**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Факультет **Прикладной информатики**

Направление подготовки **09.03.03 Прикладная информатика**

Образовательная программа **Мобильные и сетевые технологии**

**К У Р С О В О Й П Р О Е К Т**

Тема: «Разработка веб-приложения для обмена видео-потоками (ВКС)»

Обучающийся: Глазыкин Владимир Вадимович, К3140

Санкт-Петербург 2024

# Оглавление

[**Оглавление 2**](#_2yh8t6oe4fzt)

[**Введение 3**](#_wc1hs0qo095b)

[**Организация команды 3**](#_wtatcfb21763)

[**Моя роль в команде 5**](#_ylbro9718b2m)

[**Технологии и подходы 6**](#_sa92cfadcjhz)

[Компонентный подход 6](#_rpa1mjgiygvm)

[Управление состоянием 8](#_3zc1jg9eu3g4)

[Адаптивная вёрстка 10](#_kfndwr44m5wv)

[**WebRTC 12**](#_e4d558p0ii9t)

[**Мои впечатления 14**](#_d6jwny3h2t4h)

[**Итоги 15**](#_rzns47lajrjg)

[**Список использованных источников 16**](#_dto3y255mydr)

# Введение

В наше время потребность в онлайн-встречах сильно возросла.

Коллеги, друзья, родственники, зачастую находящиеся на расстоянии сотен километров друг от друга, поддерживают друг с другом контакт через различные сервисы для совместной работы и отдыха, интернет-звонков и конференций - например, Zoom, Google Meet, Skype.

Востребованность подобных сервисов и активная конкуренция между ними на мировом рынке показывают актуальность сферы видеоконференцсвязи и онлайн-коммуникаций в целом.

Итак, цель курсовой работы - создание сервиса для проведения многопользовательских видеоконференций и звонков формата “один на один” с использованием технологии WebRTC в формате веб-приложения.

Среди основных задач проекта:

* Проектирование макета пользовательского интерфейса в Figma
* Вёрстка пользовательского интерфейса на фреймворке React
* Реализация механизма обмена видео- и аудио-потоками через WebRTC
* Создание сигнального сервера на Node.js
* Реализация бэкенда приложения на Node.js+Express
* Подключение базы данных PostgreSQL к бэкенду
* Написание автотестов для QA-тестирования приложения

# Организация команды

# 

Чтобы реализовать поставленные в рамках проекта задачи, наш руководитель Николай Константинов согласовал роли каждого члена команды.

Так, к началу непосредственной работы над проектом, в нашей команде образовались: фронтенд-разработчик (Егор Янин), бэкенд-разработчик (Андрей Останин), фуллстек-разработчик (я), дизайнер (Александр Кучин) и QA-специалист (Буй Тхук Хуен).

Мы вступили в групповой чат в Telegram, где на протяжении всей работы над проектом периодически проводили созвоны - чтобы обсудить поставленные задачи или уточнить организационные подробности.

На мой взгляд, мы довольно быстро и хорошо сработались.

Первым делом оба разработчика и я прошли вводный видеоурок по созданию фуллстек-приложения на React и Node.js, а дизайнер создал первый вариант макета приложения в Figma.

Я с React уже был знаком, но повторить некоторую информацию точно не было лишним.

В связи с тем, что видеоурок был записан ещё в 2019 году, в процессе его прохождения возникли некоторые трудности, в основном касавшиеся работы с React - за несколько лет фреймворк сильно изменился в результате многочисленных обновлений, поэтому информация из некоторых частей видеоурока оказалась устаревшей, а применение использованных в уроке методов вызывало ошибки. Мы с коллегами несколько раз созванивались, чтобы обсудить возникшие ошибки, и в результате успешно завершили прохождение этого видеоурока, в проблемных местах заменив устаревший синтаксис React и конфигурационных файлов на более современный.

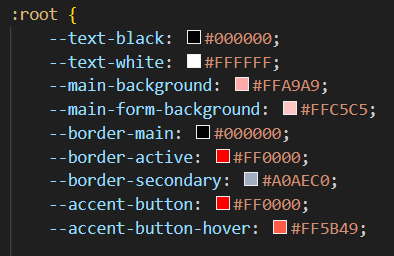
Наш руководитель зачастую также участвовал в подобных созвонах, проверял написанный нами код и давал рекомендации.

# Моя роль в команде

Так как я довольно быстро вспомнил основы разработки на React, а с Node.js и PostgreSQL работаю на постоянной основе, я приступил к реализации базовой вёрстки основных компонентов приложения и порученных мне экранов, а также поднял сервер для базы данных PostgreSQL и заполнил таблицы тестовыми данными для отладки приложения в процессе разработки.

В процессе вёрстки я сразу же применил компонентный подход.

Также я вынес глобальные css-стили в отдельный файл с переменными, чтобы при необходимости не редактировать стили в каждом файле вручную.



css-переменные

Тем временем, дизайнер уже завершил работу над первым вариантом макета приложения и согласовал необходимые детали и правки с руководителем проекта.

Макет получился удобным и детальным: основные элементы (кнопки, текстовые поля, заголовки) были вынесены на отдельные фреймы, для многих элементов были определены варианты состояний (states).

По такому можно сверстать хороший UI.

# Технологии и подходы

Итак, клиентскую часть приложения мы пишем на языке TypeScript с использованием фреймворка React.

Выбирая современный стек, важно понимать и современные требования к разработке веб-приложений.

Вот некоторые из реализованных в нашем проекте требований:

## **Компонентный подход**

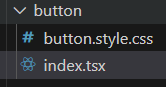
- Форма организации структуры проекта, предусматривающая вынесение многократно использующихся элементов пользовательского интерфейса в отдельные компоненты (файлы), так называемая “декомпозиция”.

Зачем нам компонентный подход?

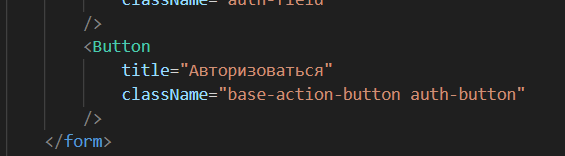
* Позволяет ускорить процесс разработки за счёт написания меньшего количества кода
* Улучшает масштабируемость приложения благодаря созданию готовых элементов интерфейса
* Упрощает тестирование отдельных компонентов

В нашем проекте основные компоненты UI, например, кнопки, поля ввода текста, попап (элемент “всплывающего сообщения”), вынесены в отдельные папки. В каждой папке основной код компонента находится в файле index.tsx, а рядом с ним в файле .style.css лежат его стили.

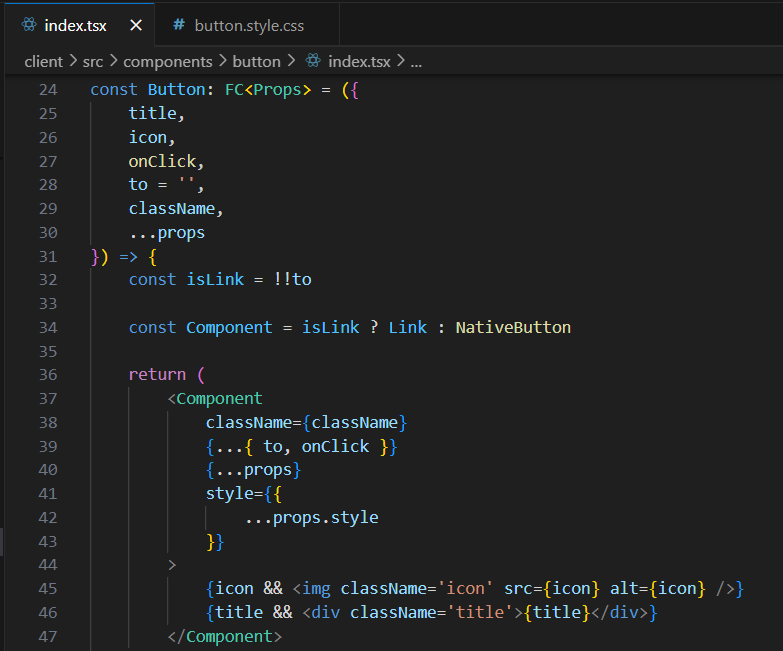
Например, компонент Button:



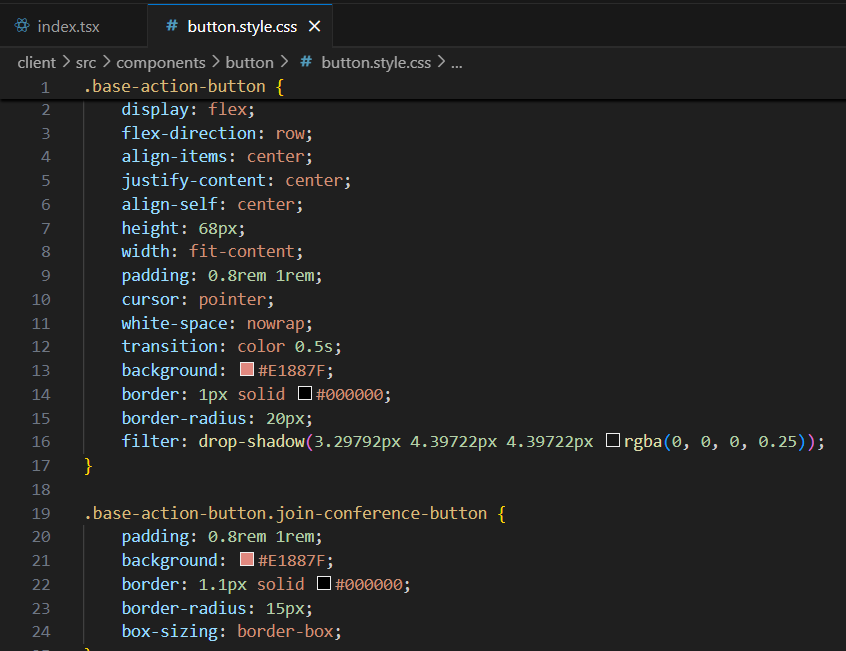
Структура папки с компонентом



Использование компонента в коде другого компонента



Часть index.tsx компонента



Часть .style.css компонента

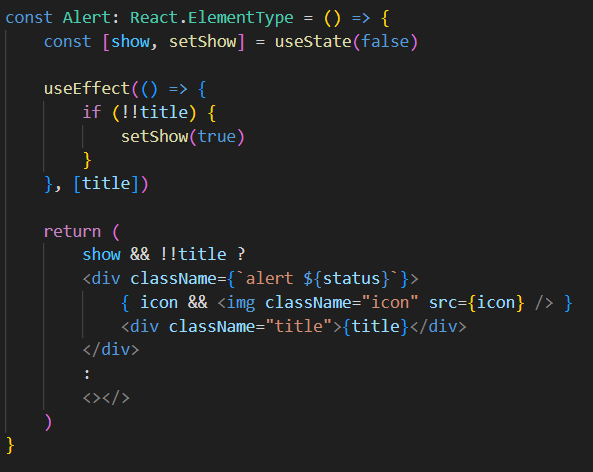
## Управление состоянием

- Способ организации данных для взаимодействия между разными частями приложения. Особенно полезен в компонентном подходе.

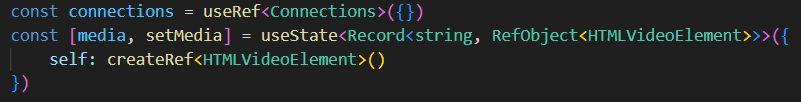
Предусматривает:

* Создание локального хранилища для любого компонента (хук useState) для управления уникальным для данного компонента состоянием без задействования глобального хранилища
* Создание глобального хранилища (хук useReducer), к которому может обращаться любой компонент приложения и изменять данные в реальном времени

В компонентах нашего приложения используется локальное хранилище useState - например, чтобы показать/скрыть попап с предупреждением или динамически изменить источник видео.



Использование локального хранилища useState в совокупности с хуком useEffect, чтобы показать/скрыть компонент Alert в зависимости от того, есть ли значение в переменной title



useState используется для динамического обновления ссылки на видеопоток (например, пользователь включил/выключил камеру во время конференции)

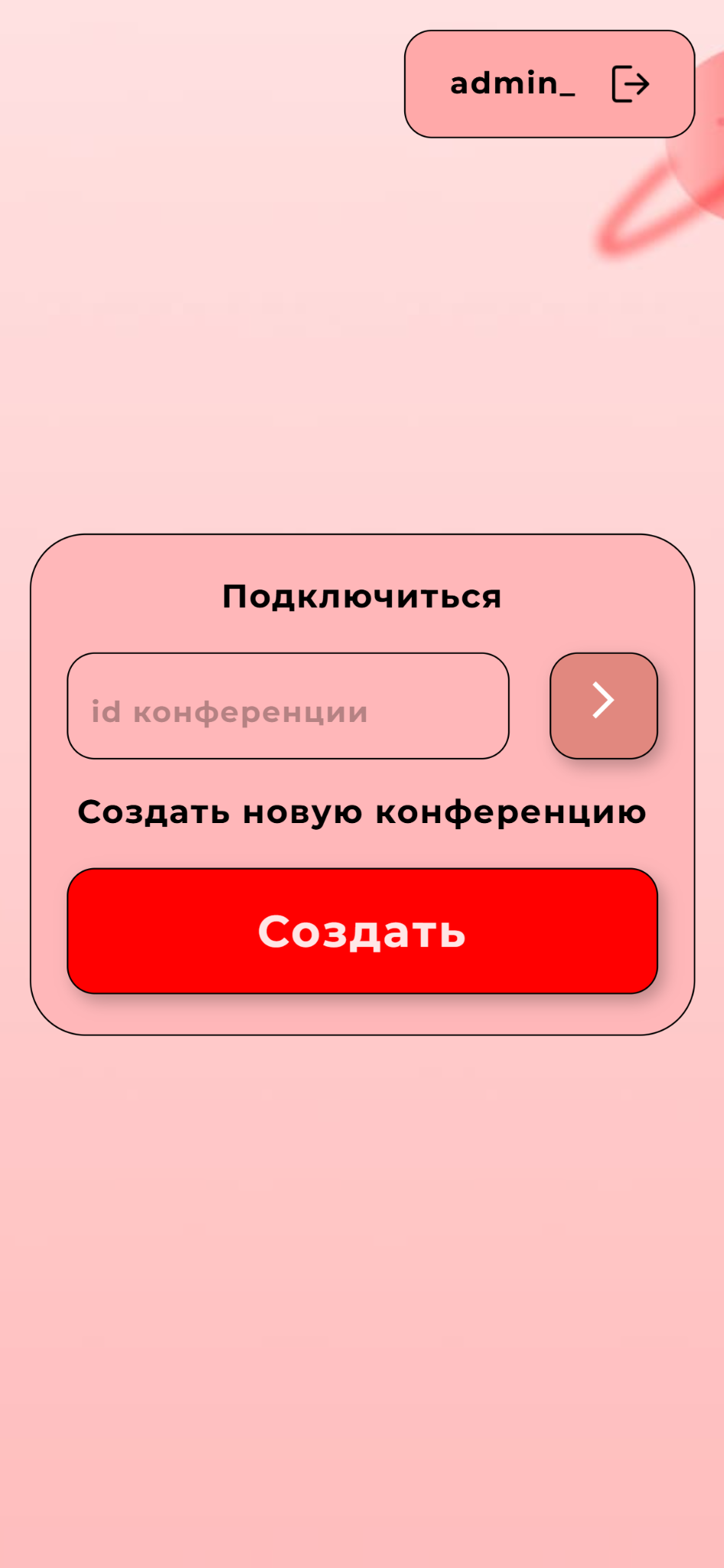
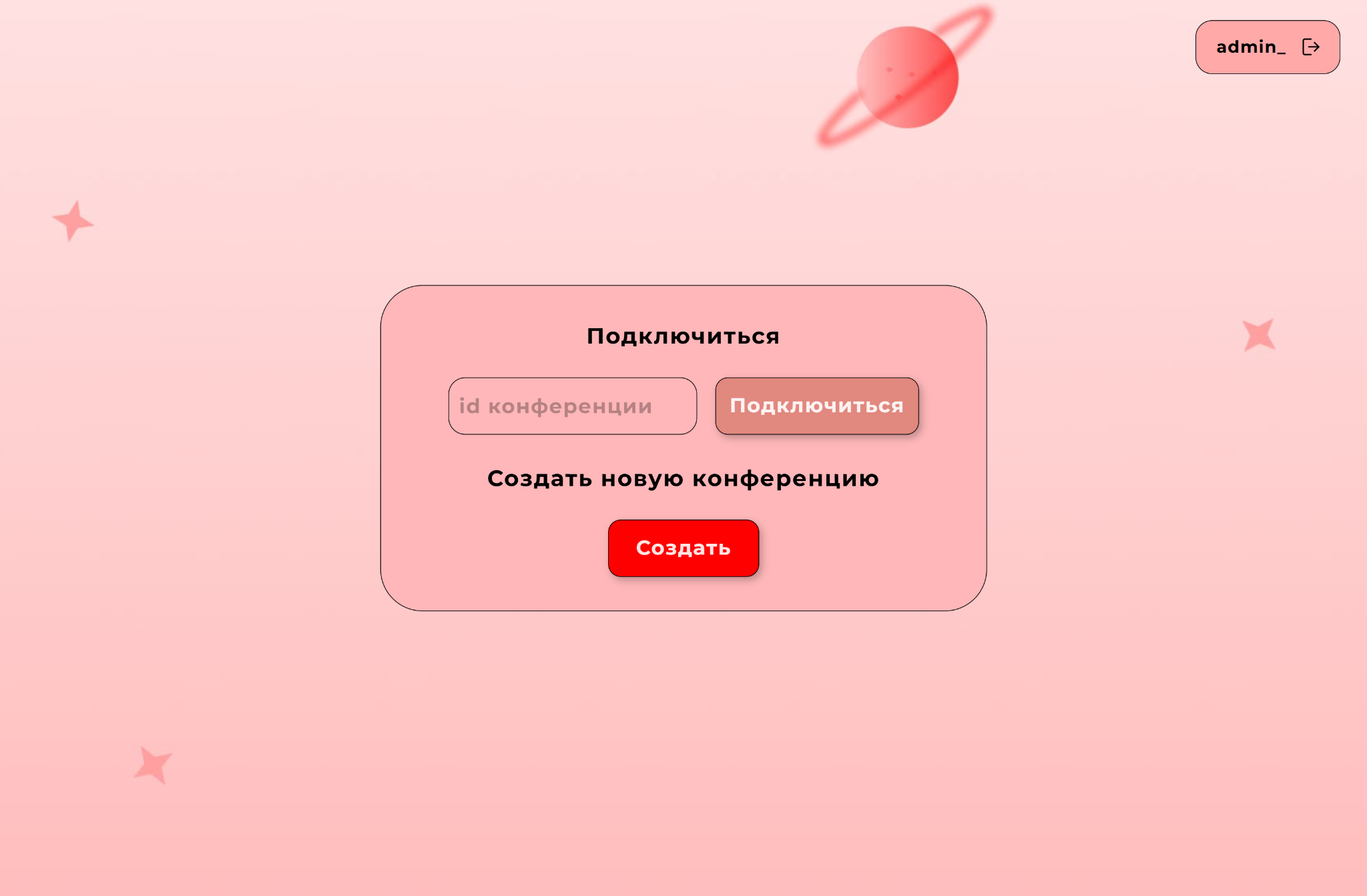
## Адаптивная вёрстка

Подход к разработке UI, при котором интерфейс приложения автоматически подстраивается под размеры экрана устройства.

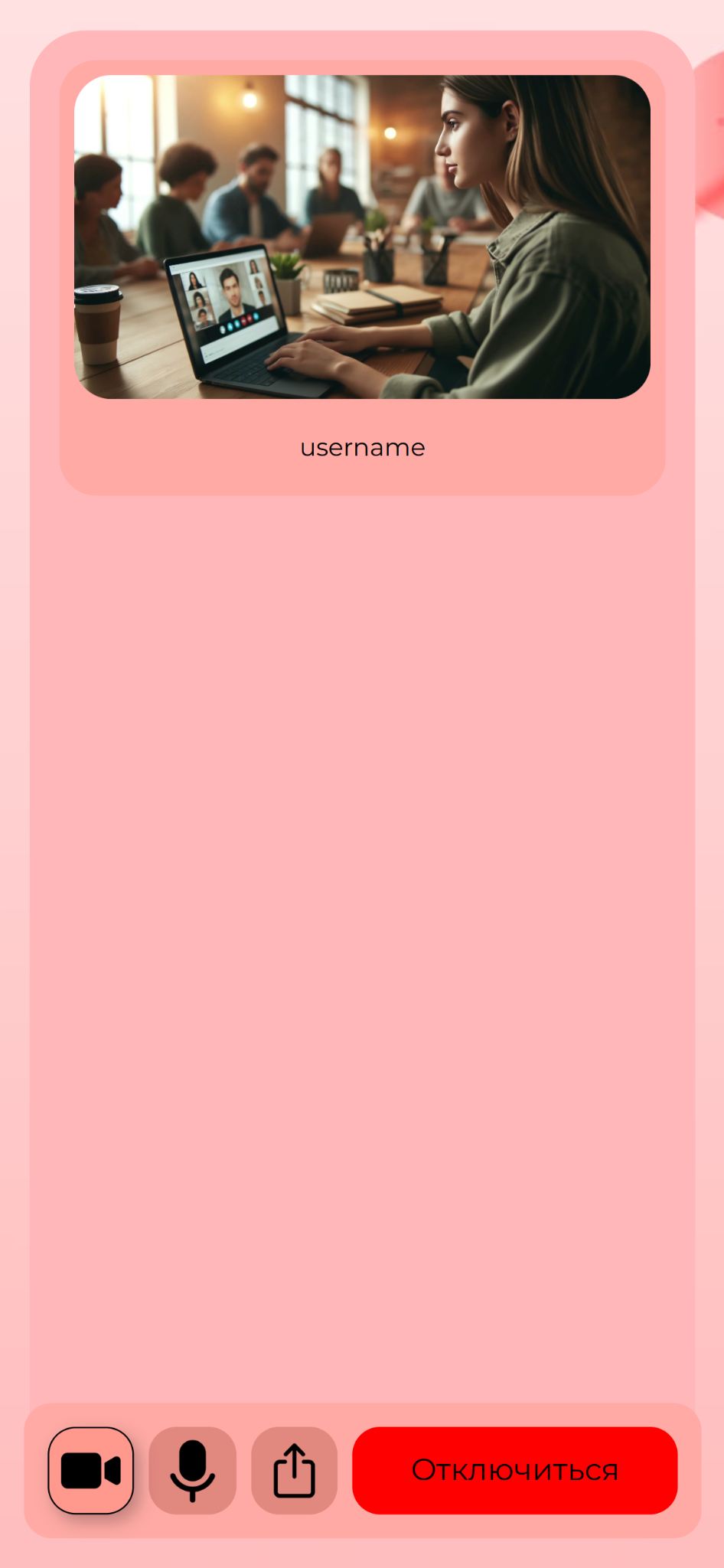
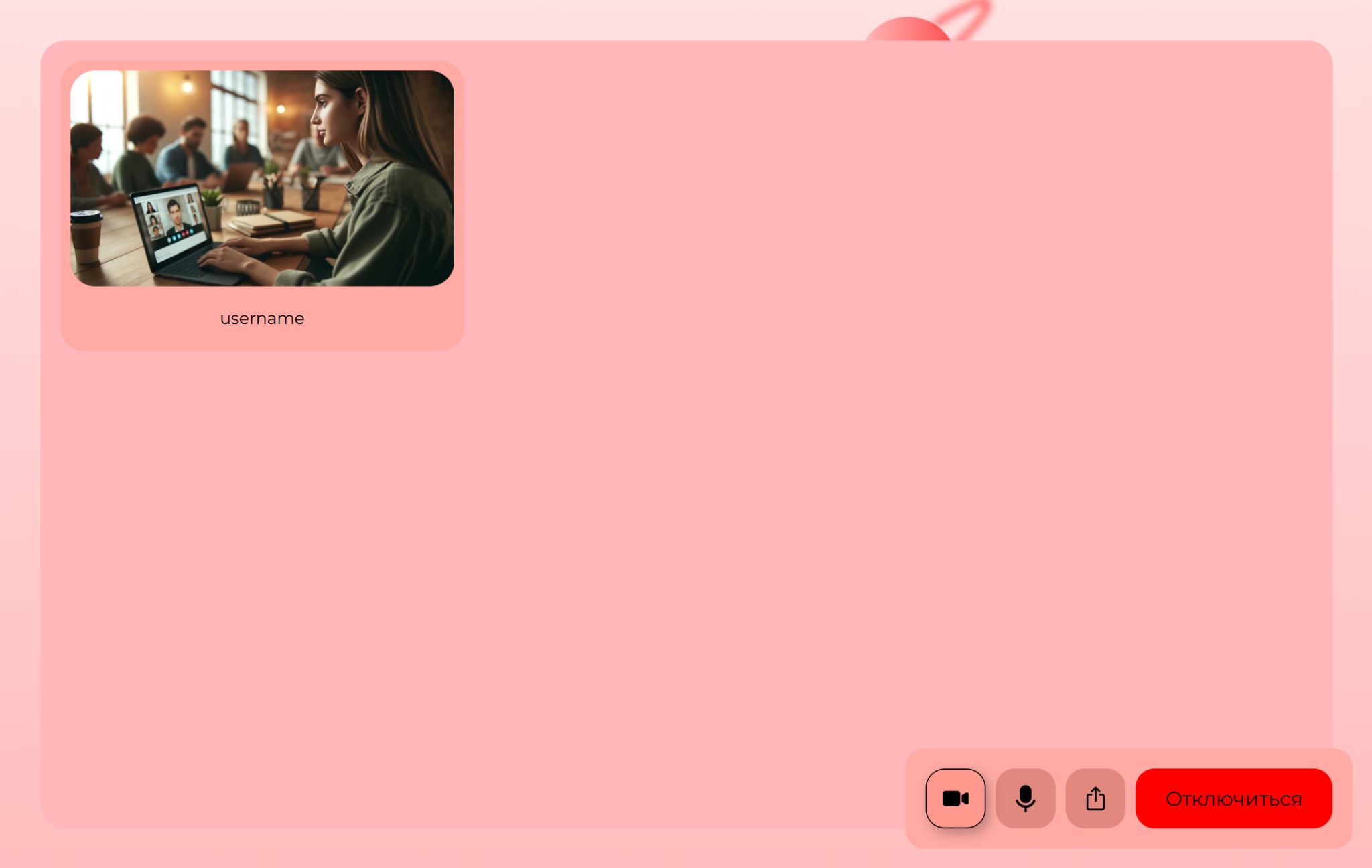
Приложение удобно использовать как на компьютере, так и на телефоне или планшете.

В зависимости от ширины и высоты экрана изменяются размеры компонентов, отступы, размер шрифта, некоторые элементы перемещаются.

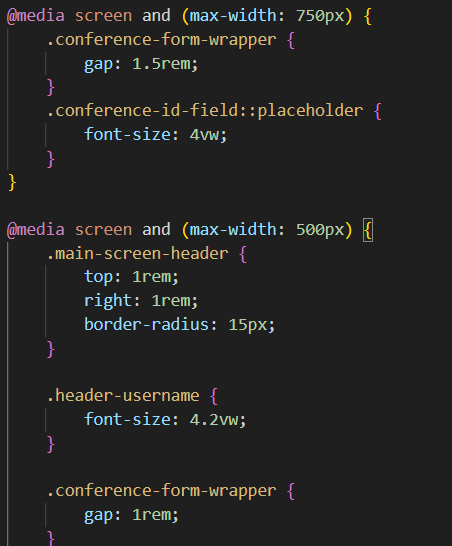
Требования к дизайн-макету этого приложения изначально не предусматривали адаптивной вёрстки, поэтому я составил стили для неё самостоятельно.



Адаптивная вёрстка главного экрана на широком экране (>750px в ширину) и на телефоне (<500px в ширину)



Адаптивная вёрстка экрана конференции с тестовым видео-фреймом



Адаптивные стили в файле .style.css главного экрана через @media-запросы в css

# WebRTC

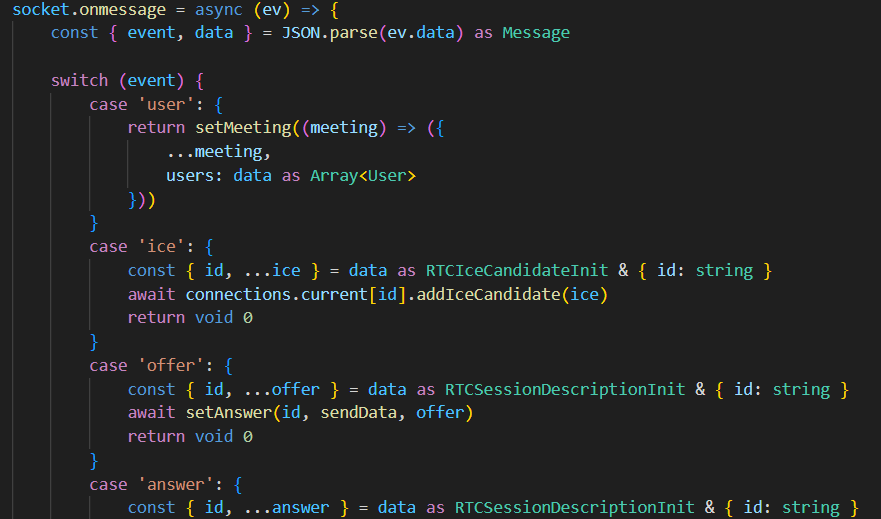
Для реализации возможностей передачи видео- и аудио-потоков между участниками конференции в реальном времени наш сервис использует технологию WebRTC.

WebRTC позволяет обмениваться данными в формате peer-to-peer, то есть напрямую между устройствами клиентов, что значительно снижает задержки и не сильно нагружает сервер приложения.

В данном проекте мы реализовали передачу медиа-потоков по схеме Mesh-сети, т.е. обработка потоков осуществляется на клиентских устройствах.

Для небольших конференций (на 5-10 человек) или звонков “один на один” это отличное решение, так как трафик не проходит через сторонние сервера, и, как уже сказано выше, задержки в передаче данных существенно уменьшаются по сравнению с методом “клиент-сервер-клиент”.

В перспективе эффективным решением будет реализовать схемы SFU или MCU для конференций с большим количеством участников, чтобы не вызывать большой нагрузки на клиентские устройства, однако такой подход потребует создания отдельного медиа-сервера, через который будут проходить медиа-потоки.



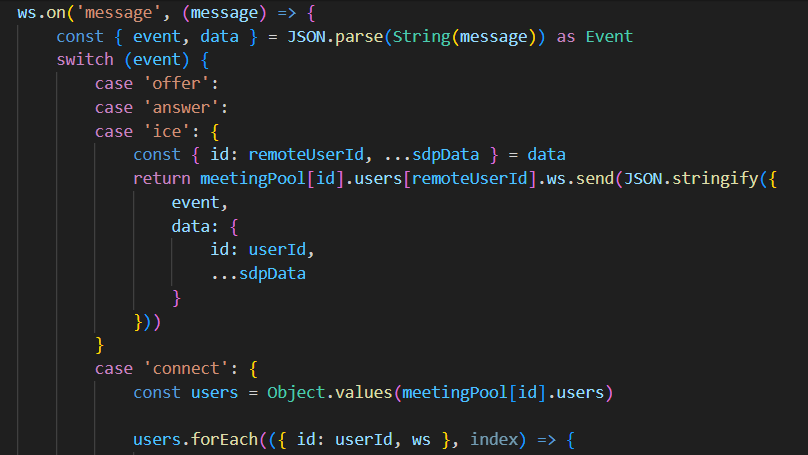
Обработчик событий на клиенте

Для работы WebRTC требуется также сигнальный сервер.

Сигнальный сервер не участвует в передаче аудио- или видео-потоков.

Он нужен, чтобы помочь участникам конференции найти друг друга и согласовать параметры соединения, поэтому задействуется только в момент подключения клиентов.

Например, он передаёт ip-адреса и username клиентов, а также уведомляет приложение, когда кто-то из пользователей подключается ко встрече или выходит из неё.



Обработчик событий на сигнальном сервере

# Мои впечатления

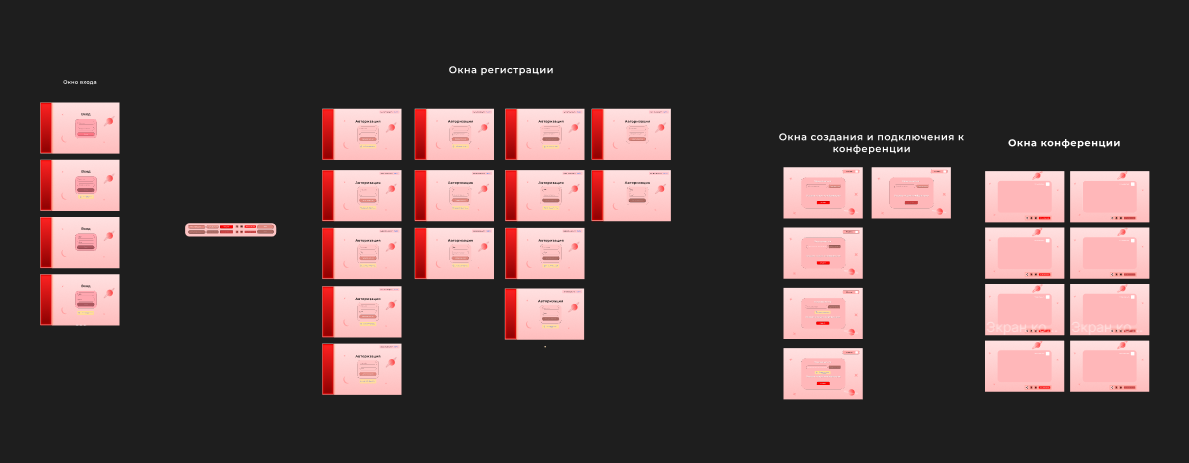
В рамках курсового проекта я в первый раз применил технологию WebRTC на практике. Помимо этого, я улучшил свои навыки работы с фреймворком React, а в процессе прохождения видеоурока изучил особенности миграции со старых версий React и Vite на современные.

С интеграцией WebRTC в проект помогали ChatGPT, Copilot и руководитель проекта.

Так, Николай Константинов отправил нам одну из своих лекций по WebRTC, а также записал несколько видео-рекомендаций по дизайну интерфейсов и архитектуре веб-приложений.

На созвонах я задавал ему вопросы об особенностях работы этой технологии, а также уточнял, как мне лучше реализовать те или иные функции и обработчики и каким образом проект может быть масштабирован в будущем.

Работа в команде проходила гладко. Каждый член команды так или иначе участвовал в реализации курсового проекта - от дизайна и разработки до тестирования и деплоя.



Скриншот нашего рабочего пространства в Figma

# Итоги

Цели проекта, обозначенные в рамках курсового проекта, достигнуты.

Сервис создан и работает.

На мой взгляд, наш сервис, покуда в нём существует потребность, не имеет конечной стадии развития - как большинство приложений и других программных продуктов на массовом рынке.

Я внёс свой вклад в реализацию проекта: верстал компоненты и экраны приложения, инициализировал базу данных, участвовал в тестировании приложения, помогал другим членам команды.

Также внесли свой вклад в развитие проекта эти замечательные люди:

Андрей Останин - бэкенд-разработчик

Егор Янин - фронтенд-разработчик

Александр Кучин - дизайнер

Буй Тхук Хуен - QA-ассистент

Николай Константинов - руководитель

# Список использованных источников

* ChatGPT - <https://chatgpt.com>
* GitHub Copilot - <https://github.com/features/copilot>
* Форум StackOverflow - https://stackoverflow.com
* WebRTC API - <https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API>
* Справочник по веб-разработке W3Schools - https://w3schools.com
* Справочник React - <https://reactdev.ru>
* Документация Vite - https://vite.dev
* Видеоурок по созданию web-приложения на React и Node.js - [Создайте Fullstack-приложение за 3 часа с помощью React и NodeJS: MERN](https://www.youtube.com/watch?v=ivDjWYcKDZI&ab_channel=%D0%92%D0%BB%D0%B0%D0%B4%D0%B8%D0%BB%D0%B5%D0%BD%D0%9C%D0%B8%D0%BD%D0%B8%D0%BD)

Дата последнего обращения к источникам: 05.12.2024