

## РЕФЕРАТ

ВКР бакалавра «Система для онлайн-консультирования с врачами. Разработка клиентской части».

Пояснительная записка: 96 страниц, 55 рисунков, 30 таблиц, 9 источников, 2 приложения.

Ключевые слова: онлайн-консультации с врачами, web-приложение, JavaScript, информационная система (далее ИС), программный продукт (далее ПП).

Объектом исследования в данной работе являются системы для онлайн-консультирования с врачами.

Цель работы – разработка клиентской части web-приложения для поиска врача и проведения онлайн-консультации.

Web-приложение предназначено для пациентов желающих получить медицинскую консультацию, а также для врачей, осуществляющих эту консультацию за оплату.

Для пациента приложение позволяет удобно выполнять поиск врача по заданным фильтрам, записываться на консультацию с выбором удобного для пациента времени, вести медицинскую карту, хранить анализы и снимки. Для врача сервис предоставляет выбирать удобный рабочий график и проводить онлайн-консультации за назначенную врачом оплату.

В системе предусмотрены мероприятия защиты персональных данных и разграничения доступа. Приложение имеет интуитивно понятный дружественный интерфейс.

Пояснительная записка состоит из введения, 3 разделов и заключения, в которых проводится анализ информационных процессов в задаче, а именно, исследование истории вопроса и его состояния на сегодняшний день, рассмотрены существующие на сегодняшний день аналоги, а также выполнена разработка получившегося программного модуля.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, СУЩЕСТВУЮЩИХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, СИСТЕМ ИЛИ МЕТОДОВ И АЛГОРИТМОВ, КОТОРЫЕ РЕШАЮТ АНАЛОГИЧНЫЕ ЗАДАЧИ .....	6
1.1 Анализ предметной области и её информационные характеристики .....	6
1.2 Обзор существующих аналогов .....	11
1.3 Обоснование выбора инструментальных средств .....	15
Выводы по разделу 1 .....	17
2 СИСТЕМОТЕХНИЧЕСКИЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ КЛИЕНТСКОЙ ЧАСТИ ДЛЯ СИСТЕМЫ ОНЛАЙН-КОНСУЛЬТИРОВАНИЯ С ВРАЧАМИ .....	18
2.1 Проектирование архитектуры .....	18
2.2 Моделирование бизнес-процессов .....	22
2.3 Проектирование сценариев пользователя .....	23
Выводы по разделу 2 .....	40
3 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ДЛЯ СИСТЕМЫ ОНЛАЙН- КОНСУЛЬТИРОВАНИЯ С ВРАЧАМИ .....	41
3.1 Разработка компонентов программного модуля .....	41
3.2 Разработка интерфейса программного модуля .....	46
Выводы по разделу 3 .....	68
ЗАКЛЮЧЕНИЕ .....	69
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ .....	71
ПРИЛОЖЕНИЕ А. Моделирование бизнес-процессов с помощью BPMN ...	72
ПРИЛОЖЕНИЮ Б. Код программы .....	73

## ВВЕДЕНИЕ

**Актуальность темы.** В связи с нынешней эпидемиологической ситуацией в мире остро стоит проблема записи на прием к врачу и получения как минимум первичной медицинской консультации, по причине того, что государственные больницы не справляются с большим количеством пациентов. Кроме того, посещение поликлиник увеличивает количество потенциально опасных контактов. Так же часто встречаются ситуации, связанные с ограничениями по месту жительства и отсутствия возможности своевременно получить консультацию специалиста.

Существует большое количество готовых программных продуктов, однако среди них отсутствуют системы где врачи имеют возможность зарегистрироваться, пройти верификацию и предоставлять услуги консультирования пациентам.

Целью выпускной квалификационной работы является разработка web-приложения для онлайн-консультирования с врачами. В процессе достижения поставленной цели решались следующие задачи:

- 1) проанализировать функциональные обязанности врача, пациента и типовых вариантов реализации web-приложений;
- 2) разработать алгоритмы функционирования приложения, выбор платформы и языка программирования для создания web-приложения;
- 3) разработать структурную схему приложения, дизайн и реализовать его в виде web-приложения.

**Объектом исследования** являются процессы оказания услуг телемедицины.

**Предметом исследования** настоящей работы является web-приложение для проведения онлайн-консультаций.

**Практическое значение работы.** Результат данной работы будет представлять интерес для врачей, которые по каким-либо причинам не могут

вести прием очно, а также для людей, которые не имеют возможности лично присутствовать на консультации у врача.

**Научная и практическая новизна.** Разработано оригинальное web-приложение, отличающееся широкой функциональностью, информационной безопасностью и простотой использования.

**Структура работы.** Данная работа состоит из пояснительной записки, включающей в себя введение, три раздела, выводы, список использованных источников и приложения.

Во введении приведено обоснование актуальности решаемой задачи и сформирована цель и задачи работы.

В первом разделе выполнен анализ информационных процессов в задаче разработки web-приложения для онлайн-консультирования с врачами. Проведен сравнительный анализ существующих web-приложений, в которых решаются схожие задачи.

Во втором разделе разработаны алгоритмы функционирования приложения, выбрана платформа и язык программирования для создания web-приложения, выполнено проектирование архитектуры и сценариев пользовательского интерфейса.

В третьем разделе разработана структурная схема приложения, пользовательский интерфейс, реализованы модули web-приложения, выполнено тестирование доступности интерфейса.

В заключении сделаны выводы во всей работе.

# **1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, СУЩЕСТВУЮЩИХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, СИСТЕМ ИЛИ МЕТОДОВ И АЛГОРИТМОВ, КОТОРЫЕ РЕШАЮТ АНАЛОГИЧНЫЕ ЗАДАЧИ**

## **1.1 Анализ предметной области и её информационные характеристики**

Предметной областью данной дипломной работы является – сервис для онлайн консультирования с врачами.

Основной целью данной области выступает получение консультации, а в следствии и рекомендаций по лечению, пациентом от специалиста. Как правило в данные сервисы входят возможности по поиску врача нужной специальности, записи на консультацию, получение консультации через чат, голосовой чат, видеосвязь. На главной странице сайта отображаются отзывы о сервисе, список врачей с наилучшим рейтингом, форма записи на консультацию, список цен.

Разрабатываемое web-приложение позволит пользователям, которые желают получить рекомендации от грамотного специалиста, осуществить консультацию не выходя из дома. Для специалиста, в свою очередь, приложение позволит устроиться на работу, выставляя удобный для себя график работы.

С помощью сервиса пользователи смогут удобно выбрать подходящего под их проблему врача, записаться к нему на прием, выбрав удобное для себя время и, оплатив прием, перейти к консультации посредством видеосвязи, аудиозвонка или сообщениями в чате.

Специалист сможет пройти верификацию по личным данным (диплом, паспорт), после чего настроить график работы, выбрать стоимость консультации и заполнить информацию о себе. Во время консультации врач сможет удобно просматривать медицинские карты своих пациентов, смотреть их анализы, рост, вес, вредные привычки и оставлять рекомендации, получая за это оплату.

В системе онлайн консультаций с врачами основным процессом является регистрация врачей/пациентов, поиск врачей, запись на консультацию и ее последующее проведение. Для своей работы система использует внешние сущности: врач и пациент.

При регистрации пользователя как пациента происходит процесс верификации по почте и заполнение медицинской карты в личном кабинете. После перечисленных выше процессов, пациент находит врача с помощью фильтров или поиска, далее происходит процесс записи на прием, во время которого пациент выбирает доступное время консультации, заполняет список своих симптомов, после чего, оплачивая услугу, он переходит к процессу консультации. После прохождения консультации пациент получает результат в виде рекомендаций. После консультации пользователь может оставить отзыв о консультирующем его специалисте.

При регистрации врача происходит проверка подлинности документов и квалификации врача модератором, если ответ положительный, то врач может перейти к процессу консультации с пациентами, которые записались к нему на прием.

На рисунке 1.1 представлена DFD-диаграмма основного процесса.

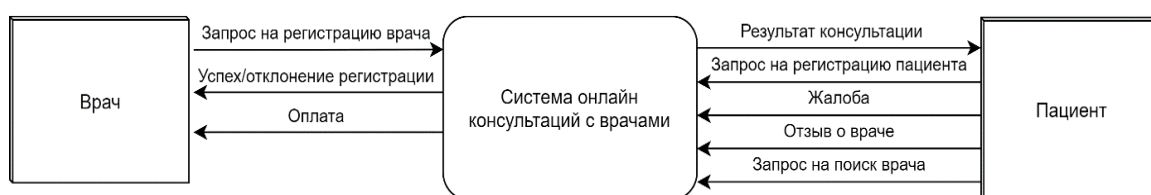


Рисунок 1.1 – DFD-диаграмма основного процесса

Основной процесс включает в себя следующие процессы: регистрация, запись на консультацию, поиск врача пациентом, консультация, составление отзыва после консультации. На рисунках 1.2-1.4 представлены DFD-диаграммы полученные в ходе декомпозиции основного процесса.

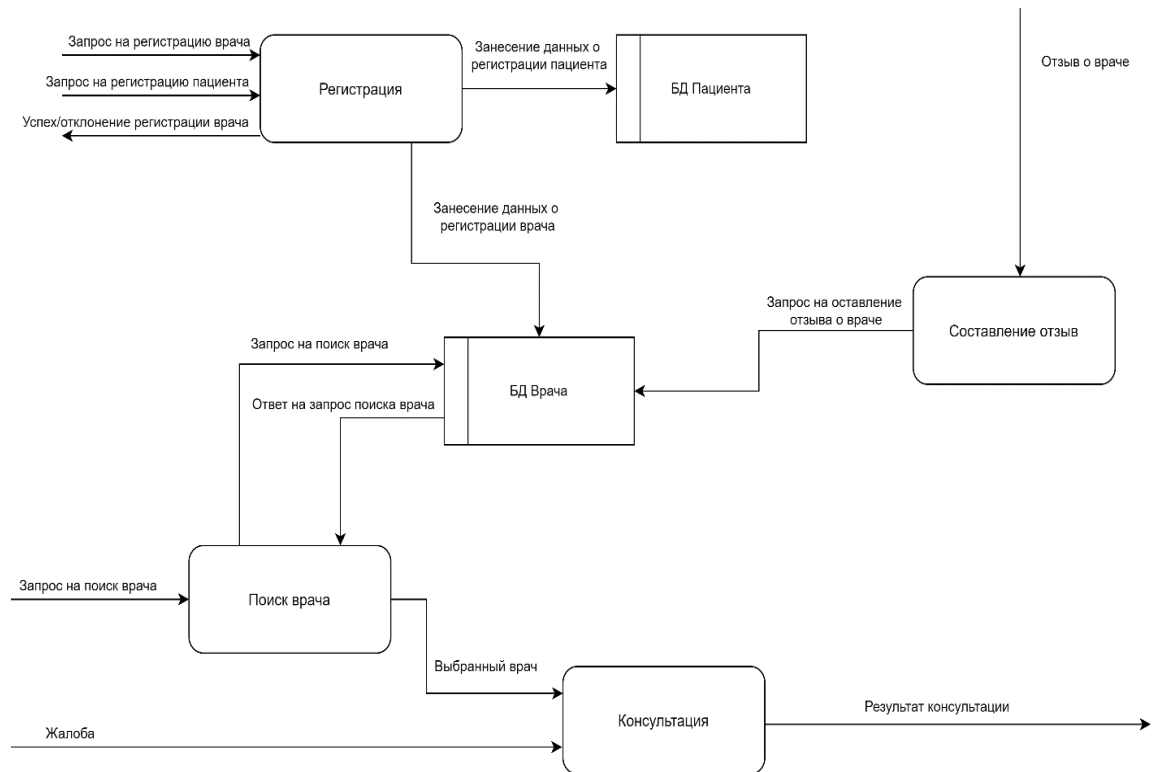


Рисунок 1.2 – DFD-диаграмма декомпозиции основного процесса

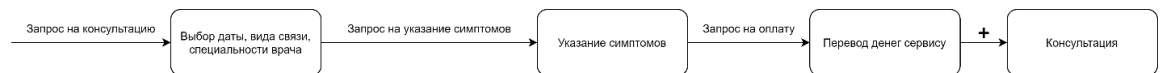


Рисунок 1.3 – DFD-диаграмма декомпозиции процесса консультации

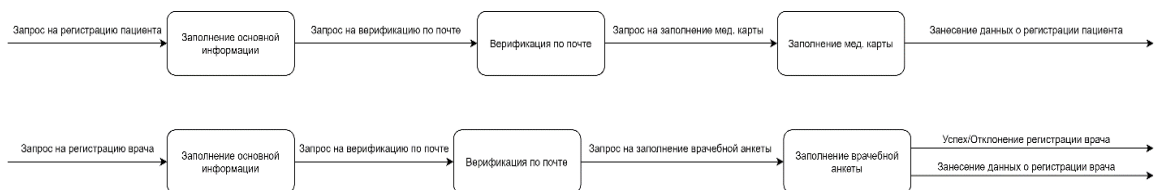


Рисунок 1.4 – DFD-диаграмма декомпозиции процесса регистрации

Любой сервис по предоставлению услуг формирует определенный процесс. Для моделирования данного процесса применим нотацию IDEF0.

Основным элементом модели IDEF0, как показано на рисунке 1.5, является блок, содержащий фразу глагола, описывающую действие или преобразование, которое происходит внутри блока.

В синтаксисе IDEF0 входные данные показаны стрелками, входящими с левой стороны поля, а выходные данные представлены стрелками, выходящими

с правой стороны блока. Элементы управления отображаются в виде стрелок, входящих в верхнюю часть окна, а механизмы отображаются в нижней части.

IDEF0-диаграмма основного процесса проектируемой системы представлена на рисунке 1.5, а также табличный вид диаграммы в таблице 1.1.

На вход бизнес-процесса поступают запросы на регистрацию и авторизацию, которые затем преобразуются в результаты консультации для пациентов, результаты регистрации или выплату врачам. В процессе предоставления услуги участвует врач, пациент и модератор. Бизнес-процесс происходит в рамках существующего прайс-листа и форм регистраций и всевозможных анкет.

Таблица 1.1 – Процессы диаграммы A1

Шифр	Название процесса	Входные данные	Управляющие данные	Механизм	Результат процесса
A1	Предоставить услуги консультации	Запрос на регистрацию врача и пациента,	Прайс лист, установленное время, форма регистрации и тестирования	Врач, пациент, модератор	Результат консультации, оплата, отклонение оплаты, отклонение регистрации врача

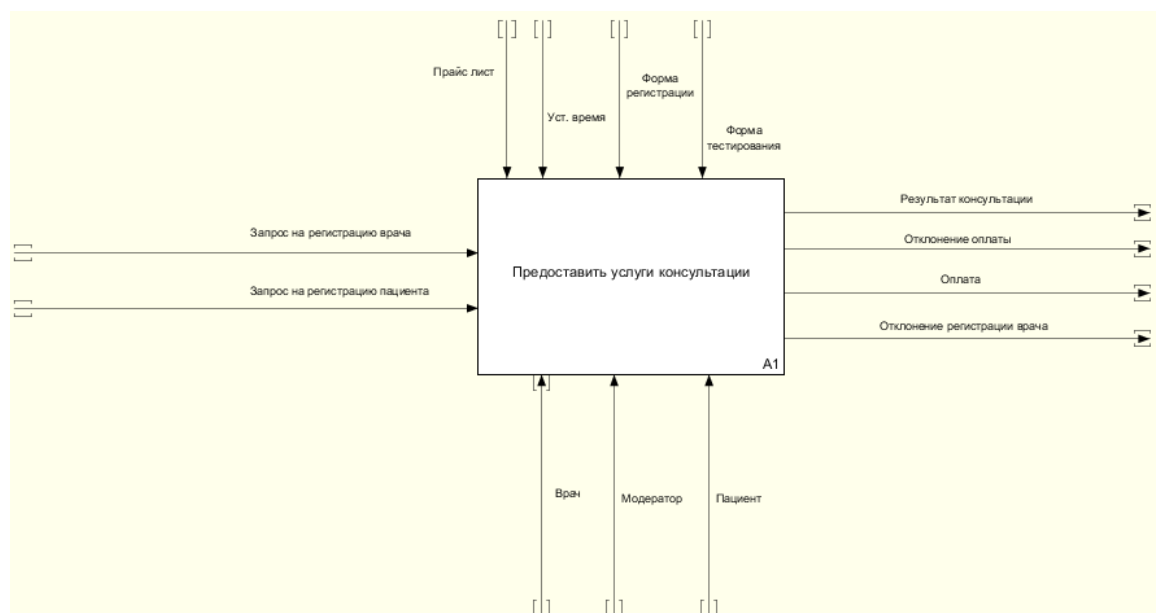


Рисунок 1.5 – IDEF0-диаграмма основного процесса



На рисунке 1.6 изображена IDEF0-диаграмма декомпозиция основного процесса “Предоставить услуги консультации”, а в таблице 1.2 ее табличные значения.

Таблица 1.2 – Декомпозиция процесса A1

Шифр	Название процесса	Входные данные	Управляющие данные	Механизм	Результат процесса
A11	Регистрация врача	Запрос на регистрацию врача	Форма регистрации	Врач, модератор	Отклонение регистрации врача
A12	Регистрация пациента	Запрос на регистрацию пациента	Форма регистрации	Пациент, модератор	Жалоба
A13	Поиск врача	Жалоба	Форма тестирования	Пациент, врач	Выбранный врач
A14	Оплачивать	Выбранный врач	Прайс лист	Пациент	Чек об оплате
A15	Консультация	Чек об оплате	Установленное время	Врач, пациент	Результат консультации
A16	Составить отзыв	Результат консультации	Результат консультации	Пациент	Отклонение оплаты
A17	Перевод врачу	Положительный отзыв	Положительный отзыв	Модератор	Оплата

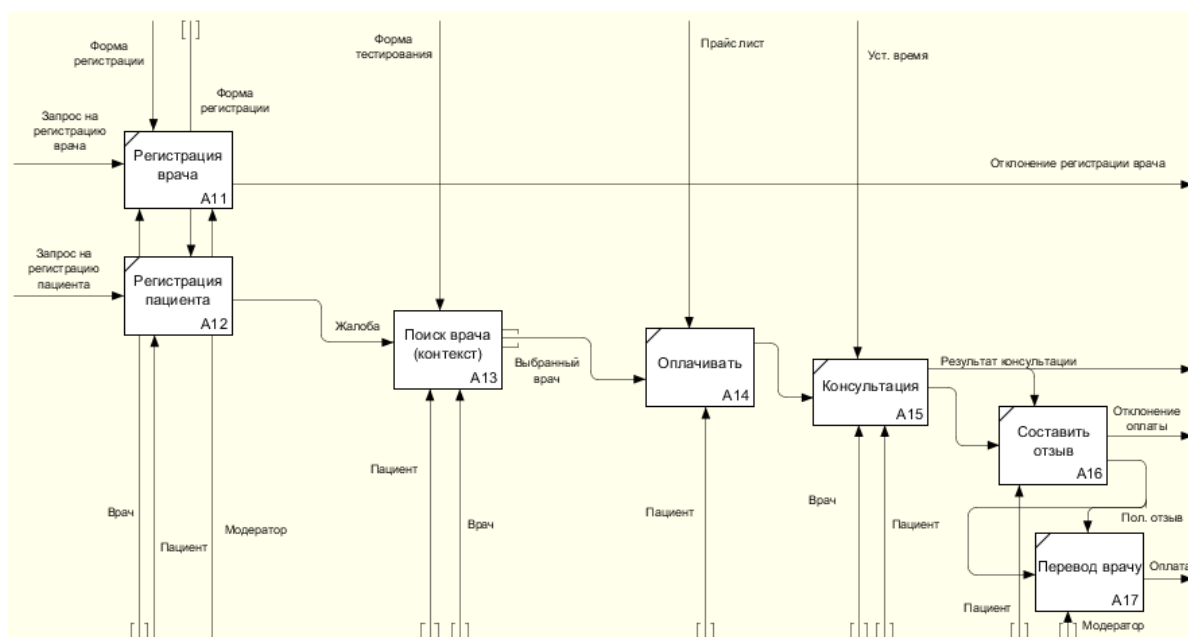


Рисунок 1.6 – IDEF0-диаграмма декомпозиции основного процесса

## 1.2 Обзор существующих аналогов

В данном разделе были выбраны и рассмотрены 3 самые популярные российские аналоги сервисов телемедицины, рассмотрены их преимущества и недостатки, которых лишен наш сервис. Поиск данных сервисов осуществлялся поисковой системы по запросам «Онлайн-консультации с врачами» и «Телемедицина».

### 1.2.1 Медведь.Телемед

Медведь.Телемед (<https://telemed.chat/>) – это онлайн-сервис для получения медицинских консультаций в виде диагноза. Данный сервис имеет довольно устаревший UI, но с хорошим UX. На главной странице отображается информация о сервисе, такая как:

- блок выбора специалиста;
- принцип работы онлайн-консультаций;
- список цен на консультации;
- отзывы;
- о нас;
- секция «Заказать обратный звонок».

Последний элемент списка очень интересен, так как пользователю предоставляется возможность проконсультироваться со специалистом общего профиля и получить совет к какому врачу лучше идти на прием.

На странице «Врачи» при выборе специалиста отсутствует его фотография, а также не указано время его работы (рисунок 1.1).

Данный недостаток является довольно весомым т.к. пользователю будет сложнее доверять своему будущему врачу, не видя даже его лица.

Главная > Врачи

Выберите специалиста  
 974 врача. Выбрать

Выберите клинику  
 Выберите клинику

ФИО нужного врача  
 Например, Иванов Иван

Сортировать: По стоимости По стажу По рейтингу Найдено: 1835 врачей

Стоматолог  
**Аверьянова Елена Владимировна**  
 ★★★★★ Стоматолог-терапевт

Урсула, стоматологическая клиника  
 Екатеринбург, Кировоградская, 4, 2

Стоматолог  
**Баранец Марина Валерьевна**  
 ★★★★★ Стоматолог-терапевт

Dr. Baranets, стоматологическая клиника  
 Екатеринбург, 8 Марта, 194

Рисунок 1.1 – Список врачей сервиса Медведь.Телемед

Следующим недостатком системы является привязка аккаунта Медведь.Телемед к portalу Госуслуги. Пользователю необходимо иметь аккаунт в данном portalе для возможности пользоваться сервисом.

На сайте установлен список фиксированных цен на консультации. Данный момент является как преимуществом, так и недостатком. В случае если сервис имеет фиксированную стоимость консультаций для всех врачей, то может произойти такое что врач с высоким рейтингом будет иметь полностью заполненный график работы. В то время как врач, который только пришел на сервис и не имеет рейтинга будет иметь довольно маленькое количество пациентов или не иметь их вообще. С обратной же стороны если цены на свои приемы будут устанавливать сами врачи, то доктора с большим рейтингом будут стоить выше чем новопришедшие. В таком случае, людям, которые не обладают средствами для консультации со специалистом, обладающим большим рейтингом, будут вынуждены записываться к врачам с более низким рейтингом.

Главным недостатком всей системы является необходимость проведения сперва очной консультации. На ней врач уже назначит дату и время онлайн-

консультации. При таком подходе теряются многие преимущества телемедицины, но зато врач имеет право поставить полноценный диагноз пациенту.

### 1.2.2 Яндекс.Здоровье

На сайте (<https://health.yandex.ru/consultation/>) доступен выбор специалиста по категориям, разделенным на две категории: взрослые и дети. После выбора категории врача нам предлагают выбор, состоящих из двух режимов. Первый режим предназначен для обстоятельного ответа с продолжительностью консультации 10 минут. Второй режим длится всего 5 минут, во время которых специалист даст четкий отчет на один вопрос. После выбора режима сервис сразу требует оплату, не позволяя клиенту заранее выбрать врача, к которому он желает обратиться, и удобное для себя время консультации. Также стоит отметить очень короткий список доступных врачей на выбор, который состоит из всего 3 специалистов (рисунок 1.2).

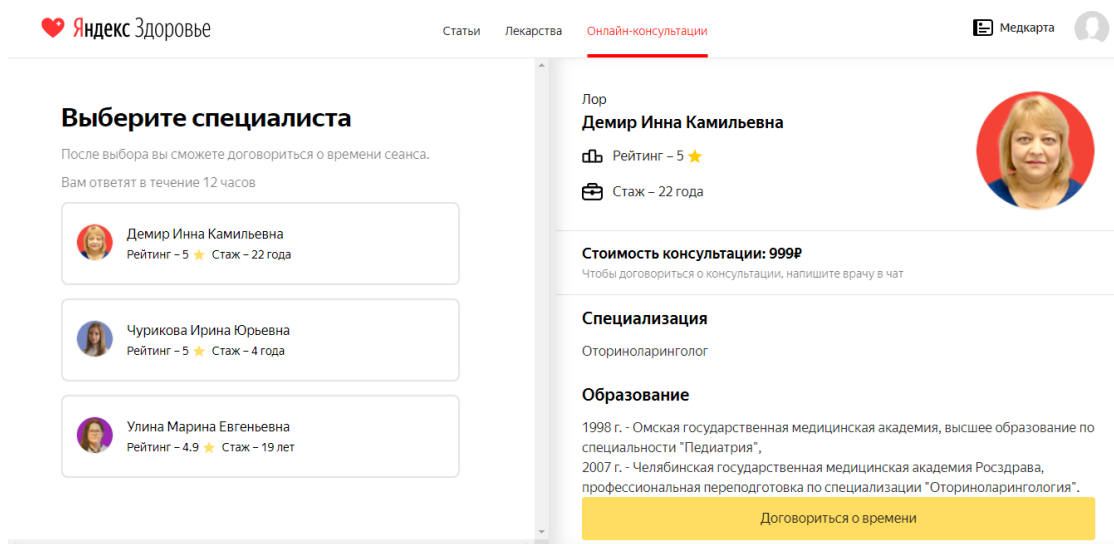


Рисунок 1.2 – Список врачей сервиса Яндекс.Здоровье

Перейдя в медицинскую карту на сервисе можно увидеть довольно скудный функционал. Здесь отсутствует возможность внести свои личные медицинские данные, добавить фото анализов или справок.

Кроме того, сайт не предоставляет возможность специалисту самостоятельно устроиться врачом в данную систему, пройдя верификацию по навыкам, диплому и остальным личным данным.

### **1.2.3 Онлайн доктор**

В отличие от прошлого аналога, данный сайт (<https://onlinedoctor.ru/doctors/>) имеет удобную фильтрацию специалистов по категории, ФИО врача, времени работы и их направленности (взрослые или дети). Также плюсом стоит отметить возможность “просто спросить”, которая позволяет оставить короткий вопрос доктору и получить на него ответ до определенной даты. Но данная дата может варьироваться от 2 до 3 и более дней, что является небольшим минусом.

Следующим минусом данного сервиса также является отсутствие медицинской карты. Данный исход является минусом, так как при каждом обращении к врачу, пациенту приходится заново рассказывать о себе, своих болезнях, медицинских показателях и так далее, а имея сразу медицинскую карту пациента перед собой, врач видит полную ситуацию и может дать более точные рекомендации.

Сама консультация на сайте длится всего 30 минут и с учетом описанных выше минусов, этого может не хватить специалисту для написания рекомендаций.

В качестве плюса стоит отметить удобный и понятный дизайн сайта, на главной странице сразу виден список специалистов, откуда можно за пару кликов записаться на прием.

Данный сайт также наследует основной минус прошлого аналога (Яндекс.Здоровье) – отсутствие возможности устроиться на работу специалисту самостоятельно.

### 1.3 Обоснование выбора инструментальных средств

В ходе работы на стороне клиента было решено использовать язык TypeScript и библиотеку для разработки пользовательских интерфейсов React. Для управления состоянием приложения была выбрана библиотека MobX. В качестве CSS препроцессора был взят SASS. Для организации соединения в режиме реального времени была использована библиотека Socket.IO.

TypeScript – это язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript. TypeScript отличается от JavaScript возможностью явного статического назначения типов, поддержкой использования полноценных классов, а также поддержкой подключения модулей, что призвано повысить скорость разработки, облегчить читаемость, рефакторинг и повторное использование кода, помочь осуществлять поиск ошибок на этапе разработки и компиляции, и, возможно, ускорить выполнение программ.

React – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. Из особенностей можно выделить следующее:

- Virtual DOM - легковесная копия DOM дерева, в которую вносятся изменения, после чего происходит сравнение DOM дерева с его виртуальной копией, определяется разница и происходит перерисовка того, что было изменено;
- JSX – расширение синтаксиса JavaScript, которое позволяет использовать HTML синтаксис для описания структуры интерфейса;
- методы жизненного цикла, при помощи которых разработчик может описывать поведение компонента на каждом этапе его жизни (при монтировании, обновлении данных, либо его удалении).

MobX – это автономная библиотека, для управления фронтенд-состоянием приложения. MobX обеспечивает консистентность и согласованность внутреннего состояния фронтенд-приложения, предоставляя удобные инструменты для его изменения.

MobX имеет следующие преимущества по сравнению с его аналогом Redux:

- эффективен сразу после установки, отсутствие “Многословности” по сравнению с Redux;
- объектно-ориентированный подход;
- упрощение работы с асинхронными действиями;
- производительность и скорость разработки.

SASS (Syntactically Awesome Stylesheets) — это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS-кода и упрощения файлов каскадных таблиц стилей.

Язык SASS имеет два синтаксиса:

- SASS – отличается отсутствием фигурных скобок, в нём вложенные элементы реализованы с помощью отступов;
- SCSS (Sassy CSS) – использует фигурные скобки, как и сам CSS.

Socket.IO – это библиотека JavaScript для веб-приложений реального времени. Он обеспечивает двустороннюю связь в реальном времени между веб-клиентами и серверами. Он состоит из двух частей: клиентской библиотеки, которая запускается в браузере, и серверной библиотеки для node.js. Оба компонента имеют идентичный API.

Написание приложения для реального времени с использованием популярных стеков веб-приложений, таких как LAMP (PHP), традиционно было очень трудным. Он включает в себя опрос сервера на наличие изменений, отслеживание временных меток, и это намного медленнее, чем должно быть.

Сокеты традиционно были решением, вокруг которого строится большинство систем реального времени, обеспечивая двунаправленный канал

связи между клиентом и сервером. Это означает, что сервер может отправлять сообщения клиентам. Всякий раз, когда происходит событие, идея заключается в том, что сервер получит его и отправит заинтересованным подключенным клиентам.

Socket.IO довольно популярен, его используют Microsoft Office, Yammer, Zendesk, Trello и многие другие организации для создания надежных систем реального времени. Это одна из самых мощных JavaScript-фреймворков на GitHub и наиболее зависимая от модуля NPM (Node Package Manager). Socket.IO также имеет огромное сообщество, что означает, что найти помощь довольно легко.

### **Выводы по разделу 1**

В данном разделе был проведен предметный анализ сервиса для онлайн-консультаций с врачами, во время которого были составлены DFD и IDEF0-диаграммы, и проанализированы основные аналоги разрабатываемого сервиса, выделены их преимущества и недостатки.

Были описаны технологии, применение которых целесообразно в разрабатываемой информационной системе. Путем анализа преимуществ и недостатков различных фронтенд библиотек для построения web-приложений, оптимальным выбором технологий стал выбор создания реактивного приложения при помощи библиотеки React JS, а в качестве менеджера управления состоянием приложения MobX. Для обеспечения двусторонней связи в реальном времени между веб-клиентами и серверами выбрана библиотека Socket.IO из-за ее возможности отправки сообщения всем подключенным клиентам и возможностью автоматического переподключения при разрыве соединения, что очень удобно при создании чата.



## 2 СИСТЕМОТЕХНИЧЕСКИЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ КЛИЕНТСКОЙ ЧАСТИ ДЛЯ СИСТЕМЫ ОНЛАЙН- КОНСУЛЬТИРОВАНИЯ С ВРАЧАМИ

### 2.1 Проектирование архитектуры

Основываясь на лучших практиках разработки реактивных приложений с стеком технологий React + MobX в качестве архитектуры приложения был выбран подход Atomic Design.

Данный подход представляет из себя атомарный подход к проектированию компонентов, который включает в себя следующие группы:

- Атомы – самые мелкие компоненты приложения, которые отвечают за какую-то маленькую и конкретную функцию. Например, кнопка, текстовое поле, индикатор загрузки, иконка и так далее.
- Молекулы – это компонент, включающий в себя несколько атомарных компонентов. В данную категорию могут входить такие компоненты, как модальное окно, хлебные крошки, кнопка с иконкой и т.п.
- Организмы – компоненты, отвечающие за функциональность приложения. Это могут быть следующие компоненты: формы, таблицы, карточки.

Исходя из описанного выше подхода предполагается следующая структура приложения:

```
api/  
  ChatApi.ts  
  ...  
components/  
  Message.tsx  
  ...  
fonts/  
icons/  
pages/  
  Chat/  
    components/  
    ChatPage.tsx  
  ...  
stores/
```

```

    ChatStore.ts
    RootStore.ts
    ...
  styles/
  utils/
  App.tsx
  index.tsx

```

Папка `components` отвечает за хранение двух групп компонентов (атомы и молекулы). Сами организмы располагаются в директории `pages`, в которой также находятся компоненты страниц, которые размещают в себе организмы и служат для формирования шаблона страницы.

Директория `api` содержит методы для общения с REST API, методы используют библиотеку `axios`, для более удобной обработки результатов.

В директории `utils` находятся вспомогательные функции приложения, такие как форматирование дат, валидация полей или различные текстовые преобразования.

Директории `fonts`, `icons` и `styles` служат для кастомизации внешнего вида приложения и содержат в себе шрифты, иконки и CSS-стили соответственно.

Классы из директории `stores` представляют из себя функционально законченные фрагменты программы — модули, которые взаимодействуют с серверной частью приложения посредством HTTP-запросов.

Итак, все данные выделяются в отдельные сущности. Наше приложение стремится к набору из двух папок:

- `components`, где лежат все view-компоненты;
- `stores`, где будут содержаться данные, а также логика работы с ними.

Например, типовой компонент для ввода данных у нас состоит из двух файлов: `ChatInput` и `ChatStore`. Первый файл — это глупый компонент React, отвечающий строго за отображение, второй — данные этого компонента, правила работы с пользователем (`onClick`, `onChange` и т.д.)

Для отображения всего приложения используется корневой компонент `App`, а для управления потоками данных основное хранилище `RootStore`.

Принцип взаимодействия автономных компонентов View-Store простой: RootStore знает всё обо всех хранилищах всех нужных компонентов (рисунок 2.1). Другие хранилища ничего не знают об окружающем мире вообще. Таким образом, мы гарантированно знаем, куда идут наши данные и где их перехватывать.

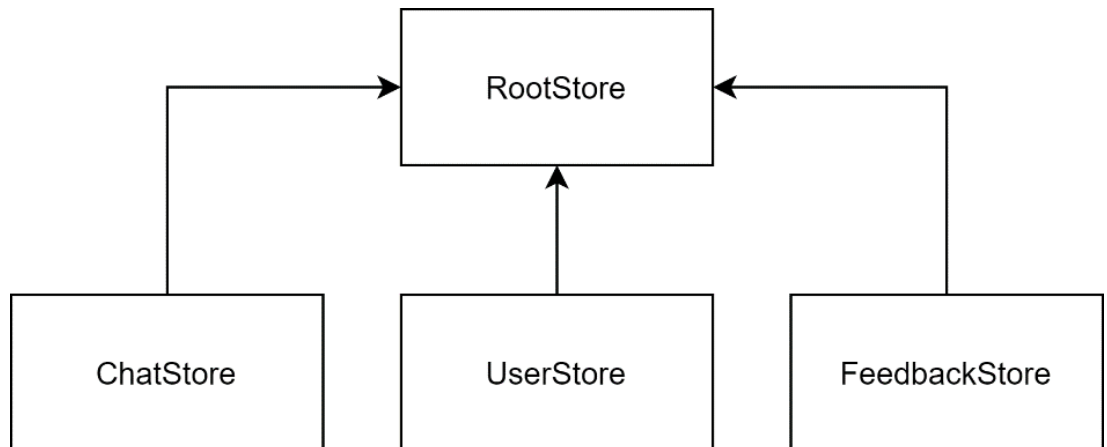


Рисунок 2.1. – Принцип взаимодействия хранилищ

Каждое событие (например, клик) вызывает действие, обновляющее наблюдаемое состояние. Изменения наблюдаемого состояния распространяются на все вычисления и побочные эффекты (View), которые зависят от вносимых изменений. Данный принцип работы представлен на рисунке 2.2.

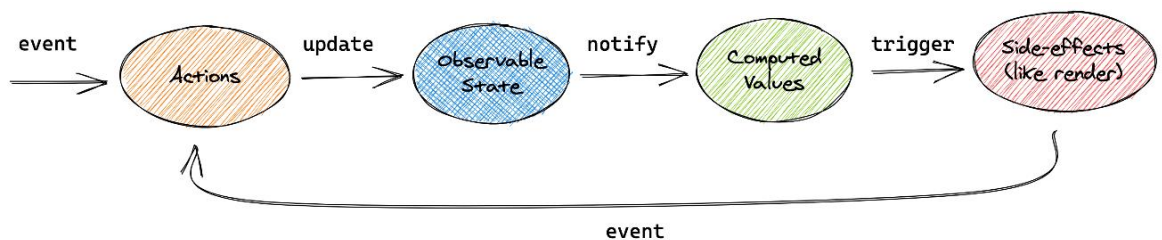


Рисунок 2.2 – Принцип работы MobX

Есть ещё две основные концепции Mobx – inject и observer.

inject внедряет только необходимый хранилища в приложение. Разные части нашего приложения используют разные store, которые перечисляются в

`inject`. Естественно, подключаемые хранилища должны быть изначально перечислены в `Provider`, описанном в корневом файле `index.ts`.

`observer` – декоратор указывает, что компонент будет подписан на данные, которые изменяются с помощью `Mobx`. Данные изменились – в компоненте возникла реакция.

Таким образом, последовательность выполнения действий следующая. Компоненты отслеживают пользовательские события и изменяют своё состояние. `RootStore` через `computed` вычисляет результат, основанный только на тех `state`, которые изменились. Разные `computed` смотрят за изменением разных состояний в разных хранилищах. Далее через `reaction` на основе результатов `computed` выполняются действия с `observables` и различные сайд-эффекты. На `observables` подписаны дочерние компоненты, которые при необходимости перерисовываются. Это называется однонаправленный поток данных с полным контролем над тем, где и что меняется.

Таким образом можно выделить основные преимущества данной архитектуры:

1. Группировка компонентов по их функциональности (атомы, молекулы или организмы).
2. Разделение бизнес-логики и компонентов отображения, модули являются независимыми и ничего не знают друг о друге.
3. Удобство тестирования за счет возможности запускать тесты для одного компонента.
4. За счет использования `MobX` присутствует поддержка объектно-ориентированного программирования.
5. Две основные концепции – `inject` и `observer` позволяют легко отслеживать изменения в хранилищах и перерисовывать `view`-компоненты.
6. Производительность. Скорость работы `MobX` не зависит от количества компонентов, потому что мы заранее знаем список компонентов, которые надо обновить, – вместо  $O(n)$  сложности `Redux`.

## 2.2 Моделирование бизнес-процессов

Для моделирования бизнес-процессов воспользуемся спецификацией BPMN, основной целью которого является создание стандартного набора условных обозначений, понятных всем бизнес-пользователям. BPMN нотация призвана служить связующим звеном между фазой дизайна бизнес-процесса и фазой его реализации.

В системе онлайн консультирования с врачами выделены три роли: пациент, врач, модератор. В нотации данные роли отображаются в виде прямоугольника, который содержит несколько объектов потока управления или артефактов (данные или текстовые нотации).

Объекты потока управления разделяются на три основных типа: события, действия и логические операторы.

События изображаются окружностью и означают какое-либо происшествие в мире. События инициируют действия или являются их результатами. В процессе моделирования были выделены события ожидания подтверждения верификации врача или ее отклонении и событие ожидания начала консультации.

Действия изображаются прямоугольниками со скругленными углами. В разработанной BPMN-нотации действиями являются процессы регистрации, верификации, заполнение личного кабинета, поиск специалиста, запись на прием, оплата, сам процесс консультации и последующее вынесение рекомендации или, в случае пациента, оставление отзыва.

С помощью логических операторов организуется ветвление и синхронизация потоков управления в модели процесса. В процессе моделирования был выбран оператор “ИЛИ”, реализующий ветвление к ветке с процессом заполнения личного кабинета специалиста.

Результат моделирования бизнес-процессов системы онлайн консультирования с врачами представлен в приложении А.

### 2.3 Проектирование сценариев пользователя

В таблицах 2.1-2.28 спроектированы сценарии взаимодействия пользователя с интерфейсом приложения.

Таблица 2.1 – UC-1. Просмотр главной страницы

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь находится на главной странице	<p>Пользователю доступны следующие действия:</p> <ul style="list-style-type: none"> <li>• Авторизация (UC-3)</li> <li>• Регистрация (UC-4)</li> <li>• Просмотр списка всех специалистов (UC-5)</li> <li>• Написать в поддержку (UC-2)</li> </ul> <p>Если пользователь имеет статус авторизованного в системе, то ему доступен переход в личный кабинет:</p> <ul style="list-style-type: none"> <li>• Личный кабинет для пациента (UC-8).</li> <li>• Личный кабинет для специалиста (UC-9)</li> </ul>	
2	Пользователь просматривает слайдер с самыми опытными специалистами	<p>Пользователь видит следующие данные по каждому из специалистов:</p> <ul style="list-style-type: none"> <li>• Фотография специалиста</li> <li>• Специальности</li> <li>• ФИО специалиста</li> <li>• Опыт работы</li> </ul>	Не найдено ни одного специалиста - отображается заглушка "Специалисты не найдены"
3	Пользователь просматривает информацию о сервисе на данной странице	Пользователь видит блок с преимуществами сервиса и этапами процесса записи на консультацию.	
4	Пользователь просматривает слайдер с отзывами о сервисе	<p>Пользователь видит следующие данные для каждого отзыва:</p> <ul style="list-style-type: none"> <li>• Фото пациента</li> <li>• Фамилия и имя пациента</li> <li>• Рейтинг отзыва</li> <li>• Текст отзыва</li> </ul>	Не найдено ни одного отзыва - отображается заглушка "Специалисты не найдены"

Таблица 2.2 – УС-2. Отправка письма в поддержку

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь находится на главной странице		
2	Пользователь заполняет поля формы для обращения в поддержку	Перечень заполняемых полей: <ul style="list-style-type: none"> <li>• Имя</li> <li>• E-mail</li> <li>• Тема письма</li> <li>• Текст вопроса</li> </ul>	Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
3	Пользователь нажимает “Готово”	Кнопка “Готово” не активна в случае, если: <ul style="list-style-type: none"> <li>• Хотя бы одно из полей формы заполнено не верно</li> <li>• Пользователь не дал согласия на обработку своих персональных данных</li> </ul>	
4	Отображается статус отправки письма под формой		

Таблица 2.3 – УС-3. Авторизация

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь нажимает на кнопку “Авторизация”		
2	Пользователь вводит логин и пароль в открывшееся модальное окно		Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
3	Пользователь нажимает кнопку “Войти”	Во время данного шага на кнопке отображается индикатор загрузки и кнопка становится не активной.	Пользователь с введенным логином и паролем не найден: <ul style="list-style-type: none"> <li>• Отказ в авторизации</li> <li>• Отображается соответствующее сообщение</li> </ul>
4	Закрывается модальное окно и пользователю становится доступен переход в личный кабинет		

Таблица 2.4 – УС-4. Регистрация

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь находится на странице регистрации		
2	Пользователь заполняет поля формы		Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
3	Пользователь нажимает “Зарегистрироваться”	Кнопка не активна в случае, если пользователь не дал согласия на обработку своих персональных данных	
4	Отображается модальное окно с просьбой подтвердить свой аккаунт по почте.	Пользователю доступна возможность повторной отправки письма на почту	

Таблица 2.5 – УС-5. Просмотр списка всех специалистов

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь находится на странице “Специалисты”		
2	Пользователь просматривает список специалистов	<p>Пользователь видит следующие данные по каждому из специалистов:</p> <ul style="list-style-type: none"> <li>• Фотография специалиста</li> <li>• Специальности</li> <li>• ФИО специалиста</li> <li>• Время работы</li> <li>• Рейтинг</li> <li>• Информация о себе</li> <li>• Опыт работы</li> <li>• Стоимость консультации</li> </ul> <p>Пользователю доступен фильтр по специальности и поиск по ФИО специалиста.</p> <p>Пользователь имеет возможность перейти к просмотру подробной</p>	Не найдено ни одного специалиста - отображается заглушка “Специалисты не найдены”



		<p>информации о специалисте (УС-6)</p> <p>Список отображается в режиме пейджинга по 3 позиции.</p>	
--	--	--	--

Таблица 2.6 – УС-6. Просмотр профиля специалиста

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь просматривает список специалистов		
2	Пользователь переходит к просмотру профиля специалиста	<p>Отображаемая информация:</p> <ul style="list-style-type: none"> <li>• Фотография специалиста</li> <li>• Специальности</li> <li>• ФИО специалиста</li> <li>• Время работы</li> <li>• Рейтинг</li> <li>• Информация о себе</li> <li>• Опыт работы</li> <li>• Стоимость консультации</li> <li>• Список отзывов</li> <li>• Список образования</li> <li>• Список опыта работы</li> <li>• Основные направления</li> </ul> <p>Пациент имеет возможность записаться на прием к специалисту (УС-7)</p>	Специалист не найден - отображается заглушка 404

Таблица 2.7 – УС-7. Запись на прием к специалисту

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент просматривает профиль специалиста		
2	Пациент нажимает на “Записаться на прием”		
3	Пациент автоматически переходит на первый этап записи на прием		<p>Специалист по данному id не найден:</p> <ul style="list-style-type: none"> <li>• Отображается заглушка 404</li> </ul>

4	Пациент заполняет поля формы	Перечень заполняемых полей: <ul style="list-style-type: none"> <li>• Дата приема</li> <li>• Время приема</li> <li>• Способ связи</li> <li>• Специальность</li> </ul>	Одно из возможных полей заполнено не верно: <ul style="list-style-type: none"> <li>• Отображается предупреждение под невалидным полем</li> </ul>
5	Пациент нажимает кнопку “Выбрать и продолжить”		
6	Пациент автоматически переходит на второй этап		
7	Пациент заполняет симптомы		Поле симптомов не заполнено или заполнено не верно - отображается предупреждение
8	Пациент нажимает кнопку “Продолжить”		
9	Пациент автоматически переходит на третий этап		
10	Пациент оплачивает услугу	Кнопка не активна в случае, если: <ul style="list-style-type: none"> <li>• Пациент не поставил чекбокс, что он проверил данные и готов к оплате</li> </ul>	
11	Пациент нажимает “Продолжить”		
12	Пациент автоматически переходит на последний этап		
13	Пациент успешно записался на прием	Отображается текст “Вы успешно записались на прием” и список деталей записи.  Пациент имеет возможность: <ul style="list-style-type: none"> <li>• Перейти в личный кабинет пациента (UC-8)</li> <li>• Перейти на главную страницу (UC-1)</li> </ul>	

Таблица 2.8 – UC-8. Просмотр личного кабинета пациента

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент нажимает кнопку “Личный кабинет”	Если пользователь не заполнил анкету пациента, то его автоматически перекидывает на страницу заполнения анкеты (UC-10)	

2	Пациент автоматически переходит на главную страницу личного кабинета	Пациенту доступны следующие действия: <ul style="list-style-type: none"> <li>• Просмотр списка записей на прием (УС-12)</li> <li>• Просмотр назначений специалистов (УС-14)</li> <li>• Просмотр своих анализов (УС-15)</li> <li>• Просмотр медицинской карты (УС-19)</li> <li>• Страница сообщений (УС-27)</li> <li>• Страница настроек (УС-28)</li> </ul>	
---	--	--	--

Таблица 2.9 – УС-9. Просмотр личного кабинета специалиста

Шаг	Действие	Комментарий	Возможная ошибка
1	Специалист нажимает кнопку “Личный кабинет”	Если специалист не прошел верификацию, то его автоматически перекидывает на страницу заполнения заявки на верификацию (УС-11)	Если специалист не прошел верификацию или его заявка еще обрабатывается, то отображается заглушка с соответствующей информацией
2	Специалист автоматически переходит на главную страницу личного кабинета	Специалисту доступны следующие действия: <ul style="list-style-type: none"> <li>• Просмотр информации о себе (УС-21)</li> <li>• Просмотр графика работы (УС-23)</li> <li>• Просмотр своих пациентов (УС-25)</li> <li>• Страница сообщений (УС-27)</li> <li>• Страница настроек (УС-28)</li> </ul>	

Таблица 2.10 – УС-10. Заполнение анкеты пациента

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь попадает на страницу заполнения анкеты пациента”		

2	Пациент заполняет поля формы	Перечень заполняемых полей: <ul style="list-style-type: none"> <li>• Вес</li> <li>• Рост</li> <li>• Группа крови</li> <li>• Резус фактор</li> <li>• Аллергия</li> <li>• Хронические заболевания</li> <li>• Операции</li> <li>• Курение</li> <li>• Алкоголь</li> <li>• Другие вредные привычки</li> <li>• Была ли процедура по переливаю крови</li> </ul>	Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
3	Пациент нажимает “Сохранить анкету”		
4	Пациента автоматически перекидывает на главную страницу личного кабинета		

Таблица 2.11 – УС-11. Заполнение заявки на верификацию специалиста

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь попадает на страницу заполнения заявки на верификацию специалиста”		
2	Специалист заполняет поля формы	Перечень заполняемых полей: <ul style="list-style-type: none"> <li>• ИИН</li> <li>• Специальности</li> <li>• Стаж работы</li> <li>• Фото</li> <li>• Резюме</li> <li>• Диплом</li> </ul>	Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
3	Специалист нажимает “Отправить модератору”		
4	Специалиста автоматически перекидывает на страницу заглушки с информацией о статусе проверки заявки		

Таблица 2.12 – УС-12. Просмотр списка записей на прием к специалистам

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент находится в личном кабинете”		

2	Пациент открывает раздел “Мои записи”		
3	Пациент просматривает список записей на консультацию	<p>Отображается три списка (активные, запланированные и история) Каждый элемент списка содержит:</p> <ul style="list-style-type: none"> <li>• Дату и время консультации</li> <li>• Информацию о враче (фото, ФИО и специальность)</li> </ul> <p>Пациенту доступны следующие действия:</p> <ul style="list-style-type: none"> <li>• Перейти к консультации (UC-27)</li> <li>• Оставить отзыв (UC-13)</li> <li>• Записаться на прием (UC-7)</li> <li>• Просмотр профиля специалиста (UC-6)</li> <li>• Отменить запись на консультацию</li> </ul>	Если какой-либо список пустой, то в соответствующем списке отображается информация: “Консультаций не найдено”

Таблица 2.13 – UC-13. Отправка отзыва о специалисте

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент нажимает на кнопку “Оставить отзыв”		
2	Пациент заполняет форму отправки отзыва в открывшемся модальном окне	<p>Перечень заполняемых полей:</p> <ul style="list-style-type: none"> <li>• Текст отзыва</li> <li>• Рейтинг</li> </ul>	Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
3	Пациент нажимает кнопку “Отправить отзыв”	Во время данного шага на кнопке отображается индикатор загрузки и кнопка становится не активной.	
4	Закрывается модальное окно, отзыв успешно отправлен		

Таблица 2.14 – UC-14. Просмотр списка назначений пациентом

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент находится в личном кабинете”		
2	Пациент открывает раздел “Назначения”		
3	Пациент просматривает список назначений специалистов	<p>Пациент видит следующие данные по каждому объекту списка:</p> <ul style="list-style-type: none"> <li>• Информация о враче (фото, ФИО и специальность)</li> <li>• Дата оставления назначения</li> <li>• Текст назначения</li> </ul> <p>При клике на информацию о специалисте, пациент переходит на страницу просмотра его профиля (UC-6)</p>	Список назначений не найден или является пустым - отображается информация “Назначений не найдено”

Таблица 2.15 – UC-15. Просмотр списка анализов пациентом

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент находится в личном кабинете”		
2	Пациент открывает раздел “Мои анализы”		
3	Пациент просматривает список добавленных анализов или снимков	<p>Пациент видит следующие данные по каждому объекту списка:</p> <ul style="list-style-type: none"> <li>• Фото анализа или снимка</li> <li>• Название документа</li> <li>• Дата прохождения анализа или создания снимка</li> </ul> <p>Пациенту доступен фильтр по категории документа, возможные состояния фильтра:</p> <ul style="list-style-type: none"> <li>• Все</li> <li>• Анализы</li> <li>• Снимки</li> </ul> <p>По умолчанию фильтр находится в состоянии “Все”.</p> <p>Пациенту доступны следующие опции:</p>	Анализы не найдены или список является пустым - отображается информация “Анализы не найдены”

		<ul style="list-style-type: none"> <li>• Добавления нового документа (UC-16)</li> <li>• Удаление документа (UC-17)</li> <li>• Просмотр документа (UC-18)</li> </ul>	
--	--	---	--

Таблица 2.16 – UC-16. Добавление нового анализа или снимка

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент находится в разделе личного кабинета “Мои анализы”		
2	Пациент нажимает на кнопку добавления документа		
3	Отображается форма создания нового документа		
4	Пациент заполняет поля формы	Перечень заполняемых полей: <ul style="list-style-type: none"> <li>• Файл документа</li> <li>• Название документа</li> <li>• Дата сдачи анализа или создания списка</li> <li>• Тип документа (анализ или снимок)</li> </ul>	Анализы не найдены или список является пустым - отображается информация “Анализы не найдены”
5	Пациент нажимает кнопку “Загрузить”		Одно из возможных полей заполнено не верно: Отображается предупреждение под невалидным полем
6	Новый анализ/снимок создан	Новый документ отображается в списке анализов данного раздела	
7	Пациент автоматически переходит к просмотру списка загруженных документов		

Таблица 2.17 – UC-17. Удаление анализа или снимка

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент находится в разделе личного кабинета “Мои анализы”		

2	Пациент просматривает список загруженных анализов или снимков		
3	Пациент выбирает опцию “Удалить документ”		
4	Пациент подтверждает удаление документа в модальном окне		
5	Документ удален		

Таблица 2.18 – УС-18. Просмотр документа

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь нажимает на документ		
2	Открывается окно просмотра выбранного документа	Пользователь имеет возможность: <ul style="list-style-type: none"> <li>• Увеличить/уменьшить документ</li> <li>• Перейти к следующему/предыдущему</li> <li>• Закрыть средство просмотра документа</li> </ul>	

Таблица 2.19 – УС-19. Просмотр медицинской карты пациентом

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент находится в личном кабинете”		
2	Пациент открывает раздел “Медицинская карта”		
3	Пациент автоматически переходит на страницу просмотра медицинской карты	Отображается следующая информация: <ul style="list-style-type: none"> <li>• Фото</li> <li>• ФИО</li> <li>• Дата рождения</li> <li>• Рост</li> <li>• Вес</li> <li>• ИМТ</li> <li>• Группа крови</li> <li>• Хронические заболевания</li> <li>• Вредные привычки</li> <li>• Операции</li> <li>• Аллергия</li> <li>• Переливание крови</li> </ul>	



		Пациенту доступна опция редактирования информации медицинской карты (UC-20)	
--	--	---	--

Таблица 2.20 – UC-20. Редактирование медицинской карты

Шаг	Действие	Комментарий	Возможная ошибка
1	Пациент находится в разделе личного кабинета “Медицинская карта”		
2	Пациент выбирает опцию “Редактировать данные”		
3	Отображается редактируемая форма с информацией об выбранном объекте	В поля формы внесена информация, указанная при создании либо последнем редактировании объекта	
4	Пациент заполняет поля формы		
5	Пациент нажимает кнопку “Изменить”		Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
6	Информация об объекте изменяется на указанную в форме редактирования		

Таблица 2.21 – UC-21. Просмотр профиля специалиста в личном кабинете

Шаг	Действие	Комментарий	Возможная ошибка
1	Специалист находится в личном кабинете”		
2	Специалист открывает раздел “Главная”		
3	Специалист автоматически переходит к просмотру информации о себе	Отображается следующая информация: <ul style="list-style-type: none"> <li>• Фото</li> <li>• ФИО</li> <li>• Стоимость консультации</li> <li>• Дополнительная информация о себе</li> <li>• Рейтинг</li> <li>• Отзывы</li> <li>• Образование</li> <li>• Опыт работы</li> <li>• Основные направления</li> </ul>	

		Специалисту доступна опция изменения информации (стоимость консультации, о себе, образование и опыт работы) (УС-22)	
--	--	---	--

Таблица 2.22 – УС-22. Редактирование информации специалиста

Шаг	Действие	Комментарий	Возможная ошибка
1	Специалист находится в разделе личного кабинета “Главная”		
2	Специалист выбирает опцию “Редактировать данные”		
3	Отображается редактируемая форма с информацией об выбранном объекте	В поля формы внесена информация, указанная при создании либо последнем редактировании объекта	
4	Специалист заполняет поля формы		
5	Специалист нажимает кнопку “Изменить”		Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
6	Информация об объекте изменяется на указанную в форме редактирования		

Таблица 2.23 – УС-23. Просмотр графика работы специалиста

Шаг	Действие	Комментарий	Возможная ошибка
1	Специалист находится в личном кабинете”		
2	Специалист открывает раздел “График работы”		
3	Специалист автоматически переходит к просмотру своего графика работы	В блоке отображаются редактируемые поля “с” и “по” для каждого дня недели.  Специалисту доступна опция изменения полей “с” и “по” при клике на них.	

Таблица 2.24 – UC-24. Просмотр списка пациентов специалистом

Шаг	Действие	Комментарий	Возможная ошибка
1	Специалист находится в личном кабинете”		
2	Специалист открывает раздел “Пациенты”		
3	Специалист автоматически переходит к просмотру своих консультаций с пациентами	<p>Раздел разделен на два блока:</p> <ul style="list-style-type: none"> <li>• Календарь</li> <li>• Списки пациентов</li> </ul> <p>Специалисту доступен фильтр по состоянию консультации на выбранную дату в календаре, возможные состояния фильтра:</p> <ul style="list-style-type: none"> <li>• Новые</li> <li>• История</li> </ul> <p>По умолчанию фильтр находится в состоянии “Новые”.</p> <p>Каждый элемент списка консультаций содержит следующую информацию:</p> <ul style="list-style-type: none"> <li>• Тип консультации (первичная или повторная)</li> <li>• Статус консультации или время до ее начала</li> <li>• Дата начала консультации</li> <li>• Информация о пациенте (фото, ФИО)</li> <li>• Способ связи (видеосвязь, аудиосообщения или сообщения в чате)</li> </ul> <p>Специалисту доступна опция просмотра пациента при клике на консультацию (UC-25)</p>	Консультации не найдены или список является пустым - отображается информация “Записей на прием не найдено”

Таблица 2.25 – UC-25. Просмотр профиля пациента специалистом

Шаг	Действие	Комментарий	Возможная ошибка
1	Специалист находится в разделе личного кабинета “Пациенты”		

2	Специалист кликает по позиции списка		
3	Специалист автоматически переходит к просмотру медицинской карты пациента	<p>Отображается следующая информация:</p> <ul style="list-style-type: none"> <li>• Фото</li> <li>• ФИО</li> <li>• Дата рождения</li> <li>• Рост</li> <li>• Вес</li> <li>• ИМТ</li> <li>• Группа крови</li> <li>• Хронические заболевания</li> <li>• Вредные привычки</li> <li>• Операции</li> <li>• Аллергия</li> <li>• Переливание крови</li> <li>• Симптомы</li> <li>• Анализы</li> <li>• Назначения</li> </ul> <p>Специалисту доступен фильтр по категории анализов, возможные состояния фильтра:</p> <ul style="list-style-type: none"> <li>• Все</li> <li>• Анализы</li> <li>• Снимки</li> </ul> <p>По умолчанию фильтр находится в состоянии “Все”.</p> <p>Специалисту доступны следующие опции:</p> <ul style="list-style-type: none"> <li>• Просмотр анализа или снимка (UC-18)</li> <li>• Назначить рекомендации (UC-26)</li> <li>• Связаться с пациентом (UC-27)</li> </ul>	Пациент по данному id не найден или у специалиста отсутствует доступ к его просмотру - отображается заглушка 404

Таблица 2.26 – UC-26. Назначение рекомендаций специалистом

Шаг	Действие	Комментарий	Возможная ошибка
1	Специалист находится в режиме просмотра профиля пациента		
2	Специалист нажимает на кнопку “Назначить”		

3	Специалист заполняет форму назначений рекомендаций в открывшемся модальном окне		
4	В модальном окне специалист нажимает кнопку “Назначить”	Во время данного шага на кнопке отображается индикатор загрузки и кнопка становится не активной.	Текстовое поле назначения не заполнено или заполнено неверно - отображается предупреждение под невалидным полем
5	Закрывается модальное окно, рекомендации назначены успешно		

Таблица 2.27 – UC-27. Консультация

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь находится в разделе личного кабинета “Сообщения”		
2	Пользователь автоматически переходит на страницу связи с пациентом/врачом	<p>Каждый элемент списка диалогов содержит:</p> <ul style="list-style-type: none"> <li>• Фото</li> <li>• ФИО специалиста или пациента</li> <li>• Специальность (только для пациентов)</li> </ul> <p>Раздел страницы для связи содержит в себе следующую информацию:</p> <ul style="list-style-type: none"> <li>• Фото</li> <li>• ФИО специалиста или пациента</li> <li>• Специальность (только для пациентов)</li> <li>• Сообщения</li> <li>• Текстовое поле для отправки сообщения</li> </ul> <p>Пользователю доступен функционал поиска по диалогам:</p> <ul style="list-style-type: none"> <li>• Поиск производится от одного символа в любой части слова</li> </ul>	Консультация закончена или еще не началась - отображается заглушка с текстом “К сожалению, доступ к чату разрешен только во время консультации”

		<ul style="list-style-type: none"> <li>Поиск производится по значениям поля "ФИО"</li> </ul> <p>Пользователю доступны следующие опции:</p> <ul style="list-style-type: none"> <li>Связаться по видео- или аудио- связи (если она доступна)</li> <li>Прикрепление файла</li> <li>Отправка аудиосообщения</li> <li>Отправка текстового сообщения</li> </ul>	
--	--	---	--

Таблица 2.28 – УС-28. Настройка аккаунта

Шаг	Действие	Комментарий	Возможная ошибка
1	Пользователь находится в разделе личного кабинета “Настройки”		
2	Пользователь автоматически переходит на страницу редактирования информации об аккаунте	<p>Отображается следующая информация для редактирования</p> <ul style="list-style-type: none"> <li>Фото пользователя</li> <li>Фамилия</li> <li>Имя</li> <li>Отчество</li> <li>Дата рождения</li> <li>Телефон</li> <li>Пол</li> </ul>	
3	Пользователь редактирует доступные поля в форме		
4	Пользователь нажимает кнопку “Сохранить изменения”		Одно из возможных полей заполнено не верно - отображается предупреждение под невалидным полем
5	Редактируемая информация изменяется на указанную в форме редактирования		

Таким образом были описаны основные сценарии взаимодействия пользователя с интерфейсом.

## **Выводы по разделу 2**

В данном разделе был выбран подход для проектирования архитектуры, описаны принципы взаимодействия модулей и выделены преимущества выбранного подхода.

Также в данном разделе были спроектированы сценарии взаимодействия пользователя с пользовательским интерфейсом системы.

## **3 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ДЛЯ СИСТЕМЫ ОНЛАЙН-КОНСУЛЬТИРОВАНИЯ С ВРАЧАМИ**

### **3.1 Разработка компонентов программного модуля**

Следуя из выбранной во втором разделе текущей пояснительной записки архитектуры приложения, при разработке web-приложения были спроектированы независимые друг от друга компоненты, описание которых представлено ниже.

Модуль HomeStore используется на главной странице для вывода специалистов с наивысшим опытом работы и отправки формы обратной связи. Для данной реализации он содержит в себе массив специалистов и логический индикатор, для отображения анимации загрузки, а также два объекта с полями формы обратной связи и объект для валидации данных полей. Модуль взаимодействует с двумя конечными точками серверной части:

- GET /api/v1/doctor/most-experienced?count={count} – отвечает за получение самых опытных специалистов;
- POST /api/v1/feedback/leave – отвечает за отправку формы обратной связи.

Для выполнения процесса авторизации используется модуль SignInStore, который содержит объекты с данными полей для входа и функцию отправки данного объекта. Данная функция использует метод POST-запроса /api/v1/auth/sign-in. После удачного выполнения данного запроса, внутри функции выполняется установка в localStorage access-токена, который затем используется для отправки в заголовках запросов, чтобы идентифицировать пользователя.

Следующий модуль SignUpStore используется на странице регистрации нового пользователя и содержит метод doSignUp для отправки POST запроса на сервер. В тело запроса передается объект с полями формы, такие как: тип



пользователя (пациент или доктор), имя, фамилия, отчество, дата рождения, пол, телефон, пароль и адрес электронной почты, который затем используется для подтверждения регистрации. Модуль взаимодействует с двумя конечными точками сервера:

- POST /api/v1/auth/sign-up – отвечает за отправку формы регистрации;
- POST /api/v1/auth/send-email-with-token – метод для повторной отправки сообщения с подтверждением на почту.

В приложении также реализуется модуль UserStore, предназначенный для хранения текущей информации о пользователе, его статусе авторизации и реализующий функции получения информации о пользователе по access-токену и функцию выхода из аккаунта, в которой происходит очистка закешированных в других модулях полей. В процессе реализации данного класса применялись следующие API-методы:

- GET /api/v1/user/info – получение информации пользователя по токену в заголовках запроса;
- GET /api/v1/user/fresh-token – метод для обновления токена доступа при каждом входе в приложение.

Для вывода и поиска по фильтрам специалистов используется модуль SearchDoctorStore, который реализует HTTP-методы, представленные ниже, и состоит из массива найденных специалистов, объекта с текущей страницей специалистов, булевых переменных для отображения анимаций загрузки и строкового свойства для реализации поиска по ФИО:

- GET /api/v1/doctor/paginate?page={page}&count={count}&specialty={specialty} – отвечает за поиск по странице или специальности;
- GET /api/v1/doctor/paginate?page=1&count=4&fio={fullName} – запрос для поиска специалистов по имени в строке поиска.

Для просмотра выбранного врача существует отдельный модуль DoctorStore. В данном модуле находятся две функции, реализующие запрос на

поиск доктора по id из адресной строки и загрузку комментариев. Методы, используемые в данном модуле:

- GET /api/v1/doctor/info?id={id} – запрос на получение информации о докторе по id;
- GET /api/v1/doctor/review/list?reviewId={reviewId}&doctorId={doctorId}&count={count} – метод для получения комментариев по последнему id комментария в списке, id доктора и параметром для получения необходимого количества комментариев.

Модуль AppointmentStore применяется в процессе записи пациентом на консультацию. Данный модуль объединяет в себе всю логику четырех этапов записи. При открытии первого этапа происходит автоматически запрос на получение информации о выбранном специалисте, после чего выполняется получение его расписания на текущее время. В случае если выбранный врач не принимает пациентов в выбранный день, то в соответствующем поле на форме будет установлено “Нет приема”. Также данный класс содержит функцию отправки, заполненной на прошлых этапах формы. Модуль записи на консультацию использует три конечных точки:

- GET /api/v1/consultation/appointment/meta-info?doctorId={doctorId} – получение информации о выбранном докторе по его id;
- GET /api/v1/consultation/appointment/free-doctor-time?doctorId={doctorId}&date={date} – запрос на получение свободного времени по id специалиста и дате;
- POST /api/v1/consultation/appointment/create – применяется для отправки итоговой формы записи.

Модуль ModalsStore представляет из себя некий класс-менеджер, который управляет всеми диалоговыми окнами, их открытием и закрытием. Для этого в нем реализуются две соответствующие функции и массив всех существующих в приложении диалогов с их статусом (показывается или скрыто). Данный модуль

не реализует запросов к серверу и является компонентом, который может применяться в любой части приложения.

Далее идут модули, применяемые в личном кабинете пациента или специалиста.

Модуль `DashboardConsultationsStore` применяется на странице консультаций пациента и предназначен для выполнения запросов на получение списков консультаций по типу (активные, предстоящих и завершенных). Таким образом он содержит три специализированных под каждый тип консультации списков и соответствующих функций, реализующих их получение. Также в модуле содержится функция отмены консультации. Данный модуль реализует два запроса на сервер:

- `GET /api/v1/consultation/doctors-for-patient?consultationState={type}` – запрос, в котором зависимости от типа параметра будет возвращаться соответствующий список консультаций;
- `POST /api/v1/consultation/cancel` – применяется для отмены консультации.

Для управления анализами пациента применяется специализированный для этого модуль `DashboardAnalyzesStore`. В модуле реализуются функции для получения анализов, добавления файла и его удаление из списка. Таким образом в модуле применяются следующие конечные точки сервера:

- `GET /api/v1/patient/analysis/all` – запрос для получения всех анализов;
- `POST /api/v1/patient/analysis/append` – добавление анализа или снимка в список;
- `POST /api/v1/patient/analysis/delete` – применяется для удаления файла.

Для получения списка назначений от специалиста на соответствующей странице пациента применяется модуль `DashboardResultsStore`. Он реализует один запрос на сервер, который возвращает список назначений. Данный

эндпоинт представляет из себя простой GET запрос без параметров – `/api/v1/consultation/appointments`.

В личном кабинете пациента существует страница просмотра и редактирования своей медицинской карты. Для данного процесса был реализован модуль `DashboardMedicalCardStore`. Как уже написано раньше, в нем находится функция редактирования, которая выполняет POST запрос на url `/api/v1/patient/profile/change-medical-card`, в теле которого отправляется весь объект дополнительной информации о пользователе с уже измененными полями.

Для просмотра своего профиля с аккаунта специалиста в личном кабинете применяется модуль `DashboardDoctorProfileStore`. По аналогии с модулем для изменения медицинской карты пациента `DashboardMedicalCardStore`, текущий класс также содержит множества функций для заполнения объекта с дополнительной информацией о себе и функцию отправки этого объекта на сервер. Данный модуль реализует POST запрос `/api/v1/doctor/profile/change-info`.

Для следующей странице, странице просмотра пациента, был реализован модуль `DashboardPatientsStore`, в котором на каждое изменения даты в календаре или типа списка (новые или история) выполняется GET запрос `/api/v1/consultation/patients-for-doctor?date={date}&state={state}`, куда подставляются измененные параметры. После успешного выполнения запроса в массив пациентов заносится ответ с сервера и выполняется перерисовка списка на странице.

Так как доктор имеет возможность просмотра профиля пациента, записанного на консультацию, то целесообразно создать отдельный модуль, в котором хранится вся основная информация пациента и реализуются соответствующие запросы на получения этой информации. Данный модуль называется `DashboardPatientInfoStore`. В модуле применяется один GET запрос `/api/v1/patient/consultation-info?patientId={patientId}&consultationId={consultationId}` в который передается id пациента и текущей консультации.

Для редактирования расписания специалиста существует модуль `DashboardScheduleStore`. Данный класс уже содержит реализацию двух запросов:

- GET /api/v1/doctor/profile/schedule – запрос на получение текущего расписания;
- POST /api/v1/doctor/profile/change-schedule – запрос на изменение расписания.

В личном кабинете имеется общий модуль DashboardSettingsStore, который предназначен для редактирования личной информации о пользователе: имя, фамилия, отчество, дата рождения, номер телефона и пол. Для этого происходит взаимодействие с POST методом /api/v1/user/change-user-info и методом для изменения фотографии /api/v1/user/change-photo.

Основная коммуникация пациента и специалиста происходит в модулях ChatStore и SocketsStore. Данные модули взаимосвязаны и реализуют функции отправки сообщений и различных типов файлов, получения диалогов, подгрузки старых сообщений. В модуле SocketsStore реализуется сокет-соединение, а в модуле ChatStore применяются следующие запросы к серверу:

- GET /api/v1/chat/list – получение списка диалогов;
  - GET /api/v1/chat/message/list?chatId={chatId}&count={count}&lastMessageId={lastMessageId} – запрос на получение старых сообщений по id последнего сообщения в списке и id текущего чата;
  - POST /api/v1/chat/message/send-media – запрос для отправки файла.
- Листинг всех программных модулей представлен в приложении А.

### **3.2 Разработка интерфейса программного модуля**

При запуске web-приложения открывается главная страница, изображенная на рисунке 3.1, далее представлена секция с наилучшими специалистами (Рисунок 3.2), после чего следует основная информация о сервисе (Рисунок 3.3) и раздел с отзывами пациентов (Рисунок 3.4).

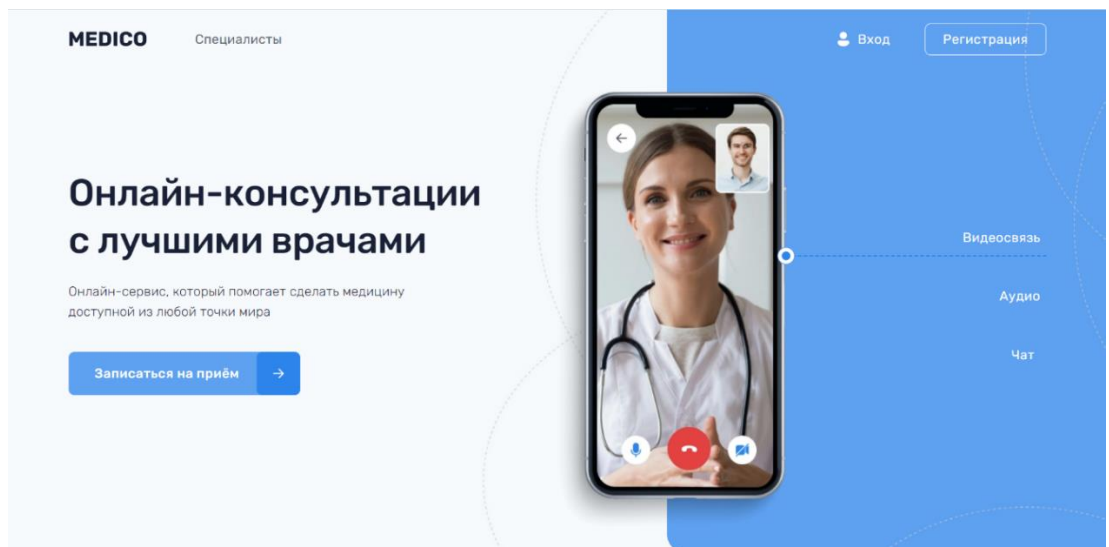
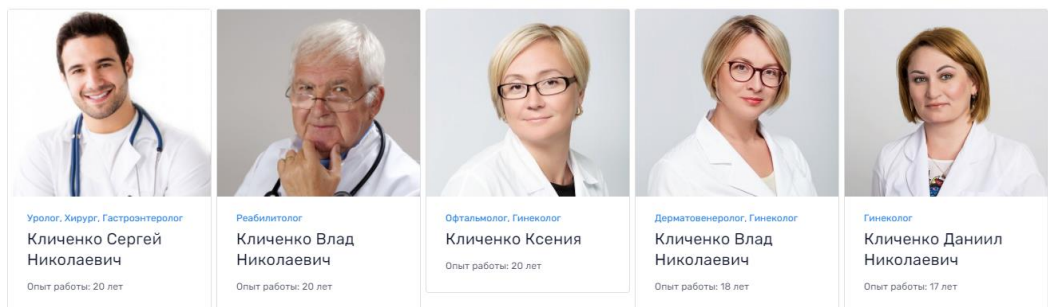


Рисунок 3.1 – Главный экран

## Специалисты

Поможем найти проверенного врача и записаться на консультацию в удобное время



Все врачи

Рисунок 3.2 – Секция со специалистами

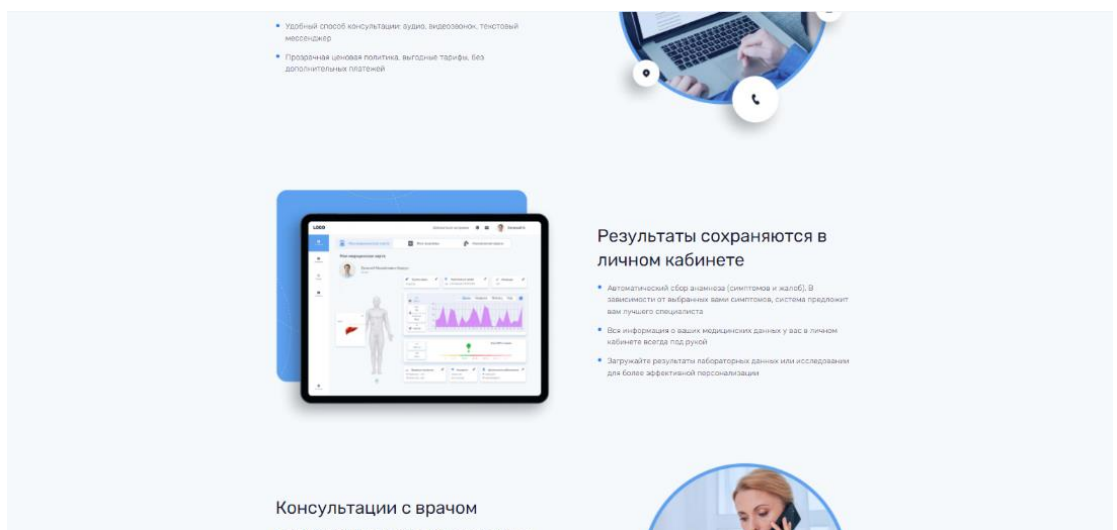


Рисунок 3.3 – Информация о сервисе

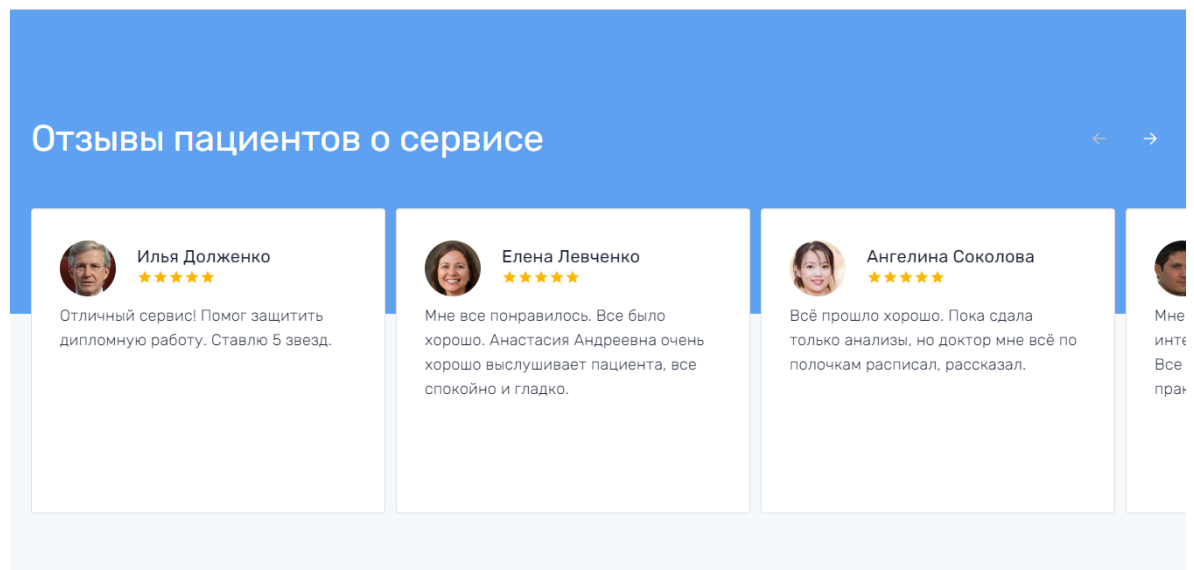


Рисунок 3.4 – Раздел с отзывами о сервисе

Снизу главной страницы располагается форма обратной связи, представленная на рисунке 3.5.

Рисунок 3.5 – Форма обратной связи

При нажатии на кнопку «Войти» открывается модальное окно с формой входа состоящие из полей «Email» и «Пароль» (Рисунок 3.6). Так же на этом окне находится кнопка перехода к регистрации (Рисунок 3.7)

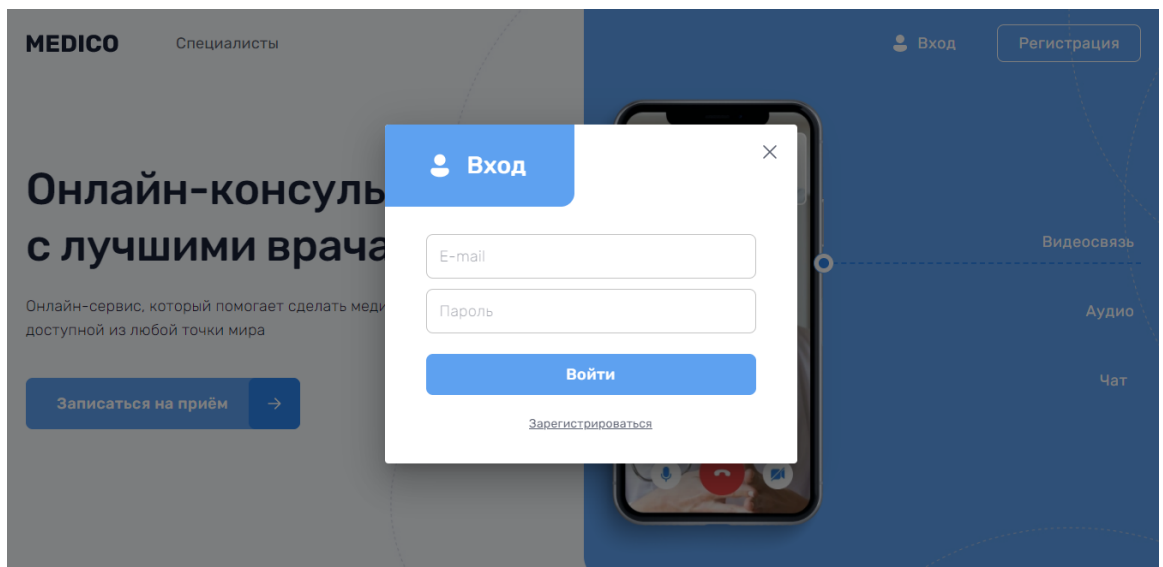


Рисунок 3.6 – Модальное окно входа

Рисунок 3.7 – Страница регистрации

После регистрации на почту пользователю приходит письмо с подтверждением аккаунта (Рисунок 3.8), и появляется модальное окно информирующее об этом (Рисунок 3.9).



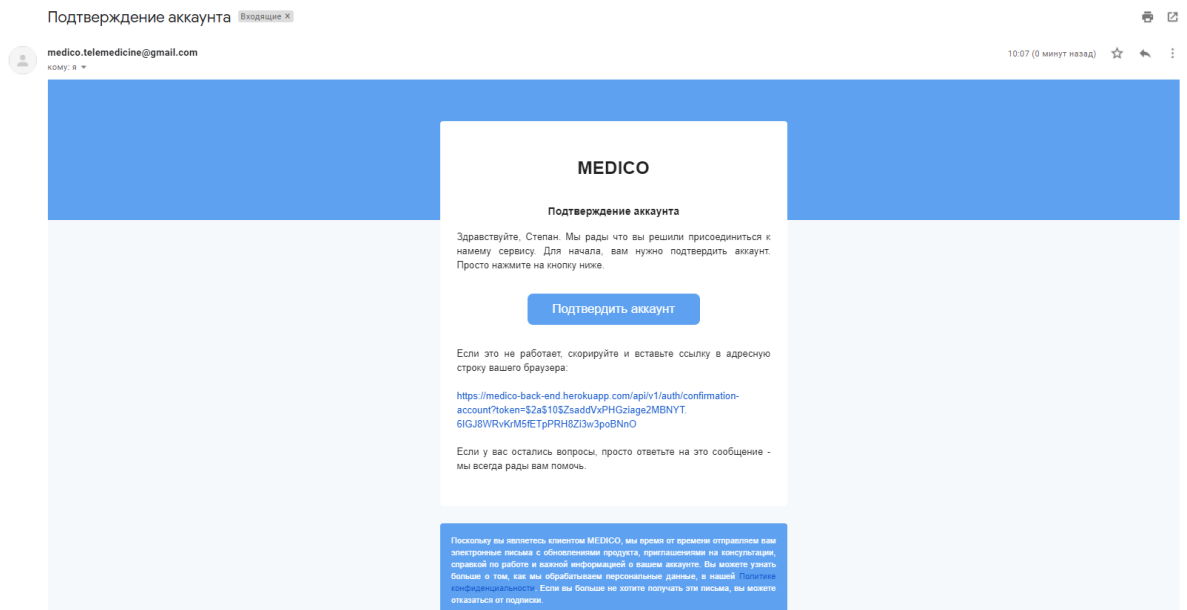


Рисунок 3.8 – Письмо на почте

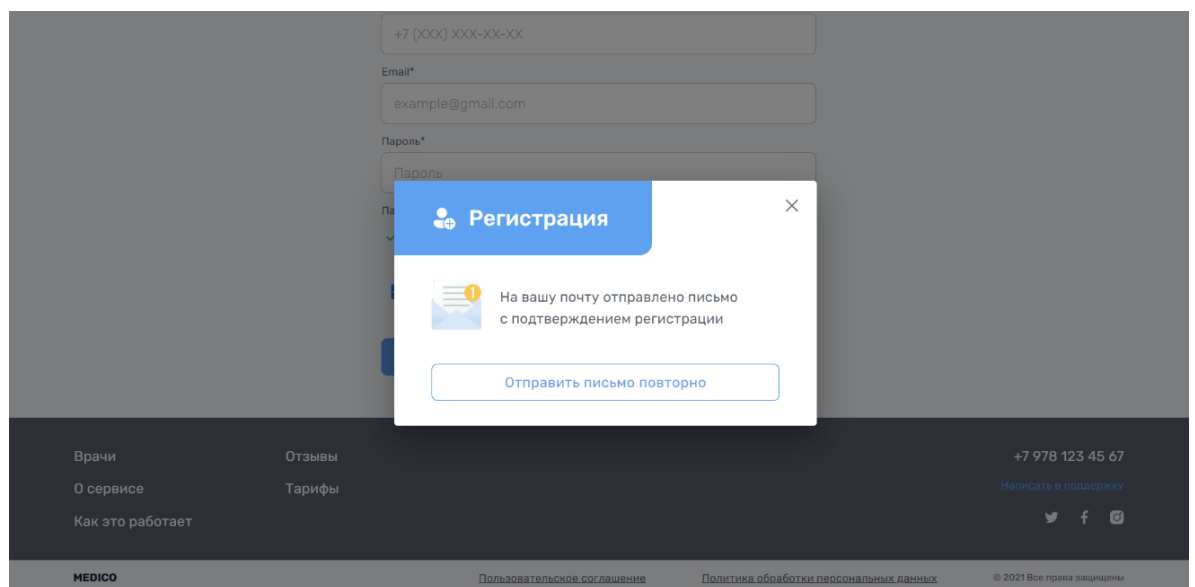


Рисунок 3.9 – Сообщение об отправке письма

После нажатия на кнопку «Подтвердить аккаунт» в письме пользователь перенаправляется на web-приложение (Рисунок 3.10), на главной странице которого открывается модальное окно с информацией об успешном подтверждении аккаунта и предложением войти в систему, нажав на соответствующую кнопку.

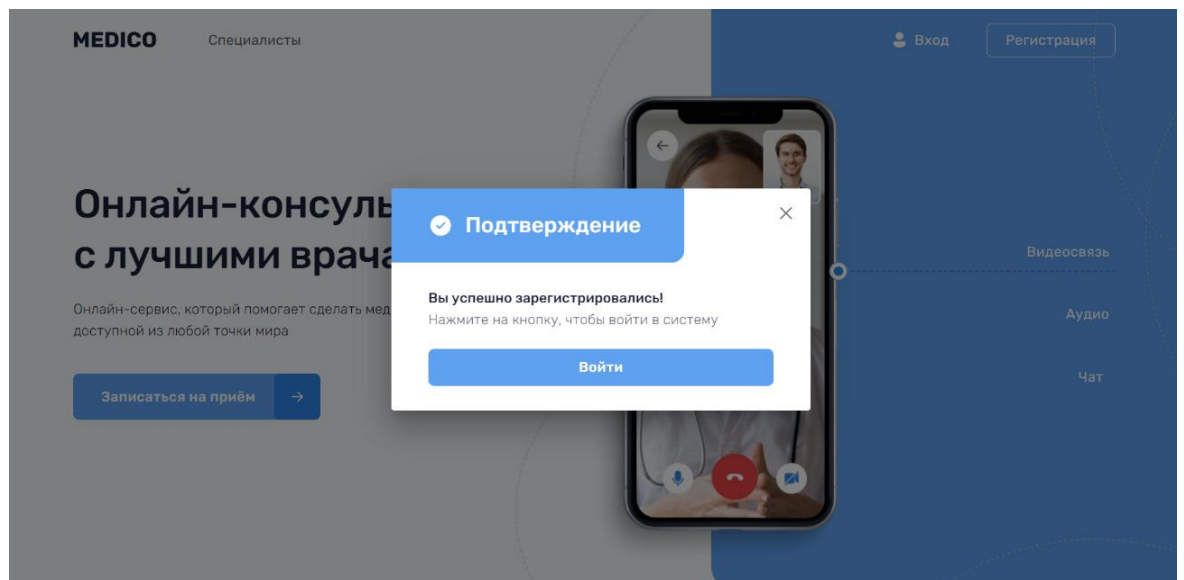


Рисунок 3.10 – Сообщение об успешном подтверждении аккаунта

Если пользователь является пациентом, то после авторизации необходимо заполнить анкету, нажав на личный кабинет, чтобы иметь доступ к данному разделу (Рисунки 3.11-3.12).

 The image shows a web interface for 'MEDICO' with a light blue header. The main content area has a light blue background. A white form titled 'Анкета' (Form) is centered. It contains several fields for patient information:
 

- 1. Укажите Ваш вес (кг) (Specify your weight in kg): Input field with '60 кг'.
- 2. Укажите Ваш рост (см) (Specify your height in cm): Input field with '180 см'.
- 3. Группа крови на руке: (Blood group on the arm): Radio buttons for I, II, III, IV.
- 4. Резус фактор: (Rh factor): Radio buttons for Rh+ and Rh-.
- 5. Есть ли у вас аллергия? Если да, то на что? (Do you have allergies? If yes, to what?): Text input field with 'Да, есть аллергия сезонная на пыльцу'.
- 6. Хронические заболевания (Chronic diseases): Text input field with 'Гайморит, астма'.
- 7. Операции с датой (Operations with date): Text input field with 'Операция - дата, ...'.

 A small note at the top right of the form states: 'Данные из анкеты будут использоваться врачом в качестве первичной медицинской карты' (Data from the form will be used by the doctor as the primary medical record).

Рисунок 3.11 – Анкета пациента. Часть 1

7. Операции с дётой

Операция - дата, ...

8. Курение:

☐ Да

☐ Нет

☐ Иногда

9. Алкоголь:

☐ 1 раз в год

☐ 1 раз в месяц

☐ 1 раз в неделю

☐ более 3 раз в неделю

10. Другие вредные привычки:

Заполните поле, если таковые имеются

11. Была ли процедура по переливанию крови? (Гемотрансфузия)

☐ Да

☐ Нет

12. Переносили ли вы следующие заболевания? (Можно выбрать несколько)

☐ Гепатит

☐ ТСБ

☐ Кожно-венерологические заболевания

Сохранить анкету

Рисунок 3.12 – Анкета пациента. Часть 2

Если пользователь является доктором, то ему необходимо заполнить заявку на врача чтобы модератор мог его подтвердить (Рисунки 3.13-3.14).

MEDICO Специалисты

Евгений К.

Заявка на врача

Данные из анкеты будут использоваться в качестве отображения ваших персональных данных

Фамилия Имя Отчество

Фамилия Имя Отчество

ИИН

XXX.XXX.XXX.XXX

Пол

Мужской

Ваш e-mail

doctor...agyn@gmail.com

Ваш номер телефона

+7 (XXX) XXX XX XX

Специальность

Терапевт

Стаж

1 Месяцев

Загрузить фото

Выбрать файл

Плющистимые флпматы: lplal plal

Рисунок 3.13 – Заявка на врача. Часть 1

Ваш номер телефона

+7 (000) 000 00 00

Специальность

Терапевт

Стаж

1 Месяцев

Загрузить фото

Выбрать файл

Допустимые форматы: jpeg, png

Загрузить резюме

Выбрать файл

Допустимые форматы: jpeg, png, txt, pdf

Загрузить диплом

Выбрать файл

Допустимые форматы: jpeg, png, pdf

Отправить модератору

Врачи  
О сервисе  
Как это работает

Отзывы  
Тарифы

+7 978 123 45 67  
Написать в поддержку

Twitter Facebook Instagram

Рисунок 3.14 – Заявка на врача. Часть 2

После заполнения анкеты пациент имеет возможность выполнить поиск врача с дальнейшей записью на прием (Рисунки 3.15-3.16).

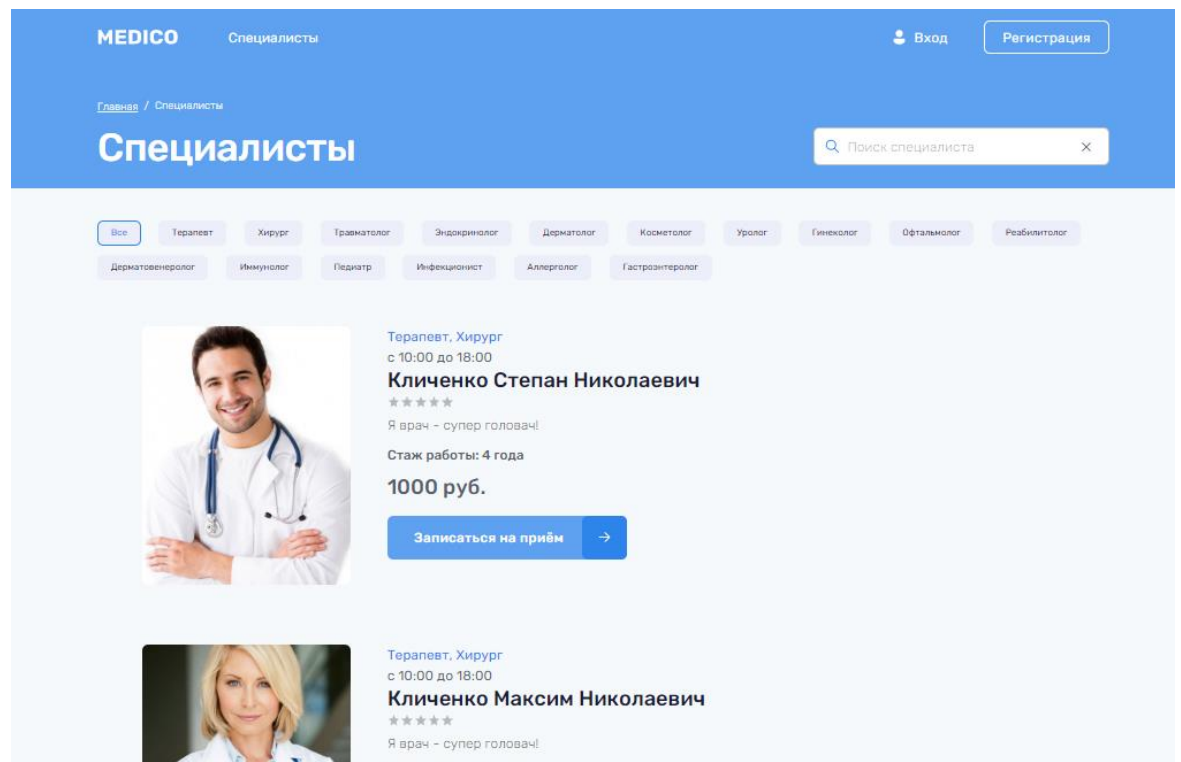


Рисунок 3.15 – Страница со специалистами. Часть 1

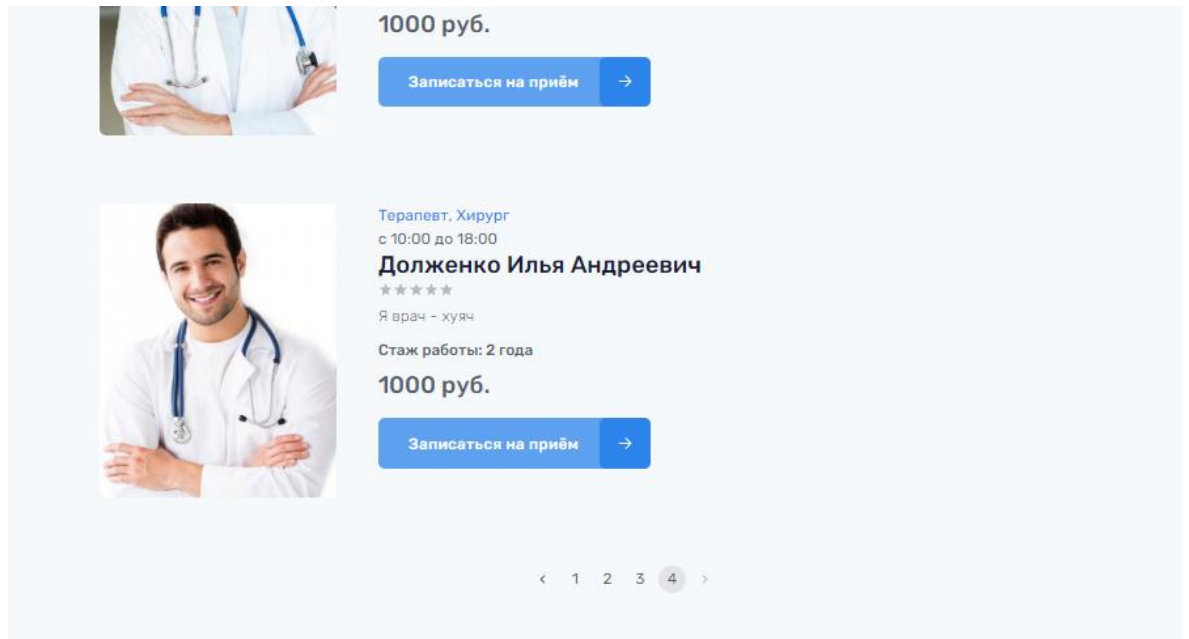


Рисунок 3.16 – Страница со специалистами. Часть 2

На рисунках 3.17-3.18 представлена детальная страница врача, на которой можно посмотреть отзывы пациентов, образование, опыт работы и основные направления по которым работает врач. А также здесь располагается кнопка перехода на процесс записи на прием к выбранному врачу.

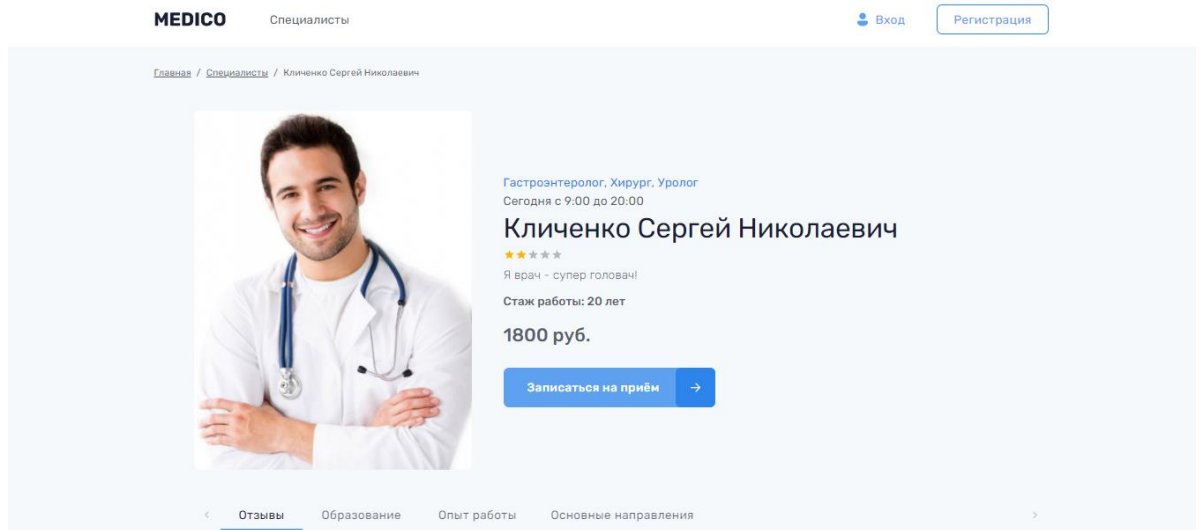


Рисунок 3.17 – Детальная страница врача. Часть 1

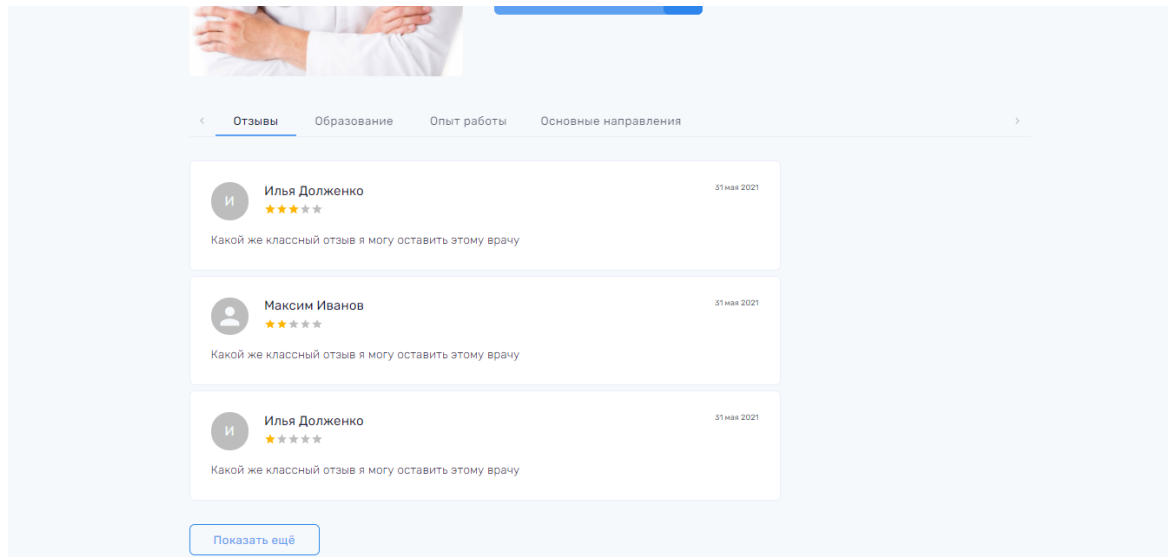


Рисунок 3.18 – Детальная страница врача. Часть 2

Если пользователь авторизован и является пациентом, то кнопка “Записаться на приём” активна и при клике на нее открывается первый этап записи на консультацию с выбором даты, времени, способом связи и необходимой специальностью. Весь процесс записи состоит из четырех этапов. Первый из них представлен на рисунке 3.19.

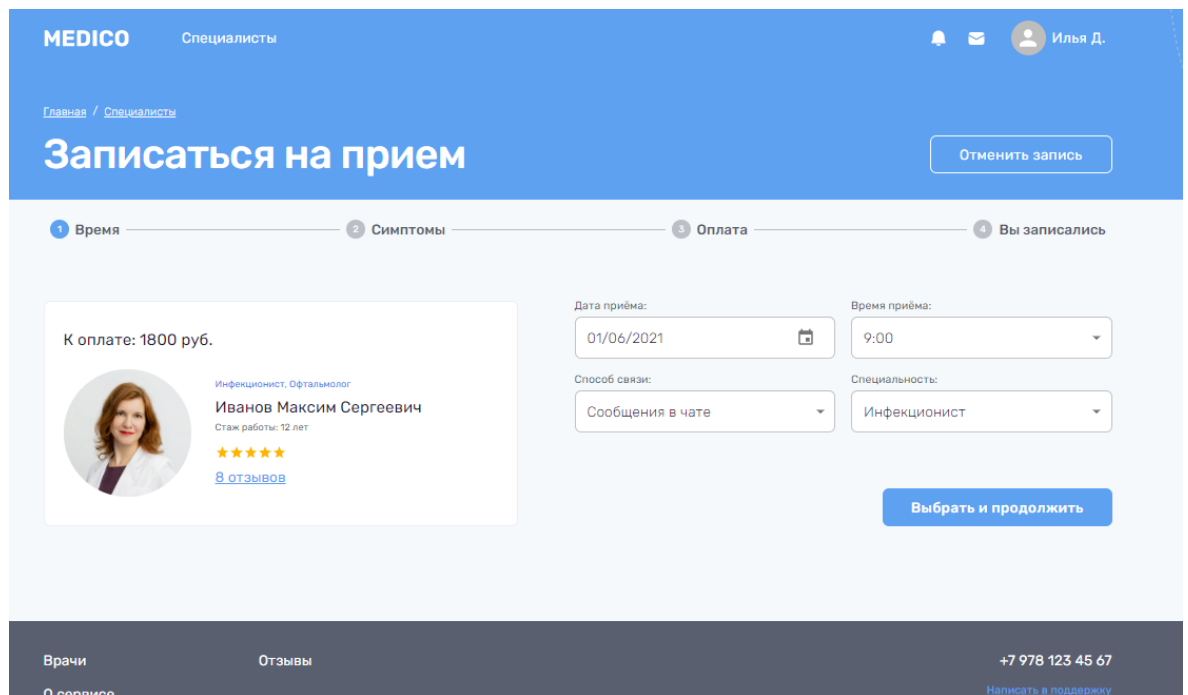


Рисунок 3.19 – Первый этап записи на консультацию

На втором этапе, представленном на рисунке 3.20, пациенту представляется возможность заполнить свои симптомы.

Рисунок 3.20 – Второй этап записи на консультацию

На третьем этапе пациенту предоставляется возможность удостовериться в введенных данных и оплатить консультацию. Данный этап представлен на рисунке 3.21.

Рисунок 3.21 – Третий этап записи на консультацию

На завершающем этапе изображен результат записи на консультацию, также продублирована основная информация записи: имя специалиста, дата, время и способ коммуникации. Завершающий этап изображен на рисунке 3.22.

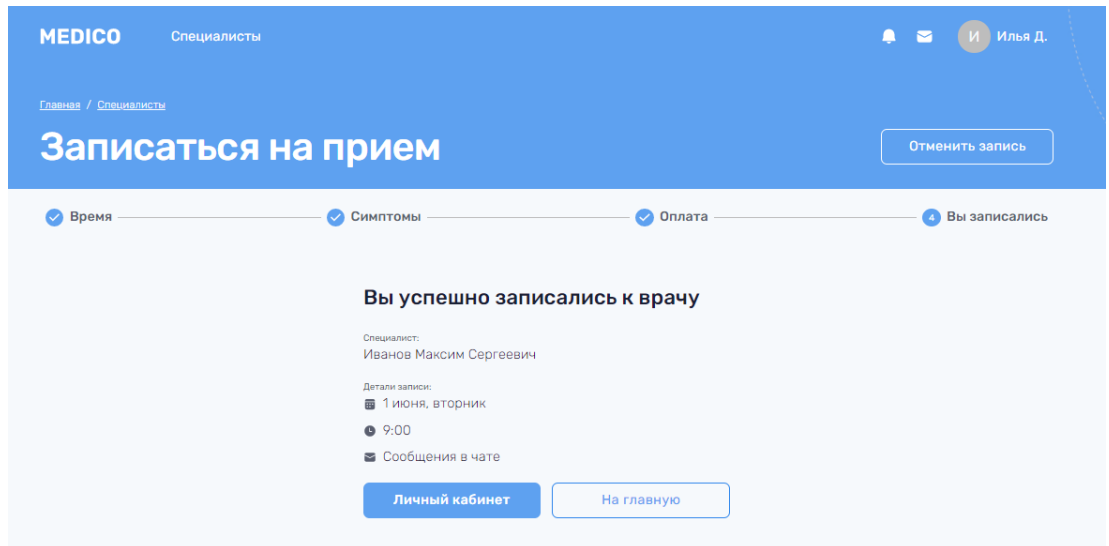


Рисунок 3.22 – Завершающий этап записи на консультацию

Дальнейшие действия пациента представляет из себя работу в личном кабинете, который состоит из следующих страниц: мои записи, назначения, мои анализы, мед. карта, сообщения и настройки.

Страница “Мои записи” (рисунок 3.23) представляет из себя три списка: активные консультации, запланированные и история.

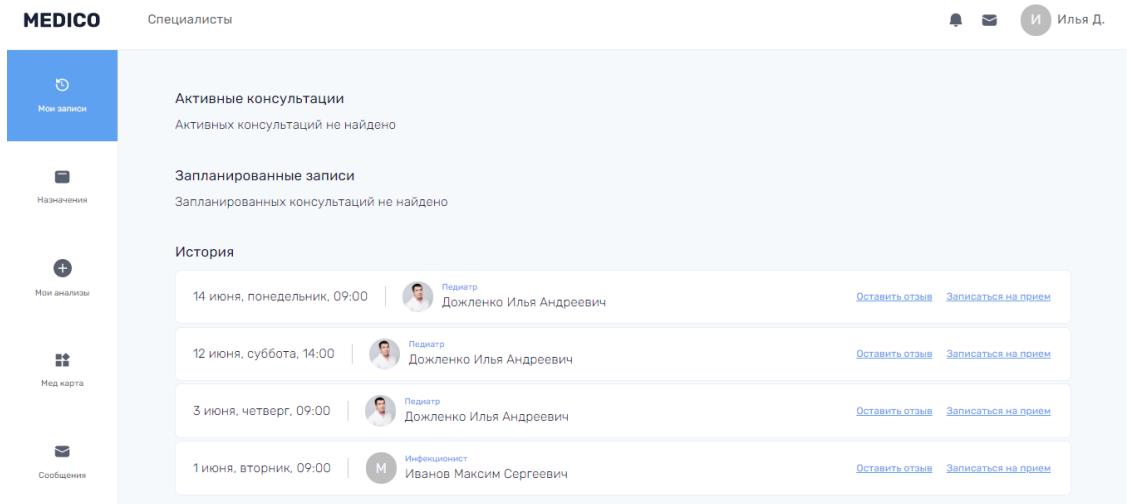


Рисунок 3.23 – Страница записей к специалистам



После консультации пациент имеет возможность оставить отзыв о враче, кликнув на соответствующую кнопку на записи в списке истории. Диалоговое окно отправки отзыва представлено на рисунке 3.24.

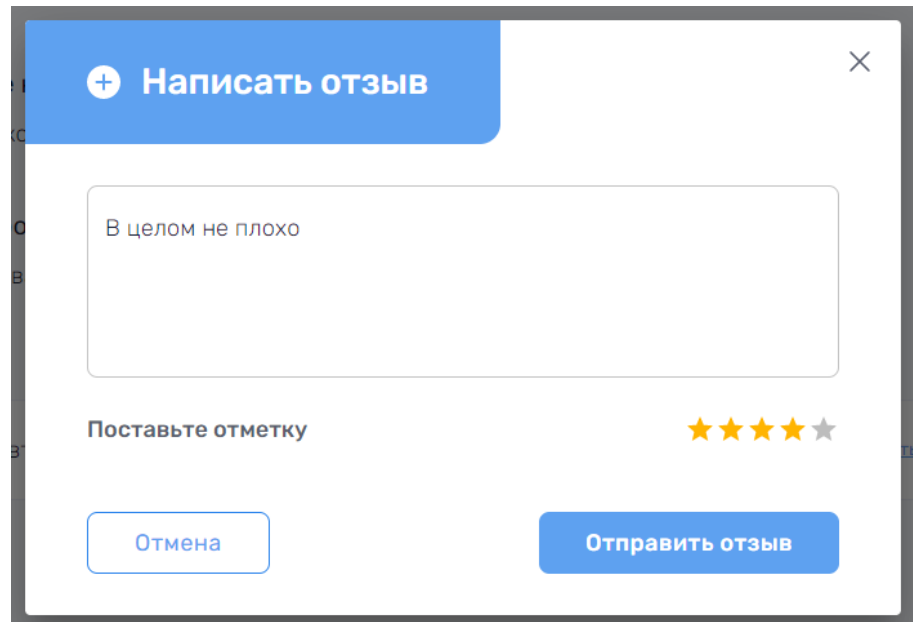


Рисунок 3.24 – Диалоговое окно отправки отзыва о специалисте

Следующая страница “Назначения” содержит список рекомендаций от врача, заполняемых после процесса консультации с пациентом. Данная страница представлена на рисунке 3.25.

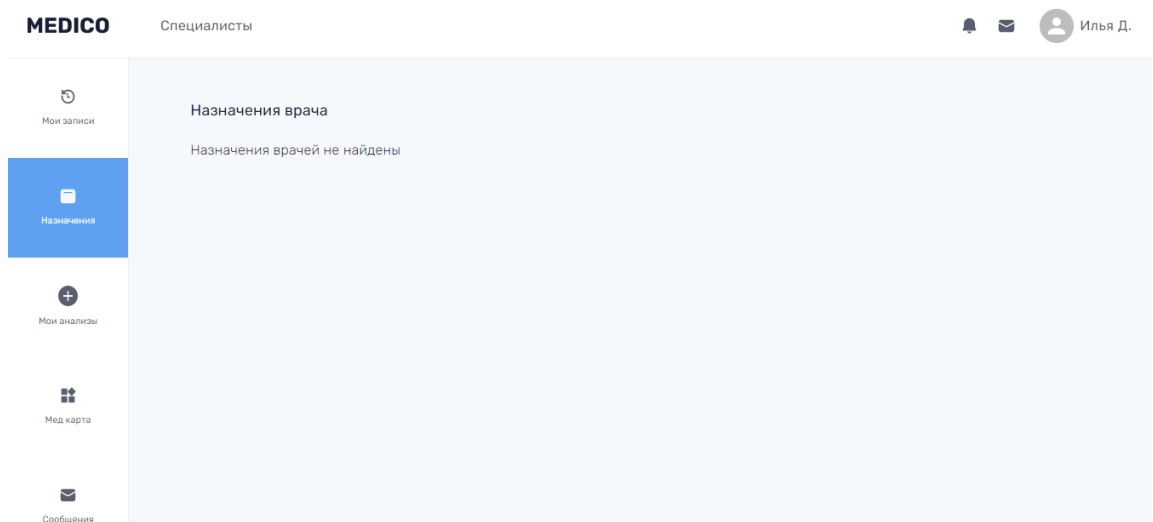


Рисунок 3.25 – Страница назначений

На странице “Мои анализы” пациент имеет возможность загрузить анализ либо снимок. Диалог загрузки файла, представленный на рисунке 3.26, вызывается при клике на кнопку, расположенную правее названия страницы.

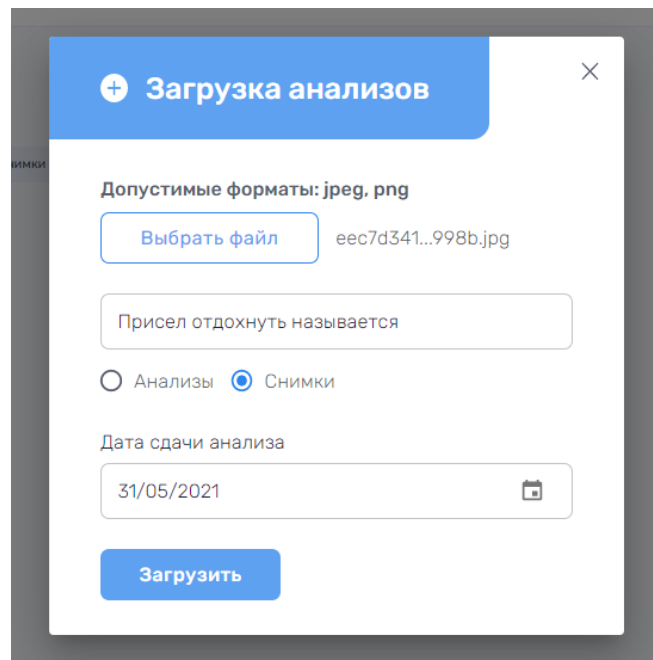


Рисунок 3.26 – Диалог загрузки анализов

После успешной загрузки файла на странице появится загружаемый анализ/снимок с подписью в виде даты сдачи и названия. Также все загружаемые файлы можно сортировать по их типу. Данная страница представлена на рисунке 3.27.

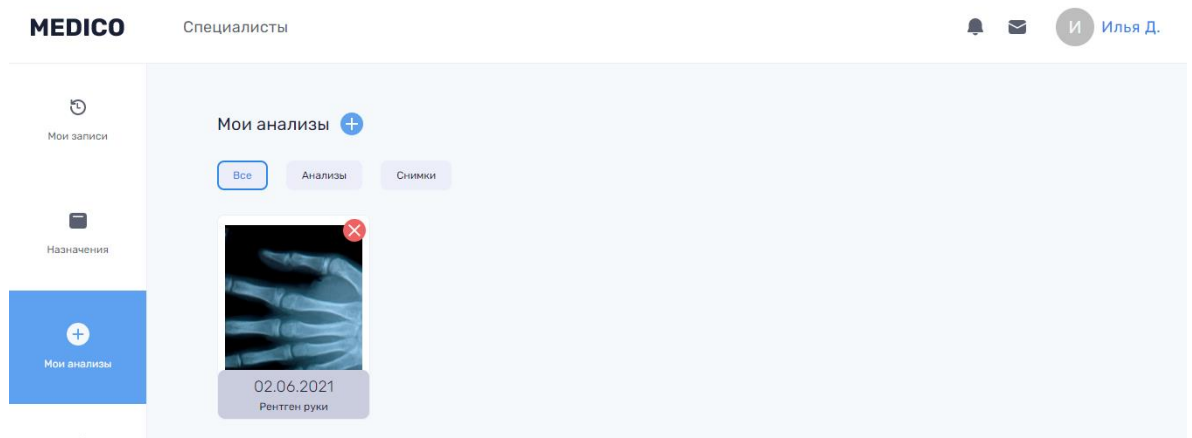


Рисунок 3.27 – Страница анализов

Также загружаемые файлы можно посмотреть более детально, кликнув по изображению анализа. Данная возможность представлена на рисунке 3.28.

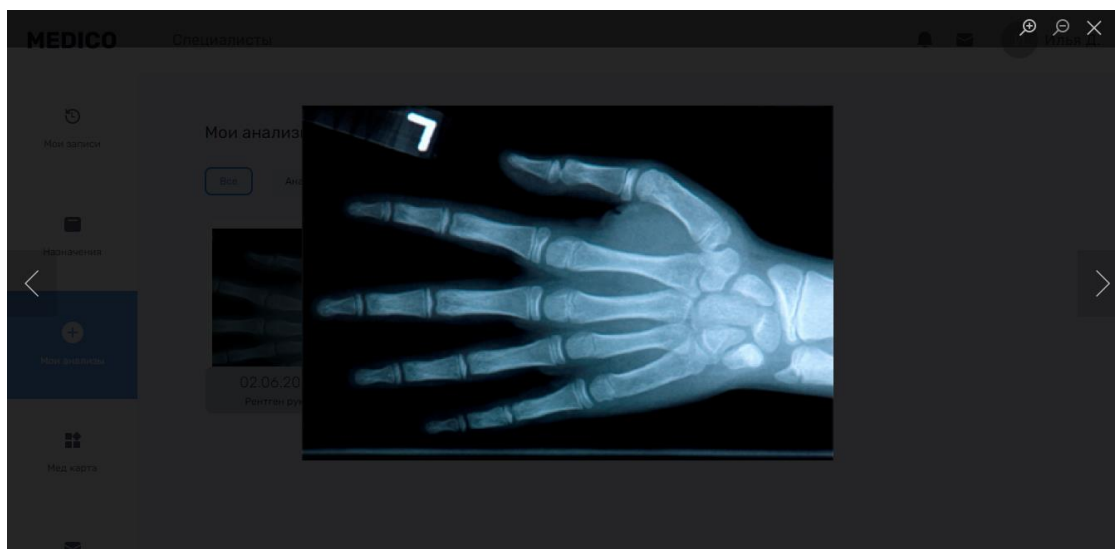


Рисунок 3.28 – Детальный просмотр анализа

Также при клике на кнопку удаления анализа или снимка появится диалоговое окно, внешний вид которого изображен на рисунке 3.29.

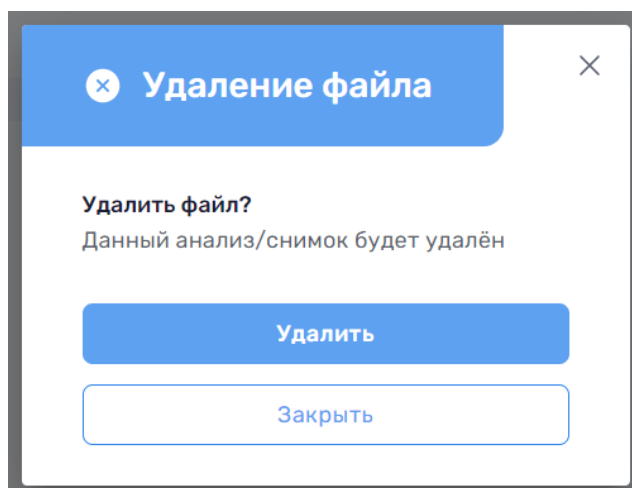


Рисунок 3.29 – Диалоговое окно удаления файла

Следующая страница “Мед. карта”, представленная на рисунке 3.30, предназначена для заполнения информации о себе, такой как: рост, вес, группа крови, аллергия, вредные привычки, переливание крови, операции и

хронические заболевания. Также на данной странице автоматически считается индекс массы тела с соответствующим пояснением.

The screenshot shows a user interface for a patient's medical card. On the left is a sidebar with navigation links: 'Мои записи' (My appointments), 'Назначения' (Prescriptions), 'Мои анализы' (My tests), 'Мед карта' (Medical card - highlighted in blue), 'Сообщения' (Messages), and 'Настройки' (Settings). The main area is titled 'Моя медицинская карта' (My medical card) and displays the patient's name 'Долженко Илья Андреевич' and age '21 лет'. Below this, there are several data cards with edit icons:

- Рост** (Height): 178 см
- Вес** (Weight): 68 кг
- ИМТ в норме** (BMI is normal): 21.46
- Группа крови** (Blood group): II группа, Rh+
- Хронические заболевания** (Chronic diseases): Не заполнено
- Вредные привычки** (Bad habits): Курение - Да, Алкоголь - 1 раз в неделю
- Операции** (Operations): Не заполнено
- Аллергия** (Allergy): Аллергия на диплом
- Переливание крови** (Blood transfusion): Нет

Рисунок 3.30 – Страница медицинской карты пациента

При клике на иконку редактирования открывается диалоговое окно с выбранными полями. Данный процесс представлен на рисунке 3.31.

The screenshot shows a dialog box titled 'Изменение мед. карты' (Change medical card) with a close button (X) in the top right corner. The dialog contains three sections for editing bad habits:

- Курение:** Radio buttons for 'Да' (selected), 'Нет', and 'Иногда'.
- Алкоголь:** Radio buttons for '1 раз в год', '1 раз в месяц', '1 раз в неделю' (selected), and 'более 3 раз в неделю'.
- Другие вредные привычки:** A text input field with the placeholder 'Заполните поле, если таковые имеются'.

At the bottom of the dialog is a blue button labeled 'Изменить' (Change).

Рисунок 3.31 – Процесс изменения мед. Карты

Страница настроек является общей и доступна как для пациента, так и для врача. На данной странице, изображенной на рисунке 3.32, можно изменять как свое имя, фамилию, отчество, так и дату рождения, номер телефона, пол.

Рисунок 3.32 – Общая страница настроек

На главной странице личного кабинета врача находится информация о своем профиле. Имеется возможность просмотра отзывов, основных специальностей, изменение стоимости консультации, образования и опыта работы. Данный профиль врача представлен на рисунке 3.33.

Рисунок 3.33 – Главная страница врача

При нажатии на кнопку редактирования стоимости консультации или информации о себе появляется диалоговое окно с соответствующей формой. Процесс редактирования информации о себе представлено на рисунке 3.34.

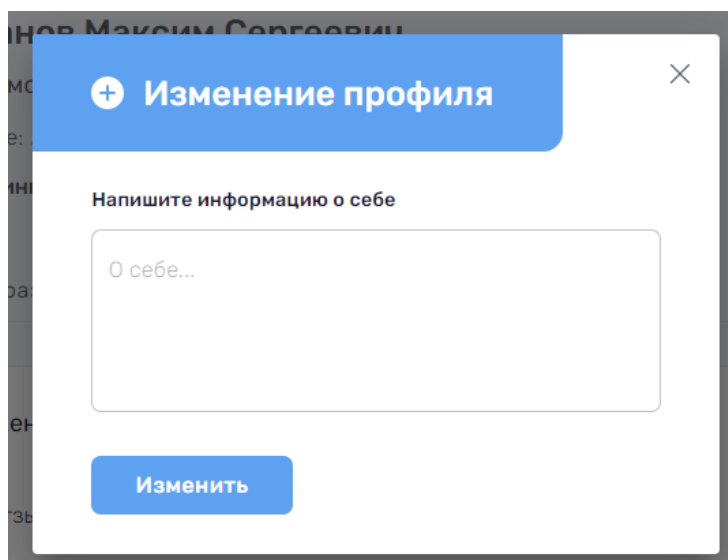
The image shows a mobile application interface with a dialog box titled "Изменение профиля" (Change profile) in a blue header bar. Below the header, the text "Напишите информацию о себе" (Write information about yourself) is displayed. A large text input field contains the placeholder "О себе..." (About me...). At the bottom of the dialog is a blue button labeled "Изменить" (Change). The background shows a blurred view of the user's profile card with the name "Иванов Максим Сергеевич".

Рисунок 3.34 – Форма редактирования “О себе”

Редактирование образования и опыта работы происходит по аналогичному сценарию. Только теперь они содержат динамическое количество записей, т.е. имеется возможность добавить в одной форме сразу несколько записей. Пример редактирования приведен на рисунке 3.35.

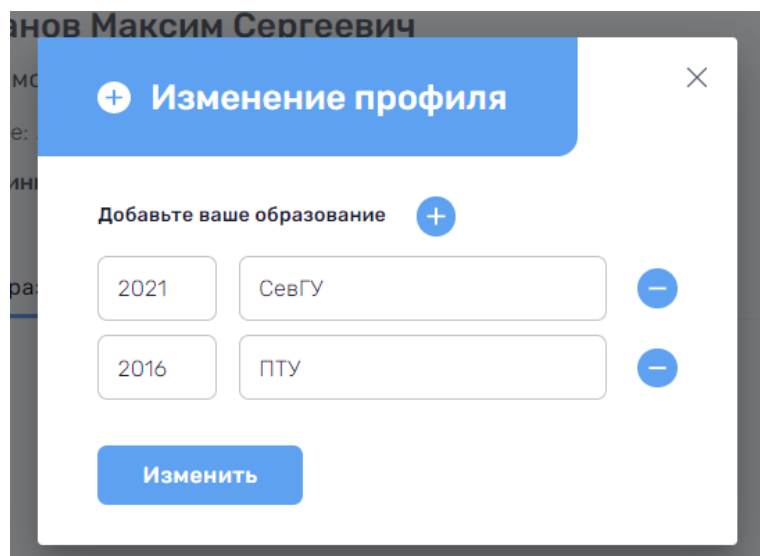
The image shows the same "Изменение профиля" (Change profile) dialog box, but for editing education. Below the header, the text "Добавьте ваше образование" (Add your education) is followed by a blue "+" button. There are two rows of input fields. The first row has a year field with "2021", an institution field with "СевГУ", and a blue "-" button. The second row has a year field with "2016", an institution field with "ПТУ", and a blue "-" button. At the bottom is a blue button labeled "Изменить" (Change). The background shows the same blurred profile card.

Рисунок 3.35 – Пример редактирования образования

На следующей странице “График работы” (рисунок 3.36) врач может изменить время своего приема по дням, либо установить дни без приема.

**MEDICO** Специалисты

Главная

График работы

Пациенты

Сообщения

Настройки

### График работы

Понедельник	Вторник	Среда
с 9:00	с 9:00	с 9:00
по 20:00	по 20:00	по 20:00

Четверг	Пятница	Суббота
с 9:00	с 9:00	с 10:00
по 20:00	по 20:00	по 20:00

**Воскресенье**

с 10:00
по 18:00

**Сохранить изменения**

Рисунок 3.36 – Страница редактирования графика работы

Страница “Пациенты” (рисунок 3.37) разделена на два блока: список с пациентами по категории (новые или история) и календарь с выбором даты, по которой необходимо отображать пациентов.

**MEDICO** Специалисты

Главная

График работы

Пациенты

Сообщения

Максим И.

### Пациенты

Новые История

Пациенты на 01 июня

Первичная консультация	01 июня 2021	Илья Долженко
до начала консультации 9 часов, 30 минут, 11 секунд	09:00 – 10:00	Сообщения в чате

июня 2021

пн	вт	ср	чт	пт	сб	вс
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Рисунок 3.37 – Страница просмотра пациентов

При клике на пациента открывается страница просмотра его профиля, которая содержит его медицинскую карту, симптомы, анализы и прошлые назначения врачей. Страница профиля пациента представлена на рисунке 3.38.

The screenshot shows a patient profile for 'Долженко Илья Андреевич' (Dolzenko Ilya Andreevich), born 16.08.2000. The page has tabs for 'Мед. карта' (Medical card), 'Анализы' (Analyses), and 'Назначения' (Prescriptions). The 'Мед. карта' tab is active, showing a 'Назначить' (Prescribe) button. The profile includes a 'Симптомы' (Symptoms) section with the text 'Очень сильно болит голова! Помогите!!!' (Headache is very strong! Help!!!). Below this are several data fields: 'Рост' (Height) 178 см, 'Вес' (Weight) 68 кг, 'ИМТ в норме' (BMI is normal), and 'ИМТ' (BMI) 21.46. There are also sections for 'Группа крови' (Blood group) II группа, Rh+, 'Хронические заболевания' (Chronic diseases) (Not filled), 'Вредные привычки' (Bad habits) (Smoking - Yes, Alcohol - 1 time a week), 'Операции' (Operations) (Not filled), 'Аллергия' (Allergy) (Allergy on diploma), and 'Переливание крови' (Blood transfusion) (No).

Рисунок 3.38 – Страница просмотра профиля пациента

На данной странице также находится кнопка назначения рекомендаций, при клике на которую открывается диалоговое окно с текстовым полем (рисунок 3.39).

The screenshot shows a dialog box titled 'Назначение врача' (Assign doctor) with a close button (X). Inside the dialog is a text input field containing the text 'Парацетамол - по 1 таблетке - 3 раза в день, ...' (Paracetamol - 1 tablet 3 times a day, ...). Below the input field is a blue button labeled 'Назначить' (Assign).

Рисунок 3.39 – Окно назначения рекомендаций пациенту



Также на странице находится кнопка “связаться с пациентом”, при клике на которую открывается диалог на общей странице чата. Страница “Сообщения” с выбранным диалогом изображена на рисунке 3.40.

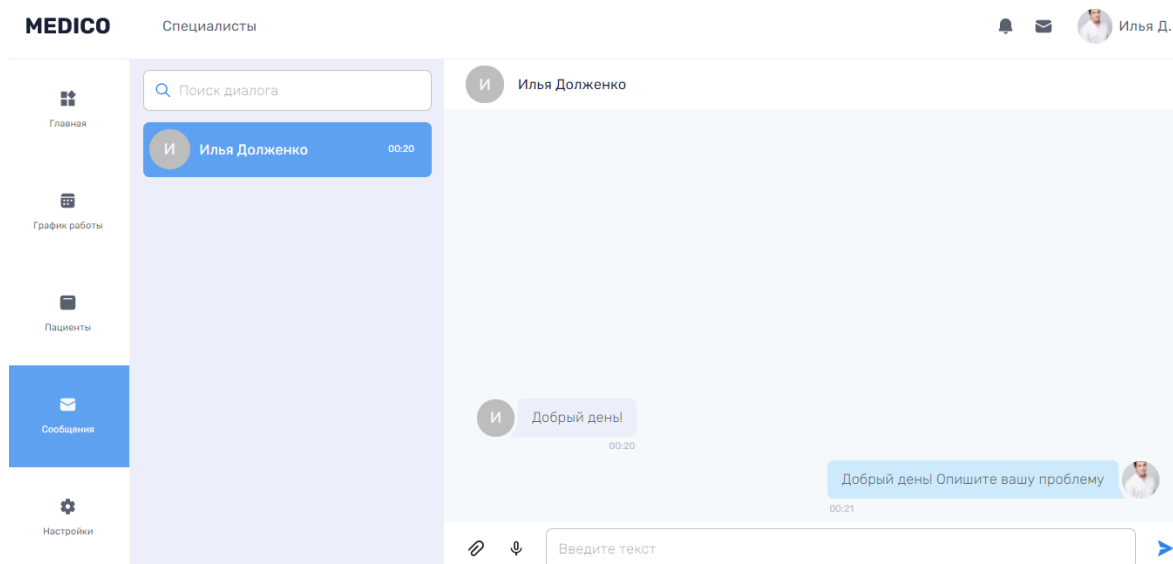


Рисунок 3.40 – Страница диалога с пациентом

В случае если диалог не выбран, то на странице отображается анимированное изображение с текстом “Выберите диалог”. Данная ситуация представлена на рисунке 3.41.

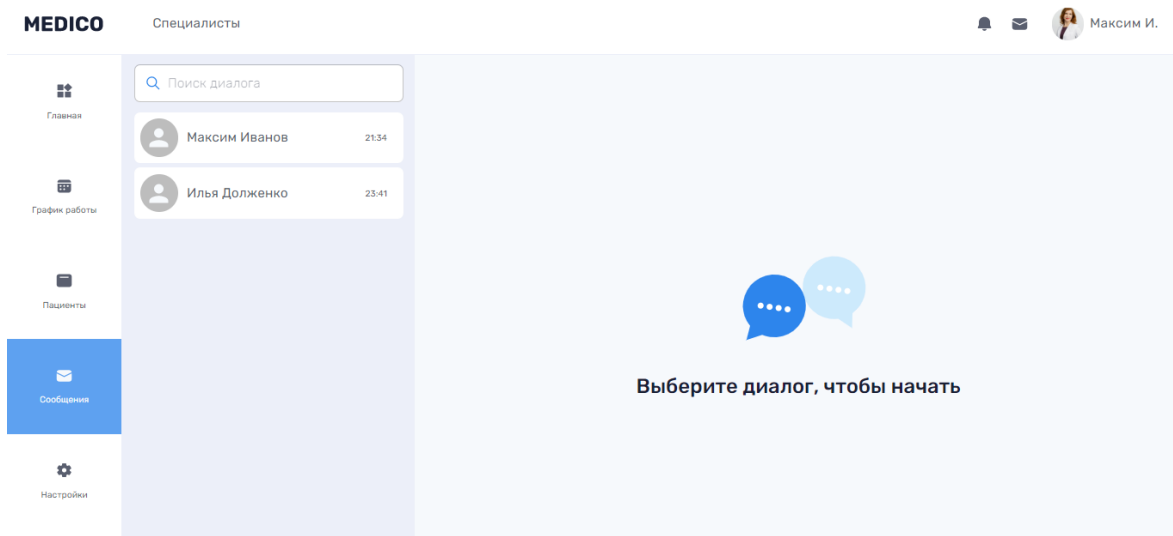


Рисунок 3.41 – Страница сообщений с невыбранным диалогом

Та же страница, но уже со стороны пациента имеет некоторое отличие – под именем врача в диалогах и верхней части сообщений написаны специальности, которыми обладает врач. Данный вид страницы представлен на рисунке 3.42.

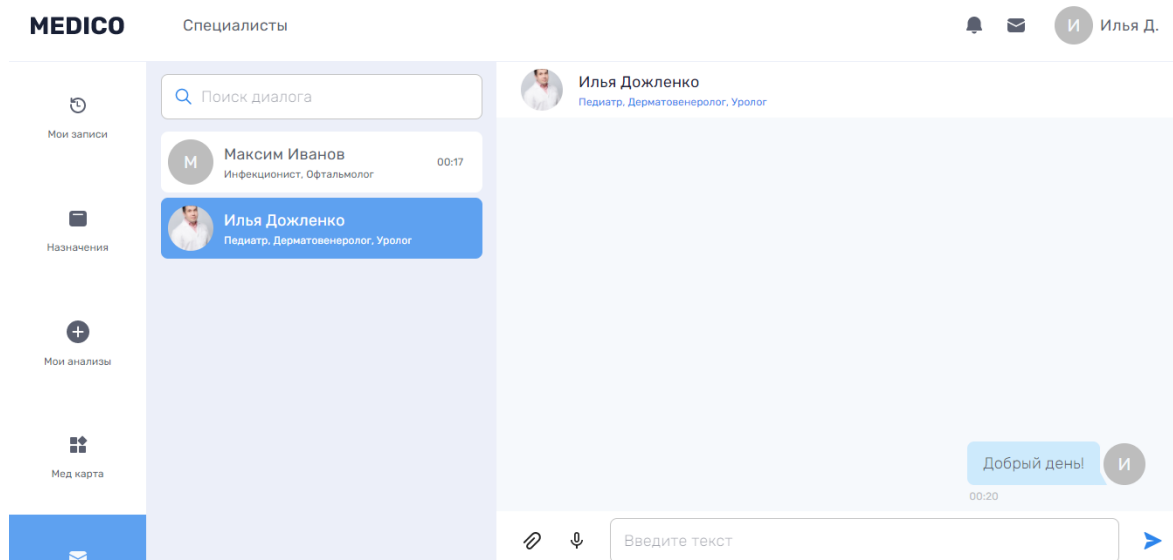


Рисунок 3.42 – Страница сообщений со стороны пациента

При общении в чате как пациент, так и врач имеют возможность отправить:

- фото (рисунок 3.43);

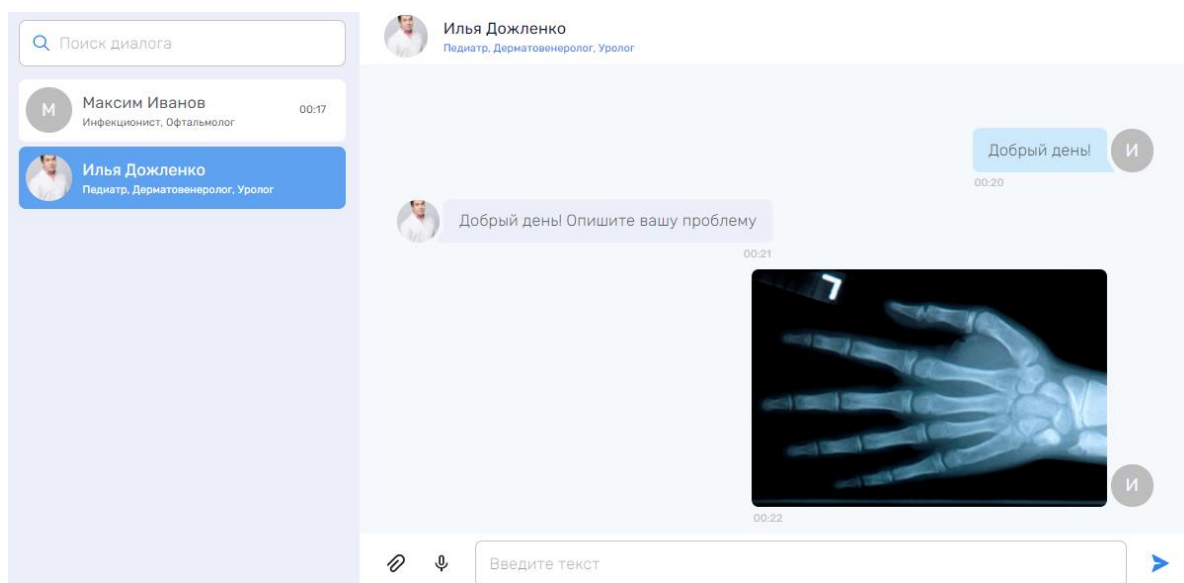


Рисунок 3.43 – Пример отправки изображения в чат

- аудиосообщение (рисунок 3.44);

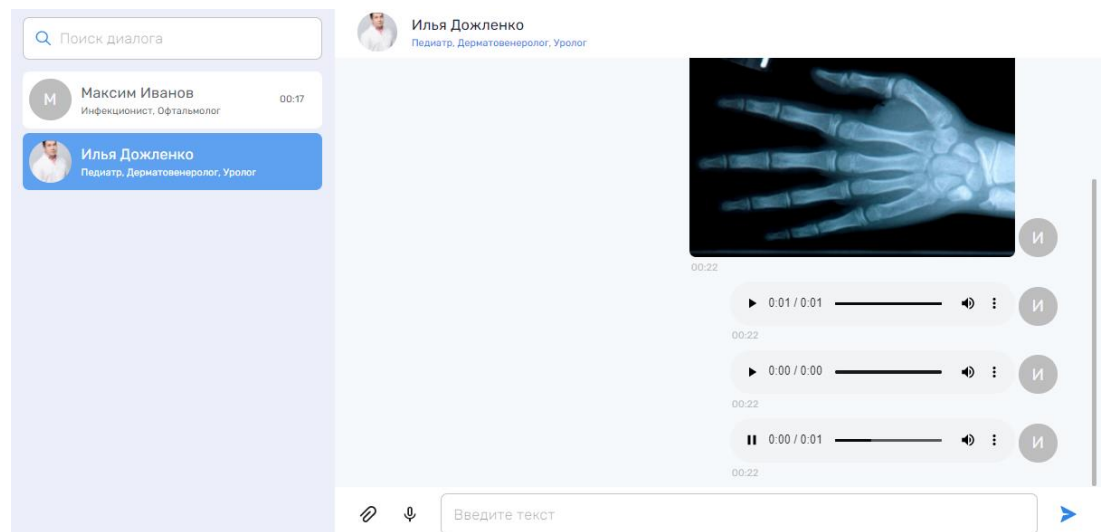


Рисунок 3.44 – Пример отправки аудиосообщения

- файл (рисунок 3.45).

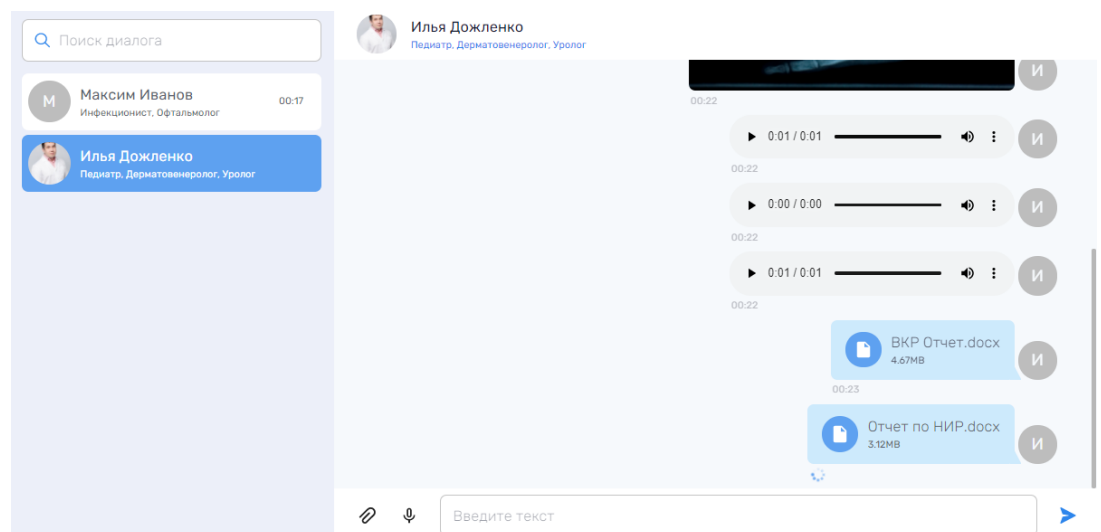


Рисунок 3.45 – Пример отправки файла

### Выводы по разделу 3

В данном разделе была проведена разработка программных модулей web-приложения и разработан пользовательский интерфейс системы для онлайн-консультаций с врачами.

## ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе бакалавра разработана клиентская часть системы для онлайн-консультаций с врачами. Клиентская часть полностью соответствует техническому заданию на разработку подсистемы.

В пояснительной записке освещены все аспекты создания клиентской части Web-приложения, начиная от исследования информационных процессов предметной области и заканчивая описанием разработанных программных модулей и внешнего вида клиентской части приложения.

Результатом разработки проекта стало web-приложение, призванное помочь пациентам, желающим получить медицинскую консультацию, не выходя из дома, а также для врачей, осуществляющих эту консультацию за оплату.

Особенностью разработанной клиентской части является:

1. Возможность для специалистов попробовать себя в сфере телемедицины и пройдя верификацию устроиться на работу.
2. Гибкое регулирование рабочего графика специалистом с указанием стоимости своей консультации.
3. Предоставление пациентам возможности хранения анализов и введения своей медицинской карты.
4. Широкие возможности по поиску специалиста для записи на консультацию по различным фильтрам, рейтингу и отзывам.
5. Удобный интерфейс и легкость освоения.
6. Кроссплатформенность и адаптивность. Возможность одновременного доступа с персональных и мобильных устройств.

При разработке проекта использован ряд различных технологий, фреймворков, библиотек и инструментов, в числе которых:

- типизированный язык TypeScript, расширяющий возможности JavaScript;

- библиотека для построения пользовательских интерфейсов React с менеджером управления состоянием приложения MobX;
- метаязык на основе CSS - SASS;
- JavaScript библиотека для обмена данными в реальном времени - Socket.IO.

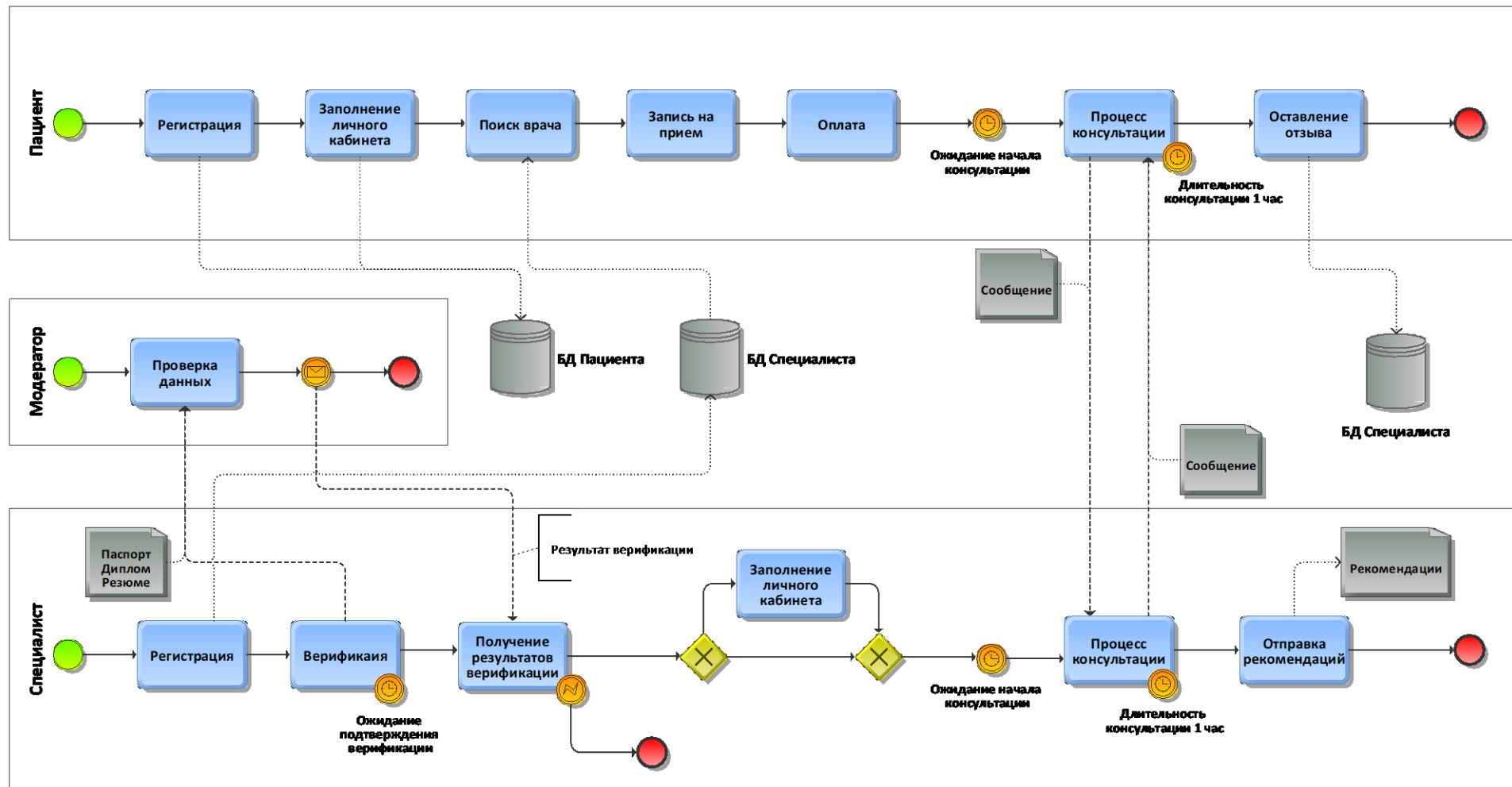
В дальнейшем планируется внедрение web-приложения и добавление новых функциональных возможностей.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. React – библиотека для построения пользовательских интерфейсов [Электронный ресурс]. URL: <https://reactjs.org/> (дата обращения: 05.02.2021).
2. TypeScript – типизированный язык, расширяющий возможности JavaScript [Электронный ресурс]. URL: <https://www.typescriptlang.org/> (дата обращения: 05.02.2021).
3. Create React App – инструмент для быстрого создания React-приложений. [Электронный ресурс]. URL: <https://create-react-app.dev/> (дата обращения: 05.02.2021).
4. Документация платформы NodeJS [Электронный ресурс]. URL: <https://nodejs.org/en/docs/> (дата обращения: 05.02.2021).
5. MobX – библиотека для управления состоянием React-приложения [Электронный ресурс]. URL: <https://mobx.js.org/README.html> (дата обращения: 05.02.2021).
6. Socket.io – JavaScript - библиотека предоставляющая интерфейс взаимодействия с WebSocket [Электронный ресурс]. URL: <https://socket.io/docs/v4/> (дата обращения: 24.04.2021).
7. Запись звука JS с микрофона или голосовые комментарии [Электронный ресурс]. URL: <https://habr.com/ru/post/484196/> (дата обращения: 02.05.2021).
8. Что такое DFD (диаграммы потоков данных) [Электронный ресурс]. URL: <https://habr.com/ru/company/trinion/blog/340064/> (дата обращения: 18.06.2021).
9. Atomic Design in React [Электронный ресурс]. URL: <https://medium.com/engineering-zemoso/atomic-design-in-react-react-native-using-a-theming-library-part-1-4fc2e0e2ccc8> (дата обращения: 18.06.2021)

## ПРИЛОЖЕНИЕ А

### Моделирование бизнес-процессов с помощью методологии BPMN



## ПРИЛОЖЕНИЕ Б

### Код программы

index.tsx:

```
import React from "react";
import ReactDOM from "react-dom";
import { Router } from "react-router-dom";
import { createBrowserHistory, History } from "history";
import { MuiThemeProvider } from "@material-ui/core";
import { MuiPickersUtilsProvider } from "@material-ui/pickers";
import DateFnsUtils from "@date-io/date-fns";
import ruLocale from "date-fns/locale/ru";

import { App } from "../App";
import { RootStore } from "../stores";
import { StoreProvider } from "../stores/useStore";
import { main } from "../styles/material";
import "../styles/index.scss";

const history: History = createBrowserHistory();
const rootStore = new RootStore(history);

ReactDOM.render(
  <StoreProvider store={rootStore}>
    <MuiThemeProvider theme={main}>
      <MuiPickersUtilsProvider utils={DateFnsUtils} locale={ruLocale}>
        <Router history={history}>
          <App />
        </Router>
      </MuiPickersUtilsProvider>
    </MuiThemeProvider>
  </StoreProvider>,
  document.getElementById("root")
);
```

App.tsx:

```
export const App: React.FC = observer(() => {
  const { userStore } = useStores();
  const { isAuthorized, pending, fetchUser } = userStore;

  useEffect(() => {
    if (localStorage.getItem("accessToken")) {
      fetchUser();
    }
  }, [fetchUser]);

  if (pending) {
    return <Backdrop />;
  }

  return (
    <React.Fragment>
      <CssBaseline />
      <ScrollHandler />
      <Drawer />
    </React.Fragment>
  );
});
```



```

    <Switch>
      <Route
        exact
        path={['/', '/home', '/sign-up-confirmation']}
        component={HomePage}
      />
      <Route
        exact
        path={['
          "/doctors",
          "/doctors/:specialty",
          "/doctors/:specialty/:page"
        ]}
        component={DoctorsPage}
      />
      <Route exact path="/doctor/:id" component={DoctorPage} />
      <PrivateRoute
        exact
        path="/sign-up"
        component={SignUpPage}
        canRoute={!isAuthorized}
      />
      <PrivateRoute
        exact
        path="/appointment/:id"
        component={AppointmentPage}
        canRoute={isAuthorized}
      />
      <PrivateRoute
        exact
        path="/questionnaire"
        component={QuestionnairePage}
        canRoute={isAuthorized}
      />
      <PrivateRoute
        path="/dashboard"
        component={DashboardPage}
        canRoute={isAuthorized}
      />
      <Route component={ErrorPage} />
    </Switch>

    <DialogSignIn />
    <DialogEmail />
  </React.Fragment>
);
});

```

## RootStore.ts:

```

export class RootStore implements IStores {
  routerStore: IRouterStore;
  signUpStore: ISignUpStore;
  signInStore: ISignInStore;
  userStore: IUserStore;
  modalsStore: IModalsStore;
  questionnaireStore: IQuestionnaireStore;
  specialtiesStore: ISpecialtiesStore;
  doctorStore: IDoctorStore;
  searchDoctorStore: ISearchDoctorStore;
  homeStore: IHomeStore;
  drawerStore: IDrawerStore;

```

```

appointmentStore: IAppointmentStore;
commentStore: ICommentStore;
dashboardConsultationsStore: IDashboardConsultationsStore;
dashboardAnalyzesStore: IDashboardAnalyzesStore;
dashboardResultsStore: IDashboardResultsStore;
dashboardMedicalCardStore: IDashboardMedicalCardStore;
dashboardSettingsStore: IDashboardSettingsStore;
dashboardDoctorProfileStore: IDashboardDoctorProfileStore;
dashboardScheduleStore: IDashboardScheduleStore;
dashboardPatientsStore: IDashboardPatientsStore;
dashboardPatientInfoStore: IDashboardPatientInfoStore;
chatStore: IChatStore;
socketsStore: ISocketsStore;

constructor(history: History) {
  this.routerStore = new RouterStore(history);
  this.signUpStore = new SignUpStore(this);
  this.signInStore = new SignInStore(this);
  this.userStore = new UserStore(this);
  this.modalsStore = new ModalsStore();
  this.questionnaireStore = new QuestionnaireStore(this);
  this.specialtiesStore = new SpecialtiesStore();
  this.doctorStore = new DoctorStore();
  this.searchDoctorStore = new SearchDoctorStore(this);
  this.homeStore = new HomeStore();
  this.drawerStore = new DrawerStore();
  this.appointmentStore = new AppointmentStore();
  this.commentStore = new CommentStore(this);
  this.dashboardConsultationsStore = new DashboardConsultationsStore();
  this.dashboardAnalyzesStore = new DashboardAnalyzesStore(this);
  this.dashboardResultsStore = new DashboardResultsStore();
  this.dashboardMedicalCardStore = new DashboardMedicalCardStore(this);
  this.dashboardSettingsStore = new DashboardSettingsStore(this);
  this.dashboardDoctorProfileStore = new DashboardDoctorProfileStore(this);
  this.dashboardScheduleStore = new DashboardScheduleStore();
  this.dashboardPatientsStore = new DashboardPatientsStore();
  this.dashboardPatientInfoStore = new DashboardPatientInfoStore(this);
  this.chatStore = new ChatStore(this);
  this.socketsStore = new SocketsStore(this);
}
}

```

## SignUpStore.ts:

```

const INITIAL_SIGN_UP_FORM: ISignUpForm = {
  userType: "patient",
  firstName: "",
  lastName: "",
  middleName: "",
  birthDate: new Date(),
  gender: "male",
  phoneNumber: "",
  email: "",
  password: "",
  acceptedUserAgreement: false
};

const INITIAL_SIGN_UP_FORM_ERRORS: ISignUpFormErrors = {
  firstName: undefined,
  lastName: undefined,
  birthDate: undefined,
  phoneNumber: undefined,

```

```

    email: undefined,
    password: {
      isLength: false,
      isUppercase: false,
      isLowercase: false,
      isNumber: false
    }
  };

export class SignUpStore implements ISignUpStore {
  signUpForm = INITIAL_SIGN_UP_FORM;

  signUpFormErrors = INITIAL_SIGN_UP_FORM_ERRORS;

  submissionError: string | undefined = undefined;

  pending: boolean = false;

  sentEmail: boolean = false;

  private rootStore: IStores;

  constructor(rootStore: IStores) {
    this.rootStore = rootStore;

    makeObservable(this, {
      signUpForm: observable,
      signUpFormErrors: observable,
      submissionError: observable,
      pending: observable,
      sentEmail: observable,
      doSignUp: action,
      setFormValue: action,
      validateForm: action,
      resetForm: action
    });

    reaction(
      () => this.signUpForm.lastName,
      lastName =>
        lastName &&
        (this.signUpFormErrors.lastName = isOnlyLetters(lastName))
    );

    reaction(
      () => this.signUpForm.firstName,
      firstName =>
        firstName &&
        (this.signUpFormErrors.firstName = isOnlyLetters(firstName))
    );

    reaction(
      () => this.signUpForm.birthDate,
      birthDate =>
        birthDate.toLocaleDateString() !== new Date().toLocaleDateString
    ) &&
      (this.signUpFormErrors.birthDate = isAdult(birthDate))
    );

    reaction(
      () => this.signUpForm.phoneNumber,
      phoneNumber =>
        phoneNumber &&

```

```

        (this.signUpFormErrors.phoneNumber = isPhoneNumber(phoneNumber))
    );

    reaction(
        () => this.signUpForm.email,
        email => email && (this.signUpFormErrors.email = isEmail(email))
    );

    reaction(
        () => this.signUpForm.password,
        password =>
            password && (this.signUpFormErrors.password = isPassword(password))
    );
}

doSignUp = () => {
    if (!this.validateForm()) {
        return;
    }

    this.pending = true;
    this.submissionError = undefined;

    const postData: ISignUpPostData = {
        userType: this.signUpForm.userType,
        name: this.signUpForm.firstName,
        surname: this.signUpForm.lastName,
        middleName: this.signUpForm.middleName,
        birthDate: this.signUpForm.birthDate.toISOString(),
        sex: this.signUpForm.gender,
        phone: this.signUpForm.phoneNumber,
        email: this.signUpForm.email,
        password: this.signUpForm.password,
        acceptedUserAgreement: this.signUpForm.acceptedUserAgreement
    };

    SignUpApi.signUp(postData)
        .then(
            action(() => {
                this.rootStore.modalsStore.setModalIsOpen("email", true);
            })
        )
        .catch(
            action((error: AxiosError<ISignUpErrorResponse>) => {
                this.submissionError = error.response?.data.message;
            })
        )
        .finally(
            action(() => {
                this.pending = false;
            })
        );
};

sendMail = () => {
    const postData: ISendMailPostData = {
        email: this.signUpForm.email
    };

    SignUpApi.sendMail(postData).then(
        action(() => {
            this.sentEmail = true;
        })
    );
};

```

```

        ))
    );
};

validateForm = () => {
    this.signUpFormErrors = {
        ...this.signUpFormErrors,
        firstName: isOnlyLetters(this.signUpForm.firstName),
        lastName: isOnlyLetters(this.signUpForm.lastName),
        birthDate: isAdult(this.signUpForm.birthDate),
        phoneNumber: isPhoneNumber(this.signUpForm.phoneNumber),
        email: isEmail(this.signUpForm.email),
        password: isPassword(this.signUpForm.password)
    };

    return Boolean(
        !(
            this.signUpFormErrors.firstName ||
            this.signUpFormErrors.lastName ||
            this.signUpFormErrors.birthDate ||
            this.signUpFormErrors.phoneNumber ||
            this.signUpFormErrors.email ||
            !this.signUpFormErrors.password.isLength ||
            !this.signUpFormErrors.password.isUppercase ||
            !this.signUpFormErrors.password.isLowercase ||
            !this.signUpFormErrors.password.isNumber
        )
    );
};

setFormValue = <K extends KeysOfSignUpForm>(key: K, value: ISignUpForm[K]) =
> {
    this.signUpForm[key] = value;
};

resetForm = () => {
    this.signUpForm = INITIAL_SIGN_UP_FORM;
    this.signUpFormErrors = INITIAL_SIGN_UP_FORM_ERRORS;
    this.submissionError = undefined;
};
}

```

### SignInStore.ts:

```

const INITIAL_SIGN_IN_FORM: ISignInForm = {
    email: "",
    password: ""
};

export class SignInStore implements ISignInStore {
    signInForm = INITIAL_SIGN_IN_FORM;

    submissionError: string | undefined = undefined;

    pending: boolean = false;

    private rootStore: IStores;

    constructor(rootStore: IStores) {
        this.rootStore = rootStore;

        makeObservable(this, {

```

```

        signInForm: observable,
        submissionError: observable,
        pending: observable,
        doSignIn: action,
        setFormValue: action,
        resetForm: action
    });
}

doSignIn = () => {
    this.pending = true;
    this.submissionError = undefined;

    const postData: ISignInPostData = {
        email: this.signInForm.email.toLowerCase(),
        password: this.signInForm.password.trim()
    };

    SignInApi.signIn(postData)
        .then(
            action(({ data }: AxiosResponse<ISignInSuccessResponse>) => {
                localStorage.setItem("accessToken", data.data.accessToken);
                this.rootStore.userStore.fetchUser();
                this.resetForm();
                this.rootStore.modalsStore.setModalIsOpen("sign-in", false);
            })
        )
        .catch(
            action((error: AxiosError<ISignInErrorResponse>) => {
                this.submissionError = error.response?.data.message;
            })
        )
        .finally(
            action(() => {
                this.pending = false;
            })
        );
};

setFormValue = <K extends KeysOfSignInForm>(key: K, value: ISignInForm[K]) => {
    this.signInForm[key] = value;
};

resetForm = () => {
    this.signInForm = INITIAL_SIGN_IN_FORM;
    this.submissionError = undefined;
};
}

```

### SocketsStore.ts:

```

export class SocketsStore {
    socket: Socket = io(
        process.env.REACT_APP_API_BASE_URL || "http://localhost:3003",
        {
            autoConnect: false
        }
    );

    private rootStore: IStores;
}

```

```

constructor(rootStore: IStores) {
    this.rootStore = rootStore;

    makeObservable(this, {
        socket: observable,
        initSocket: action,
        sendMessage: action
    });
}

initSocket = (accessToken: string) => {
    this.socket.io.opts.query = {
        token: accessToken
    };
    this.socket.connect();

    this.socket.on("disconnect", () => {
        alert(
            "Соединение прервано. Закройте все вкладки с данным приложением
и обновите страницу"
        );
    });

    this.socket.on("newMessage-success", (data: Message) => {
        this.rootStore.chatStore.appendMessage(data);
    });
};

sendMessage = (data: SendMessageSocketData) => {
    this.socket.emit("newMessage", data);
};
}

```

### UserStore.ts:

```

export class UserStore implements IUserStore {
    currentUser: IUser | undefined = undefined;

    isAuthorized: boolean = !!localStorage.getItem("accessToken");

    pending: boolean = false;

    private rootStore: IStores;

    constructor(rootStore: IStores) {
        this.rootStore = rootStore;

        makeObservable(this, {
            currentUser: observable,
            isAuthorized: observable,
            pending: observable,
            fetchUser: action,
            doLogout: action
        });
    }

    fetchUser = () => {
        this.pending = true;

        UserApi.refreshToken().then(
            action(({ data }: AxiosResponse<ISignInSuccessResponse>) => {
                localStorage.setItem("accessToken", data.data.accessToken);
            })
        );
    };
}

```

```

        this.rootStore.socketsStore.initSocket(data.data.accessToken);
    })
    );

    UserApi.getUser()
        .then(
            action(({ data }: AxiosResponse<IGetUserSuccessResponse>) => {
                this.currentUser = data.data;
                this.isAuthorized = true;

                if (data.data.additionalData) {
                    if (data.data.userType === "patient") {
                        this.rootStore.dashboardMedicalCardStore.setChangeCa
rdForm(
                            data.data.additionalData
                        );
                    } else {
                        this.rootStore.dashboardDoctorProfileStore.setDoctor
ProfileForm(
                            data.data.additionalData
                        );
                    }
                }

                this.rootStore.dashboardSettingsStore.setUpdateInfoForm(
                    data.data
                );
            })
        )
        .catch(
            action(() => {
                localStorage.removeItem("accessToken");
                this.isAuthorized = false;
            })
        )
        .finally(
            action(() => {
                this.pending = false;
            })
        );
    };

    doLogout = () => {
        this.currentUser = undefined;
        this.isAuthorized = false;
        localStorage.removeItem("accessToken");
        this.rootStore.chatStore.resetAll();
        this.rootStore.dashboardPatientsStore.resetAll();
        this.rootStore.dashboardResultsStore.resetAll();
    };
}

```

### DoctorStore.ts:

```

export class DoctorStore implements IDoctorStore {
    currentDoctor: IDoctor | null = null;

    pendingProfile: boolean = false;

    pendingProfileReviews: boolean = false;

    fetchingProfileError: boolean = false;

```



```

constructor() {
    makeAutoObservable(this);
}

getDoctorProfile = (id: number) => {
    this.pendingProfile = true;
    this.fetchingProfileError = false;

    DoctorApi.getDoctor(id)
        .then(
            action(({ data }: AxiosResponse<IGetDoctorSuccessResponse>) => {
                this.currentDoctor = data.data;
            })
        )
        .catch(
            action(() => {
                this.fetchingProfileError = true;
            })
        )
        .finally(
            action(() => {
                this.pendingProfile = false;
            })
        );
};

fetchReviews = () => {
    if (!this.currentDoctor) {
        return;
    }

    this.pendingProfileReviews = true;

    const lastReviewId = this.currentDoctor.reviews[
        this.currentDoctor.reviews.length - 1
    ].id;
    const doctorId = this.currentDoctor.id;

    DoctorApi.getReviews(lastReviewId, doctorId)
        .then(
            action(({ data }: AxiosResponse<IGetReviewsSuccessResponse>) => {
                this.currentDoctor!.reviews.push(...data.data);
            })
        )
        .finally(
            action(() => {
                this.pendingProfileReviews = false;
            })
        );
};

resetProfile = () => {
    this.fetchingProfileError = false;
};
}

```

### CommentStore.ts:

```

const INITIAL_SEND_COMMENT_FORM: ICommentForm = {
    text: "",
    estimation: null
}

```

```

};

const INITIAL_SEND_COMMENT_FORM_ERRORS: ICommentFormErrors = {
  text: undefined,
  estimation: undefined
};

export class CommentStore implements ICommentStore {
  doctorId: number | undefined = undefined;

  commentForm = INITIAL_SEND_COMMENT_FORM;

  commentFormErrors = INITIAL_SEND_COMMENT_FORM_ERRORS;

  pending: boolean = false;

  submissionError: string | undefined = undefined;

  private rootStore: IStores;

  constructor(rootStore: IStores) {
    this.rootStore = rootStore;

    makeObservable(this, {
      doctorId: observable,
      commentForm: observable,
      commentFormErrors: observable,
      pending: observable,
      submissionError: observable,
      sendComment: action,
      validateForm: action,
      setFormValue: action,
      setDoctorId: action,
      resetForm: action
    });

    reaction(
      () => this.commentForm.text,
      text => text && (this.commentFormErrors.text = isLength(text, 5, 200
0))
    );

    reaction(
      () => this.commentForm.estimation,
      estimation =>
        estimation &&
        (this.commentFormErrors.estimation = isRating(estimation))
    );
  }

  sendComment = () => {
    if (!this.validateForm()) {
      return;
    }

    this.pending = true;

    const postData: ISendCommentPostData = {
      doctorId: this.doctorId!,
      text: this.commentForm.text,
      estimation: this.commentForm.estimation!
    };
  }

```

```

    CommentApi.sendComment(postData)
      .then(() => {
        this.rootStore.modalsStore.setModalIsOpen("add-comment", false);
        this.resetForm();
      })
      .catch(
        action((error: AxiosError<ISendCommentErrorResponse>) => {
          this.submissionError = error.response?.data.message;
        })
      )
      .finally(
        action(() => {
          this.pending = false;
        })
      );
  };

  validateForm = () => {
    this.commentFormErrors.text = isLength(this.commentForm.text, 5, 2000);
    this.commentFormErrors.estimation = isRating(this.commentForm.estimation);
  };

  return Boolean(
    !(this.commentFormErrors.text || this.commentFormErrors.estimation)
  );
};

setFormValue = <K extends KeysOfCommentForm>(key: K, value: ICommentForm[K])
=> {
  this.commentForm[key] = value;
};

setDoctorId = (id: number) => {
  this.doctorId = id;
};

resetForm = () => {
  this.commentForm = INITIAL_SEND_COMMENT_FORM;
  this.commentFormErrors = INITIAL_SEND_COMMENT_FORM_ERRORS;
  this.submissionError = undefined;
};
}

```

### ChatStore.ts:

```

export class ChatStore implements IChatStore {
  dialogs: Dialog[] = [] as Dialog[];

  currentDialog: Dialog | undefined = undefined;

  pendingDialogs: boolean = false;

  pendingMessages: boolean = false;

  hasMore: boolean = true;

  messageText: string = "";

  audioBlobUrl: string | undefined = undefined;

  private rootStore: IStores;
}

```

```

constructor(rootStore: IStores) {
  this.rootStore = rootStore;

  makeAutoObservable(this, {
    dialogs: observable,
    currentDialog: observable,
    pendingDialogs: observable,
    pendingMessages: observable,
    hasMore: observable,
    messageText: observable,
    getDialogs: action,
    getMessages: action,
    setCurrentDialog: action,
    setMessageText: action,
    sendMessage: action,
    sendAudio: action,
    appendMessage: action,
    setAudioBlobUrl: action,
    resetCurrentDialog: action,
    resetAll: action
  });
}

getDialogs = () => {
  this.pendingDialogs = true;

  ChatApi.getDialogs()
    .then(
      action(({ data }: AxiosResponse<IGetDialogsSuccessResponse>) => {
        this.dialogs = data.data;
      })
    )
    .finally(
      action(() => {
        this.pendingDialogs = false;
      })
    );
};

getMessages = () => {
  if (!this.currentDialog) {
    return;
  }

  this.pendingMessages = true;

  const lastMessageId =
    this.currentDialog.messages[this.currentDialog.messages.length - 1].
id;

  const currentDialogId = this.currentDialog.id;

  ChatApi.getMessages(currentDialogId, 20, Number(lastMessageId))
    .then(
      action(({ data }: AxiosResponse<IGetMessagesSuccessResponse>) => {
        if (data.data.length > 0) {
          this.dialogs
            .filter(dialog => dialog.id === currentDialogId)[0]
            .messages.push(...data.data);
          this.hasMore = true;
        }
      })
    )

```

```

        if (data.data.length < 20) {
            this.hasMore = false;
        }
    })
}
    .finally(
        action(() => {
            this.pendingMessages = false;
        })
    );
};

setCurrentDialog = (id: string) => {
    this.currentDialog = this.dialogs.filter(
        dialog => dialog.id === Number(id)
    )[0];
};

setMessageText = (text: string) => {
    if (text.length <= MAX_MESSAGE_COUNT) {
        this.messageText = text;
    }
};

sendMessage = () => {
    if (
        !this.messageText ||
        !this.currentDialog ||
        !this.rootStore.userStore.currentUser
    ) {
        return;
    }

    const user = this.rootStore.userStore.currentUser;

    const dialog = this.currentDialog;

    const messageText = this.messageText;

    const avatar = user.additionalData
        ? user.userType === "doctor"
            ? user.additionalData.photo
            : user.additionalData.avatar
        : null;

    const randomId = uuidv4();

    const message: Message = {
        id: randomId,
        chatId: dialog.id,
        text: messageText,
        createdAt: new Date(),
        user: {
            id: user.id,
            avatar: avatar,
            name: user.name
        },
        pending: true
    };

    dialog.messages.unshift(message);

    const socketMessage: SendMessageSocketData = {

```

```

        chatId: dialog.id,
        text: messageText,
        uuid: randomId
    };

    this.rootStore.socketsStore.sendMessage(socketMessage);

    this.messageText = "";
};

sendAudio = async () => {
    if (
        !this.audioBlobUrl ||
        !this.currentDialog ||
        !this.rootStore.userStore.currentUser
    ) {
        return;
    }

    const user = this.rootStore.userStore.currentUser;

    const dialog = this.currentDialog;

    const audioBlobUrl = this.audioBlobUrl;

    const audioBlob = await fetch(audioBlobUrl).then(r => r.blob());

    const avatar = user.additionalData
        ? user.userType === "doctor"
          ? user.additionalData.photo
            : user.additionalData.avatar
          : null;

    const randomId = uuidv4();

    const message: Message = {
        id: randomId,
        chatId: dialog.id,
        file: {
            path: audioBlobUrl,
            type: "audio"
        },
        createdAt: new Date(),
        user: {
            id: user.id,
            avatar: avatar,
            name: user.name
        },
        uuid: randomId,
        pending: true
    };

    dialog.messages.unshift(message);

    const formData = new FormData();
    formData.append("chatId", dialog.id.toString());
    formData.append("file", audioBlob);
    formData.append("type", "audio");
    formData.append("uuid", randomId);

    ChatApi.sendMedia(formData);

    this.audioBlobUrl = undefined;

```

```

};

sendFile = (file: File) => {
  if (!file || !this.currentDialog || !this.rootStore.userStore.currentUse
r) {
    return;
  }

  const user = this.rootStore.userStore.currentUser;

  const dialog = this.currentDialog;

  const avatar = user.additionalData
    ? user.userType === "doctor"
      ? user.additionalData.photo
      : user.additionalData.avatar
    : null;

  const randomId = uuidv4();

  const isImage = isFileImage(file);

  const message: Message = {
    id: randomId,
    chatId: dialog.id,
    file: {
      path: URL.createObjectURL(file),
      type: isImage ? "image" : "file",
      name: file.name,
      size: file.size
    },
    createdAt: new Date(),
    user: {
      id: user.id,
      avatar: avatar,
      name: user.name
    },
    uuid: randomId,
    pending: true
  };

  dialog.messages.unshift(message);

  const formData = new FormData();
  formData.append("chatId", dialog.id.toString());
  formData.append("file", file);
  formData.append("type", isImage ? "image" : "file");
  formData.append("uuid", randomId);

  ChatApi.sendMedia(formData);

  this.audioBlobUrl = undefined;
};

appendMessage = (message: Message) => {
  const appendedDialog = this.dialogs.filter(
    dialog => dialog.id === message.chatId
  )[0];

  if (appendedDialog) {
    if (
      this.rootStore.userStore.currentUser &&
      message.user.id === this.rootStore.userStore.currentUser.id
    )

```

```

    ) {
      const pendingMessage = appendedDialog.messages.filter(
        item => item.id === message.uuid
      )[0];

      if (pendingMessage) {
        pendingMessage.id = message.id;
        pendingMessage.createdAt = message.createdAt;
        pendingMessage.pending = false;
      }
    } else {
      message.uuid = undefined;
      appendedDialog.messages.unshift(message);
    }
  }
};

setAudioBlobUrl = (blobUrl: string) => {
  this.audioBlobUrl = blobUrl;
};

resetCurrentDialog = () => {
  this.currentDialog = undefined;
};

resetAll = () => {
  this.currentDialog = undefined;
  this.dialogs = [];
  this.pendingDialogs = false;
  this.pendingMessages = false;
  this.hasMore = true;
  this.messageText = "";
};
}

```

### SearchDoctorStore.ts:

```

export class SearchDoctorStore implements ISearchDoctorStore {
  doctors: IDoctor[] = [] as IDoctor[];

  dropdownDoctors: IDoctor[] = [] as IDoctor[];

  pagination: IPagination | null = null;

  searchText: string = "";

  pending: boolean = false;

  pendingSearchDoctors: boolean = false;

  fetchingDoctorsError: boolean = false;

  private rootStore: IStores;

  constructor(rootStore: IStores) {
    this.rootStore = rootStore;

    makeObservable(this, {
      doctors: observable,
      dropdownDoctors: observable,
      pagination: observable,
      searchText: observable,

```



```

        pending: observable,
        pendingSearchDoctors: observable,
        fetchingDoctorsError: observable,
        getDoctors: action,
        searchDoctors: action,
        setSearchText: action
    });

    reaction(
        () => this.searchText,
        debounce(inputValue => {
            inputValue.trim() && this.searchDoctors(inputValue);
        }, 300)
    );
}

getDoctors = (page: number, specialty: string) => {
    this.pending = true;
    this.fetchingDoctorsError = false;

    DoctorApi.getDoctors(page, 3, specialty)
        .then(
            action(({ data }: AxiosResponse<IGetDoctorsSuccessResponse>) =>{
                this.doctors = data.data.items;
                this.pagination = data.data.meta;
            })
        )
        .catch(
            action(() => {
                this.fetchingDoctorsError = true;
            })
        )
        .finally(
            action(() => {
                this.pending = false;
            })
        );
};

searchDoctors = (searchValue: string) => {
    this.pendingSearchDoctors = true;

    DoctorApi.searchDoctors(searchValue)
        .then(
            action(({ data }: AxiosResponse<IGetDoctorsSuccessResponse>) =>{
                this.dropdownDoctors = data.data.items;
            })
        )
        .catch(
            action(() => {
                this.dropdownDoctors = [];
            })
        )
        .finally(
            action(() => {
                this.pendingSearchDoctors = false;
            })
        );
};

setSearchText = (searchText: string) => {
    this.searchText = searchText;
}; }

```

**DashboardDoctorProfileStore.ts:**

```

export class DashboardDoctorProfileStore implements IDashboardDoctorProfileStore
{
    pendingReviews: boolean = false;

    doctorProfileForm = INITIAL_UPDATE_DOCTOR_PROFILE_FORM;

    currentModalState: ChangeDoctorProfileTypes | null = null;

    pendingUpdate: boolean = false;

    submissionError: string | undefined = undefined;

    private rootStore: IStores;

    constructor(rootStore: IStores) {
        this.rootStore = rootStore;

        makeObservable(this, {
            pendingReviews: observable,
            doctorProfileForm: observable,
            currentModalState: observable,
            pendingUpdate: observable,
            submissionError: observable,
            fetchReviews: action,
            updateDoctorProfile: action,
            setCurrentModalState: action,
            setFormValue: action,
            setDoctorProfileForm: action,
            setTabValue: action,
            removeTabValue: action,
            addTabValue: action,
            resetForm: action
        });
    }

    fetchReviews = () => {
        this.pendingReviews = true;

        const user = this.rootStore.userStore.currentUser!;
        const additionalData = user.additionalData!;

        const lastReviewId =
            additionalData.reviews[additionalData.reviews.length - 1].id;

        DoctorApi.getReviews(lastReviewId, user.id)
            .then(
                action(({ data }: AxiosResponse<IGetReviewsSuccessResponse>) =>
                {
                    additionalData.reviews.push(...data.data);
                })
            )
            .finally(
                action(() => {
                    this.pendingReviews = false;
                })
            );
    };

    updateDoctorProfile = () => {
        this.pendingUpdate = true;
        this.submissionError = undefined;
    };

```

```

const postData: IUpdateDoctorProfilePostData = {
  costOfConsultation: Number(this.doctorProfileForm.cost),
  about: this.doctorProfileForm.about,
  education: this.doctorProfileForm.education,
  workplaces: this.doctorProfileForm.workplaces
};

DashboardDoctorApi.updateProfile(postData)
  .then(
    action(
      ({
        data
      }: AxiosResponse<IUpdateDoctorProfileSuccessResponse>) => {
        if (this.rootStore.userStore.currentUser) {
          this.rootStore.userStore.currentUser.additionalData
            ...this.rootStore.userStore.currentUser
              .additionalData,
            ...data.data
        };
        this.rootStore.modalsStore.setModalIsOpen(
          "update-doctor-profile",
          false
        );
        this.setDoctorProfileForm(data.data);
        this.resetForm();
      }
    )
  )
  .catch(
    action((error: AxiosError<IUpdateDoctorProfileErrorResponse>) => {
      this.submissionError = error.response?.data.message;
    })
  )
  .finally(
    action(() => {
      this.pendingUpdate = false;
    })
  );
};

setCurrentModalState = (state: ChangeDoctorProfileTypes) => {
  this.currentModalState = state;
};

setFormValue = <K extends KeysOfDoctorProfileForm>(
  key: K,
  value: IUpdateDoctorProfileForm[K]
) => {
  this.doctorProfileForm[key] = value;
};

setDoctorProfileForm = (data: AdditionalData) => {
  this.doctorProfileForm = {
    cost: data.costOfConsultation.toString(),
    about: data.about,
    education: data.education,
    workplaces: data.workplaces
  };
};

```

```

setTabValue = (
  object: "education" | "workplaces",
  key: "year" | "name",
  index: number,
  value: string
) => {
  if (this.doctorProfileForm[object][index]) {
    if (key === "name") {
      this.doctorProfileForm[object][index].name = value;
    } else {
      this.doctorProfileForm[object][index].year = Number(value);
    }
  }
};

removeTabValue = (object: "education" | "workplaces", index: number) => {
  const tempArray = [...this.doctorProfileForm[object]];
  tempArray.splice(index, 1);
  this.doctorProfileForm[object] = tempArray;
};

addTabValue = (object: "education" | "workplaces") => {
  this.doctorProfileForm[object].push({
    year: new Date().getFullYear(),
    name: ""
  });
};

resetForm = () => {
  this.submissionError = undefined;
};
}

```

## DashboardPatientsStore.ts:

```

export class DashboardPatientsStore implements IDashboardPatientsStore {
  patients: PatientItem[] = [] as PatientItem[];

  currentDate: Date = new Date();

  currentType: GetPatientsType = "new";

  pending: boolean = false;

  constructor() {
    makeAutoObservable(this);

    reaction(
      () => this.currentDate,
      date => this.getPatients(date, this.currentType)
    );

    reaction(
      () => this.currentType,
      type => this.getPatients(this.currentDate, type)
    );
  }

  getPatients = (date = this.currentDate, type = this.currentType) => {
    this.pending = true;
    this.patients = [];
  }
}

```

```

const correctDate = new Date(date);
correctDate.setUTCHours(0, 0, 0, 0);

DashboardDoctorApi.getPatients(correctDate.toISOString(), type)
  .then(
    action(({ data }: AxiosResponse<IGetPatientsSuccessResponse>) =>
{
      this.patients = data.data;
    })
  )
  .finally(
    action(() => {
      this.pending = false;
    })
  );
};

setCurrentDate = (date: Date) => {
  this.currentDate = date;
};

setCurrentType = (type: GetPatientsType) => {
  this.currentType = type;
};

resetAll = () => {
  this.patients = [];
  this.currentType = "new";
};
}

```

### DashboardConsultationsStore.ts:

```

export class DashboardConsultationsStore implements IDashboardConsultationsStore
{
  activeConsultations: Consultation[] = [] as Consultation[];

  waitingConsultations: Consultation[] = [] as Consultation[];

  doneConsultations: Consultation[] = [] as Consultation[];

  pendingActiveConsultations: boolean = false;

  pendingWaitingConsultations: boolean = false;

  pendingDoneConsultations: boolean = false;

  cancelConsultationId: number | null = null;

  constructor() {
    makeAutoObservable(this);
  }

  getWaitingConsultations = () => {
    this.pendingWaitingConsultations = true;

    DashboardPatientApi.getConsultations("waiting")
      .then(
        action(
          ({ data }: AxiosResponse<IGetConsultationsSuccessResponse>)
=> {

```

```

        this.waitingConsultations = data.data;
    }
    )
    )
    .finally(
        action(() => {
            this.pendingWaitingConsultations = false;
        })
    );
};

getDoneConsultations = () => {
    this.pendingDoneConsultations = true;

    DashboardPatientApi.getConsultations("done")
        .then(
            action(
                ({ data }: AxiosResponse<IGetConsultationsSuccessResponse>)
=> {
                    this.doneConsultations = data.data;
                }
            )
        )
        .finally(
            action(() => {
                this.pendingDoneConsultations = false;
            })
        );
};

getActiveConsultations = () => {
    this.pendingActiveConsultations = true;

    DashboardPatientApi.getConsultations("active")
        .then(
            action(
                ({ data }: AxiosResponse<IGetConsultationsSuccessResponse>)
=> {
                    this.activeConsultations = data.data;
                }
            )
        )
        .finally(
            action(() => {
                this.pendingActiveConsultations = false;
            })
        );
};

cancelConsultation = () => {
    if (!this.cancelConsultationId) {
        return;
    }

    const postData: ICancelConsultationPostData = {
        consultationId: this.cancelConsultationId
    };

    DashboardPatientApi.cancelConsultation(postData).then(
        action(() => {
            this.waitingConsultations = this.waitingConsultations.filter(
                item => item.id !== this.cancelConsultationId
            );
        })
    );
};

```

```
        ))
    );
};

setCancelConsultationId = (id: number) => {
    this.cancelConsultationId = id;
};
}
```