

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра _____ *Вычислительной техники* _____
(полное название кафедры)

Утверждаю

Зав. кафедрой Якименко А.А.

(подпись, инициалы, фамилия)

«___» _____ 2018г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА
НА ТЕМУ**

Разработка сервиса видеоконференцсвязи для врачебных консультаций
(тема работы)

Автор дипломной работы *Тонких Роман Константинович*
(подпись студента, выполнившего дипломный проект или работу)

Тонких Роман Константинович Группа *АВТ-409*
(фамилия, инициалы студента) (в которой обучался студент)

Автоматики и вычислительной техники
(полное название факультета)

Направление подготовки *09.03.01 Информатика и вычислительная техника*
(код и наименование специальности)

Руководитель работы _____
(подпись, дата)

Васюткина Ирина Александровна
(фамилия, инициалы)

Новосибирск 2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Вычислительной техники

(полное название кафедры)

Утверждаю

Зав. кафедрой Якименко А.А.

(подпись, инициалы, фамилия)

«___» _____ 2018г.

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

студенту Тонких Роману Константиновичу группы АВТ-409
(фамилия, инициалы)

1. Тема Разработка сервиса видеоконференцсвязи для врачебных консультаций
(полное название темы)

утверждена приказом по НГТУ № _____ от «___» _____ 201__ г.,
изменена приказом по НГТУ № _____ от «___» _____ 201__ г.

2. Дата представления работы к защите «___» _____ 2018 г.

3. Исходные данные для проекта _____
Среда разработки WebStorm, браузеры, Postman клиент, средство виртуализации
окружения Docker, язык программирования JavaScript, библиотека разработки
пользовательских интерфейсов React, фреймворк Express, ORM-система mongoose.

4. Содержание пояснительной записки _____

Введение

4.1. Постановка задачи

4.2. Обзор существующих аналогов

4.3. Функционал приложения

4.4. Архитектура и реализация функционала приложения

4.5. Потенциальные доработки системы

Заключение

Список используемых источников

Тонких Роман

Задание принял к исполнению

(подпись студента, дата)

Константинович

(фамилия, инициалы студента)

Дипломная работа сдана в ГЭК № _____, тема сверена с данными приказа

(подпись секретаря государственной экзаменационной комиссии по защите ВКР, дата)

Реферат

Тонких Роман Константинович. Разработка сервиса видеоконференцсвязи для врачебных консультаций.

Работа выполнена на 67 страницах, включает введение, 5 глав, заключение, 36 рисунков, 5 таблиц. Список используемых источников включает 10 наименований.

Место выполнения выпускной квалификационной работы – НГТУ, АВТФ, руководитель – к.т.н. доцент, Васюткина И.А., 2018г.

Цель работы: разработка сервиса для организации удаленных *on-line* консультаций пациентов врачами с помощью средств видеосвязи. Проект является инструментом реализации телемедицины.

В работе приведен сравнительный анализ аналогичных сервисов. Продемонстрирован разработанный функционал приложения. Дано описание используемых архитектурных решений и технологий для решения поставленных задач. Проведен анализ потенциальных доработок приложения.

В рамках выпускной квалификационной работы выполнена разработка информационной системы для удаленных видео-консультаций. Для организации передачи медиа-потокa использовалась технология *WebRTC*.

Основные термины: телемедицина, видео-консультации, *WebRTC*, *P2P*, *UI*, *API*, *SDP*, *ICE*, *WebSocket*, *MVC*, *Flux*, *HTTP*, *react*, *redux*, *nodejs*, *js*.

Оглавление

Введение.....	8
1 Постановка задачи.....	9
1.1 Актуальность темы	9
1.2 Цель работы	10
1.3 Задачи, подлежащие решению для достижения цели	10
1.4 Методология, средства решения задач	10
2 Обзор существующих аналогов	14
2.1 Яндекс Здоровье.....	14
2.2 Doc+ Онлайн.....	15
2.3 Доктор рядом.....	16
2.4 Онлайн Доктор	17
2.5 Общие требования и процесс получения консультаций	18
2.6 Итоги сравнения.....	18
3 Функционал приложения.....	20
3.1 Аутентификация, авторизация, регистрация	20
3.2 Главная страница	23
3.3 Функционал пользователей	25
3.4 Окно видео-консультаций.....	30
3.5 Личный профиль	31
3.6 Тест на соответствие требованиям видео-консультаций.....	32
4 Архитектура и реализация функционала приложения.....	34
4.1 Инструменты и технологии реализации	34

4.1.1 Клиент-серверная архитектура.....	35
4.1.2 Пиринговая архитектура	36
4.2 Сервер.....	37
4.2.1 <i>REST</i> -Архитектура	38
4.2.2 Структура проекта	38
4.2.3 Объектное представление базы данных	41
4.2.4 Схема запрос-ответ	41
4.2.5 Диаграмма классов	42
4.2.6 Виртуализация окружения разработки.....	43
4.2.7 Итог разработки сервера	45
4.3 Разработка клиентской части.....	45
4.3.1 <i>Flux</i> -паттерн.....	46
4.3.2 Структура проекта	48
4.3.3 Сторонние эффекты.....	49
4.3.4 Использование компонентов	51
4.4 Технология <i>WebRTC</i>	54
4.4.1 Использование и детали технологии <i>WebRTC</i>	55
4.4.2 Логическое соединение – <i>SDP</i>	56
4.4.3 Физическое соединение	57
4.4.4 Сигнальный сервер	58
4.5 Технология веб-сокеты.....	59
4.6 Итоги	61
5. Потенциальные доработки системы	62

5.1 Хранение персональных данных.....	62
5.2 Видео-конференция	62
5.3 Мобильное приложение	62
Заключение	64
Основные обозначения и сокращения	65
Список используемых источников.....	67

Введение

В последние годы с появлением быстрого, стабильного и доступного интернета, мощных телефонов и компьютеров появилась возможность создавать автоматизированные сервисы для уменьшения стоимости услуг путем ликвидации промежуточных, ненужных звеньев в бизнес-процессах. Например, сервисы заказа такси, такие как *Uber*, *Yandex Taxi* и *Get* пришли на смену традиционным диспетчерским службам. Всего за несколько лет стоимость услуг такси снизилась на порядок, а качество обслуживания в свою очередь только улучшилось. На данный момент активно развивается рынок оказания медицинских услуг удаленно. Конечно, полностью очную медицину подобный сервис не ликвидирует, однако однозначно меняет представление об оказании медицинских услуг.

Сервис удаленных консультаций поможет пациентам, проживающим в труднодоступных местах, неподвижным пациентам, а также тем кому необходима срочная медицинская консультация. Для получения услуг нет необходимости стоять в очереди, необходимо лишь записаться на прием, либо связаться со свободным данным момент врачом. Внедрение услуг удаленных видео-консультаций потенциально может снизить нагрузку на медицинские учреждения, а также стоимость врачебных услуг.

Разработанное в выпускной квалификационной работе приложение по организации медицинских *on-line* консультаций с некоторыми оговорками может внедряться и использоваться по прямому назначению.

1 Постановка задачи

1.1 Актуальность темы

Востребованность телемедицинских услуг становится очевидной при рассмотрении результатов исследования, проведенного в высококвалифицированном учреждении – Институте хирургии им. А.В. Вишневского РАМН. Из 12–15 тыс. обратившихся за год больных лишь 10% нуждались в госпитализации и хирургическом лечении. Следовательно, остальные 90% обратившихся тратили деньги, и немалые, время и здоровье лишь на то, чтобы получить квалифицированную консультацию [7]. В США телемедицину начали внедрять в 2008 году. Но уже через восемь лет, к 2016 году, в стране уменьшилась обращаемость пациентов за медпомощью амбулаторно на 70 процентов, число койко-мест - на 19 процентов, стационарных больниц - на 26 процентов [8]. Таким образом, телемедицина высвобождает огромные ресурсы.

Однако существует ряд и проблем. На данный момент эта область находится вне правового поля. Возникает острый правовой вопрос - можно ли врачам оказывать консультативные услуги через интернет или по телефону. Можно предположить, что легализация телемедицины сильно изменит структуру медицинского рынка, так как позволит получать лицензию не только на целую компанию, но и на частное лицо. Врачи давно могут консультировать и отвечать на вопросы пользователей на форумах и в соцсетях, не неся за это какой-либо ответственности. Лицензирование онлайн-консультаций позволит повысить персональную ответственность врачей и сделать такого рода услуги доступными для широкой аудитории.

Разработка приложения для проведения удаленных видео-консультаций, несомненно является актуальной.

1.2 Цель работы

Конечной целью данного проекта является разработка сервиса для организации удаленных *on-line* консультаций пациентов врачами с помощью средств видеосвязи. Проект станет инструментом реализации телемедицины.

1.3 Задачи, подлежащие решению для достижения цели

Для достижения поставленной цели были определены следующие задачи:

1. Произвести поиск и анализ существующих аналогичных сервисов.
2. Спроектировать архитектуру приложения.
3. Выбрать технологии для реализации как клиента, так и сервера приложения.
4. Выбрать метод передачи потока данных между пользователями.
5. Реализовать клиент-серверное приложение с использованием выбранных технологий.

1.4 Методология, средства решения задач

Для выполнения поставленных задач без современных средств разработки обойтись сложно. Современные инструменты позволяют вести разработку быстрее и качественнее чем это было раньше. Разрабатывать продукт становится проще. За качеством кода и соблюдением стандартов кодирования помогает следить среда разработки. Отладка, тестирование, поддержка, синхронизация и развёртка проекта производиться с помощью специальных утилит. В конечном итоге это сказывается на стоимости разработки и поддержки проекта.

Операционная система

Основной операционной системой для разработки и функционирования информационной системы как для клиента, так и для сервера была выбрана ОС *Linux Debian 9.4 x64*. Данная ОС была выбрана за следующие достоинства:

- бесплатная открытая разработка;
- наличие командного терминала;
- гибкая система настройки безопасности;
- быстроедействие;
- большое сообщество.

Среда разработки

WebStorm v2018.1.4 - средство разработки от компании *JetBrains*. Предназначена для веб-разработки. В частности, поддерживает различные диалекты *JavaScript*. Есть функция генерации кода, авто-исправления, встроенная оболочка для работы с системой контроля версий, система плагинов и открытый репозиторий с ними. Использовалась для разработки клиентской и серверной частей приложения. Продукт является платным. Получить лицензию на продукт позволил статус студента НГТУ.

Проектирование

Для проектирования системы, визуализации диаграммы классов, диаграмм последовательностей, диаграммы вариантов использования (*use-case* диаграмма) и других диаграмм *UML* использовалась программа *StartUML*.

Шаблоны проектирования

Для решения задач при разработке использовались общепринятые подходы к проектированию информационных систем – шаблоны (паттерны) проектирования.

Например, следующие: поведенческие паттерны “наблюдатель” и “цепочка обязанностей”, архитектурный паттерн *MVC*, паттерн обработки объектно-

реляционных методанных “репозиторий”, паттерн первоначальной обработки входящего *http*-запроса “фронт-контроллер” и др.

Подходы к разработке

Язык *Javascript* позволяет писать как в функциональном стиле, так и объектно-ориентированном. В работе были использованы оба подхода. ООП подход для серверной стороны и смесь ООП и ФП для реализации клиентской части приложения.

Postman

Во время разработки *API* сервера для проверки написанного функционала приложения использовался *Postman*-клиент. *Postman* даёт возможность отправлять на сервер *HTTP*-запросы с указанием заголовка и тела запроса и получать обратно ответы. Является разработкой компании *Google* и распространяется бесплатно.

Система контроля версий

Для контроля версионности программного кода использовалась система контроля версий *git*. *Git* является полезным инструментом, в особенности для разработки в команде. Инструмент позволяет фиксировать внесенные изменения маркируя их. В ходе разработки, если появится такая необходимость, можно вернуться на какой-либо зафиксированный этап назад и продолжить разработку с него. В качестве *Git* сервера использовался бесплатный *BitBucket* сервис.

Вспомогательные инструменты

При разработке клиентских приложений на технологии *react-redux* есть необходимость отслеживать внутреннее состояния и свойства компонентов, их жизненный цикл, общее состояние приложения и зависимость всего ранее перечисленного от момента исполнения. Для этого используются библиотека *react/redux-dev-tools* которая даёт возможность отладки приложения по шагам. Библиотека использовалась исключительно в целях разработки и отладки. В

окончательной версии продукта библиотека исключается из списка зависимостей.

2 Обзор существующих аналогов

Для обзора были выбраны 4 сервиса по оказанию удаленных медицинских консультаций, воспользоваться которыми может каждый - “Doc+ Онлайн”, “Доктор рядом”, “Онлайн Доктор” и “Яндекс.Здоровье”.

2.1 Яндекс Здоровье

Компания Яндекс запустила сервис “Яндекс Здоровье” в 2016 году, который предоставляет возможность оказывать медицинские услуги от разных врачей с помощью видео-консультаций. Имеется возможность получить консультации от терапевта, педиатра, гинеколога и др. врачей, а также получить услугу расшифровки результатов анализов. Плюс ко всему есть консультации ветеринара.

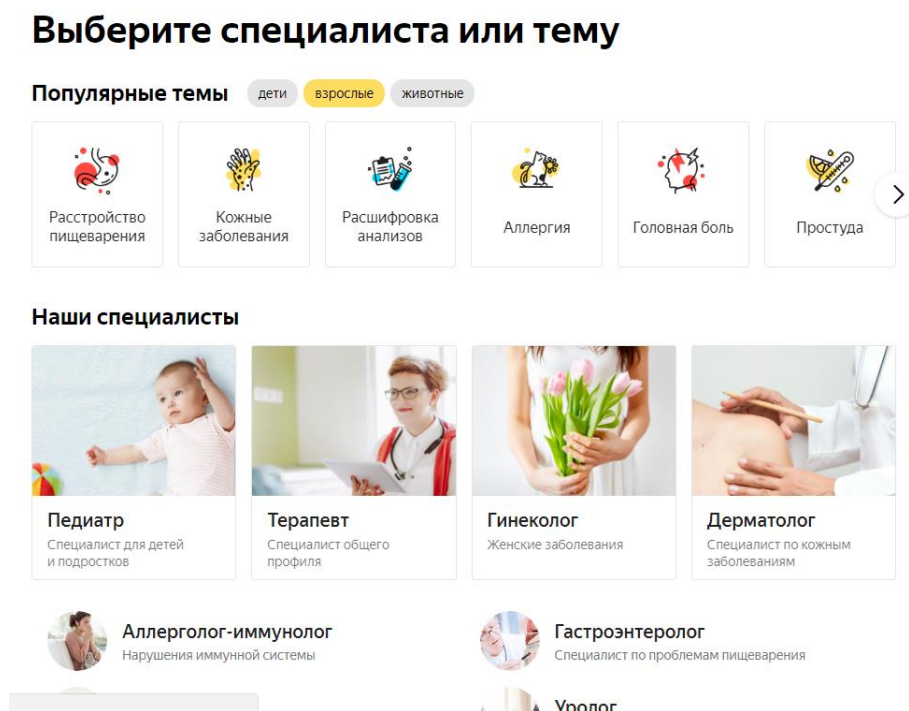


Рисунок 2.1 – сервис “Яндекс Здоровье”

Одной из особенностей сервиса является технология передачи заявки свободному врачу, что сокращает время ожидания консультации. Данная

особенность имеет и недостаток в виде получения консультации от ранее незнакомого врача, что может ухудшить качество услуги.

Преимущества:

- низкие цены;
- отсутствие очередей;
- множество врачей;
- история приёмов;
- возможность напоминаний.

Недостатки:

- не всегда “свой” врач;
- отсутствие своего мед.центра.

2.2 Doc+ Онлайн

Сервис по оказанию медицинских услуг на базе частного медицинского учреждения. Одно из направлений – видео консультации врачей на расстоянии.

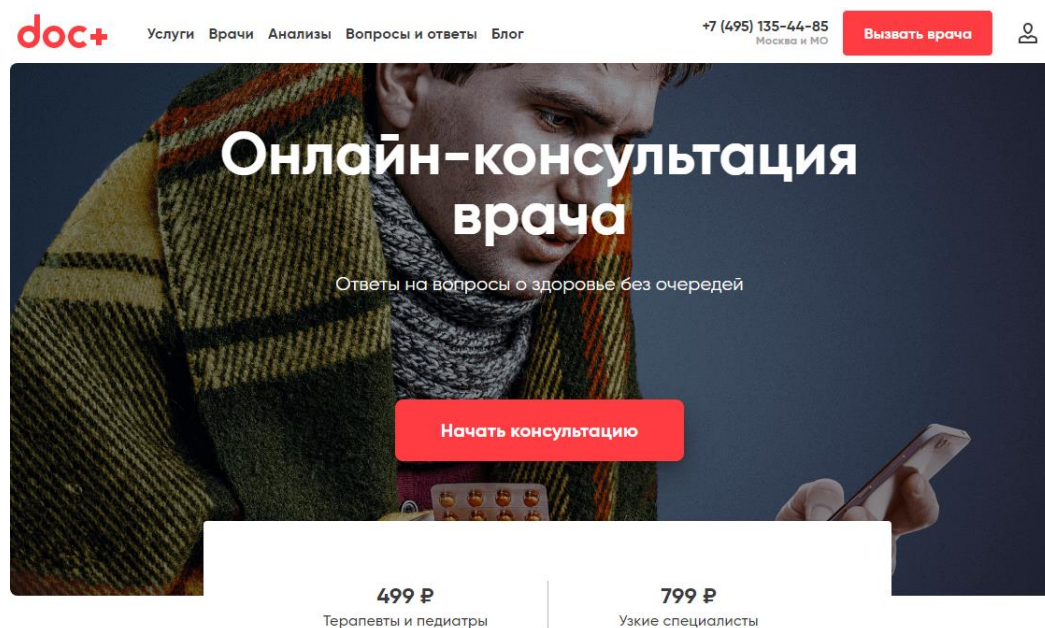


Рисунок 2.2 – сервис “Doc+ онлайн”

Преимущества:

- отсутствие очередей;
- множество врачей;
- свой медицинский центр;
- низкие цены.

Недостатки:

- неудобный интерфейс приложения;
- нет возможности получить консультацию в выходной день;
- видео-консультаций не являются основным направлением.

2.3 Доктор рядом

Также как и «Doc+ Онлайн» является частной медицинский клиникой и предоставляет услуги удалённых консультаций. Имеет множество врачей различных направлений и специализаций. Главным недостатком является предоставление услуг через программу “Скайп”. Из этого вытекают некоторые неудобства в виде необходимости устанавливать программу.

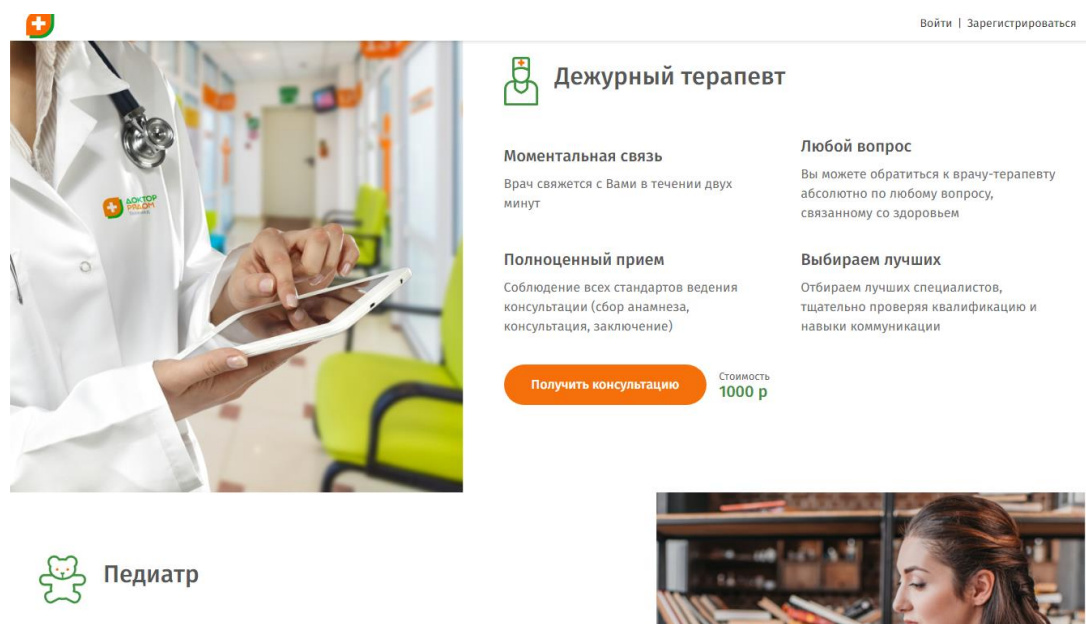


Рисунок 2.3 – сервис “Доктор рядом”

Преимущества:

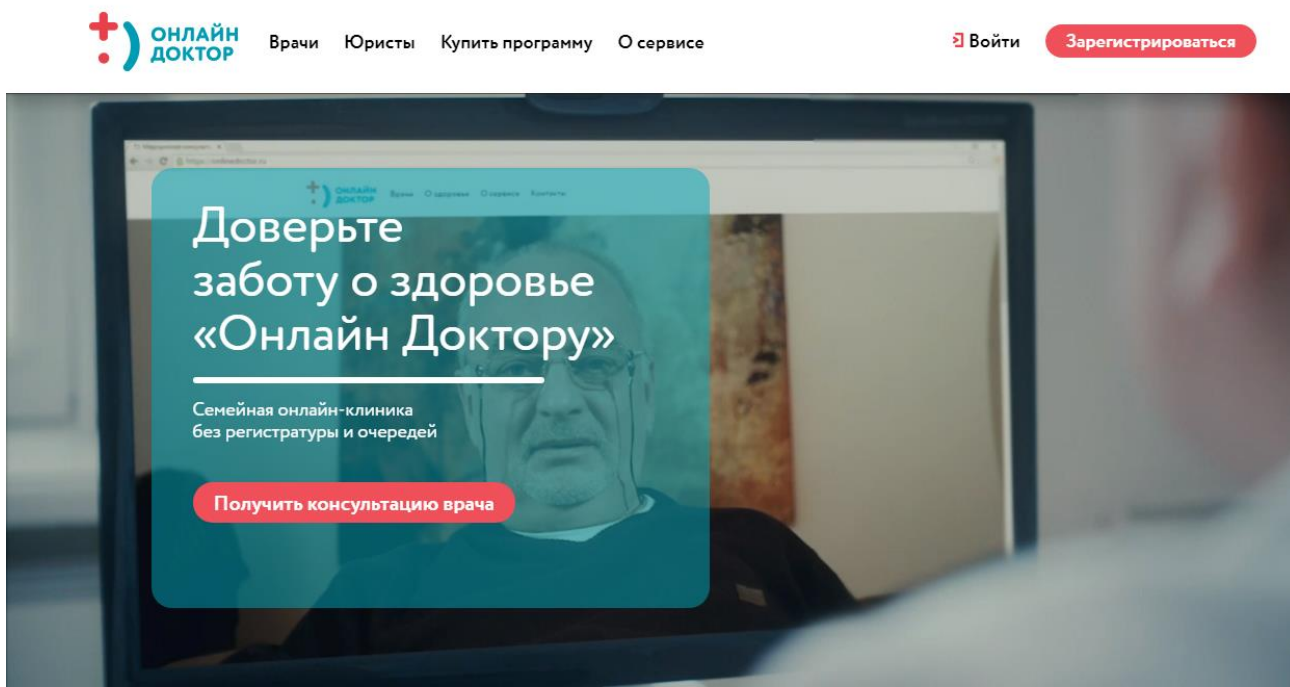
- отсутствие очередей;
- множество врачей;
- свой медицинский центр.

Недостатки:

- нет возможности получить консультацию в выходной день;
- предоставление услуг через ПО “Скайп”.

2.4 Онлайн Доктор

В отличие от “Doc+ Онлайн” и “Доктор рядом” есть возможность получить консультацию в любое время суток. Есть удобное мобильное приложение, история консультаций, напоминания о предстоящей консультации и др. функционал. Однако, стоимость услуг сильно зависит от опыта и специализации врача и сильно отличается от других аналогичных сервисов.



Срочная помощь и плановые приемы

Рисунок 2.4 – Сервис “Онлайн доктор”

Преимущества:

- отсутствие очередей;
- множество врачей;
- многофункциональное приложение;
- оказание услуг круглосуточно.

Недостатки:

- высокие цены.

2.5 Общие требования и процесс получения консультаций

Для получения консультации необходимо мобильное устройство или персональный компьютер, мобильное приложение или веб-браузер, стабильное соединение с интернет, микрофон и, желательно, но не обязательно веб-камера.

Далее необходимо зарегистрироваться в сервисе через мобильное приложение или веб-интерфейс, определиться с врачом и временем консультации забронировав услугу, после чего оплатить.

2.6 Итоги сравнения

Подводя итоги, можно с уверенностью сказать, что “Яндекс Здоровье” является серьезным сервисом, который благодаря своим ресурсам может предоставлять качественные (с технической точки зрения) и недорогие услуги. Другие же сервисы предоставляют качественные мед.услуги, однако цены на эти услуги совсем немного ниже очных медицинских консультаций.

Ниже приведена сравнительная таблица цен сервисов (табл. 2.1). Стоимость варьируется в зависимости от сервиса и часто сильно зависит от квалификации врача и его специализации.

Таблица 2.1- сравнительный анализ цен аналогичных сервисов

Сервис	Стоимость услуг терапевта, рублей	Стоимость услуг специализированного врача, рублей
“Дос+ Онлайн”	499	799
“Доктор рядом”	1200	1200
“Онлайн Доктор”	800-1800*	800-2500*
“Яндекс Здоровье”	Первая консультация - 199, последующие - 499	

3 Функционал приложения

В ходе выполнения выпускной квалификационной работы помимо основного функционала, такого как видеосвязь и регистрация, были реализованы дополнительные функции, например, поиск врачей, проверка соответствия требованиям для оказания услуг видео-консультаций, чат с возможностью прикрепления материала к консультации и т.д. Ниже приведена диаграмма, описывающая основной функционал приложения.

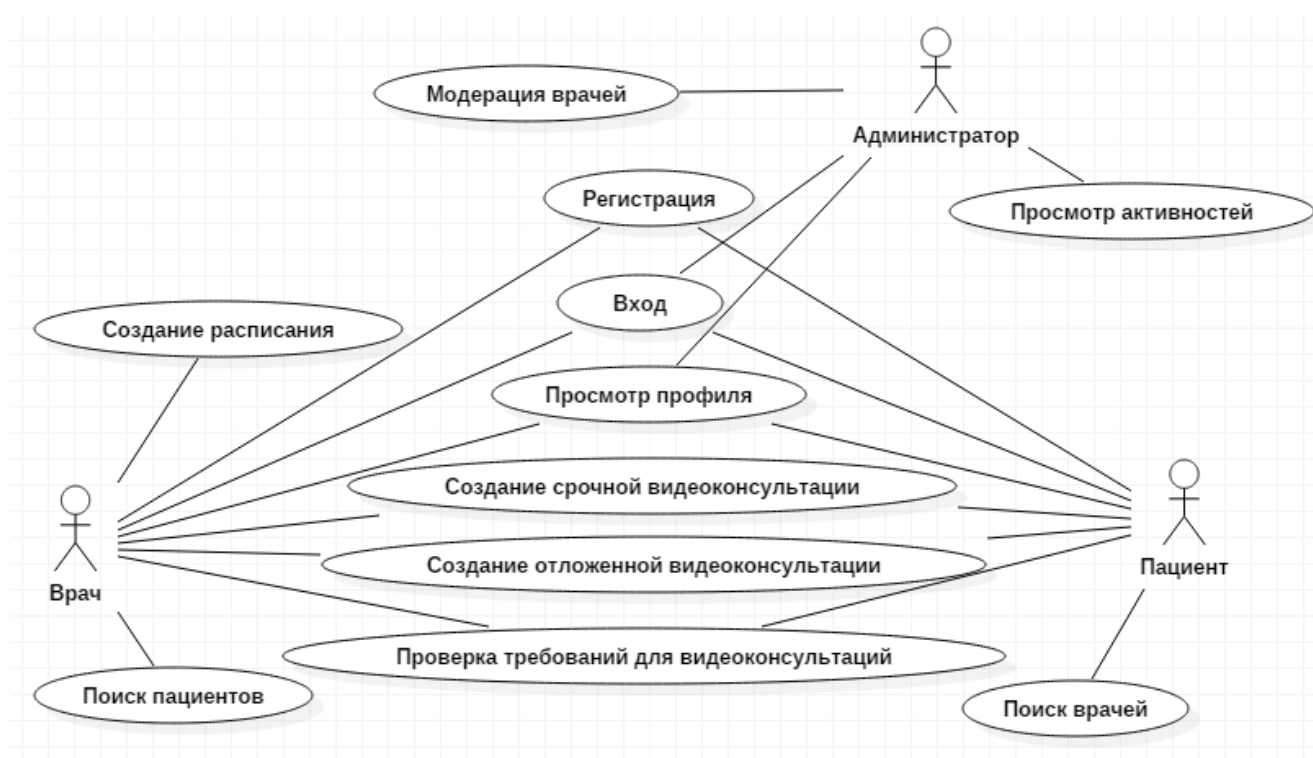


Рисунок 3.1 - use-case диаграмма

3.1 Аутентификация, авторизация, регистрация

Для любого приложения где имеются пользователи необходимо реализовать как минимум аутентификацию, а так как в реализуемом приложении имеются на данный момент три роли (врачи, пациенты, администраторы), то необходима авторизация пользователей.

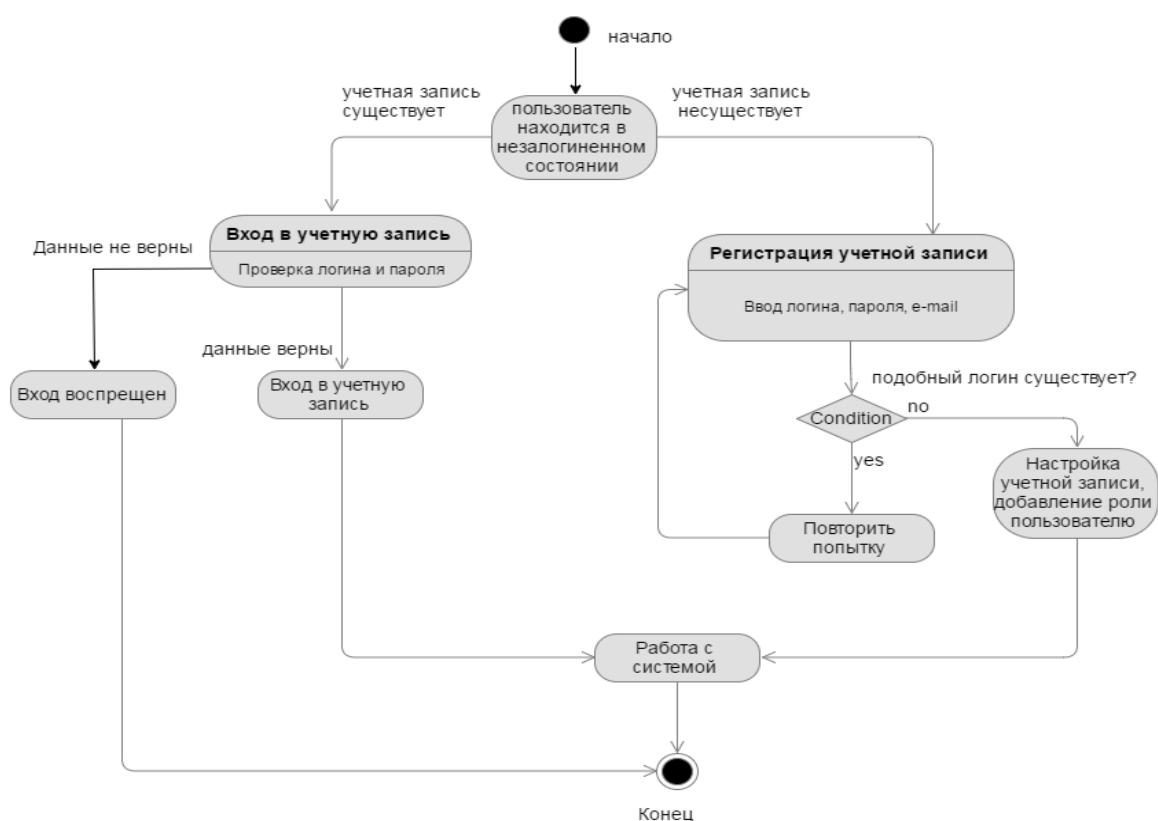


Рисунок 3.2 - диаграмма деятельности для авторизации и регистрации пользователей

Самостоятельный процесс регистрации в приложении реализован только для пациентов. Регистрация врачей является несколько более сложным процессом. Для регистрации врача необходима проверка администратора. Другими словами, врач подаёт заявку на регистрацию и ожидает ее подтверждение администратором. Предполагается что администраторы будут внесены в базу данных напрямую.

Для ясности дадим определения терминам аутентификация и авторизация.

Аутентификация – процедура проверки подлинности. Т.е. процедура установления личности на основе предоставленных данных. Например, логина/пароля.

Авторизация - процесс проверки наличия прав на совершаемое действие на основе аутентификации.

Так как приложение реализуется по *RESTful* архитектуре (см. раздел 4) и имеются как клиент так и сервер, то функционал необходимо реализовать с обеих сторон.

Процесс регистрации пользователей является стандартным: пользователь отправляет данные для регистрации, и в случае успеха сервер отвечает 200 статус-кодом. Данные посылаются на сервер клиентом в теле запроса через *HTTP* протокол методом *POST*.

Вводимые пользователем данные валидируются. Валидация данных – это процесс проверки корректности вводимых данных. Валидация выполняется как до отправки данных на клиенте, так и в момент обработки данных на сервере.

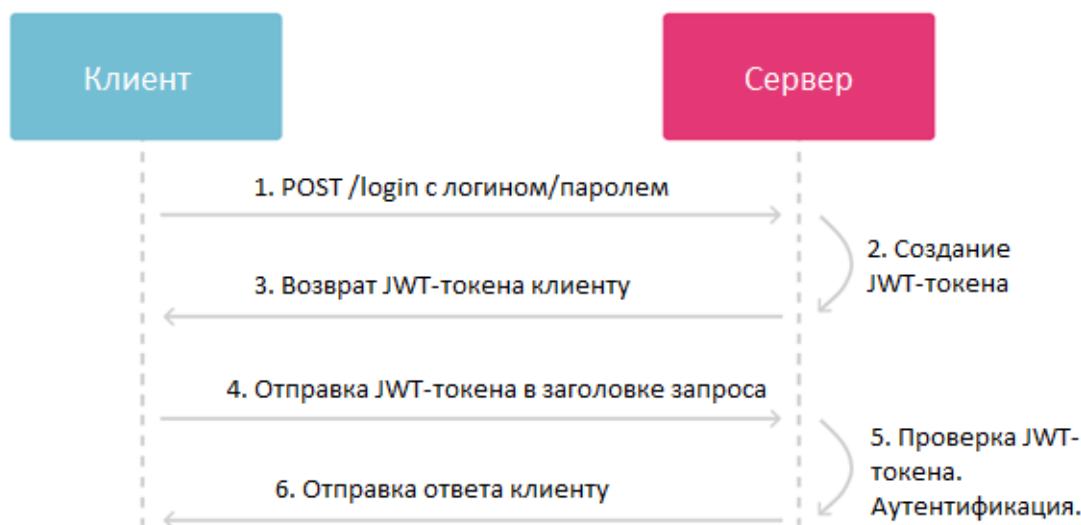


Рисунок 3.3 - схема получения токена и его использование

После успешной регистрации пользователь может воспользоваться своими данными и войти в приложение. Клиент отправляет данные пользователя серверу. Сервер проверяет соответствие логина/пароля пользователя и, в случае успеха, отправляет клиенту токен доступа – *JWT*-токен. На основе этого токена

клиент в дальнейшем может получать доступ к личным данным пользователя на сервере. Ниже приведена схема получения *JWT*-токена и дальнейшее его использование клиентом.

JWT-токен – это сгенерированная на сервере последовательность символов, содержащая в себе метаданные о пользователе, например, можно «защитить» в токен роль пользователя, логин, почтовый ящик и др. данные, что и было сделано. Токен имеет срок жизни, после чего его необходимо обновить. Срок жизни токена конфигурируется на сервере. Далее полученный токен сохраняется на клиенте и при каждом запросе передается серверу для авторизации.

Для хранения токена на клиенте использовалось стандартное браузерное хранилище – *localStorage*.

3.2 Главная страница

Зайдя на сайт, неавторизованный пользователь попадает на главную страницу с поиском врача по специализации и/или имени (рис. 3.4). У пользователя также есть возможность войти в личный кабинет. Практикующий, но не зарегистрированный врач может подать заявление на регистрацию в системе.

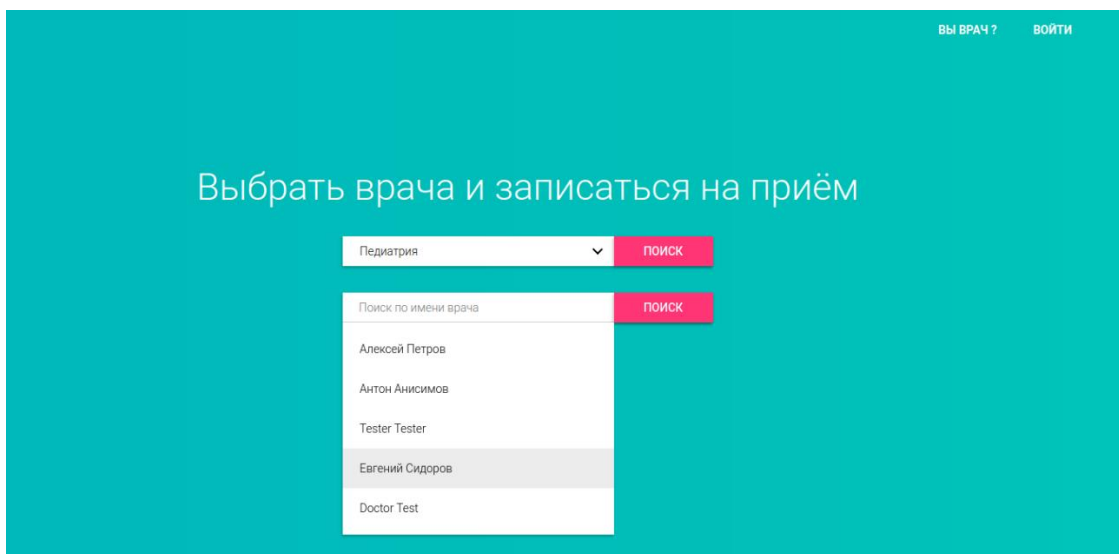


Рисунок 3.4 - главная страница

Интересной особенностью поиска является функция автозаполнения и поиск без перезагрузки страницы. Достигается это с помощью технологии фоновых запросов *XMLHttpRequest*. При изменении поля ввода или после простоя в одну секунду, посылается фоновый запрос на сервер. Поиск происходит по следующим полям: имени, фамилии, имени и фамилии.

Скорость поиска улучшена с помощью индексирования полей имени и фамилии как вместе, так и по отдельности в базе данных.

После выбора врача происходит переход на страницу с результатом поиска (рис. 3.5), где можно выбрать время будущей консультации в соответствии с расписанием нужного врача.

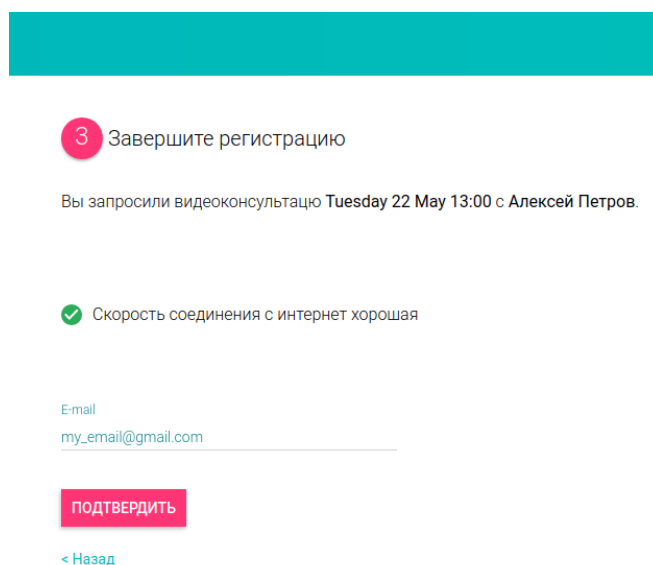
Для отображения врачей был сделан список с возможностью фильтрации по статусу и поиску по имени врача. Добавлена постраничная навигация по списку, если количество врачей в выдаче превышает 10.

The screenshot shows a web interface for finding a doctor. At the top, there is a teal header with links "ВЫ ВРАЧ ?" and "ВОЙТИ". Below the header, a pink step indicator shows "1 Скорректировать поиск". A search bar contains "Алексей Петров". Below the search bar, there are filter options: "Фильтровать по специализации" and "Фильтровать по доступности" (which is checked). It says "1 врач найден". A second pink step indicator shows "2 Выберите врача и назначьте консультацию". Below this, a card for "М Алексей Петров" is displayed. The card includes a profile picture, name, ID "№ RPPS: 5439875", cost "Стоимость : 150,00 Руб.", and duration "Продолжительность консультации : 60 минут". To the right of the card is a calendar view for May 20-26, showing consultation times: 13:00, 14:00, 15:00, 16:00, 17:00, and 18:00. The time 06:30 is highlighted for Thursday, May 24.

Рисунок 3.5 – результаты поиска врача

Из интересных технологий можно отметить применение вебсокетов. Вебсокеты были использованы для отображения статусов врачей - находится ли врач сети или нет. Технология вебсокетов описана в разделе 4.5.

Третьим шагом пациенту необходимо указать почтовый адрес. На почтовый адрес придут дальнейшие инструкции и напоминания о запланированной консультации пациенту.



The screenshot shows a registration completion screen. At the top is a solid teal header bar. Below it, a pink circle with the number '3' is followed by the text 'Завершите регистрацию'. Underneath, it says 'Вы запросили видеоконсультацию Tuesday 22 May 13:00 с Алексей Петров.' Below this is a green checkmark icon and the text 'Скорость соединения с интернет хорошая'. Further down, there is an 'E-mail' label and a text input field containing 'my_email@gmail.com'. At the bottom, there is a pink button with the text 'ПОДТВЕРДИТЬ' and a blue link with a left arrow and the text '< Назад'.

Рисунок 3.6 – 3 шаг - завершение регистрации

3.3 Функционал пользователей

Пациент после входа в приложение, попадает на страницу с историей консультаций. Здесь определены два блока - текущие и прошедшие консультации. У прошедших консультаций отображается полное имя доктора, дата, время и стоимость консультации. На страницу попадают первые 10 консультаций, далее появляется постраничная навигация. У пациента есть возможность подключиться к текущей консультации, нажав кнопку “Начать”, или завершить ее, нажав кнопку “Завершить”. Также пациент может запланировать будущую консультацию. Нажав на соответствующую кнопку, пациент перейдет на страницу поиска врача (рис 3.5).

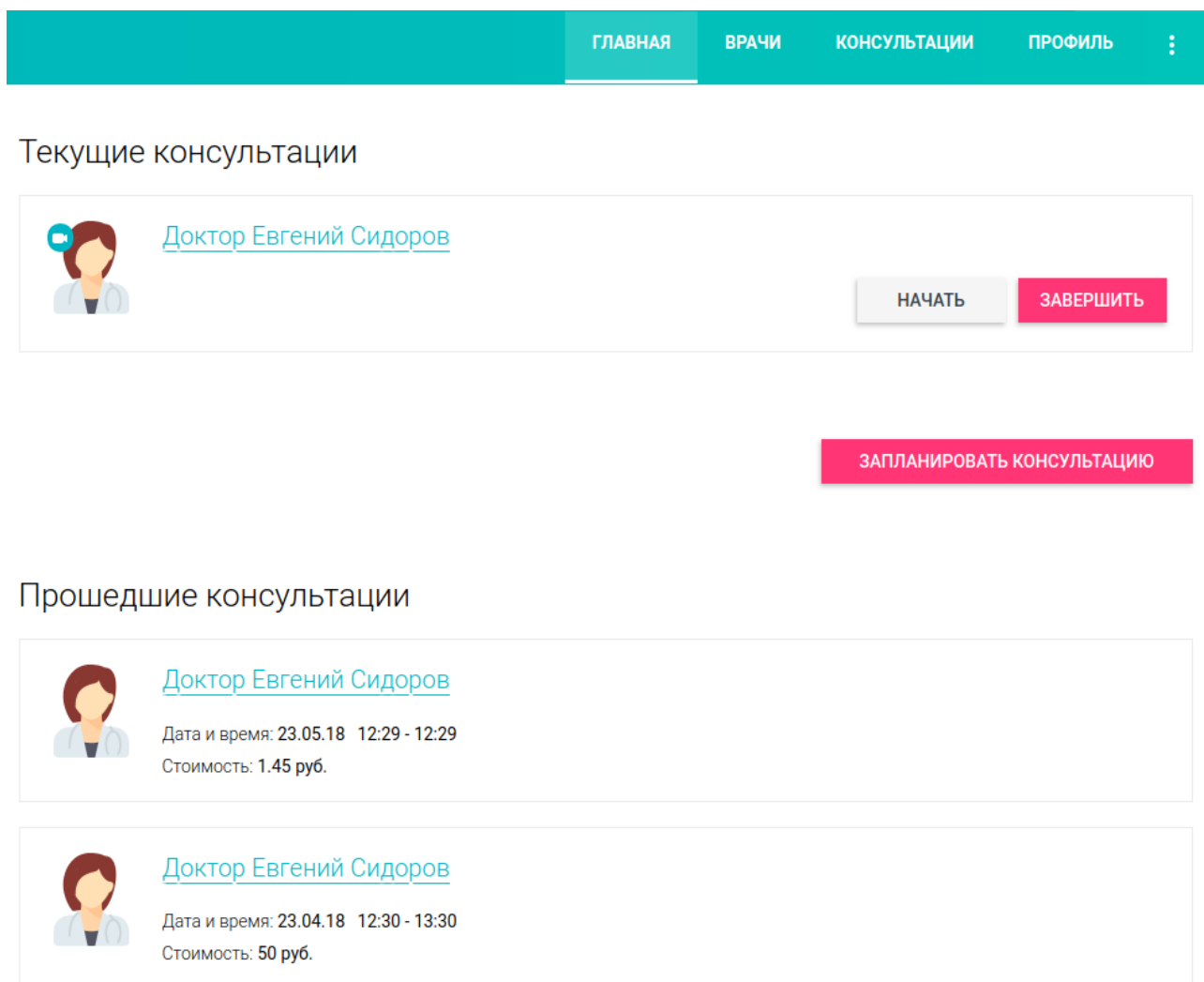


Рисунок 3.7 - главная страница пациента

Главная страница врача (рис. 3.8) содержит очередь консультаций в хронологическом порядке, разбитую по дням с возможностью переключения. Консультации имеют статусы: “подтверждена”, “отклонена”, “ожидает подтверждения”. Также указывается полное имя пациента и время консультации. На текущей консультации - т.е. на консультации проходящей в текущий момент, отображается иконка для перехода в окно видео-консультации.

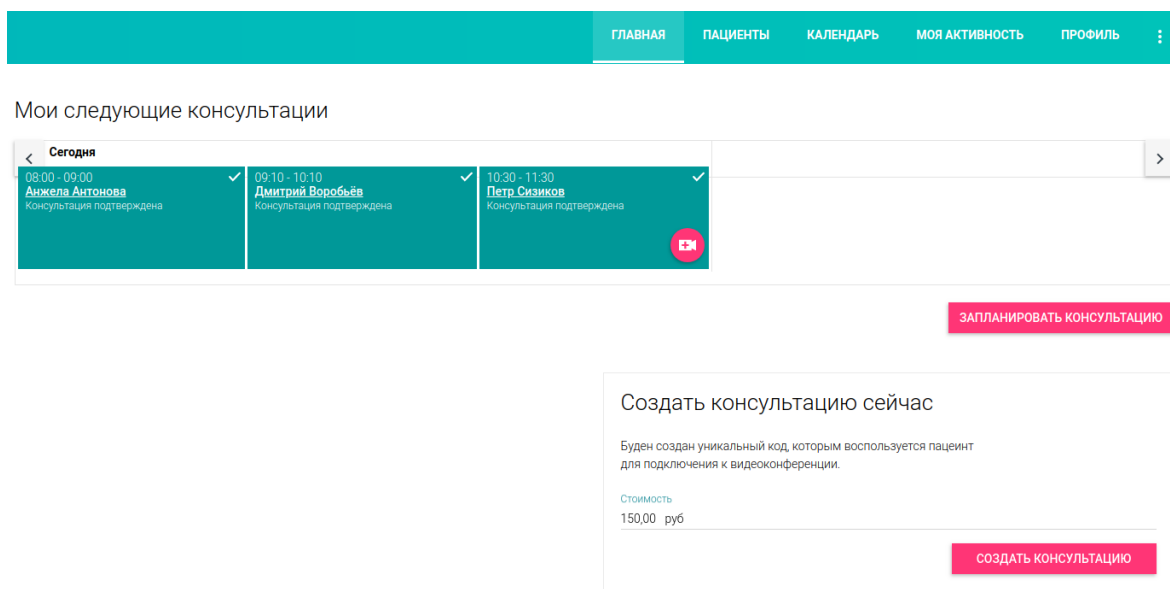


Рисунок 3.8 - главная страница врача

У врача, как и у пациента, есть возможность инициализировать будущую консультацию, нажав на кнопку “Запланировать консультацию”. Врач может запланировать консультацию только с уже зарегистрированным пациентом в системе, выбрав его в выпадающем поиске по имени и назначив время консультации (рис. 3.9).

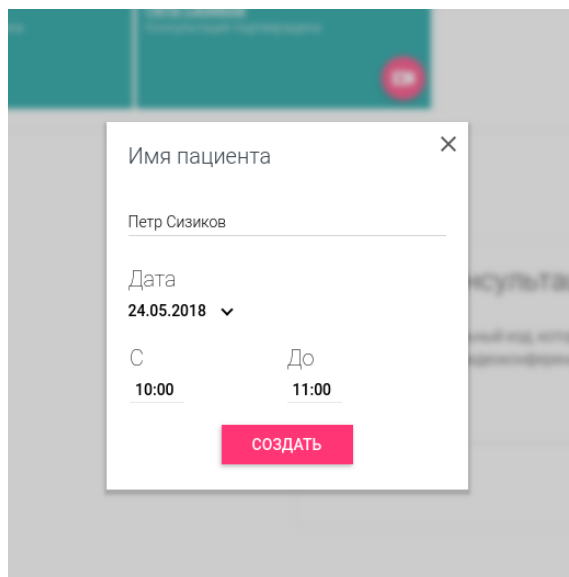


Рисунок 3.9 – планирование консультации врачом

Для того, чтобы пригласить незарегистрированных пациентов предусмотрена функция приглашения и консультации по уникальному коду. Врач, предварительно указав стоимость будущей консультации, нажимает на кнопку “Создать консультацию” и получает окно с кодом (рис. 3.10).

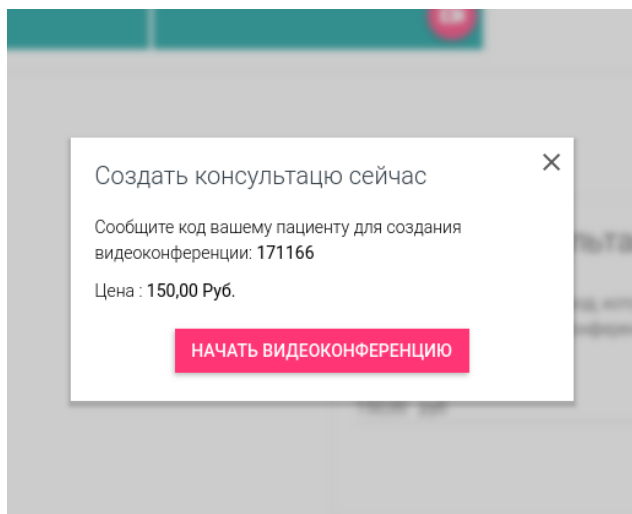


Рисунок 3.10 - приглашение на моментальную консультацию пациента по коду

Далее врач должен сообщить код консультации своему пациенту, например по телефону, и пациент сможет присоединиться к созданной консультации на странице входа в приложение (рис. 3.11 и рис.3.12).

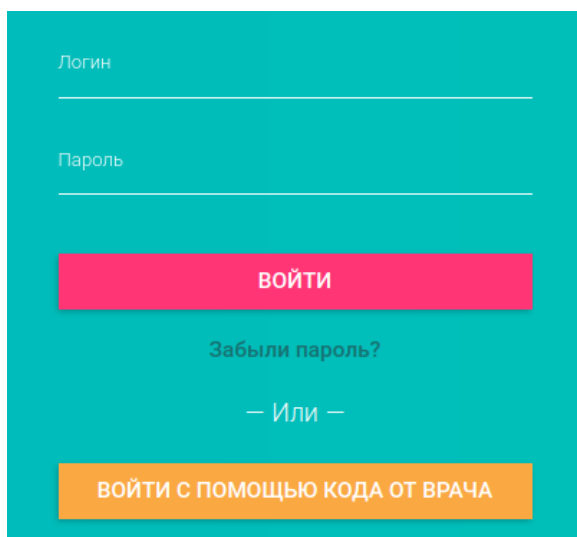


Рисунок 3.11 – вход в приложение. Обычный и по пригласительному коду.

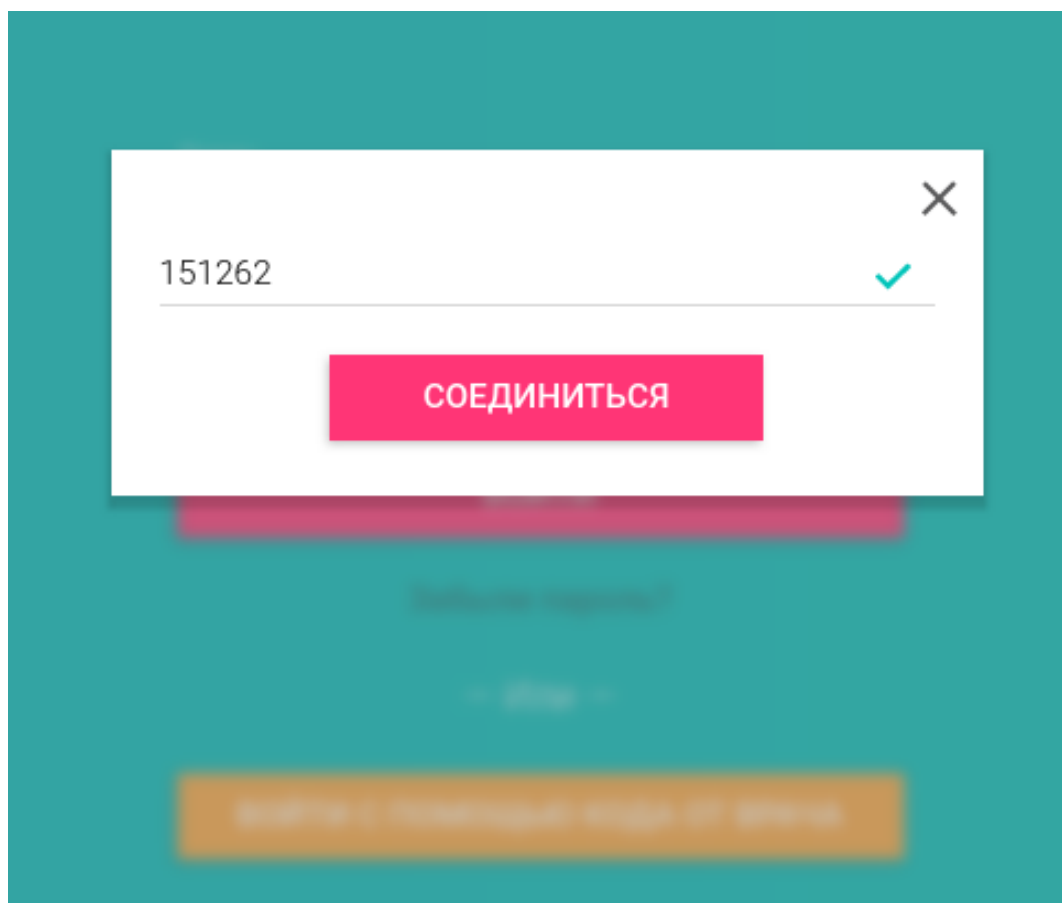


Рисунок 3.12 – ввод пригласительного кода

Для описания процесса создания моментальной видео-консультации ниже приведена диаграмма последовательности:

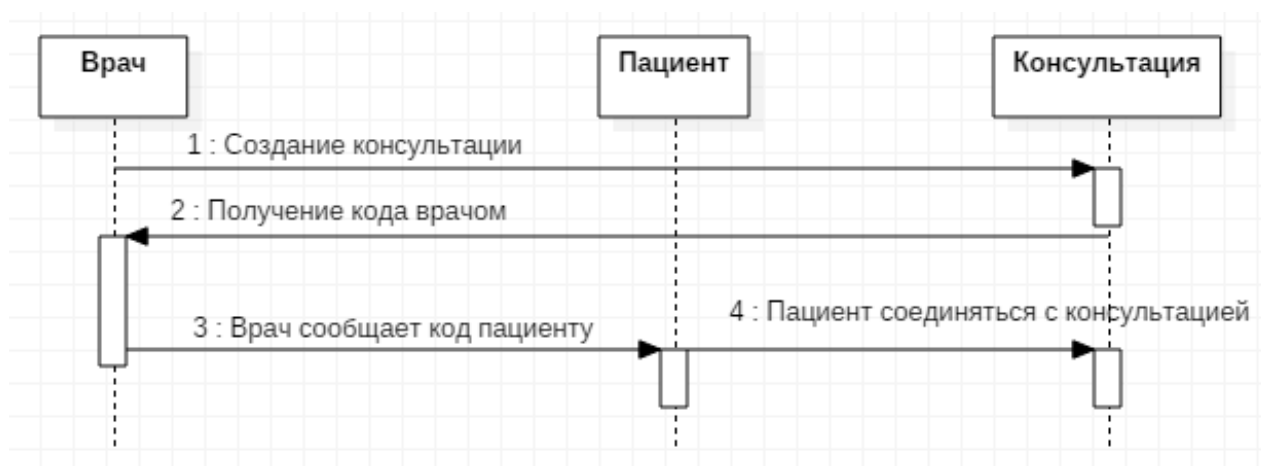


Рисунок 3.13 – процесс моментальной консультации

3.4 Окно видео-консультаций

После создания запланированной или моментальной консультации пациент и врач могут начать видео-консультацию.

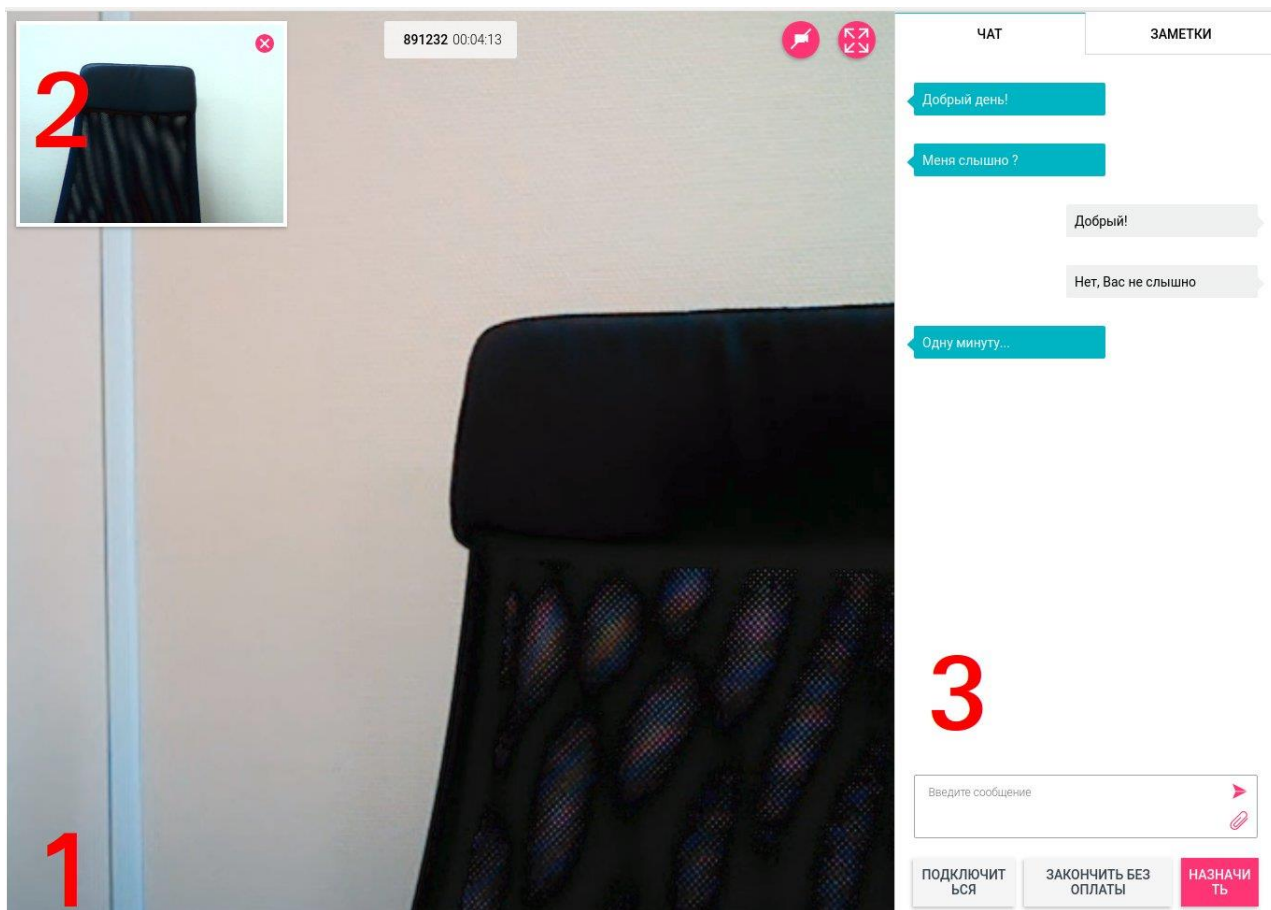


Рисунок 3.14 - окно видео-консультаций

Окно консультаций состоит из видео-окна пациента (1) и врача (2) (либо наоборот в зависимости от роли), окна чата и окна заметок (3). Пациент и врач могут обмениваться материалами, например, результатами анализов или рекомендациями пациенту от врача. История сообщений и прикрепленные файлы сохраняются в базе данных и остаются доступными после консультации. Имеются кнопки управления: развернуть видео на весь экран, отключить видео. Время консультации фиксируется и отображается на экране (рис. 3.14).

3.5 Личный профиль

У всех пользователей есть свой личный кабинет с профилем, но наибольшего внимания заслуживает профиль врача (рис. 3.15). Он включает в себя функционал профилей пациента и администратора, а также свой, дополнительный, необходимый только врачу.

В профиле врача можно изменять личные данные такие как имя, фамилия, дата рождения, телефон и почта. Есть возможность выбрать специализацию, по которой пациент будет иметь возможность находить нужного врача. Врач может написать текст, описывающий его специализацию или услугу, который будет отображаться при поисковой выдаче. Имеется функционал изменения почтового ящика, смены пароля, установки личной фотографии и удаления аккаунта.

Идентификатор
12345673

Имя *
Алексей

Фамилия *
Петров

Дата рождения *
22.03.1962 ▼

Телефон *
🇷🇺 ▼ +7 954 786 57 34

E-mail
doctor1@doctor.com

ИЗМЕНИТЬ ПОЧТУ

ИЗМЕНИТЬ ПАРОЛЬ

Специализация *

☐ Наркомания

☐ гериатрической психиатрии

☐ Питание

☒ Педиатрия

☒ Общая психиатрия

☐ Табакология

☐ Другое

О себе
Презентационный текст

СОХРАНИТЬ ИЗМЕНЕНИЯ

УДАЛИТЬ ПРОФИЛЬ

СМЕНИТЬ ФОТОГРАФИЮ

Рисунок 3.15 - личный профиль врача

3.6 Тест на соответствие требованиям видео-консультаций

Для оказания качественных услуг видео-консультаций необходимо выполнить несколько требований:

- наличие веб-камеры и микрофона;
- разрешение доступа к веб-камере и микрофону;
- поддерживаемая и включенная технология *WebRTC* в браузере;
- стабильное и достаточно быстрое соединение с интернет.

В случае несоответствия одному из требований пользователь будет об этом оповещен (рис. 3.16).

Проверка камеры

ПРОВЕРИТЬ КАМЕРУ

Не удалось подключиться к вашей камере или WebRTC

- Убедитесь что Ваш браузер совместим
- Убедитесь что к компьютеру подключена камера
- Убедитесь что Вы разрешили доступ к камере

Проверить скорость интернета

ПРОВЕРИТЬ СКОРОСТЬ СОЕДИНЕНИЯ

✓ Ваше интернет-соединение удовлетворяет условиям для видеоконсультации.

Рисунок 3.16 - тестирование требований для оказания видео-консультаций

Итоги

В данной главе был описан функционал приложения для оказания услуг удаленных видео-консультаций – т.е. инструмент для телемедицинских услуг. Врач для своего удобства может планировать расписание приёмов пациентов, видеть будущие, запланированные консультации и самостоятельно их инициировать, приглашая пациентов. Пациент, как и врач, видит историю консультаций, заметки от врача, а также имеет доступ к поиску врачей.

4 Архитектура и реализация функционала приложения

От заложенной архитектуры и выбранных инструментов на начальном этапе для реализации информационной системы зависит множество вещей. Правильно спроектированная архитектура в дальнейшем облегчает жизнь разработчикам и владельцам продукта при поддержке и доработке нового функционала. Основные критерии, предъявляемые к качеству архитектуры информационной системы можно выделить следующие: [9]

- *эффективность* – система должна решать поставленные перед ней задачи в различных условиях;
- *гибкость* - существующий функционал должен иметь возможность быстро изменяться под новые требования;
- *расширяемость* – система должна иметь возможность добавлять новые функции в приложение, не нарушая функционирование старых;
- *повторное использование* – принцип *DRY (don't repeat yourself)* – не повторяйся. Повторное использование компонентов системы;
- *тестируемость* – функционал системы должен быть атомарен на столько, на сколько это является возможным. Атомарные фрагменты легче тестируются;
- *связь компонентов* – компоненты системы должны быть связаны “слабой” связью (*tight coupling*).

4.1 Инструменты и технологии реализации

Серверная сторона проекта реализована на *NodeJS* с использованием фреймворка *Express*. В качестве базы данных использовалась *MongoDB* – нереляционная база данных документного типа. Для платформонезависимой разработки выбран *Docker*.

В качестве языка для разработки клиентского приложения был выбран набирающий популярность скриптовый язык с динамической типизацией *javascript*. Для построения графических элементов выбрана библиотека от *facebook react* в связке с менеджером состояний приложения *redux* и библиотекой создания сторонних эффектов *saga*. В проекте был написан по стандарту *ES6* (*ECMAScript 6*), который на данный момент поддерживается очень не многими браузерами. Для решения этой проблемы применялся промежуточный транпилятор *Babel*, который транпилирует код из *ES5* в *ES6*. Для сборки клиентской части проекта выбран инструмент *Webpack*.

В проекте были использованы два типа архитектуры:

1. Клиент-серверная.
2. Архитектура клиент-клиент (*Peer-to-Perr, P2P*).

4.1.1 Клиент-серверная архитектура

Архитектура клиент-серверных приложений является самой распространенной в сети интернет. Сама всемирная паутина, изобретенная физиком из *CERN* Тимом Бернерсом-Ли, была разработана как клиент-серверная система.

Клиент-сервер – это такая архитектура, в которой задачи распределены между поставщиками услуг, серверами, и заказчиками услуг, клиентами. Эта архитектура изначально предусматривала взаимодействие по схеме запрос-ответ. Клиент выполняет активную функцию инициирования запросов, а сервер отвечает на них по мере их поступления.

Клиент-серверное взаимодействие будет происходить в формате *JSON* поверх *HTTP* протокола. Для этих целей была реализована *REST*-архитектура. Такой подход позволяет отделить бизнес-логику приложения от представления.

В рамках проекта планируется реализовать только веб-клиент. Однако, такой подход даёт возможность в дальнейшем без проблем подключить, например, клиенты мобильных приложений.

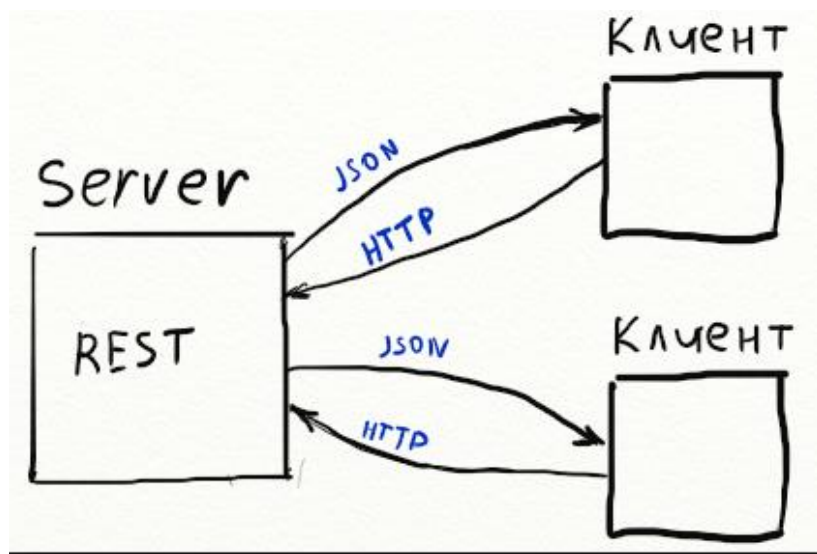


Рисунок 4.1 – RESTFul архитектура

4.1.2 Пиринговая архитектура

Пиринговая архитектура или ещё её называют архитектура точка-точка (*Peer-2-Peer, p2p*) – является альтернативной клиент-серверной. Однако в проекте будут использоваться оба подхода.

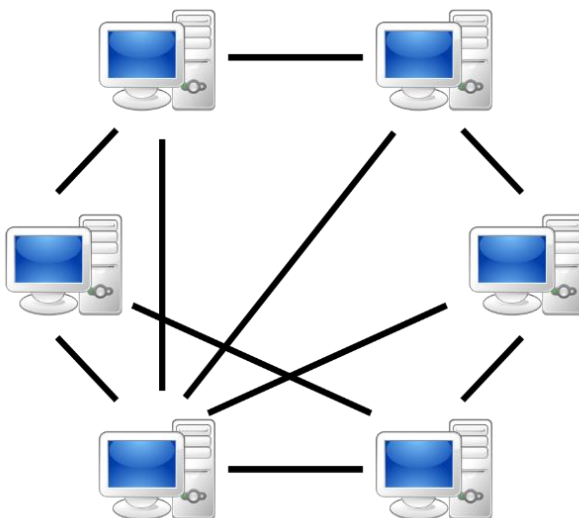


Рисунок 4.2 – пиринговая архитектура точка-точка (p2p)

Суть архитектуры заключается в том, что точки (узды) является одновременно как сервером, так и клиентом. Такая архитектура будет использоваться во время передачи медиа-потокa между клиентами в обход сервера.

4.2 Сервер

В качестве серверного языка программирования был использован *NodeJS*. *NodeJS* - это платформа, транслирующая код написанный на *JavaScript* в низкоуровневый машинный код. Для трансляции используется “движок” *V8* написанный в *Google*. *NodeJs* в качестве серверного языка используют такие крупные компании как *Walmart*, *e-Bay*, *PayPal*, *Google*, *Uber*, что является признаком состоятельности технологии.

Скорость разработки является одним из факторов определяющим стоимость продукта. В связи с этим для ускорения процесса разработки информационных систем по возможности применяют написанные ранее библиотеки и фреймворки.

В проекте использовался фреймворк *Express* - наиболее популярный фреймворк для платформы *NodeJS*. Фреймворк предоставляет инструменты для обработки *http*-запросов, организации веб-сервера, маршрутизации, аутентификации, авторизации и т.д.

Express не является *MVC (Model-View-Controller)* фреймворком, но для разработки использовалась подобная схема работы за исключением одного - так как серверная часть не имеет представления, а выдаёт данные в формате *JSON*, то компонента представления (*View*) не нужна. Остальные составляющие *MVC* – модель и контроллеры используются в проекте.

4.2.1 REST-Архитектура

REST — (*Representation State Transfer*) — передача состояния представления. Является архитектурным стилем построенным на основе *HTTP* протокола. Был предложен в 2000 году одним из разработчиков протокола *HTTP*. Для управления и получения информации используются *HTTP* методы: для создания записи — *POST*, для получения — *GET*, для редактирования *PUT* или *PATCH*, для удаления *DELETE*.

Например, создание пользователя */user* — *POST*, получение пользователя */user/9* — *GET*, редактирование пользователя */user/9* — *PUT|PATCH*, удаление пользователя */user/9* — *DELETE*.

Как можно заметить, в зависимости от метода запроса один и тот же адрес реагирует по-разному – это один из основных принципов *REST*-архитектуры.

Также хорошей практикой является отдавать клиенту информативный статус код в ответе. Например, при удачном создании записи 201, при ошибке авторизации 401 и т.д. Статус-коды описаны в стандарте *HTTP* протокола [10].

4.2.2 Структура проекта

В начале проекта необходимо качественно определить структуру проекта и разделить зоны ответственности директорий. На старте проекта была проведена такая работа.

Таблица 4.1 – описание корневой директории серверной части приложения

Название директории	Описание
<i>app</i>	Доменная директория
<i>bootstrap</i>	Скрипты для запуска сервисов
<i>config</i>	Конфигурация
<i>routes</i>	Конфигурация <i>url</i> - запросов и соответствующих контроллеров
<i>app.js</i>	Распределитель запросов

В корне проекта находятся директории конфигураций, скриптов для запуска, маршрутов (роутов) и входной файл приложения *app.js* - так называемый фронт-контроллер. Фронт-контроллер выполняет роль получения запроса и старта дальнейшей его обработки, путем инициализации приложения и сопоставления маршрута запроса и соответствующего контроллера. Другими словами, фронт-контроллер является дирижёром *http*-запросов.

Наибольший интерес представляет директория *app*, которая содержит бизнес-логику приложения. Ниже в таблице приведено краткое описание директорий *app*.

Таблица 4.2 – содержимое директории app серверной части приложения

Директория	Описание
<i>controllers</i>	Контроллеры
<i>exceptions</i>	Исключения
<i>formatters</i>	Шаблоны ответов
<i>middlewares</i>	Промежуточные слои
<i>models</i>	Модели
<i>repositories</i>	Репозитории
<i>services</i>	Сервисы
<i>utils</i>	Утилиты
<i>validators</i>	Валидаторы

Контроллеры - выполняют роль связующего звена между запросом и ответом. Приложение получает запрос и в соответствии с маршрутами (роутами) определяет нужный контроллер. Контроллер в соответствии с параметрами запроса выполняет какие-либо действия, например, сохранение или изменение в базе данных конкретной модели, а после высылает соответствующий ответ обратно клиенту.

Модели – это объекты, описывающие предметы реального мира. Содержат описание свойств объектов предметной области и методы работы с ними. На основе моделей строится схема базы данных и дальнейшее взаимодействие с ней.

Репозитории - хранилище методов для запросов в базу данных. При необходимости сделать запрос в базу данных вызываются соответствующие методы репозитория из контроллера. Например, извлечь пользователя по имени. Контроллер, отвечающий за пользователя, вызывает метод репозитория отвечающий за получение пользователя по имени. Метод репозитория, используя модель пользователя, параметры запроса и слой абстракции строит поисковый запрос к базе данных и исполняет его. Репозиторий является одним из общепринятых паттернов проектирования [1].

Промежуточные слои обработки (middlewares) - являются фильтрами обработки запросов. Здесь расположены такие промежуточные слои как авторизация пользователей. Каждый запрос проходит проверку на доступность ресурса пользователю.

Сервисы - содержат какую-либо бизнес-логику приложения. Например, алгоритм проверки наличия прав пользователя на ресурс. В отличие от промежуточных слоёв обработки, сервисы только предоставляет методы.

Валидаторы - отвечают за проверку правильности сохраняемых данных моделей.

Исключения - хранилище исключений приложения.

Утилиты - маленькие, вспомогательные куски кода. Например, для формирования дат.

4.2.3 Объектное представление базы данных

В проекте используется *ORM* – *mongoose* для работы с не реляционной базой данных *mongoDB*. *ORM* – *object relation mapping* – объектно-реляционное отображение. Это слой абстракции для взаимодействия с базой данных на объектном уровне. *ORM* “соединяет” объектное представление с базой данных, сопоставляет объектные модели таблицам базы данных для упрощения взаимодействия.

Например, в проекте имеется сущность пользователя. Сущность описывается как объект с набором свойств. Далее вызывается команда генерации схемы в базе данных на основе объекта. Каждое свойство объекта соответствует свойству таблицы базы данных. Для запроса к базе данных вызывается объект *orm* с указанием сущности и варианта запроса к базе данных.

4.2.4 Схема запрос-ответ

Часто *Express* фреймворк называют *Request-Response* (*запрос-ответ*) фреймворком, это связано с его принципом работы: получение *http*-запроса — его обработка — некоторые действия — *http*-ответ.

Нужно также отметить, что из-за принципов работы *http* протокола - сервер после отправки ответа клиент “умирает” и возможности работы в “ран-тайм” не имеется. На диаграмме ниже (рис. 4.3) описан частный случай обработки *http*-запроса и ответа:

1. Клиент посылает *http*-запрос.
2. Фронт-контроллер принимает запрос, инициализирует приложение, и в соответствии с запросом и таблицей маршрутизации передает управление контроллеру.
3. Контроллер обращается к репозиторию для извлечения записи некоторой сущности.

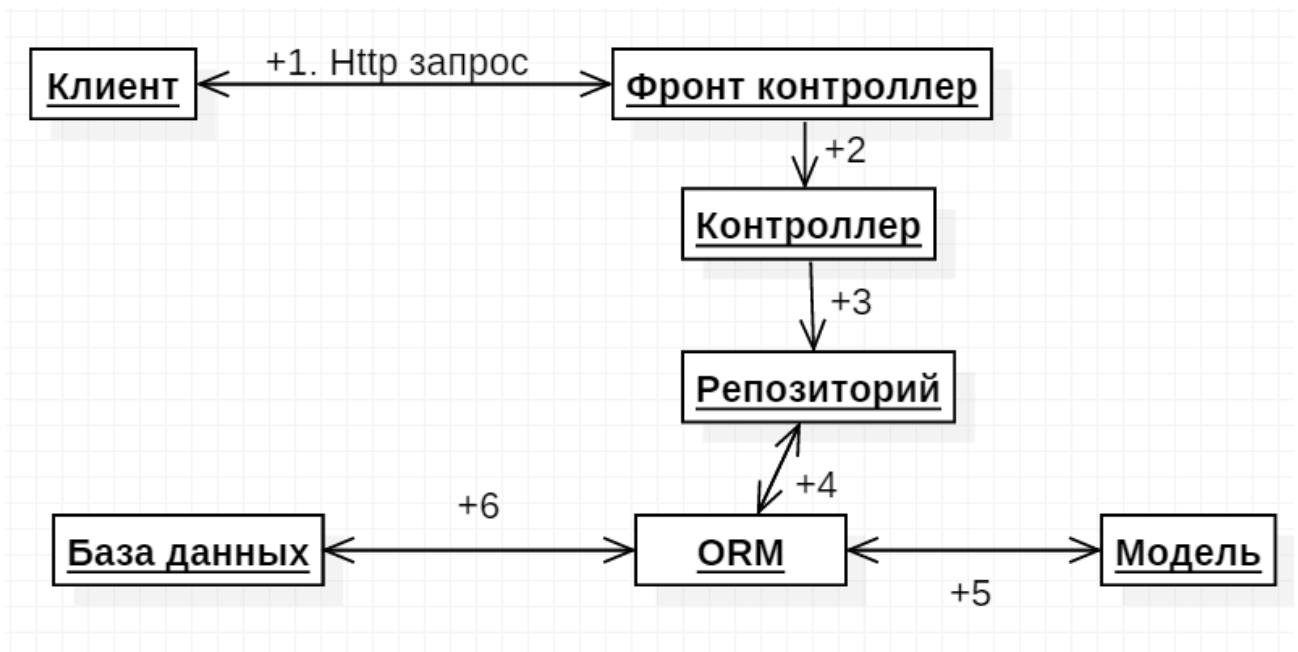


Рисунок 4.3 – схема запроса-ответа сервера

4. Репозиторий обращается к системе *ORM*.
5. *ORM* обращается к соответствующей запросу объектной модели.
6. *ORM* делает запрос на основе объектной модели к базе данных.
7. База данных возвращает результат.
8. *ORM* преобразует результат в объектный вид, т.е. в модель.
9. Дальше ответ отдается вверх по цепочке до контроллера, где контроллер может модифицировать результат и отдать его клиенту.

4.2.5 Диаграмма классов

Данный вид диаграммы предоставляет возможность наглядно увидеть сущности предметной области, атрибуты сущностей и их отношения между собой.

User – пользователь, *role* – роли пользователей, *token* – токены доступа, *speciality* – специализация врача, *schedule* – расписание врача, *consultation* –

консультация, *consultation_note* – заметки и чат в консультации, *file* – файлы прикрепленные к консультации.

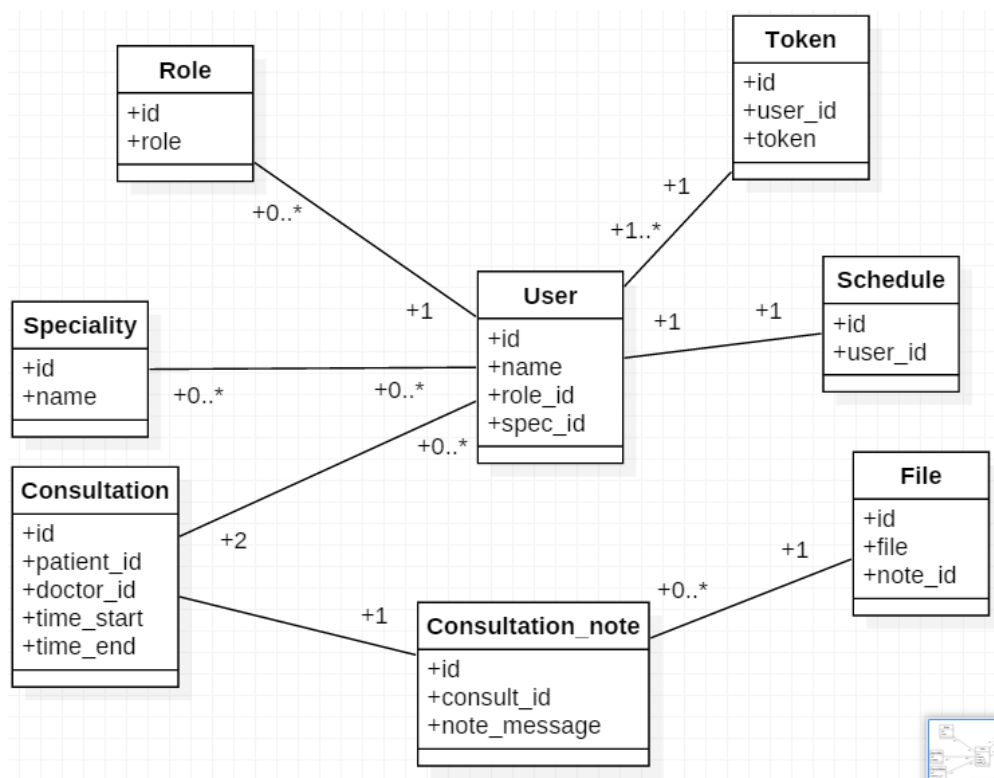


Рисунок 4.4 – диаграмма классов предметной области

4.2.6 Виртуализация окружения разработки

Проектирование и разработка информационной системы это один из этапов жизненного цикла информационной системы. Далее в ЖЦ ИС следуют этапы тестирования, поддержки и развертывания. В разработке ИС также участвуют множество людей: программисты, тестировщики, инженеры автоматизации, системные администраторы и т.д. Возникает острая проблема — ИС может неадекватно и не предсказуемо вести себя в разных окружениях: разные версии операционных систем, библиотек, зависимостей и т.д.

Для решения этой проблемы имеется современный инструмент — *Docker*. *Docker* — это ПО для виртуализации окружения на уровне операционной системы. *Docker* создаёт изолированный контейнер, который практически не

влияет на основное окружение. *Docker* позволяет создать образ текущей системы или её части и запустить его в контейнере виртуализации. Инструмент имеет обширное сообщество и репозиторий образов. В репозитории имеются множество конфигураций различных систем, которые можно использовать в новых проектах как основу.

Из этого можно подчеркнуть следующие плюсы:

1. Быстрое развёртывание приложений.
2. Быстродействие работы за счёт виртуализации на уровне ОС.
3. Платформонезависимость.

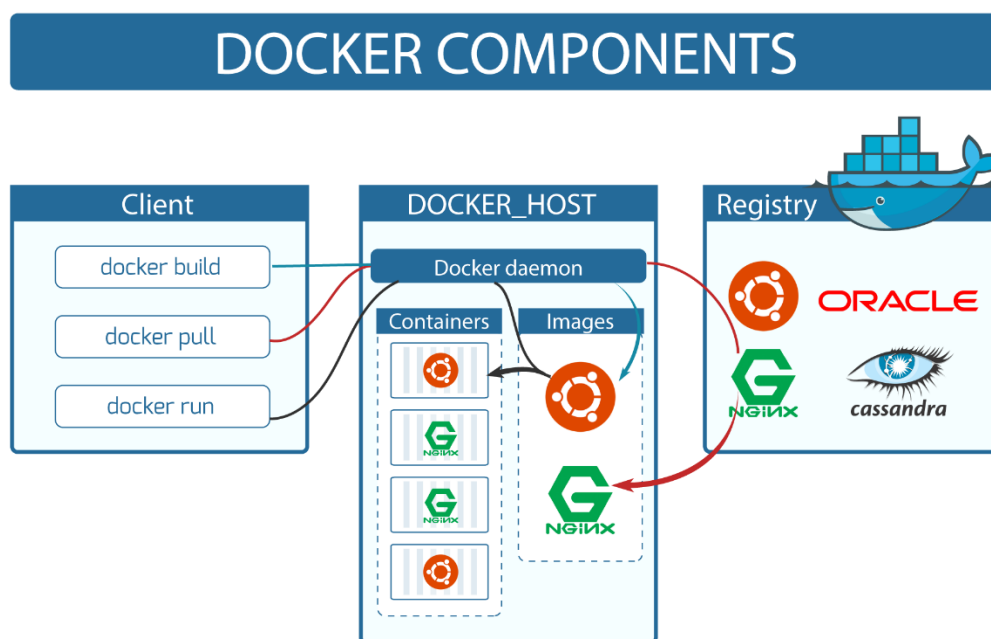


Рисунок 4.5 – архитектура Docker

Клиент коммуницирует с контейнерами как с отдельными хостами запущенными на различных портах на локальной машине. Контейнеры собираются из образов. Образы можно создать самостоятельно либо взять из общедоступного репозитория.

4.2.7 Итог разработки сервера

REST подход на сегодняшний день используется во многих случаях. Такой подход несколько сложнее в реализации на начальном этапе, однако в последствии при необходимости добавить новый клиент, будь то мобильное приложение или веб-приложение, необходимость разрабатывать бизнес-логику с нуля отпадает. Всё необходимое уже имеется и нужно лишь интерпретировать полученные данные на клиенте.

4.3 Разработка клиентской части

На данный момент разработка графических интерфейсов представляет большой интерес. Если раньше было достаточно простых визуальных элементов в браузере с простой логикой взаимодействия с пользователем, то сейчас приложения содержат элементы со сложным поведением. Раньше пользователь устанавливал приложения на компьютер или телефон как отдельное ПО, сейчас же большинство приложений работает в браузере: компания *Google* предоставляет сервис *google docs*, Яндекс предоставляет сервис Яндекс.Музыка, даже Skype предлагает веб-версию своего приложения.

В качестве языка для разработки клиентского приложения был выбран набирающий популярность скриптовый язык с динамической типизацией *JavaScript*. Других аналогов, для создания пользовательских приложений в веб-браузере на данный момент нет. Плюс ко всему, *JS* настолько универсален, что позволяет создавать в том числе и серверные части приложений на платформе *NodeJS*, а также, мобильные приложения путём компиляции написанного кода на нативный язык (*Java* для *android* и *Objective-C* для *IOS*) что обуславливает популярность языка в последнее время. Для построения графических элементов выбрана библиотека от *facebook react* в связке с менеджером состояний приложения *redux*.

4.3.1 *Flux*-паттерн

Клиентское приложение было построено по паттерну *Flux*, а точнее его конкретной реализации в виде библиотеки от *facebook redux*.

Проблема

Предположим следующую схему взаимодействия компонентов приложения (рис.4.6), которая имеет несколько моделей и представлений.

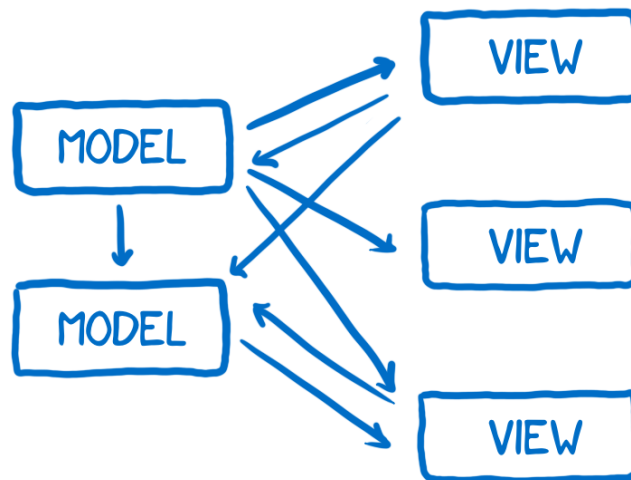


Рисунок 4.6 - путаница во взаимодействии компонентов

Представления в свою очередь могут быть зависимы от нескольких моделей, и модели также могут взаимодействовать между собой и изменять друг-друга. Модели отправляют данные в компоненты представления после того как что-то измениться. Появляются некоторые трудности в отладке приложения если что-то пошло не так. Трудности в отслеживании потока данных добавляет еще и то, что изменения могут быть асинхронными. Одно изменение может вызвать целый каскад других.

Для решения проблемы в компании *Facebook* придумали и начали использовать другой тип архитектуры - *Flux*. Во *Flux* поток данных в приложении является максимально предсказуемым и очевидным. Данные движутся только в одном направлении. На рис.4.7 представлена схема движения потока данных.

Рассмотрим конкретную реализацию паттерна *Flux* - библиотеку *Redux*. В данной системе фигурируют 4 основных элемента - *Action*, *Reducer*, *Store* и *View*.

Action – далее экшн, отвечает за создание действий. В действии содержится информация о том, что произошло в приложении, а также сам тип действия - т.е. какое действие произошло. Обычно типы действий в приложении определяются константами в одном месте для дальнейшего переиспользования.

Reducer – далее редюсер. Экшн содержит информацию о событии, но при этом не знает что делать. Что делать с этой информацией знает редюсер. Редюсер в соответствии с типом вызванного экшена меняет состояние хранилища, о котором речь пойдет дальше.

Store – далее хранилище. Единое хранилище общего состояния приложения. Хранилище выполняет две основные роли - хранит состояние и при изменении состояния информирует об этом компоненты представления (*View*).

View - представление приложения. Обычно сгенерированная *HTML* страница на основе *JS*. Подписывается на состояние хранилища и при изменении получает новые данные. После получения данных от хранилища обновляет соответствующие элементы представления.

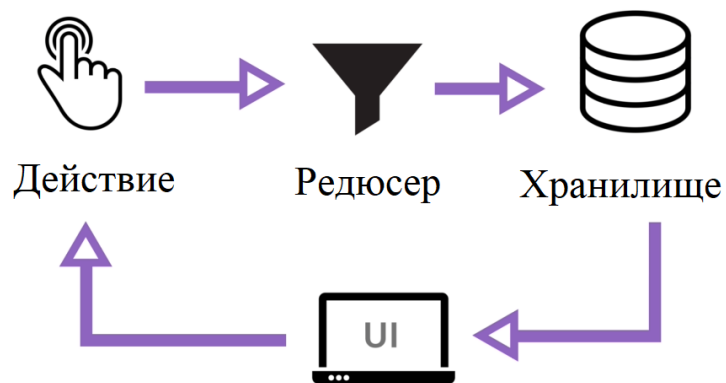


Рисунок 4.7 – схема взаимодействия компонентов

Другими словами схему можно описать так: пользователь кликает на элемент пользовательского интерфейса (*UI*) и вызывает действие в приложении.

Действие содержит информацию о том что произошло и тип события. Редюсер в соответствии с типом события изменяет текущее состояние приложения. После изменения состояния, все элементы представления которые были подписаны на хранилище получают новые данные и в соответствии с ними обновляются. Подход *Flux* сильно похож на паттерн проектирования “Наблюдатель”.

4.3.2 Структура проекта

В силу специфики фронт-энд разработки и *Flux*-подхода в этой области разработки применяется паттерн организации структуры проекта под названием *Ducks*. Фронт-энд разработка предполагает большое количество блоков, каждый из которых обычно имеет свой файл для действия, типа действия, редюсера, состояния, компонента и контейнера. Организовывать структуру проекта исходя из назначения конкретного файла, например, все действия хранить в директории *Actions*, было бы неразумно. При росте проекта переходить из одной директории в другую было бы очень утомительно. Для решения этой проблемы имеется паттерн организации директорий – *Ducks*. *Ducks* предполагает содержать всё что относиться к конкретному блоку (действие, редюсер и т.д.) в одной директории, для каждого блока своей.

Таблица 4.2 – описание корневой директории клиентской части приложения

Название директории	Описание
<i>Components</i>	Содержит компоненты
<i>Containers</i>	Содержит контейнеры
<i>Ducks</i>	Содержит экшены, типы экшенов, редюсеры для каждого блока
<i>Sagas</i>	Отвечает за сторонние эффекты
<i>Services</i>	Сервисы приложения
<i>router.js</i>	Маршруты
<i>store.js</i>	Конфигурация состояния

Разработка с использованием *React* строится вокруг компонентов.

Компонент - это блок пользовательского интерфейса, выполняющий некоторый функционал и имеющий возможность хранить внутреннее состояние. Компонентом может быть выбор цвета продукта или навигационная панель. Компонент внутри содержит некоторую логику и может отрисовывать *html* в соответствии с ней.

Однако, в сообществе разработчиков *React* приложений принято условно разделять компоненты на компоненты и контейнеры. С технической точки зрения различий нет, но с точки зрения назначения она есть.

Контейнеры должны содержать в себе логику работы блока, манипулировать его состоянием, показывать различные компоненты и передавать им данные. Другими словами выполнять всю “умную”, высокоуровневую работу.

Компоненты, в свою очередь, выполняют очень простую функцию – в соответствии с полученными данными от контейнеров отображают *html*. Здесь не должно быть никакого манипулирования информацией.

Файл маршрутов (*router.js*) отвечает за выдачу контейнеров в соответствии с *url*-запросом.

Файл *store.js* отвечает за создание общего состояния приложения.

Директория *Saga* содержит в себе реализацию сторонних эффектов.

4.3.3 Сторонние эффекты

Как было описано ранее и изображено на рис.4.8 во Flux архитектуре имеются действия, редюсеры, состояние (*store* - стор) и представления.

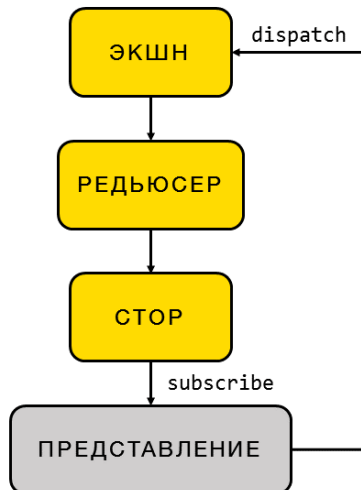


Рисунок 4.8 - схема без сторонних эффектов

Ни один из этих компонентов по назначению не подходит для содержания таких функций приложения, как например, запросы к серверу, логирование событий или отображение иконки загрузки во всём приложении. Такие действия называются *side-effects*, далее будем их называть сторонние эффекты.

Для организации сторонних эффектов используется библиотека *Saga*. Она “встраивается” в схему, слушает и реагирует на определенные типы действий. В схеме на рис.4.9 этот момент обозначен под названием “Мидлвейр”.



Рисунок 4.9 – схема со сторонними эффектами

Например, пользователь пытается загрузить страницу со списком врачей. Генерируется событие загрузки, срабатывает сторонний эффект в котором посылается запрос на сервер. Далее редюсер изменяет состояние приложения на “загрузка”. Компонент представления показывает загрузку. Тем временем “мидлвейр” продолжает ожидать ответ от сервера и в момент получения ответа генерирует событие успешного окончания загрузки. На это событие реагирует редюсер, изменяя состояние на “успешная загрузка”, и добавляет список врачей в общее состояние приложения. Компоненты представления, подписанные на состояние списка врачей, обновляют своё отображение со списком врачей.

4.3.4 Использование компонентов

Как было описано ранее, клиентское приложение, написанное с использованием *React*, в своей основе содержит компоненты, а именно контейнеры – “умные компоненты” и компоненты визуализации.

Опишем более подробно как контейнеры управляют компонентами на примере страницы видео-консультаций пациента с врачом (рис. 3.14).

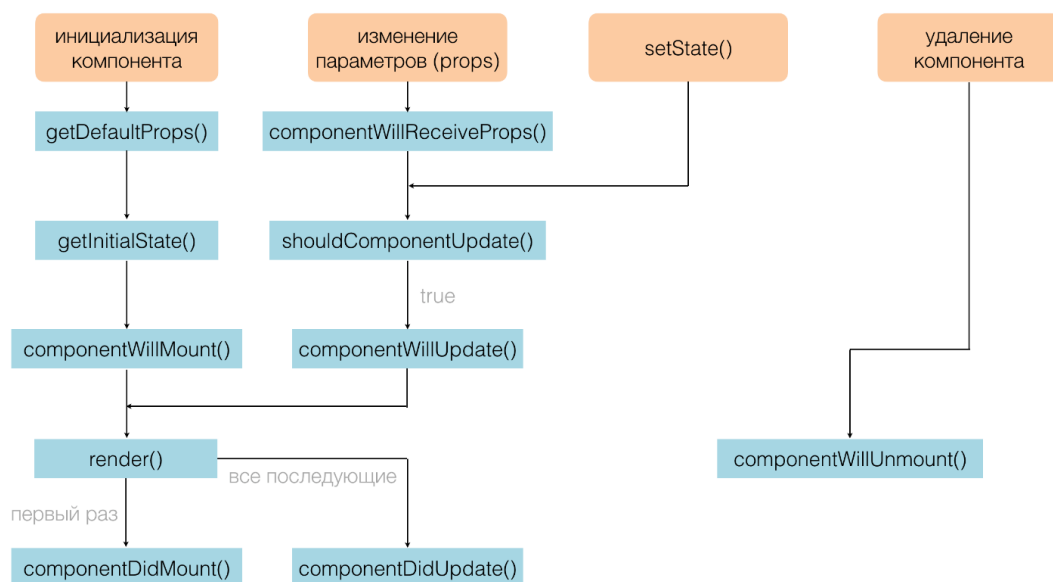


Рисунок 4.10 - жизненный цикл React компонента

Жизненный цикл компонента состоит из нескольких этапов:

- *componentWillMount* – вызывается перед отрисовкой компонента;
- *render* – отрисовка компонента;
- *componentDidMount* – вызывается после отрисовки компонента;
- *componentWillUnmount* – вызывается перед удалением компонента из объектного дерева браузера (*DOM*);
- *componentWillReceiveProps* – вызывается перед получением свойств компонента;
- *shouldComponentUpdate* – вызывается после получения свойств или состояния и должна вернуть *true* или *false* для обновления или отмены обновления компонента;
- *componentWillUpdate* – вызывается после *shouldComponentUpdate* если последний возвращает *true*.

Так как компонент является всё-же классом, то помимо перечисленных выше этапов ЖЦ также имеется стандартный этап – конструктор компонента, который вызывается при инициализации объекта класса.

ЖЦ контейнера окна видео-консультаций начинается с метода *componentWillMount*, который вызывается перед началом отрисовки. Далее происходит следующее (рис. 4.11):

1. В *componentWillMount* вызывается действие загрузки консультации по идентификатору, полученному от контейнера выше по иерархии.
2. На это действие реагирует *Saga*, которая отвечает за сторонние эффекты и делает запрос на сервер.
3. ЖЦ доходит до момента отрисовки контейнера и попадает в условие проверки наличия консультации. В первый цикл консультация отсутствует, т.к. запрос от сервера не успел ещё прийти и отрисовка отменяется.

4. Сразу после успешной загрузки консультации с сервера *Saga* генерирует соответствующее событие, и редюсер изменяет общее состояние приложения, в частности состояние консультации.

5. Так как контейнер подписан на состояние консультации, а оно только что изменилось, то *Redux* передает сигнал об обновлении контейнеру. Контейнер получает обновлённое состояние.

6. ЖЦ контейнера доходит до момента отрисовки и передает текущие свойства контейнера компонентам видео, чату и кнопкам.

```
46 class PatientVideoContainer extends Component {
47   componentWillMount = () => {
48     this.props.loadConsultation(this.props.params.consultationId);
49   };
50
51   render() {
52     if (!this.props.consultation.id) {
53       return null;
54     }
55
56     return (
57       <div className="video-container content-container -wide" sty
58         <div className="video-container__video">
59           <Video {...this.props} />
60         </div>
61         <div className="video-container__tabs">
62           <div className="video-container__chat-without-tabs">
63             <Chat {...this.props} />
64           </div>
65           <div className="video-container__buttons">
66             <CallButtons {...this.props} />
67           </div>
68         </div>
69       </div>
70     );
71   }
72 }
```

Рисунок 4.11 - код контейнера видео-консультации

4.4 Технология *WebRTC*

Для обмена медиа-данными, а именно видео и аудио в работе использовалась технология *WebRTC*. *WebRTC* - (*Web Real-Time Communication*) - открытая технология передачи потоковых данных в браузере по технологии точка-точка. На данный момент поддерживается большинством современных браузеров. Использование технологии позволяет отказаться от сторонних приложений для обмена потоками данных, например, от приложения скайп.

Перед началом использования технологии был проведен анализ преимуществ и недостатков технологий, а также была составлена таблица поддержки браузерами технологии *WebRTC* (табл. 4.2).

Таблица 3.2 - поддержка технологии WebRTC браузерами

Браузер	Поддержка <i>WebRTC</i>	Поддержка <i>VP8</i>	Поддержка <i>H.264</i>	Захват экрана
<i>Google Chrome</i>	Да	Да	Да	Да
<i>Mozilla FireFox</i>	Да	Да	Да	Да
<i>Opera</i>	Да	Да	Да	Да
<i>Microsoft Edge</i>	Да	Нет	Да	Нет
<i>Safari</i>	Да	Нет	Да	Нет
<i>IE</i>	Нет	Нет	Нет	Нет

Из таблицы совместимости браузеров (табл. 4.2) видно, что только *Internet Explorer* не поддерживает технологию полностью. Браузеры *Safari* и *Microsoft Edge* поддерживают технологию частично: поддержка *WebRTC* технологии и поддержка кодирования в *H.264* имеется в этих браузерах. Современные

браузеры, такие как *google chrome*, *firefox* и *opera* поддерживают технологию полностью.

Из преимуществ технологии можно отметить следующее:

- не требует установки;
- высокое качество связи;
- защищенность соединения;
- гибкость в реализации интерфейса;
- открытый исходный код;
- кроссплатформенность.

А из недостатков:

- отсутствие стандарта сигнализации;
- отсутствие возможности видеоконференций по умолчанию.

Учитывая то небольшое количество недостатков, а также большое количество преимуществ, технология была выбрана для организации видео-сообщения между узлами.

4.4.1 Использование и детали технологии *WebRTC*

Каждый клиент (браузер) поддерживающий технологию *WebRTC* имеет доступ к *API* (*application programming interface* – интерфейс программирования) медиа объекта (*MediaStream*), в который необходимо указать тип и источник медиа данных. В свою очередь медиа поток состоит из каналов (*MediaTrack* объектов), где нужно сконфигурировать различные виды потоков (аудио, видео). Например, в случае данной работы необходимо было создать на каждом клиенте один общий медиа поток, состоящий из двух медиа-каналов - аудио и видео и синхронизировать их между собой. Для указания типов медиа-каналов нужно указать в конфигурации свойство *kind* (*video* или *audio*), а для

синхронизации достаточно воспользоваться свойством *label* и установить его одинаковым для синхронизируемых каналов.

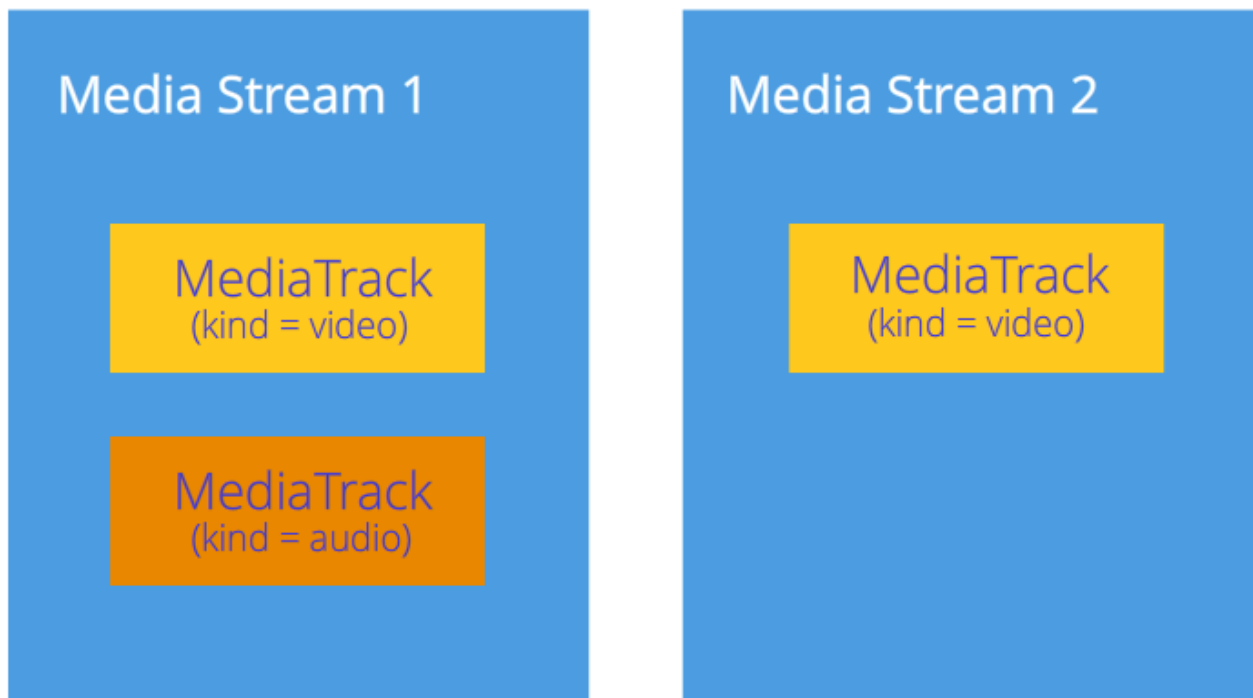


Рисунок 4.12 – конфигурация медиа-потоков

Для создания соединения клиенты должны договориться о двух вещах:

1. О логическом соединении - какой формат данных, какие кодеки, тип данных.
2. О физическом соединении - *IP*-адреса, порты, тип соединения (*TCP*, *UDP*).

4.4.2 Логическое соединение – *SDP*

SDP - (*Session Description Protocol*) - протокол дескриптора сессии. Так как устройства на клиентах всегда будут разные - различные веб-камеры, микрофоны, различные кодеки и драйвера используемые на этих устройствах, то их необходимо каким-то образом скоординировать для взаимодействия. Для этого в *WebRTC* имеется дескриптор сессии - *SDP*. Дескриптор *SDP* передается от одного клиента другому с помощью сигнального механизма, речь о котором

пойдет дальше. После получения вторым клиентом *SDP* дескриптора, он создаст свой дескриптор на основе совпадений конфигураций с полученным дескриптором от первого узла, который необходим будет отправить обратно.

Например, на рис. 4.13 клиент 1 отправляет в дескрипторе информацию о поддержке кодеков *A* и *B*, клиент 2 поддерживает кодеки *B* и *C*. На основе этого клиент 2 решает, что информация для клиента 1 будет кодироваться кодеком *B* и сообщает ему об этом. Таким образом клиенты обмениваются дескрипторами и приходят к общему соглашению о формате обмена данными.

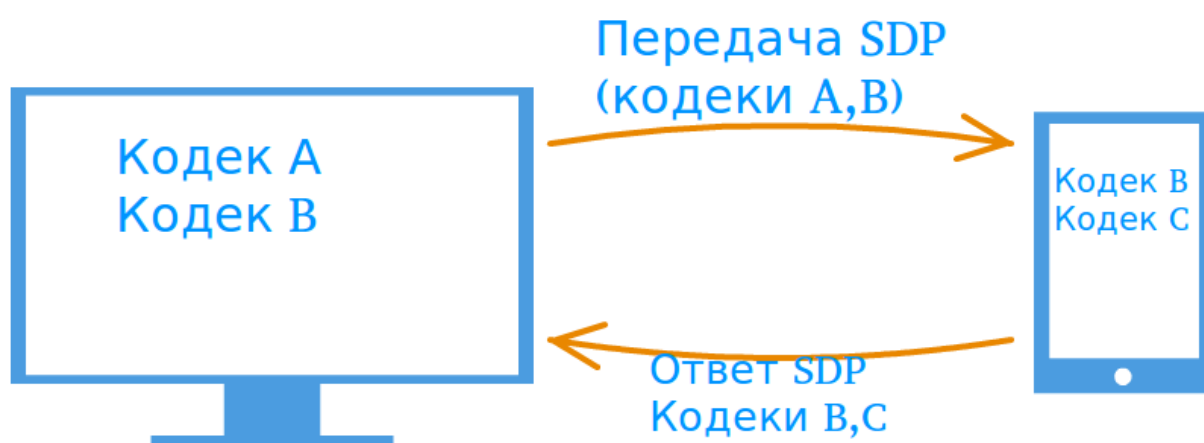


Рисунок 4.13 – обмен клиентов дескрипторами *SDP*

4.4.3 Физическое соединение

Логическое соединение с помощью *SDP* дескриптором было установлено. Теперь необходимо установить физическое. Для этого используется *ICE* протокол (*Ice candidate*). Он содержит информацию о физическом расположении клиента и протоколе передачи информации по сети (*IP, PORT, TCP/UDP*). Как и в случае с *SDP*, токены *Ice candidate* генерируются в *WebRTC* и передается сигнальному серверу для дальнейшей отправки второму клиенту. Второй клиент, получив *Ice candidate* токены содержащие информацию о расположении в сети будет знать с кем и по какому адресу устанавливать соединение.

4.4.4 Сигнальный сервер

Сигнальный сервер, как описано выше, необходим для координации узлов, а именно для обмена данными о физическом расположении узлов в сети и их способам кодирования информации. В качестве сигнального сервера для координирования клиентов используется сторонний сервис *Twilio*. Сервис предлагает *STUN*-сервер, необходимый для физического соединения узлов, не имеющих публичных *IP*-адресов, а находящихся “за маршрутизаторами”. Другими словами, *STUN*-сервер делает доступным клиент за пределами сети.

Суммируя описанное выше, процесс работы технологии *WebRTC* можно описать всего в основных 5 шагов:

1. Инициализируется медиа-объект *WebRTC* узлом один.
2. Информация о инициализирующем узле (один) отправляется на сигнальный сервер.
3. Сигнальный сервер отправляет информацию от узла один узлу два.
4. Узел два обрабатывает информацию узла один и в соответствии с ней отправляет ответную информацию узлу один через сигнальный сервер.
5. Начинается процесс передачи закодированных данных между узлами один и два в обход сигнального сервера.



Рисунок 4.14 – схема работы *WebRTC*

Непосредственно для передачи медиа-потоков сигнальный сервер не нужен. Как видно из пунктов выше, сигнальный сервер нужен лишь для обмена метаданными для установления соединения типа точка-точка.

4.5 Технология веб-сокетов

Для списка врачей и отображения их текущих статусов использовались веб-сокеты. Веб-сокеты позволяют создать соединение в режиме реального времени и дают возможность обмениваться данными между клиентом и сервером. В отличие от *HTTP*, веб-сокеты могут работать двунаправленно, не разрывая связи между узлами. Ниже приведен пример работы *HTTP* протокола:

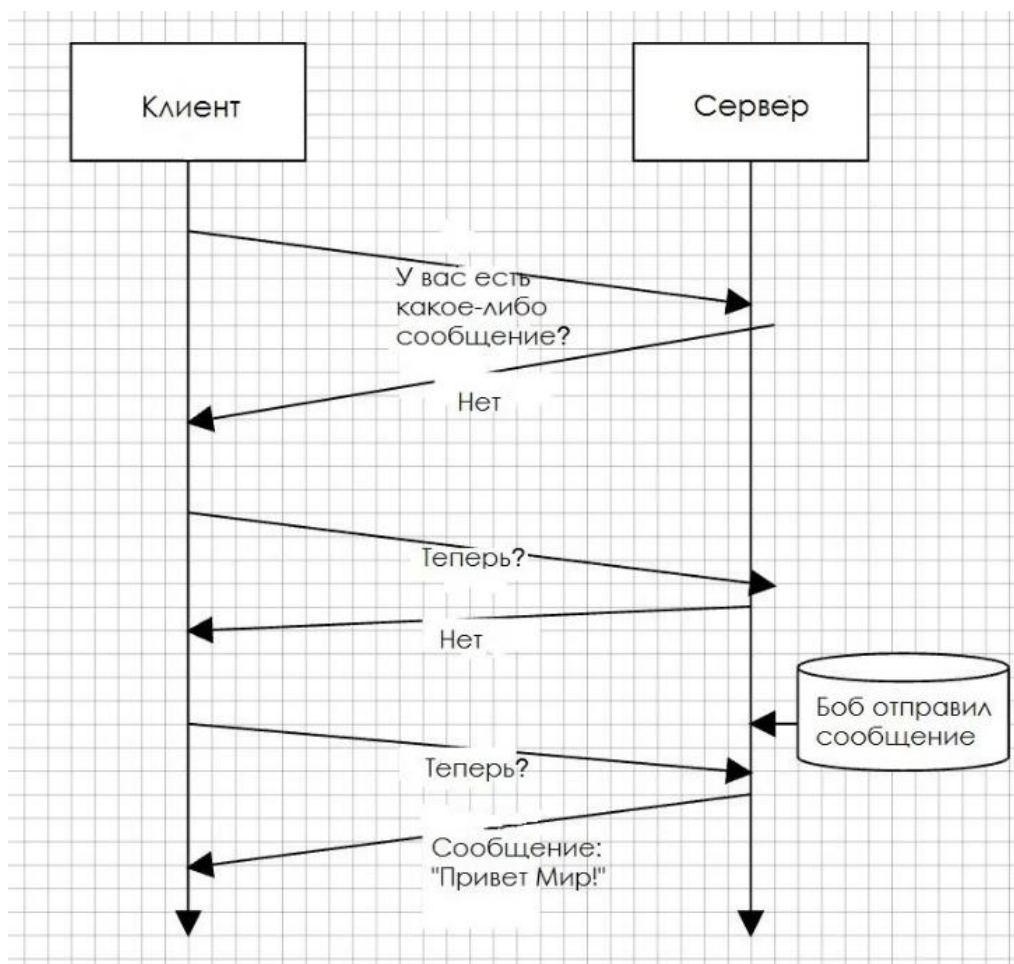


Рисунок 4.15 – схема работы HTTP-протокола

Как видно из схемы на рис. 4.15 для того, чтобы узнать появилось ли новое сообщение от сервера, клиент должен постоянно проверять наличие обновлений. Из этого вытекает одна из двух проблем: либо слишком большая нагрузка на сервер с большим количеством обращений от клиента, либо недостаточная интерактивность для клиента.

Веб-сокеты, в отличие от *HTTP*, устанавливают соединение единожды и в момент появления обновлений сервер сигнализирует клиенту об этом и посылает сообщение.

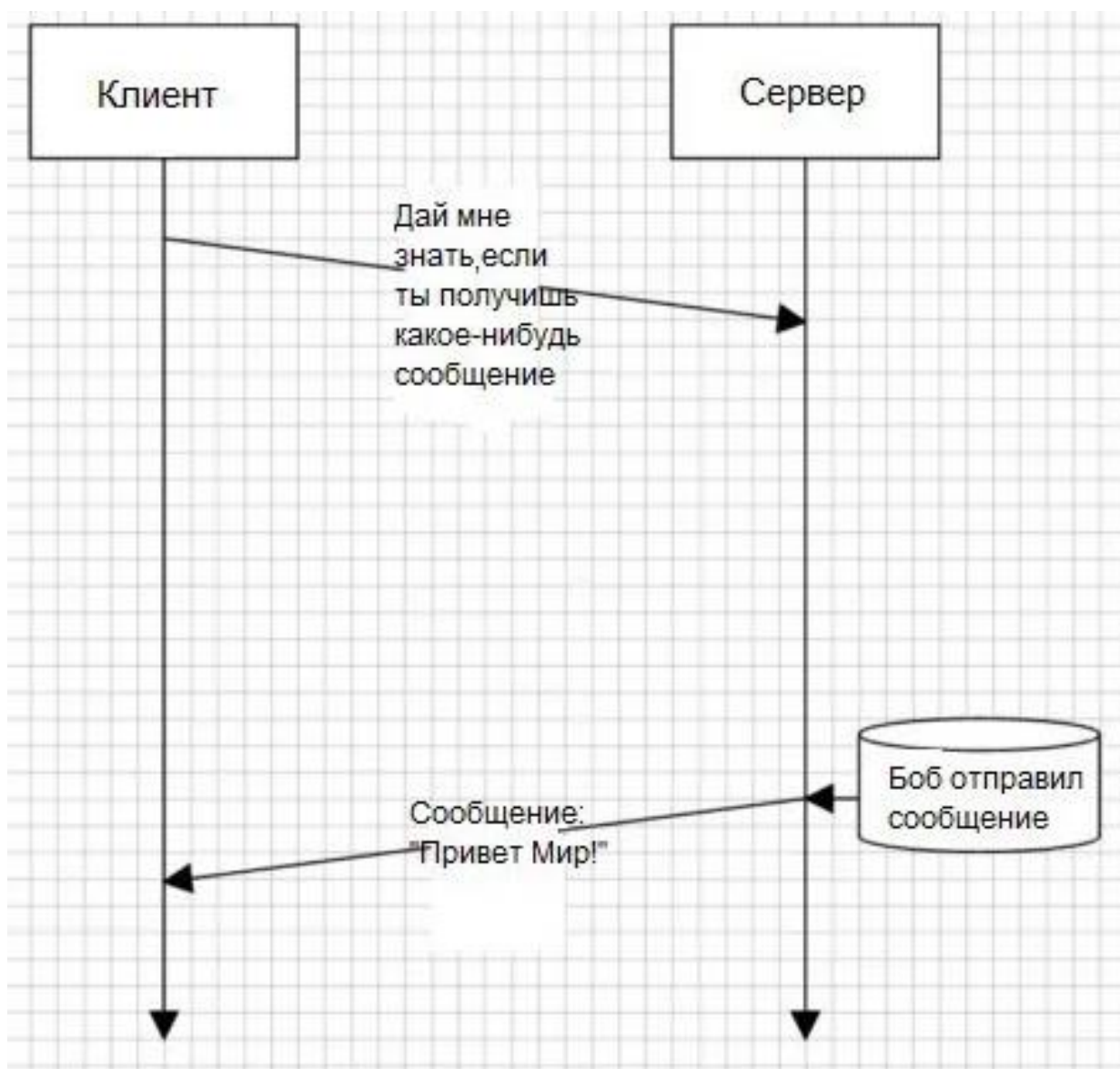


Рисунок 4.16 – схема работы веб-сокета

4.6 Итоги

Спроектировано и разработано клиент-серверное приложение по *REST*-архитектуре с применением современных технологий и подходов: фреймворк – *express* вместе с *ORM*-системой *mongoose*, библиотек – *react*, *redux*, *saga*, контейнера виртуализации в виде *docker*, технологии передачи медиа-контента *WebRTC* по схеме точка-точка, а также технологии веб-сокетов. В проекте были использованы архитектурные паттерны *MVC* на серверной стороне и *Flux* на стороне клиента. Поставленные цели достигнуты.

5. Потенциальные доработки системы

5.1 Хранение персональных данных

Хранение и обработка персональных пользовательских данных регулируется законодательством стран. К хранению и обработке данных пациента следует подходить ещё более ответственно: ни один пациент не захочет, чтоб данные о его болезнях и о прошедших консультациях с врачом попали в руки третьих лиц. В связи с этим в ИС для внедрения в медицинское учреждение необходимо разработать комплекс мер по обеспечению информационной безопасности. На данный момент, личные данные пользователя, за исключением пароля, хранятся в первоначальном, незашифрованном виде.

5.2 Видео-конференция

Зачастую для того, чтобы определиться с диагнозом пациенту, необходимо собрать медицинский совет, состоящий из разноплановых врачей – т.е. консилиум. К сожалению, технология на которой реализована передача медиа-потокa (*WebRTC*) работает по схеме точка-точка, т.е. передача данных между двумя клиентами. В связи с этим без доработок информационной системы создать видео-конференцию не представляется возможным.

5.3 Мобильное приложение

В работе был разработан сервер, отдающий и обрабатывающий данные и клиентское веб-приложение. На данный момент многие пользователи аналогичных сервисов пользуются мобильными приложениями для получения видео-консультаций. Приложения доступны для различных платформ: *IOS*, *Android* и, иногда *Windows*. Разработка мобильного приложения занимает схожее время с разработкой веб-приложения (в случае автора, полтора месяца), что

является большим сроком. В связи с этим было принято решение отказаться от разработки мобильного приложения.

Вместо этого, в веб-приложении была разработана мобильная версия, которая адаптируется под разрешение экрана устройства. В мобильной версии сохраняется весь функционал, доступный в полной версии приложения.

Реализовать мобильное приложение в дальнейшем будем не так сложно: веб-сервер готов, все необходимые данные он предоставляет и обрабатывает. Остаётся вопрос об организации видео-связи. При использовании технологии *WebRTC* кодирование и декодирование видео и аудио потоков реализовано по умолчанию.

Заключение

Телемедицина является неотъемлемой частью технического прогресса и её широкое распространение, по мнению автора, дело времени. На данный момент имеется всё необходимое с технической точки зрения для оказания данного вида медицинских услуг. Имеются некоторые трудности в правовом поле, которые всё же постепенно решаются. Например, подписан закон, по которому с 1 января 2019 года будет реализована возможность получать от врача электронные рецепты. По некоторым данным [8] считается, что услуги телемедицины дешевле в 20 раз чем поездка на очную консультацию с врачом с Урала в Москву и в 40 раз для жителей Забайкалья и Якутии. Оказание услуг удаленных видео-консультаций однозначно поможет сэкономить денежные средства и время пациентов.

В выпускной квалификационной работе была разработана клиент-серверная система для оказания услуг дистанционных видео-консультаций с применением современных инструментов и подходов к разработке информационных систем. Данная система готова к внедрению в медицинские учреждения с некоторыми оговорками (см. раздел 5).

Для работы в приложении имеется весь необходимый функционал:

- бронирование времени консультаций, как для врачей, так и для пациентов;
- поиск врачей и пациентов;
- приглашение пациентов врачом;
- личные кабинеты пользователей;
- обмен сообщениями между врачом и пациентом;
- возможность прикрепления материалов;
- видео-консультации.

Поставленные задачи выполнены, цель работы достигнута.

Основные обозначения и сокращения

API – *Application Program Interface*, интерфейс программирования приложений

P2P – *Peer to Peer*, пиринговая архитектура типа «клиент-клиент»

TCP – *Transmission Control Protocol*, протокол управления передачей IP – Internet Protocol, межсетевой протокол

SDP - (*Session Description Protocol*) - протокол дескриптора сессии

ICE – протокол физического соединения узлов

WebRTC – *p2p* технология передачи медиа-потокa в браузере

WebSocket – протокол полнодуплексной связи поверх *TCP*-соединения

UI – *User Interface*, пользовательский интерфейс

MVC – архитектурный паттерн. Схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер

JS – (*JavaScript*) – скриптовый язык программирования с динамической типизацией

NodeJs – *JS* платформа для разработки серверных приложений

Express – *NodeJs* фреймворк

Framework – Фреймворк. Набор вспомогательных модулей для ускорения разработки

React – *JS* библиотека для разработки пользовательских интерфейсов

Redux – *JS* библиотека для организации общего состояния приложения и однонаправленного потока данных

Action – действие. Содержит тип и данные действия

Reducer – редюсер. Меняет состояние хранилища

Store – хранилище. Содержит общее состояние приложения

View – представление. Отображает данные в виде html-кода

Saga – JS библиотека для организации сторонних эффектов

Docker – ПО для виртуализации на уровне операционной системы

Postman – ПО для отладки API серверов путём отправки http-запросов

JSON – *Javascript Object Notation*. Легко читаемый для человека формат обмена данными

HTML – язык разметки гипертекста

RESTful – архитектурный стиль организации клиент-серверный приложений

HTTP – протокол передачи гипертекста. Протокол прикладного уровня передачи данных

ОС – Операционная Система

Список используемых источников

1. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Эрих Гамма [и др.]. – М.: Питер, 2016. – 366с.
2. С. Макконнелл. Совершенный код. Мастер-класс. М.: БВХ-Петербург, 2017. – 896с.
3. С. Чакон, Б. Штрауб. *Git* для профессионального программиста. М.: Питер, 2017. – 496с.
4. И. Браун. Веб-разработка с использованием *Node* и *Express*. Полноценное использование стека *JavaScript*. М.: Питер, 2017. – 366с.
5. А. Бэнкс, Е. Порселло. *React* и *Redux*. Функциональная веб-разработка. М.: Питер, 2018. – 336с.
6. К. Аквино, Т. Ганди, *Front-end*. Клиентская разработка для профессионалов. *Node.js, ES6, REST*. М.:Питер, 2017. – 512с.
7. Информационное общество в РФ [электронный ресурс]. – Режим доступа: <http://emag.iis.ru/arc/infosoc/emag.nsf/BPA/91281>, (дата обращения: 30.04.2018) свободный. – Загл. с экрана.
8. Закон о телемедицине. [электронный ресурс]. – Режим доступа: <https://rg.ru/2018/01/01/v-rossii-vstupayet-v-silu-zakon-o-telemedicine.html>, (дата обращения: 02.05.2018) свободный. – Загл. с экрана.
9. Создание архитектуры программы. [электронный ресурс]. – Режим доступа: <https://habr.com/post/276593/>, (дата обращения: 08.05.2018) свободный. – Загл. с экрана.
10. Коды ответа *HTTP*. [электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/ru/docs/Web/HTTP/Status>, (дата обращения: 18.05.2018) свободный. – Загл. с экрана.