Оглавление

[Введение 3](#_Toc515319621)

[1. Постановка задачи 4](#_Toc515319622)

[1.1 Актуальность темы 4](#_Toc515319623)

[1.2 Цель работы 5](#_Toc515319624)

[1.3 Задачи, подлежащие решению для достижения цели 5](#_Toc515319625)

[1.4 Методология, средства решения задач 5](#_Toc515319626)

[1.5 Стартовые данные для выполнения работы 5](#_Toc515319627)

[2. Обзор существующих аналогов 5](#_Toc515319628)

[2.1 Общие требования и процесс получения консультаций 6](#_Toc515319629)

[2.2 Яндекс.Здоровье 6](#_Toc515319630)

[2.3 Doc+ Онлайн 7](#_Toc515319631)

[2.4 Доктор рядом 7](#_Toc515319632)

[2.5 Онлайн Доктор 8](#_Toc515319633)

[2.6 Итоги сравнения 8](#_Toc515319634)

[3. Функционал приложения 10](#_Toc515319635)

[3.1 Аутентификация, авторизация, регистрация 10](#_Toc515319636)

[3.2 Главная страница 13](#_Toc515319637)

[3.3 Функционал пользователей 16](#_Toc515319638)

[3.4 Окно видео-консультаций 20](#_Toc515319639)

[3.5 Личный профиль 21](#_Toc515319640)

[3.6 Тест на соответствие требованиям видео-консультаций 22](#_Toc515319641)

[4. Архитектура и реализация функционала приложения 24](#_Toc515319642)

[4.1 Сервер 25](#_Toc515319643)

[4.1.1 REST-Архитектура 25](#_Toc515319644)

[4.1.2 Структура проекта 26](#_Toc515319645)

[4.1.3 ORM 28](#_Toc515319646)

[4.1.4 Схема запрос-ответ 29](#_Toc515319647)

[4.1.5 Диаграмма классов 30](#_Toc515319648)

[4.1.6 Docker 31](#_Toc515319649)

[4.1.7 Итог 33](#_Toc515319650)

[4.2 Разработка клиентской части 33](#_Toc515319651)

[4.2.1 *Flux*-паттерн 33](#_Toc515319652)

[Проблема 33](#_Toc515319653)

[4.2.2 Структура проекта 36](#_Toc515319654)

[4.2.3 Сторонние эффекты 38](#_Toc515319655)

[4.3 Технология WebRTC 40](#_Toc515319656)

[4.3.1 Использование и детали технологии 41](#_Toc515319657)

[4.3.2 Логическое соединение – SDP 42](#_Toc515319658)

[4.3.3 Физическое соединение 43](#_Toc515319659)

[4.3.4 Сигнальный сервер 44](#_Toc515319660)

[4.4 Технология Вебсокетов 45](#_Toc515319661)

# 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 и 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 на ликвидирует, однако однозначно поменяет представление об оказании медицинских услуг. Сервис удаленных консультаций поможет пациентам, проживающим в труднодоступных местах, неподвижным пациентам, пациентам, кому необходима срочная мед. консультация. Для получения услуг нет необходимости стоять в очереди, необходимо лишь записаться на прием, либо связаться со свободным врачом в данный момент. Также данный сервис потенциально может сохранять историю пациента, его обращения, рекомендации от врачей, и, даже, анализировать состояние здоровья пациента и делать на основе этого автоматические рекомендации.

# Постановка задачи

## 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. 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. Таким образом, телемедицина высвобождает огромные ресурсы.

Однако существует ряд проблем. На данный момент эта область находится вне правового поля. Возникает острый 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 на целую компанию, а 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.

## Цель работы

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

## Задачи, подлежащие решению для достижения цели

Для достижения поставленной цели были определены следующие задачи:

1. Произвести поиск и анализ существующих аналогичных сервисов
2. Изучить предметную область
3. Спроектировать архитектуру приложения
4. Выбрать технологии для реализации как клиента, так и сервера
5. Выбрать один из методов передачи потока данных между пользователями
6. Написать клиент-серверное приложение с использованием выбранных технологий

## Методология, средства решения задач

## Стартовые данные для выполнения работы

# Обзор существующих аналогов

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

## 2.1 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 видео камера.

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

## 2.2 Яндекс.Здоровье

Компания Яндекс запустила сервис «Яндекс.Здоровье» в 2016 году, которое предоставляет возможность оказывать медицинские услуги от разных врачей с помощью видеоконференций. Имеется возможность получить консультации от терапевта, педиатра, гинеколога и др. врачей, а также получить услугу расшифровку результатов анализов. Плюс ко всему есть консультации ветеринара.

Одной из особенностей сервиса является технология передачи заявку ближайшему необходимому 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**

1. Низкие цены
2. Отсутствие очередей
3. Множество врачей
4. История приёмов
5. Возможность напоминаний

**Недостатки**

1. Не всегда “свой” врач
2. Отсутствие своего мед.центра

## 2.3 Doc+ Онлайн

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

**Преимущества**

1. Отсутствие очередей
2. Множество врачей
3. Свой медицинский центр
4. Низкие цены

**Недостатки**

1. Неудобный интерфейс приложения
2. Нет возможности получить консультацию в выходной
3. Видео-консультаций не являются основным направлением

## 2.4 Доктор рядом

Также как и «Doc+ Онлайн» является частной медицинский клиникой и предоставляет услуги удалённых консультаций. Имеет множество врачей различных направлений и специализаций.

**Преимущества**

1. Отсутствие очередей
2. Множество врачей
3. Свой медицинский центр

**Недостатки**

1. Нет возможности получить консультацию в выходной и ночью
2. Отсутствие мобильного приложения

## 2.5 Онлайн Доктор

В отличии 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.

**Преимущества**

1. Отсутствие очередей
2. Множество врачей
3. Многофункционалное приложение
4. Круглосуточно

**Недостатки**

1. Высокие цены

## 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. Стоимость варьируется в зависимости от сервиса и часто сильно зависит от квалификации врача и его специализации.

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

# Функционал приложения

В ходе выполнения дипломной работы помимо основного функционала, такого как видеосвязь и регистрация, были реализованы сопутствующие удобства, например, поиск врачей, проверка соответствия требованиям для оказания услуг видео-консультаций, чат с возможностью прикрепления материала к консультации и т.д. Ниже приведена диаграмма, описывающие наиболее примечательный функционал приложения.

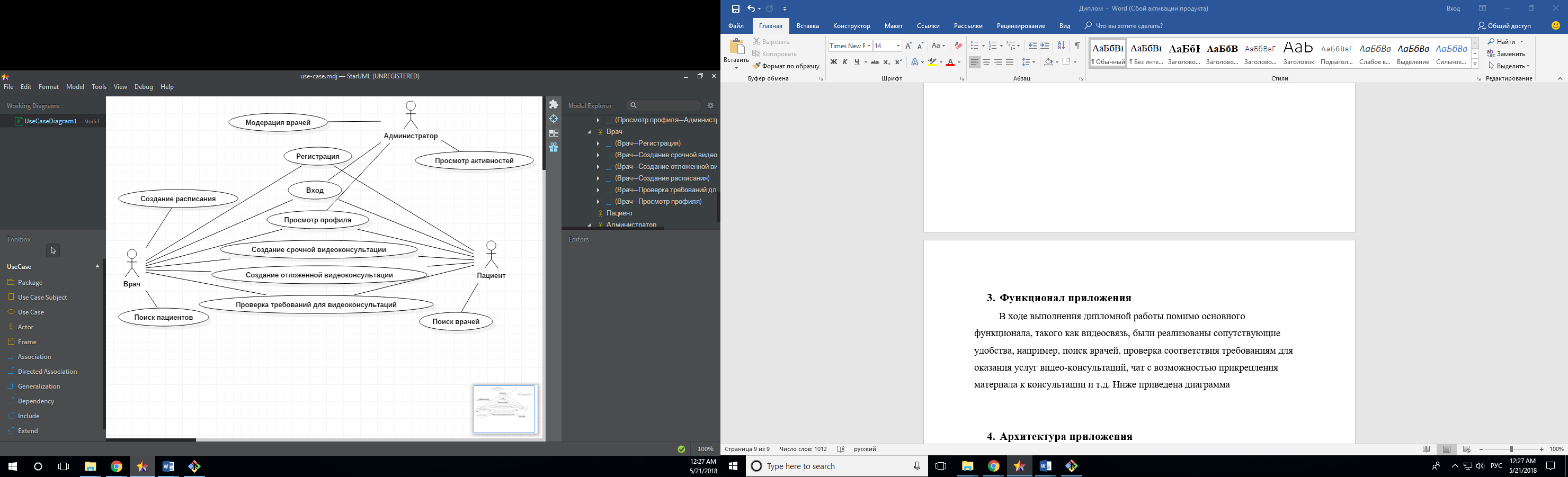


Рисунок 1 - use-case диаграмма

## Аутентификация, авторизация, регистрация

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

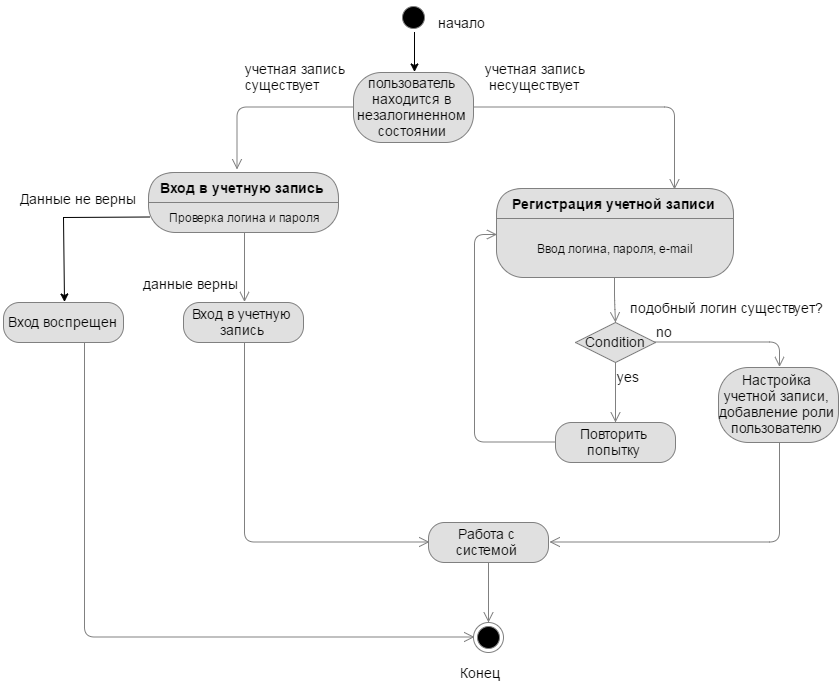


Рисунок 2 - диаграмма деятельности для авторизации и регистрации пользователей

Самостоятельный процесс регистрации 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 подтерждение администратором. Предполагается что администраторы будут внесены в базу данных напрямую.

Для ясности дадим определения терминам аутентфиикация и авторизация.

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

*Авторизация* - процесс провекри наличия прав на совершаемое действие на основе аутентификации.

Так как приложение реализуется по RESTful архитектуре и имеются как клиент так и сервер то, функционал необходимо было его реализовать с обоих сторон.

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

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

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, отсылает 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.

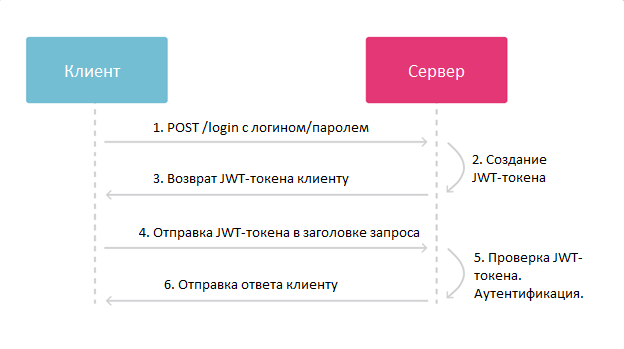


Рисунок 3 - схема получения токена и его использование

JWT-токен – это сгенерированная на сервере последовательность символов, содержащая в себе метаданные о пользователе, например, можно «зашить» в токен роль пользователя, логин, почтовый ящик и др. данные, что и было сделанно. Токен имеет срок жизни, после чего его необходимо обновить. Срок жизни токена также конфигурируется на сервере. Далее получнный 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. 4). У пользователя также есть возможность войти в личный кабинет. Практикующий, но не зарегистрированный врач может подать заявление на регистрацию в системе.

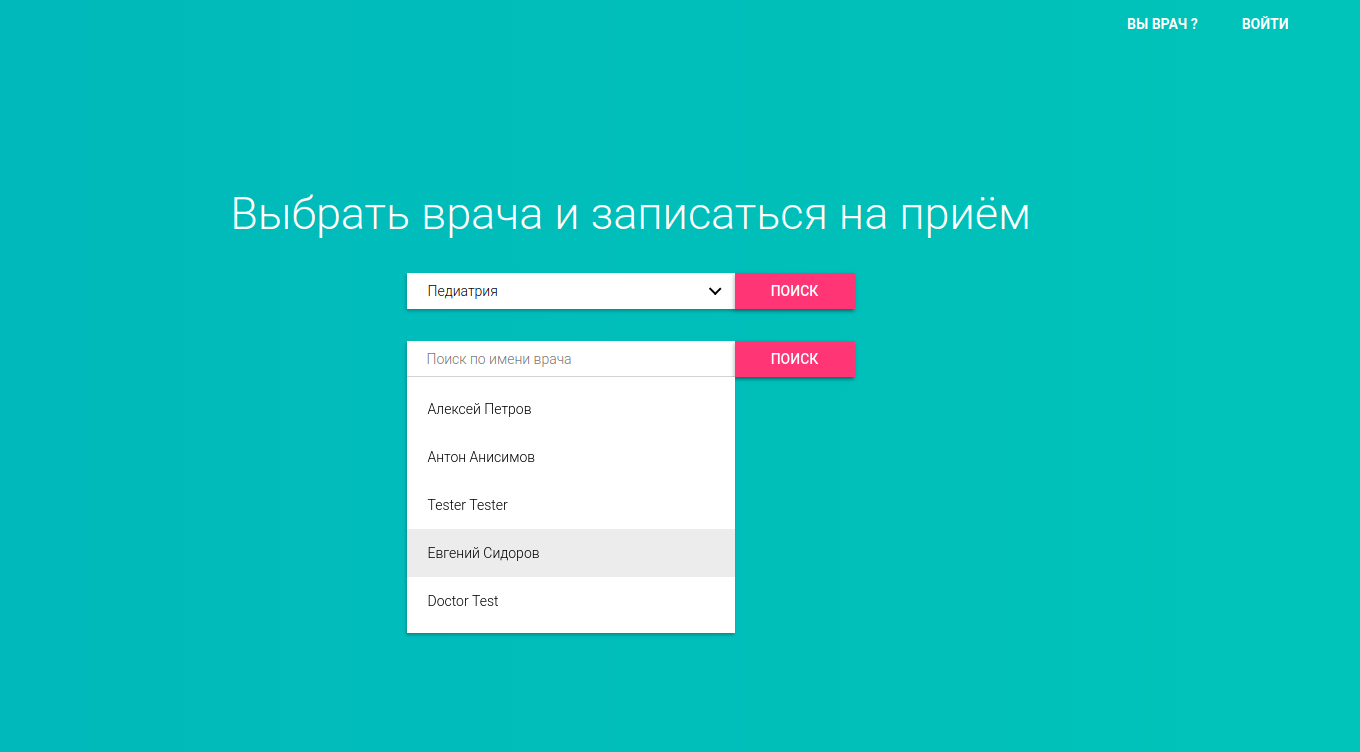
****

Рисунок 4 - главная страница

Интересной особенностью поиска является автозаполнение 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 улучшалась 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. 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.

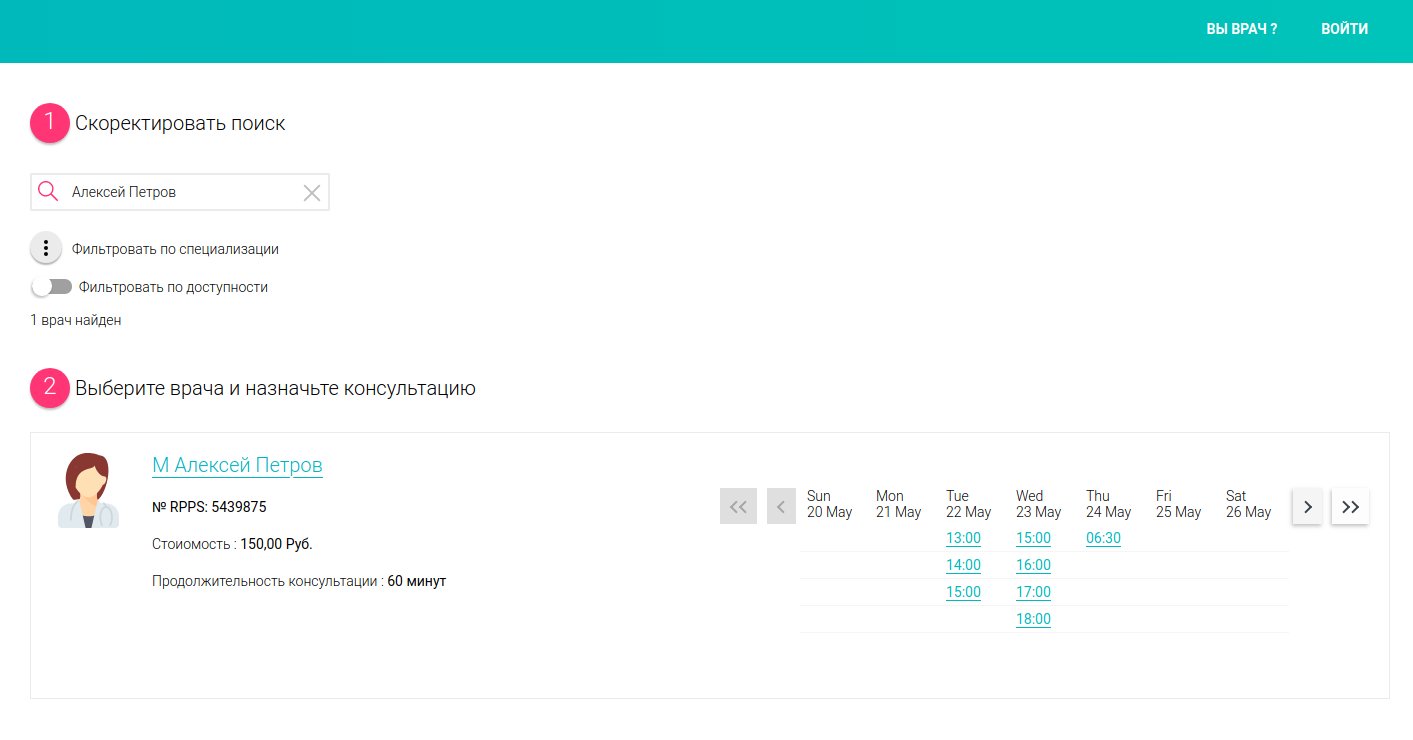


Рисунок 5 – результаты поиска врача

Из интересных технологий можно отметить применение вебсокетов. Вебсокеты были использованы для оторбажения статусов врачей - находиться ли врач сети или нет. Технология вебсокетов будет описана в разделе [].

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

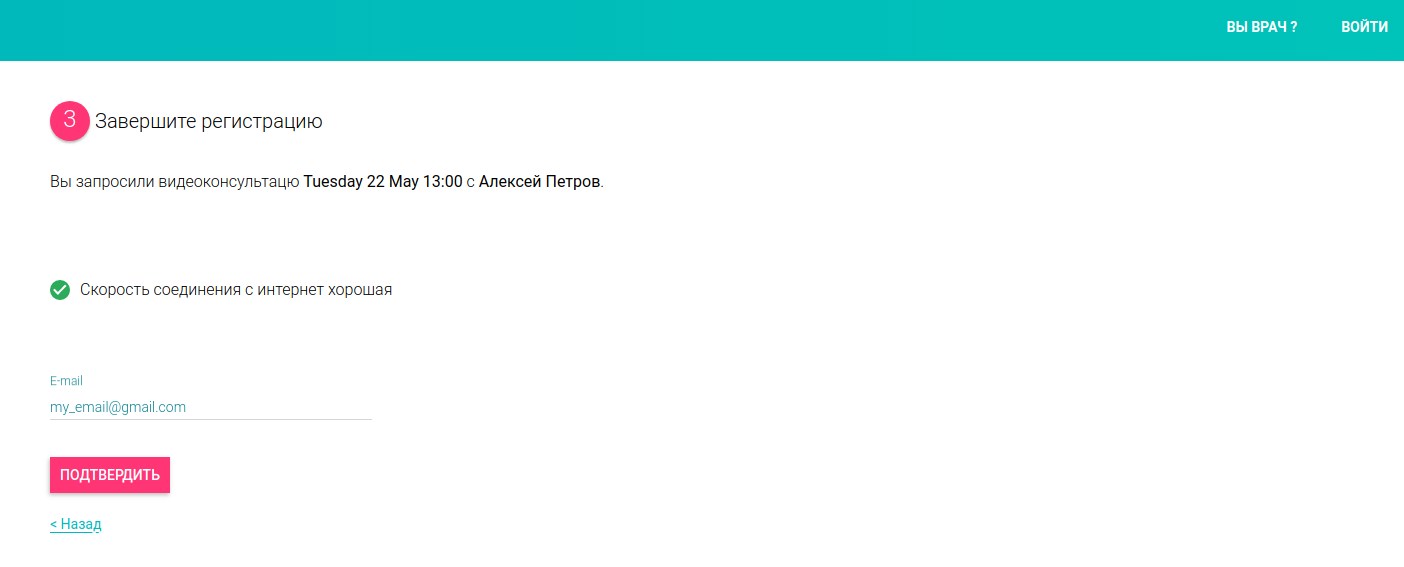


Рисунок 6 – 3 шаг - завершение регистрации

## 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 попадает 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 5).

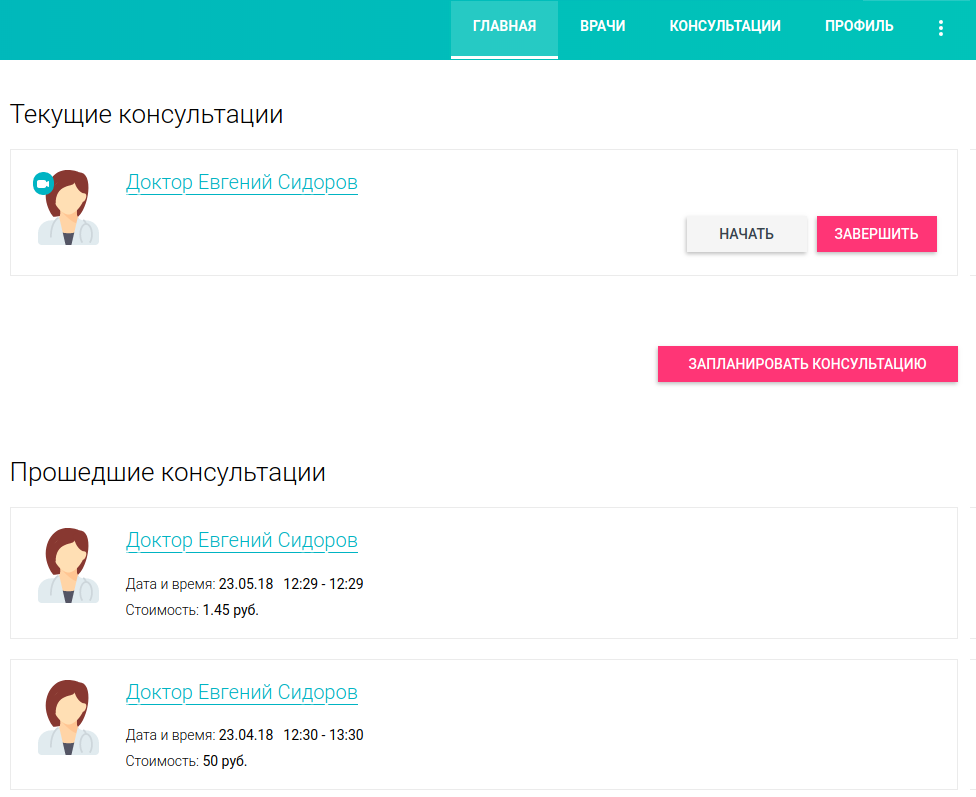


Рисунок 7 - главная страница пациента

Главная страница врача (рис. 8) содержит очередь консультаций в хронологическом порядке разбитую по дням с возможностью переключаться. 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.

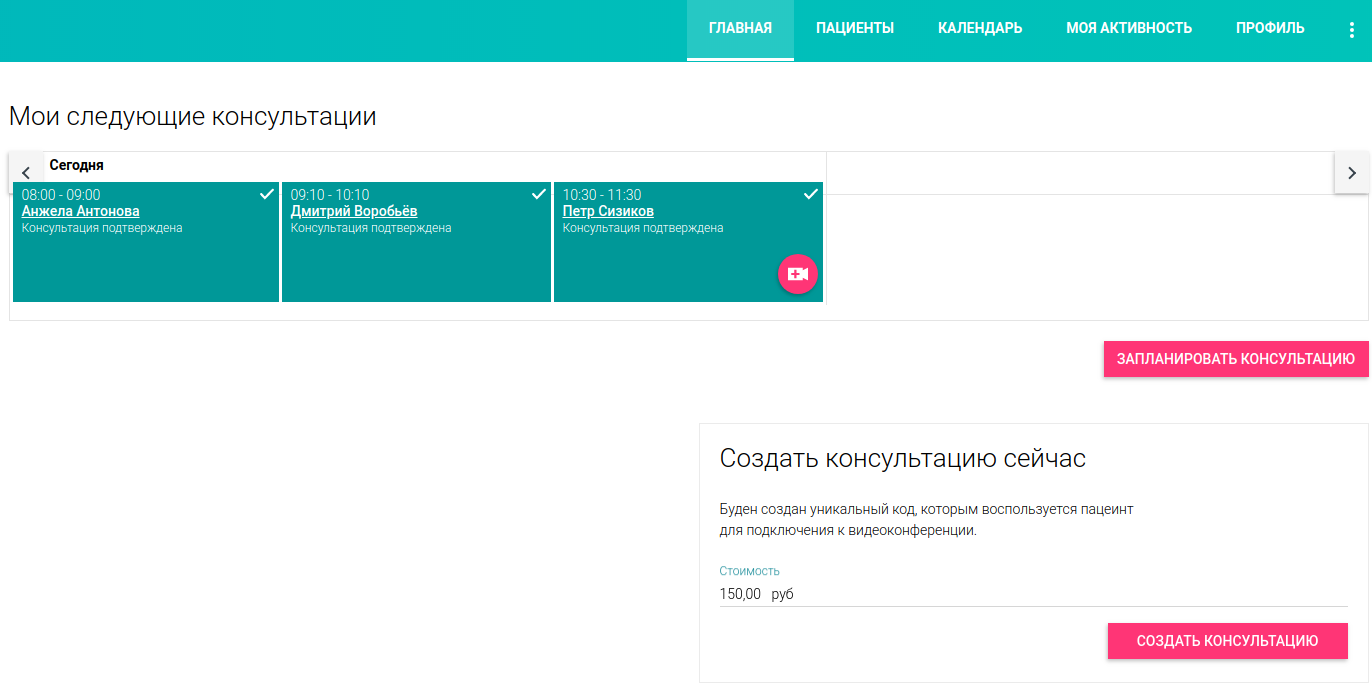


Рисунок 8 - главная страница врача

У врача, как и 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. 9).

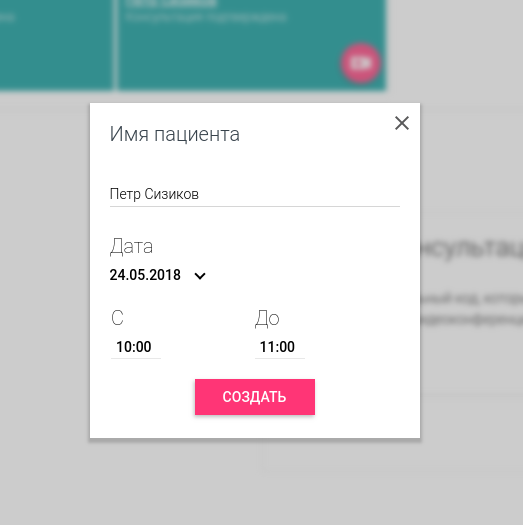


Рисунок 9 – планирование консультации врачом

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.10).

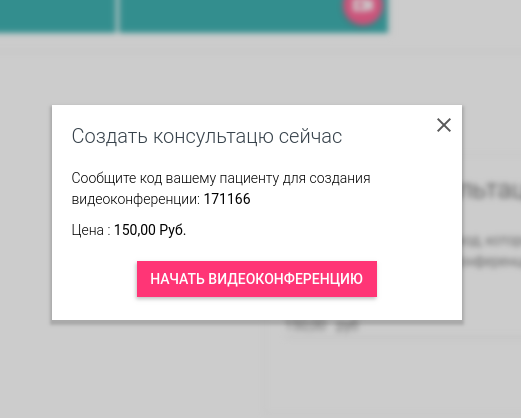


Рисунок 10 - приглашение на моментальную консультацию пациента по коду

Далее врач должен сообщить код консультации своему пациенту, например по телефону, и пациент сможет присоединиться к созданной консультации на страницы входа в приложение (рис. 11).

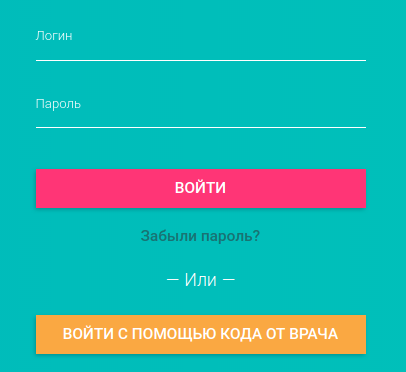


Рисунок 11 – вход в приложение. Обычный и по пригласительному коду.

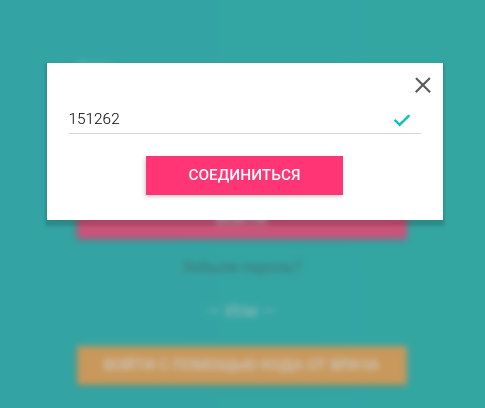


Рисунок 12 – ввод пригласительного кода

Для описания процесса моментальной ниже приведена диаграмма последовательности:

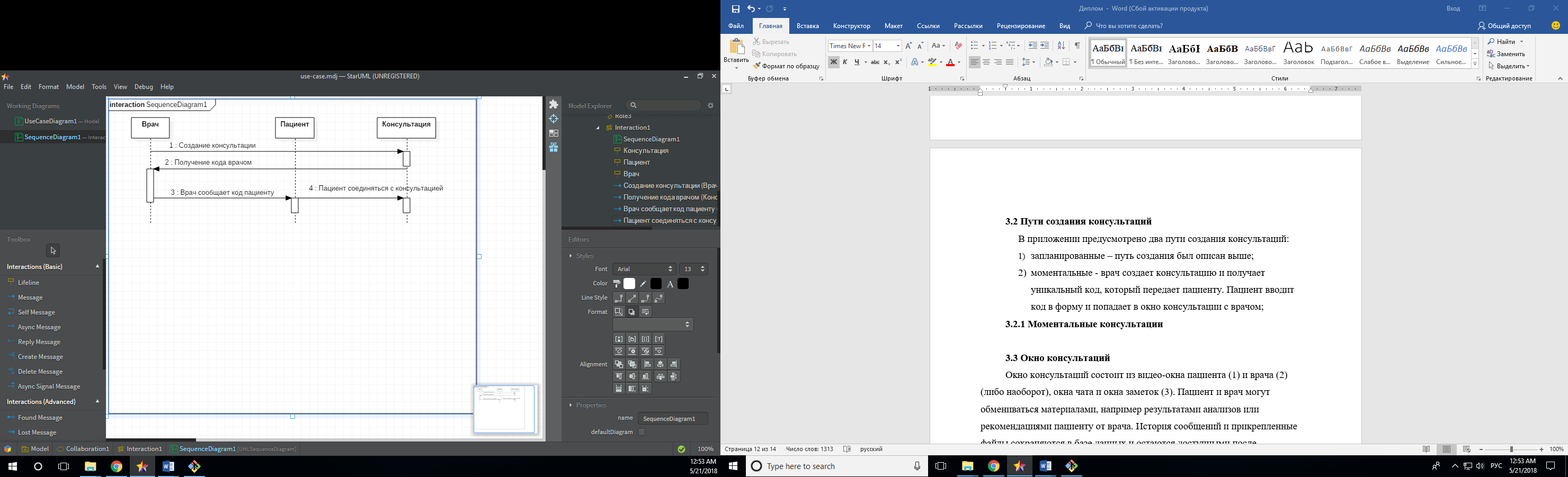


Рисунок 13 – процесс моментальной консультации

## 3.4 Окно видео-консультаций

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

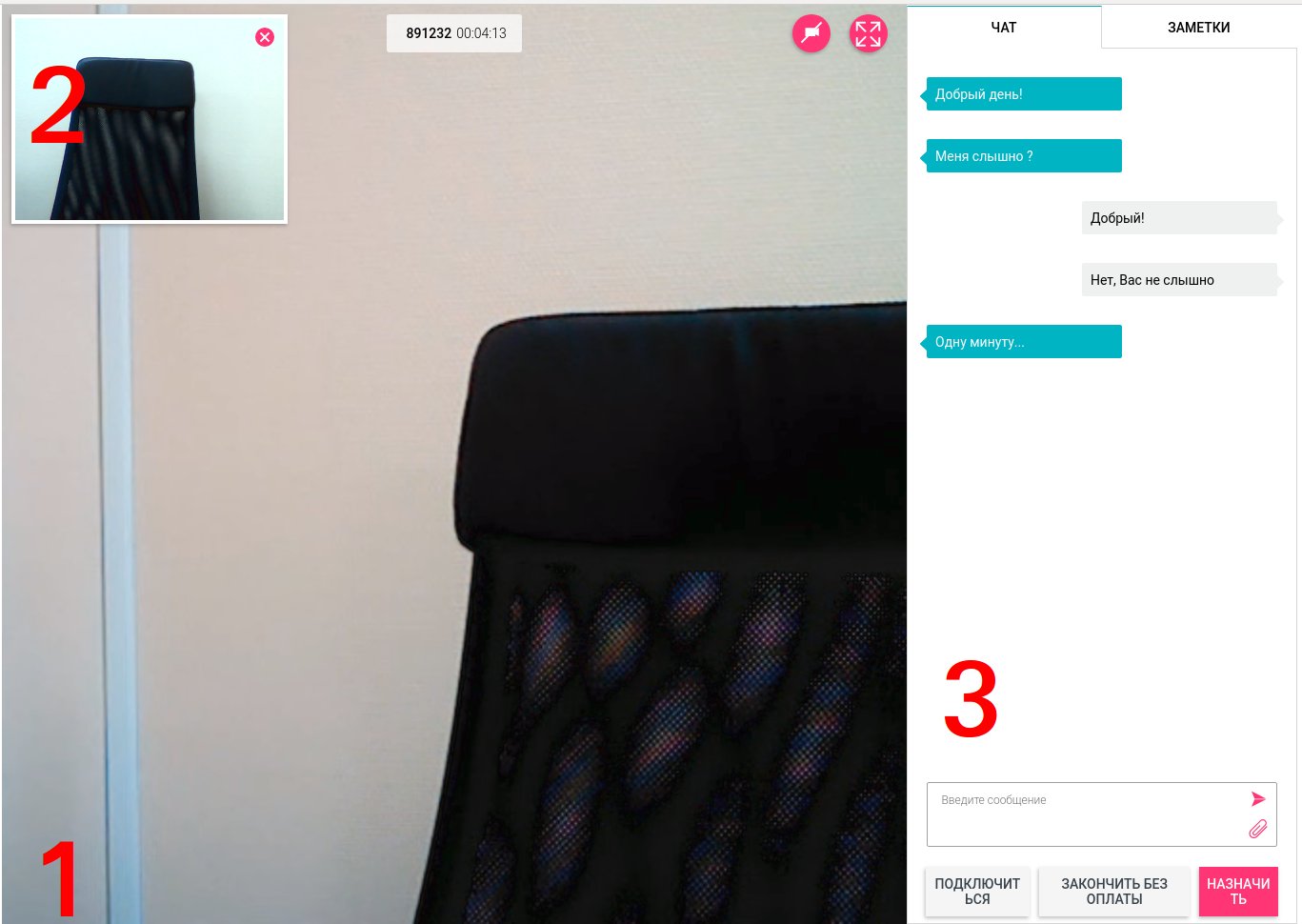


Рисунок 14 - окно видео-консультаций

Окно консультаций состоит из видео-окна пациента (1) и врача (2) (либо наоборот), 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. 14).

## 3.5 Личный профиль

У всех пользователей есть свой личный кабинет с профилем, но наибольшего внимания заслуживает профиль врача. 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.

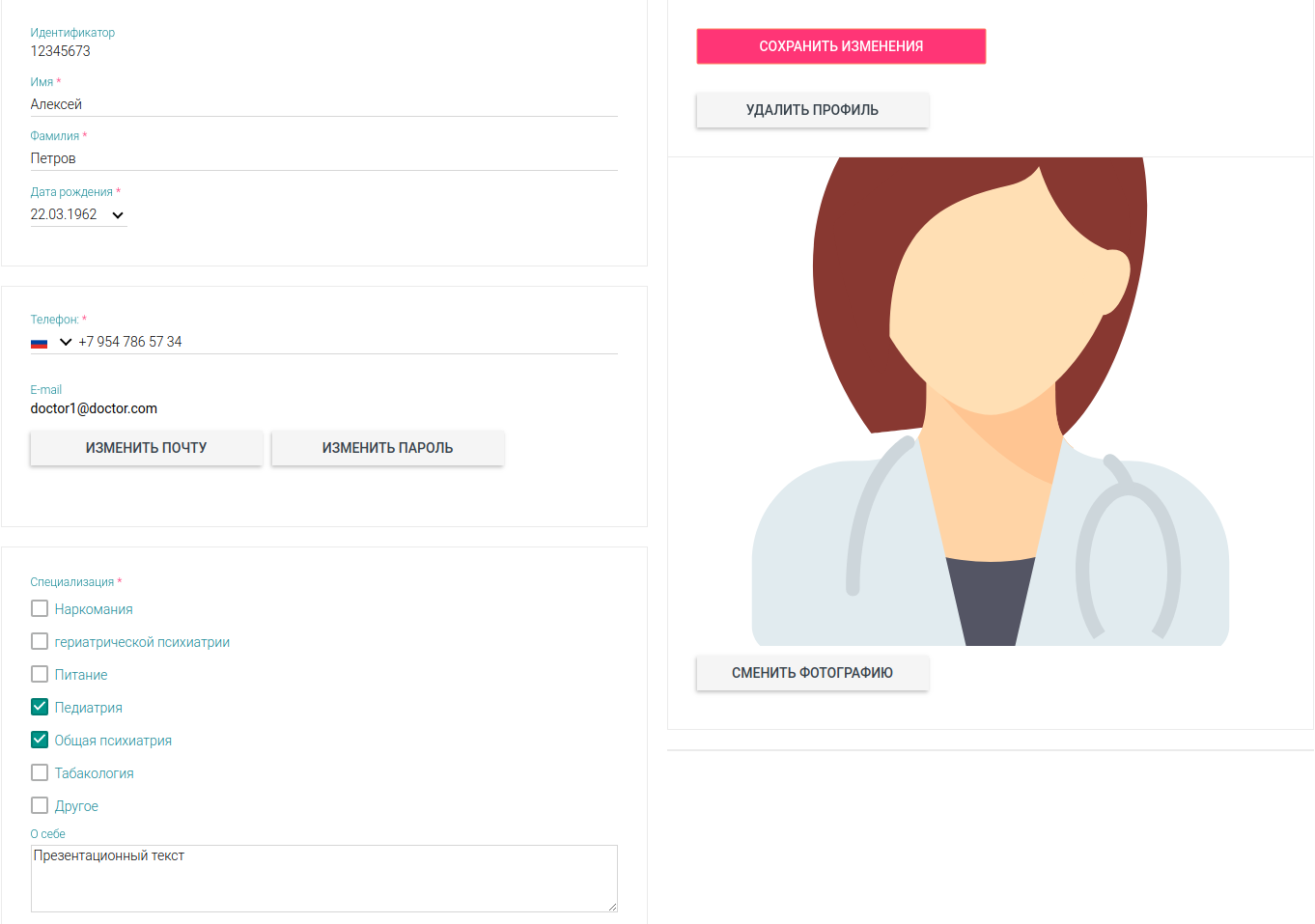


Рисунок 15 - личный профиль врача

## 3.6 Тест на соответствие требованиям видео-консультаций

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

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

В случае несоответствия одному из требований пользователь будет об этом оповещен (рис 16).

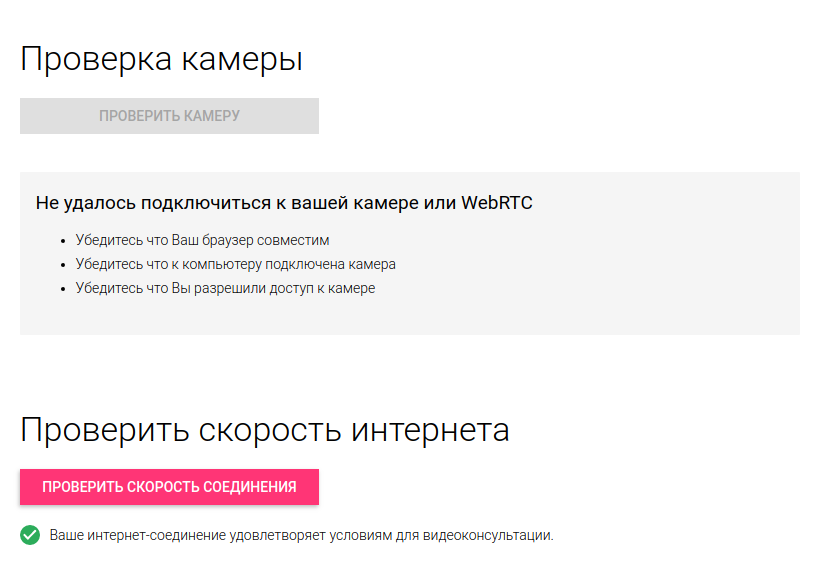


Рисунок 16 - тестирование требований для оказания видео-консультаций

**Итоги**

В данной работе был выполнен функционала 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.

# Архитектура и реализация функционала приложения

Серверная сторона проекта реализована на NodeJS с использованием фреймворка Express. В качестве базы данных будет использоваться 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 планируется использовать стандарт ECMAScript 6, который на данный момент поддерживается очень не многими браузерами. Для решения этой проблемы будет использоваться промежуточный траспилятор Babel. Для сборки клиентской части проекта выбран инструмент Webpack.

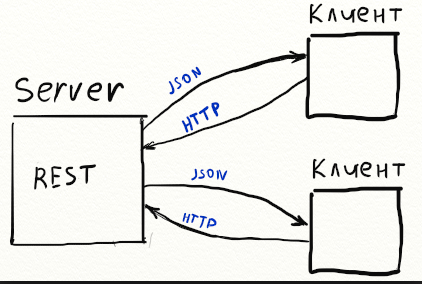
****

Рисунок 17 – RESTFul архитектура

Клиент-серверное взаимодействие будет происходить в формате JSON поверх HTTP протокола. Для этих целей была реализована REST-архитектура (см. 4.4.1).

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.

# 4.1 Сервер

В качестве серверного языка программирования был использован NodeJS. NodeJS - это платформа транслирующая код написанный на Javascript в низкоуровневый машинный код. Для ускорения процесса разработки использовался фреймворк Express - наиболее популярный фреймворк для платформы. Express не является MVC (Model-View-Controller) фреймворком, но для разработки использовалась подобная схема работы за исключением одного - так как серная 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.

## 4.1.1 REST-Архитектура

*REST* — (*Representation State Transfer*) — передача состояние представления. Является архитектурным стилем построенным на основе HTTP протокла. 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.

## 4.1.2 Структура проекта

В начале проекта необходимо качественно определить структуру проекта и разделить зоны ответственности директорий. На старте проекта была проведена такая работа. Для ясности опишем структуру и назначения корневой директорий серверной части проекта.

Таблица 1 – описание корневой директории серверной части приложения

|  |  |
| --- | --- |
| **Название директории** | **Описание** |
| app | Доменная директория |
| bootstrap | Скрипты для запуска сервисов |
| config | Конфигурция |
| routes | Конфигурация url - запросов и соответствующих контроллеров |
| app.js | 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 представления директория *app*, которая содержит бизнес-логику приложения. Ниже в таблице приведено краткое описание директорий *app*.

Таблица 2 – содержимое директории app серверной части приложения

|  |  |
| --- | --- |
| **Директория** | **Описание** |
| controllers | Контроллеры |
| exceptions | Исключения |
| formatters | Шаблоны ответов |
| middlewares | Промежуточные слои |
| models | Модели |
| repositories | Репозитории |
| services | Сервисы |
| utils | Утилиты |
| validators | 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 запросом выполняет какие-либо действия, например запрос, сохранение или изменение в базе данных конкретной модели, а после высылает соответствующий ответ обратно.

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

***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 отличии от *middleware*, только предоставляет методы, а *middleware* их использует.

***Валидаторы*** - отвечают за проверку правильности сохраняемых данных моделей. Исключения - хранилище исключений приложения.

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

## 4.1.3 ORM

В проекте используется ORM – *mongoose* для работы с нереляционной базой даных *mongoDB*. *ORM* – *object relation mapping* – объектно-реляционной отображение. Это слой абстракции для взаимодействия с базой данных на объектном уровне. ORM “соединяет” объектное представление с базой данных, делает соответствие объектных моделей проекта таблицам базы данных для упрощения взаимодействия. Например, в проекте имеется сущность пользователя. Сущность описывается как объект с набором свойств, далее вызвается 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.

## 4.1.4 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 рантайме не имеется. На диаграмме ниже (рис. 18) описан частный случай обработки *http*-запроса и ответа.

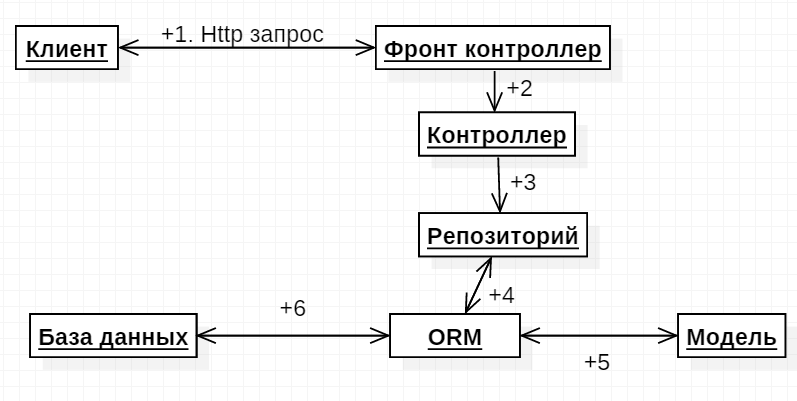


Рисунок 18 – схема запроса-ответа сервера

На диаграмме выше описан частный случай обработки http-запроса и ответа.

1. Клиент посылает http-запрос
2. Фронт контроллер применимает запрос, инициализирует приложение и в соответствии с запросом и таблицей маршрутизации передает управление контроллеру
3. Контроллер обращается к репозиторию для извлечения записи некоторой сущности
4. Репозиторий обращается к системе ORM
5. ORM обращается к соответствующей запросу объектной сущности
6. ORM в соотвтествии делает запрос на основе объектной сущности к базе данных
7. База данных возвращает результат
8. ORM преобразует результат в объектный вид
9. Дальше ответ отдается вверх по цепочке до контроллера, где контроллер может модифицировать результат и отдать его клиенту

## 4.1.5 Диаграмма классов

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

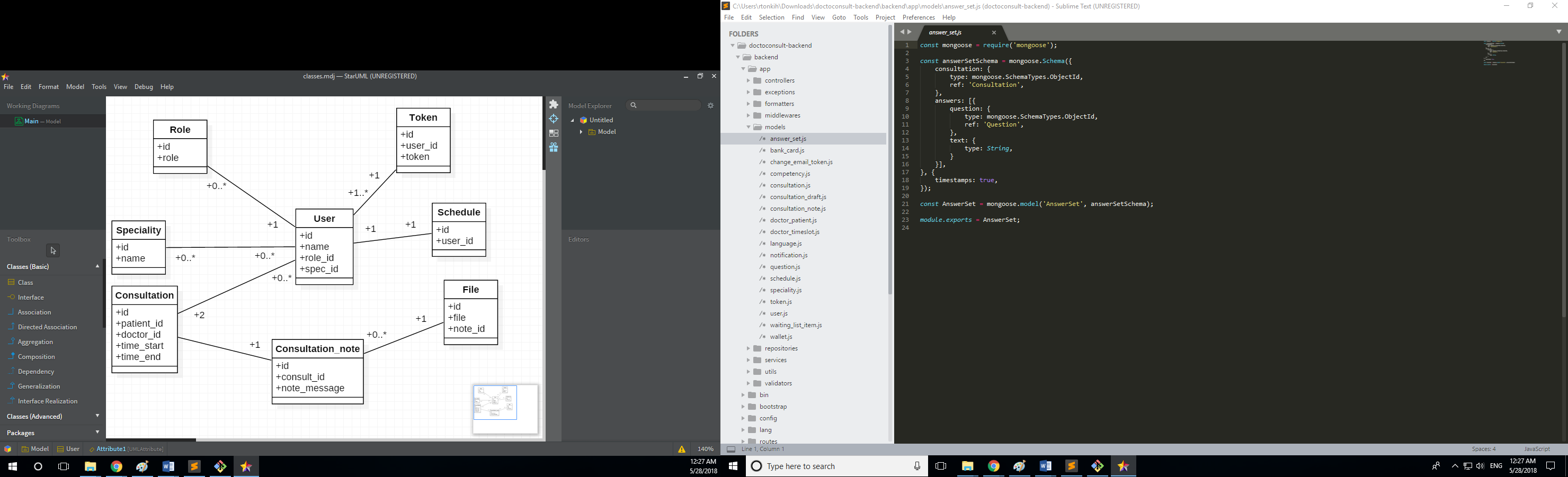


Рисунок 19 – диаграмма классов предметной области

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.

## 4.1.6 Docker

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:

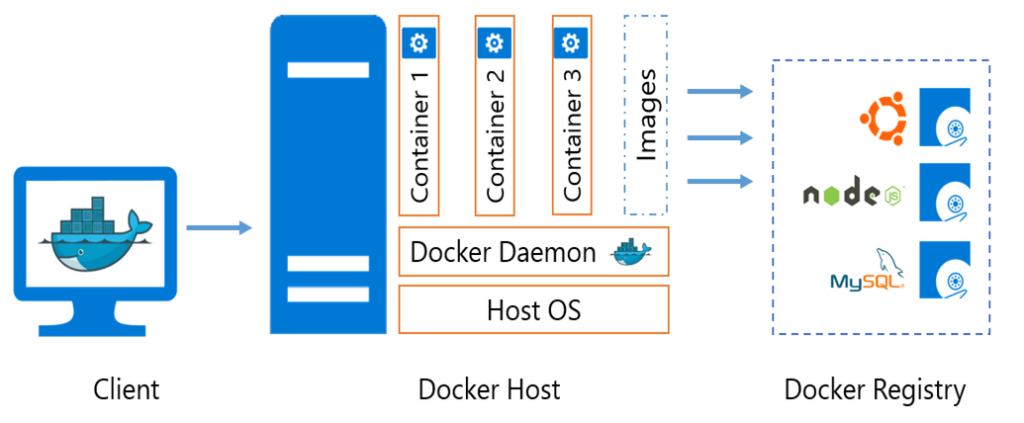
* 1. Быстрое развёртывание приложений
  2. Быстродействие работы за счёт виртуализации на уровне ОС
  3. Платформонезависимость

Рисунок 20 – архитектура Docker

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

## 4.1.7 Итог

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.

# 4.2 Разработка клиентской части

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-С 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.

## 4.2.1 *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 21), которая имеет несколько моделей и представлений. Представления в свою очередь могут быть зависимы от нескольких моделей, и модели также могут взаимодействовать между собой и изменять друг-друга. Модели отправляют данные на 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.

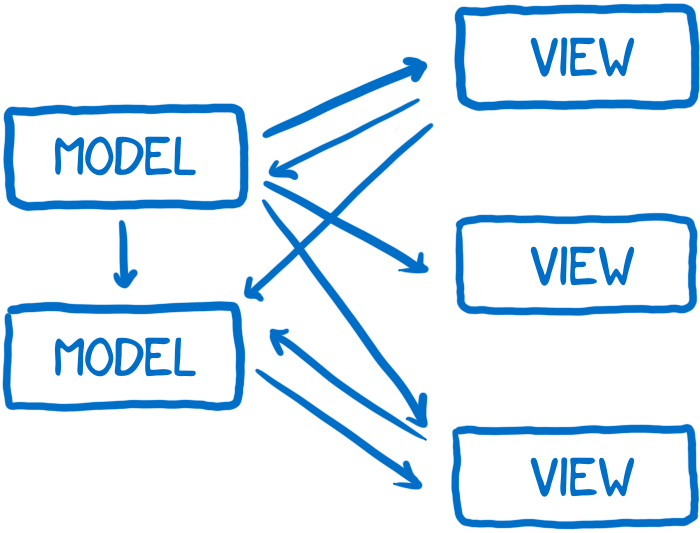
****

Рисунок 21 - путаница во взаимодействии компонентов

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.22 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.

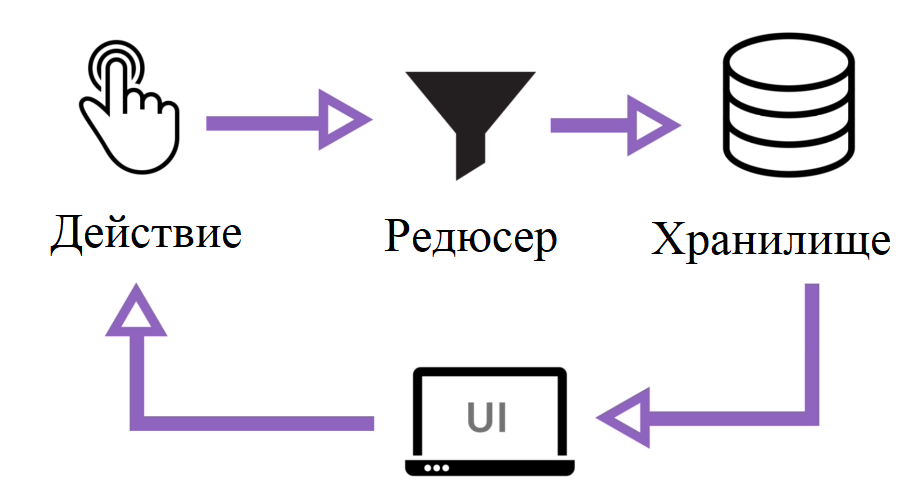


Рисунок 22 – схема взаимодействия компонентов

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 действия изменяет текущее состояние приложения. После изменения состояния, все элементы представления которые были подписаны на хранилище получаются новые данные и в соответствии с ними обновляются.

## 4.2.2 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 структура 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 при разработке 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.

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

Разработка с использованием React строится вокруг компонентов. Компонент - это блок пользовательского интерфейса, выполняющий некоторый функционал и имеющий возможность хранить внутренее состояние. Компонентом может быть выбор цвета продукта или навигационная панель. Компонент внутри содержит некоторую логику и может отрисосывать 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 содержит 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 недолжно быть никакого манипулирования информацией.

Файл маршрутов (*router.js*), отвечат 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. О ней в разделе дальше.

## 4.2.3 Сторонние эффекты

Как было описано ранее и изображено на рис.22 во Flux архитектуре имеются действия, редюсеры, состояние и представления. Ни один из этих компонентов по назначению не подходит для содержания таких функций приложения, как например, запросы к серверу, логирования событий или отображения иконки загрузки во всём приложении. Такие действия называются *side-effects*, далее будем их называть сторонние эффекты.

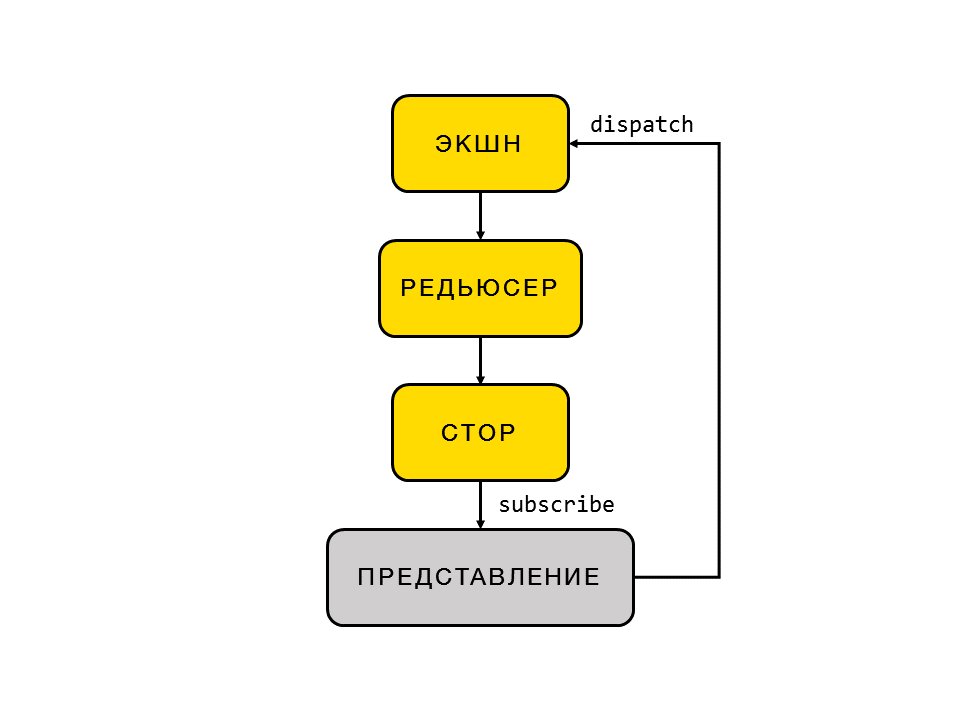


Рисунок 23 - схема без сторонних эффектов

Для организации сторонних эффектов используется библиотека *Saga*. Она “встраивается” в схему, слушает и реагирует на определенные типы действий. В схеме на рис.24 этот момент обозначен под названием “Мидлвейр”.

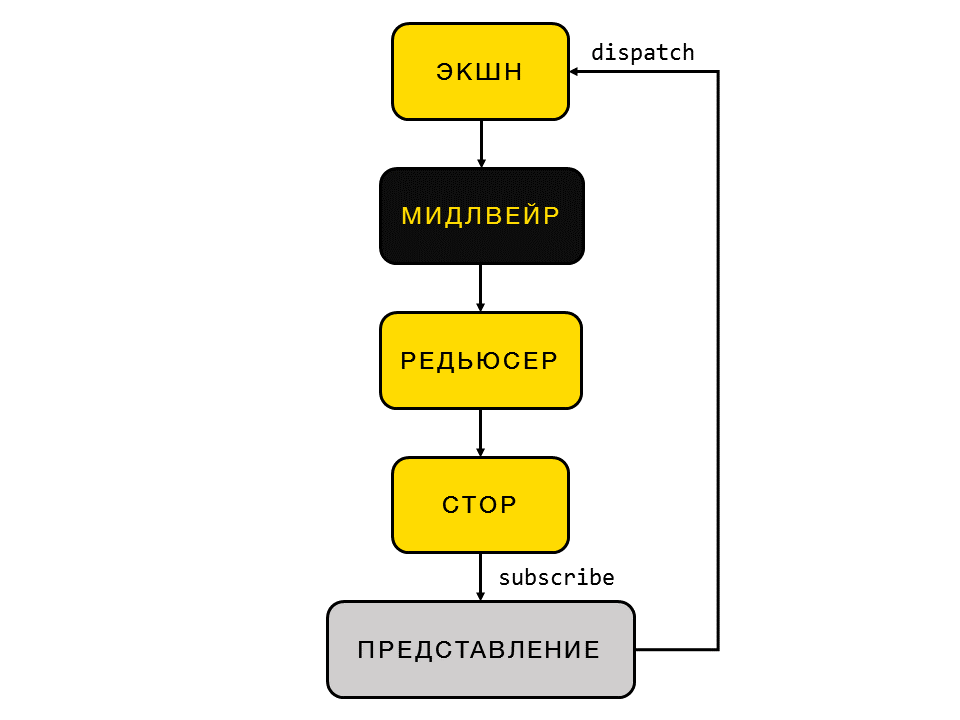


Рисунок 24 – схема со сторонними эффектами

Например, пользователь пытается загрузить страницу со списком врачей. Генерируется событие загрузки, срабатывает сторонний эффект в котором посылается запрос на сервер. Дальше 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 редьюсер изменяя состояние на “успешная загрузка” и добовляя список врачей в общее состояние приложения. Компоненты представления, подписанные на состояние списка врачей, обновляют своё отображение.

# 4.3 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 поддержи браузерами технологии *WebRTC* (табл.1).

Таблица 3- поддержка технологии WebRTC браузерами

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Браузер** | **Поддержка WebRTC** | **Поддержка VP8** | **Поддержка H.264** | **Захват экрана** |
| *Google Chrome* | Да | Да | Да | Да |
| *Mozilla FireFox* | Да | Да | Да | Да |
| *Opera* | Да | Да | Да | Да |
| *Microsoft Edge* | Да | Нет | Да | Нет |
| *Safari* | Да | Нет | Да | Нет |
| *IE* | Нет | Нет | Нет | Нет |

Из таблицы совместимости браузеров (табл. 3) 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:

1. Не требует установки
2. Высокое качество связи
3. Защищенность соединения
4. Гибкость в реализации интерфейса
5. Открытый исходный код
6. Кроссплатформенность

А из недостатков:

1. Отсутствие стандарта сигнализации
2. Отсутствие возможности видеоконференций по умолчанию

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

## 4.3.1 Использование и детали технологии

Каждый клиент (браузер) поддерживающий технологию WebRTC имеет доступ к API (application programming interface) медиа объекта (MediaStream), в который необходимо указать тип и источник медиа данных. В свою очередь медиа поток состоит из каналов (MediaTrack объектов), где нужно cконфигурировать различные виды поток (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*, *audio*), а для синхронизации достаточно воспользоваться свойством *label* и установить его одинаковым для синхронизируемых каналов.

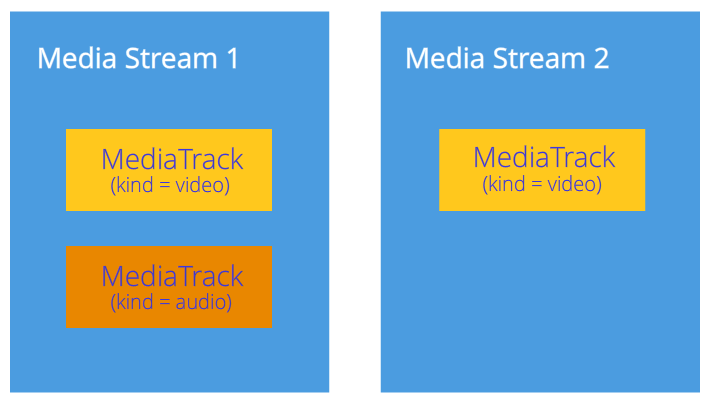


Рисунок 25 – конфигурация медиа-потоков

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)

## 4.3.2 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. 26 клиент 1 отправляет в дескрипторе информацию о поддержке кодеков А 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.

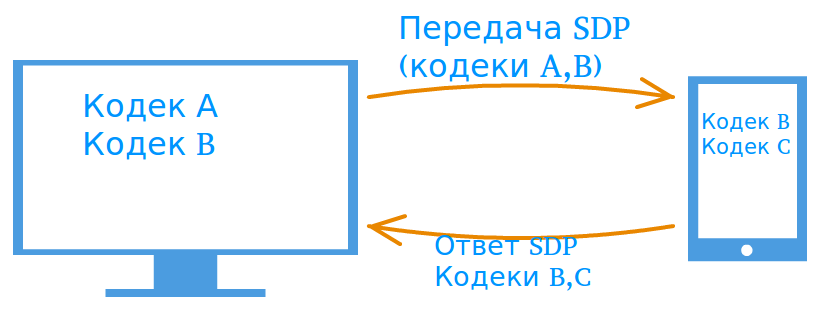


Рисунок 26 – обмен клиентов дескрипторами SDP

## 4.3.3 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). Также как и SDP генерируется в WebRTC и передается сигнальному серверу для дальнейшей отправки второму клиенту. Второй клиент, получив *Ice candidate* токены содержащие информацию о расположении будет знать с кем и по какому адресу устанавливать соединение.

## 4.3.4 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 *WebRTС* 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 клиентами в обход сигнального сервера

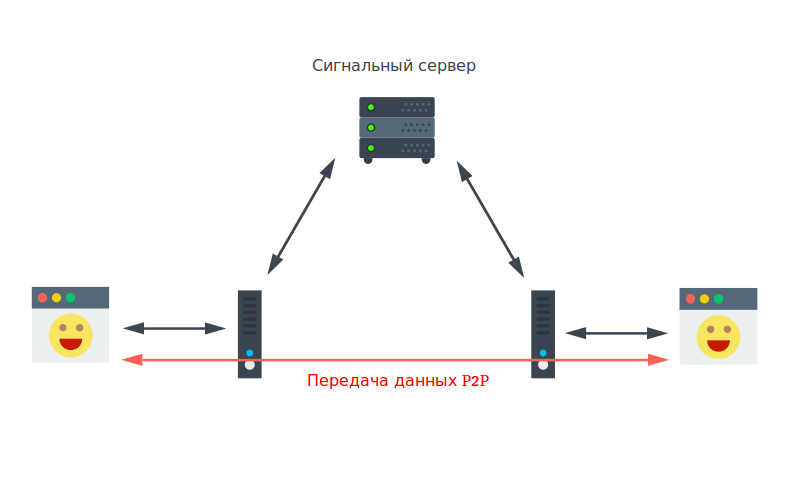


Рисунок 27 – схема работы WebRTC

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.

# 4.4 Технология Вебсокетов

Для списка врачей и отображения их текущих статусов использовались вебсокеты. Вебсокеты позволяют создать соединение в режиме реального времени и даёт возможность обмениваться данными между клиентом и сервером. В отличии от HTTP, вебсокеты могут работать двунаправленно, не разрывая связи между узлами. Ниже приведен пример работы HTTP протокола:

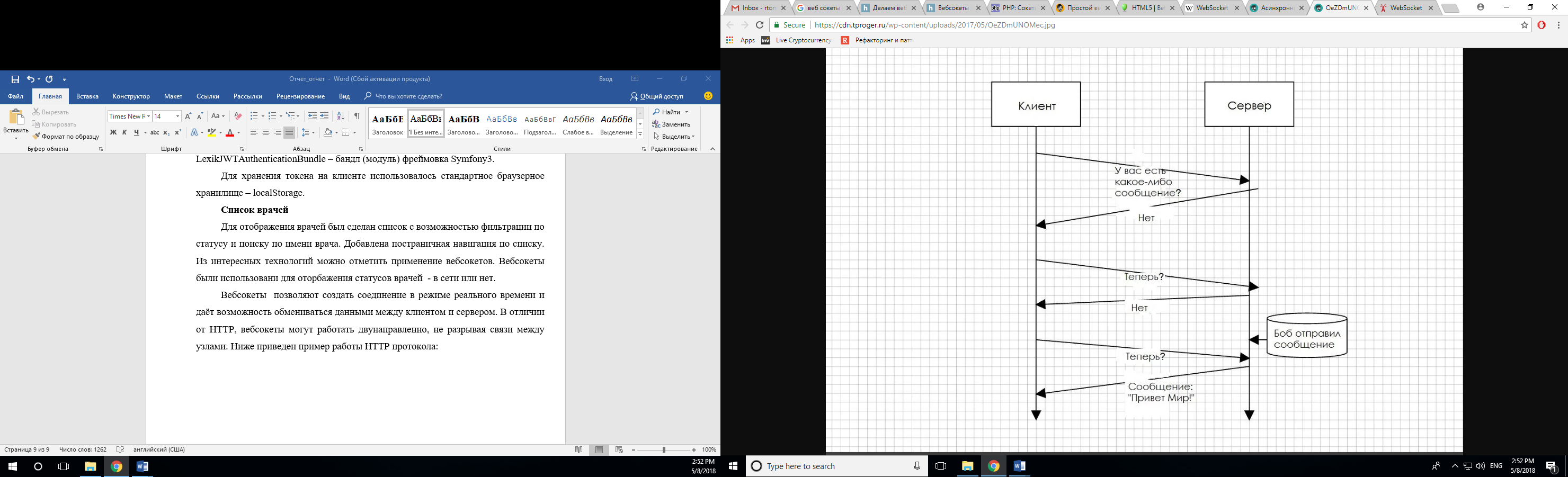


Рисунок 28 – схема работы HTTP-протокола

Как видно из схемы на рис.28 для того, чтобы узнать появилось ли новое сообщение от сервера клиент должен постоянно проверять наличие обновлений. Из этого вытекают две проблемы: либо слишком большая нагрузка на сервер с большим количество обращений от клиента, либо недостаточная интерактивность для клиента.

Вебсокеты, в отличии от HTTP, устанавливают соединение единожды и в момент появления обновлений сервер сигнализирует клиенту об этом и посылает сообщение.

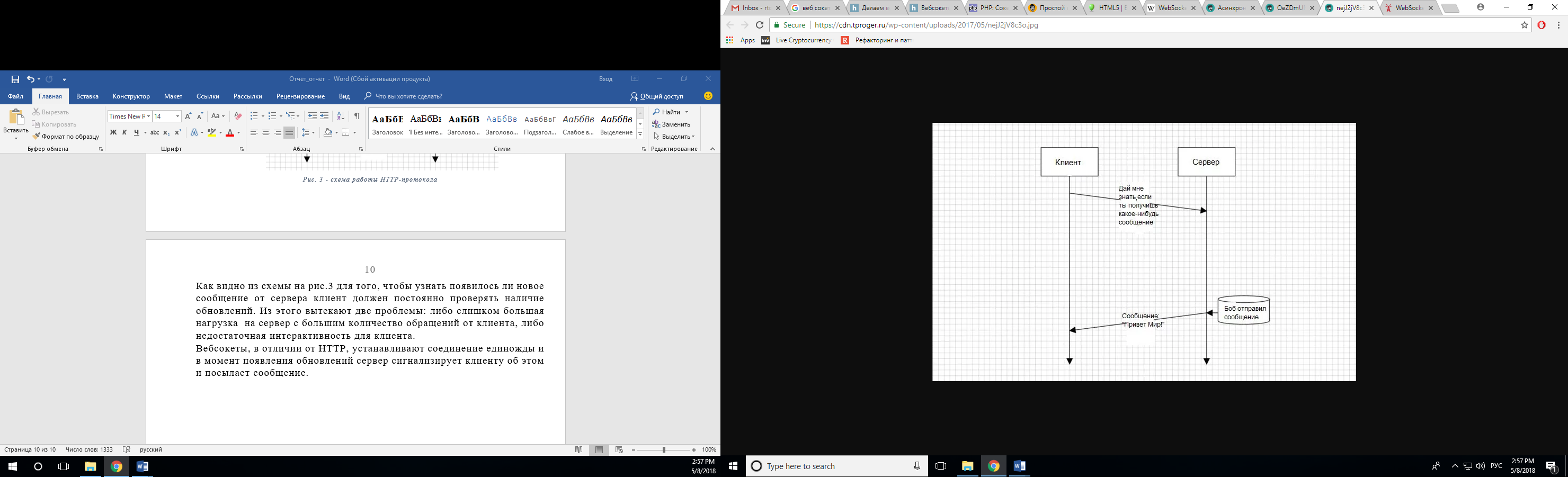


Рисунок 29 – схема работы вебсокета