**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

|  |
| --- |
| Институт информационных технологий и управления в технических системах |
| (полное название института) |

|  |
| --- |
| кафедра «Информационные системы» |
| (полное название кафедры) |

**Пояснительная записка**

к курсовому проекту

по дисциплине «Проектный практикум»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| на тему | **Веб-ориентирования система для консультирования с врачами** | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Выполнил: студент | | | IV | | | курса, группы: | | | | | **ИС/б-17-2-о** | | | | | |
|  | | | | | | | | | | | | | | | | |
| Направления подготовки (специальности) | | | | | | | | | | | 09.03.02 | | | | | |
|  | | | | | | | | | | | | | | | | |
| Информационные системы и технологии | | | | | | | | | | | | | | | | |
| (код и наименование направления подготовки (специальности)) | | | | | | | | | | | | | | | | |
| профиль (специализация) | | | | | |  | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| **Долженко Илья Андреевич** | | | | | | | | | | | | | | | | |
| (фамилия, имя, отчество студента) | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Руководитель | | | Строганов В.А. | | | | | | | | | | | | | |
|  | | | (фамилия, инициалы, степень, звание, должность) | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Защита « | |  | | » |  | | | | 20 | | 21 | г. | | Оценка | |  |
|  | | | | | | | | | | | | | | | | |
| Руководитель | | | | | | |  | | | | | |  | |  | |
|  | | | | | | | (подпись) | | | | | |  | | (инициалы, фамилия) | |
|  | | | | | | | | | | | | | | | | |
| Ведущий преподаватель | | | | | | |  | | | | | |  | | Строганов В.А. | |
|  | | | | | | | (подпись) | | | | | |  | | (инициалы, фамилия) | |
|  | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | 21 | | г. | | | | | | |

**АННОТАЦИЯ**

В данной пояснительной записке представлено описание основных этапов выполнения курсового проекта по дисциплине «Проектный практикум». Курсовой проект посвящен разработке WEB-сервиса для консультаций с врачами. Приведено техническое задание, по которому реализован данный проект.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ**

**1 Цели создания и целевая аудитория**

Цели создания сайта:

* Предоставление услуг консультации с врачами;
* Предоставить удобную возможность удаленного общения с врачом из любой точки России;
* Помощь клиенту в выборе врача.

Целевой аудиторией сайта являются клиенты, заинтересованные в получении рекомендаций от врачей разных специальностей, путем проведения онлайн-консультации.

**2 Структура сайта**

В разрабатываемом Web-приложении предполагается создание трех независимых интерфейсов пользователей: общий интерфейс, личный кабинет пациента и врача.

Общий интерфейс содержит:

* главную страницу с описанием основной информации о сервисе, отзывами наших клиентов и предоставляемыми тарифами;
* страница выбора врача с категориями и строкой поиска;
* детальная страница с подробной информацией о враче и его отзывами;
* форма регистрации пациента и врача.

Личный кабинет пациента содержит:

* страницу для просмотра истории записей пациента и запланированных встреч с возможностью отмены;
* страницу анализов пациента с возможностью добавить новые документы/снимки или удалить старые;
* страницу для просмотра медицинской карты пациента;
* страницу диалогов с врачами
* настройки личного кабинета с возможностью изменить свои личные данные, предпочитаемый вариант оплаты и деактивацией аккаунта.

Личный кабинет врача содержит:

* страницу с подробной информацией о враче и отзывами о нем;
* календарь с датой и временем запланированных консультаций;
* страницу с ближайшими запланированными консультациями и информацией о пациенте;
* страницу диалогов с пациентами;

**3. Пожелания по сайту**

Все страницы разрабатываемого сайта должны быть выдержаны в одном стиле. Корпоративные цвета: белый, синий, серый. Ширина: 1400px.

**4. Технические требования к сайту**

Сайт должен работать на основных современных браузерах (Google Chrome, Safari, Opera, Mozilla Firefox). Кроме того, сайт должен быть адаптивным: работать на устройствах с любым разрешением экрана (компьютерах, смартфонах, планшетах).

Инструменты для разработки:

Frontend (TypeScript, React, MobX, Material UI).

Backend (Node, PostgreSQL, TypeScript, Express, Sequelize)

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 6

1. ПОСТАНОВКА ЗАДАЧИ 7

2. АНАЛИЗ ПРОБЛЕМНОЙ СИТУАЦИИ И ОБЗОР АНАЛОГОВ 8

2.1. Описание ситуации 8

2.2. Обзор аналогов 8

3. ПЛАН РЕАЛИЗАЦИИ ПРОЕКТА 10

4. ВЫБОР ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ 12

5. ОПИСАНИЕ РЕЗУЛЬТАТА 14

5.1. Реализация интерфейса 14

5.2. Разработка программных модулей 22

6. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО САЙТА 31

ЗАКЛЮЧЕНИЕ 40

СПИСОК ЛИТЕРАТУРЫ И ИНФОРМАЦИОННЫХ РЕСУРСОВ 41

ПРИЛОЖЕНИЕ А. ДИАГРАММА ГАНТА 42

ПРИЛОЖЕНИЕ Б. КОД ПРИЛОЖЕНИЯ 43

**ВВЕДЕНИЕ**

Данная работа состоит из пояснительной записки, включающей в себя аннотацию, содержание, введение, шесть разделов, заключение, список использованных источников и два приложения.

В первом разделе подробно описывается постановка задачи.

Во втором разделе был проведен анализ проблемной ситуации и обзор аналогов. Были выявлены проблемы, которые нас сервис может решить.

В третьем разделе описаны этапы работы на протяжении семестра.

В четвертом разделе были выбраны и описаны инструментальные средства, используемые при разработке сервиса.

В пятом разделе находится описание продуктового результата. Здесь описано состояние проекта на момент завершения семестра.

В шестом разделе было произведено тестирование разработанного сервиса на наличие валидации и правильной работы таких функций, как: авторизация/регистрация, заполнение анкеты пациента и заявки на врача, поиск врача, запись на прием.

**1 ПОСТАНОВКА ЗАДАЧИ**

Определим основные функциональные требования для приложения:

* разрабатываемый WEB-сервис по предоставлению услуг онлайн-консультаций должен состоять из трех основных частей: публичные страницы и личные кабинеты пациента/врача.
* необходимо реализовать авторизацию и регистрацию нового пациента/врача;
* для доступа в личный кабинет пользователю необходимо заполнить соответствующую его типу (пациент или врач) анкету;
* необходимо реализовать админ панель для подтверждения заявки на врача;
* для управления версиями исходного кода проекта, а также для упрощения групповой разработки, необходимо использовать систему контроля версий Git;
* разработанный интерфейс сервиса должен проходить тест на кроссбраузерность и адаптивность на разных устройствах;
* структура публичной части сайта: главная страница, список специалистов, страница просмотра специалиста, страница регистрации, диалоговое окно авторизации на сайте;
* пациенты и врачи помимо доступа к публичным страницам должны иметь доступ к личному кабинету, который содержит страницы просмотра истории записей и запланированных встреч, анализы пациента, медицинскую карту, диалоги с врачами и настройки для пациента, а также профиль врача, ближайшие запланированные встречи, диалоги с пациентами и настройки для врача.

**2 АНАЛИЗ ПРОБЛЕМНОЙ СИТУАЦИИ И ОБЗОР АНАЛОГОВ**

## 2.1 Описание ситуации

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. Разрабатываемый сервис позволит решить описанные проблемы, обеспечив консультации с врачами в дистанционном формате, в удобное для пользователя время.

## 2.2 Обзор аналогов

В процессе анализа проблемной ситуации был проведен обзор существующих аналогов, среди которых выделены сервисы, представленные ниже.

Яндекс Здоровье (https://health.yandex.ru/):

1. Невозможность выбрать время сеанса до оплаты;

2. Малый выбор врачей (всего 3 на выбор).

3. Отсутствие возможности подать заявку на врача.

Медведь.Телемед (https://telemed.chat/):

1. Врач может ограничить тип связи для консультации (только чат или видеосвязь);

2. Консультации доступны только в мобильном приложении.

3. Отсутствие возможности подать заявку на врача.

Федеральный научно-клинический центр (https://fnkc-fmba.ru/telemeditsina/):

1. Консультации не проводятся по мобильному телефону

2. Отсутствие возможности подать заявку на врача.

**3 ПЛАН РЕАЛИЗАЦИИ ПРОЕКТА**

Для определения времени осуществления мероприятий, направленных на достижение целей проекта, и для установления взаимосвязей между ними по временному параметру с учётом наиболее рисковых событий, составляется календарный план проекта. Календарное планирование заключается в создании и последующем уточнении расписания, которое учитывает состав работ, риски, ограничения. Поскольку календарный план в виде перечня исключительно плановых параметров работ без сравнения с фактическими сроками выполнения утрачивает свой смысл, нередко, вместо календарного плана, применяют название календарного графика.

В таблице 3.1 показан план реализации проекта, где описаны этапы проекта, сроки выполнения, результаты, участники проекта и задействованные материально-технические ресурсы.

Таблица 3.1 – План реализации проекта

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Этапы проекта / конкретные мероприятия, детализирующие этапы | Cрок выпол-нения | Результат | Участники проекта (О – ответственный / У – участники) | Задействован-ные материально-технические  ресурсы |
| Этап 1 | Проектирование БД | 10.02.21 – 25.02.21 | draw.io-файл со схемой БД | О – Иванов М. С..  У – Долженко И. А.  Маметов Э. Н. | Draw.io  PostgreSQL,  Sequelize |
| Этап 2 | Разработка дизайна | 25.02.21 – 03.03.21 | Макет в Figma | О – Маметов Э. Н.  У – Иванов М. С.  Долженко И. А. | Figma,  креатив |
| Этап 3 | Верстка публичной части сайта | 03.03.21 – 23.03.21 | Html, CSS, TS-файлы | О – Долженко И. А.  У – Маметов Э. Н. | Время и психическое здоровье |
| Этап 4 | Разработка функционала авторизации и регистрации | 23.03.21 – 25.03.21 | Функционал для сервиса | О – Иванов М. С. Долженко И. А.  У – Маметов Э. Н. | Время и психическое здоровье |
| Этап 5 | Разработка функционала анкетирования врача и пациента | 26.03.21-04.04.21 | Функционал анкетирования | О – Иванов М. С. Долженко И. А.  У – Маметов Э. Н. | Время и психическое здоровье |
| Этап 6 | Разработка профиля врача | 04.04.21-05.04.21 | Профиль врача в сервисе | О – Иванов М. С. Долженко И. А.  У – Маметов Э. Н. | Время и психическое здоровье |
| Этап 7 | Постраничный вывод врачей, вывод специальностей, вывод самых опытных врачей на главную страницу. | 06.04.21-07.04.21 | Функционал вывода врачей и специальностей | О – Иванов М. С. Долженко И. А.  У – Маметов Э. Н. | Время и психическое здоровье |

**4 ВЫБОР 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.

**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.

**5 ОПИСАНИЕ РЕЗУЛЬТАТА**

**5.1 Реализация интерфейса**

При запуске web-приложения открывается главная страница, на которой представлена основная информация о нашем сервисе (Рисунок 3) имеется секция с наилучшими специалистами (Рисунок 2) и раздел с отзывами пациентов о сервисе (Рисунок 4).

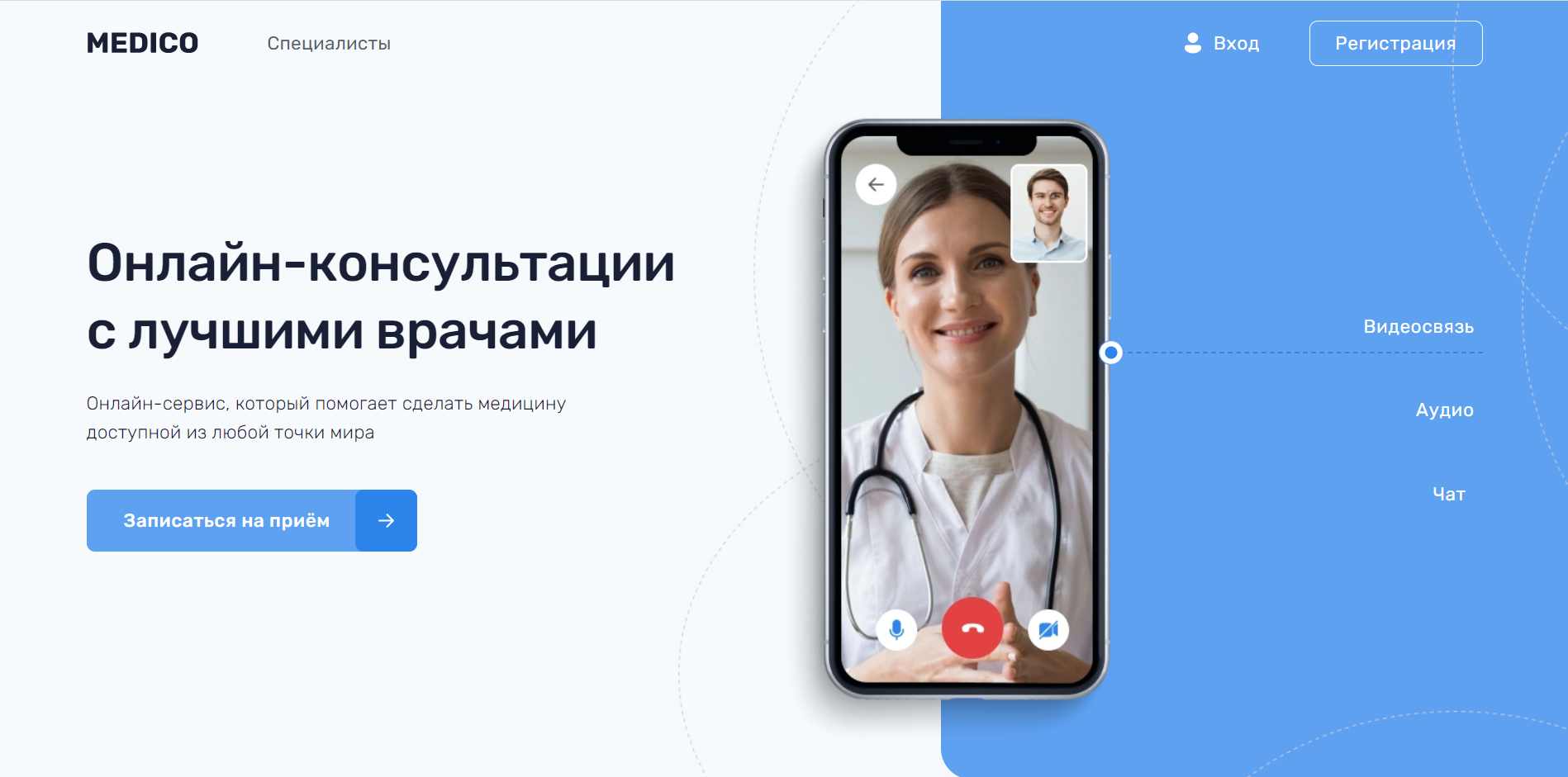


Рисунок 1 – Главный экран

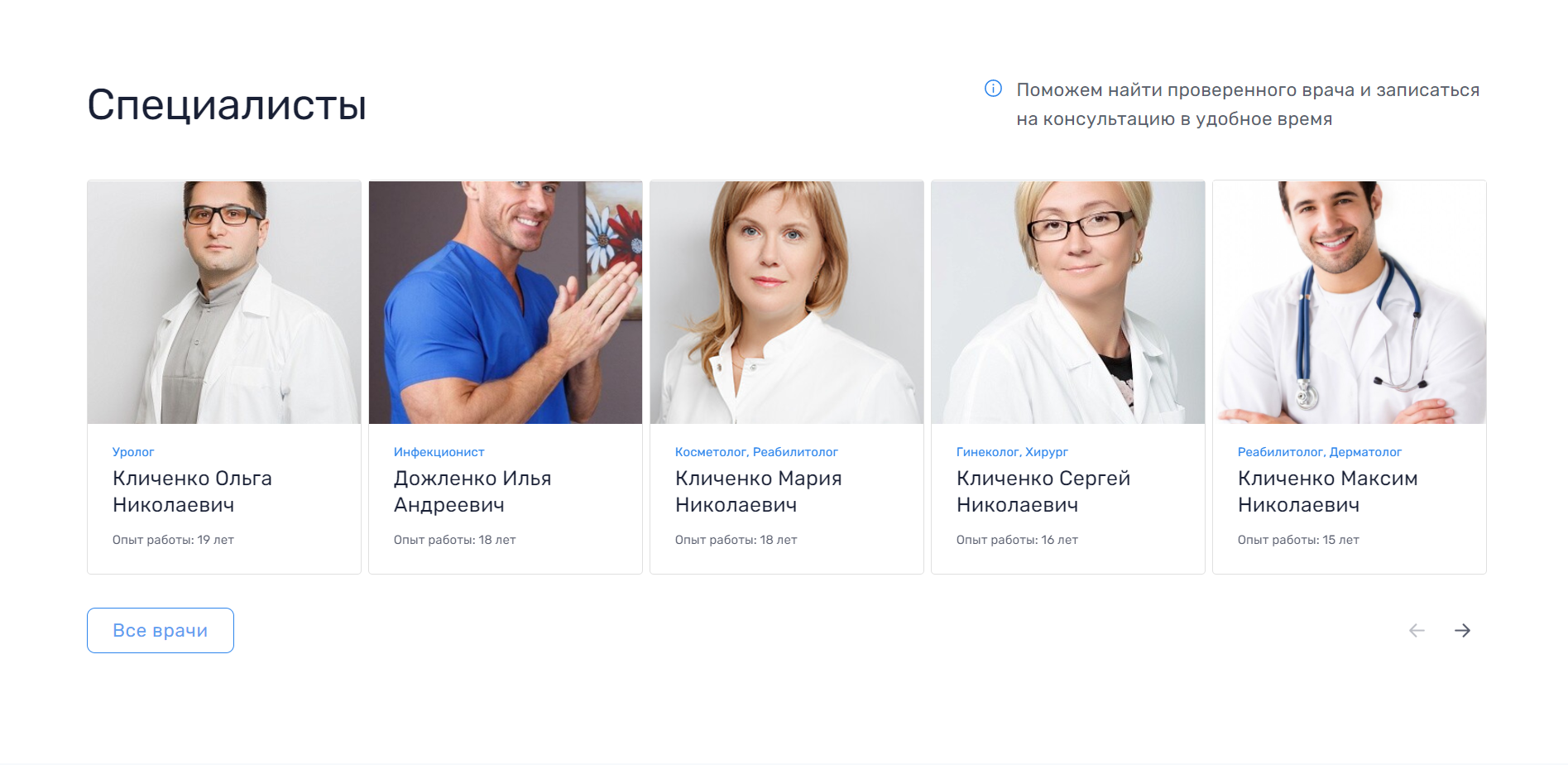


Рисунок 2 – Секция со специалистами

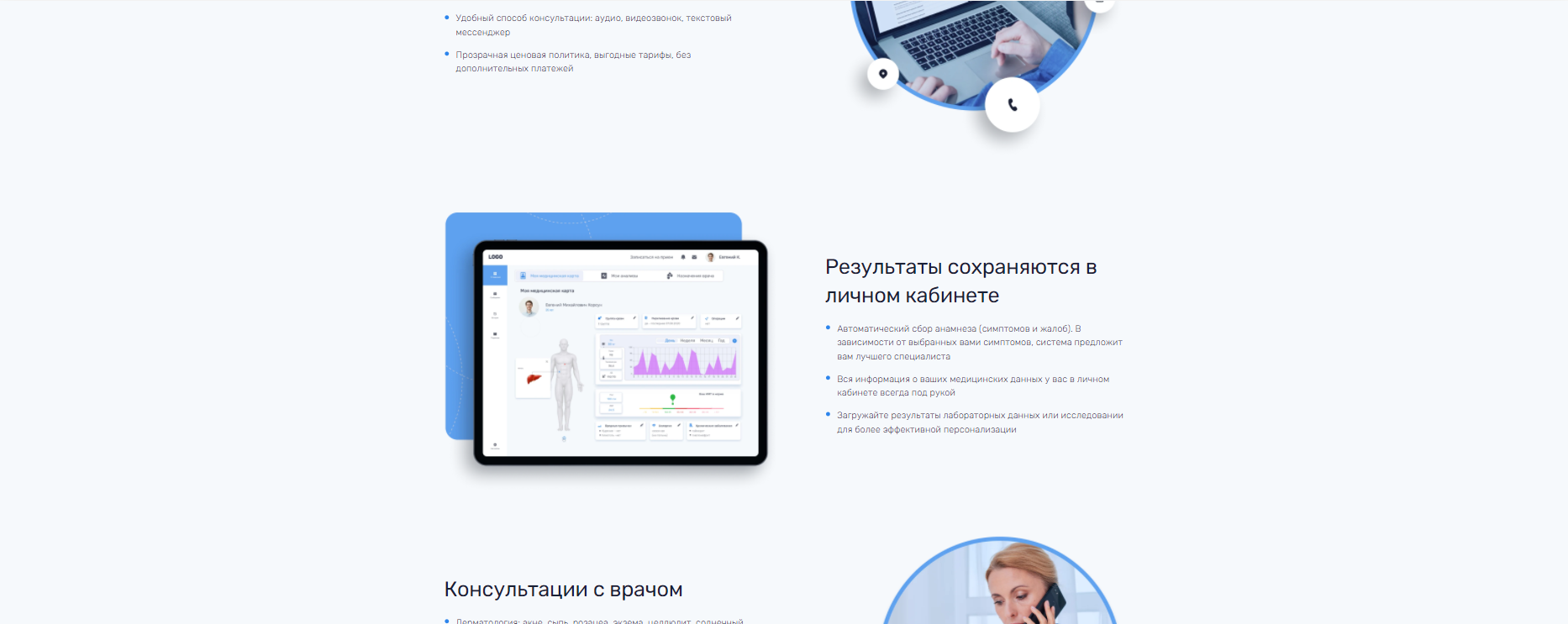


Рисунок 3 – Информация о сервисе

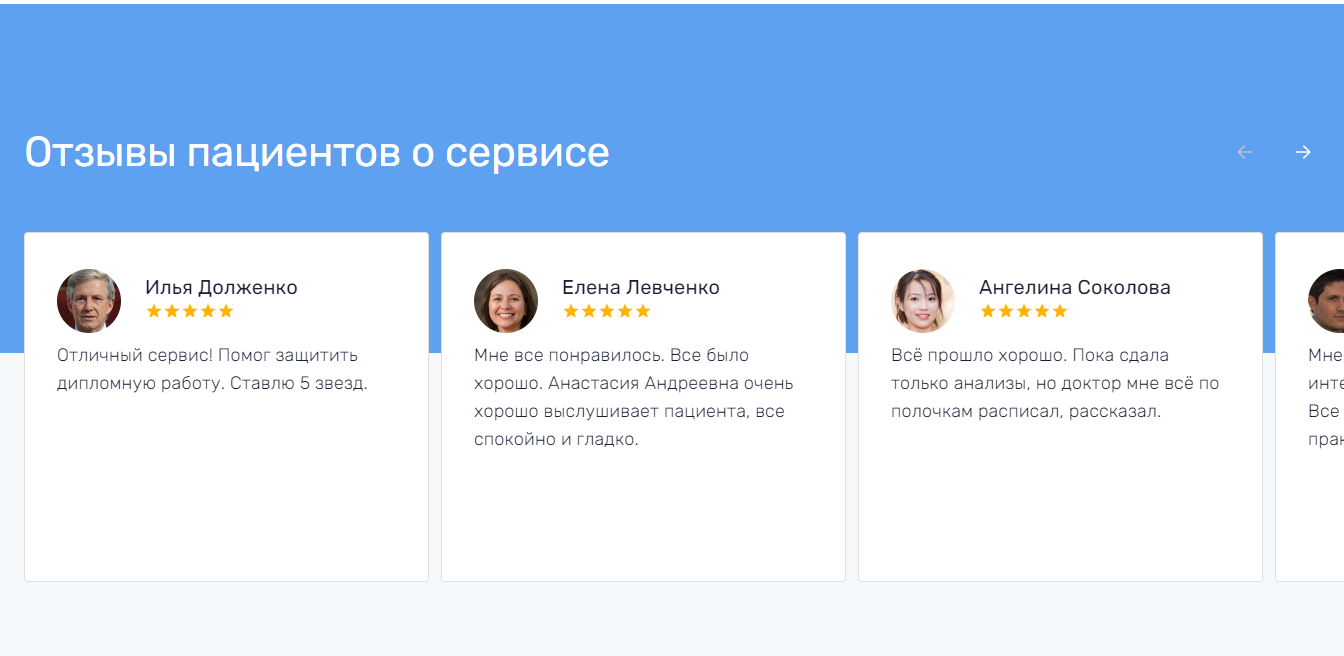


Рисунок 4 – Раздел с отзывами о сервисе

При нажатии на кнопку «Войти» открывается модальное окно с формой входа состоящие из полей «Email» и «Пароль». Так же на этом окне находится кнопка восстановления пароля и регистрации (Рисунок 6)

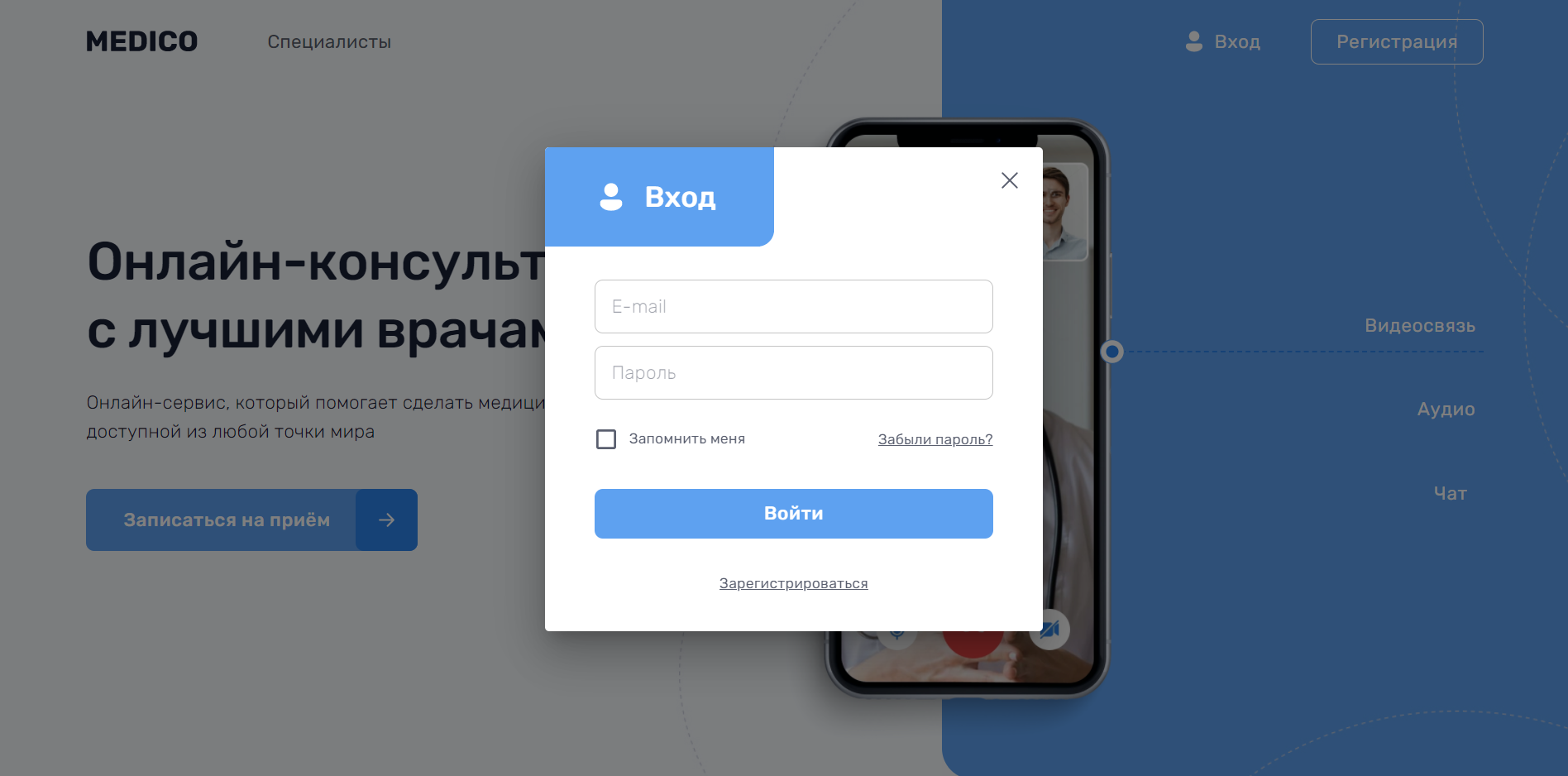


Рисунок 5 – Модальное окно входа

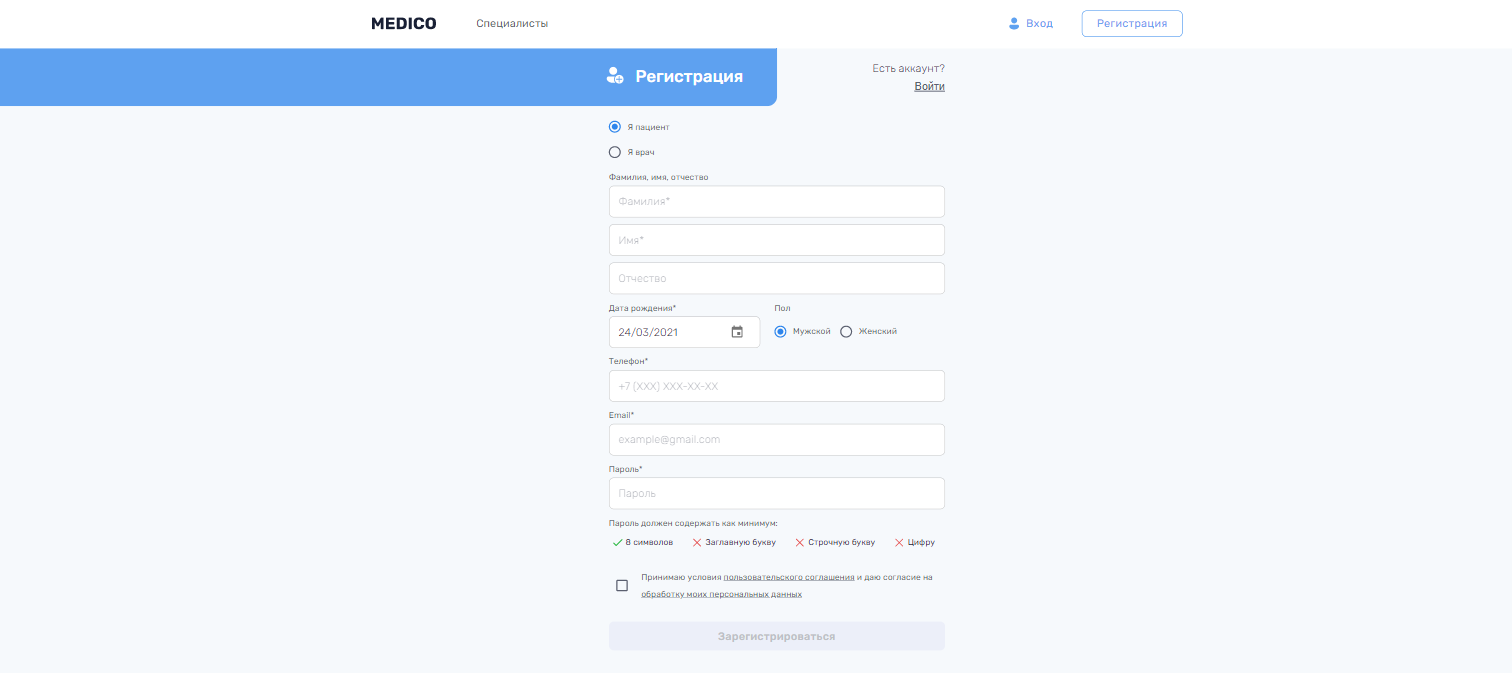


Рисунок 6 – Страница регистрации

После регистрации на почту пользователю приходит письмо с подтверждением аккаунта (Рисунок 7), и появляется модальное окно сообщающее об этом (Рисунок 8).

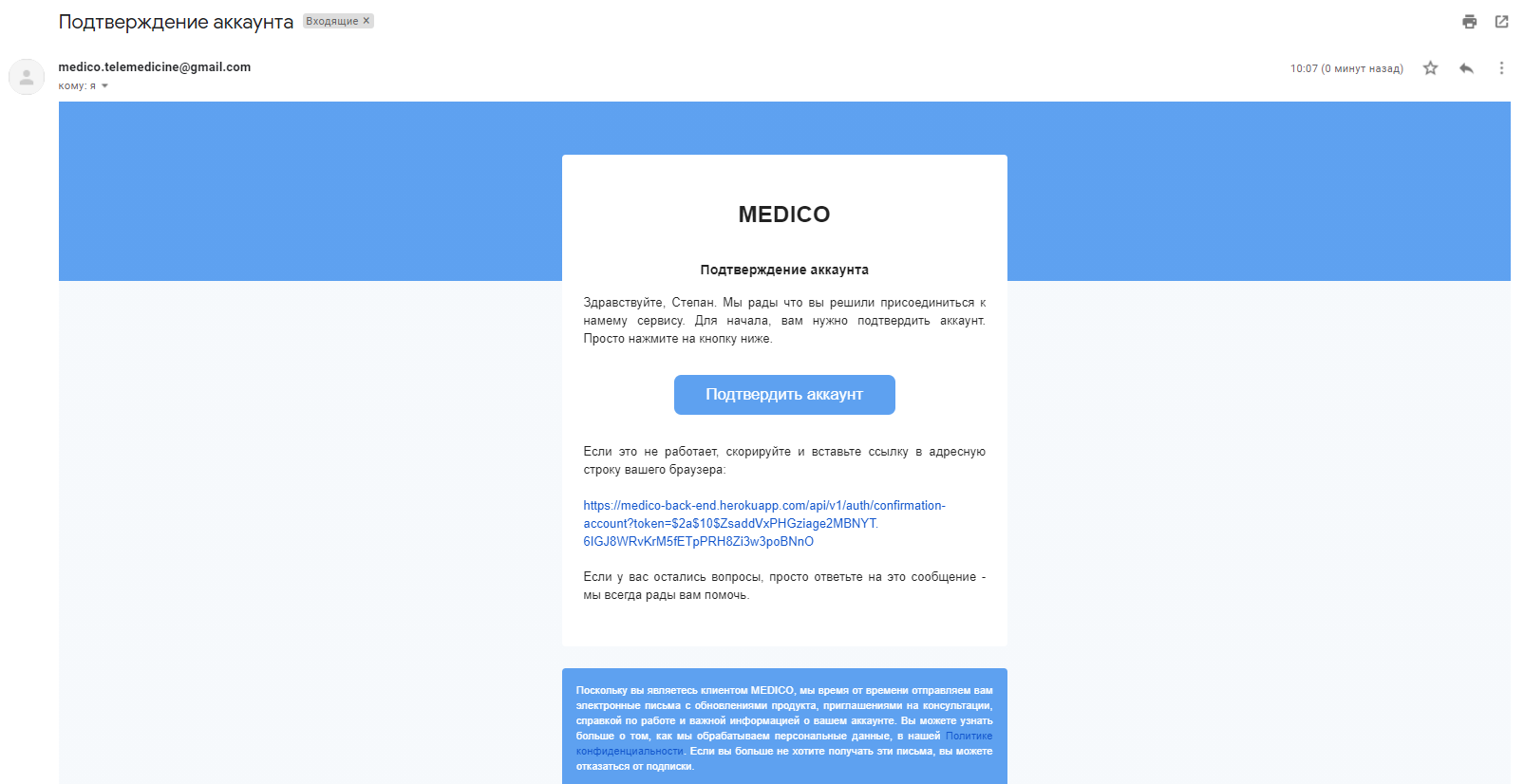


Рисунок 7 – Письмо на почте

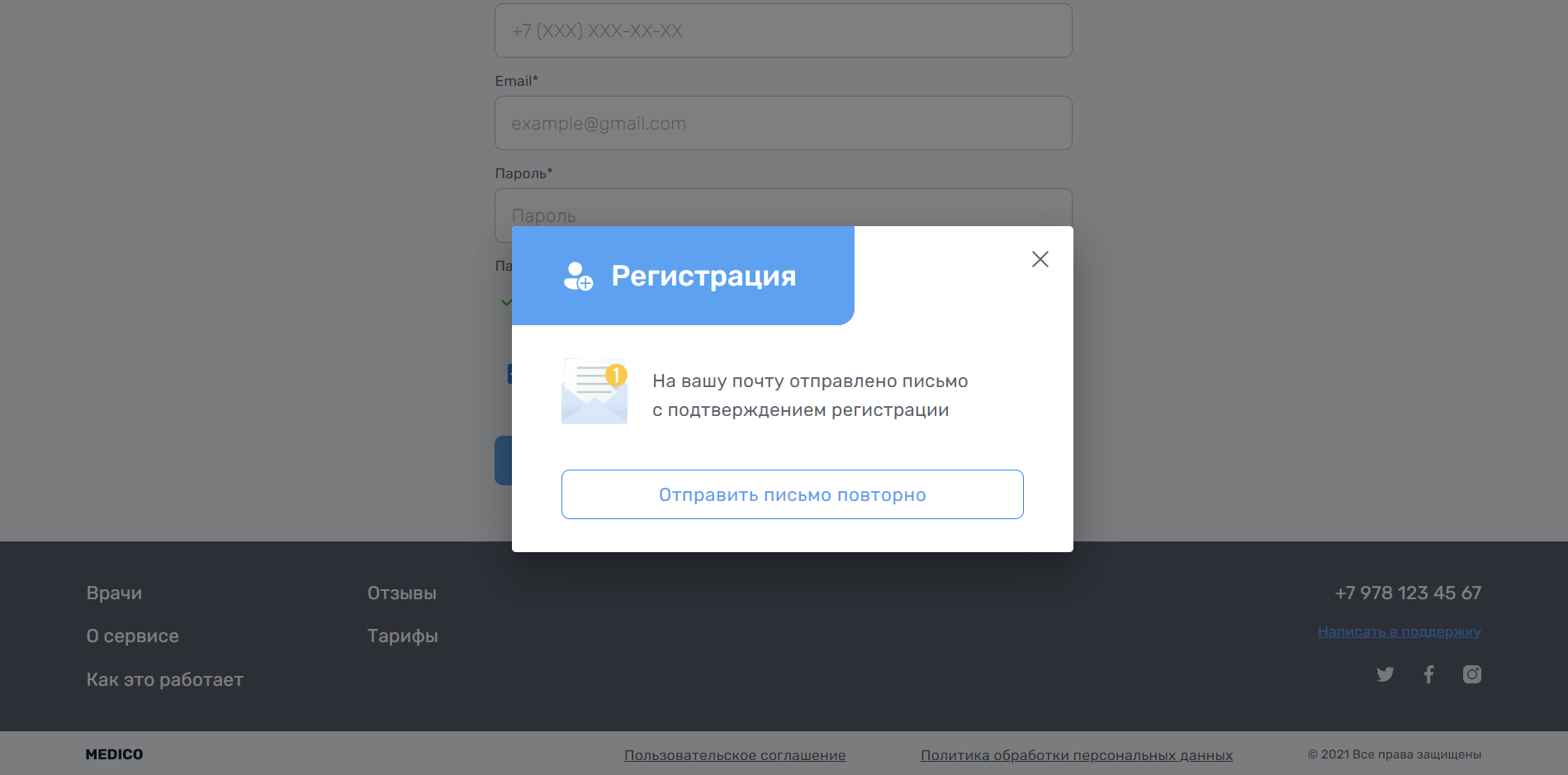


Рисунок 8 – Сообщение об отправки письма

После нажатия на кнопку «Подтвердить аккаунт» в письме пользователь перенаправляется на web-приложение (Рисунок 9).

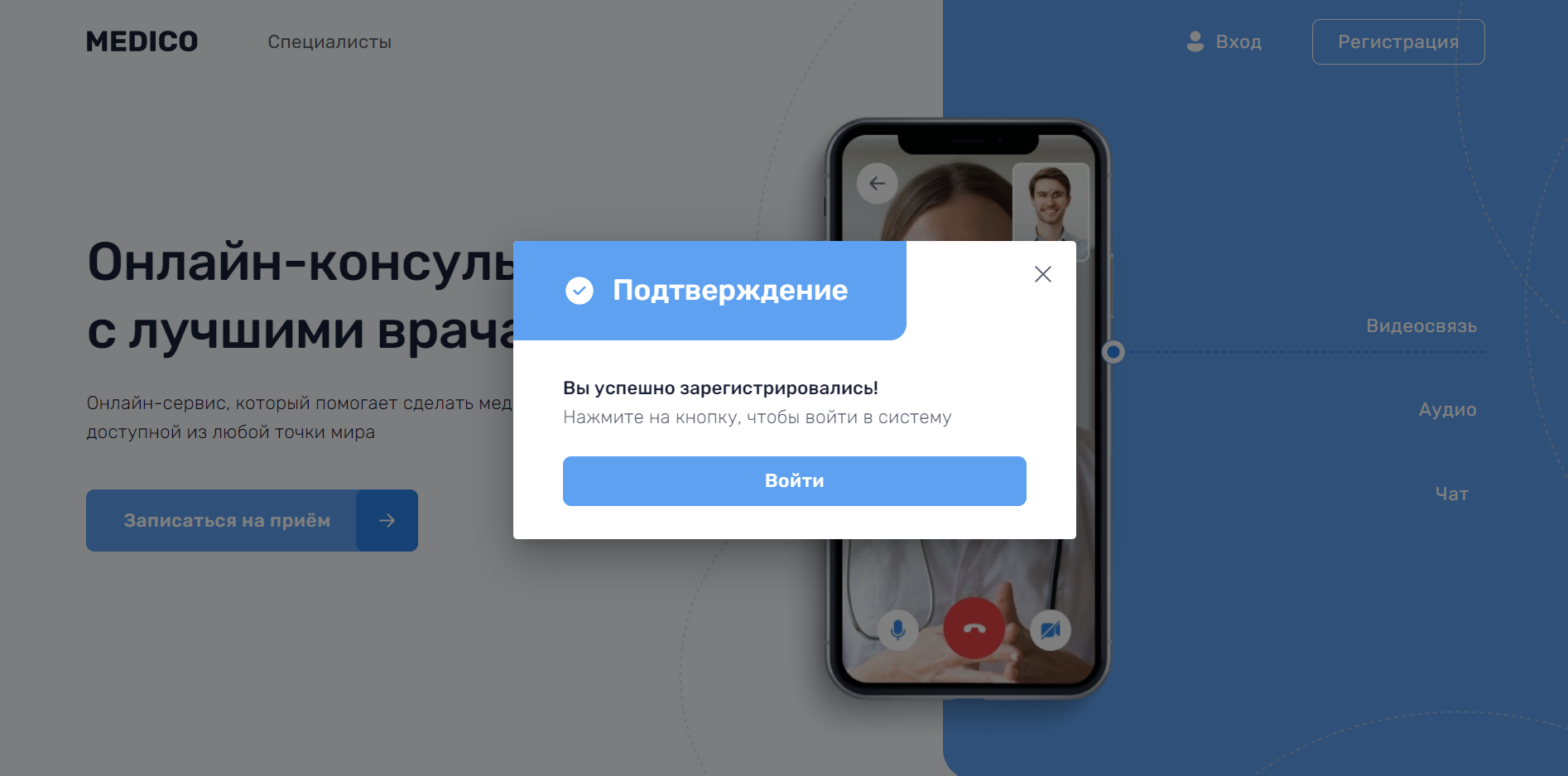


Рисунок 9 – Сообщение об успешном подтверждении аккаунта

После авторизации пациенту нужно заполнить анкету чтобы иметь доступ к личному кабинету (Рисунки 10-11).

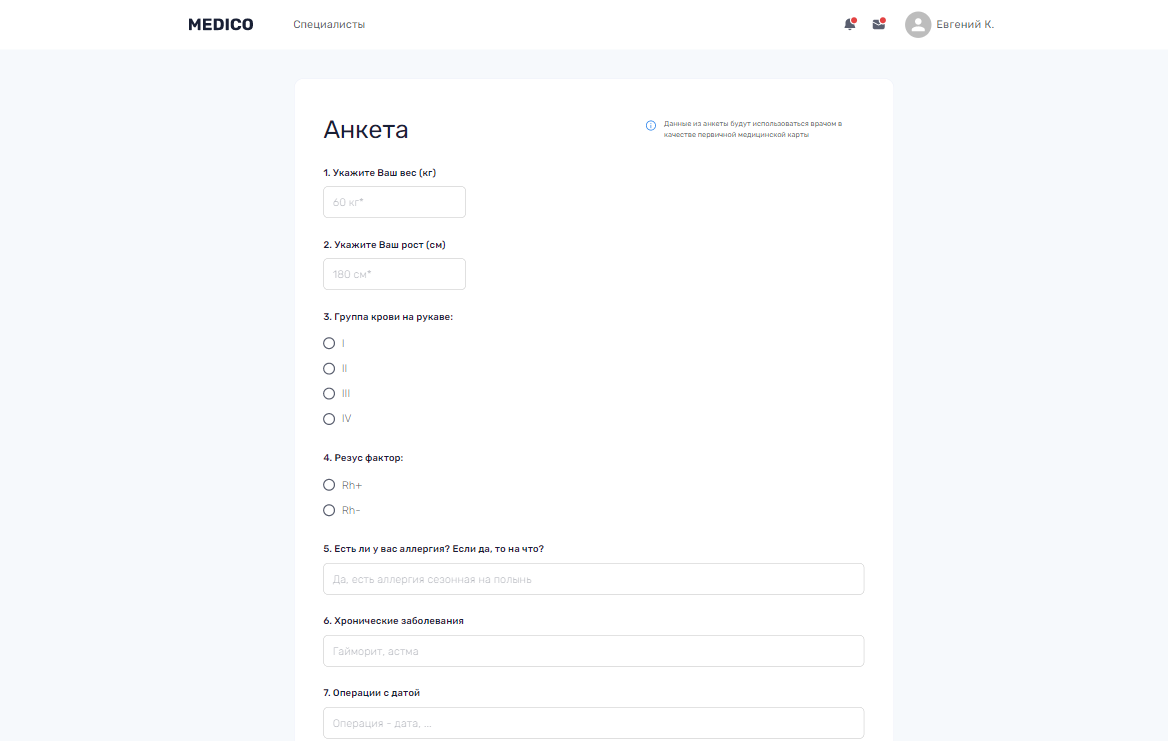


Рисунок 10 – Анкета пациента

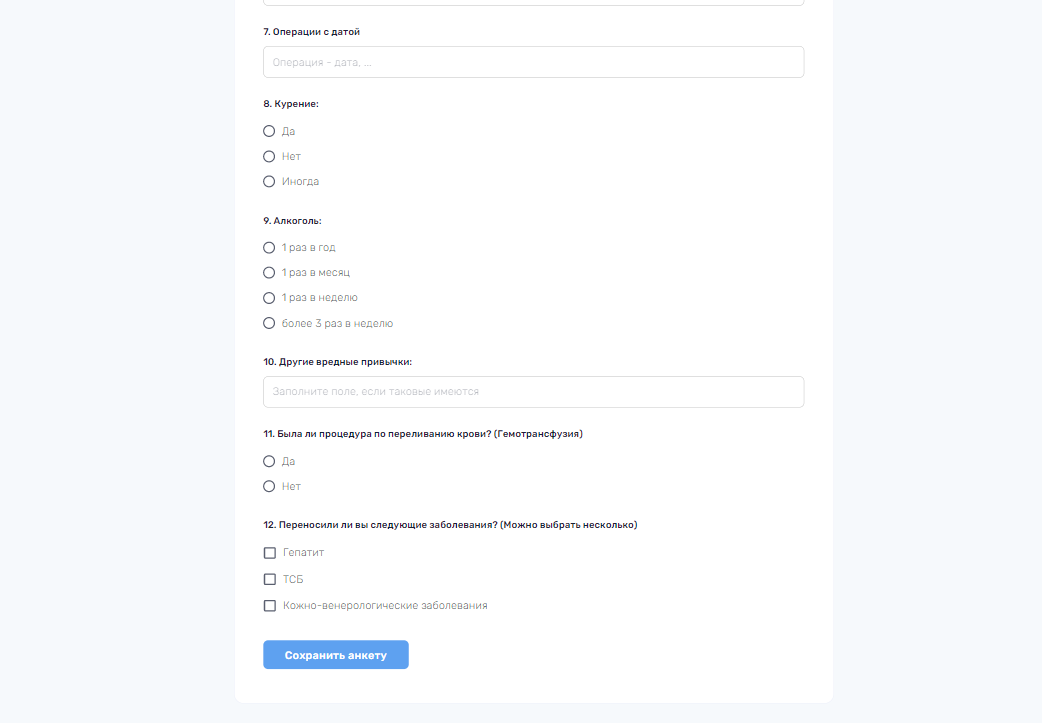


Рисунок 11 – Анкета пациента

Если пользователь является доктором, то ему необходимо заполнить заявку на врача чтобы модератор мог подтвердить его (Рисунки 12-13).

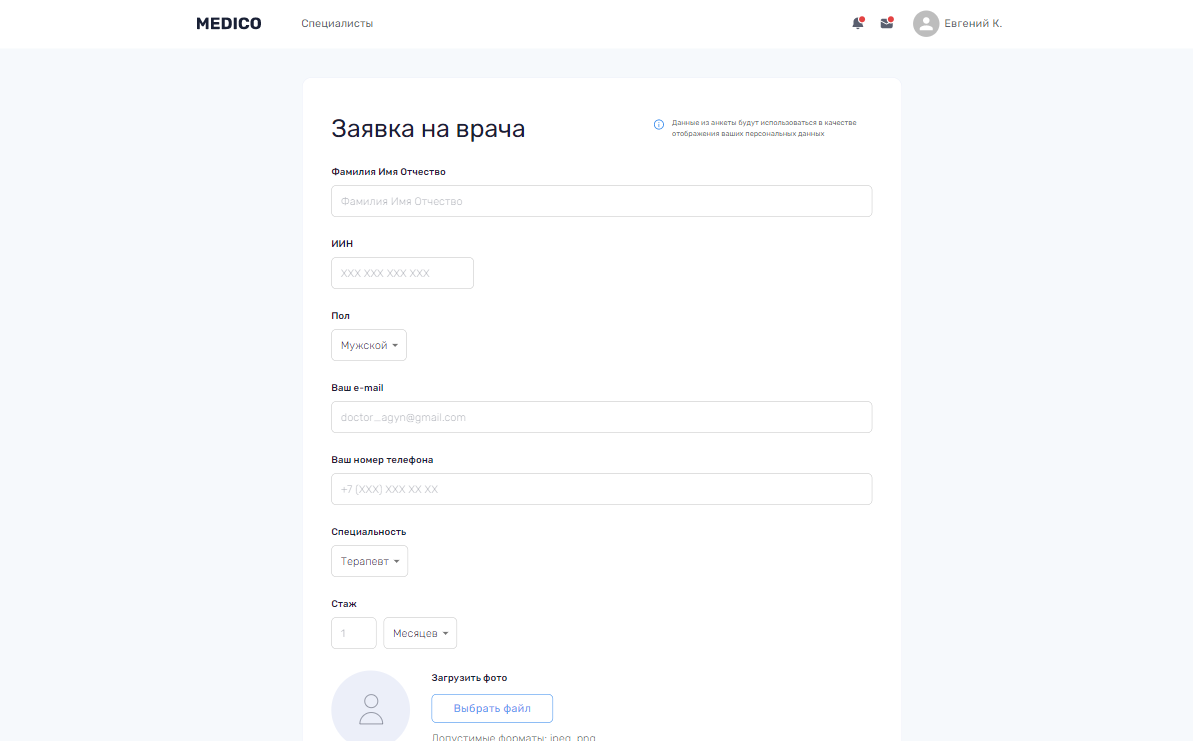


Рисунок 12 – Заявка на врача

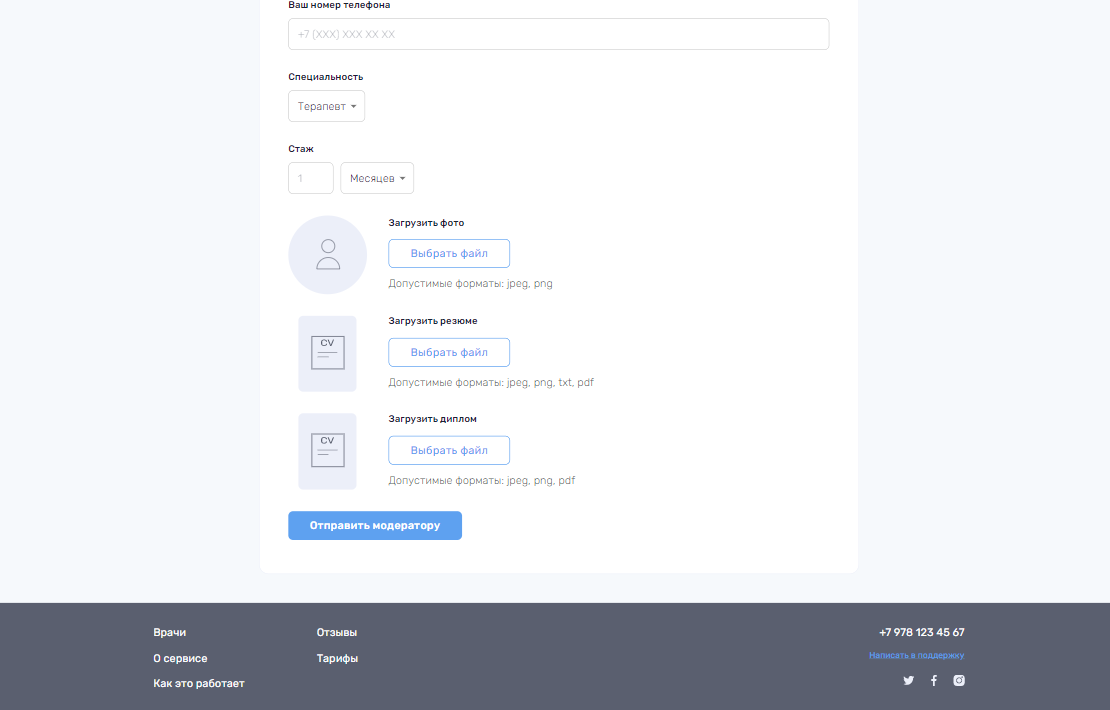


Рисунок 13 – Заявка на врача

После заполнения анкеты пациент имеет возможность найти врача с дальнейшей записью на прием (Рисунки 14-15).

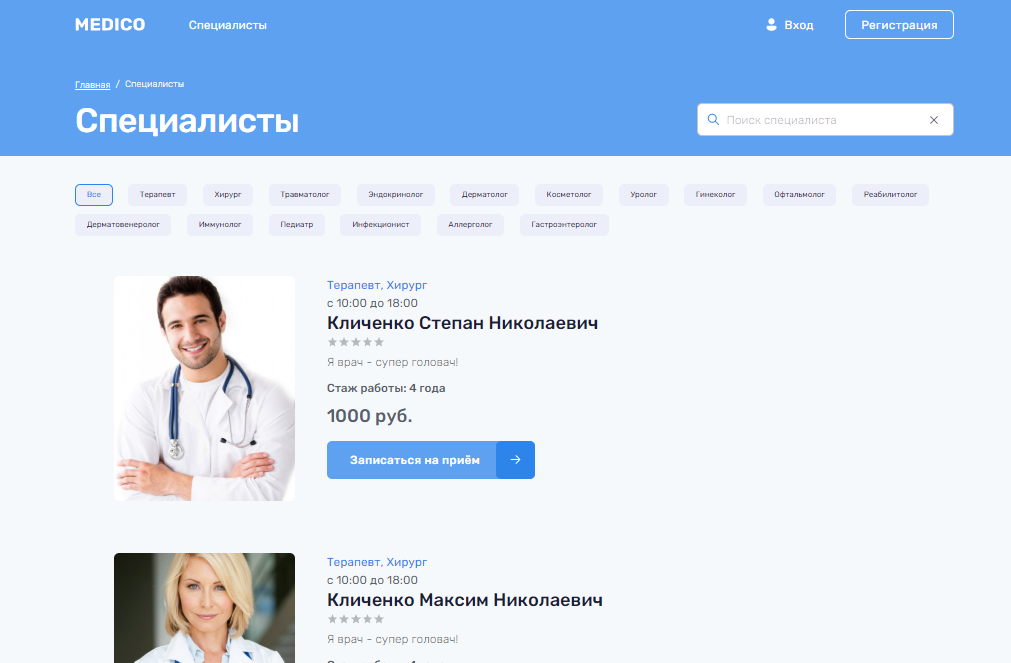


Рисунок 14 – Страница со специалистами

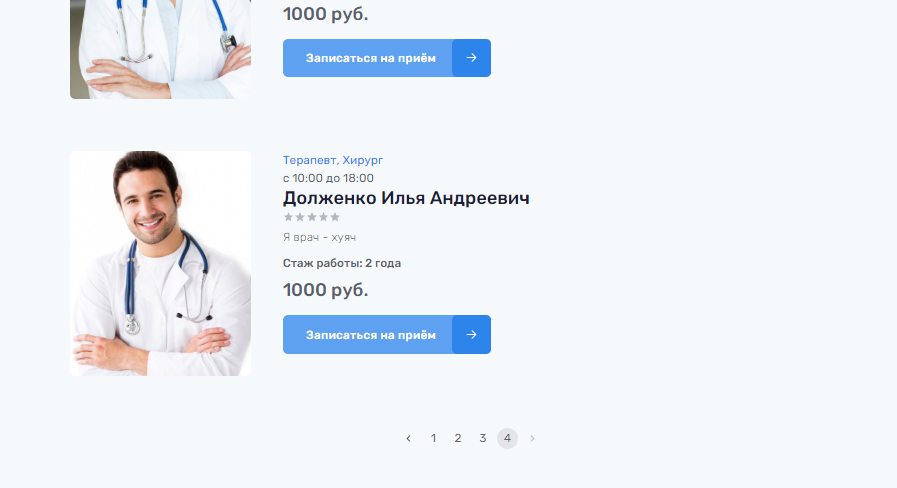


Рисунок 15 – Страница со специалистами

На рисунках 16-17 представлена детальная страница врача, на которой можно посмотреть отзывы пациентов, образование, опыт работы и основные направления по которым работает врач.

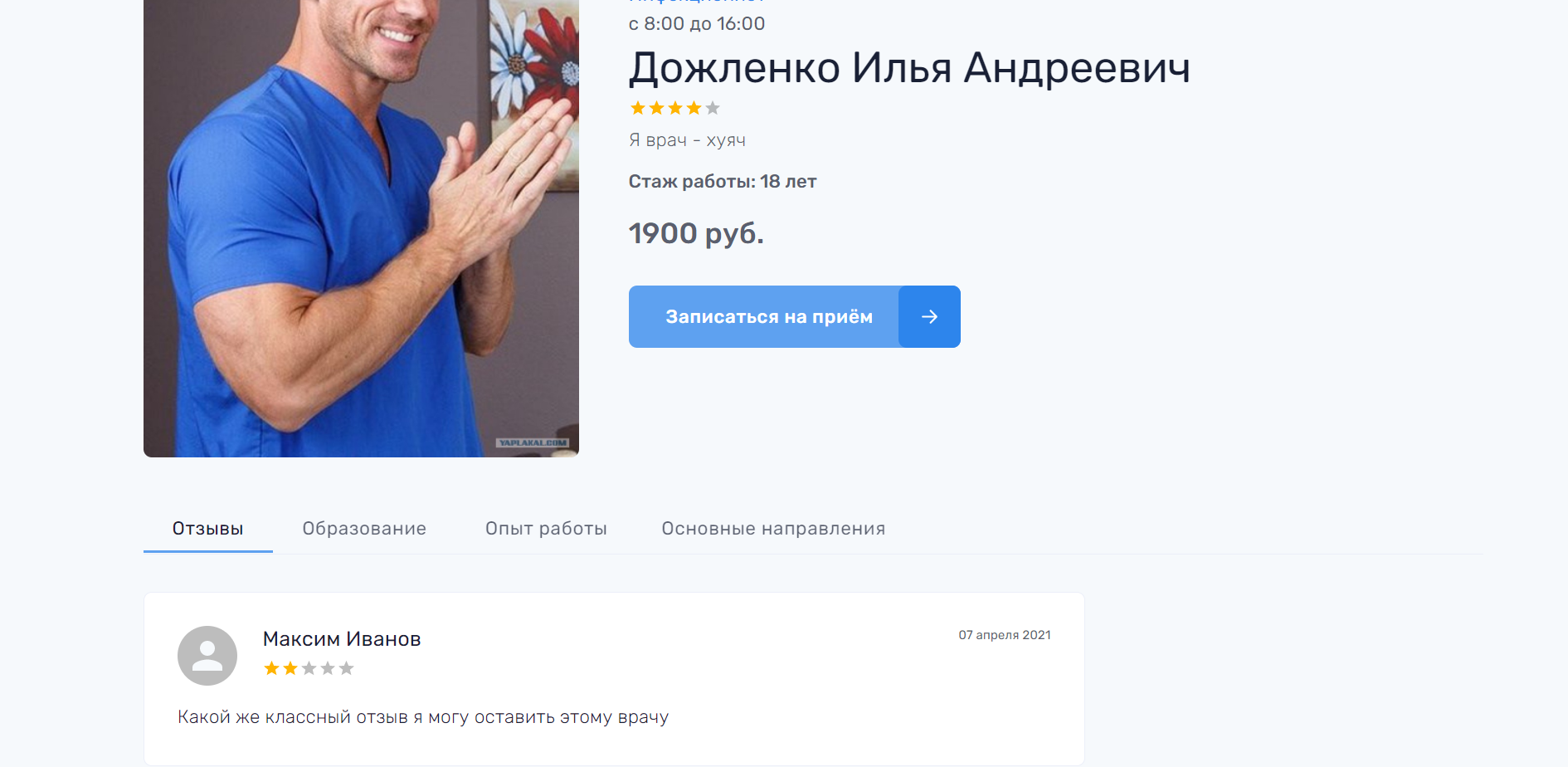


Рисунок 16 – Детальная страница врача

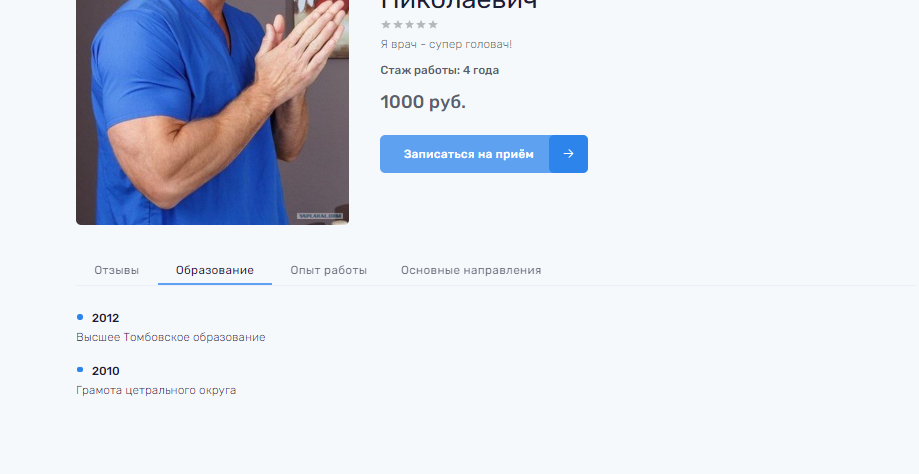


Рисунок 17 – Детальная страница врача

**5.2 Разработка программных модулей**

При разработки клиентской части использовалась библиотека для разработки пользовательских интерфейсов React. На рисунке 18 представлена структура клиентской части.

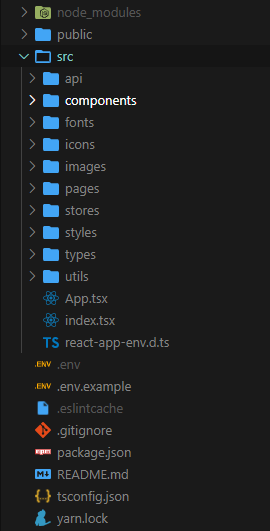


Рисунок 18 – Структура клиентского приложения

В папка images и icons содержатся картинки и иконки всего приложения.

Так как в работе использовался препроцессор для стилей SCSS, то в папке styles содержатся его файлы, разделенные на файл с переменными и файл с глобальными стилями, а также ts-файл с описанием конфигурации Material UI.

Папки utils и api содержат вспомогательные функции, участвующие в дальнейшей разработке, такие как описание api методов, функции для работы со строками и массивами, константы и файл с базовыми настройками библиотеки axios.

В папке fonts соответственно содержатся шрифты, используемые во всем приложении.

Так как в работе использовалась библиотека MobX для управления состоянием приложения, то в проекте находится папка stores, которая содержит подпапку interfaces для определения интерфейсов хранилищ и сами классы сторов, объекты которых создаются в приведенном ниже фрагменте кода из файла index.ts:

*export* class RootStore implements IStores {

    routerStore: IRouterStore;

    signUpStore: ISignUpStore;

    signInStore: ISignInStore;

    userStore: IUserStore;

    modalsStore: IModalsStore;

    questionnaireStore: IQuestionnaireStore;

    specialtiesStore: ISpecialtiesStore;

    doctorStore: IDoctorStore;

    homeStore: IHomeStore;

    constructor(history: History) {

*this*.routerStore = new RouterStore(history);

*this*.signUpStore = new SignUpStore(*this*);

*this*.signInStore = new SignInStore(*this*);

*this*.userStore = new UserStore();

*this*.modalsStore = new ModalsStore();

*this*.questionnaireStore = new QuestionnaireStore(*this*);

*this*.specialtiesStore = new SpecialtiesStore();

*this*.doctorStore = new DoctorStore(*this*);

*this*.homeStore = new HomeStore();

    }

}

Также в проекте имеется папка components для хранения общих компонентов всего приложения, таких как: Header, Footer, Loader, Button, Dialog и т.п.

Все страницы приложения также хранятся в папке pages, структура которых приведена на рисунке 19.

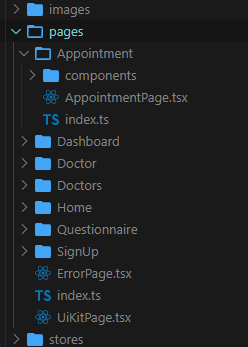


Рисунок 19 – Структура папки pages

Как видно из скриншота выше, папка pages состоит из соответствующих подпапок для каждой страницы, а также в каждой из них есть папка components с локальными компонентами, которые используются только на этой странице.

Старт приложения начинается с файла index.ts, который находится в корневой папке src. В нем компонент App оборачивается в провайдеры и все приложение рендерится у блока с классом root. Фрагмент данного файла представлен ниже:

*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")

);

Сам компонент App содержит jsx-разметку глобальных компонентов, конфигурация роутинга и проверку на авторизованного пользователя, в случае авторизации которого при каждом запуске приложения будет отрисовываться лоадер, тем самым скрывая видимость выполнения запроса на получения пользователя. Фрагмент файла App.tsx представлен ниже:

*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 />

            <Switch>

                <Route *exact* *path*="/ui-kit" *component*={UiKitPage} />

                <Route

*exact*

*path*={["/", "/home", "/sign-up-confirmation"]}

*component*={HomePage}

                />

                <Route

*exact*

*path*={["/doctors", "/doctors/:page"]}

*component*={DoctorsPage}

                />

                <Route *exact* *path*="/doctor/:id" *component*={DoctorPage} />

                <PrivateRoute

*exact*

*path*="/sign-up"

*component*={SignUpPage}

*isAuthorized*={!isAuthorized}

                />

                <PrivateRoute

*exact*

*path*="/appointment"

*component*={AppointmentPage}

*isAuthorized*={isAuthorized}

                />

                <PrivateRoute

*exact*

*path*="/questionnaire"

*component*={QuestionnairePage}

*isAuthorized*={isAuthorized}

                />

                <PrivateRoute

*path*="/dashboard"

*component*={DashboardPage}

*isAuthorized*={isAuthorized}

                />

                <Route *component*={ErrorPage} />

            </Switch>

            <DialogSignIn />

            <DialogReset />

            <DialogEmail />

        </React.Fragment>

    );

});

Основные классы для взаимодействия с серверной части находятся в папке api, классы DoctorApi, QuestionnaireApi, SignIn, SignUp и UserApi представлены ниже:

*export* class DoctorApi {

    static getDoctor(id: number) {

*return* axiosInstance.get(`/api/v1/doctor/info?id=${id}`);

    }

    static getDoctors(page: number, count: number = 3) {

*return* axiosInstance.get(

            `/api/v1/doctor/paginate?page=${page}&count=${count}`

        );

    }

    static getDoctorsByCount(count: number = 7) {

*return* axiosInstance.get(`/api/v1/doctor/most-experienced?count=${count}`);

    }

}

*export* class QuestionnaireApi {

    static sendPatient(postData: IPatientQuestionnairePostData) {

*return* axiosInstance.post("/api/v1/patient/profile/questionnaire", postData);

    }

    static sendDoctor(postData: FormData) {

*return* axiosInstance.post("/api/v1/doctor/profile/questionnaire", postData);

    }

}

*export* class SignInApi {

    static signIn(postData: ISignInPostData) {

*return* axiosInstance.post("/api/v1/auth/sign-in", postData);

    }

}

*export* class SignUpApi {

    static signUp(postData: ISignUpPostData) {

*return* axiosInstance.post("/api/v1/auth/sign-up", postData);

    }

    static sendMail(postData: ISendMailPostData) {

*return* axiosInstance.post("/api/v1/auth/send-email-with-token", postData);

    }

}

*export* class UserApi {

    static getUser() {

*return* axiosInstance.get("/api/v1/user/");

    }

}

Данные методы вызываются в методах store-классов, один из них представлен ниже:

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,

            password: *this*.signInForm.password

        };

        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;

    };

}

А также интерфейс этого класса, находящийся в подпапке interfaces:

*export* interface ISignInStore {

    signInForm: ISignInForm;

    submissionError: string | undefined;

    pending: boolean;

    doSignIn: () => void;

    setFormValue: <K extends KeysOfSignInForm>(

        key: K,

        value: ISignInForm[K]

    ) => void;

    resetForm: () => void;

}

*export* interface ISignInForm {

    email: string;

    password: string;

}

*export* type KeysOfSignInForm = keyof ISignInForm;

Таким образом была описана основная структура клиентского приложения и приведены фрагменты кода основных компонентов и классов приложения.

**6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО САЙТА**

Для начала протестируем работу клиентского приложения. Для этого перейдем к регистрации нового пользователя. При нажатии на кнопку “Регистрация” в верхнем меню пользователь попадает на соответствующую форму (Рисунок 6). Введем неверный формат в полях ввода для проверки валидации (Рисунок 18).

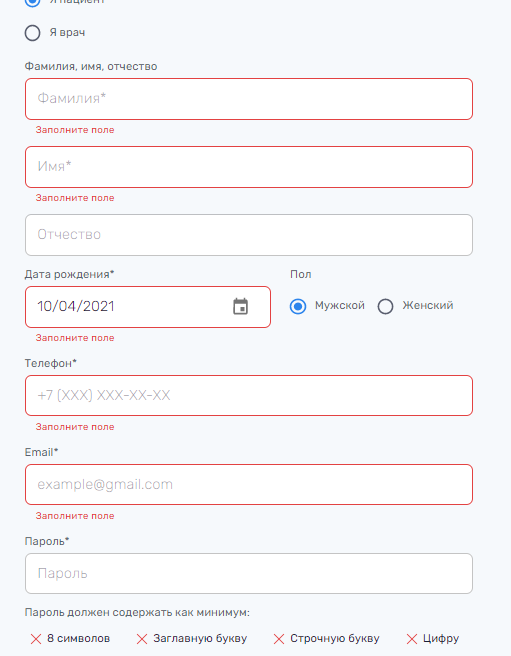


Рисунок 18 – Неверный формат ввода в полях формы

После успешной регистрации появляется уведомление, изображенное на рисунке 19, подтверждающее успешное выполнение регистрации и предлагающее подтвердить аккаунт путем перехода по ссылке из письма на почте.

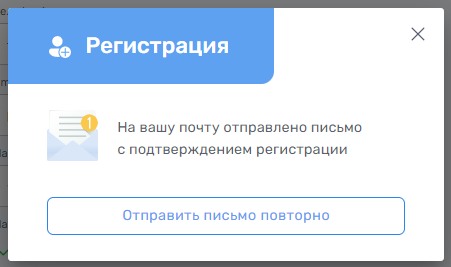


Рисунок 19 – Уведомление после отправки формы

После перехода по ссылке из письма перед пользователем появится модальное окно с уведомление о успешной регистрации и предлагающее перейти на форму входа (Рисунок 20).

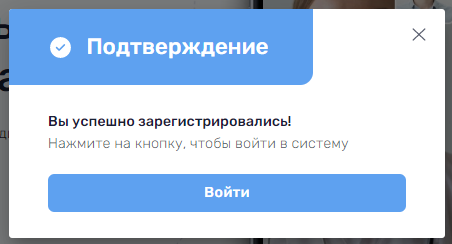


Рисунок 20 – Уведомление о успешной регистрации

Если ввести неверные данные или не подтвердить аккааунт, то появится надпись, указывающая на определенную ошибку. Данное уведомление представлено на рисунке 21.

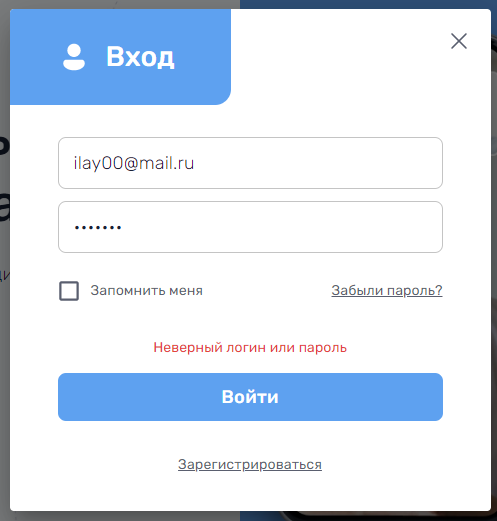


Рисунок 21 – Ошибка о неверном логине или пароле

Далее проверим корректную работу валидации на анкетах пациента и врача. Валидация формы пациента представлена на рисунке 22, а формы врача на рисунке 23.

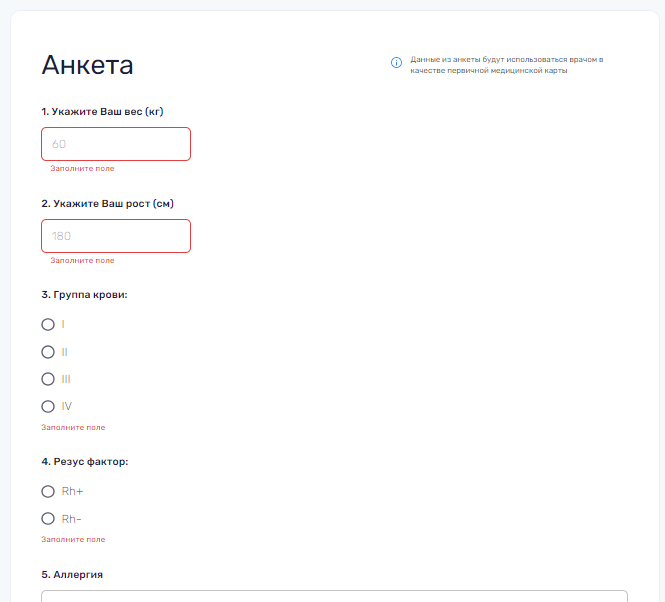


Рисунок 22 – Валидация формы пациента

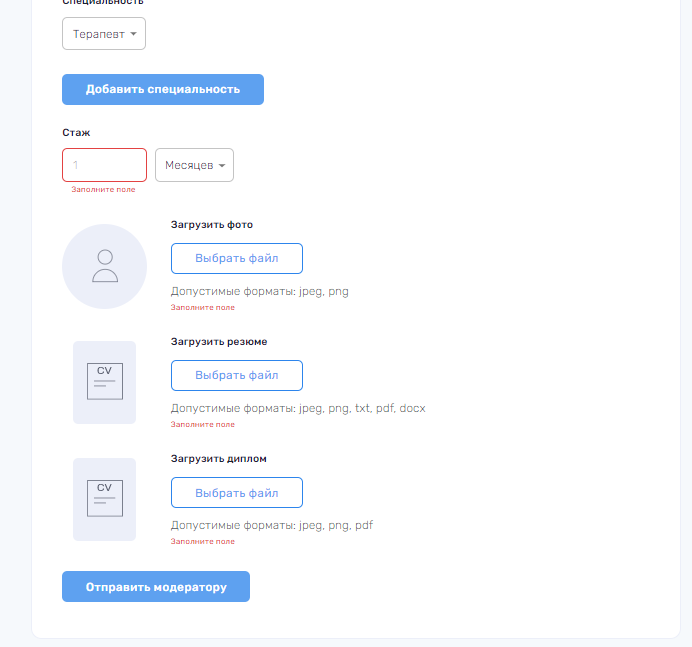


Рисунок 23 – Валидация формы врача

Далее проведем тестирование поиска врача (Рисунок 24-25) и его просмотр (Рисунок 26).

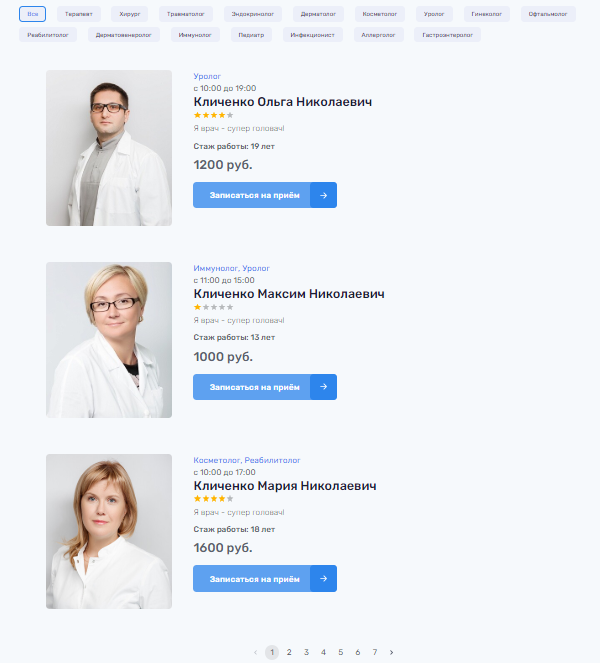


Рисунок 24 – Просмотр первой страницы

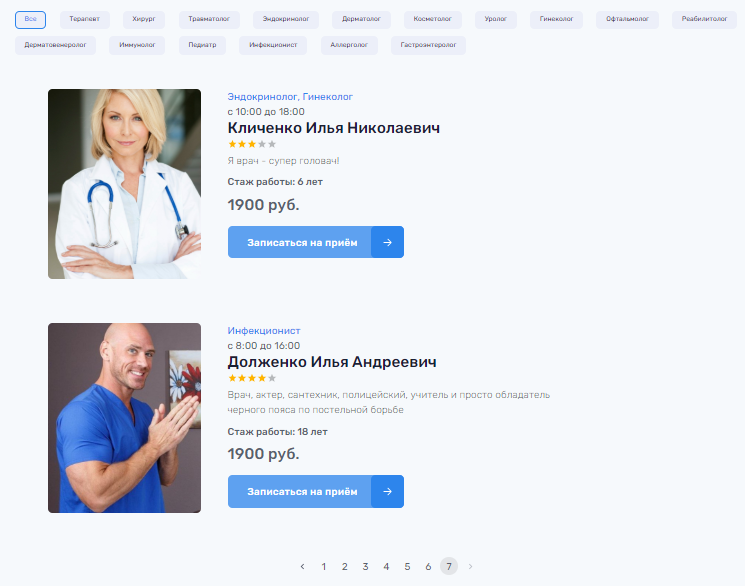


Рисунок 25 – Просмотр другой страницы

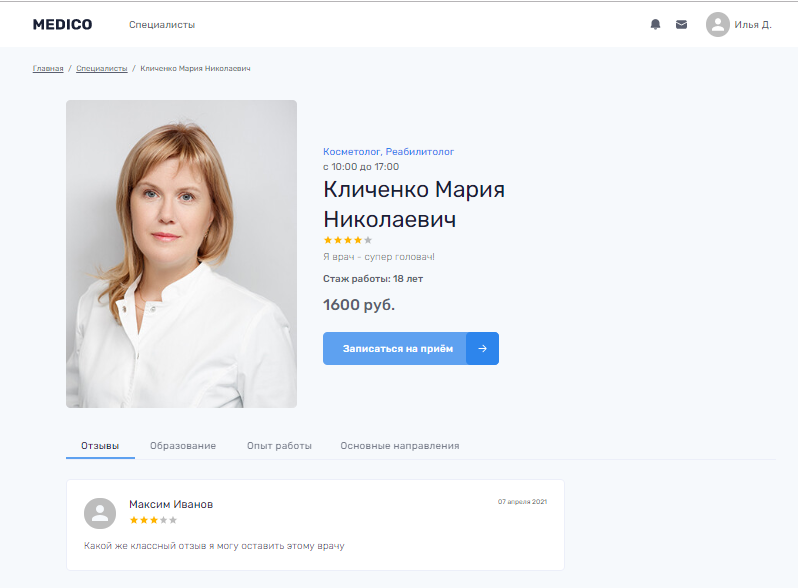


Рисунок 26 – Просмотр профиля врача

Теперь выполним запись на прием, для этого жмем на кнопку “Записаться на прием” в профиле врача и переходим на форму, состоящая из 4 этапов, результатом прохождения которой будет успешная, либо неуспешная запись. Данная форма записи на прием представлена на рисунке 27-30.

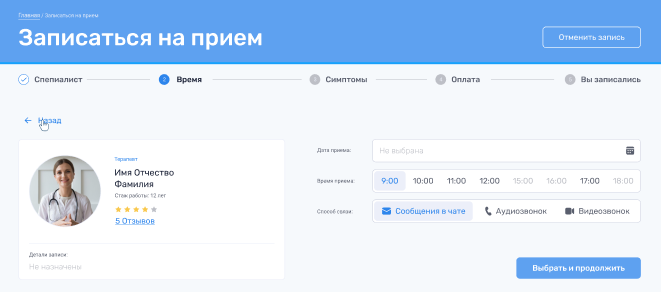


Рисунок 27 – Запись на прием. Этап 1

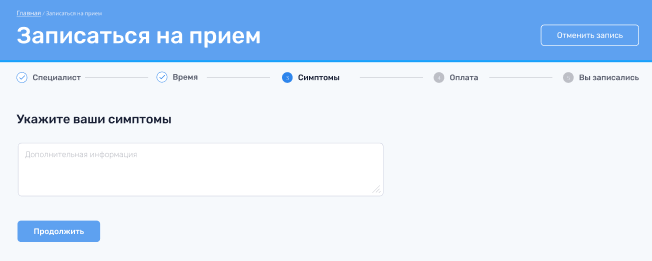


Рисунок 28 – Запись на прием. Этап 2

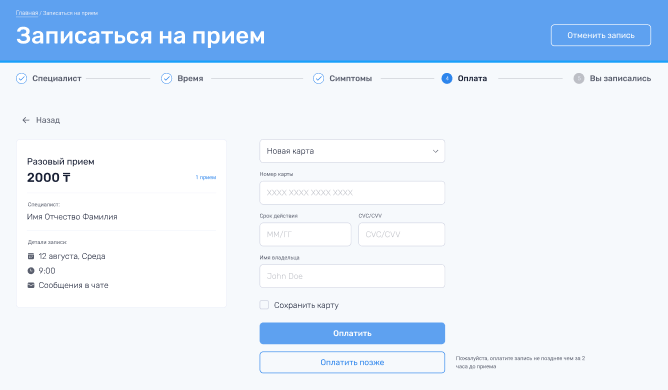


Рисунок 29 – Запись на прием. Этап 3

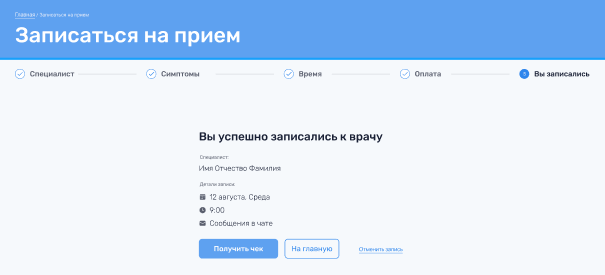


Рисунок 30 – Запись на прием. Этап 4

После успешной записи, в личном кабинете мы увидим запланированную встречу, представленную на рисунке 31.

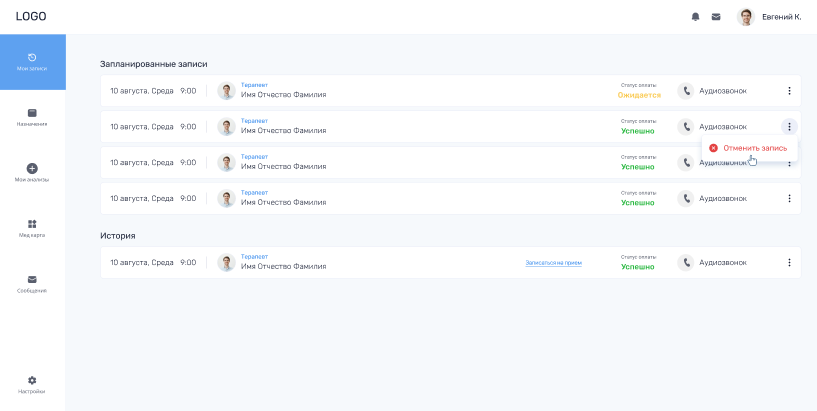


Рисунок 31 – Страница запанированных встреч

В результате тестирования интерфейса пользователя не было выявлено дефектов, вызывающих ошибки работы приложения.

**ЗАКЛЮЧЕНИЕ**

В данном курсовом проекте была разработана WEB-ориентированная система по предоставлению услуг консультирования с врачами. Было разработано три независимых интерфейса: публичный интерфейс и интерфейсы личных кабинетов пациента и врача.

Для интерфейса пациента и врача был организован авторизированный доступ. Интерфейс пользователя-пациента обеспечивает пользователю возможность записи на прием к врачу, заполнения медицинской карты, просмотра списка запланированных встреч, загружать анализы в личный кабинет, общаться с врачами. Интерфейс пользователя-врача позволяет отправить заявку на врача, общаться с пациентами путем просмотра списка запланированных встреч, а также редактировать личный профиль и выставлять время работы в личном кабинете.

Для публичного интерфейса организован свободный доступ. Публичный интерфейс содержит как статические страницы, так и страницы выбора специалистов и формы регистрации.

Разработанный WEB-сервис был протестирован на наличие корректной валидации формы, правильной работы регистрации/авторизации, поиска врачей и просмотра их профиля, протестирован на правильную работу записи на прием.

**СПИСОК ЛИТЕРАТУРЫ И ИНФОРМАЦИОННЫХ РЕСУРСОВ**

1. Node.js [Электронный ресурс]. URL: https://nodejs.org/en/ (дата обращения: 10.04.2021).

2. React - JavaScript-библиотека для создания пользовательских интерфейсов [Электронный ресурс]. URL: https://ru.reactjs.org/ (дата обращения: 10.04.2021).

3. MobX – Библиотека для управления состоянием приложения [Электронный ресурс]. URL: <https://mobx.js.org/README.html> (дата обращения: 10.04.2021).

4. Socket.IO – JavaScript-библиотека для веб-приложений и обмена данными в реальном времени [Электронный ресурс]. URL: https://socket.io/ (дата обращения: 10.04.2021).

**ПРИЛОЖЕНИЕ А**

ДИАГРАММА ГАНТА

| Неделя | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Дата | | Фев  10 | фев | фев | фев | фев | фев | фев | март  10 | март | март | март | март | март | март | март | Апр  1 |
| Этап 1 | Проектирование БД |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Этап 2 | Разработка дизайна |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Этап 3 | Верстка публичной части сайта |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Этап 4 | Разработка функционала авторизации и регистрации |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Этап 5 | Разработка функционала анкетирования врача и пациента |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Этап 6 | Разработка профиля врача |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Этап 7 | Постраничный вывод врачей, вывод специальностей, вывод самых опытных врачей на главную страницу. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Диаграмма Ганта – линейная диаграмма, демонстрирующая очередность и длительность отдельных действий по проекту.

Обозначения:

|  |  |
| --- | --- |
|  | - продолжительность выполнения работ |

**ПРИЛОЖЕНИЕ Б**

КОД ПРИЛОЖЕНИЯ

Файл 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")

);

Файл App.tsx:

import React, { useEffect } from "react";

import { Route, Switch } from "react-router-dom";

import { observer } from "mobx-react";

import { CssBaseline } from "@material-ui/core";

import {

UiKitPage,

HomePage,

DoctorsPage,

DoctorPage,

SignUpPage,

QuestionnairePage,

DashboardPage,

AppointmentPage,

ErrorPage

} from "./pages";

import {

Backdrop,

ScrollHandler,

DialogSignIn,

DialogReset,

DialogEmail,

PrivateRoute

} from "components";

import { useStores } from "stores/useStore";

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 />

<Switch>

<Route exact path="/ui-kit" component={UiKitPage} />

<Route

exact

path={["/", "/home", "/sign-up-confirmation"]}

component={HomePage}

/>

<Route

exact

path={["/doctors", "/doctors/:page"]}

component={DoctorsPage}

/>

<Route exact path="/doctor/:id" component={DoctorPage} />

<PrivateRoute

exact

path="/sign-up"

component={SignUpPage}

isAuthorized={!isAuthorized}

/>

<PrivateRoute

exact

path="/appointment"

component={AppointmentPage}

isAuthorized={isAuthorized}

/>

<PrivateRoute

exact

path="/questionnaire"

component={QuestionnairePage}

isAuthorized={isAuthorized}

/>

<PrivateRoute

path="/dashboard"

component={DashboardPage}

isAuthorized={isAuthorized}

/>

<Route component={ErrorPage} />

</Switch>

<DialogSignIn />

<DialogReset />

<DialogEmail />

</React.Fragment>

);

});

HomeStore.ts:

import { AxiosResponse } from "axios";

import { makeAutoObservable, action } from "mobx";

import { DoctorApi, IGetMostExperienceDoctorSuccessResponse } from "api";

import { HomeDoctor, IHomeStore } from "./interfaces/IHomeStore";

export class HomeStore implements IHomeStore {

doctors: HomeDoctor[] | null = null;

pending: boolean = false;

constructor() {

makeAutoObservable(this);

}

getDoctors = () => {

this.pending = true;

DoctorApi.getDoctorsByCount(10)

.then(

action(

({

data

}: AxiosResponse<IGetMostExperienceDoctorSuccessResponse>) => {

this.doctors = data.data;

}

)

)

.finally(

action(() => {

this.pending = false;

})

);

};

}

SignInStore.ts:

import { makeObservable, action, observable } from "mobx";

import { AxiosError, AxiosResponse } from "axios";

import {

SignInApi,

ISignInPostData,

ISignInSuccessResponse,

ISignInErrorResponse

} from "api";

import {

ISignInStore,

ISignInForm,

KeysOfSignInForm

} from "./interfaces/ISignInStore";

import IStores from "./interfaces";

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,

password: this.signInForm.password

};

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;

};

}

SignUpStore.ts:

import { makeObservable, action, observable } from "mobx";

import { AxiosError, AxiosResponse } from "axios";

import {

SignInApi,

ISignInPostData,

ISignInSuccessResponse,

ISignInErrorResponse

} from "api";

import {

ISignInStore,

ISignInForm,

KeysOfSignInForm

} from "./interfaces/ISignInStore";

import IStores from "./interfaces";

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,

password: this.signInForm.password

};

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;

})

);import { makeObservable, action, observable, reaction } from "mobx";

import { AxiosError } from "axios";

import {

SignUpApi,

ISignUpPostData,

ISignUpErrorResponse,

ISendMailPostData

} from "api";

import {

ISignUpStore,

ISignUpForm,

ISignUpFormErrors,

KeysOfSignUpForm

} from "./interfaces/ISignUpStore";

import IStores from "./interfaces";

import {

isOnlyLetters,

isAdult,

isPhoneNumber,

isEmail,

isPassword

} from "utils/validation";

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,

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;

};

}

};

setFormValue = <K extends KeysOfSignInForm>(key: K, value: ISignInForm[K]) => {

this.signInForm[key] = value;

};

resetForm = () => {

this.signInForm = INITIAL\_SIGN\_IN\_FORM;

this.submissionError = undefined;

};

}

UserStore.ts:

import { AxiosResponse } from "axios";

import { action, makeAutoObservable } from "mobx";

import { UserApi, IGetUserSuccessResponse } from "api";

import { IUserStore, IUser } from "./interfaces/IUserStore";

export class UserStore implements IUserStore {

currentUser: IUser | undefined = undefined;

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

pending: boolean = false;

constructor() {

makeAutoObservable(this);

}

fetchUser = () => {

this.pending = true;

UserApi.getUser()

.then(

action(({ data }: AxiosResponse<IGetUserSuccessResponse>) => {

this.currentUser = data.data;

this.isAuthorized = true;

})

)

.catch(

action(() => {

localStorage.removeItem("accessToken");

this.isAuthorized = false;

})

)

.finally(

action(() => {

this.pending = false;

})

);

};

doLogout = () => {

this.currentUser = undefined;

this.isAuthorized = false;

localStorage.removeItem("accessToken");

};

}

ModalsStore.ts:

import { makeAutoObservable } from "mobx";

import { IModalsStore, IModal, TModalsName } from "./interfaces/IModalsStore";

export class ModalsStore implements IModalsStore {

modals: IModal[] = [

{

name: "sign-in",

isOpen: false

},

{

name: "confirmation",

isOpen: false

},

{

name: "email",

isOpen: false

},

{

name: "reset",

isOpen: false

}

];

constructor() {

makeAutoObservable(this);

}

getModalIsOpen = (modalName: TModalsName) => {

const currentModal = this.modals.find(modal => modal.name === modalName);

return currentModal ? currentModal.isOpen : false;

};

setModalIsOpen = (modalName: TModalsName, isOpen: boolean) => {

const currentModal = this.modals.find(modal => modal.name === modalName);

if (currentModal) {

currentModal.isOpen = isOpen;

}

};

}