (рис. 41).

```
function CreateServerPanel({createServerPanel, setCreateServerPanel, setServers}) {
  // панель создания серверов
  const defaultServerPanelData = {
    current_page: 'methods',
    previous_pages: [],
    server_form: {
      server_name: '',
      server_public_uid: ''
    },
    error: null
  };
  const [serverPanelData, setServerPanelData] = useState(defaultServerPanelData);
  const serverPanelDataRef = useRef(serverPanelData); // useRef для selectedContact
  useEffect(() => {
    serverPanelDataRef.current = serverPanelData; // Обновляем useRef при изменении selectedContact
  }, [serverPanelData]);

  const [settings_are_closed, setSettingsAreClosed] = useState(false);
  const [theme_is_closed, setThemeIsClosed] = useState(false);

  const resetEverything = () => {
    setCreateServerPanel(false);
    setTimeout(() => {
      setServerPanelData(defaultServerPanelData);
    }, 500);
  };

  const ref = useOutsideClick((event) => {
    resetEverything();
  });

  const gotoPage = (page, data) => {
    setSettingsAreClosed(true);
    setThemeIsClosed(true);
    setTimeout(() => {
      let current_data = serverPanelData;
      current_data.current_page = page;
      current_data.previous_pages.push(page);
      if(data && data.error) {
        current_data.error = data.error || null;
      } else {
        current_data.error = null;
      }
      setServerPanelData(current_data);

      setSettingsAreClosed(false);
      setThemeIsClosed(false);
    }, 500);
  };
}
```

Рисунок 41. CreateServerPanel.js, компонент окна создания сервера, 2 часть кода
Далее описываем JSX (рис. 42).

```

<div id="server_panel_group" className={createServerPanel ? 'opened' : 'closed'} ref={ref}>
  <div id="create_server_panel" className={createServerPanel ? 'opened white-window' : 'closed white-window'} >
    <div id="settings" className={settings_are_closed ? 'closed' : 'opened'}>
      /* тут будут кнопки и различные настройки */
      {serverPanelData.current_page === 'methods' &&
        <button onClick={() => {
          gotoPage('server');
        }}>Сервер для себя или коммерции</button>
      }
      {serverPanelData.current_page === 'server' &&
        <div id="server_create_form" style={{display: 'flex', flexDirection: 'column'}}>
          <input placeholder="введите название" onChange={(e) => {
            setServerPanelData(prevData => ({
              ...prevData,
              server_form: {
                ...prevData.server_form,
                server_name: e.target.value
              }
            }));
          }} style={{width: '80%', paddingLeft: '5px', marginLeft: '8%', height: '15px', borderRadius: '10px', border: '1px solid #ccc'}}>
          <input placeholder="пример: @150Y_01_15" onChange={(e) => {
            setServerPanelData(prevData => ({
              ...prevData,
              server_form: {
                ...prevData.server_form,
                server_public_uid: e.target.value
              }
            }));
          }} style={{width: '80%', paddingLeft: '5px', margin: '8px 8px 15px 8%', height: '15px', border: '1px solid #ccc'}}>
        </div>
      }
      {serverPanelData.current_page === 'server' &&
        <button className="create_btn_from_form" onClick={() => {
          gotoPage('loading');

          throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
            action: "user_gate",
            arguments: {user_action: "create_server", form: serverPanelDataRef.current.server_form}
          }).then((data) => {
            setTimeout(() => {
              if(serverPanelDataRef.current.previous_pages[serverPanelDataRef.current.previous_pages.length - 1] === 'server')
                if(data.status.type == "ycnex"){
                  setServers(prevServers => [...prevServers, data.delivery.new_server]);
                  resetEverything()
                }
              else{
                gotoPage('error', {error: data.status.message});
              }
            }, 1000);
          });
        }}>Создать сервер</button>
      }
    </div>
  </div>
</div>

```

Рисунок 42. CreateServerPanel.js, компонент окна создания сервера

На следующем этапе мы создаем компонент для конечной точки /messenger и /invite, но прежде чем сделать вполне хорошо функционирующий компонент, нам нужно импортировать готовые компоненты, которые помогут упростить разработку messenger.js. Именно поэтому мы импортируем различные компоненты и библиотеки (рис. 43).

```

JS Messengerjs X
backend > frontend > src > components > JS Messenger.js > Messenger > handleHomeButton
1  import React, { useState, useEffect, useLayoutEffect, useRef } from 'react';
2  import { useLocation, useNavigate } from 'react-router-dom';
3  import { throwRequest } from '../modules/main_functions.js';
4
5  import '../styles/Messenger.css';
6  import '../styles/LoadingChunk.css';
7
8  import Dropdown_menu from './Dropdown_menu';
9  import Notification from './Notification';
10 import Switcher from './Switcher';
11
12 import CreateServerPanel from './CreateServerPanel';
13 import UserSettings from './UserSettings';
14 import ServerSettings from './ServerSettings';
15 import BubbleWindows from './BubbleWindows';
16

```

Рисунок 43. Messenger.js, основной компонент мессенджера, 1 часть кода

Определив точку входа, исходя из паттерна создания React-компонентов, определяем состояния, которые пригодятся для работы с другими компонентами (рис. 44-45).

```

function Messenger() {
  const [first_name, setFirst_name] = useState('');
  const [last_name, setLast_name] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [social_role, setSocial_role] = useState('ученик');
  const [friend_for_request, setFriend_for_request] = useState('');
  const [visible, setVisible] = useState(false);
  const [authState, setAuthState] = useState('loading'); // 'loading', 'sign-in', 'sign-up', 'authenticated'

  const [notificationMessage, setNotificationMessage] = useState('');
  const [notificationColor, setNotificationColor] = useState('usual');
  const [notificationPosition, setNotificationPosition] = useState('bottom');

  const [user_metadata, setUser_metadata] = useState(null)

  const [resolved_contacts, setResolved_contacts] = useState(null)
  const [resolved_open_contacts, setResolved_open_contacts] = useState(null)
  const [selected_contact, setSelected_contact] = useState(null);
  const [contacts_requests, setContacts_requests] = useState([]);
  const [servers, setServers] = useState(null);
  const serversRef = useRef(servers);
  useEffect(() => {
    serversRef.current = servers;
  }, [servers]);

  const location = useLocation();
  const navigate = useNavigate();

  const selectedContactRef = useRef(selected_contact); // useRef для selectedContact
  useEffect(() => {
    selectedContactRef.current = selected_contact; // Обновляем useRef при изменении selectedContact
  }, [selected_contact]);

  const [current_chunk, setCurrent_chunk] = useState(0); // это текущий чанк сообщений, который мы получаем из бэкенда

  const conversation_ref = useRef(null); // это реф для скролла, чтобы мы могли прокручивать чат вниз
  const [messages, setMessages] = useState(null); // это массив сообщений, которые будут отображаться в чате

  const messagesRef = useRef(messages); // useRef для messages
  useEffect(() => {
    messagesRef.current = messages; // Обновляем useRef при изменении selectedContact
  }, [messages]);

  const [scroll_pusher_style, setScroll_pusher_style] = useState({}); // это стиль для скролла, чтобы он был внизу

```

Рисунок 44. Messenger.js, основной компонент мессенджера, 2 часть кода

```

let [final_name, setFinal_name] = useState(null)
const [socket, setSocket] = useState(null);
const observerRef = useRef(null);

const [written_message, setWritten_message] = useState(''); // это состояние для ввода сообщения в чат

```

Рисунок 45. Messenger.js, основной компонент мессенджера, 3 часть кода

Здесь же устанавливаем callback-функции обработчиков, которые будем далее настраивать (рис. 46).

```

const handleCloseNotification = () => {
  setNotificationMessage('');
};

//обработчики контактов
let [subpage, setSubpage] = useState({
  subpage: 'home',
  menu1: 'contacts',
  menu2: {
    primary: 'friends',
    secondary: 'all_friends'
  }
});

const handleClick = (e) => {
  const target = e.target.closest('#contact');
  if (target) {
    setSelected_contact({ uid: target.getAttribute('id2'), type: target.getAttribute('type') });
    setSubpage({
      ...subpage,
      menu2: {
        ...subpage.menu2,
        primary: 'chat',
      }
    });
  }
}

```

Рисунок 46. Messenger.js, основной компонент мессенджера, 4 часть кода

Создаём callback-функции для кнопок принятия запросов на дружбу: находим в HTML-документе кнопку, проверяем её существование, а затем определяем соответствующий контакт для удаления из списка запросов. Одновременно клиент отправляет запрос, чтобы уведомить сервер о том, что контакт был добавлен в друзья (рис. 47). Параллельно создаётся callback-функция для удаления контакта из списка запросов на тот случай, если пользователь отклонит приглашение.

```
// обработчики
const handleAcceptfriendrequest = (e) => {
  const target = e.target.closest("#accept_btn");
  if (target) {
    const contact = target.parentElement;
    const public_uid = contact.querySelector("#friend_public_uid").textContent.replace('@', '');

    //удаляем локально
    let contacts_requests_copy = contacts_requests;
    contacts_requests_copy = contacts_requests_copy.filter(req => {
      if (req.public_uid !== public_uid) {
        return true
      }
      else {
        navigator.sendBeacon(`${process.env.REACT_APP_BACKEND_URL}/api/beacon_drain`, JSON.stringify({
          action: "user_gate",
          arguments: { user_action: "accept_contact_request", contact_account_uid: req.account_uid }
        }));
        //req.account_uid - это уникальный идентификатор аккаунта который будет твоим другом :з
        return false
      }
    });
    setContacts_requests(contacts_requests_copy);
  }
}

const handleDeclinefriendrequest = (e) => {
  const target = e.target.closest("#decline_btn");
  if (target) {
    const contact = target.parentElement;
    const public_uid = contact.querySelector("#friend_public_uid").textContent.replace('@', '');

    //удаляем локально
    let contacts_requests_copy = contacts_requests;
    contacts_requests_copy = contacts_requests_copy.filter(req => {
      if (req.public_uid !== public_uid) {
        return true
      }
      else {
        //req.account_uid - это уникальный идентификатор аккаунта который не будет твоим другом :<
        navigator.sendBeacon(`${process.env.REACT_APP_BACKEND_URL}/api/beacon_drain`, JSON.stringify({
          action: "user_gate",
          arguments: { user_action: "decline_contact_request", contact_account_uid: req.account_uid }
        }));
        return false
      }
    });
    setContacts_requests(contacts_requests_copy);
  }
}
```

Рисунок 47. Messenger.js, основной компонент мессенджера, 5 часть кода

Также мы определили callback-функции для отправки запроса дружбы и обработки клавиш клавиатуры в случае, если пользователь захочет отправить запрос не по кнопке в HTML-документе, а по клавише клавиатуры (рис. 48).

```
const handleSendfriendrequest = () => {
  if (friend_for_request.trim() !== '' && friend_for_request.indexOf(' ') === -1) {
    setFriend_for_request("");

    //делаем ws запрос для отправки запроса в друзья
    socket.send(JSON.stringify({
      settings: {
        send_message_to_endpoint: true,
        handlers: [{ handler: "send_friend_request", condition: "before the sending message" }]
      },
      body: {
        endpoint_public_uid: friend_for_request,
        message: "new_friend_request"
      }
    }));
  }
  else {
    setNotificationMessage("пустые поля или идентификатор пробелом не допускается!");
    setNotificationColor("red");
    setNotificationPosition("bottom");
  }
}

const handleKeyUp = (e) => {
  if ((e.key === 'Enter' || e.keyCode === 13)) {
    handleSendfriendrequest();
  }
};

const handleKeyUpSendMessage = (e) => {
  if ((e.key === 'Enter' || e.keyCode === 13)) {
    handleSendMessage();
  }
};

const handleChange = (e) => {
  setFriend_for_request(e.target.value);
};
```

Рисунок 48. Messenger.js, основной компонент мессенджера, 6 часть кода

Далее создаем обработчики для обработки нажатий кнопки Home и открытия какой-либо беседы, внутри которой мы будем загружать новые сообщения по механизму «lazy loading» (рис. 49). А вообще что такое «lazy loading»? Lazy loading - это техника оптимизации, при которой загрузка определенных ресурсов (например, изображений, видео или скриптов) откладывается до тех пор, пока они не потребуются пользователю. Таким образом Utail загружает только 5 последних сообщений из базы данных.

```
const handleAnyContactClickForConversation = (contact, callback) => {
  //тут мы будем получать историю сообщений с этим контактом
  //и отображать их в чате. Эта функция вызывается каждый раз когда пользователь открывает чат с контактом
  //это функция служит для того, чтобы обновить состояние чата или создать новый чат, если его нет
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "user_gate",
    arguments: { user_action: "get_conversation_history", contact_account_uid: contact.account_uid || contact.server_uid, chunk: 'last chunk' }
  }).then((data) => {
    if (data.status.type == "ycnex") {
      setMessages(data.delivery.resolved_chunk_metadata);
      setCurrent_chunk(data.delivery.current_chunk);
      if (callback != undefined && typeof callback === 'function' && callback != null) callback(data);
    }
    else {
      if (data.status.can_be_ignored == true) {
        setMessages([]);
        setCurrent_chunk(0);
        if (callback != undefined && typeof callback === 'function' && callback != null) callback(data);
        return;
      }
      setNotificationMessage(data.status.message);
      setNotificationColor('red');
      setNotificationPosition('bottom');
      if (callback != undefined && typeof callback === 'function' && callback != null) callback(data);
    }
  })
}

const handleHomeButton = () => {
  setSubpage({
    ...subpage,
    subpage: 'home',
    menu1: 'contacts',
    menu2: {
      ...subpage.menu2,
      primary: 'friends',
    }
  });
  setSelected_contact(null);
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "user_gate",
    arguments: { user_action: "get_contacts" }
  }).then((data) => {
    if (data.status.type == "ycnex") setResolved_contacts(data.delivery.resolved_contacts);
  })
}
```

Рисунок 49. Messenger.js, основной компонент мессенджера, 7 часть кода

Далее создадим обработчики для кнопки регистрации аккаунта и кнопки скрытия пароля. При успехе действия мессенджер будет отображать зелёное окно уведомления (рис. 50).

```
const handleVisibilityPasswordButton = () => {
  visible ? setVisible(false) : setVisible(true);
};

const handleSignUp = () => {
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "sign_up", arguments: { email: email, password: password, social_role: social_role, school_uid: final_name, first_name: first_name, last_name: last_name },
    settings: {
      withCredentials: true
    }
  })
  .then((data) => {
    setNotificationMessage(data.status.message);
    setNotificationColor(data.status.type == 'ycnex' ? 'green' : 'red');
    setNotificationPosition('bottom');
  })
  .catch((error) => {
    console.error("Error in throwRequest:", error);
  });
};
```

Рисунок 50. Messenger.js, основной компонент мессенджера, 8 часть кода

Далее, создаём обработчик кнопки входа в аккаунт. Такой обработчик будет отправлять запрос публичной функции для входа в аккаунт. Если запрос успешно выполнится, то работает функция для изменения страницы вместе с handleMetadata (рис. 51). Также здесь имеется обработчик для отправки сообщения на сервер по веб-сокету.

```

const handleSignIn = () => {
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, { action: "sign_in", arguments: { email: email, password: password, social_role: social_role } })
    .then((data) => {
      if (data.status.type === 'успех') {
        setAuthState('authenticated');

        //получаем метаданные пользователя
        handleMetadata()
      }
    })
    .catch((error) => {
      console.error("Error in throwRequest:", error);
    });
};

const handleSendMessage = () => {
  if (written_message.trim() !== '') {
    // отправляем сообщение через вебсокет
    socket?.send(JSON.stringify({
      settings: {
        send_message_to_endpoint: true,
        handlers: [{ handler: "add_message_to_conversation", condition: "before the sending message" }]
      },
      body: {
        send_message_type: 'contact',
        endpoint_uid: selected_contact.uid,
        message: "new_message",
        message_content: written_message,
      }
    }));
    let messages_in_array = messages != null ? messages : [];
    setMessages([...messages_in_array, {
      owner_uid: user_metadata.account_uid,
      public_name: user_metadata.public_name,
      social_role: user_metadata.social_role,
      message: written_message,
    }]);
    setWritten_message(''); // очищаем поле ввода сообщения
  } else {
    console.log('Сообщение не может быть пустым!');
  }
};

```

Рисунок 51. Messenger.js, основной компонент мессенджера, 9 часть кода

Первый блок кода, заключенный в `useEffect`, отвечает за автоматическую проверку аутентификации пользователя при загрузке приложения. При монтировании компонента выполняется асинхронный POST-запрос на сервер по адресу, указанному в переменных окружения (рис. 52). Запрос передает действие «`authenticate_by_session_uid`». Ключевой параметр `withCredentials: true` гарантирует, что браузер отправит на сервер cookies (включая идентификатор сессии), что является стандартным механизмом для аутентификации. Если сервер возвращает статус «успех», функция `setAuthState(«authenticated»)` переводит приложение в аутентифицированное состояние. После этого вызывается функция `handleMetadata()`. Если сервер возвращает любой другой статус, состояние аутентификации сбрасывается `setAuthState(«sign-in»)`, и пользователю отображается уведомление об ошибке с сообщением от сервера.

В случае если сетевой запрос завершится неудачно (например, нет соединения с интернетом), код в блоке `catch` также переведет пользователя на экран входа «`sign-in`». Этот механизм обеспечивает вход для пользователя, который уже авторизован в системе, без необходимости повторного ввода логина и пароля. Второй блок кода, использующий хук `useLayoutEffect`, реализует механизм ленивой загрузки «`lazy loading`» данных. Конкретно в интерфейсе чата это реализовано в виде паттерна «`infinity scroll`», который подгружает предыдущие порции сообщений по мере необходимости, когда пользователь прокручивает историю вверх (рис. 52).

На рисунке 53 представлен код, реализующий два важных аспекта пользовательского интерфейса: автоматическую прокрутку чата и систему обработки глубоких ссылок «`deeplinks`» для приглашений.

```

useEffect(() => {
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "authenticate_by_session_uid", arguments: {},
    settings: {
      withCredentials: true
    }
  })
  .then((data) => {
    if (data.status.type === 'ycnex') {
      setAuthState('authenticated');

      //получаем метаданные пользователя
      handleMetadata()
    }
    else {
      setNotificationMessage(data.status.message);
      setNotificationColor(data.status.type === 'ycnex' ? 'green' : 'red');
      setNotificationPosition('bottom');
      setAuthState('sign-in');
    }
  })
  .catch((error) => {
    setAuthState('sign-in');
  });
}, []);
useLayoutEffect(() => {
  //Менюем setScroll_pusher_style чтобы объект для observe был сверху всегда за сообщениями
  //let scrollable_conversation = document.getElementById('scrollable_conversation');
  let scrollable_conversation = conversation_ref.current;
  if (scrollable_conversation) {
    let size_each_message = 98; // высота каждого сообщения в пикселях
    let all_messages_height = messages ? messages.length * size_each_message : 0; // высота всех сообщений в пикселях

    let scroll_pusher_height = Math.max(0, scrollable_conversation.clientHeight - all_messages_height);
    setScroll_pusher_style({ height: `${scroll_pusher_height}px`, width: '100%' });
    // if (scroll_pusher_height > 0) {
    //   setScroll_pusher_style({ height: `${scroll_pusher_height}px`, width: '100%' });
    // }

    if (observerRef.current) observerRef.current.disconnect(); // отключаем наблюдатель после загрузки следующего чанка
    //ставим observe для загрузки старых сообщений
    // если текущий чанк больше 1, то мы можем наблюдать за предыдущим чанком
    const observer = new IntersectionObserver((entries, observer) => {
      entries.forEach(entry => {
        if (entry.isIntersecting && current_chunk > 1) {
          //const originalScrollTop = scrollable_conversation.scrollTop;
          throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
            action: "user_gate",
            arguments: { user_action: "get_conversation_history", contact_account_uid: selected_contact.uid, chunk: current_chunk - 1 }
          });
          .then((data) => {
            if (data.status.type === "ycnex") {
              setMessages([...data.delivery.resolved_chunk_metadata, ...messages]);
              setCurrent_chunk(data.delivery.current_chunk);

              // const newHeight = scrollable_conversation.scrollHeight;
              // scrollable_conversation.scrollTop = newHeight - originalScrollTop;

              // прокручиваем вниз, чтобы пользователь не заметил скачок
              // scrollable_conversation.scrollTop = originalScrollTop;
            }
            else {
              setNotificationMessage(data.status.message);
              setNotificationColor('red');
              setNotificationPosition('bottom');
            }
          })
        }
      });
    }, { threshold: 0.8 }); // 0.8 - это порог видимости, можно настроить по желанию
    observerRef.current = observer; // сохраняем observer в реф, чтобы можно было отключить его позже

    const targetElement = document.getElementById('previous_chunk');
    if (targetElement) {
      observer.observe(targetElement);
    }
  });
  return () => {
    if (observerRef.current) {
      observerRef.current.disconnect();
    }
  };
}, [messages])

```

Рисунок 52. Messenger.js, основной компонент мессенджера, 10 часть кода

```

useEffect(() => {
  // наблюдаем за изменениями scroll_pusher_style и прокручиваем чат вниз
  const scrollable_conversation = conversation_ref.current;
  if (scrollable_conversation) {
    scrollable_conversation.scrollTop = scrollable_conversation.scrollHeight;
  }
}, [scroll_pusher_style])

// система определения, что хочет ссылка сделать при загрузке страницы
useEffect(() => {
  if (authState === 'authenticated') {
    const pathnames = location.pathname.substring(1).split('/');
    if (pathnames[0] === "invite") {
      //открываем тут bubble-window
      bubbleWindowsRef.current["addWindow"]("invite_request");

      let invite_uid = pathnames[1];
      if (invite_uid !== undefined && invite_uid !== null) {
        if (invite_uid.length === 0) {
          bubbleWindowsRef.current["invoke_invite_error"]();
          return;
        }
        throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
          action: "user_gate",
          arguments: { user_action: "get_invite_metadata", uid: invite_uid }
        });
        .then((data) => {
          if (data.status.type === "ycnex") {
            bubbleWindowsRef.current["invoke_invite_success"]([...data.delivery, invite_uid: invite_uid]);
          }
          else {
            bubbleWindowsRef.current["invoke_invite_error"]();
          }
        })
      }
    }
    else {
      bubbleWindowsRef.current["invoke_invite_error"]();
    }
  }
  else if (authState === 'sign-in' || authState === 'sign-up') {
    navigate('/messenger')
  }
}, [authState])

```

Рисунок 53. Messenger.js, основной компонент мессенджера, 11 часть кода

Алгоритм обработки глубоких ссылок активируется при изменении статуса аутентификации: для авторизованных пользователей система анализирует текущий URL-путь, извлекает сегменты адреса и при обнаружении пути «invite» открывает всплывающее окно для обработки приглашения, извлекает идентификатор приглашения, отправляет запрос на сервер для получения метаданных и в зависимости от ответа сервера либо отображает интерфейс принятия приглашения, либо показывает сообщение об ошибке, в то время как для пользователей, у которых не получилось прочитать глубокую ссылку, автоматически срабатывает перенаправление на основную страницу мессенджера.

Далее создаём простой механизм, который будет менять идентификатор тела для использования разных стилей в разных страницах (рис. 54).

```
document.body.id = `body-${authState}`;
```

Рисунок 54. Messenger.js, основной компонент мессенджера, 12 часть кода

Таким образом, мы завершили разработку мессенджера на Node.JS. Полный код проекта находится в Приложении.

ЗАКЛЮЧЕНИЕ

Проведя анализ теоретической и научно-методической литературы, проанализировав некоторые существующие мессенджеры, был разработан веб-мессенджер на Node.JS, реализующий следующие функции:

- организация обмена сообщениями;
- возможность просмотра истории сообщения;
- контролирование информационных потоков администратором портала;
- обеспечение различных методов поиска информации;
- разделение прав доступа пользователей к информации.

Был создан цельный визуальный прототип школьного мессенджера с полной системой экранов, выдержанных в едином стиле. Разработаны UX-сценарии, вайрфреймы, UI-дизайн и кликабельный прототип. Интерфейс получился современным, лёгким, понятным и дружелюбным для школьной аудитории.

Монохромная палитра в рамках одного ключевого оттенка, округлые формы и простая типографика создают ощущение спокойствия, а единая структура экранов обеспечивает удобство навигации.

Главная идея дизайна – «простота общения и человеческое взаимодействие в цифровой среде». Проект показал, что даже школьное приложение может быть современным, технологичным и эстетически цельным.

Сервис по обмену сообщений реализован современными методами программирования с использованием системного подхода решения задач. В дальнейшем проект требует доработки по следующим пунктам:

1. Оптимизация и безопасность через JWT.
2. Доработка функций коммуникаций пользователей через WebSocket.
3. Добавление функций видеосвязи.
4. Усовершенствование CSS-Стилей.
5. Доработка статуса человека — онлайн или офлайн.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Абрамова, Л.В. Инструментальные средства информационных систем [Электронный ресурс]: учебное пособие / Л.В. Абрамова ; Министерство образования и науки Российской Федерации, Архангельск : САФУ, 2013.
2. Анисимов, В. В. Протоколы аутентификации(идентификации),2013
3. Анисимов, В.В. // Anisimovkhv [Электронный ресурс]. URL:
4. Антонов, В.Ф. Методы и средства проектирования информационных систем [Электронный ресурс]: учебное пособие / В.Ф. Антонов, А.А. Москвитин ; - Ставрополь : СКФУ, 2016.
5. Баженова, И.Ю. SQL и процедурно-ориентированные языки [Электронный ресурс]/ И.Ю. Баженова. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016.
6. Буренин, С. Н. Web-программирование и базы данных [Электронный ресурс] : учеб. практикум / С. Н. Буренин. - Москва : Моск. гуманит. ун-т, 2014.
7. Гуцин, А.Н. Базы данных [Электронный ресурс]: учебник / А.Н. Гуцин. - М. : Директ-Медиа, 2014.
8. Карпова, Т.С. Базы данных: модели, разработка, реализация [Электронный ресурс]: учебное пособие / Т.С. Карпова. - 2-е изд., исправ. - М. : Национальный Открытый Университет «ИНТУИТ», 2016.
9. Маркин, А.В. Построение запросов и программирование на SQL [Электронный ресурс]: учебное пособие / А.В. Маркин. - 3-е изд., перераб. и доп. - М. : Диалог-МИФИ, 2014.
10. Прохорова, О.В. Информатика [Электронный ресурс]: учебник / О.В. Прохорова; Самара, 2013.
11. Савельева, Н.В. Язык программирования PHP [Электронный ресурс]/ Н.В. Савельева. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016.
12. Строганов, А.С. Ваш первый сайт с использованием PHP-скриптов [Электронный ресурс]: учебное пособие / А.С. Строганов. - 3-е изд.. испр. и доп. - М. : Диалог-МИФИ, 2015.
13. Салий, В. Н. Криптографические методы и средства защиты информации : учеб. пособие [Электронный ресурс] / В. Н. Салий. Саратов : 2012.
14. Шелухин, О. И. Моделирование информационных систем [Электронный ресурс] : учеб. пособие. 004 / О. И. Шелухин. - 2-е изд., перераб. и доп. - Москва : Горячая линия - Телеком, 2012.
15. Управление данными [Электронный ресурс]: учебник / Ю.Ю. Громов, О.Г. Иванова, А.В. Яковлев, В.Г. Однолько ; Тамбов:Издательство ФГБОУ ВПО «ТГТУ», 2015.
16. Документация Telegram [Электронный ресурс]. URL: <https://tlgrm.ru/docs/mtproto>
17. Viber [Электронный ресурс]. URL: <http://www.viber.com/ru/>
18. WhatsApp [Электронный ресурс]. URL: <https://www.whatsapp.com/>
19. ICQ [Электронный ресурс]. URL: <https://icq.com/android/ru>
20. Google Hangouts [Электронный ресурс].URL: <https://hangouts.google.com/>
21. Skype [Электронный ресурс]. URL: <https://www.skype.com/ru/>
22. Telegram Messenger [Электронный ресурс]. URL: <https://telegram.org/>

Программный код мессенджера на Node.JS
Dockerfile

```
FROM node:23-alpine AS frontend
WORKDIR /app/frontend
COPY ./frontend/package*.json ./
COPY ./frontend .
RUN npm install
ARG REACT_APP_BACKEND_URL
ENV REACT_APP_BACKEND_URL=$REACT_APP_BACKEND_URL
ARG REACT_APP_BACKEND_WS_URL
ENV REACT_APP_BACKEND_WS_URL=$REACT_APP_BACKEND_WS_URL
ENV LANG=C.UTF-8
ENV LC_ALL=C.UTF-8
RUN npm run build
FROM node:23-alpine AS backend
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
ARG REACT_APP_BACKEND_URL
ENV REACT_APP_BACKEND_URL=$REACT_APP_BACKEND_URL
ARG REACT_APP_BACKEND_WS_URL
ENV REACT_APP_BACKEND_WS_URL=$REACT_APP_BACKEND_WS_URL
ARG BACKEND_WS_PORT
ENV BACKEND_WS_PORT=$BACKEND_WS_PORT
ARG BACKEND_PORT
ENV BACKEND_PORT=$BACKEND_PORT
COPY --from=frontend /app/frontend/build ./static/build
EXPOSE $BACKEND_PORT
ENV LANG=C.UTF-8
```

```
ENV LC_ALL=C.UTF-8
```

```
CMD ["npm", "run", "start"]
```

main.js

```
const express = require('express');
const cors = require('cors');
const path = require('path')
const cookieParser = require('cookie-parser');
const cookie = require('cookie');
const url = require('node:url');
const WebSocket = require('ws');

const websocket_server = new WebSocket.Server({ noServer: true })
const app = express();
const {EventsGet, EventsPost, createTables} = require('./events/http_events.js');
const {PostFunctions} = require('./events/http_events_modules.js')

const port = process.env.BACKEND_PORT;

app.use(cors({origin: process.env.REACT_APP_BACKEND_URL }));
app.use(express.json());
app.use(cookieParser());
app.use(express.static(path.resolve('static', 'build')));

// http:// && https://
app.post('/api/post', async (req, res) => {
  const action = req.body.action;

  console.log(`[backend-server] получил POST запрос: ${action}`);

  if (EventsPost.eventNames().includes(action)) {
```

```

try {
  EventsPost.emit(action, req, res);
} catch (error) {
  console.error(`Ошибка при эмитинге действия ${action}:`, error);
  res.status(500).json({ error: "Internal server error" });
}
} else {
  console.log(`Событие ${action} не зарегистрировано.`);
  res.status(400).json({ error: `Событие "${action}" не зарегистрировано.` });
}
});
app.get('/api/get', async (req, res) => {
  const scheme = req.protocol || 'http';
  const host = req.get('host');
  const path = req.originalUrl;

  const baseURL = `${scheme}://${host}${path}`;
  const parsedURL = url.parse(baseURL, true);

  const action = parsedURL.query.action
  console.log(`[backend-server] получил GET запрос: ${action}`);
  if (EventsGet.eventNames().includes(action)) {
    try {
      EventsGet.emit(action, req, res, parsedURL);
    } catch (error) {
      console.error(`Ошибка при эмитинге действия ${action}:`, error);
      res.status(500).json({ error: "Internal server error" });
    }
  } else {
    console.log(`Событие ${action} не зарегистрировано.`);

```

```

    res.status(400).json({ error: `Событие "${action}" не зарегистрировано.` });
  }
});
app.get('/*name', (req, res) => {
  res.sendFile(path.resolve('static','build', 'index.html'));
});
app.post('/api/beacon_drain', express.json(), express.text(), async (req, res) => {
  if (typeof req.body === "string"){
    req.body = JSON.parse(req.body)
    //gate
    const action = req.body.action;
    console.log(`[backend-server] получил POST-beacon запрос: ${action}`);
    if (EventsPost.eventNames().includes(action)) {
      try {
        EventsPost.emit(action, req, res);
      } catch (error) {
        console.error(`Ошибка при эмитинге действия ${action}:`, error);
        res.status(500).json({ error: "Internal server error" });
      }
    } else {
      console.log(`Событие ${action} не зарегистрировано.`);
      res.status(400).json({ error: `Событие "${action}" не зарегистрировано.` });
    }
  }
  else{
    res.status(500).json({error: 'данный запрос не соответствует требованиям'})
  };
})
const httpServer = app.listen(port, () => {
  createTables();
});

```

```

    console.log(`Backend listening on port: ${port}`);
  });
  // вебсокеты (ws://) - является обратной, непрерывной связью
  const {executeHandler} = require('./events/websocket_handlers.js');

  httpServer.on('upgrade', (req, socket, head) => {
    websocket_server.handleUpgrade(req, socket, head, (ws) => {
      websocket_server.emit('connection', ws, req)
    })
  })

  let connections = {};

  websocket_server.on('connection', async (ws_client_stream, incoming_request) => {
    console.log('Новый клиент подключился');

    let utail_session_uid;
    let utail_account_uid;

    // Получение куки файлов
    if (incoming_request.headers.cookie) {
      const cookies = cookie.parse(incoming_request.headers.cookie);
      utail_session_uid = cookies.utail_session_uid;
      utail_account_uid = cookies.utail_account_uid;
    }

    // Проверка наличия куки файлов
    if (!utail_session_uid || !utail_account_uid) {
      console.warn("Клиент подключился без utail_session_uid или utail_account_uid. Закрываем
соединение.");

      ws_client_stream.close(1000, "Требуется аутентификация."); //Закрываем с кодом ошибки

      return;
    }

    // Валидация формата куки файлов
    if (typeof utail_session_uid !== 'string' || utail_session_uid.trim() === "" ||
      typeof utail_account_uid !== 'string' || utail_account_uid.trim() === "") {

```

```

    console.warn("Обнаружена попытка взлома. Неверный формат utail_session_uid или
utail_account_uid. Закрываем соединение.");

    ws_client_stream.close(1008, "Попытка взлома.");

    return;
}

try {
    //Получение и проверка сессии (ПРИ ПОДКЛЮЧЕНИИ)

    let session_obj = await PostFunctions.getSession_obj({ session_uid: utail_session_uid });

    if (!session_obj) throw new Error('Ошибка формата сессии. Пожалуйста, попробуйте в другой
раз.');
```

```

    if ((new Date()).getTime() > session_obj.metadata.expires || session_obj.metadata.status != "auth" ||
session_obj.metadata.owner_uid != utail_account_uid)

        throw new Error('Попытка вызвать несанкционированный запрос! Неверная сессия.');
```

```

    //Получение и проверка аккаунта (ПРИ ПОДКЛЮЧЕНИИ)

    let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });

    if (!account_obj) throw new Error('Ошибка. Такого аккаунта не существует!');
```

```

    if(connections[utail_account_uid] == null || connections[utail_account_uid] == undefined){
        connections[utail_account_uid] = []
    }

    connections[utail_account_uid].push({
        ws: ws_client_stream,
        session_uid: utail_session_uid,
        account_uid: utail_account_uid,
    });

    let last_index = connections[utail_account_uid].length - 1

    connections[utail_account_uid][last_index].ws.on('message', async message => {

        const buffered = Buffer.from(message)

        const string_message = buffered.toString();

        const json_message = JSON.parse(string_message);

        console.log(`Получено сообщение сервером: ${string_message}`);

```

```

let result = []
let handlers_state = {
  clients: {
    first_client: {
      states: []
    },
    second_client: {
      states: []
    }
  }
}
// Проверка наличия нужных процедур в сообщении
if (json_message.settings && json_message.settings.handlers) {
  for (const handlerConfig of json_message.settings.handlers) {
    if (handlerConfig.condition === "before the sending message") {
      try {
        let res = await executeHandler(handlerConfig.handler, {
          utail_session_uid: utail_session_uid,
          utail_account_uid: utail_account_uid,
          body: json_message.body
        }, handlers_state)
        result.push(res);
      } catch (handlerError) {
        console.error(`Ошибка при выполнении обработчика ${handlerConfig.handler}:`,
handlerError);
        result.push(-1);
      }
    }
  }
}
// сделать проверку на наличие hasPermission - в случае если это сообщение собирается

```

отправиться другому клиенту, который заблокировал отправителя

```
// Отправка сообщения другому клиенту, если endpoint_uid указан и соединение существует
// let sent_pong = false;

if(!result.includes(-1) && result.includes(0)){

    // Здесь логика для отправки сообщения другим клиентам, которые подключены к этому
серверу, указанные handler(ом).

    for (const key in handlers_state.clients) {

        if (key !== 'first_client' && key !== 'second_client') {

            const client = handlers_state.clients[key];

            console.log(`клиент ${key} существует!`)

            let sender_account = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid })

            console.log(connections[key])

            if(connections[key]){

                connections[key].forEach(function (connection, index) {

                    if(connection.ws.readyState === WebSocket.OPEN){

                        connection.ws.send(JSON.stringify({

                            status: {

                                type: "успех",

                                message: "рукопожатие и передача данных прошли успешно!"

                            },

                            sender_public_info: {

                                public_name: sender_account.metadata.first_name + ' ' +
sender_account.metadata.last_name,

                                public_uid: sender_account.public_uid,

                                account_uid: sender_account.account_uid,

                                social_role: sender_account.metadata.social_role,

                            },

                            handlers_state: client,

                            message: json_message.body.message,

                            initiated: "server",

                        }));
```

```

    }
  });
}
}
}
if(json_message.settings.send_message_to_endpoint != null
  && json_message.settings.send_message_to_endpoint != undefined
  && json_message.settings.send_message_to_endpoint == true)
{
  if(json_message.body.endpoint_uid != undefined
    && connections[json_message.body.endpoint_uid] != null
    && connections[json_message.body.endpoint_uid] != undefined)
  {
    let sender_account = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid })
    if(connections[json_message.body.endpoint_uid]){
      connections[json_message.body.endpoint_uid].forEach(function (connection, index) {
        if (connection.ws.readyState === WebSocket.OPEN) {
          connection.ws.send(JSON.stringify({
            status: {
              type: "успех",
              message: "рукопожатие и передача данных прошли успешно!"
            },
            sender_public_info: {
              public_name: sender_account.metadata.first_name + ' ' +
sender_account.metadata.last_name,
              public_uid: sender_account.public_uid,
              account_uid: sender_account.account_uid,
              social_role: sender_account.metadata.social_role,
            },
            handlers_state: handlers_state.clients.second_client,
            message: json_message.body.message,

```

```

        initiated: "user",
    )))
}
})
}
}
else if(json_message.body.endpoint_public_uid !== undefined){
    let correct_endpoint_public_uid = (json_message.body.endpoint_public_uid).replace("@", "");
    let resolved_endpoint_uid = (await PostFunctions.getAccount_obj({ public_uid:
correct_endpoint_public_uid })).account_uid
    let sender_account = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid })
    if(connections[resolved_endpoint_uid] !== null && connections[resolved_endpoint_uid] !==
undefined){
        connections[resolved_endpoint_uid].forEach(function (connection, index) {
            if (connection.ws.readyState === WebSocket.OPEN) {
                connection.ws.send(JSON.stringify({
                    status: {
                        type: "успех",
                        message: "рукопожатие и передача данных прошли успешно!"
                    },
                    sender_public_info: {
                        public_name: sender_account.metadata.first_name + ' ' +
sender_account.metadata.last_name,
                        public_uid: sender_account.public_uid,
                        account_uid: sender_account.account_uid,
                        social_role: sender_account.metadata.social_role,
                    },
                    handlers_state: handlers_state.clients.second_client,
                    message: json_message.body.message,
                    initiated: "user",
                })))

```

```

    }
  })
}
}
}
}
}
//обеспечиваем heartbeat 0x9 - ping фрейм, а 0xA - pong
else if(parseInt(string_message) === 0x9){
  if(connections[utail_account_uid]){
    if(connections[utail_account_uid][last_index].ws.readyState === WebSocket.OPEN){
      connections[utail_account_uid][last_index].ws.send(0xA);
    }
  }
}
if(parseInt(string_message) !== 0x9 && connections[utail_account_uid]){
  if(connections[utail_account_uid][last_index].ws.readyState === WebSocket.OPEN){
    connections[utail_account_uid][last_index].ws.send(JSON.stringify({
      status: {
        type: "успех",
        message: "рукопожатие и передача данных прошли успешно!"
      },
      handlers_state: handlers_state.clients.first_client,
      message: "handlers_results",
    }));
  }
}
});
connections[utail_account_uid].forEach(function (connection, index) {
  connection.ws.send(JSON.stringify({status: {
    type: "успех",

```

```

    message: "клиент успешно подключился!"
  }));
})
connections[utail_account_uid].forEach(function (connection, index) {
  connection.ws.on('close', async () => {
    console.log(`Клиент ${utail_account_uid} отключился (websocket)`);
    delete connections[utail_account_uid][index];
  });
})
}
catch(error){
  console.error("Ошибка в обработчике WebSocket:", error);
  ws_client_stream.send(JSON.stringify({ status: { type: "ошибка", message: error.toString() } })); //
Отправляем ошибку клиенту
  ws_client_stream.close(1011, "Внутренняя ошибка сервера."); // Закрываем соединение с кодом
ошибки
}
});
console.log(`WebSocket сервер запущен на ${process.env.REACT_APP_BACKEND_WS_URL}`);

```

http_events_modules.js

```

const database_db = require('../library/postgresql')

//Post функции
const checkExistingSession = async (email) => {
  const db_table = (await database_db.query("SELECT * FROM sessions WHERE key = $1",
[email])).rows[0];
  if (db_table) {
    const json_table = db_table.metadata;
    if (json_table.status === 'unauth') {
      const dateNow = new Date();
      if (json_table.expires <= dateNow.getTime()) {

```

```

    await database_db.query("DELETE FROM sessions WHERE key = $1", [email]);
  } else {
    return {
      status: {
        type: "ошибка",
        message: "Ошибка. Запрос на регистрацию уже существует! Ожидаем
подтверждение!"
      }
    };
  }
}
}
return null;
};

const getSession_obj = async ({key, session_uid}) =>{
  let session_obj = null;
  if(key !== undefined){
    session_obj = (await database_db.query("SELECT * FROM sessions WHERE key = $1",
[key])).rows[0];
  }
  else if(session_uid !== undefined){
    session_obj = (await database_db.query("SELECT * FROM sessions WHERE session_uid = $1",
[session_uid])).rows[0];
  }
  else{
    throw new Error('Нет подходящего параметра для нахождения сессии!')
  }
  return session_obj;
};

const getAccount_obj = async ({key, account_uid, public_uid}) =>{
  let account_obj = null;

```

```

    if(key !== undefined){
        account_obj = (await database_db.query("SELECT * FROM accounts WHERE key = $1",
[key])).rows[0];
    }
    else if(account_uid !== undefined){
        account_obj = (await database_db.query("SELECT * FROM accounts WHERE account_uid = $1",
[account_uid])).rows[0];
    }
    else if(public_uid !== undefined){
        account_obj = (await database_db.query("SELECT * FROM accounts WHERE public_uid = $1",
[public_uid])).rows[0];
    }
    else{
        throw new Error('Нет подходящего параметра для нахождения сессии!')
    }
    return account_obj;
};

const getServer_obj = async ({server_uid, public_uid}) =>{
    let server_obj = null;
    if(server_uid !== undefined){
        server_obj = (await database_db.query("SELECT * FROM servers WHERE server_uid = $1",
[server_uid])).rows[0];
    }
    else if(public_uid !== undefined){
        server_obj = (await database_db.query("SELECT * FROM servers WHERE public_uid = $1",
[public_uid])).rows[0];
    }
    else{
        throw new Error('Нет подходящего параметра для нахождения сессии!')
    }
    return server_obj;
};

```

```

const checkBlacklist = async (email) => {
    const db_table_blacklist = (await database_db.query("SELECT * FROM blacklist WHERE key = $1",
[email])).rows[0];
    if (db_table_blacklist) {
        return {
            status: {
                type: "ошибка",
                message: "Ошибка. Пожалуйста подождите!"
            }
        };
    }
    return null;
};

const checkExistingAccount = async (email) => {
    const db_table_account = (await database_db.query("SELECT * FROM accounts WHERE key = $1",
[email])).rows[0];
    if (db_table_account) {
        return {
            status: {
                type: "ошибка",
                message: "Ошибка. По такой электронной почте уже существует аккаунт!"
            }
        };
    }
    return null;
};

//GET функции
module.exports = {
    PostFunctions : {checkExistingAccount, checkBlacklist, checkExistingSession, getSession_obj,
getAccount_obj, getServer_obj},
    GetFunctions : null,

```

```
}
```

http_events.js

```
const EventEmitter = require('events');
const validator = require('validator');
const bcrypt = require('bcrypt');
const nodemailer = require("nodemailer");
const path = require('path')
const cookieParser = require('cookie-parser');
const {
  v1: uuidv1,
  v4: uuidv4,
} = require('uuid');
const {secret_admin_key} = require('../data/global/admin_config.json')
const sendConfirmationEmail = require("../library/send_email_message.js")
const sendHtml = require('../library/sendHtml.js');
const {PostFunctions} = require('./http_events_modules.js')
const EventsPost = new EventEmitter;
const EventsGet = new EventEmitter;
const ActionEvents = new EventEmitter;
//количество случайных генерация хэша
const salt = 12;
//служебные настройки
//— база данных
const database_db = require('../library/postgresql');
async function createTables() {
  try {
    await database_db.query(`
      CREATE TABLE IF NOT EXISTS sessions (
        session_uid TEXT PRIMARY KEY,
        key TEXT,
```

```

    metadata JSONB
  );
`);
console.log('Таблица sessions была создана или уже существует!');
    await database_db.query(`
CREATE TABLE IF NOT EXISTS accounts (
  account_uid TEXT PRIMARY KEY,
  key TEXT,
  password TEXT,
  metadata JSONB,
  public_uid TEXT,
  counter_uid SERIAL
);
`);
console.log('Таблица accounts была создана или уже существует!');
await database_db.query(`
CREATE TABLE IF NOT EXISTS blacklist (
  blacklisted_uid TEXT PRIMARY KEY,
  key TEXT
);
`);
console.log('Таблица blacklist была создана или уже существует!');
//сами чанки сообщений
await database_db.query(`
CREATE TABLE IF NOT EXISTS chunks_messages (
  messages_uid TEXT NOT NULL,
  metadata JSONB,
  messages_limit INTEGER,
  chunk_uid BIGINT NOT NULL,
  global_uid SERIAL PRIMARY KEY

```

```

);
`);
console.log("Таблица chunks_messages была создана или уже существует!");

//предназначена для того, чтобы сервер понимал, что пользователь один *уже* имеет
историю сообщений с другим человеком

await database_db.query(`
CREATE TABLE IF NOT EXISTS owners_of_messages_uids (
account_uid1 TEXT,
account_uid2 TEXT,
messages_uid TEXT,
metadata JSONB,
global_uid SERIAL PRIMARY KEY
);
`);
console.log("Таблица owners_of_messages_uids была создана или уже существует!");
await database_db.query(`
CREATE TABLE IF NOT EXISTS servers (
server_uid TEXT PRIMARY KEY,
metadata JSONB,
public_uid TEXT,
counter_uid SERIAL
);
`);
// Пример структуры metadata для servers:
// {
// members_uids: [];
// server_uid: "Уникальный идентификатор сервера"; // уникальный идентификатор
сервера по которому можно найти сервер + пользователи могут зайти свободно на него, если
is_private === false
// is_private: true; // если true, то сервер приватный и только приглашённые пользователи
могут присоединиться
// server_name: "Название сервера";

```

```

    // server_description: "Описание сервера";

    // invites: [ // люди могут зайти на сервер по приглашению через ссылку
<сайт>?action=use_invite&invite_uid=<invite_uid>
    // {
    //   invite_uid: "Уникальный идентификатор приглашения";
    //   created: "Дата создания приглашения";
    //   expires: "Дата истечения приглашения";
    //   owner_uid: "UID владельца приглашения";
    // }
    // ];
    // }

    console.log('Таблица servers была создана или уже существует!');
  } catch (error) {
    console.error('Ошибка при создании таблиц:', error);
  }
}

//— служебная почта noreply.utail@gmail.com

const config_email = {
  service: 'gmail',
  auth: {
    user: 'noreply.utail@gmail.com',
    pass: 'hvlutvgrcdsrxb',
  }
}

const transporter = nodemailer.createTransport(config_email)

//события /api/post
EventsPost.on("authenticate_by_session_uid", async (req, res) => {
  const session_uid = req.cookies.utail_session_uid;
  const account_uid = req.cookies.utail_account_uid;
  if(session_uid == " || session_uid == undefined || account_uid == " || account_uid == undefined)
  {

```

```

res.status(200).json({ status:
  {
    type: "ошибка",
    message: "Ошибка формата. Пожалуйста попробуйте в другой раз"
  }
});
return;
}
try{
  let db_table = (await database_db.query("SELECT * FROM sessions WHERE session_uid = $1",
[session_uid])).rows[0];
  if(db_table !== undefined){
    let json_table = db_table.metadata;
    if(json_table.status !== 'auth')
      throw new Error("Сессия не подходит под требования входа!");
  }
  else throw new Error("Сессия не найдена!");
  let account_table = (await database_db.query("SELECT * FROM accounts WHERE account_uid =
$1", [account_uid])).rows[0];
  if(account_table === undefined)
    throw new Error("Сессия не подходит под требования входа!");
}
catch(error){
  res.status(200).json({ status:
    {
      type: "ошибка",
      message: error.toString(),
    }
  });
  return;
}

```

```

res.status(200).json({ status:
  {
    type: "успех",
    message: "ВЫ УСПЕШНО ВОШЛИ В АККАУНТ!"
  },
});
})

EventsPost.on("sign_in", async (req, res) => {
  const email = req.body.arguments.email;
  const password = req.body.arguments.password;
  const social_role = req.body.arguments.social_role;
  try{
    if(email == " || password == " || social_role == " || email == undefined || password == undefined ||
social_role == undefined)
      throw new Error("Ошибка формата. Пожалуйста попробуйте в другой раз");
    if(validator.isEmail(email) == false)
      throw new Error("Ошибка формата. Неправильный e-mail");
    let db_table_account = (await database_db.query("SELECT * FROM accounts WHERE key = $1",
[email])).rows[0];
    const accountCheckResult = await PostFunctions.checkExistingAccount(email);
    if(!accountCheckResult) throw new Error("Ошибка. Такого аккаунта не существует!");
    const blacklistCheckResult = await PostFunctions.checkBlacklist(email);
    if (blacklistCheckResult) throw new Error("Пожалуйста подождите!");
    await database_db.query("INSERT INTO blacklist VALUES ($1, $2)", [uuidv4(), email]);
    const match = await bcrypt.compare(password, db_table_account.password);
    if(match){
      let created = new Date();
      let expires = new Date(created.getTime() + (7 * 24 * 60 * 60 * 1000)); //7 дней
      let generated_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
      while((await database_db.query("SELECT * FROM sessions WHERE session_uid = $1",
[generated_uid])).rows[0] != undefined){

```

```

    generated_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
};
await database_db.query("INSERT INTO sessions VALUES ($1, $2, $3)", [generated_uid, email, {
    "status" : "auth",
    "created" : created.getTime(),
    "expires" : expires.getTime(),
    "owner_uid": db_table_account.account_uid,
}]);
res.cookie('utail_session_uid', generated_uid, {
    maxAge: 7 * 24 * 60 * 60 * 1000, //7 дней
    secure: true,
    sameSite: 'strict',
    httpOnly: true,
});
res.cookie('utail_account_uid', db_table_account.account_uid, {
    maxAge: 7 * 24 * 60 * 60 * 1000, //7 дней
    secure: true,
    sameSite: 'strict',
    httpOnly: true,
});
res.status(200).json({ status:
    {
        type: "успех",
        message: "ВЫ УСПЕШНО ВОШЛИ В АККАУНТ!"
    },
});
await database_db.query("DELETE FROM blacklist WHERE key = $1", [email]);
}
else{
    await database_db.query("DELETE FROM blacklist WHERE key = $1", [email]);
}

```

```

    throw new Error("Упс! Что-то пошло не так!")
  };
}
catch(error){
  console.log(error.toString())
  res.status(200).json({ status:
    {
      type: "ошибка",
      message: `Ошибка: ${error.toString()}`,
    },
  });
  await database_db.query("DELETE FROM blacklist WHERE key = $1", [email]);
};
})
EventsPost.on("sign_up", async (req, res) => {
  const { email, password, social_role, first_name, last_name } = req.body.arguments;
  const rules = [
    {
      condition: email == "" || password == "" || social_role == ""
      // || school_uid == ""
      || email == undefined || password == undefined || social_role == undefined,
      status: { type: "ошибка", message: "Ошибка формата. Пожалуйста, попробуйте в другой раз"
    }
  },
  {
    condition: !validator.isEmail(email),
    status: { type: "ошибка", message: "Ошибка формата. Неправильный e-mail" }
  },
  {
    condition: first_name.indexOf(' ') >= 0 || last_name.indexOf(' ') >= 0 || first_name == "" || last_name
    == "" || first_name == "" || last_name == "",

```

```

        status: { type: "ошибка", message: "Ошибка формата. Пожалуйста, попробуйте в другой раз" }
    }
];
for (const rule of rules) {
    if (rule.condition) {
        return res.status(200).json({ status: rule.status });
    }
}
const sessionCheckResult = await PostFunctions.checkExistingSession(email);
if (sessionCheckResult) return res.status(200).json(sessionCheckResult);
const blacklistCheckResult = await PostFunctions.checkBlacklist(email);
if (blacklistCheckResult) return res.status(200).json(blacklistCheckResult);
const accountCheckResult = await PostFunctions.checkExistingAccount(email);
if (accountCheckResult) return res.status(200).json(accountCheckResult);
await database_db.query("INSERT INTO blacklist VALUES ($1, $2)", [uuidv4(), email]);
let generated_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
let created = new Date();
let expires = new Date(created.getTime() + (5 * 60 * 1000)); //5 минут
while((await database_db.query("SELECT * FROM sessions WHERE session_uid = $1",
[generated_uid])).rows[0] != undefined){
    generated_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
}
const htmlPath = path.join(__dirname, '..', 'static', 'email_confirmation.html');
try{
    bcrypt.hash(password, salt, async (err, hash) => {
        if (err) {
            return;
        }
    });
    switch (social_role) {
        case 'ученик':
            await database_db.query("INSERT INTO sessions VALUES ($1, $2, $3)", [generated_uid, email,

```

```

{
  "status" : "unauth",
  "first_name" : (first_name).charAt(0).toUpperCase() + (first_name).slice(1).toLowerCase(),
  "last_name" : (last_name).charAt(0).toUpperCase() + (last_name).slice(1).toLowerCase(),
  "created" : created.getTime(),
  "expires" : expires.getTime(),
  "social_role" : social_role,
  // "school_uid" : school_uid,
  "password" : hash,
}]);
break;
default:
  await database_db.query("INSERT INTO sessions VALUES ($1, $2, $3)", [generated_uid, email,
{
  "status" : "unauth",
  "first_name" : (first_name).charAt(0).toUpperCase() + (first_name).slice(1).toLowerCase(),
  "last_name" : (last_name).charAt(0).toUpperCase() + (last_name).slice(1).toLowerCase(),
  "created" : created.getTime(),
  "expires" : expires.getTime(),
  "social_role" : social_role,
  "password" : hash,
}]);
break;
}
await sendConfirmationEmail(req, transporter, {
  PathToHtml : htmlPath,
  subject : "регистрация и подтверждение почты",
  changes: {
    'href="#"' :
`href="${process.env.REACT_APP_BACKEND_URL}/api/get?action=confirm_email_to_create&session_uid=${generated_uid}"`

```

```

    }
  })
  res.status(200).json({ status:
    {
      type: "успех",
      message: "подтвердите, перейдя по ссылке на почте!"
    },
  });

  await database_db.query("DELETE FROM blacklist WHERE key = $1", [email]);
});
}
catch(error){
  res.status(200).json({ status:
    {
      type: "ошибка",
      message: `Что-то случилось! Упс! Ошибка: ${error.toString()}`
    },
  });
  return;
}
})
// const whitelist = ["school", "grade", "letter"]
// EventsPost.on("get_grades", async (req, res) => {
//   const get_grades_by_obj = req.body.arguments.get_grades_by_obj
//   const argument2 = req.body.arguments.argument2
//   const school = req.body.arguments.school
//   if(school !== undefined && argument2 !== undefined && get_grades_by_obj !== undefined &&
whitelist.includes(get_grades_by_obj)){
//     if(get_grades_by_obj === "grade"){
//       const school_Db = (await database_db.query(`SELECT * FROM schools_v2 WHERE school =
$1;`, [school])).rows[0];

```

```

//      res.status(200).json({
//          status: {
//              type: "успех",
//              message: "Вы успешно получили все уникальные объекты grade!"
//          },
//          delivery: {
//              grades: school_Db.data.classes.map(row => row.grade),
//          }
//      });
//  }
//  else if(get_grades_by_obj === "letter"){
//      const school_Db = (await database_db.query(`SELECT * FROM schools_v2 WHERE school =
//      $1;`, [school])).rows[0];
//      let current_grade = argument2;
//      const letters = school_Db.data.classes.filter(cls => cls.grade === current_grade);
//      res.status(200).json({
//          status: {
//              type: "успех",
//              message: `Вы успешно получили все объекты letters по классу ${argument2}!`
//          },
//          delivery: {
//              letters: letters.map(row => row.letter),
//          }
//      });
//  }
//  else if(get_grades_by_obj === "school"){
//      const result = await database_db.query(`SELECT DISTINCT school FROM schools_v2;`);
//      const schools = result.rows.map(row => row.school);
//      res.status(200).json({
//          status: {
//              type: "успех",

```

```

//      message: "Вы успешно получили все уникальные объекты school!"
//      },
//      delivery: {
//          schools: schools,
//      }
//  });
//  }
//  } else {
//      console.log(`Запрещено получение данных по полю: ${get_grades_by_obj}`);
//      return res.status(400).json({ error: "Недопустимый параметр запроса." });
//  }
// })
EventsPost.on("user_gate", async (req, res) => {
    try{
        const {utail_session_uid, utail_account_uid} = req.cookies;
        const user_action = req.body.arguments.user_action;

        if(!utail_session_uid || !utail_account_uid) throw new Error("Требуется аутентификация. Попытка
        вызвать несанкционированный запрос!");

        if(!utail_session_uid || typeof utail_session_uid !== 'string' || utail_session_uid.trim() === "" ||
        !utail_account_uid || typeof utail_account_uid !== 'string' || utail_account_uid.trim() === "") throw
        new Error("Попытка вызвать несанкционированный запрос! (обнаружена попытка взлома)");

        let session_obj = await PostFunctions.getSession_obj({session_uid: utail_session_uid});

        if(session_obj === undefined || session_obj === null) throw new Error("Ошибка формата.
        Пожалуйста, попробуйте в другой раз");

        if((new Date()).getTime() > session_obj.metadata.expires || session_obj.metadata.status !== "auth" ||
        session_obj.metadata.owner_uid !== utail_account_uid) throw new Error("Попытка вызвать
        несанкционированный запрос!");

        let account_obj = await PostFunctions.getAccount_obj({account_uid: utail_account_uid});

        if(account_obj === undefined || account_obj === null) throw new Error("Ошибка. Такого аккаунта не
        существует!");

        let allowedActions = [
            "get_account_metadata",

```

```

    "get_contacts",
    "get_resolved_open_contacts",
    "get_resolved_contacts_requests",
    "add_open_contact",
    "accept_contact_request",
    "decline_contact_request",
    "get_conversation_history",
    "add_message_to_conversation",
    "get_resolved_servers",
    "create_server",
    "get_invite_metadata",
    "create_invite",
    "join_server_with_invite",
    "get_server_members",
    "ban_member",
    "unban_member",
    "get_banned_members",
];

if(account_obj.role === 'администратор') {allowedActions.push('admin_action1');
allowedActions.push('admin_action2')};

//вызываем тут функцию пользователя

if(allowedActions.includes(user_action) && ActionEvents.eventNames().includes(user_action)){
    ActionEvents.emit(user_action, req, res);
}

else{
    throw new Error('Попытка вызвать несанкционированный запрос!');
}

}catch(error){

    console.error("Ошибка в user_gate:", error);
    res.status(200).json({ status: { type: "ошибка", message: error.toString() } })
}

```

```

});
//события для user_gate
ActionEvents.on('get_account_metadata', async (req, res) => {
  try{
    const {utail_account_uid} = req.cookies;
    let account_obj = await PostFunctions.getAccount_obj({account_uid: utail_account_uid});
    //я должен вернуть (публичную информацию)/(метаданные) пользователя прямым образом
    по res в формате json
    res.status(200).json({
      status: {
        type: "успех",
        message: "данные успешно были получены!"
      },
      delivery: {
        public_uid: account_obj.public_uid,
        public_name: account_obj.metadata.first_name + ' ' + account_obj.metadata.last_name,
        social_role: account_obj.metadata.social_role,
        // school_uid: account_obj.metadata.school_uid,
        contacts: account_obj.metadata.contacts,
        contacts_requests: account_obj.metadata.contacts_requests,
        account_uid: account_obj.account_uid,
      }
    })
  }catch(error){
    res.status(500).json({ status:
      {
        type: "ошибка",
        message: error.toString()
      },
    });
  }
}

```

```

});
ActionEvents.on('get_contacts', async (req, res) => {
  try{
    const {utail_account_uid} = req.cookies;
    let account_obj = await PostFunctions.getAccount_obj({account_uid: utail_account_uid});
    let resolved_contacts = await Promise.all(account_obj.metadata.contacts.map(async (contact) => {
      let contact_account_obj = await PostFunctions.getAccount_obj({account_uid:
contact.account_uid});
      return {
        public_name: contact_account_obj.metadata.first_name + ' ' +
contact_account_obj.metadata.last_name,
        public_uid: contact_account_obj.public_uid,
        social_role: contact_account_obj.metadata.social_role,
        account_uid: contact_account_obj.account_uid,
      };
    }));
    res.status(200).json({
      status: {
        type: "успех",
        message: "данные успешно были получены!"
      },
      delivery: {
        resolved_contacts: resolved_contacts,
      }
    })
  }catch(error){
    res.status(500).json({ status:
    {
      type: "ошибка",
      message: error.toString()
    },
  },

```

```

    });
  }
});
ActionEvents.on('get_server_members', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    const {server_uid} = req.body.arguments;

    // Проверяем аргументы, чтобы избежать некорректных данных и SQL-инъекций
    if(!server_uid || typeof server_uid !== 'string' || server_uid.trim() === "") throw new Error("Ошибка формата. Попытка взлома!")

    //получаем участников сервера

    let server_obj = await PostFunctions.getServer_obj({server_uid: server_uid})

    let resolved_contacts = await Promise.all(server_obj.metadata.members_uids.map(async (member_uid)
=> {
      let member_account_obj = await PostFunctions.getAccount_obj({account_uid: member_uid});
      return {
        public_name: member_account_obj.metadata.first_name + ' ' +
member_account_obj.metadata.last_name,
        public_uid: member_account_obj.public_uid,
        social_role: member_account_obj.metadata.social_role,
        account_uid: member_account_obj.account_uid,
      };
    }));
    res.status(200).json({
      status: {
        type: "успех",
        message: "данные успешно были получены!"
      },
      delivery: resolved_contacts
    })
  }
}

```

```

catch(error){
  res.status(500).json({ status:
    {
      type: "ошибка",
      message: error.toString()
    },
  });
}
})
ActionEvents.on('get_banned_members', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    const { server_uid } = req.body.arguments;
    // Проверяем аргументы, чтобы избежать некорректных данных и SQL-инъекций
    if(!server_uid || typeof server_uid !== 'string' || server_uid.trim() === "") throw new Error("Ошибка
формата. Попытка взлома!")
    //получаем участников сервера
    let server_obj = await PostFunctions.getServer_obj({server_uid: server_uid})
    let resolved_contacts = await Promise.all(server_obj.metadata.banned_uids.map(async (member_uid)
=> {
      let member_account_obj = await PostFunctions.getAccount_obj({account_uid: member_uid});
      return {
        public_name: member_account_obj.metadata.first_name + '' +
member_account_obj.metadata.last_name,
        public_uid: member_account_obj.public_uid,
        social_role: member_account_obj.metadata.social_role,
        account_uid: member_account_obj.account_uid,
      };
    }));
    res.status(200).json({
      status: {

```

```

    type: "успех",
    message: "данные успешно были получены!"
  },
  delivery: resolved_contacts
})
}
catch(error){
  res.status(500).json({ status:
    {
      type: "ошибка",
      message: error.toString()
    },
  });
}
})
ActionEvents.on('get_resolved_servers', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
    let resolved_servers = await Promise.all(account_obj.metadata.servers.map(async (server) => {
      let server_obj = await PostFunctions.getServer_obj({ server_uid: server.server_uid})
      if(server_obj){
        return {
          server_uid: server_obj.server_uid,
          server_name: server_obj.metadata.server_name,
          full_name: server_obj.metadata.server_name,
          public_uid: server_obj.public_uid,
          public_description: server_obj.metadata.server_description || "",
          is_private: server_obj.metadata.is_private || false,
          members_uids: server_obj.metadata.members_uids || [],

```

```

    invites: server_obj.metadata.invites || [],
    owner_uid: server_obj.metadata.owner_uid
  };
}
else{
  account_obj.metadata.servers = account_obj.metadata.servers.filter(server_rawr =>
server_rawr.server_uid != server.server_uid)
  return null;
}
}));
await database_db.query(
  'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
  [account_obj.metadata, utail_account_uid]
);
resolved_servers = resolved_servers.filter(server => server !== null);
res.status(200).json({
  status: {
    type: "успех",
    message: "данные успешно были получены!"
  },
  delivery: {
    resolved_servers: resolved_servers || [],
  }
});
}
catch(error){
  res.status(500).json({ status:
  {
    type: "ошибка",
    message: error.toString()
  },

```

```

    });
  }
});
ActionEvents.on('get_resolved_open_contacts', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
    const openContacts = account_obj.metadata.opened_contacts || [];
    let resolved_open_contacts;
    if (openContacts.length === 0) {
      resolved_open_contacts = [];
    } else {
      resolved_open_contacts = await Promise.all(openContacts.map(async (contact) => {
        let contact_account_obj = await PostFunctions.getAccount_obj({ account_uid: contact.account_uid
      }));
      return {
        public_name: contact_account_obj.metadata.first_name + ' ' +
contact_account_obj.metadata.last_name,
        public_uid: contact_account_obj.public_uid,
        social_role: contact_account_obj.metadata.social_role,
        account_uid: contact_account_obj.account_uid,
      };
    }));
  }
  res.status(200).json({
    status: {
      type: "успех",
      message: "данные успешно были получены!"
    },
    delivery: {
      resolved_open_contacts: resolved_open_contacts,

```

```

    }
  });
} catch(error){
  res.status(500).json({ status:
    {
      type: "ошибка",
      message: error.toString()
    },
  });
}
});
ActionEvents.on('get_resolved_contacts_requests', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
    const contacts_requests = account_obj.metadata.contacts_requests || [];
    let resolved_contacts_requests;
    if (contacts_requests.length === 0) {
      resolved_contacts_requests = [];
    } else {
      resolved_contacts_requests = await Promise.all(contacts_requests.map(async (contact) => {
        let contact_account_obj = await PostFunctions.getAccount_obj({ account_uid: contact.account_uid
      }));
      return {
        public_name: contact_account_obj.metadata.first_name + '' +
contact_account_obj.metadata.last_name,
        public_uid: contact_account_obj.public_uid,
        social_role: contact_account_obj.metadata.social_role,
        account_uid: contact_account_obj.account_uid,
      };
    });
  });
}

```

```

}

res.status(200).json({
  status: {
    type: "успех",
    message: "данные успешно были получены!"
  },
  delivery: {
    resolved_contacts_requests: resolved_contacts_requests,
  }
});
} catch(error){
  res.status(500).json({ status:
    {
      type: "ошибка",
      message: error.toString()
    },
  });
}
})

ActionEvents.on('add_open_contact', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
    let metadata = account_obj.metadata || {};
    if (!metadata.opened_contacts.some(contact => contact.account_uid ===
req.body.arguments.contact_account_uid)) {
      metadata.opened_contacts.push({
        account_uid: req.body.arguments.contact_account_uid
      })
    }
  }
}

```

```

await database_db.query(
  'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
  [metadata, utail_account_uid]
);
res.status(200).json({
  status: {
    type: "успех",
    message: "данные успешно были получены!"
  },
});
} catch(error){
  res.status(500).json({ status:
    {
      type: "ошибка",
      message: error.toString()
    },
  });
}
})
ActionEvents.on('accept_contact_request', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
    let metadata = account_obj.metadata || {};
    let contact_obj = await PostFunctions.getAccount_obj({ account_uid:
req.body.arguments.contact_account_uid });
    let contact_metadata = contact_obj.metadata || {};
    if (metadata.contacts_requests.some(contact => contact.account_uid ===
req.body.arguments.contact_account_uid)) {
      metadata.contacts.push({

```

```

    account_uid: req.body.arguments.contact_account_uid
  })
  contact_metadata.contacts.push({
    account_uid: account_obj.account_uid,
  })
}

metadata.contacts_requests = metadata.contacts_requests.filter((contact) => contact.account_uid !==
req.body.arguments.contact_account_uid)

await database_db.query(
  'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
  [metadata, utail_account_uid]
);

await database_db.query(
  'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
  [contact_metadata, contact_obj.account_uid]
);

res.status(200).json({
  status: {
    type: "успех",
    message: "данные успешно были получены!"
  },
});
} catch(error){
  res.status(500).json({ status:
  {
    type: "ошибка",
    message: error.toString()
  },
  });
  console.error(error)
}

```

```

})
ActionEvents.on('decline_contact_request', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
    let metadata = account_obj.metadata || {};
    metadata.contacts_requests = metadata.contacts_requests.filter((contact) => contact.account_uid !==
req.body.arguments.contact_account_uid)
    await database_db.query(
      'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
      [metadata, utail_account_uid]
    );
    res.status(200).json({
      status: {
        type: "успех",
        message: "данные успешно были получены!"
      },
    });
  }catch(error){
    res.status(500).json({ status:
      {
        type: "ошибка",
        message: error.toString()
      },
    });
    console.error(error)
  }
})
//нужно сделать автовозрастание chunk_uid когда добавляется новое сообщение с одинаковым
messages_uid, чтобы не было проблем с чанками
//нужно сделать

```

```

// case "contact": чѐто делать
// case "server": чѐто делать
// case "group": чѐто делать
ActionEvents.on('add_message_to_conversation', async (req, res) => {
  try{
    // case "contact":
    const { utail_account_uid } = req.cookies;
    const contact_account_uid = req.body.arguments.contact_account_uid;
    const message = req.body.arguments.message;
    let messages_uid; // уникальный идентификатор переписки
    // проверка на пустое сообщение
    if (!message || typeof message !== 'string' || message.trim() === "") throw new Error('Сообщение не может быть пустым!');
    if (!contact_account_uid || typeof contact_account_uid !== 'string' || contact_account_uid.trim() === "") throw new Error('contact_account_uid не может быть пустым!');
    // запрос к базе данных для получения уникального идентификатора переписки
    let conversation_ownership = (await database_db.query(
      `SELECT * FROM owners_of_messages_uids
      WHERE (account_uid1 = $1 AND account_uid2 = $2)
      OR (account_uid1 = $2 AND account_uid2 = $1)`,
      [utail_account_uid, contact_account_uid]
    )).rows[0];
    //получаем messages_uid и проверяем, есть ли переписка между пользователем и контактом
    if(!conversation_ownership){
      let account_obj = await PostFunctions.getAccount_obj({ account_uid: contact_account_uid });
      let server_obj = await PostFunctions.getServer_obj({ server_uid: contact_account_uid });
      if (!account_obj && !server_obj) throw new Error('Такого контакта или сервера не существует!');
      messages_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4(); //генерируем уникальный идентификатор переписки
      try {
        await database_db.query(

```

```

    'INSERT INTO owners_of_messages_uids (account_uid1, account_uid2, messages_uid, metadata)
VALUES ($1, $2, $3, $4)',

    [utail_account_uid, contact_account_uid, messages_uid, { created: new Date().getTime() }]
);
} catch (dbError) {
    console.error("Ошибка при создании ownership:", dbError);
    throw new Error("Ошибка при создании переписки: " + dbError.message);
}
} //нужно сделать в этом блоке: создаём переписку и добавляем сообщение. А перед тем, как
создать нужно проверить, есть ли account_uid или server_uid в базе данных
else{
    messages_uid = conversation_ownership.messages_uid;
}
const totalChunksResult = await database_db.query(
    'SELECT COUNT(*) FROM chunks_messages WHERE messages_uid = $1',
    [messages_uid]
); //количество всего чанков с одинаковым messages_uid
const totalChunks = parseInt(totalChunksResult.rows[0].count) || 0; //количество чанков <int>
let chunk_from_DB = (await database_db.query('SELECT * FROM chunks_messages WHERE
chunk_uid = $1 AND messages_uid = $2', [totalChunks, messages_uid])).rows[0];
if(!chunk_from_DB) {
    await database_db.query('INSERT INTO chunks_messages (messages_uid, metadata,
messages_limit, chunk_uid) VALUES ($1, $2, $3, $4)', [messages_uid, {
    messages:[{
        owner_uid: utail_account_uid,
        message: message,
    }],
    }, 5, totalChunks + 1]); // 5 - (для теста) лимит сообщений в чанке
}
else{
    if(!chunk_from_DB.metadata.messages) throw new Error("Проблема с метаданными сообщений в
чанке!");
}

```

```

if(chunk_from_DB.metadata.messages.length >= chunk_from_DB.messages_limit){
    //создаем новый чанк и добавляем туда сообщение

    await database_db.query('INSERT INTO chunks_messages (messages_uid, metadata,
messages_limit, chunk_uid) VALUES ($1, $2, $3, $4)', [messages_uid, {
        messages:[{
            owner_uid: utail_account_uid,
            message: message,
        }],
        }, 5, totalChunks + 1]); // 5 - (для теста) лимит сообщений в чанке
    }
else{
    //добавляем сообщение в существующий чанк

    chunk_from_DB.metadata.messages.push({
        owner_uid: utail_account_uid,
        message: message,
    });

    await database_db.query('UPDATE chunks_messages SET metadata = $1 WHERE messages_uid =
$2 AND chunk_uid = $3', [chunk_from_DB.metadata, messages_uid, totalChunks]);

    };
}
res.status(200).json({ status: {
    type: "успех",
    message: "Сообщение успешно добавлено в переписку!"
}});
}
catch(e){
    console.error("Ошибка в add_message_to_conversation:", e);
    res.status(500).json({
        status: {
            type: "ошибка",
            message: e.toString(),
        }
    });
}

```

```

    },
  });
};
});
ActionEvents.on('get_conversation_history', async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    const contact_account_uid = req.body.arguments.contact_account_uid;
    if (!contact_account_uid || typeof contact_account_uid !== 'string' || contact_account_uid.trim() === "")
      throw new Error('contact_account_uid не может быть пустым!');
    //*****[заглушка для предотвращения получения сообщений при разных
ситуациях]*****
    let account_obj = await PostFunctions.getAccount_obj({ account_uid: contact_account_uid });
    let server_obj = await PostFunctions.getServer_obj({ server_uid: contact_account_uid });
    if(!account_obj && !server_obj) throw new Error('Такого контакта или сервера не существует!');
    //когда пользователь пытается получить переписку с сервером, в котором он не состоит
    if(!account_obj && !server_obj.metadata.members_uids.includes(utail_account_uid)){
      //удаляем сервер из массива servers пользователя, чтобы он не видел его в интерфейсе
      let user_account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid }); //
получаем объект аккаунта пользователя отправляющего запрос
      user_account_obj.metadata.servers = user_account_obj.metadata.servers.filter(server =>
server.server_uid !== server_obj.server_uid); // удаляем сервер из массива servers
      await database_db.query('UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
[user_account_obj.metadata, utail_account_uid]);
      //проверяем, существует ли сервер
      if(!server_obj)
        throw new Error('Такого сервера не существует!');
      //выдаём ошибку, что пользователь не состоит в этом сервере
      throw new Error('Вы не являетесь участником этого сервера!');
    }
  }
  //*****

```

```
let get_type = "contact"; //по умолчанию получаем переписку с контактом
if(server_obj && !account_obj) get_type = "server"; //если есть аккаунт сервера, то получаем
переписку с сервером

let messages_uid; // уникальный идентификатор переписки
// Проверяем, есть ли переписка между пользователем и контактом
let conversation_ownership = (await database_db.query(
`SELECT * FROM owners_of_messages_uids
WHERE (account_uid1 = $1 AND account_uid2 = $2)
OR (account_uid1 = $2 AND account_uid2 = $1)`,
[utail_account_uid, contact_account_uid]
)).rows[0];
if(!conversation_ownership) {
if(get_type === "contact") {
messages_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
}
else if(get_type === "server") {
messages_uid = server_obj.server_uid;
}
try {
await database_db.query(
`INSERT INTO owners_of_messages_uids (account_uid1, account_uid2, messages_uid, metadata)
VALUES ($1, $2, $3, $4)`,
[utail_account_uid, contact_account_uid, messages_uid, { created: new Date().getTime() }]
);
} catch (dbError) {
console.error("Ошибка при создании ownership:", dbError);
throw new Error("Ошибка при создании переписки: " + dbError.message);
}
} else {
messages_uid = conversation_ownership.messages_uid;
```

```

}

let chunk = req.body.arguments.chunk;

let current_chunk = null;

if(chunk === 'last chunk'){

  const totalChunksResult = await database_db.query(
    'SELECT COUNT(*) FROM chunks_messages WHERE messages_uid = $1',
    [messages_uid]
  );

  const totalChunks = parseInt(totalChunksResult.rows[0].count) || 0; //количество чанков
  current_chunk = totalChunks;
}

else{

  chunk = parseInt(chunk) || 0;

  if(isNaN(chunk) || chunk <= 0) throw new Error('Неверные параметры запроса. Пожалуйста,
  проверьте chunk.');
```

current_chunk = chunk;

```

}

let chunk_from_DB = (await database_db.query('SELECT * FROM chunks_messages WHERE
chunk_uid = $1 AND messages_uid = $2', [current_chunk, messages_uid])).rows[0];

//тут нужно произвести resolve сообщений, чтобы клиент знал, чьё это сообщение
//если чанк не найден, то нужно вернуть ошибку

if(!chunk_from_DB) throw new Error('Чанк не найден!');

if(!chunk_from_DB.metadata.messages) throw new Error('Проблема с метаданными сообщений в
чанке!');

chunk_from_DB.metadata.messages = await
Promise.all(chunk_from_DB.metadata.messages.map(async (message_obj) => {

  let contact_account_obj = await PostFunctions.getAccount_obj({ account_uid:
message_obj.owner_uid });

  return {

    public_name: contact_account_obj.metadata.first_name + '' +
contact_account_obj.metadata.last_name,

    public_uid: contact_account_obj.public_uid,
```

```

    social_role: contact_account_obj.metadata.social_role,
    owner_uid: contact_account_obj.account_uid,
    message: message_obj.message,
  };
  }));
  //добавляем сервер к пользователю, чтобы пользователь мог видеть сервер у себя в интерфейсе
  res.status(200).json({
    status: {
      type: "успех",
      message: "данные успешно были получены!"
    },
    delivery: {
      current_chunk: current_chunk,
      resolved_chunk_metadata: chunk_from_DB.metadata.messages,
    }
  })
}
catch(e){
  console.error("Ошибка в get_conversation_history:", e);
  res.status(200).json({
    status: {
      type: "ошибка",
      message: e.toString(),
      can_be_ignored: e.message.includes("Чанк не найден!"),
    },
  });
}
});
ActionEvents.on('create_server', async (req, res) => {
  try{

```

```

const { utail_account_uid } = req.cookies;

//проверяем аргументы, чтобы не совершить sql инъекцию
if(!req.body || !req.body.arguments || !req.body.arguments.form)
  throw new Error('Ошибка формата. Пожалуйста, попробуйте в другой раз');

  const {server_name} = req.body.arguments.form;
let {server_public_uid} = req.body.arguments.form;
if (!server_name || typeof server_name !== 'string' || server_name.trim() === "")
  throw new Error('server_name не может быть пустым!');
if (server_name.length > 25) throw new Error('server_name не может быть длиннее 25 символов!');
if (!server_public_uid || typeof server_public_uid !== 'string' || server_public_uid.trim() === "")
  throw new Error('server_public_uid не может быть пустым!');
if (server_public_uid.length < 3 || server_public_uid.length > 25)
  throw new Error('server_public_uid должен быть от 3 до 25 символов!');
if(server_public_uid.trim() !== "" && server_public_uid.indexOf(' ') === -1){}
else throw new Error('server_public_uid не может содержать пробелы!');
//проверяем, есть ли запрещенные символы в server_public_uid
const forbiddenSymbols = "()!№@#$$%^&*()-=+{}[];\"'<>.,?/\\";
for (const symbol of [...forbiddenSymbols]) {
  if (server_public_uid.includes(symbol)) {
    throw new Error(`server_public_uid не может содержать символ "${symbol}"!`);
  }
}
let cooked_server_public_uid = "";
let at_counter = 0;
for( const symbol of server_public_uid) {
  if(symbol === '@'){
    at_counter++;
    if(at_counter > 1) throw new Error('server_public_uid не может содержать более одного символа "@"!');
  }
  else{

```

```

    cooked_server_public_uid += symbol;
  }
}

server_public_uid = cooked_server_public_uid;

server_public_uid = server_public_uid.toLowerCase(); //приводим к нижнему регистру
//проверяем, есть ли сервер с таким public_uid

let existingServer = await PostFunctions.getServer_obj({ public_uid: server_public_uid });

if (existingServer) throw new Error('Сервер с таким public_uid уже существует! Пожалуйста,
выберите другой public_uid');

//проверяем, если есть emoji в server_public_uid

const emojiRegex = /\p{Emoji}/u;

if(emojiRegex.test(server_public_uid)) throw new Error('Сервер не может содержать стикеры в
server_public_uid!')

//создаем новый сервер

let generated_server_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();

while ((await database_db.query("SELECT * FROM servers WHERE server_uid = $1",
[generated_server_uid])).rows[0] != undefined) {

    generated_server_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();

}

let created_ISO_date = new Date().toISOString();

let server_metadata = {

    server_uid: generated_server_uid,

    public_uid: server_public_uid,

    metadata: {

        server_name: server_name,

        server_description: "",

        is_private: false, //по умолчанию сервер открытый

        owner_uid: utail_account_uid, //владелец сервера

        members_uids: [utail_account_uid], //владелец сервера

        invites: [],

        created_at: created_ISO_date,

```

```

    banned_uids: [], //идентификаторы, для того, чтобы не дать зайти на сервер
  },
}

//добавляем сервер в базу данных
await database_db.query(
  "INSERT INTO servers (server_uid, public_uid, metadata) VALUES ($1, $2, $3)",
  [generated_server_uid, server_public_uid, server_metadata.metadata]
);

let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
if (!account_obj) throw new Error("Такого аккаунта не существует!");

//добавляем сервер к пользователю, чтобы пользователь мог видеть сервер у себя в интерфейсе
account_obj.metadata.servers.push({
  server_uid: generated_server_uid,
});

await database_db.query(
  'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
  [account_obj.metadata, utail_account_uid]
);

let server_info_to_send_back = {
  ...server_metadata.metadata,
  public_uid: server_metadata.public_uid,
  server_uid: server_metadata.server_uid,
  full_name: server_metadata.metadata.server_name
};

res.status(200).json({
  status: {
    type: "успех",
    message: "данные успешно были получены!"
  },
  delivery: {

```

```

    new_server: server_info_to_send_back,
  }
});
}
catch(e){
  console.error("Ошибка в create_server:", e);
  res.status(200).json({
    status: {
      type: "ошибка",
      message: e.toString(),
    },
  });
}
});
ActionEvents.on("ban_member", async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    // проверяем аргументы, чтобы не совершить sql инъекцию
    if(!req.body || !req.body.arguments) throw new Error('Ошибка формата. Пожалуйста, попробуйте в другой раз');
    const {server_uid, member_uid} = req.body.arguments;
    if(!server_uid || typeof server_uid !== 'string' || server_uid.trim() === "") throw new Error('server_uid не может быть пустым!');
    if(!member_uid || typeof member_uid !== 'string' || member_uid.trim() === "") throw new Error('member_uid не может быть пустым!');
    // \проверяем аргументы, чтобы не совершить sql инъекцию
    let server_obj = await PostFunctions.getServer_obj({server_uid: server_uid});
    if(!server_obj) throw new Error('Такого сервера не существует!')

    //проверяем, если человек является создателем группы
    if(server_obj.metadata.owner_uid === member_uid) throw new Error('Создателя нельзя выкинуть из группы!')
  }
}
});

```

```

//проверяем, если человек является создателем группы, потому что только создатель может
выкидывать людей

if(server_obj.metadata.owner_uid !== utail_account_uid) throw new Error('Только создатель группы
может выкидывать людей!')

server_obj.metadata.members_uids = server_obj.metadata.members_uids.filter(member_uid_arg =>
member_uid_arg !== member_uid);

server_obj.metadata.banned_uids.push(member_uid);

let member_account_obj = await PostFunctions.getAccount_obj({account_uid: member_uid});

member_account_obj.metadata.servers = member_account_obj.metadata.servers.filter(server =>
server.server_uid !== server_uid)

//обновляем состояние

//удаляем у участника сервера

await database_db.query(

  'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',

  [member_account_obj.metadata, member_uid]

);

//удаляем участника из сервера

await database_db.query(

  'UPDATE servers SET metadata = $1 WHERE server_uid = $2',

  [server_obj.metadata, server_uid]

);

res.status(200).json({

  status: {

    type: "успех",

    message: "транзакция успешно выполнена!"

  },

});

}

catch(e){

  console.error("Ошибка в ban_member:", e);

  res.status(200).json({

    status: {

```

```

    type: "ошибка",
    message: e.toString(),
  },
});
}
})
ActionEvents.on("unban_member", async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    // проверяем аргументы, чтобы не совершить sql инъекцию
    if(!req.body || !req.body.arguments) throw new Error('Ошибка формата. Пожалуйста, попробуйте в другой раз');
    const {server_uid, member_uid} = req.body.arguments;
    if(!server_uid || typeof server_uid !== 'string' || server_uid.trim() === "") throw new Error('server_uid не может быть пустым!');
    if(!member_uid || typeof member_uid !== 'string' || member_uid.trim() === "") throw new Error('member_uid не может быть пустым!');
    // \проверяем аргументы, чтобы не совершить sql инъекцию
    let server_obj = await PostFunctions.getServer_obj({server_uid: server_uid});
    if(!server_obj) throw new Error("Такого сервера не существует!")
    server_obj.metadata.banned_uids = server_obj.metadata.banned_uids.filter(member_uid_arg => member_uid_arg !== member_uid);
    //обновляем состояние
    //удаляем участника как забанненого из сервера
    await database_db.query(
      'UPDATE servers SET metadata = $1 WHERE server_uid = $2',
      [server_obj.metadata, server_uid]
    );
    res.status(200).json({
      status: {
        type: "успех",
        message: "транзакция успешно выполнена!"
      }
    });
  }
});
}
})

```

```

    },
  });
}
catch(e){
  console.error("Ошибка в unban_member:", e);
  res.status(200).json({
    status: {
      type: "ошибка",
      message: e.toString(),
    },
  });
}
})
ActionEvents.on("join_server_with_invite", async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    //проверяем аргументы, чтобы не совершить sql инъекцию
    if(!req.body || !req.body.arguments || !req.body.arguments.invite_uid)
      throw new Error('Ошибка формата. Пожалуйста, попробуйте в другой раз');
    if (!req.body.arguments.invite_uid || typeof req.body.arguments.invite_uid !== 'string' ||
req.body.arguments.invite_uid.trim() === "")
      throw new Error('invite_uid не может быть пустым!');
    let invite_uid = req.body.arguments.invite_uid;
    let server_obj = (await database_db.query(
      `SELECT * FROM servers WHERE EXISTS (
      SELECT 1 FROM jsonb_array_elements(metadata->'invites') AS invite
      WHERE invite->>'invite_uid' = $1
      )`,
      [invite_uid]
    )).rows[0];
    if(!server_obj) throw new Error('Такого приглашения не существует!');
  }
}

```

```

let IsBanned = server_obj.metadata.banned_uids.some(member => member === utail_account_uid);
if (IsBanned) throw new Error("Вы были заблокированы на этом сервере!");
let IsMember = server_obj.metadata.members_uids.some(member => member === utail_account_uid);
if(IsMember) throw new Error("Вы уже являетесь участником этого сервера!");
server_obj.metadata.members_uids.push(utail_account_uid);
await database_db.query(
  'UPDATE servers SET metadata = $1 WHERE server_uid = $2',
  [server_obj.metadata, server_obj.server_uid]
);
let account_obj = await PostFunctions.getAccount_obj({account_uid: utail_account_uid});
account_obj.metadata.servers.push({
  server_uid: server_obj.server_uid,
});
await database_db.query(
  'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
  [account_obj.metadata, utail_account_uid]
);
let server_info_to_send_back = {
  ...server_obj.metadata,
  public_uid: server_obj.public_uid,
  server_uid: server_obj.server_uid,
  full_name: server_obj.metadata.server_name
};
res.status(200).json({
  status: {
    type: "успех",
    message: "данные успешно были получены!"
  },
  delivery: server_info_to_send_back
});

```

```

}
catch(e){
  console.error("Ошибка в join_server_with_invite:", e);
  res.status(200).json({
    status: {
      type: "ошибка",
      message: e.toString(),
    },
  });
}
})
ActionEvents.on("create_invite", async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    //проверяем аргументы, чтобы не совершить sql инъекцию
    if(!req.body || !req.body.arguments || !req.body.arguments.server_uid)
      throw new Error('Ошибка формата. Пожалуйста, попробуйте в другой раз');
    if (!req.body.arguments.server_uid || typeof req.body.arguments.server_uid !== 'string' ||
req.body.arguments.server_uid.trim() === "")
      throw new Error('server_uid не может быть пустым!');
    let server_obj = await PostFunctions.getServer_obj({ server_uid: req.body.arguments.server_uid });
    if(!server_obj) throw new Error("Такого сервера не существует!");
    let IsMember = server_obj.metadata.members_uids.some(member => member === utail_account_uid);
    if(!IsMember) throw new Error("Вы не являетесь участником этого сервера!");
    let generated_invite_uid = uuidv4();
    server_obj.metadata.invites.push({
      invite_uid: generated_invite_uid
    })
    await database_db.query(
      'UPDATE servers SET metadata = $1 WHERE server_uid = $2',
      [server_obj.metadata, server_obj.server_uid]

```

```

);
let invite_link = `${process.env.REACT_APP_BACKEND_URL}/invite/${generated_invite_uid}`
res.status(200).json({
  status: {
    type: "успех",
    message: "данные успешно были получены!"
  },
  delivery: invite_link
});
}
catch(e){
  console.error("Ошибка в create_invite:", e);
  res.status(200).json({
    status: {
      type: "ошибка",
      message: e.toString(),
    },
  });
}
})
ActionEvents.on("get_invite_metadata", async (req, res) => {
  try{
    const { utail_account_uid } = req.cookies;
    //проверяем аргументы, чтобы не совершить sql инъекцию
    if(!req.body || !req.body.arguments || !req.body.arguments.uid) throw new Error('Ошибка формата.
Пожалуйста, попробуйте в другой раз');
    if (!req.body.arguments.uid || typeof req.body.arguments.uid !== 'string' ||
req.body.arguments.uid.trim() === "") throw new Error('req.body.arguments.uid не может быть
пустым!');
    const invite_uid = req.body.arguments.uid;
    let server_obj = (await database_db.query(

```

```

`SELECT * FROM servers WHERE EXISTS (
SELECT 1 FROM jsonb_array_elements(metadata->'invites') AS invite
WHERE invite->>'invite_uid' = $1
),
[invite_uid]
)).rows[0];
if (!server_obj) throw new Error('Сервер с таким приглашением не найден!');
let IsMember = server_obj.metadata.members_uids.some(member => member === utail_account_uid);
let IsBanned = server_obj.metadata.banned_uids.some(member => member === utail_account_uid);
if (IsBanned) throw new Error('Вы были заблокированы на этом сервере!');
res.status(200).json({
  status: {
    type: "успех",
    message: "данные успешно были получены!"
  },
  delivery: {
    server_uid: server_obj.server_uid,
    full_name: server_obj.metadata.server_name,
    server_name: server_obj.metadata.server_name,
    public_uid: server_obj.public_uid,
    IsMember: IsMember,
  }
});
} catch(e){
  console.error("Ошибка в get_invite_metadata:", e);
  res.status(200).json({
    status: {
      type: "ошибка",
      message: e.toString(),
    },
  },

```

```

});
}
})
//события /api/get
EventsGet.on("confirm_email_to_create", async (req, res, parsedURL) => {
  let session_uid = parsedURL.query.session_uid;
  let admin_key = parsedURL.query.admin_key;
  let db_table = (await database_db.query("SELECT * FROM sessions WHERE session_uid = $1",
[session_uid])).rows[0];
  if(db_table !== undefined){
    let db_table_metadata_json = db_table.metadata;
    let db_table_account = (await database_db.query("SELECT * FROM accounts WHERE key = $1",
[db_table.key])).rows[0];
    if((new Date()).getTime() < db_table_metadata_json.expires && db_table_metadata_json.status ===
"unauth" && db_table_account === undefined){
      await database_db.query("DELETE FROM sessions WHERE session_uid = $1", [session_uid]);
      let social_role = db_table_metadata_json.social_role;
      // let school_uid = db_table_metadata_json.school_uid;
      let generated_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
      let generated_account_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
      let created = new Date();
      let expires = new Date(created.getTime() + (7 * 24 * 60 * 60 * 1000)); //7 дней
      let role = 'пользователь';
      if(admin_key === secret_admin_key){
        role = 'администратор';
      }
      while((await database_db.query("SELECT * FROM sessions WHERE session_uid = $1",
[generated_uid])).rows[0] !== undefined){
        generated_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
      };
      while((await database_db.query("SELECT * FROM accounts WHERE account_uid = $1",
[generated_account_uid])).rows[0] !== undefined){

```

```

    generated_account_uid = uuidv4() + '-' + uuidv4() + '-' + uuidv4();
};

await database_db.query("INSERT INTO sessions VALUES ($1, $2, $3)", [generated_uid,
db_table.key, {
    "status" : "auth",
    "created" : created.getTime(),
    "expires" : expires.getTime(),
    "owner_uid": generated_account_uid,
}])

const result = await database_db.query('SELECT COUNT(*) FROM accounts');
const max_counter_uid = parseInt(result.rows[0].count) || 0;

// плохой способ получения максимального counter_uid, но он работает
// const result = await database_db.query('SELECT MAX(counter_uid) FROM accounts;');
// const max_counter_uid = result.rows[0].max !== undefined ? result.rows[0].max : 0;

    let accountData = {
        first_name: (db_table_metadata_json.first_name).charAt(0).toUpperCase() +
(db_table_metadata_json.first_name).slice(1).toLowerCase(),
        last_name: (db_table_metadata_json.last_name).charAt(0).toUpperCase() +
(db_table_metadata_json.last_name).slice(1).toLowerCase(),
        role: role,
        social_role: social_role,
        contacts_requests: [],
        contacts: [],
        opened_contacts: [], // открытые контакты, которые находятся слева
        servers: [],
    };

    // if (social_role === 'ученик') {
    //   accountData.school_uid = {
    //     data: school_uid,
    //   };
    // }

    // //выполняем отправку запроса к form_teacher_uid на добавление в базу данных в школу под
учеником

```

```

// }
try {
  await database_db.query(
    "INSERT INTO accounts VALUES ($1, $2, $3, $4, $5)",
    [
      generated_account_uid,
      db_table.key,
      db_table_metadata_json.password,
      accountData,
      'user' + String(max_counter_uid+1),
    ]
  );
} catch (error) {
  console.error("Ошибка при вставке данных в accounts:", error);
}

sendHtml({
  path: role === 'администратор' ? path.join(__dirname, '..', 'static', 'confirm_email_admin.html') :
path.join(__dirname, '..', 'static', 'confirm_email.html'),
  changes: {
    'href="#"' : `href="${process.env.REACT_APP_BACKEND_URL}"`
  },
  cookies: (res) => {
    res.cookie('utail_session_uid', generated_uid, {
      maxAge: 7 * 24 * 60 * 60 * 1000, //7 дней
      secure: true,
      sameSite: 'strict',
      httpOnly: true,
    });
    res.cookie('utail_account_uid', generated_account_uid, {
      maxAge: 7 * 24 * 60 * 60 * 1000, //7 дней
      secure: true,

```

```

    sameSite: 'strict',
    httpOnly: true,
  });
},
res: res,
});
}
else{
  res.status(500).json({ status:
    {
      type: "ошибка",
      message: "запрос БЫЛ ОТКЛОНЕН!"
    },
  });
}
}
else{
  res.status(500).json({ status:
    {
      type: "ошибка",
      message: "запрос БЫЛ ОТКЛОНЕН!"
    },
  });
};
})
module.exports = {EventsGet, EventsPost, createTables}

```

websocket_handlers.js

```

const { PostFunctions } = require('./http_events_modules.js')
const database_db = require('./library/postgresql')

```

```

async function executeHandler(handlerName, messageData, handlers_state) {
  switch (handlerName) {
    case "send_friend_request":
      try {
        const { utail_account_uid } = messageData;

        let contact_public_uid = messageData.body.endpoint_public_uid;
        contact_public_uid = contact_public_uid.replace("@", "");

        if (contact_public_uid.trim() !== "" && contact_public_uid.indexOf(' ') === -1) {
          let contact_account_obj = await PostFunctions.getAccount_obj({ public_uid:
contact_public_uid });

          if (contact_account_obj === undefined || contact_account_obj === null)
            throw new Error('Такого аккаунта не существует!');

          let contact_account_uid = contact_account_obj.account_uid;

          let metadata = contact_account_obj.metadata || {};

          if (!Array.isArray(metadata.contacts_requests)) {
            metadata.contacts_requests = [];
          }

          const alreadyRequested = metadata.contacts_requests.some(
            (req) => req.account_uid === utail_account_uid
          );

          if (alreadyRequested) throw new Error('Вы уже отправили запрос этому пользователю.')

          const alreadyFriend = metadata.contacts.some(
            (req) => req.account_uid === utail_account_uid
          );

          if (alreadyFriend) throw new Error('Вы уже являетесь другом этого пользователя')

          if (contact_account_uid === utail_account_uid) throw new Error('Вы не можете отправить
запрос самому себе! Но уввы...')

          metadata.contacts_requests.push({
            account_uid: utail_account_uid,
          });
        }
      }
    }
  }
}

```

```

//последний этап

let account_obj = await PostFunctions.getAccount_obj({ account_uid: utail_account_uid });
let metadata2 = account_obj.metadata || {};
if (!Array.isArray(metadata2.contacts_requests)) {
    metadata2.contacts_requests = [];
}
const contact_account_is_in_requests = metadata2.contacts_requests.some(
    (req) => req.account_uid === contact_account_uid
);
if (contact_account_is_in_requests) {
    // Добавляем друг друга в друзья
    if (!Array.isArray(metadata.contacts)) metadata.contacts = [];
    if (!Array.isArray(metadata2.contacts)) metadata2.contacts = [];
    // Добавляем друг друга, если ещё нет
    if (!metadata.contacts.some(contact => contact.account_uid === utail_account_uid)) {
        metadata.contacts.push({ account_uid: utail_account_uid });
    }
    // Проверяем наличие contact_account_uid в contacts отправителя
    if (!metadata2.contacts.some(contact => contact.account_uid === contact_account_uid)) {
        metadata2.contacts.push({ account_uid: contact_account_uid });
    }
    //устанавливаем владельцев в историях сообщениях, после чего генерируем
уникальных идентификатор истории сообщений
    // Удаляем запросы из contacts_requests
    metadata.contacts_requests = metadata.contacts_requests.filter(req => req.account_uid !==
utail_account_uid);
    metadata2.contacts_requests = metadata2.contacts_requests.filter(req => req.account_uid
!== contact_account_uid);
    // Обновляем базу данных для обоих пользователей
    await database_db.query(

```

```

        'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
        [metadata, contact_account_uid]
    );
    await database_db.query(
        'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
        [metadata2, utail_account_uid]
    );
    handlers_state.clients.first_client.states.push({
        show_state: true,
        handlerName: handlerName,
        type: "успех",
        message: `вы стали другом: @${contact_account_obj.public_uid}`
    });
    handlers_state.clients.second_client.states.push({
        show_state: true,
        handlerName: handlerName,
        type: "успех",
        message: `вы стали другом: @${account_obj.public_uid}`
    });
}
else {
    await database_db.query(
        'UPDATE accounts SET metadata = $1 WHERE account_uid = $2',
        [metadata, contact_account_uid]
    );
    handlers_state.clients.first_client.states.push({
        show_state: true,
        handlerName: handlerName,
        type: "успех",
        message: "запрос успешно отправлен!"
    });
}

```

```

    });
  }
  return 0;
} else throw new Error('пустые поля или идентификатор с пробелом не допускаются!');
} catch (error) {
  console.warn(`Ошибка: ${error}`);
  handlers_state.clients.first_client.states.push({
    show_state: true,
    handlerName: handlerName,
    type: "ошибка",
    message: `Ошибка: ${error}`
  });
  return -1;
}
break;
case "upload_to_db":
  // логика для загрузки данных в базу данных тут
  break;
case "heartbeat":
  return 0;
  break;
case "kick_procedure":
  try{
    const { utail_account_uid } = messageData;
    const {server_uid} = messageData.body;
    let can_be_trusted = false;
    if (!server_uid || typeof server_uid !== 'string' || server_uid.trim() === "") throw new
Error('server_uid не может быть пустым!');
    let server_obj = await PostFunctions.getServer_obj({server_uid: server_uid});
    if(!server_obj) throw new Error('Такого сервера не существует!')
    if(server_obj.metadata.owner_uid === utail_account_uid){

```

```

        can_be_trusted = true;
    }

    handlers_state.clients.second_client.states.push({
        show_state: false,
        handlerName: handlerName,
        type: "успех",
        message: {
            can_be_trusted: can_be_trusted,
            server_uid: server_uid,
        },
    });
}
catch(e){
    console.error(e);
    return -1;
}
return 0;
break;
case "add_message_to_conversation":
    try {
        const { utail_account_uid } = messageData;
        const contact_account_uid = messageData.body.endpoint_uid;
        const message = messageData.body.message_content;
        // проверка на пустое сообщение
        if (!message || typeof message !== 'string' || message.trim() === "") throw new
Error('Сообщение не может быть пустым!');
        // запрос к базе данных для получения уникального идентификатора переписки
        let conversation_ownership = (await database_db.query(
            `SELECT * FROM owners_of_messages_uids
            WHERE (account_uid1 = $1 AND account_uid2 = $2)

```

```

    OR (account_uid1 = $2 AND account_uid2 = $1)`,
    [utail_account_uid, contact_account_uid]
  )).rows[0];

  let account_obj = await PostFunctions.getAccount_obj({ account_uid: contact_account_uid });
  let server_obj = await PostFunctions.getServer_obj({ server_uid: contact_account_uid });
  if (!account_obj && !server_obj) throw new Error("Такого контакта или сервера не существует!");

  let type = "contact"; //по умолчанию получаем переписку с контактом
  if (server_obj && !account_obj) type = "server";
  if(type === "server"){
    // создаём заглушку, запрет в конкретных ситуациях

    let IsMember = server_obj.metadata.members_uids.some(member => member ===
    utail_account_uid);

    if(!IsMember) throw new Error("Ошибка доступа!");
  }

  //получаем messages_uid и проверяем, есть ли переписка между пользователем и
  КОНТАКТОМ
  if (!conversation_ownership) throw new Error('Переписка не найдена!');
  else {
    messages_uid = conversation_ownership.messages_uid;
  }

  const totalChunksResult = await database_db.query(
    'SELECT COUNT(*) FROM chunks_messages WHERE messages_uid = $1',
    [messages_uid]
  ); //количество всего чанков с одинаковым messages_uid

  const totalChunks = parseInt(totalChunksResult.rows[0].count) || 0; //количество чанков <int>

  let chunk_from_DB = (await database_db.query('SELECT * FROM chunks_messages WHERE
  chunk_uid = $1 AND messages_uid = $2', [totalChunks, messages_uid])).rows[0];

  if (!chunk_from_DB) {
    await database_db.query('INSERT INTO chunks_messages (messages_uid, metadata,

```

```

messages_limit, chunk_uid) VALUES ($1, $2, $3, $4)', [messages_uid, {
    messages: [
        {
            owner_uid: utail_account_uid,
            message: message,
        }
    ],
}, 5, totalChunks + 1]); // 5 - (для теста) лимит сообщений в чанке
}
else {
    if (!chunk_from_DB.metadata.messages) throw new Error('Проблема с метаданными
сообщений в чанке!');
    if (chunk_from_DB.metadata.messages.length >= chunk_from_DB.messages_limit) {
        //создаем новый чанк и добавляем туда сообщение
        await database_db.query('INSERT INTO chunks_messages (messages_uid, metadata,
messages_limit, chunk_uid) VALUES ($1, $2, $3, $4)', [messages_uid, {
            messages: [{
                owner_uid: utail_account_uid,
                message: message,
            }],
        }, 5, totalChunks + 1]); // 5 - (для теста) лимит сообщений в чанке
    }
    else {
        //добавляем сообщение в существующий чанк
        chunk_from_DB.metadata.messages.push({
            owner_uid: utail_account_uid,
            message: message,
        });
        await database_db.query('UPDATE chunks_messages SET metadata = $1 WHERE
messages_uid = $2 AND chunk_uid = $3', [chunk_from_DB.metadata, messages_uid, totalChunks]);
    };
}
};

```

```

}
if(type === "contact") {
  handlers_state.clients.second_client.states.push({
    show_state: false,
    handlerName: handlerName,
    type: "успех",
    message: message
  });
}
else if(type === "server") {
  server_obj.metadata.members_uids.forEach(member_uid => {
    if (member_uid !== utail_account_uid) {
      if(!handlers_state.clients[member_uid]) {
        // если клиента нет в состоянии обработчиков, создаем его
        handlers_state.clients = { ...handlers_state.clients, [member_uid]: {
          states: [],
        }
      };
    }
    // добавляем сообщение в состояние клиента
    handlers_state.clients[member_uid].states.push({
      show_state: false,
      handlerName: handlerName,
      type: "успех",
      message: message,
    });
  }
});
}
return 0;
}

```

```

    catch (e) {
        console.error("Ошибка в add_message_to_conversation:", e);
        return -1;
    };
default:
    console.warn(`Неизвестный обработчик: ${handlerName}`);
    throw new Error(`Неизвестный обработчик: ${handlerName}`);
}
}
module.exports = { executeHandler }

```

postgresql.js

```

const {Pool} = require('pg')
const pool = new Pool({
    user: "root",
    password: "root",
    host: "postgresql_container",
    port: 5432,
    database: "utail_db",
})
module.exports = pool

```

send_email_message.js

```

const fs = require("fs");
async function sendConfirmationEmail(req, transporter, data) {
    return new Promise((resolve, reject) => {
        const htmlPath = data.PathToHtml;
        const subject = data.subject;
        fs.readFile(htmlPath, 'utf8', (err, htmlContent) => {
            if (err) {
                console.error('Error reading HTML file:', err);
                reject(err);
            }
        });
    });
}

```

```

    return;
  }
  if (data.changes !== undefined){
    for(var key in data.changes){
      htmlContent = htmlContent.replace(key, data.changes[key])
    };
  };
  const options = {
    from: 'noreply.utail@gmail.com',
    to: req.body.arguments.email,
    subject: subject,
    html: htmlContent,
  };
  transporter.sendMail(options, (error, info) => {
    if (error) {
      console.error('Error sending email:', error);
      reject(error);
      return;
    }
    console.log('Email sent:', info.response);
    resolve(info);
  });
});
});
}

```

module.exports = sendConfirmationEmail

sendHtml.js

```

const fs = require('fs');
async function sendFile(params) {
  fs.readFile(params.path, 'utf8', (err, htmlContent) => {

```

```

if (err) {
  console.error('Error reading HTML file:', err);
  reject(err); // Reject the promise on error
  return;
}
if (params.changes !== undefined){
  for(var key in params.changes){
    htmlContent = htmlContent.replace(key, params.changes[key])
  };
};
if(params.cookies !== undefined){
  params.cookies(params.res);
}
params.res.writeHead(200, {
  'Content-Type': 'text/html; charset=utf-8',
});
params.res.end(htmlContent, 'utf8');
});
}
module.exports = sendFile;

```

index.js

```

import React from 'react';
import ReactDOM from 'react-dom/client';
//КОМПОНЕНТЫ
import App from './App';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <App />
);

```

App.js

```
import React, { useState, useEffect, useRef } from 'react';
import {BrowserRouter, Navigate, Route, Routes, useLocation } from "react-router-dom";
import './styles/Main.css';
//страницы
import Messenger from './components/Messenger';
import About from './components/About';
import NotFound from './components/NotFound';
import Example from './components/Example';
function App() {
  const location = useLocation();
  useEffect(() => {
    // console.log('Location changed!', location.pathname);
  }, [location]);
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<About />} />
        <Route path="/about" element={<About />} />
        <Route path="/messenger" element={<Messenger />} />
        <Route path="/invite/*" element={<Messenger />} />
        <Route path="/example" element={<Example />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </div>
  );
}
export default function AppWrapper() {
  return (
    <BrowserRouter>
```

```

    <App />
  </BrowserRouter>
);
}

```

main_functions.js

```

async function throwRequest(url = "", data = {}) {
  try {
    const response = await fetch(url, {
      method: "POST",
      mode: "cors",
      cache: "no-cache",
      withCredentials: data.settings && data.settings.withCredentials ? data.settings.withCredentials : false,
      credentials: data.settings && data.settings.credentials ? data.settings.credentials : "same-origin",
      headers: {
        "Content-Type": "application/json",
      },
      redirect: "follow",
      referrerPolicy: "no-referrer",
      body: JSON.stringify(data),
    });
    if (!response.ok) {
      // Сервер вернул код ошибки (400, 500 и т.д.)
      const errorMessage = await response.text(); // Получаем текст ошибки от сервера
      throw new Error(`HTTP error! status: ${response.status}, message: ${errorMessage}`);
    }
    const json = await response.json(); // Попытка распарсить JSON
    return json; // Возвращаем распарсенный JSON
  } catch (error) {
    // Перехват сетевых ошибок, ошибок парсинга JSON и ошибок HTTP
    console.error("Fetch error:", error);
  }
}

```

```

    throw error; // Важно: Перебрасываем ошибку, чтобы ее мог обработать вызывающий код
  }
};
export {
  throwRequest,
}

```

About.js

```

import React, { useState, useEffect } from 'react';
import {Navigate } from "react-router-dom";
import './styles/About.css';

//ассеты
import utail_bar from './html_assets/utail_bar.png'
//block1
import Background_40 from './html_assets/block1/Background_40.png';
import DeviceMacbook_Air from './html_assets/block1/computer/DeviceMacbook_Air.png';
import Line_1_2 from './html_assets/block1/computer/Line_1_2.png';
import Rectangle_45_2 from './html_assets/block1/computer/Rectangle_45_2.png';
import Rectangle_46_2 from './html_assets/block1/computer/Rectangle_46_2.png';
import Rectangle_41_3 from './html_assets/block1/Rectangle_41_3.png';
import Rectangle_43_2 from './html_assets/block1/Rectangle_43_2.png';
//block2
import background1 from './html_assets/block2/background.png'
import Line_1 from './html_assets/block2/Line_1.png'
import Rectangle_45 from './html_assets/block2/Rectangle_45.png'
import Rectangle_46 from './html_assets/block2/Rectangle_46.png'
import Background_header from './html_assets/block2/Background_header.png'
import Background_subtitle from './html_assets/block2/Background_subtitle.png'
//block3
import Background_39 from './html_assets/block3/Background_39.png'

```

```

import Line_2 from '../html_assets/block3/Line_2.png'
import Rectangle_47 from '../html_assets/block3/Rectangle_47.png'
import Rectangle_48_2 from '../html_assets/block3/Rectangle_48_2.png'
import Rectangle_49 from '../html_assets/block3/Rectangle_49.png'
import Rectangle_50 from '../html_assets/block3/Rectangle_50.png'
import Rectangle_51 from '../html_assets/block3/Rectangle_51.png'
import Rectangle_52 from '../html_assets/block3/Rectangle_52.png'
function About() {
  const [page, setPage] = useState('about');
  useEffect(() => {
    document.body.id = `body-${page}`;
  })
  if(page === 'about'){
    return (
      <div className="About">
        <Navigate to={'/'+page}/>
        <div className="background"></div>
        <div className="utail_bar_container">
          <img className="utail_bar" src={utail_bar} alt='utail_bar'/>
        </div>
        <div className="logo"></div>
        <div className='block1'>
          <img src={Background_40} className="Background_40" alt='Background_40'/>
          <img src={DeviceMacbook_Air} className="DeviceMacbook_Air"
alt='DeviceMacbook_Air'/>
          <img src={Line_1_2} className="Line_1_2" alt='Line_1_2'/>
          <img src={Rectangle_45_2} className="Rectangle_45_2" alt='Rectangle_45_2'/>
          <img src={Rectangle_46_2} className="Rectangle_46_2" alt='Rectangle_46_2'/>
          <img src={Rectangle_41_3} className="Rectangle_41_3" alt='Rectangle_41_3'/>
          <img src={Rectangle_43_2} className="Rectangle_43_2" alt='Rectangle_43_2'/>
          <label className="Rectangle_41_3_text1" htmlFor="Rectangle_41_3">Начните безопасное

```

общение прямо сейчас!</label>

<label className="Rectangle_43_2_text1" htmlFor="Rectangle_43_2"> Utail - простой и надёжный способ общения в учебной сфере </label>

</div>

<div className='block2'>

<label className="Background_header_text1"> Создавайте школьные группы</label>

<label className="Background_subtitle_text1"> Хотите собрать одноклассников или хотите объединиться для нового проекта? Создайте сервер или групповой чат! </label>

</div>

<div className='block3'>

<label className="Rectangle_49_text1"> Почему Utail?</label>

<label className="Rectangle_50_text1">

1. Безопасность — защищённые чаты

2. Скорость — моментальная доставка сообщений

3. Простота — интуитивно понятный интерфейс

4. Без рекламы — только общение без лишнего шума

```
</label>
```

```
<label className="Rectangle_52_text1"> Функции </label>
```

```
<label className="Rectangle_51_text1">
```

```
  1. Групповые чаты — общайтесь с друзьями, семьёй и коллегами <br/>
```

```
  2. Темы и настройки — персонализируйте интерфейс под себя <br/>
```

```
  3. Файлы и медиа — обменивайтесь документами и фотографиями без ограничений
```

```
</label>
```

```
</div>
```

```
<button className="enter-btn"
```

```
  onClick={() => {
```

```
    setPage('messenger')
```

```
  }}>
```

```
  Открыть
```

```
</button>
```

```
</div>
```

```
);
```

```
}
```

```
else {
```

```
  return (
```

```
    <Navigate to={'/'+page}/>
```

```
  )
```

```
}
```

```
}
```

```
export default About;
```

BubbleWindows.js

```
import React, { useState, useImperativeHandle, useCallback, useMemo, useRef, useEffect } from 'react';
```

```
import ReactDOM from 'react-dom/client';
```

```
import { throwRequest } from '../modules/main_functions.js';
```

```
import '../styles/BubbleWindows.css'
```

```

function BubbleWindows({ ref, navigate, GoToServer, setServers, socket }) {
  const [windowsArray, setWindowsArray] = useState([]);
  let initedInfo = useRef({});
  const clearWindows = useCallback(() => {
    setWindowsArray(prevArray => {
      if (prevArray.length === 0) return prevArray;
      const newArray = prevArray.map(window => { // Копируем каждый window, чтобы создать
        НОВЫЙ ЭЛЕМЕНТ
        return React.cloneElement(window, { className: 'closed' }); // Создаем новый элемент с классом
        "closed"
      });
      setWindowsArray(newArray); // Обновляем состояние с новым массивом элементов
      setTimeout(() => {
        setWindowsArray([]); // Очищаем массив после задержки
      }, 200);
      return newArray;
    });
  }, []);
  const cached_pages = useMemo(() => ({
    "main": async (data) => {
      //создаём элемент DOM
      const div_container = document.createElement("div");
      return div_container;
    },
    "members": async (data) => {
      //создаём элемент DOM
      const div_container = document.createElement("div");
      div_container.className = "members";
      div_container.style.display = "flex";
      div_container.style.flexDirection = "column";
      div_container.style.width = "100%";
    }
  }));
}

```

```

div_container.style.height = "100%";

//получаем участников из базы данных

let members = []

try {

  members = (await throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "user_gate",
    arguments: { user_action: "get_server_members", server_uid: data.server_uid }
  })).delivery || [];

}

catch (e) {

  console.log(e);

};

// парсим участников и вставляем их в родительский DOM контейнер

members.forEach(member => {

  const member_container = document.createElement("div");

  member_container.className = "member_container";

  member_container.style.display = "flex";

  member_container.style.flexDirection = "row";

  member_container.style.width = "80%";

  member_container.style.height = "65px";

  member_container.style.alignItems = "center";

  member_container.style.backgroundColor = "#AF8A9D";

  member_container.style.marginTop = "5px";

  member_container.style.marginBottom = "5px";

  member_container.style.marginLeft = "10px";

  member_container.style.borderRadius = "20px";

  member_container.setAttribute("account_uid", member.account_uid)

  const contact_avatar = document.createElement("div");

  contact_avatar.id = "friend_avatar";

  contact_avatar.style.border = "solid 2px #633A4F";

```

```

contact_avatar.className = "loading_chunk";
const shine = document.createElement("div");
shine.id = "shine";
contact_avatar.appendChild(shine);
const member_info_container = document.createElement("div");
member_info_container.style.display = "flex";
member_info_container.style.flexDirection = "column";
member_info_container.style.width = "auto";
member_info_container.style.height = "100%";
member_info_container.style.justifyContent = "center";
const public_name = document.createElement("label");
public_name.innerHTML = member.public_name;
public_name.style.fontFamily = "'Montserrat', sans-serif";
public_name.style.fontWeight = "bold";
const public_uid = document.createElement("label");
public_uid.innerHTML = `@${member.public_uid}`;
public_uid.style.fontFamily = "'Montserrat', sans-serif";
public_uid.style.color = "#61F8FF";
member_info_container.appendChild(public_name);
member_info_container.appendChild(public_uid);
member_container.appendChild(contact_avatar);
member_container.appendChild(member_info_container);
if (data.owner_uid !== member.account_uid) {
  const kick_button = document.createElement("button");
  kick_button.id = "kick_btn";
  kick_button.style.marginLeft = "auto";
  kick_button.style.marginRight = "10px";
  kick_button.addEventListener("click", () => {
    let container = document.querySelector(`[account_uid="${member.account_uid}"]`);
    //ставим контейнеру загрузку, что он собирается удалиться

```

```

kick_button.remove();

const loader = document.createElement("div");

loader.className = "loader custom_2"

loader.style.marginLeft = "auto";

loader.style.marginRight = "15px";

member_container.appendChild(loader);

throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
  action: "user_gate",
  arguments: { user_action: "ban_member", server_uid: data.server_uid, member_uid:
member.account_uid }
}).then((res_data) => {
  if (res_data.status.type === 'успех') {
    //если fetch сработал то удаляем этот объект
    if (container) {
      container.remove();
    }
    //сообщаем конечному пользователю о удалении
    socket.send(JSON.stringify({
      settings: {
        send_message_to_endpoint: true,
        handlers: [{ handler: "kick_procedure", condition: "before the sending message" }]
      },
      body: {
        server_uid: data.server_uid,
        endpoint_uid: member.account_uid,
        message: "kick_procedure"
      }
    }));
  }
})
})

```

```

    member_container.appendChild(kick_button);
  }
  else {
    const indicator_owner = document.createElement("label");
    indicator_owner.id = "indicator_owner";
    indicator_owner.style.marginLeft = "auto";
    indicator_owner.style.marginRight = "10px";
    indicator_owner.innerHTML = "создатель группы"
    indicator_owner.style.fontFamily = "'Montserrat', sans-serif";
    member_container.appendChild(indicator_owner);
  }
  div_container.appendChild(member_container);
});
return div_container;
},
"ban_manager": async (data) => {
  //создаём элемент DOM
  const div_container = document.createElement("div");
  div_container.className = "members";
  div_container.style.display = "flex";
  div_container.style.flexDirection = "column";
  div_container.style.width = "100%";
  div_container.style.height = "100%";
  //получаем участников из базы данных
  let members = []
  try {
    members = (await throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
      action: "user_gate",
      arguments: { user_action: "get_banned_members", server_uid: data.server_uid }
    })).delivery || [];
  }

```

```

}
catch (e) {
  console.log(e);
};

// парсим участников и вставляем их в родительский DOM контейнер
members.forEach(member => {
  const member_container = document.createElement("div");
  member_container.className = "member_container";
  member_container.style.display = "flex";
  member_container.style.flexDirection = "row";
  member_container.style.width = "80%";
  member_container.style.height = "65px";
  member_container.style.alignItems = "center";
  member_container.style.backgroundColor = "#AF8A9D";
  member_container.style.marginTop = "5px";
  member_container.style.marginBottom = "5px";
  member_container.style.marginLeft = "10px";
  member_container.style.borderRadius = "20px";
  member_container.setAttribute("account_uid", member.account_uid)
  const contact_avatar = document.createElement("div");
  contact_avatar.id = "friend_avatar";
  contact_avatar.style.border = "solid 2px #633A4F";
  contact_avatar.className = "loading_chunk";
  const shine = document.createElement("div");
  shine.id = "shine";
  contact_avatar.appendChild(shine);
  const member_info_container = document.createElement("div");
  member_info_container.style.display = "flex";
  member_info_container.style.flexDirection = "column";
  member_info_container.style.width = "auto";

```

```

member_info_container.style.height = "100%";
member_info_container.style.justifyContent = "center";
const public_name = document.createElement("label");
public_name.innerHTML = member.public_name;
public_name.style.fontFamily = "'Montserrat', sans-serif";
public_name.style.fontWeight = "bold";
const public_uid = document.createElement("label");
public_uid.innerHTML = `@${member.public_uid}`;
public_uid.style.fontFamily = "'Montserrat', sans-serif";
public_uid.style.color = "#61F8FF";
member_info_container.appendChild(public_name);
member_info_container.appendChild(public_uid);
member_container.appendChild(contact_avatar);
member_container.appendChild(member_info_container);
if (data.owner_uid !== member.account_uid) {
  const kick_button = document.createElement("button");
  kick_button.id = "kick_btn";
  kick_button.style.marginLeft = "auto";
  kick_button.style.marginRight = "10px";
  kick_button.addEventListener("click", () => {
    let container = document.querySelector(`[account_uid="${member.account_uid}"]`);
    //ставим контейнеру загрузку, что он собирается удалиться
    kick_button.remove();
    const loader = document.createElement("div");
    loader.className = "loader custom_2"
    loader.style.marginLeft = "auto";
    loader.style.marginRight = "15px";
    member_container.appendChild(loader);
    throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
      action: "user_gate",

```

```

arguments: { user_action: "unban_member", server_uid: data.server_uid, member_uid:
member.account_uid }

}).then((res_data) => {
  if (res_data.status.type === 'успех') {
    //если fetch сработал то удаляем этот объект
    if (container) {
      container.remove();
    }
  }
})
})

member_container.appendChild(kick_button);
}

else {
  const indicator_owner = document.createElement("label");
  indicator_owner.id = "indicator_owner";
  indicator_owner.style.marginLeft = "auto";
  indicator_owner.style.marginRight = "10px";
  indicator_owner.innerHTML = "создатель группы"
  indicator_owner.style.fontFamily = "'Montserrat', sans-serif";
  member_container.appendChild(indicator_owner);
}

div_container.appendChild(member_container);
});

return div_container;
}

}), []);

const __GoToPage = useCallback(async (new_page, e) => {
  let panel_menu = document.getElementById("panel_menu");
  if (!panel_menu) {
    // Попробуем снова через короткий промежуток времени

```

```

setTimeout(() => __GoToPage(new_page), 0);

return;
};

const container = document.getElementById("dark_panel");
if (container) {
    const buttons = container.querySelectorAll("button");
    buttons.forEach(button => {
        button.classList.remove("selected");
    });
} else {
    console.warn("Контейнер не найден.");
}
if(e) e.target.classList.add("selected");
//очищаю страницу
while (panel_menu.firstChild) panel_menu.firstChild.remove();
const div_container = document.createElement("div");
div_container.style.width = "100%";
div_container.style.height = "100%";
div_container.style.display = "flex";
div_container.style.justifyContent = "center"
div_container.style.alignContent = "center";
div_container.style.alignItems = "center";
const loader = document.createElement("div");
loader.className = "loader custom_2"
div_container.appendChild(loader);
panel_menu.appendChild(div_container);
//вставляю страницу
let data = initedInfo.current["server_settings"];
let node = await cached_pages[new_page](data)
panel_menu.appendChild(node);

```

```

    div_container.remove();
  }, []);
useEffect(() => {
  __GoToPage("main");
}, []);
const cached_windows = {
  "invite_panel": (
    <div id="invite_panel">
      <div
        style={{
          backgroundColor: "#442233",
          width: "500px",
          height: "300px",
          borderRadius: "20px",
          boxShadow: "5px 5px 15px 5px rgba(0,0,0,0.64)"
        }}
      >
        <div
          id="invite_countainer"
          style={{
            width: "100%",
            height: "100%",
            display: "flex",
            flexDirection: "column",
            alignItems: "center",
            justifyContent: "center"
          }}
        >
          <button
            id="generate_invite-btn"

```

```

onClick={async (event) => {
  if (
    event.target.className === "loading" ||
    event.target.className === "loader"
  )
    return;
  event.target.className = "loading";
  event.target.innerHTML =
    "<div class='loader' style='margin-left: 45%'></div>";
  let server_uid = initInfo.current["invite_panel"].server_uid;
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "user_gate",
    arguments: { user_action: "create_invite", server_uid: server_uid }
  }).then((data) => {
    if (data.status.type === "успех") {
      try {
        event.target.className = "";
        event.target.remove();
        let invite_link = data.delivery;
        let text_link = document.getElementById("invite_link");
        text_link.innerHTML = invite_link;
        let invite_countainer = document.getElementById(
          "invite_countainer"
        );
        const copyBtn = document.createElement("button");
        copyBtn.id = "copy-btn"
        copyBtn.textContent = "скопировать ссылку";
        copyBtn.onclick = async () => {
          try {
            await navigator.clipboard.writeText(invite_link);

```

```

        copyBtn.textContent = "ссылка скопирована!";
        setTimeout(() => {
            copyBtn.textContent = "скопировать ссылку";
        }, 1000);
    } catch (err) {
        copyBtn.textContent = "ошибка";
    }
};

invite_container.appendChild(copyBtn);
} catch (e) {
    console.log(e);
}
});
})
}}
>
    сгенерировать ссылку
</button>
<label style={{
    fontFamily: "'Montserrat', sans-serif"
}} id="invite_link"></label>
</div>
</div>
<button id="close-btn" onClick={() => {
    clearWindows();
    navigate('/messenger')
}}></button>
</div>
),
"server_settings": (

```

```

<div id="panel">
  <button id="close-btn" onClick={() => {
    clearWindows();
    navigate('/messenger')
  }}></button>
  <div id="dark_panel">
    <div id="buttons_menu" style={{
      width: "250px",
      height: "100%",
      position: "relative",
      display: "flex",
      flexDirection: "column"
    }}>
      <label id="server_name_indicator">Тут будет название сервера!</label>
      <div className='menu_separator'></div>
      <button onClick={(e) => {
        __GoToPage("members", e)
      }}>участники</button>
      <button onClick={(e) => {
        __GoToPage("invites_manager", e)
      }}>приглашения</button>
      <button onClick={(e) => {
        __GoToPage("ban_manager", e)
      }}>лист заблокированных</button>
      <div className='menu_separator'></div>
    </div>
  </div>
  <div id="panel_menu">
  </div>
</div>

```

```

),
"invite_request": (
  <div id="invite_request">
    <button id="close-btn" onClick={() => {
      clearWindows();
      navigate('/messenger')
    }}></button>
    <div
      style={{
        backgroundColor: "#442233",
        width: "500px",
        height: "300px",
        borderRadius: "20px",
        boxShadow: "5px 5px 15px 5px rgba(0,0,0,0.64)"
      }}
    >
    <div
      id="invite_request_container"
      style={{
        width: "100%",
        height: "100%",
        display: "flex",
        flexDirection: "column",
        alignItems: "center",
        justifyContent: "center"
      }}
    >
    <div className="loader custom_2" id="invite-loader"></div>
  </div>
</div>

```

```

    </div>
  ),
};
const addWindow = useCallback((window_name, data) => {
  setWindowsArray(prevArray => [
    ...prevArray,
    cached_windows[window_name]
  ]);
  initedInfo.current[window_name] = data;
}, []);
const invoke_invite_error = useCallback(() => {
  let element = document.getElementById("invite-loader");
  if (!element) {
    setTimeout(() => invoke_invite_error(), 0);
    return;
  }
  element.remove();
  let invite_request_container = document.getElementById("invite_request_countainer");
  const label = document.createElement("label");
  label.style.fontFamily = "'Montserrat', sans-serif";
  label.textContent = "Упс! Что-то пошло не так...";
  invite_request_container.appendChild(label);
}, []);
const invoke_invite_success = useCallback((data) => {
  let invite_request_container = document.getElementById("invite_request_countainer");
  if (!invite_request_container) {
    setTimeout(() => invoke_invite_success(data), 0);
    return;
  }
}

```

```

let element = document.getElementById("invite-loader");
if (!element) {
  setTimeout(() => invoke_invite_success(data), 0);
  return;
}
element.remove();
const server_name_label = document.createElement("label");
server_name_label.style.fontFamily = "'Montserrat', sans-serif";
server_name_label.textContent = data.server_name;
server_name_label.style.fontWeight = '800';
server_name_label.style.fontSize = '25px'
const public_uid_label = document.createElement("label");
public_uid_label.style.fontFamily = "'Montserrat', sans-serif";
public_uid_label.style.color = "#5FC3FF";
public_uid_label.textContent = `@${data.public_uid}`;
public_uid_label.style.fontSize = '20px'
invite_request_container.appendChild(server_name_label);
invite_request_container.appendChild(public_uid_label);
if (data?.IsMember) {
  //создаём кнопку, чтобы перейти на сервер, то есть нужно закрыть окно и открыть сервер
  let GotoButton = document.createElement("button")
  GotoButton.textContent = "Перейти на сервер";
  GotoButton.style.backgroundColor = "#7D3C98";
  GotoButton.style.color = "#fff";
  GotoButton.style.fontFamily = "'Montserrat', sans-serif";
  GotoButton.style.border = "none";
  GotoButton.style.borderRadius = "8px";
  GotoButton.style.padding = "12px 32px";
  GotoButton.style.fontSize = "18px";
  GotoButton.style.fontWeight = "600";

```

```

GotoButton.style.cursor = "pointer";
GotoButton.style.marginTop = "24px";
GotoButton.onclick = () => {
  navigate('/messenger');
  GoToServer(data);
  clearWindows();
  //переходим в группу
};
invite_request_container.appendChild(GotoButton);
}
else {
  //создаём кнопку, чтобы войти в сервер по приглашению
  let GotoButton = document.createElement("button")
  GotoButton.textContent = "принять приглашение";
  GotoButton.style.backgroundColor = "#50df25ff";
  GotoButton.style.color = "#fff";
  GotoButton.style.fontFamily = "'Montserrat', sans-serif";
  GotoButton.style.border = "none";
  GotoButton.style.borderRadius = "8px";
  GotoButton.style.padding = "12px 32px";
  GotoButton.style.fontSize = "18px";
  GotoButton.style.fontWeight = "600";
  GotoButton.style.cursor = "pointer";
  GotoButton.style.marginTop = "24px";
  GotoButton.onclick = () => {
    GotoButton.remove();
    let loader_element = document.createElement("div")
    loader_element.className = "loader"
    invite_request_container.appendChild(loader_element);
  }
}

```

```

throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
  action: "user_gate",
  arguments: { user_action: "join_server_with_invite", invite_uid: data.invite_uid }
}).then((data) => {
  if (data.status.type === "успех") {
    try {
      //получаем сервер и добавляем в списки серверов
      //сервер должен добавить нас как участника сервера
      setServers(prevServers => [...prevServers, data.delivery]);
      navigate('/messenger');
      GoToServer(data.delivery);
      clearWindows();
    } catch (e) {
      console.log(e);
    }
  };
})
);
invite_request_container.appendChild(GotoButton);
}
}, []);
const initServerPanel = useCallback(() => {
  let server_name_indicator = document.getElementById("server_name_indicator");
  if (!server_name_indicator) {
    setTimeout(() => initServerPanel(), 0);
    return;
  };
  let data = initedInfo.current["server_settings"];
  server_name_indicator.innerHTML = data.full_name;
})

```

```

// Предоставляем родительскому компоненту
useImperativeHandle(ref, () => ({
  addWindow: addWindow,
  clearWindows: clearWindows,
  invoke_invite_error: invoke_invite_error,
  invoke_invite_success: invoke_invite_success,
  initServerPanel: initServerPanel,
}), [addWindow, clearWindows]);
return (
  <div className="BubbleWindows">
    {windowsArray.map((window, index) => (
      <React.Fragment key={index}>{window}</React.Fragment>
    ))}
  </div>
);
};
export default BubbleWindows;

```

CreateServerPanel.js

```

import React, {useState, useEffect, useRef} from 'react';
import { throwRequest } from '../modules/main_functions.js';
import '../styles/CreateServerPanel.css'
const useOutsideClick = (callback) => {
  const ref = useRef();
  useEffect(() => {
    const handleClick = (event) => {
      if (ref.current && !event.target.closest('.white-window') && event.target.closest('.opened') &&
event.target.closest('#server_panel_group')) {
        callback(event);
      }
    };
  });
};

```

```

document.addEventListener('click', handleClick);

return () => {
  document.removeEventListener('click', handleClick);
};
}, [ref, callback]);

return ref;
};

function CreateServerPanel({createServerPanel, setCreateServerPanel, setServers}) {
  // панель создания серверов
  const defaultServerPanelData = {
    current_page: 'methods',
    previous_pages: [],
    server_form: {
      server_name: "",
      server_public_uid: ""
    },
    error: null
  };

  const [serverPanelData, setServerPanelData] = useState(defaultServerPanelData);
  const serverPanelDataRef = useRef(serverPanelData); // useRef для selectedContact
  useEffect(() => {
    serverPanelDataRef.current = serverPanelData; // Обновляем useRef при изменении
selectedContact
  }, [serverPanelData]);

  const [settings_are_closed, setSettingsAreClosed] = useState(false);
  const [theme_is_closed, setThemeIsClosed] = useState(false);

  const resetEverything = () => {
    setCreateServerPanel(false);
    setTimeout(() => {
      setServerPanelData(defaultServerPanelData);

```

```

    }, 500);
  }
  const ref = useOutsideClick((event) => {
    resetEverything();
  });
  const gotoPage = (page, data) => {
    setSettingsAreClosed(true);
    setThemeIsClosed(true);
    setTimeout(() => {
      let current_data = serverPanelData;
      current_data.current_page = page;
      current_data.previous_pages.push(page);
      if(data && data.error) {
        current_data.error = data.error || null;
      } else {
        current_data.error = null;
      }
      setServerPanelData(current_data);

      setSettingsAreClosed(false);
      setThemeIsClosed(false);
    }, 500);
  }
  return (
    <div id="server_panel_group" className={createServerPanel ? 'opened' : 'closed'} ref={ref}>
      <div id="create_server_panel" className={createServerPanel ? 'opened white-window' : 'closed white-window'} >
        <div id="settings" className={settings_are_closed ? 'closed' : 'opened'}>
          {/* тут будут кнопки и различные настройки */}
          {serverPanelData.current_page === 'methods' &&
            <button onClick={() => {

```

```

        gotoPage('server');
    }}>Сервер для себя или коммерции</button>
}
{serverPanelData.current_page === 'server' &&
  <div id="server_create_form" style={{display: 'flex', flexDirection: 'column'}}>
    <input placeholder='введите название' onChange={(e) => {
      setServerPanelData(prevData => ({
        ...prevData,
        server_form: {
          ...prevData.server_form,
          server_name: e.target.value
        }
      }));
    }} style={{width: '80%', paddingLeft: '5px', marginLeft: '8%',height: '15px',
borderRadius: '10px', border: 'none', outline: 'none', fontSize: '6px', backgroundColor: '#987185', color:
'white'}} type="text"></input>
    <input placeholder='пример: @МБОУ_СШ_№5' onChange={(e) => {
      setServerPanelData(prevData => ({
        ...prevData,
        server_form: {
          ...prevData.server_form,
          server_public_uid: e.target.value
        }
      }));
    }} style={{width: '80%', paddingLeft: '5px', marginTop: '8px', marginLeft:
'8%',height: '15px', borderRadius: '10px', border: 'none', outline: 'none', fontSize: '6px', backgroundColor:
'#987185', color: 'white'}} type="text"></input>
  </div>
}
{serverPanelData.current_page === 'server' &&
  <button className='create_btn_from_form' onClick={() => {
    gotoPage('loading');

```

```

        throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
            action: "user_gate",
            arguments: {user_action: "create_server", form:
serverPanelDataRef.current.server_form}
        }).then((data) => {
            setTimeout(() => {

if(serverPanelDataRef.current.previous_pages[serverPanelDataRef.current.previous_pages.length - 1] !==
'loading') return; //отмена если человек не захотел ждать

                if(data.status.type === "успех"){
                    setServers(prevServers => [...prevServers, data.delivery.new_server]);
                    resetEverything()
                }
                else{
                    gotoPage('error', {error: data.status.message});
                };
            }, 1000);
        })
    }}>создать</button>
}

{serverPanelData.current_page === 'server' &&
    <button onClick={() => {
        gotoPage('methods');
    }}>Вернуться к выбору метода создания</button>
}

{serverPanelData.current_page === 'loading' &&
    <div className="loader" style={{marginLeft: '40%'}}></div>
}

{serverPanelData.current_page === 'methods' &&
    <button onClick={() => {

```

```

        gotoPage('loading')
        setTimeout(() => {

if(serverPanelDataRef.current.previous_pages[serverPanelDataRef.current.previous_pages.length - 1] !==
'loading') return; //отмена если человек не захотел ждать

            gotoPage('teacher_server');
            }, 5000)
        }}>Вы - учитель? Мы поможем вам создать группу!</button>
    }
    {serverPanelData.current_page === 'error' &&
        <button onClick={() => {
            gotoPage('methods');
        }}>вернуться назад</button>
    }
</div>

<div id="theme" className={theme_is_closed ? 'closed' : 'opened'}>
    /* тут будет описание того, что ты выбрал */
    {serverPanelData.current_page === 'methods' &&
        <label style={{fontFamily: "'Montserrat', sans-serif", fontWeight: '500', color: 'white',
fontSize: '10px', textAlign: 'center', marginLeft: '4px'}}>Как вы хотите создать сервер?</label>
    }
    {serverPanelData.current_page === 'server' &&
        <label style={{fontFamily: "'Montserrat', sans-serif", fontWeight: '500', color: 'white',
fontSize: '10px', textAlign: 'center', marginLeft: '4px'}}>настройте свой сервер!</label>
    }
    {serverPanelData.current_page === 'Бислан' &&
        <label style={{fontFamily: "'Montserrat', sans-serif", fontWeight: '500', color: 'white',
fontSize: '10px', textAlign: 'center', marginLeft: '4px'}}>Бислан сидит в тихаря и наяривает</label>
    }
    {serverPanelData.current_page === 'loading' &&
        <div className="loader" style={{marginLeft: '40%'}}></div>
    }
}

```

```

        {serverPanelData.current_page === 'error' &&
          <label style={{fontFamily: "'Montserrat', sans-serif", fontWeight: '500', color: 'white',
fontSize: '10px', textAlign: 'center', marginLeft: '4px'}} dangerouslySetInnerHTML={{__html: `:(
${serverPanelDataRef.current.error}`}}></label>
        }
      </div>
    </div>
  </div>
)
}
export default CreateServerPanel;

```

Dropdown_menu.js

```

import React, {useState, useEffect, useRef} from 'react';
import './styles/Dropdown_menu.css'
import { throwRequest } from './modules/main_functions.js';
const useOutsideClick = (callback) => {
  const ref = useRef();
  useEffect(() => {
    const handleClick = (event) => {
      if (ref.current && !ref.current.contains(event.target) && !event.target.closest('.white-window')) {
        callback(event);
      }
    };
    document.addEventListener('click', handleClick);
    return () => {
      document.removeEventListener('click', handleClick);
    };
  }, [ref, callback]);
  return ref;
};
function DropdownMenu({dropdown_text, result}) {

```

```

let [final_className, setFinal_className] = useState("")
let [clicked_state, setClicked_state] = useState(false)
let [letters_container_active, setLetters_container_active] = useState(false)
let [letters_container2_active, setLetters_container2_active] = useState(false)
let [numberClass, setNumberClass] = useState(null)
let [letterClass, setLetterClass] = useState(null)
let [letterClass2, setLetterClass2] = useState(null)
const [data, setData] = useState([]);
const [data2, setData2] = useState([]);
const [data3, setData3] = useState([]);
const whiteWindowRef = useRef(null);
const handleClickOutside = (event) => {
  setClicked_state(false);
  setLetters_container_active(false)
  setNumberClass(null)
  setLetterClass(null)
  setLetters_container2_active(false)
};
const ref = useOutsideClick(handleClickOutside);
const handleClickDropdownNumObj = (item) => {
  setNumberClass(item)
  setLetters_container_active(true)
  setLetterClass(null)
  setLetterClass2(null);
  setLetters_container2_active(false);
  //тут я должен отправить запрос на сервер для получения всех классов ПО КЛЮЧУ item
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, { action: "get_grades",
arguments: { get_grades_by_obj: "grade", argument2: "???", school: item },
  settings: {
    withCredentials: true
  }})
}

```

```

.then((data) => {
  setData2(data.delivery.grades)
})
.catch((error) => {
  console.error("Error in throwRequest:", error);
});
};

const handleClickDropdownGradeObj = (item) => {
  if(numberClass !== null){
    setLetterClass(item);
    setLetters_container2_active(true)
    //тут я должен отправить запрос на сервер для получения всех классов ПО КЛЮЧУ item
    throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, { action: "get_grades",
arguments: { get_grades_by_obj: "letter", argument2: item, school: numberClass},
  settings: {
    withCredentials: true
  }})
  .then((data) => {
    setData3(data.delivery.letters)
  })
  .catch((error) => {
    console.error("Error in throwRequest:", error);
  });
}
};

const handleClickDropdownLetterObj = (item) => {
  setLetterClass2(item);
  result({school: numberClass, grade: letterClass, letter: item})
  setClicked_state(false)
  setLetters_container_active(false)
  setNumberClass(null)
}
};

```

```

setLetterClass(null)

setLetterClass2(null);

setLetters_container2_active(false)

setFinal_className(numberClass + '' + letterClass + item)

};

return (
  <div id="Dropdown_menu_container">
    <button id="dropdown_menu1" className={clicked_state ? 'active' : ''} ref={ref} onClick={() =>
{
  setClicked_state(!clicked_state)
  setLetters_container_active(false)
  setNumberClass(null)
  setLetterClass(null)
  setLetterClass2(null);
  setLetters_container2_active(false)
  setFinal_className("")
  result({school: null, grade: null, letter: null})
  if(clicked_state===false){
    //получаем школы
    throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, { action:
"get_grades", arguments: { get_grades_by_obj: "school", argument2: "???", school: "???" },
    settings: {
      withCredentials: true
    }
  })
    .then((data) => {
      setData(data.delivery.schools)
    })
    .catch((error) => {
      console.error("Error in throwRequest:", error);
    });
  }
}

```

```

    }}> {final_className != " ? final_className : dropdown_text} </button>
    <div id="Dropdown_menu_content_container" className="white-window"
ref={whiteWindowRef}>
    <div id="numbers_container" className={clicked_state ? " : 'closed'}>
        {clicked_state && (
            <div>
                {data.map((item, index) => (
                    <button key={index} className='dropdown_object' id={item === numberClass ?
'selected' : ""} onClick={()=>{
                        handleClickDropdownNumObj(item)
                    }}> {item} </button>
                ))}
            </div>
        )}
    </div>
    <div id="letters_container" className={letters_container_active ? " : 'closed'}>
        {clicked_state && numberClass != null && (
            <div>
                {data2.map((item, index) => (
                    <button key={index} className='dropdown_object' id={item === letterClass ?
'selected' : ""} onClick={()=>{
                        handleClickDropdownGradeObj(item)
                    }}> {item + " клас"} </button>
                ))}
            </div>
        )}
    </div>
    <div id="letters_container2" className={letters_container2_active ? " : 'closed'}>
        {letters_container2_active && numberClass != null && (
            <div>
                {data3.map((item, index) => (

```

```

        <button key={index} className='dropdown_object' id={item === letterClass2 ?
'selected' : ''} onClick={()=>{
            handleClickDropdownLetterObj(item)
        }}> {item} </button>
    )}
</div>
    )}
</div>
</div>
</div>
)
}
export default DropdownMenu

```

Example.js

```

import React from 'react';
function About() {
    return (
        <div className="About">
            <p>
                привет! Это всего лишь тестирование! - Дмитрий Кузнецов
            </p>
        </div>
    );
}
export default About;

```

Messenger.js

```

import React, { useState, useEffect, useLayoutEffect, useRef } from 'react';
import { useLocation, useNavigate } from 'react-router-dom';
import { throwRequest } from '../modules/main_functions.js';
import '../styles/Messenger.css';
import '../styles/LoadingChunk.css'

```

```

import Dropdown_menu from './Dropdown_menu'
import Notification from './Notification';
import Switcher from './Switcher';
import CreateServerPanel from './CreateServerPanel';
import UserSettings from './UserSettings';
import ServerSettings from './ServerSettings';
import BubbleWindows from './BubbleWindows';
function Messenger() {
  const [first_name, setFirst_name] = useState("");
  const [last_name, setLast_name] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [social_role, setSocial_role] = useState('ученик');
  const [friend_for_request, setFriend_for_request] = useState("");
  const [visible, setVisible] = useState(false);
  const [authState, setAuthState] = useState('loading'); // 'loading', 'sign-in', 'sign-up', 'authenticated'
  const [notificationMessage, setNotificationMessage] = useState("");
  const [notificationColor, setNotificationColor] = useState('usual');
  const [notificationPosition, setNotificationPosition] = useState('bottom');
  const [user_metadata, setUser_metadata] = useState(null)
  const [resolved_contacts, setResolved_contacts] = useState(null)
  const [resolved_open_contacts, setResolved_open_contacts] = useState(null)
  const [selected_contact, setSelected_contact] = useState(null);
  const [contacts_requests, setContacts_requests] = useState([]);
  const [servers, setServers] = useState(null);
  const serversRef = useRef(servers);
  useEffect(() => {
    serversRef.current = servers;
  }, [servers]);

```

```

const location = useLocation();
const navigate = useNavigate();
const selectedContactRef = useRef(selected_contact); // useRef для selectedContact
useEffect(() => {
  selectedContactRef.current = selected_contact; // Обновляем useRef при изменении selectedContact
}, [selected_contact]);

const [current_chunk, setCurrent_chunk] = useState(0); // это текущий чанк сообщений, который мы
получаем из бэкенда

const conversation_ref = useRef(null); // это реф для скролла, чтобы мы могли прокручивать чат
вниз

const [messages, setMessages] = useState(null); // это массив сообщений, которые будут
отображаться в чате

const messagesRef = useRef(messages); // useRef для messages
useEffect(() => {
  messagesRef.current = messages; // Обновляем useRef при изменении selectedContact
}, [messages]);

const [scroll_pusher_style, setScroll_pusher_style] = useState({}); // это стиль для скролла, чтобы он
был внизу

let [final_name, setFinal_name] = useState(null)
const [socket, setSocket] = useState(null);
const observerRef = useRef(null);

const [written_message, setWritten_message] = useState(""); // это состояние для ввода сообщения в
чат

const handleCloseNotification = () => {
  setNotificationMessage("");
};

//обработчики контактов
let [subpage, setSubpage] = useState({
  subpage: 'home',
  menu1: 'contacts',
  menu2: {
    primary: 'friends',

```

```

    secondary: 'all_friends'
  }
});
const handleContactClick = (e) => {
  const target = e.target.closest('#contact');
  if (target) {
    setSelected_contact({ uid: target.getAttribute('id2'), type: target.getAttribute('type') });
    setSubpage({
      ...subpage,
      menu2: {
        ...subpage.menu2,
        primary: 'chat',
      }
    });
  }
}
// обработчики\
const handleAcceptfriendrequest = (e) => {
  const target = e.target.closest('#accept_btn');
  if (target) {
    const contact = target.parentElement;
    const public_uid = contact.querySelector('#friend_public_uid').textContent.replace('@', "");
    //удаляем локально
    let contacts_requests_copy = contacts_requests;
    contacts_requests_copy = contacts_requests_copy.filter(req => {
      if (req.public_uid !== public_uid) {
        return true
      }
    }
    else {
      navigator.sendBeacon(`${process.env.REACT_APP_BACKEND_URL}/api/beacon_drain`,
JSON.stringify({

```

```

    action: "user_gate",
    arguments: { user_action: "accept_contact_request", contact_account_uid: req.account_uid }
  )))

  //req.account_uid - это уникальный идентификатор аккаунта который будет твоим другом :з
  return false
};

})

setContacts_requests(contacts_requests_copy);
}
}

const handleDeclinefriendrequest = (e) => {
  const target = e.target.closest('#decline_btn');
  if (target) {
    const contact = target.parentElement;
    const public_uid = contact.querySelector('#friend_public_uid').textContent.replace('@', "");
    //удаляем локально
    let contacts_requests_copy = contacts_requests;
    contacts_requests_copy = contacts_requests_copy.filter(req => {
      if (req.public_uid !== public_uid) {
        return true
      }
    })
    else {
      //req.account_uid - это уникальный идентификатор аккаунта который не будет твоим другом
      <
      navigator.sendBeacon(`${process.env.REACT_APP_BACKEND_URL}/api/beacon_drain`,
      JSON.stringify({
        action: "user_gate",
        arguments: { user_action: "decline_contact_request", contact_account_uid: req.account_uid }
      })))
      return false
    }
  };
}

```

```

    })
    setContacts_requests(contacts_requests_copy);
  }
}
const handlesendfriendrequest = () => {
  if (friend_for_request.trim() !== "" && friend_for_request.indexOf(' ') === -1) {
    setFriend_for_request("");

    //делаем ws запрос для отправки запроса в друзья
    socket.send(JSON.stringify({
      settings: {
        send_message_to_endpoint: true,
        handlers: [{ handler: "send_friend_request", condition: "before the sending message" }]
      },
      body: {
        endpoint_public_uid: friend_for_request,
        message: "new_friend_request"
      }
    }));
  }
  else {
    setNotificationMessage('пустые поля или идентификатор с пробелом не допускаются!');
    setNotificationColor('red')
    setNotificationPosition('bottom');
  }
}
const handleKeyUp = (e) => {
  if ((e.key === 'Enter' || e.keyCode === 13)) {
    handlesendfriendrequest();
  }
}

```

```

};

const handleKeyUpSendMessage = (e) => {
  if ((e.key === 'Enter' || e.keyCode === 13)) {
    handleSendMessage();
  }
};

const handleChange = (e) => {
  setFriend_for_request(e.target.value);
};

const handleAnyContactClickForConversation = (contact, callback) => {
  //тут мы будем получать историю сообщений с этим контактом
  //и отображать их в чате. Эта функция вызывается каждый когда пользователь открывает чат с
  КОНТАКТОМ
  //это функция служит для того, чтобы обновить состояние чата или создать новый чат, если его
  НЕТ
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "user_gate",
    arguments: { user_action: "get_conversation_history", contact_account_uid: contact.account_uid ||
contact.server_uid, chunk: 'last chunk' }
  }).then((data) => {
    if (data.status.type === "успех") {
      setMessages(data.delivery.resolved_chunk_metadata);
      setCurrent_chunk(data.delivery.current_chunk);
      if (callback !== undefined && typeof callback === 'function' && callback !== null) callback(data);
    }
    else {
      if (data.status.can_be_ignored === true) {
        setMessages([]);
        setCurrent_chunk(0);
        if (callback !== undefined && typeof callback === 'function' && callback !== null) callback(data);
        return;
      }
    }
  });
};

```

```

    }
    setNotificationMessage(data.status.message);
    setNotificationColor('red')
    setNotificationPosition('bottom');
    if (callback !== undefined && typeof callback === 'function' && callback !== null) callback(data);
  }
})
}

const handleHomeButton = () => {
  setSubpage({
    ...subpage,
    subpage: 'home',
    menu1: 'contacts',
    menu2: {
      ...subpage.menu2,
      primary: 'friends',
    }
  });
  setSelected_contact(null);
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "user_gate",
    arguments: { user_action: "get_contacts" }
  }).then((data) => {
    if (data.status.type === "ycnex") setResolved_contacts(data.delivery.resolved_contacts);
  })
}

const handleFriendsRequests = () => {
  setSubpage({
    ...subpage,
    menu2: {

```

```

    ...subpage.menu2,
    secondary: 'friends_requests'
  }
})
throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
  action: "user_gate",
  arguments: { user_action: "get_resolved_contacts_requests" }
}).then((data) => {
  if (data.status.type === "ycnex") setContacts_requests(data.delivery.resolved_contacts_requests);
})
}
const handleSelectAllFriends = () => {
  setSubpage({
    ...subpage,
    menu2: {
      ...subpage.menu2,
      secondary: 'all_friends'
    }
  })
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "user_gate",
    arguments: { user_action: "get_contacts" }
  }).then((data) => {
    if (data.status.type === "ycnex") setResolved_contacts(data.delivery.resolved_contacts);
  })
}
const handleAddFriend = () => {
  setSubpage({
    ...subpage,
    menu2: {

```

```

    ...subpage.menu2,
    secondary: 'add_friend'
  }
})
};

const GoToServer = (server) => {
  setSubpage({
    ...subpage,
    subpage: 'home',
    menu1: 'loading',
    menu2: {
      ...subpage.menu2,
      primary: 'loading',
    }
  });
  setTimeout(() => {
    //загружаем conversation
    setSelected_contact({ uid: server.server_uid, type: "server", metadata: server });
    handleAnyContactClickForConversation(server, (data) => {
      let proceed = () => {
        setSubpage({
          ...subpage,
          subpage: 'home',
          menu1: 'server',
          menu2: {
            ...subpage.menu2,
            primary: 'chat',
          }
        });
      };
    });
  }
}

```

```

    if (data.status.type === "успех") {
      proceed();
    }
    else {
      if (data.status.can_be_ignored === true) {
        proceed();
        return
      };
      setNotificationMessage(data.status.message);
      setNotificationColor('red')
      setNotificationPosition('bottom');
    }
  });
}, 50); // задержка для анимации перехода
}

const handleNoaccountButton = () => {
  setAuthState('sign-up');
  setFinal_name(null)
};

const handleAccountExistsButton = () => {
  setAuthState('sign-in');
  setFinal_name(null)
};

const handleVisibilityPasswordButton = () => {
  visible ? setVisible(false) : setVisible(true);
};

const handleSignUp = () => {
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "sign_up", arguments: { email: email, password: password, social_role: social_role,
    school_uid: final_name, first_name: first_name, last_name: last_name },
    settings: {

```

```

    withCredentials: true
  }
})
.then((data) => {
  setNotificationMessage(data.status.message);
  setNotificationColor(data.status.type === 'успех' ? 'green' : 'red')
  setNotificationPosition('bottom');
})
.catch((error) => {
  console.error("Error in throwRequest:", error);
});
};
// path inner windows system (BubbleWindows.js)
const bubbleWindowsRef = useRef(null);

const handleMetadata = () => {
  const ws = new WebSocket(process.env.REACT_APP_BACKEND_WS_URL);
  let interval
  ws.onopen = () => {
    console.log('Вебсокет соединение было открыто!');
    interval = setInterval(() => {
      ws.send(0x9);
    }, 5000);
  };
  ws.onmessage = (event) => {
    if (parseInt(event.data) === 0xA) console.log('heartbeat'); //получаем 0xA - pong ответ от сервера
    else {
      let json_message = JSON.parse(event.data);
      console.log(json_message);
      //визуализация ответа сервера

```

```

    if (json_message.handlers_state && json_message.handlers_state.states[0] &&
    json_message.handlers_state.states[0].show_state && json_message.handlers_state.states[0].show_state
    == true) {
        setNotificationMessage(json_message.handlers_state.states[0].message);
        setNotificationColor(json_message.handlers_state.states[0].type === 'ошибка' ? 'red' : 'green')
        setNotificationPosition('bottom');
    }
    //выполнение и обработка данных сервера
    if (json_message.message === 'new_friend_request') {
        setContacts_requests([...contacts_requests, json_message.sender_public_info])
    }
    else if (json_message.message === 'new_message') {
        console.log(selectedContactRef.current)
        if (selectedContactRef.current.type === "user") {
            if (selectedContactRef.current.uid == json_message.sender_public_info.account_uid &&
            json_message.initiated === "user") {
                //console.log(`Новое сообщение от ${json_message.sender_public_info.public_name} -
                @${json_message.sender_public_info.public_uid}: ${json_message.handlers_state.states[0].message}`)
                //если мы находимся в чате с этим контактом, то добавляем сообщение в чат
                if (json_message.handlers_state != undefined && json_message.handlers_state != null
                    && json_message.handlers_state.states[0] != undefined &&
                    json_message.handlers_state.states[0] != null
                ) {
                    setMessages([...messagesRef.current, {
                        owner_uid: json_message.sender_public_info.account_uid,
                        public_name: json_message.sender_public_info.public_name,
                        social_role: json_message.sender_public_info.social_role,
                        message: json_message.handlers_state.states[0].message,
                    }]);
                }
            }
        }
    }
}

```

```

else if (selectedContactRef.current.type === "server") {
  if (json_message.initiated === "server") {
    setMessages([...messagesRef.current, {
      owner_uid: json_message.sender_public_info.account_uid,
      public_name: json_message.sender_public_info.public_name,
      social_role: json_message.sender_public_info.social_role,
      message: json_message.handlers_state.states[0].message,
    }]);
  }
}
}
else if(json_message.message === 'kick_procedure'){
  let data_socket = json_message.handlers_state.states[0].message;
  if(data_socket && data_socket.can_be_trusted === true){
    let server_uid = data_socket.server_uid;
    if(selectedContactRef.current.type === "server" &&
selectedContactRef.current.metadata.server_uid === server_uid){
      handleHomeButton();
    }
    let current_servers = serversRef.current;
    current_servers = current_servers.filter(server => server.server_uid !== server_uid)
    setServers(current_servers);
  }
}
};
};
ws.onerror = (error) => {
  console.error('Ошибка вебсокета:', error);
};
ws.onclose = () => {
  console.log('Вебсокет соединение было закрыто!');
}

```

```

clearInterval(interval);

setNotificationMessage('Соединение потеряно!');

setNotificationColor('red')

setNotificationPosition('bottom');

};

setSocket(ws);

throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
  action: "user_gate",
  arguments: { user_action: "get_account_metadata" }
}).then((data) => {
  if (data.status.type === "успех") setUser_metadata(data.delivery);
})

throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
  action: "user_gate",
  arguments: { user_action: "get_contacts" }
}).then((data) => {
  if (data.status.type === "успех") setResolved_contacts(data.delivery.resolved_contacts);
})

throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
  action: "user_gate",
  arguments: { user_action: "get_resolved_open_contacts" }
}).then((data) => {
  if (data.status.type === "успех") setResolved_open_contacts(data.delivery.resolved_open_contacts);
})

throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
  action: "user_gate",
  arguments: { user_action: "get_resolved_contacts_requests" }
}).then((data) => {
  if (data.status.type === "успех") setContacts_requests(data.delivery.resolved_contacts_requests);
})

```

```

throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
  action: "user_gate",
  arguments: { user_action: "get_resolved_servers" }
}).then((data) => {
  if (data.status.type === "успех") setServers(data.delivery.resolved_servers);
})
}

const handleSignIn = () => {
  throwRequest(
    `${process.env.REACT_APP_BACKEND_URL}/api/post`,
    { action: "sign_in",
      arguments: { email: email, password: password, social_role: social_role }
    })
  .then((data) => {
    if (data.status.type === 'успех') {
      setAuthState('authenticated');
      //получаем метаданные пользователя
      handleMetadata()
    }
  })
  .catch((error) => {
    console.error("Error in throwRequest:", error);
  });
};

const handleSendMessage = () => {
  if (written_message.trim() !== "") {
    // отправляем сообщение через вебсокет
    socket?.send(JSON.stringify({
      settings: {
        send_message_to_endpoint: true,

```

```

    handlers: [{ handler: "add_message_to_conversation", condition: "before the sending message" }]
  },
  body: {
    send_message_type: 'contact',
    endpoint_uid: selected_contact.uid,
    message: "new_message",
    message_content: written_message,
  }
}));

let messages_in_array = messages !== null ? messages : [];
setMessages([...messages_in_array, {
  owner_uid: user_metadata.account_uid,
  public_name: user_metadata.public_name,
  social_role: user_metadata.social_role,
  message: written_message,
}]);

setWritten_message(""); // очищаем поле ввода сообщения
} else {
  console.log('Сообщение не может быть пустым!');
}
};

// настройки аккаунта
const [settingsPanel, setSettingsPanel] = useState(false);
const handleSettingsButtonClick = () => {
  setSettingsPanel(!settingsPanel);
}

// панель создания серверов (CreateServerPanel.js)
const [createServerPanel, setCreateServerPanel] = useState(false);
const handleServerCreateButtonClick = () => {
  setCreateServerPanel(!createServerPanel);
}

```

```

}
// \обработчики
useEffect(() => {
  throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
    action: "authenticate_by_session_uid", arguments: {},
    settings: {
      withCredentials: true
    }
  })
  .then((data) => {
    if (data.status.type === 'успех') {
      setAuthState('authenticated');
      //получаем метаданные пользователя
      handleMetadata()
    }
    else {
      setNotificationMessage(data.status.message);
      setNotificationColor(data.status.type === 'успех' ? 'green' : 'red')
      setNotificationPosition('bottom');
      setAuthState('sign-in');
    }
  })
  .catch((error) => {
    setAuthState('sign-in');
  });
}, []);
useLayoutEffect(() => {
  //меняем setScroll_pusher_style чтобы объект для observe был сверху всегда за сообщениями
  //let scrollable_conversation = document.getElementById('scrollable_conversation');
  let scrollable_conversation = conversation_ref.current;

```

```

if (scrollable_conversation) {
  let size_each_message = 90; // высота каждого сообщения в пикселях
  let all_messages_height = messages ? messages.length * size_each_message : 0; // высота всех
сообщений в пикселях
  let scroll_pusher_height = Math.max(0, scrollable_conversation.clientHeight - all_messages_height);
  setScroll_pusher_style({ height: `${scroll_pusher_height}px`, width: '100%' });
  // if (scroll_pusher_height > 0) {
  //   setScroll_pusher_style({ height: `${scroll_pusher_height}px`, width: '100%' });
  // }
  if (observerRef.current) observerRef.current.disconnect(); // отключаем наблюдатель после
загрузки предыдущего чанка
  //ставим observe для загрузки старых сообщений
  // если текущий чанк больше 1, то мы можем наблюдать за предыдущим чанком
  const observer = new IntersectionObserver((entries, observer) => {
    entries.forEach(entry => {
      if (entry.isIntersecting && current_chunk > 1) {
        //const originalScrollTop = scrollable_conversation.scrollTop;
        throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
          action: "user_gate",
          arguments: { user_action: "get_conversation_history", contact_account_uid:
selected_contact.uid, chunk: current_chunk - 1 }
        }).then((data) => {
          if (data.status.type === "успех") {
            setMessages([...data.delivery.resolved_chunk_metadata, ...messages]);
            setCurrent_chunk(data.delivery.current_chunk);
            // const newHeight = scrollable_conversation.scrollHeight;
            // scrollable_conversation.scrollTop = newHeight - originalScrollTop;
            // прокручиваем вниз, чтобы пользователь не заметил скачок
            // scrollable_conversation.scrollTop = originalScrollTop;
          }
        });
      }
    });
  });
  else {

```

```

        setNotificationMessage(data.status.message);

        setNotificationColor('red')

        setNotificationPosition('bottom');

    }

    })

}

});

}, { threshold: 0.8 }); // 0.8 - это порог видимости, можно настроить по желанию

observerRef.current = observer; // сохраняем observer в реф, чтобы можно было отключить его
позже

const targetElement = document.getElementById('previous_chunk');

if (targetElement) {

    observer.observe(targetElement);

}

};

return () => {

    if (observerRef.current) {

        observerRef.current.disconnect();

    }

};

}, [messages])

useEffect(() => {

    // наблюдаем за изменениями scroll_pusher_style и прокручиваем чат вниз

    const scrollable_conversation = conversation_ref.current;

    if (scrollable_conversation) {

        scrollable_conversation.scrollTop = scrollable_conversation.scrollHeight;

    }

}, [scroll_pusher_style])

// система определения, что хочет ссылка сделать при загрузке страницы

useEffect(() => {

    if (authState === 'authenticated') {

```

```

const pathnames = location.pathname.substring(1).split('/');
if (pathnames[0] === "invite") {
  //открываем тут bubble-window
  bubbleWindowsRef.current["addWindow"]("invite_request");
  let invite_uid = pathnames[1];
  if (invite_uid !== undefined && invite_uid !== null) {
    if (invite_uid.length === 0) {
      bubbleWindowsRef.current["invoke_invite_error"]();
      return;
    }
    throwRequest(`${process.env.REACT_APP_BACKEND_URL}/api/post`, {
      action: "user_gate",
      arguments: { user_action: "get_invite_metadata", uid: invite_uid }
    }).then((data) => {
      if (data.status.type === "ycnex") {
        bubbleWindowsRef.current["invoke_invite_success"]({...data.delivery, invite_uid: invite_uid});
      }
      else {
        bubbleWindowsRef.current["invoke_invite_error"]();
      }
    })
  }
  else {
    bubbleWindowsRef.current["invoke_invite_error"]();
  }
}
else if (authState === 'sign-in' || authState === 'sign-up') {
  navigate('/messenger')
}

```

```

}, [authState])
document.body.id = `body-${authState}`;
if (authState === 'sign-in') {
  return (
    <div className="SignIn">
      <h1>ВХОД</h1>
      <label className='indicators'>Почта</label>
      <div className='input-box'>
        <input size="32" placeholder="Пример: utail@mail.ru" type="email" id="emails" name="emails"
multiple maxLength="320" value={email}
          onChange={(event) => { setEmail(event.target.value) }} />
      </div>
      <label className='indicators'>Пароль</label>
      <div className='input-box'>
        <input size="32" placeholder="Введите ваш пароль" type={visible ? "text" : "password"}
id="pass" name="password" maxLength="128" value={password} required
          onChange={(event) => { setPassword(event.target.value) }} />
      </div>
      <div className='checkbox-showpassword'>
        <label><input size="32" id="checkbox" type="checkbox"
          onClick={handleVisibilityPasswordButton}
        /> показать пароль</label>
      </div>
      <button className='btns' id="LoginButton" onClick={handleSignIn}> войти </button>
      <div>
        <button className='link-btns' id="NoaccountButton" onClick={handleNoaccountButton}> Нет
аккаунта? Зарегистрируйтесь! </button>
      </div>
      <Notification
        message={notificationMessage}
        position={notificationPosition}

```

```

        onClose={handleCloseNotification}
        color={notificationColor}
    />
</div>
)
}
else if (authState === 'sign-up') {
    return (
        <div className="SignUp">
            <h1>регистрация</h1>
            <label className='indicators'>Имя</label>
            <div className='input-box'>
                <input size="32" placeholder="Введите ваше имя" type="text" id="names" maxLength="128"
value={first_name}
                onChange={(event) => { setFirst_name(event.target.value) }} autocomplete="off" />
            </div>
            <label className='indicators'>Фамилия</label>
            <div className='input-box'>
                <input size="32" placeholder="Введите вашу фамилию" type="text" id="last_names"
maxLength="128" value={last_name}
                onChange={(event) => { setLast_name(event.target.value) }} autocomplete="off" />
            </div>
            <label className='indicators'>Почта</label>
            <div className='input-box'>
                <input size="32" placeholder="Пример: utail@mail.ru" type="email" id="emails" name="emails"
multiple maxLength="320" value={email}
                onChange={(event) => { setEmail(event.target.value) }} autocomplete="off" />
            </div>
            <label className='indicators'>Пароль</label>
            <div className='input-box'>
                <input size="32" placeholder="Введите ваш пароль" type={visible ? "text" : "password"}
id="pass" name="password" maxLength="128" required value={password}

```

```

    onChange={{(event) => { setPassword(event.target.value) }}} autoComplete="off" />
</div>
<div className='checkbox-showpassword'>
  <label><input size="32" id="checkbox" type="checkbox"
onClick={handleVisibilityPasswordButton} /> показать пароль</label>
</div>
<div className='Switcher_container2'>
  <label id="item1">ученик</label>
  <Switcher
    onToggle={() => {
      if (social_role === 'ученик')
        setSocial_role('учитель');
      else
        setSocial_role('ученик');
    }}
    toggle={social_role === 'ученик' ? false : true}
  />
  <label id="item2">учитель</label>
</div>
  {/* {social_role === 'ученик' ? (
    <div>
      <Dropdown_menu dropdown_text="выберите школу и свой класс!" result={{(school,
grade, letter)}>{
        setFinal_name({
          school: school,
          grade: grade,
          letter: letter,
        })
      }}/>
    </div>
  ) : <</> */}

```

```

    <button className='btns' id="RegButton" onClick={handleSignUp}> зарегистрироваться
</button>

    <div>

        <button className='link-btns' id="AccountExistsButton" onClick={handleAccountExistsButton}>
Уже есть аккаунт? войдите! </button>

    </div>

    <Notification
        message={notificationMessage}
        position={notificationPosition}
        onClose={handleCloseNotification}
        color={notificationColor}
    />
</div>
)
}
else if (authState === 'authenticated') {
    return (
        <div className="Messenger">
            <div id="m_topbar">
                <div id="profile_bar">
                    {user_metadata != null ?
                        (
                            <div className='loaded' id="info_profile">
                                <div id="profile_avatar" className='loading_chunk">
                                    <div id="shine"></div>
                                </div>
                                <div id="info_block">
                                    <label className='loaded' id="public_name">{user_metadata.public_name}</label>
                                    <label className='loaded' id="public_uid">@{user_metadata.public_uid}</label>
                                </div>
                                <button className={`loaded ${settingsPanel ? 'pressed' : 'unpressed'}`} id="user_settings"

```

```

onClick={handleSettingsButtonClick}></button>
    </div>
)
:
(
<div id="info_profile">
    <div id="profile_avatar" className='loading_chunk'>
        <div id="shine"></div>
    </div>
    <div id="info_block">
        <label id="public_name" className='loading_chunk'>
            <div id="shine"></div>
        </label>
        <label id="public_uid" className='loading_chunk'>
            <div id="shine"></div>
        </label>
    </div>
</div>
)}
</div>
<div id="separator"></div>
{subpage.subpage === 'home' && subpage.menu2.primary === 'friends' && (<div
id='subpage_info'>
    <div id='icon'></div>
    <label id='text'>Друзья</label>
    {user_metadata != null ? (<div className='flex-container-row'>
        <div id="options_separator"></div>
        <div id="options">
            <button id="all_friends" className={subpage.menu2.secondary === 'all_friends' &&
'pressed'} onClick={handleSelectAllFriends}>Все</button>
            <button id="friends_requests_btn" className={subpage.menu2.secondary ===

```

```

'friends_requests' && 'pressed'} onClick={handleFriendsRequests}>ожидание</button>
    <button id="add_friend" className={subpage.menu2.secondary === 'add_friend' &&
'pressed'} onClick={handleAddFriend}>>Добавить в друзья</button>
    </div>
  </div> : (<div></div>)}
</div>)}
</div>
<div id="servers_bar">
  <button id="home_btn" onClick={handleHomeButton}></button>
  {user_metadata === null || servers === null ? (<div>
    <button id='server_btn' className='loading_chunk'>
      <div id="shine"></div>
    </button>
    <button id='server_btn' className='loading_chunk'>
      <div id="shine"></div>
    </button>
  </div>) : (
    <div>
      {servers.map((server, index) => {
        if (server.server_name.length >= 5) {
          server.server_name = server.server_name.slice(0, 5) + '!...!';
        }
      });
      return (
        <button
          id='server_btn'
          // style={{ backgroundImage: isValidUrl(server.server_avatar) ?
'url(${server.server_avatar})' : 'none' }}
          className={`loaded ${!(subpage.subpage === 'home' && subpage.menu1 === 'servers' &&
subpage.menu2.primary === server.server_uid ? 'selected' : 'unselected')`} `}
          onClick={() => {
            GoToServer(server);

```

```

    }}
    key={index}
  >
    <div id="server_avatar">
    </div>
    {!server.server_avatar && <label id="server_name"
className='loaded'>{server.server_name}</label>}
    </button>
  )
})
}
</div>
)}}
<button id='create_server_btn' onClick={handleServerCreateButtonClick}></button>
</div>
{/* панель создания серверов */}
{user_metadata && <CreateServerPanel createServerPanel={createServerPanel}
setCreateServerPanel={setCreateServerPanel} setServers={setServers} />}
{/* настройки аккаунта */}
{user_metadata && <UserSettings settingsPanel={settingsPanel}
setSettingsPanel={setSettingsPanel} user_metadata={user_metadata} />}
{/* всплывающие окна */}
<BubbleWindows ref={bubbleWindowsRef} navigate={navigate} GoToServer={GoToServer}
setServers={setServers} socket={socket}/>
<div id="menu_bar1">
  {subpage.menu1 === 'loading' &&
    <div style={{ height: "100%", width: "100%", display: 'flex', justifyItems: 'center', alignItems:
'center', justifyContent: 'center', alignContent: 'center' }}>
      <div className="loader custom_2"></div>
    </div>
  }
  {subpage.menu1 === 'server' &&

```

```

<div style={{ height: "100%", width: "100%", display: 'flex', flexDirection: "column" }}>
  <div style={{
    marginTop: '90px',
    width: '100%'
  }}></div>
  <ServerSettings BubbleFunctions={bubbleWindowsRef} selected_contact={selected_contact}
user_metadata={user_metadata}/>
  </div>
}
{subpage.subpage === 'home' && subpage.menu1 === 'contacts' && <div id="block_start">
  <label id="chats_text">Чаты</label>
  <button id="create_conversation_btn"></button>
</div>}
{subpage.subpage === 'home' && subpage.menu1 === 'contacts' && <div id="block1">
  <button>Все</button>
  <button>Приватные</button>
  <button>Общие</button>
  <button>Ещё</button>
</div>}
{subpage.subpage === 'home' && subpage.menu1 === 'contacts' && <div id="block2">
  <input placeholder="🔍 Поиск людей и каналов..."></input>
</div>}
{user_metadata != null && resolved_open_contacts != null ? (
  subpage.subpage === 'home' && subpage.menu1 === 'contacts' && (
    <div id="conversations">
      {resolved_open_contacts.map((contact, index) => (
        <button
          id="contact"
          className={`loaded-${(selected_contact?.uid === contact.account_uid ? 'selected' :
'unselected')}`}
          onClick={(e) => { handleContactClick(e); handleAnyContactClickForConversation(contact)

```

```
}}
```

```
id2={contact.account_uid}
```

```
type="user"
```

```
key={index}
```

```
>
```

```
<div id="profile_avatar_contact" className='loading_chunk'>
```

```
<div id="shine"></div>
```

```
</div>
```

```
<label id="public_name_contact" className='loaded'>{contact.public_name}</label>
```

```
</button>
```

```
)))
```

```
</div>
```

```
)
```

```
): (
```

```
subpage.subpage === 'home' && subpage.menu1 === 'contacts' && (
```

```
<div id="conversations">
```

```
<button id="contact">
```

```
<div id="profile_avatar_contact" className='loading_chunk'>
```

```
<div id="shine"></div>
```

```
</div>
```

```
<div id="public_name_contact" className='loading_chunk'>
```

```
<div id="shine"></div>
```

```
</div>
```

```
</button>
```

```
<button id="contact">
```

```
<div id="profile_avatar_contact" className='loading_chunk'>
```

```
<div id="shine"></div>
```

```
</div>
```

```
<div id="public_name_contact" className='loading_chunk'>
```

```
<div id="shine"></div>
```

```
</div>
</button>
<button id="contact">
  <div id="profile_avatar_contact" className='loading_chunk'>
    <div id="shine"></div>
  </div>
  <div id="public_name_contact" className='loading_chunk'>
    <div id="shine"></div>
  </div>
</button>
<button id="contact">
  <div id="profile_avatar_contact" className='loading_chunk'>
    <div id="shine"></div>
  </div>
  <div id="public_name_contact" className='loading_chunk'>
    <div id="shine"></div>
  </div>
</button>
<button id="contact">
  <div id="profile_avatar_contact" className='loading_chunk'>
    <div id="shine"></div>
  </div>
  <div id="public_name_contact" className='loading_chunk'>
    <div id="shine"></div>
  </div>
</button>
<button id="contact">
  <div id="profile_avatar_contact" className='loading_chunk'>
    <div id="shine"></div>
  </div>
  <div id="public_name_contact" className='loading_chunk'>
    <div id="shine"></div>
  </div>
</button>
```

```

    <div id="public_name_contact" className='loading_chunk'>
      <div id="shine"></div>
    </div>
  </button>
  <button id="contact">
    <div id="profile_avatar_contact" className='loading_chunk'>
      <div id="shine"></div>
    </div>
    <div id="public_name_contact" className='loading_chunk'>
      <div id="shine"></div>
    </div>
  </button>
</div>)
}}
</div>

<div id="menu_bar2" className='loading_chunk'>
  {subpage.subpage === 'home' && subpage.menu2.primary === 'friends' &&
subpage.menu2.secondary === 'all_friends' && <div>
  {user_metadata != null && resolved_contacts != null ? (
    <div>
      {resolved_contacts.map((contact, index) => (
        <button id="friend" className={index === 0 ? 'first' : ''} key={index} onClick={() => {
          const has_contact = resolved_open_contacts.some(req => req.account_uid ===
contact.account_uid)
          if (!has_contact) {
            setResolved_open_contacts([...resolved_open_contacts, contact]);

navigator.sendBeacon(`${process.env.REACT_APP_BACKEND_URL}/api/beacon_drain`,
JSON.stringify({
  action: "user_gate",
  arguments: { user_action: "add_open_contact", contact_account_uid: contact.account_uid
}

```

```

    )))
  }
  setSelected_contact({ uid: contact.account_uid, type: "user" });
  setSubpage({
    ...subpage,
    menu2: {
      ...subpage.menu2,
      primary: 'chat',
    }
  });
  handleAnyContactClickForConversation(contact)
}}>
<div id="friend_avatar" className='loading_chunk'>
  <div id="shine"></div>
</div>
<div id="flex-container-column" style={{ marginLeft: "15px" }}>
  <label id="friend_public_name" className='loaded'>{contact.public_name}</label>
  <label id="friend_public_uid" className='loaded'>{'@' + contact.public_uid}</label>
</div>
</button>
)}}
</div>
)
:(
<div>
  <div className='first' id="friend">
    <div id="friend_avatar" className='loading_chunk'> <div id="shine"></div> </div>
    <div id="flex-container-column" style={{ marginLeft: "15px" }}>
      <div id="friend_public_name" className='loading_chunk'><div id="shine"></div></div>
      <div id="friend_public_uid" className='loading_chunk'><div id="shine"></div></div>

```



```

    <div style={{ height: "100%", width: "100%", display: 'flex', justifyItems: 'center', alignItems:
'center', justifyContent: 'center', alignContent: 'center' }}>
      <div className="loader custom_2"></div>
    </div>
  }
  {subpage.subpage === 'home' && subpage.menu2.primary === 'friends' &&
subpage.menu2.secondary === 'friends_requests' && <div className='flex-container-column' style={{
height: "100vh" }}>
    <div id='empty' style={{ marginTop: "100px", width: "100%" }}></div>
    {contacts_requests.map((contact, index) => (
      <button id="friend" key={index}>
        <div id="friend_avatar" className='loading_chunk'>
          <div id="shine"></div>
        </div>
        <div className="flex-container-column" style={{ marginLeft: "15px" }}>
          <label id="friend_public_name" className='loaded'>{contact.public_name}</label>
          <label id="friend_public_uid" className='loaded'>{'@' + contact.public_uid}</label>
        </div>
        <button id="accept_btn" style={{ marginLeft: "20px" }}
onClick={handleAcceptfriendrequest}></button>
        <button id="decline_btn" style={{ marginLeft: "5px" }}
onClick={handleDeclinefriendrequest}></button>
      </button>
    )))
  </div>}
  {subpage.menu2.primary === 'chat' && <div className='chats'>
    {/* история сообщений */}
    <div id="margin_message"></div>
    <div id='message_form'>
      <input id='message_input' placeholder='Написать сообщение' value={written_message}
onChange={(e) => {
        setWritten_message(e.target.value);

```

```

    }} onKeyUp={handleKeyUpSendMessage}></input>
    <button id='send-btn' onClick={handleSendMessage}></button>
</div>

<div id="scrollable_conversation" ref={conversation_ref}>
    {current_chunk > 1 && <div id="previous_chunk" className='loading'><label>Загрузка
сообщений</label></div>}
    {messages != null && messages.map((message_obj, index) => {
        //ТУТ ОБРАБАТЫВАЕМ СООБЩЕНИЯ, МЕНЯЕМ НА ТО, ЧТО НУЖНО
        const markLinks = (text) => {
            const linkRegex = /(https?:\V|www\.)[^s]+/gi; // 'g' flag for global search
            return text.replace(linkRegex, (match) => {
                const url = match.startsWith('http') ? match : `http://${match}`; // Добавляем http, если
отсутствует
                return `<a href="${url}" target="_blank" rel="noopener noreferrer" class="turquoise-
link">${match}</a>`;
            });
        };
        const cooked_message = markLinks(message_obj.message);
        return (
            <div id="message" owner_uid={message_obj.owner_uid} key={index}>
                <div id="profile_avatar" className='loading_chunk'>
                    <div id="shine"></div>
                </div>
                <div id="message_body">
                    <label id="owner_public_name">{message_obj.public_name}</label>
                    {/* <label id="value">{message_obj.message}</label> */}
                    <label
                        id="value"
                        dangerouslySetInnerHTML={{ __html: cooked_message }} // Используем
dangerouslySetInnerHTML
                    />
                </div>
            </div>
        );
    });
}

```

```

        </div>
    )
}
)}}
    <div id="scroll_pusher" style={scroll_pusher_style}></div>
</div>
</div>
}
</div>
<Notification
    message={notificationMessage}
    position={notificationPosition}
    onClose={handleCloseNotification}
    color={notificationColor}
/>
</div>
)
}
else if (authState === 'loading') {
    return (
        <div className="loading">
            <div class="loading">
                <h1>загрузка</h1>
            </div>

            <div class="loading-animation">
                <span></span>
                <span></span>
                <span></span>
            </div>

```

```

    <Notification
      message={notificationMessage}
      position={notificationPosition}
      onClose={handleCloseNotification}
      color={notificationColor}
    />
  </div>
)
}
}

```

```
export default Messenger;
```

NotFound.js

```
import React from 'react';
import { Navigate } from 'react-router-dom';
```

```
function NotFound() {
  return (
    <div>
      <Navigate to="/error?code=404" />
      <h1>404 - Страница не найдена</h1>
      <p>Похоже вы не туда попали!</p>
    </div>
  );
}

```

```
export default NotFound;
```

Notification.js

```
import React, { useState, useEffect } from 'react';
import './styles/Notification.css';
```

```

const Notification = ({ message, position = 'bottom', onClose, color }) => {
  const [isVisible, setIsVisible] = useState(false);
  useEffect(() => {
    if (message) {
      setIsVisible(true);
      const timer = setTimeout(() => {
        setIsVisible(false);
      }, 5000); // Время отображения 5 секунды
      return () => clearTimeout(timer); // Очистка таймера при размонтировании
    }
  }, [message]);
  const handleTransitionEnd = () => {
    if (!isVisible && onClose) {
      onClose(); // Вызов onClose только после завершения анимации скрытия
    }
  };
  return (
    <div
      className={`notification ${position} ${isVisible ? 'show' : ''}`}
      id={color}
      onTransitionEnd={handleTransitionEnd}
    >
      <p> <b className="message">{message}</b> </p>
    </div>
  );
};
export default Notification;

```

ServerSettings.js

```

import React, {useState, useEffect, useRef} from 'react';

```

```

import './styles/ServerSettings.css'
import x from './html_assets/x.png';
import arrow_down from './html_assets/arrow_down.png';
const useOutsideClick = (callback) => {
  const ref = useRef(null);
  useEffect(() => {
    const handleClick = (event) => {
      if (ref.current && !ref.current.contains(event.target) && ref.current.classList.contains("opened")
&& !event.target.closest('.white-window')) { // Проверяем, является ли клик внутри элемента
        callback(event);
      }
    };
    document.addEventListener('mousedown', handleClick); // Используем mousedown, а не click
    return () => {
      document.removeEventListener('mousedown', handleClick);
    };
  }, [callback]);

  return ref;
};
function ServerSettings({BubbleFunctions, selected_contact, user_metadata}) {
  const [Dropdown, setDropdown] = useState(false);
  // const [IsOwner, setOwner] = useState(false);
  const DropdownRef = useRef(Dropdown);
  useEffect(() => {
    DropdownRef.current = Dropdown;
  }, [Dropdown]);
  let handleDropdownButton = () => {
    setDropdown(!Dropdown);
    // if(selected_contact.metadata.owner_uid === user_metadata.account_uid){
    //   setOwner(true);

```

```

// }
// else{
//   setOwner(false);
// };
}
let resetEverything = () => {
  setDropdown(false)
}
const ref = useOutsideClick((event) => {
  resetEverything();
})
return (
  <div className="ServerSettings">
    <button id="server_info_header" className={'white-window'}
onClick={handleDropdownButton}>
      <label id="server_name_v2">{selected_contact.metadata.full_name}</label>
      <img src={Dropdown ? x : arrow_down} alt=' ' style={{width: '17px', height: '17px', right:
'15px', position: 'absolute'}}></img>
    </button>
    <div id="DropdownMenu" className={` ${Dropdown ? 'opened' : ''} `} ref={ref}>
      {selected_contact.metadata.owner_uid === user_metadata.account_uid &&
        <button onClick={() => {
          BubbleFunctions.current["addWindow"]("invite_panel", selected_contact.metadata);
        }}>создать приглашение</button>
      }
      {selected_contact.metadata.owner_uid === user_metadata.account_uid &&
        <button onClick={() => {
          BubbleFunctions.current["addWindow"]("server_settings", selected_contact.metadata);
          BubbleFunctions.current["initServerPanel"]();
        }}>управление сервера</button>
      }
    </div>
  </div>
)

```

```

    </div>
    <div id="separator_v2"></div>
  </div>
);
}
export default ServerSettings;

```

Switcher.js

```

import React, { useState } from 'react';
import './styles/Switcher.css';
function Switcher({onToggle, toggle}) {
  const [toggled, setToggled] = useState(false);
  const handleClick = () => {
    setToggled(prevState => !prevState);
    onToggle(!toggled);
  };
  return (
    <div className="Switcher">
      <button
        className={`toggle-btn ${toggle ? "active" : ""}`}
        onClick={handleClick}
      >
        <div className={`thumb ${toggle ? "active" : ""}`}></div>
      </button>
    </div>
  );
}
export default Switcher;

```

UserSettings.js

```

import React, { useState, useEffect, useRef } from 'react';
import './styles/UserSettings.css';

```

```

import './styles/LoadingChunk.css'
import account_icon from './html_assets/icons8-user-64.png';
import settings_icon from './html_assets/settings-500.png';
import metadata_icon from './html_assets/metadata.png';
const useOutsideClick = (callback) => {
const ref = useRef();
useEffect(() => {
const handleClick = (event) => {
if (ref.current && !event.target.closest('.white-window') && event.target.closest('.opened') &&
event.target.closest('#settings_group')) {
callback(event);
}
};
document.addEventListener('click', handleClick);
return () => {
document.removeEventListener('click', handleClick);
};
}, [ref, callback]);

return ref;
};
function UserSettings({ settingsPanel, setSettingsPanel, user_metadata }) {
const defaultSettings = {
currentPage: 'main_page',
pages:
[
(<div id="main_page" style={{
width: '150px',
height: '225px',
backgroundColor: '#633A4F',
}}>

```

```

    { /*описываем панель страницы*/
      <div style={{ width: '100%', height: '25px', backgroundColor: 'transparent', display: 'flex',
alignItems: 'center' }}>
        <div style={{ width: '5px', height: '100%', position: 'relative' }}></div>
        <img src={settings_icon} alt=' ' style={{ width: '20px', height: '20px' }}></img>
      </div>
      <div style={{ width: '100%', height: '3px', backgroundColor: '#AF8A9D' }}>
      </div>
      <button id="account_settings_btn" style={{
        display: 'flex',
        alignItems: 'center',
      }} onClick={() => {
        //функция для перехода на другую страницу, которая будет работать с settingsData
        goto('account_settings_page');
      }}>
        <div style={{ width: '10px', height: '100%', position: 'relative' }}></div>
        <img src={account_icon} alt='[👤]' style={{ width: '15px', height: '15px' }}></img>
        <div style={{ width: '15px', height: '100%', position: 'relative' }}></div>
        настройки аккаунта
      </button>
    </div>,
  ],
}
const [settingsData, setSettingsData] = useState(defaultSettings);
const settingsDataRef = useRef(settingsData);
useEffect(() => {
  settingsDataRef.current = settingsData;
}, [settingsData]);
useEffect(() => {
  // делаем scrollToView по последнему элементу в массиве settingsData.pages

```

```

const lastPage = document.querySelector('#settings_conveyor > div:last-child');
if (lastPage) {
  lastPage.scrollToView({ behavior: 'smooth', block: 'end' });
}
5 }, [settingsData]);
const pages_ids = {
  'main_page': (settingsData.pages[0]), // используем уже существующую страницу
  'account_settings_page': (
    <div id="account_settings_page" style={{
      width: '150px',
      height: '225px',
      backgroundColor: '#633A4F',
    }}>
    /*описываем панель страницы*/
    <div style={{ width: '100%', height: '25px', backgroundColor: 'transparent', display: 'flex',
alignItems: 'center' }}>
      <div style={{ width: '5px', height: '100%', position: 'relative' }}></div>
      <img src={account_icon} alt=' ' style={{ width: '20px', height: '20px' }}></img>
    </div>
    <div style={{ width: '100%', height: '3px', backgroundColor: '#AF8A9D' }}>
    </div>
    <div style={{ width: '100%', height: '100%', backgroundColor: 'transparent', display: 'block' }}>
      <div style={{ width: '100%', height: '10px', position: 'relative' }}></div>
      <div style={{ width: '100%', height: 'auto', position: 'relative', display: 'flex', alignItems: 'center',
justifyContent: 'center', flexDirection: 'column' }}>
        <div style={{ width: '45px', height: '45px', borderRadius: '100px', backgroundColor: '#AF8A9D',
display: 'flex', overflow: 'hidden', position: 'relative' }} className='loading_chunk'>
          <div id="shine"></div>
        </div>
        <button className='btn' onClick={() => {
          goto('public_name_change_page');

```

```

    }}>{user_metadata ? user_metadata.public_name : 'загрузка'}</button>
    <button className='btn'onClick={() => {
      goto('public_uid_change_page');
    }}>@{user_metadata ? user_metadata.public_uid : 'загрузка'}</button>
  </div>
</div>
</div>
),
'public_name_change_page': (
  <div id="public_name_change_page" style={{
    width: '150px',
    height: '225px',
    backgroundColor: '#633A4F',
  }}>
    { /*описываем панель страницы*/}
    <div style={{ width: '100%', height: '25px', backgroundColor: 'transparent', display: 'flex',
alignItems: 'center' }}>
      <div style={{ width: '5px', height: '100%', position: 'relative' }}></div>
      <img src={metadata_icon} alt=' ' style={{ width: '20px', height: '20px' }}></img>
    </div>
    <div style={{ width: '100%', height: '3px', backgroundColor: '#AF8A9D' }}>
    </div>
    <div style={{ width: '100%', height: '100%', backgroundColor: 'transparent', display: 'block' }}>
      <div style={{ width: '100%', height: '10px', position: 'relative' }}></div>
      { /* Здесь будет форма изменения public_name*/}
      <div style={{ width: '100%', height: '30px', position: 'relative' }}></div>
      <div style={{ width: '100%', height: 'auto', position: 'relative', display: 'flex', alignItems: 'center',
justifyContent: 'center', flexDirection: 'column' }}>
        <button className='btn' onClick={() => {
          goto('account_settings_page');
        }}>{'<'</button>

```

```

    </div>
  </div>
</div>
),
'public_uid_change_page': (
  <div id="public_uid_change_page" style={{
    width: '150px',
    height: '225px',
    backgroundColor: '#633A4F',
  }}>
    { /*описываем панель страницы*/}
    <div style={{ width: '100%', height: '25px', backgroundColor: 'transparent', display: 'flex',
alignItems: 'center' }}>
      <div style={{ width: '5px', height: '100%', position: 'relative' }}></div>
      <img src={metadata_icon} alt=' ' style={{ width: '20px', height: '20px' }}></img>
    </div>
    <div style={{ width: '100%', height: '3px', backgroundColor: '#AF8A9D' }}>
    </div>
    <div style={{ width: '100%', height: '100%', backgroundColor: 'transparent', display: 'block' }}>
      <div style={{ width: '100%', height: '10px', position: 'relative' }}></div>
      { /* Здесь будет форма для изменения public_uid */}
      <div style={{ width: '100%', height: '30px', position: 'relative' }}></div>
      <div style={{ width: '100%', height: 'auto', position: 'relative', display: 'flex', alignItems: 'center',
justifyContent: 'center', flexDirection: 'column' }}>
        <button className='btn' onClick={() => {
          goto('account_settings_page');
        }}>{'<'</button>
      </div>
    </div>
  </div>
),

```

```

}

const debounce = useRef(false);

const goto = (page_name) => {
  if (debounce.current) return;
  debounce.current = true;
  setTimeout(() => {
    debounce.current = false;
  }, 500); // устанавливаем таймаут для предотвращения частых вызовов
  if (pages_ids[page_name]) {
    if (settingsDataRef.current.pages.some(page => page.props.id === page_name)) {
      const page = document.querySelector(`#${page_name}`);
      if (page) {
        // скроллим к существующей странице

        let TargetPageIndex = settingsDataRef.current.pages.findIndex((page) => page.props.id ===
page_name);

        let CurrentPageIndex = settingsDataRef.current.pages.findIndex((page) => page.props.id ===
settingsDataRef.current.currentPage);

        let lenghtBetween = Math.abs(TargetPageIndex - CurrentPageIndex);

        if (lenghtBetween > 1) {
          //удаляем все страницы между текущей и целевой

          setSettingsData({
            ...settingsDataRef.current,

            pages: settingsDataRef.current.pages.filter((page) => page.props.id === page_name ||
page.props.id === 'main_page' || page.props.id === settingsDataRef.current.currentPage),

          });
        }

        page.scrollIntoView({ behavior: 'smooth', block: 'end' });

        setTimeout(() => {
          setSettingsData({
            ...settingsDataRef.current,

            currentPage: page_name,

```

```

        pages: settingsDataRef.current.pages.filter((page) => page.props.id === page_name ||
page.props.id === 'main_page'),
    });
    }, 500);
}
return;
}
setSettingsData({
    ...settingsDataRef.current,
    currentPage: page_name,
    pages: [...settingsDataRef.current.pages, pages_ids[page_name]],
});
// let filteredPages = settingsData.pages.filter((page) => page.props.id === page_name || page.props.id
=== 'main_page');
// setTimeout(() => {
//   setSettingsData({
//     ...settingsDataRef.current,
//     currentPage: page_name,
//     pages: [...filteredPages, pages_ids[page_name]],
//   });
// }, 500);
} else {
    console.error(`Page ${page_name} does not exist.`);
}
}
const ref = useOutsideClick((event) => {
    setSettingsPanel(false);
    setTimeout(() => {
        setSettingsData(defaultSettings); // сбрасываем настройки при закрытии панели
    }, 500);
});

```

```

return (
  <div id="settings_group" className={settingsPanel ? 'opened' : 'closed'} ref={ref}>
    <div className={`white-window ${settingsPanel ? 'opened' : 'closed'}`} id="settings_window">
      <div id="settings_conveyor">
        {settingsData.pages.map((page, index) => (
          <React.Fragment key={index}>{page}</React.Fragment> // Оборачиваем в Fragment для
          правильного рендеринга
        )))}
      </div>
    </div>
  </div>
);
}
export default UserSettings;

```

About.css

```

/* все устройства */
.About {
  position: relative;
  width: 100vw;
  height: 100vh;
  overflow-x: hidden;
}
.About * {
  user-select: none; pointer-events: none;
}
.About .background {
  background-color: #633A4F;
  background-image: url('../public/css_assets/background1.svg');
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
}

```

```
width: 100%;
height: 2150px;
position: absolute;
top: 0;
left: 0;
z-index: 1;
}
.About .utail_bar_container{
width: 100vw;
height: 100vh;
}
.About .utail_bar {
position: absolute;
top: 0vh;
left: 0;
width: 100vw;
height: 110px;
z-index: 20;
background-color: transparent;
background-repeat: no-repeat;
background-position: center;
background-size: cover;
transition: height ease 0.3s;
}
.About .logo {
position: absolute;
left: 10%;
top: 10px;
background-image: url('../public/css_assets/logo.svg');
background-repeat: no-repeat;
```

```
background-position: center;
background-size: contain;
width: 100px;
height: 71px;
z-index: 21;
transition: top 0.3s ease;
}
.About .enter-btn {
  position: absolute;
  right: 5%;
  top: 30px;
  width: 140px;
  height: 35px;
  z-index: 21;
  background-color: #633A4F;
  color: white;
  font-family: 'Montserrat', sans-serif;
  font-size: 18px;
  border-radius: 30px;
  border: 0;
  cursor: pointer;
  transition: all 0.3s ease;
  user-select: none; pointer-events: auto;
}
.About .enter-btn:hover {
  background-color: #bd5389;
  color: white;
}
.About .enter-btn:active {
  background-color: #9c3d68;
```

```
transform: translateY(2px);
box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3);
}
/* block1 */
.About .block1 {
  width: 100vw;
  height: 100vh;
}
.About .Background_40 {
  position: absolute;
  top: 125px;
  left: 0vw;
  width: 100vw;
  height: 728px;
  transition: all ease 0.3s;
  z-index: 3;
}
.About .Rectangle_45_2 {
  position: absolute;
  top: 200px;
  right: 10vw;
  width: 750px;
  height: 573px;
  transition: all ease 0.3s;
  z-index: 4;
}
.About .Rectangle_46_2 {
  position: absolute;
  top: 200px;
  right: 8vw;
```

```
width: 900px;
height: 600px;
transition: all ease 0.3s;
z-index: 3;
}
```

```
.About .Line_1_2{
  position: absolute;
  top: 170px;
  right: 8vw;
  width: 900px;
  height: 650px;
  transition: all ease 0.3s;
  z-index: 4;
}
```

```
.About .DeviceMacbook_Air{
  position: absolute;
  top: 300px;
  right: 12vw;
  width: 650px;
  height: 400px;
  transition: all ease 0.3s;
  z-index: 5;
}
```

```
.About .Rectangle_41_3{
  position: absolute;
  top: 325px;
  left: 8vw;
  width: 684px;
  height: 230px;
  z-index: 7;
```

```
}
```

```
.About .Rectangle_43_2{
```

```
  position: absolute;
```

```
  top: 500px;
```

```
  left: 10vw;
```

```
  width: 585px;
```

```
  height: 153px;
```

```
  z-index: 6;
```

```
}
```

```
.About .Rectangle_41_3_text1 {
```

```
  position: absolute;
```

```
  top: 360px;
```

```
  left: 12vw;
```

```
  width: 500px;
```

```
  z-index: 8;
```

```
  font-size: 44px;
```

```
  color: white;
```

```
  font-family: 'Montserrat', sans-serif;
```

```
  font-weight: 700;
```

```
}
```

```
.About .Rectangle_43_2_text1 {
```

```
  position: absolute;
```

```
  top: 560px;
```

```
  left: 12vw;
```

```
  width: 500px;
```

```
  z-index: 8;
```

```
  font-size: 27px;
```

```
  color: white;
```

```
  font-family: 'Montserrat', sans-serif;
```

```
  font-weight: 400;
```

```
}
/* block2 */
.About .block2{
    position: absolute;
    width: 100vw;
    height: 600px;
    top: 900px;
    container-type: inline-size;
    container-name: block2;
}
.About .background1 {
    display: block;
    position: absolute;
    top: 0px;
    left: 10%;
    width: 80%;
    height: 100%;
    transition: all ease 0.3s;
    z-index: 3;
}
.About .Rectangle_45 {
    display: block;
    position: absolute;
    top: 5%;
    left: 16%;
    z-index: 4;
}
.About .Rectangle_46 {
    display: block;
    position: absolute;
```

```
top: 15%;
left: 14%;
z-index: 5;
}
.About .Line_1 {
display: block;
position: absolute;
top: 6%;
left: 12.5%;
z-index: 4;
}
.About .Background_header {
display: block;
position: absolute;
top: 20%;
right: 13%;
z-index: 5;
}
.About .Background_subtitle {
display: block;
position: absolute;
top: 40%;
right: 15%;
z-index: 4;
}
.About .Background_header_text1 {
display: block;
position: absolute;
top: 25%;
right: 15%;
```

```
z-index: 5;
width: 500px;
font-size: 35px;
color: #442233;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
}
```

```
.About .Background_subtitle_text1 {
display: block;
position: absolute;
top: 48%;
right: 16%;
z-index: 5;
width: 500px;
font-size: 25px;
color: #ffffff;
font-family: 'Montserrat', sans-serif;
font-weight: 400;
}
```

```
/* block3 */
```

```
.About .block3 {
position: absolute;
width: 100vw;
height: 600px;
top: 1550px;
container-type: inline-size;
container-name: block2;
}
```

```
.About .Background_39 {
display: block;
```

```
position: absolute;
top: 0%;
right: 0%;
width: 100%;
height: 100%;
z-index: 3;
}
.About .Rectangle_48_2{
display: block;
position: absolute;
top: 10%;
right: 5%;
z-index: 6;
}
.About .Rectangle_47{
display: block;
position: absolute;
top: 2%;
right: 6%;
z-index: 4;
}
.About .Line_2{
display: block;
position: absolute;
top: 4%;
right: 3%;
z-index: 5;
}
.About .Rectangle_49{
display: block;
```

```
position: absolute;
top: 15%;
left: 0%;
z-index: 6;
}
.About .Rectangle_49_text1 {
display: block;
position: absolute;
top: 20%;
left: 2%;
z-index: 7;
width: 500px;
font-size: 35px;
color: #ffffff;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
}
.About .Rectangle_50 {
display: block;
position: absolute;
top: 26%;
left: 0%;
z-index: 5;
}
.About .Rectangle_50_text1 {
display: block;
position: absolute;
top: 34%;
left: 1%;
z-index: 5;
```

```
width: 500px;
font-size: 24px;
color: #ffffff;
font-family: 'Montserrat', sans-serif;
font-weight: 400;
}
```

```
.About .Rectangle_52 {
display: block;
position: absolute;
top: 15%;
left: 35%;
z-index: 6;
}
```

```
.About .Rectangle_52_text1 {
display: block;
position: absolute;
top: 19%;
right: 33.5%;
z-index: 7;
width: 500px;
font-size: 35px;
color: #ffffff;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
}
```

```
.About .Rectangle_51 {
display: block;
position: absolute;
top: 26%;
```

```

left: 30%;
z-index: 6;
}
.About .Rectangle_51_text1 {
display: block;
position: absolute;
top: 37%;
left: 32%;
z-index: 7;
width: 550px;
font-size: 24px;
color: #ffffff;
font-family: 'Montserrat', sans-serif;
font-weight: 400;
}
/* телефон */
@media only screen and (max-width: 600px) {
.About .utail_bar {
height: 90px;
}
.About .enter-btn {
top: 20px;
}
.About .logo {
top: 7px;
}
}

```

BubbleWindows.css

```

.BubbleWindows{
width: 100vw;

```

```
height: 100vh;
position: fixed;
top: 0%;
left: 0%;
display: flex;
z-index: 10;
pointer-events: none;
align-items: center;
justify-content: center;
}
#invite_panel{
color: #FFF;
background-color: #633A4F;
width: 100%;
height: 100%;
pointer-events: auto;
animation: open 0.3s ease;
display: flex;
align-items: center;
justify-content: center;
}
#panel{
color: #FFF;
background-color: #633A4F;
width: 100%;
height: 100%;
pointer-events: auto;
animation: open 0.3s ease;
display: flex;
flex-direction: row;
```

```
    align-items: center;
}
#panel.closed{
    animation: close 0.3s ease;
}
#invite_request{
    color: #FFF;
    background-color: #633A4F;
    width: 100%;
    height: 100%;
    pointer-events: auto;
    animation: open 0.3s ease;
    display: flex;
    align-items: center;
    justify-content: center;
}
#invite_request.closed{
    animation: close 0.3s ease;
}
#invite_panel.closed{
    animation: close 0.3s ease;
}
#close-btn{
    width: 50px;
    height: 50px;
    border-radius: 100px;
    border: none;
    position: absolute;
    right: calc(0% + 10px);
    top: calc(0% + 10px);
```

```
background-color: #442233;
transition: background-color 0.2s ease;
background-image: url('../public/css_assets/x.png');
background-position: center;
background-repeat: no-repeat;
background-size: 20px;
box-shadow: 0 2px 8px rgba(0, 0, 0, 0.18), 0 0.5px 1.5px rgba(99, 58, 79, 0.15);
z-index: 10;
}
#close-btn:hover{
    background-color: #68344e;
}
#generate_invite-btn{
    background-color: #439139;
    transition: background-color 0.2s ease;
    color: #FFF;
    width: 200px;
    height: 35px;
    border: none;
    border-radius: 8px;
    font-family: 'Montserrat', sans-serif;
    box-shadow: 0 4px 16px rgba(67, 145, 57, 0.25), 0 1px 4px rgba(0, 0, 0, 0.12);
}
#generate_invite-btn:hover{
    background-color: #234b1e;
}
#copy-btn{
    width: 250px;
    height: 40px;
    transition: background-color 0.2s ease;
```

```
color: #FFF;
background-color: #27A600;
border-radius: 10px;
border: none;
font-family: 'Montserrat', sans-serif;
box-shadow: 0 4px 16px rgba(67, 145, 57, 0.25), 0 1px 4px rgba(0, 0, 0, 0.12);
}
#copy-btn:hover{
    background-color: #234b1e;
}
#dark_panel{
    position: relative;
    background-color: #442233;
    width: 35%;
    height: 100%;
    display: flex;
    flex-direction: row-reverse;
}
#panel_menu{
    position: relative;
    width: 65%;
    height: 100%;
}
#buttons_menu button{
    width: 100%;
    height: 35px;
    background-color: transparent;
    color: #cccccc;
    font-family: 'Montserrat', sans-serif;
    font-size: medium;
```

```
border: none;

border-radius: 8px;

transition: background-color 0.1s ease, color 0.1s ease;

margin-top: 5px;

margin-bottom: 5px;

}

#buttons_menu button.selected{

    background-color: rgb(255, 255, 255, 0.2);

    color: #FFF;

}

#buttons_menu .menu_separator{

    width: 100%;

    height: 1px;

    background-color: #633A4F;

    margin-top: 10px;

    margin-bottom: 10px;

    border-radius: 5px;

}

#buttons_menu label{

    width: 100%;

    height: 15px;

    color: #ffffff;

    font-family: 'Montserrat', sans-serif;

    font-size: small;

    background-color: transparent;

    margin-top: 5px;

    margin-bottom: 5px;

}

#buttons_menu button:hover{

    background-color: rgb(255, 255, 255, 0.2);
```

```
    color: #FFF;
}
#kick_btn{
    width: 41px;
    height: 41px;
    background-image: url('../public/css_assets/kick_icon.png');
    background-position: center;
    background-repeat: no-repeat;
    border-radius: 100px;
    border: none;
    background-color: #E03D3D;
    transition: background-color .2s ease;
}
#kick_btn:hover{
    background-color: #f88282;
}
@keyframes open {
    0%{
        opacity: 0;
        transform: scale(1.5);
    }
    100%{
        opacity: 1;
        transform: scale(1);
    }
}
@keyframes close {
    0%{
        transform: scale(1);
    }
}
```

```
100%{
    opacity: 0;
    transform: scale(1.5);
}
}
```

CreateServerPanel.css

```
/* панель создания серверов */
#server_panel_group {
    top: 0px;
    left: 0px;
    position: fixed;
    display: flex;
    justify-content: center;
    align-items: center;
    width: 100vw;
    height: 100vh;
    z-index: -1;
    background-color: transparent;
    transition: background-color 0.5s ease;
}
#server_panel_group.opened {
    background-color: rgba(0, 0, 0, 0.5); /* Полупрозрачный фон */
    z-index: 9;
}
#create_server_panel {
    position: absolute;
    display: flex;
    align-items: center;
    flex-direction: row;
    width: 250px;
```

```

height: 15vh;
background: #442233;
z-index: -1;
border-radius: 20px;
box-shadow: 0px 4px 4px #AF8A9D;
opacity: 0;
transition: all 0.5s ease;
overflow: hidden;
}
#create_server_panel.opened {
    opacity: 1;
    transform: scale(3); /* Увеличиваем масштаб в 2 раза */
    z-index: 10;
}
#create_server_panel #settings {
    position: relative;
    width: 150px;
    height: 100%;
    background-color: transparent;
    display: flex;
    flex-direction: column;
    justify-content: center;
    opacity: 0;
    transition: all 0.5s ease;
    left: -200px;
    pointer-events: none;
}
#create_server_panel #settings.closed {
    left: -200px !important;
    opacity: 0 !important;
}

```

```

    pointer-events: none !important;
}
/*кнопки в settings*/
#create_server_panel #settings button{
    margin-left: 10%;
    width: 80%;
    margin-top: 5px;
    border-radius: 8px;
    border: none;
    background-color: #AF8A9D;
    color: #fff;
    font-family: 'Montserrat', sans-serif;
    font-size: 10px;
    box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);
    transition: all 0.5s ease;
    min-height: 20px;
}
#create_server_panel #settings .create_btn_from_form{
    background-color: #14ca42;
}
#create_server_panel #settings button:hover{
    background-color: #dab4c7;
    transform: translateY(2px);
}
#create_server_panel #settings .create_btn_from_form:hover{
    background-color: #b2f5fa;
}
#create_server_panel #settings .create_btn_from_form:active{
    background-color: #5c5c5c;
}

```

```

#create_server_panel #settings button:active{
    background-color: #8063a1;
}

#create_server_panel.opened #settings {
    opacity: 1;
    left: 0px;
    pointer-events: auto;
}

#create_server_panel #theme {
    position: absolute;
    display: flex;
    align-items: center;
    right: 0%;
    width: 100px;
    height: 100%;
    background-color: #633A4F;
    transition: all 0.5s ease;
    box-shadow: -1px 5px 5px rgba(0, 0, 0, 0.25);
    right: -100px;
    opacity: 0;
    pointer-events: none;
}

#create_server_panel #theme.closed {
    right: -100px !important;
    opacity: 0 !important;
    pointer-events: none !important;
}

#create_server_panel.opened #theme {
    right: 0px;
}

```

```
opacity: 1;
pointer-events: auto;
}
```

Dropdown_menu.css

```
#Dropdown_menu_container{
display: flex;
width: 100%;
justify-content: center;
align-items: center;
}
```

```
#Dropdown_menu_content_container{
position: absolute;
display: flex;
width: 500px;
height: 300px;
transform: translateX(405px) translateY(-120px);
background-color: transparent;
border-radius: 20px;
}
```

```
#Dropdown_menu_content_container #numbers_container{
overflow: scroll;
overflow-x: hidden;
scrollbar-width: none;
-ms-overflow-style: none;
background: #633A4F;
background: linear-gradient(340deg, rgba(99, 58, 79, 1) 0%, rgba(175, 138, 157, 1) 100%);
width: 145px;
border-radius: 20px;
transition: all 0.5s ease;
}
```

```

#Dropdown_menu_content_container #numbers_container.closed{
    width: 0px;
}

#Dropdown_menu_content_container #letters_container{
    overflow:scroll;
    overflow-x:hidden;
    scrollbar-width: none;
    -ms-overflow-style: none;
    margin-left: 10px;
    background: #633A4F;
    background: linear-gradient(340deg, rgba(99, 58, 79, 1) 0%, rgba(175, 138, 157, 1) 100%);
    width: 145px;
    border-radius: 20px;
    transition: all 0.5s ease;
}

#Dropdown_menu_content_container #letters_container.closed{
    width: 0px;
}

#Dropdown_menu_content_container #letters_container2{
    overflow:scroll;
    overflow-x:hidden;
    scrollbar-width: none;
    -ms-overflow-style: none;
    margin-left: 10px;
    background: #633A4F;
    background: linear-gradient(340deg, rgba(99, 58, 79, 1) 0%, rgba(175, 138, 157, 1) 100%);
    width: 145px;
    border-radius: 20px;
    transition: all 0.5s ease;
}

```

```
#Dropdown_menu_content_container #letters_container2.closed{
  width: 0px;
}
#dropdown_menu1:hover{
  background-color: #a8547e;
}
#dropdown_menu1 {
  font-size: medium;
  font-family: 'Montserrat', sans-serif;
  background-color: #442233;
  color: #fff;
  border: 1px solid transparent;
  border-radius: 20px;
  width: 300px;
  height: 50px;
  transition: all 0.5s ease;
}
#dropdown_menu1.active{
  border-color: #ffffff;
}
.dropdown_object{
  display: block;
  width: 100%;
  height: 40px;
  border: none;
  border-radius: 20px;
  margin-bottom: 2px;
  background: #ac6087;
  color: #fff;
  font-family: 'Montserrat', sans-serif;
```

```
font-size: medium;
transition: all 0.5s ease;
}
.dropdown_object:hover{
background: #ffffff;
color: black;
}
.dropdown_object#selected{
border: 2px solid #fff;
}
```

LoadingChunk.css

```
.loading_chunk{
overflow: hidden;
}
.loading_chunk #shine{
position: absolute;
top: 0;
left: -100%;
width: 400%;
height: 200%;
background: linear-gradient(to right, transparent, rgba(255, 255, 255, 1), transparent);
animation: shine 2s infinite linear;
}
@keyframes shine {
0% {
opacity: 1;
left: -400%;
}
100% {
opacity: 0;
```

```
    left: 200%;  
  }  
}
```

Main.css

```
* {  
  padding: 0;  
  margin: 0;  
}
```

Messenger.css

```
#body-sign-in {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  min-height: 100vh;  
  background-image: url('../public/css_assets/authorization_[Prototype_1].png');  
}  
  
.SignIn h1 {  
  font-family: 'Montserrat', sans-serif;  
  font-weight: 700;  
  text-align: center;  
  font-size: 36px;  
}  
  
.SignIn {  
  width: 550px;  
  background-color: #633A4F;  
  color: #fff;  
  border-radius: 30px;  
}  
  
.SignIn .input-box {  
  width: 100%;
```

```
height: 50px;
color: #fff;
border: none;
outline: none;
margin: 30px 0px;
background: transparent;
display: flex;
justify-content: center;
justify-items: center;
}
.SignIn .input-box input{
background: #442233;
box-shadow: inset 0 4px 4px 0 rgba(0, 0, 0, 0.25);
border-radius: 20px;
width: 70%;
height: 100%;
border: none;
outline: none;
text-align: left;
padding-left: 30px;
color: #fff;
font-size: medium;
font-family: 'Montserrat', sans-serif;
}
.SignIn .checkbox-showpassword{
font-size: medium;
font-family: 'Montserrat', sans-serif;
margin-left: 35%;
user-select: none;
}
```

```

.SignIn .btns{
  width: 70%;
  height: 50px;
  border: none;
  outline: none;
  border-radius: 8px;
  margin-top: 5px;
  margin-left: 15%;
  box-shadow: 0 0 10px rgba(0,0,0,.1);
  background-color: #987185;
  color: #fff;
  font-size: medium;
  font-family: 'Montserrat', sans-serif;
  transition: all .5s ease;
  box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);
}

.SignIn .btns#LoginButton{
  width: 40%;
  margin-left: 30%;
}

.SignIn .btns:hover{
  background-color: #ffffff;
  color: black;
}

.SignIn .btns:active{
  background-color: gray;
  color: #fff
}

.SignIn .indicators{
  display: flex;

```

```
margin-left: 15%;
margin-bottom: -5%;
color: #fff;
font-size: large;
font-family: 'Montserrat', sans-serif;
}
/* регистрация */
#body-sign-up{
display: flex;
justify-content: center;
align-items: center;
min-height: 100vh;
background-image: url('../public/css_assets/authorization_[Prototype_1].png');
}
.SignUp h1 {
font-family: 'Montserrat', sans-serif;
font-weight: 700;
text-align: center;
font-size: 36px;
}
.SignUp {
width: 550px;
background-color: #633A4F;
color: #fff;
border-radius: 30px;
}
.SignUp .input-box{
width: 100%;
height: 50px;
color: #fff;
```

```
border: none;

outline: none;

margin: 30px 0px;

background: transparent;

display: flex;

justify-content: center;

justify-items: center;

}

.SignUp .input-box input{

background: #442233;

box-shadow: inset 0 4px 4px 0 rgba(0, 0, 0, 0.25);

border-radius: 20px;

width: 70%;

height: 100%;

border: none;

outline: none;

text-align: left;

padding-left: 30px;

color: #fff;

font-size: medium;

font-family: 'Montserrat', sans-serif;

}

.SignUp .checkbox-showpassword{

font-size: medium;

font-family: 'Montserrat', sans-serif;

margin-left: 35%;

user-select: none;

}

.SignUp .btns{

width: 70%;
```

```

height: 50px;
border: none;
outline: none;
border-radius: 8px;
margin-top: 5px;
box-shadow: 0 0 10px rgba(0,0,0,.1);
background-color: #987185;
color: #fff;
font-size: medium;
font-family: 'Montserrat', sans-serif;
transition: all .5s ease;
}
.SignUp .btns#RegButton{
width: 80%;
margin-left: 10%;
}
.SignUp .btns:hover{
background-color: #ffffff;
color: black;
}
.SignUp .btns:active{
background-color: gray;
color: #fff
}
.Switcher_container2{
display: flex;
align-items: center;
}
.SignUp #item1 {
margin-left: 17.5%;

```

```
margin-right: 10%;  
font-size: 25px;  
height: 20px;  
font-family: 'Montserrat', sans-serif;  
}
```

```
.SignUp #item2 {  
margin-left: 10%;  
margin-right: 10%;  
font-size: 25px;  
font-family: 'Montserrat', sans-serif;  
}
```

```
.SignUp .link-btns {  
width: 70%;  
height: 25px;  
margin-top: 25px;  
margin-bottom: 25px;  
margin-left: 15%;  
background: none !important;  
border: none;  
padding: 0 !important;  
font-family: 'Montserrat', sans-serif;  
color: rgb(255, 255, 255);  
text-decoration: underline;  
cursor: pointer;  
transition: color 0.4s ease;  
}
```

```
.SignUp .link-btns:hover {  
color: blue  
}
```

```

.SignIn .link-btns{
  width: 70%;
  height: 25px;
  margin-top: 25%;
  margin-bottom: 25px;
  margin-left: 15%;
  background: none!important;
  border: none;
  padding: 0!important;
  font-family: 'Montserrat', sans-serif;
  color: rgb(255, 255, 255);
  text-decoration: underline;
  cursor: pointer;
  transition: color 0.4s ease;
}

.SignIn .link-btns:hover{
  color: blue
}

.SignUp .indicators{
  display: flex;
  margin-left: 15%;
  margin-bottom: -5%;
  color: #fff;
  font-size: large;
  font-family: 'Montserrat', sans-serif;
}

/* основной мессенджер */
#body-authenticated{
  height: 100vh;
  display: flex;

```

```
background: url('../public/css_assets/background_messenger.png') no-repeat center;
}
#m_topbar{
display: flex;
align-items: center;
position: absolute;
top: 0%;
background-color: #633A4F;
width: 100%;
min-height: 90px;
z-index: 2;
box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);
}
#profile_bar{
position: relative;
display: flex;
flex-wrap: nowrap;
width: auto;
height: 80px;
box-sizing: border-box;
background: rgba(152, 113, 133, 0.46);
border: 2px solid rgba(68, 34, 51, 0.32);
box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);
border-radius: 20px;
margin-left: 5px;
}
#info_profile{
position: relative;
display: flex;
flex-wrap: nowrap;
```

```
}  
#profile_avatar{  
    position: relative;  
  
    width: 73px;  
    height: 73px;  
    left: 20px;  
    top: 3%;  
    background: #AF8A9D;  
    border-radius: 100px;  
}  
#info_block{  
    position: relative;  
    display: flex;  
    flex-direction: column;  
    width: auto;  
    margin-left: 30px;  
    margin-right: 10px;  
}  
#public_name{  
    position: relative;  
    width: 200px;  
    height: 25px;  
    background: #AF8A9D;  
    border-radius: 20px;  
    margin-top: 20px;  
    margin-bottom: 5px;  
}  
#public_uid{  
    position: relative;
```

```
width: 100px;
height: 20px;
background: #AF8A9D;
border-radius: 20px;
}
#separator{
    position: relative;
    margin-left: 10px;
    width: 3px;
    height: 90px;
    background-color: #987185;
}
#subpage_info {
    display: flex;
    flex-direction: row;
    align-items: center;
}
/* информация о под странице: "Друзья" */
#subpage_info #icon{
    position: relative;
    background: url('../public/css_assets/unknown_user.png');
    width: 50px;
    height: 50px;
}
#subpage_info #text{
    position: relative;
    font-size: 20px;
    font-family: 'Montserrat', sans-serif;
    color: #ffffff;
```

```

}
#subpage_info #options_separator{
    position: relative;
    margin-left: 10px;
    width: 2px;
    height: 50px;
    background-color: #F4E2D1;
}
#subpage_info #options{
    margin-left: 20px;
    position: relative;
    display: flex;
    flex-direction: row;
    align-items: center;
}
#subpage_info #options *{
    width: 129px;
    height: 41px;
    border-radius: 10px;
    border: none;
    margin-right: 10px;
}
/* ОПЦИИ КАСТОМНЫЕ */
#add_friend{
    background-color: #27A600;
    font-family: 'Montserrat', sans-serif;
    font-weight: 500;
    font-size: 15px;
    color: white;
    width: 170px !important;
}

```

```
    transition: all 0.5s ease;
}
#add_friend:hover{
    background-color: #FFFFFF;
    color: #633A4F;
}
#add_friend.pressed{
    background-color: #FFFFFF;
    color: #633A4F;
}
#friends_requests_btn{
    background-color: #442233;
    font-family: 'Montserrat', sans-serif;
    font-weight: 500;
    font-size: 15px;
    color: white;
    transition: all 0.5s ease;
}
#friends_requests_btn:hover{
    background-color: #FFFFFF;
    color: #633A4F;
}
#friends_requests_btn.pressed{
    background-color: #FFFFFF;
    color: #633A4F;
}
/* подстраница add_friend */
#add_friend_label{
    color: #442233;
    font-family: 'Montserrat', sans-serif;
```

```
font-size: 25px;
font-weight: 600;
}
#add_friend_input{
width: 30%;
height: 35px;
border-radius: 20px;
border: none;
margin-left: 10px;
padding-left: 20px;
background-color: #AF8A9D;
color: white;
font-size: 15px;
font-family: 'Montserrat', sans-serif;
font-weight: 500;
}
#all_friends{
background-color: #442233;
font-family: 'Montserrat', sans-serif;
font-weight: 500;
font-size: 15px;
color: white;
width: 90px !important;
transition: all 0.5s ease;
}
#all_friends:hover{
background-color: #FFFFFF;
color: #633A4F;
}
#all_friends.pressed{
```

```
background-color: #FFFFFF;
color: #633A4F;
}
#servers_bar {
    position: relative;
    display: flex;
    flex-direction: column;
    left: 0%;
    top: 0%;
    width: 75px;
    height: 100vh;
    background: #442233;
    z-index: 1;
}
#home_btn{
    margin-top: 120px;
    width: 73px;
    height: 74px;
    background-image: url('../public/css_assets/Group16.png');
    background-position: center;
    background-repeat: no-repeat;
    background-color: #633A4F;
    border: none;
    border-top-right-radius: 20px;
    border-bottom-right-radius: 20px;
    transition: all 0.5s ease;
}
#home_btn:hover{
    background-color: #ffc7e4;
}
```

```
#server_btn{
    position: relative;
    margin-left: 7.5px;
    margin-top: 5px;
    width: 60px;
    height: 60px;
    border-radius: 100px;
    border: none;
    background-color: #AF8A9D;
    color: white;
    font-family: 'montserrat', sans-serif;
    transition: border 0.3s ease;
}

#server_btn:hover{
    border: 2px solid #d597ff;
}

#create_server_btn{
    position: relative;
    margin-left: 7.5px;
    margin-top: 5px;
    width: 60px;
    height: 60px;
    border-radius: 100px;
    border: none;
    background-color: #633A4F;
    background-image: url('../public/css_assets/+.png');
    background-position: center;
    background-repeat: no-repeat;
    transition: all 0.5s ease;
}
```

```
#create_server_btn:hover{
  background-color: #ffc7e4;
}
```

```
#menu_bar1 {
  position: absolute;
  display: flex;
  flex-direction: column;
  left: 75px;
  top: 0%;
  width: 375px;
  height: 100vh;
  background: #633A4F;
  z-index: 1;
}
```

```
.loading_chunk#menu_bar2 {
  position: absolute;
  display: flex;
  flex-direction: column;
  left: 470px;
  top: 0px;
  width: 70%;
  height: 100vh;
  z-index: 1;
}
```

```
#friend {
  margin-top: 10px;
  position: relative;
  display: flex;
  width: 100%;
  height: 65px;
```

```
border-radius: 20px;
background: #633A4F;
border: none;
align-items: center;
}
#friend.first{
margin-top: 110px;
}
#friend_avatar{
position: relative;
display: flex;
margin-left: 20px;
width: 50px;
height: 50px;
border-radius: 100px;
background-color: #AF8A9D;
overflow: hidden;
}
#friend_public_name{
position: relative;
display: flex;
width: 113px;
height: 20px;
border-radius: 20px;
background-color: #AF8A9D;
overflow: hidden;
}
#friend_public_name.loaded{
width: auto;
background-color: transparent;
```

```
color: white;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
font-size: 15px;
}
#friend_public_uid{
  position: relative;
  display: flex;
  width: 80px;
  height: 20px;
  border-radius: 20px;
  background-color: #AF8A9D;
  overflow: hidden;
}
#friend_public_uid.loaded{
  width: auto;
  background-color: transparent;
  color: white;
  font-family: 'Montserrat', sans-serif;
  font-weight: 400;
  font-size: 15px;
}
#block_start{
  display: flex;
  position: relative;
  margin-top: 100px;
  width: 100%;
  height: 90px;
}
#block_start #chats_text{
```

```

position: relative;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
font-size: 48px;
color: #fff;
margin-left: 25px;
margin-top: 25px;
}
#block_start #create_conversation_btn{
    position: relative;
    width: 65px;
    height: 65px;
    border-radius: 100px;
    border: none;
    margin-left: 100px;
    margin-top: 15px;
    background-color: #987185;
    background-image: url('../public/css_assets/+.png');
    background-repeat: no-repeat;
    background-position: center;
    box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);
    border-radius: 35px;
    transition: all 0.5s ease;
}
#block_start #create_conversation_btn:hover{
    background-color: #ffc7e4;
}
#block1 {
    display: flex;
    position: relative;

```

```
width: 100%;
height: 50px;
}
#block1 *{
background: transparent;
border: none;
margin-left: 15px;
font-family: 'Montserrat', sans-serif;
font-size: 20px;
font-weight: 600;
color: rgb(255, 255, 255, 0.63);
transition: all .5s ease;
}
#block1 *:hover{
color: #ffffff;
}
#block2{
display: flex;
position: relative;
width: 100%;
height: 60px;
}
#block2 *{
width: 87%;
height: 80%;
border: 3px solid #442233;
border-radius: 20px;
background-color: #987185;
font-size: 17px;
font-family: 'Montserrat', sans-serif;
```

```

color: #000000;
padding-left: 35px;
border: none;
outline: none;
}
#conversations{
display: flex;
flex-direction: column;
position: relative;
align-items: center;
overflow:scroll;
overflow-x:hidden;
scrollbar-width: none;
-ms-overflow-style: none;
width: 100%;
height: 600px;
}
#conversations #contact{
display: flex;
align-items: center;
width: 95%;
height: 65px;
border-radius: 20px;
border: 2px solid #312028;
margin-bottom: 10px;
background-color: #442233;
transition: all 0.3s ease;
}
#conversations #contact #profile_avatar_contact{
position: relative;

```

```

display: flex;
overflow: hidden;
margin-left: 15px;
width: 55px;
height: 55px;
background: #AF8A9D;
border-radius: 100px;
}
#conversations #contact #public_name_contact {
    position: relative;
display: flex;
overflow: hidden;
margin-left: 5px;
width: 125px;
height: 20px;
background: #AF8A9D;
border-radius: 20px;
transition: all 0.3s ease;
}
#conversations #contact #public_name_contact.loaded {
    width: auto;
background-color: transparent;
color: white;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
font-size: 15px;
}
#conversations #contact.loaded-selected #public_name_contact.loaded {
    color: #442233 !important;
}

```

```
#conversations #contact:hover{
    background-color: rgb(110, 66, 91);
}
/* #conversations #contact.loaded-unselected{
} */
#conversations #contact.loaded-selected{
    background-color: #ffffff;
}
#public_name.loaded{
    width: auto;
    background: transparent;
    color: #FFFFFF;
    font-family: 'Montserrat', sans-serif;
    font-size: 20px;
    font-weight: 800;
    margin-bottom: 0px;
}
#public_uid.loaded{
    width: auto;
    background: transparent;
    color: #FFFFFF;
    font-family: 'Montserrat', sans-serif;
    font-size: 15px;
    font-weight: 400;
}
#user_settings.loaded, #user_settings.loaded.unpressed{
    position: relative;
    margin-left: 10px;
    width: 48px;
    height: 48px;
```

```
background-image: url('../public/css_assets/settings_icon.svg');
background-color: transparent;
background-position: center;
background-repeat: no-repeat;
border: none;
margin-top: 14px;
margin-right: 15px;
transition: transform 0.3s ease;
}
#user_settings.loaded.pressed{
    transform: rotate(45deg);
}
/* ЧАТЫ */
#margin_message{
    margin-top: 50px;
    width: 100%;
    height: 65px;
}
.chats {
    width: 100%;
    height: 100%;
    display: flex;
    flex-direction: column;
}
.chats #message{
    align-items: center;
    margin-top: 25px;
    position: relative;
    display: flex;
    flex-direction: row;
```

```
width: 100%;
height: 65px;
border-radius: 20px;
background: none;
}
.chats #profile_avatar{
border: 5px solid #442233;
position: relative;
left: auto;
top: auto;
}
#message_body{
border-radius: 20px;
padding-top: 10px;
padding-left: 10px;
padding-right: 10px;
margin-left: 10px;
display: flex;
flex-direction: column;
height: 60px;
width: auto;
background-color: #442233;
color: white;
font-family: 'Montserrat', sans-serif;
}
#message_form{
position: absolute;
display: flex;
flex-direction: row;
align-items: center;
```

```
width: 100%;  
height: 90px;  
bottom: 0px;  
z-index: 5;  
}  
#message_input{  
width: 98%;  
height: 55px;  
background: #442233;  
box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);  
border-radius: 20px;  
border: none;  
outline: none;  
color: white;  
font-family: 'Montserrat', sans-serif;  
font-size: 18px;  
font-weight: 600;  
padding-left: 20px;  
}  
#message_body #owner_public_name{  
font-weight: 700;  
}  
#send-btn{  
width: 55px;  
height: 55px;  
border-radius: 100px;  
border: none;  
margin-left: 10px;  
background-image: url('../public/css_assets/send.png');  
background-position: center;
```

```
background-repeat: no-repeat;
}
.flex-container-row{
display: flex;
flex-direction: row;
}
.flex-container-column{
display: flex;
flex-direction: column;
}
/* кнопки принять или отклонить запрос */
#accept_btn {
border-radius: 100px;
width: 40px;
height: 40px;
background-image: url('../public/css_assets/accept_decline_assets/accept.png');
background-color: transparent;
background-repeat: no-repeat;
background-position: center;
background-size: cover;
border: none;
}
#accept_btn:hover {
background-image: url('../public/css_assets/accept_decline_assets/accept_hovered.png');
background-repeat: no-repeat;
background-position: center;
background-size: cover;
}
#decline_btn {
border-radius: 100px;
```

```
width: 40px;
height: 40px;
background-image: url('../public/css_assets/accept_decline_assets/decline.png');
background-color: transparent;
background-repeat: no-repeat;
background-position: center;
background-size: cover;
border: none;
}
#decline_btn:hover {
background-image: url('../public/css_assets/accept_decline_assets/decline_hovered.png');
background-repeat: no-repeat;
background-position: center;
background-size: cover;
}
#scrollable_conversation {
overflow-y: auto;
overflow-x: hidden;
scrollbar-width: none;
-ms-overflow-style: none;
width: 100%;
height: 80%;
}
#previous_chunk {
display: flex;
position: relative;
width: 100%;
height: 100px;
background-color: transparent;
align-items: center;
```

```
justify-content: center;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
font-size: medium;
}
.turquoise-link {
  color: turquoise;
}
.turquoise-link:hover {
  color: #00bfff;
}
:root {
  --main_size: 3px;
  --custom_2_size: 5px;
}
.loader {
  border: var(--main_size) solid #f3f3f3; /* Light grey */
  border-top: var(--main_size) solid #8063a1; /* Blue */
  border-radius: 50%;
  width: 10px;
  height: 10px;
  animation: spin 0.5s linear infinite;
}
.loader.custom_2 {
  border: var(--custom_2_size) solid #f3f3f3; /* Light grey */
  border-top: var(--custom_2_size) solid #8063a1; /* Blue */
  width: 20px;
  height: 20px;
}
```

```
@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
```

Notification.css

```
/* Notification.css */
.notification#green{
  position: fixed;
  left: 0;
  width: 100%;
  background-color: #22aa00;
  color: white;
  font-family: 'Montserrat', sans-serif;
  font-size: medium;
  font-weight: 600;
  text-align: center;
  padding: 15px;
  box-shadow: 0px -2px 5px rgba(0, 0, 0, 0.2);
  z-index: 1000;
  display: block;
}
.notification#usual{
  position: fixed;
  left: 0;
  width: 100%;
  background-color: #a200ff;
  color: white;
  font-family: 'Montserrat', sans-serif;
  font-size: medium;
  font-weight: 600;
```

```

text-align: center;
padding: 15px;
box-shadow: 0px -2px 5px rgba(0, 0, 0, 0.2);
z-index: 1000;
display: block;
}
.notification#red{
position: fixed;
left: 0;
width: 100%;
background-color: #ff0000;
color: white;
font-family: 'Montserrat', sans-serif;
font-size: medium;
font-weight: 600;
text-align: center;
padding: 15px;
box-shadow: 0px -2px 5px rgba(0, 0, 0, 0.2);
z-index: 1000;
display: block;
}
/* Стили для выдвигания снизу */
.notification.bottom { /* Добавляем класс bottom */
bottom: 0; /* Закрепляем внизу экрана */
transform: translateY(100%); /* Скрываем за нижним краем */
transition: transform 0.3s ease-out; /* Добавляем плавную анимацию */
}
.notification.bottom.show {
transform: translateY(0); /* Выдвигаем на экран */
}

```

```
/* Стили для выдвигания сверху (если нужно) */  
.notification.top {  
  top: 0;  
  transform: translateY(-100%);  
  transition: transform 0.3s ease-out;  
}  
.notification.top.show {  
  transform: translateY(0);  
}
```

ServerSettings.css

```
/*общий контейнер*/  
.ServerSettings {  
  width: 100%;  
  height: calc(100% - 90px);  
  display: flex;  
  flex-direction: column;  
  position: relative;  
}  
#separator_v2 {  
  width: 100%;  
  height: 2px;  
  background-color: #ffffff6e;  
}  
/*объекты контейнера*/  
#server_info_header {  
  background-color: #633A4F;  
  width: 100%;  
  height: 55px;  
  display: inline-flex;  
  align-items: center;
```

```
border: none;

transition: background-color 0.1s ease;
}

#server_info_header:hover{
    background-color: #7a4862;
}

#server_name_v2{
    font-family: 'Montserrat', sans-serif;
    color: #fff;
    font-size: large;
    font-weight: 700;
}

:root{
    --width_eraser: 120px
}

#DropDownMenu{
    align-items: center;
    overflow: hidden;
    position: absolute;
    top: 0px;
    left: calc(var(--width_eraser)/2);
    width: calc(100% - var(--width_eraser));
    height: 400px;
    background-color: #995879;
    border-radius: 5px;
    opacity: 0;
    z-index: -1;
    transition: top 0.3s ease, opacity 0.3s ease, pointer-events 0.3s ease;
    pointer-events: none;
    display: flex;
```

```

    flex-direction: column;
}
#DropDownMenu button{
    margin-top: 5px;
    position: relative;
    width: 90%;
    height: 35px;
    border-radius: 5px;
    border: none;
    color: #fff;
    font-family: 'Montserrat', sans-serif;
    font-weight: 500;
    background-color: #633A4F;
    transition: background-color 0.1s ease;
}
#DropDownMenu button:hover{
    background-color: #7a4862;
}

#DropDownMenu.opened{
    top: 70px;
    opacity: 1;
    pointer-events: auto;
    animation: zChanger 0.3s forwards;
}
@keyframes zChanger{
    0% {
        z-index: -1;
    }
    99% {

```

```
    z-index: -1;
}
100% {
    z-index: 100;
}
}
```

Switcher.css

```
.Switcher * {
    padding: 0;
    margin: 0;
}
.Switcher {
    display: inline-block;
    justify-content: center;
    align-items: center;
    width: 10%;
    height: 20%; /* Чтобы выровнять по центру */
    position: relative; /* Важно для позиционирования */
    transform: scale(1.5); /* Масштабирование и перемещение */
    transform-origin: center; /* Масштабируем относительно центра Switcher */
    margin-left: 0%;
    margin-right: 0%;
    margin-top: 5%;
    margin-bottom: 5%;
}
.toggle-btn {
    background: no-repeat center;
    background-image: url('../public/css_assets/toggle_background.svg');
    border: 0px;
    border-radius: 99px;
```

```

width: 48px;
height: 21px;
cursor: pointer;
position: relative;
box-shadow: 1px 1px 10px black;
}
.thumb {
height: 21px;
width: 21px;
background-color: white;
border-radius: 99px;
position: absolute;
left: 0px;
top: 50%;
transform: translateY(-50%);
transition: left 0.4s ease; /* Плавная анимация движения */
}
.thumb.active {
left: calc(50px - 23px); /* Переместить в правую позицию */
}
.toggle-btn:hover {
border-color: #aaa;
}

```

UserSettings.css

```

#settings_group {
top: 0%;
left: 0%;
display: flex;
position: fixed;
justify-content: center;

```

```

align-items: center;
    width: 100vw;
height: 100vh;
z-index: -1;
background-color: transparent;
transition: background-color 0.3s ease-in-out;
}
#settings_group.opened{
    background-color: rgba(0, 0, 0, 0.5); /* Полупрозрачный фон */
    z-index: 10;
}
/*панель настройки аккаунта*/
#settings_window{
    overflow: hidden;
    display: flex;
    align-items: center;
    width: 150px;
    height: 225px;
    background-color: #633A4F;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
    border-radius: 10px;
    opacity: 0;
    transition: all 0.3s ease-in-out;
}
#settings_window.opened{
    opacity: 1;
    transform: scale(3);
}
#settings_conveyor{
    position: relative;

```

```

width: auto;
height: auto;
display: flex;
flex-direction: row;
justify-content: center;
align-items: center;
}
/*кнопки и страницы*/
/*панель main_page*/
#account_settings_btn{
width: 100%;
height: 25px;
background-color: #633A4F;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
transition: background-color 0.2s ease;
font-size: 7px;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
}
#account_settings_btn:hover{
background-color: #7A4F63;
}
/*панель account_settings_page*/
.btn{
width: 100%;
height: 13px;
background-color: #633A4F;

```

```
color: white;
border: none;
cursor: pointer;
transition: background-color 0.2s ease;
font-size: 7px;
font-family: 'Montserrat', sans-serif;
font-weight: 700;
}
.btn:hover{
background-color: #7A4F63;
}
```