Geekbrains

**Разработка веб-сайта Медицинского центра на React**

Программа: Разработчик

Специализация: Frontend разработчик

Баглаев А.Н.

Москва

2025

Содержание

1. Теоретическая часть. Используемые технологии.
   1. Frontend – технологии
      1. React………………………………………………………………………... 3
      2. Redux Toolkit………………………………………………………………..6
      3. React Router………………………………………………………………….8
      4. SCSS………………………………………………………………………..12
      5. Custom Hooks………………………………………………………………16
      6. Адаптивный дизайн……………………………………………………….18
   2. Архитектура проекта…………………………………………………………19
   3. Функциональные возможности ……………………………………………..20
2. Практическая часть………………………………………………………………….29
   1. Структура и взаимодействие модулей………………………………………29
      1. Маршрутизация (AppRouters.js)………………………………………….31
      2. Управление состоянием (Redux store)……………………………………32
      3. Работа компонентов……………………………………………………….36
   2. Ключевые функциональные цепочки………………………………………..39
      1. Аутентификация…………………………………………………………..39
      2. Запись на прием……………………………………………………………40
      3. Личный кабинет врача…………………………………………………….42
   3. Взаимодействие с Api………………………………………………………...43
   4. Особенности реализации …………………………………………………….44
3. Приложениия………………………………………………………………………45

**Теоретическая часть.**

1. **Frontend – технологии.**

***React.***

JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций.

**Особенности React**

* Компонентный подход.

Это методология разработки интерфейсов, при которой UI разбивается на независимые, переиспользуемые части (компоненты). В React каждый компонент отвечает за свою часть интерфейса, может содержать собственное состояние (state) и логику, а также вкладывать другие компоненты.

Основные принципы компонентного подхода в React:

**Разделение на компоненты**. Интерфейс делится на мелкие, логически связанные блоки (кнопки, формы, списки и т. д.). Каждый компонент отвечает только за свою часть функциональности.

**Переиспользование.** Компоненты можно использовать многократно в разных частях приложения. Пример: кнопка <Button /> может применяться везде, где нужна кнопка.

**Изолированность.** Компонент управляет своим состоянием (state) и стилями. Внешние зависимости задаются через props (входные параметры).

**Композиция**. Компоненты можно комбинировать, вкладывая друг в друга (как Lego).

* Виртуальный DOM

Это концепция оптимизации производительности в React, которая минимизирует количество прямых манипуляций с реальным DOM (Document Object Model). Как это работает?

Создание виртуального DOM. При изменении состояния (state или props) React создает новый виртуальный DOM (легковесную копию реального DOM в виде JavaScript-объектов).

Сравнение (Diffing Algorithm). React сравнивает новый виртуальный DOM с предыдущим (реконсиляция) и находит минимальный набор изменений.

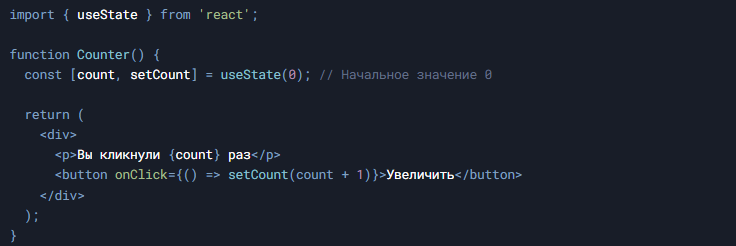
Обновление реального DOM. Только найденные изменения применяются к реальному DOM, что делает процесс обновления более эффективным.

Виртуальный DOM — ключевая оптимизация React, позволяющая делать интерфейсы быстрыми без ручного управления DOM.

* Хуки (useState, useEffect, useContext).

Хуки — это функции в React, которые позволяют использовать состояние и другие возможности React без написания классов. Они появились в версии React 16.8 и значительно упростили разработку функциональных компонентов.

**useState** — управление состоянием. Позволяет добавлять локальное состояние в функциональные компоненты. Возвращает массив из двух элементов: Текущее значение состояния. Функция для его обновления. При вызове функции обновления компонент перерендеривается.



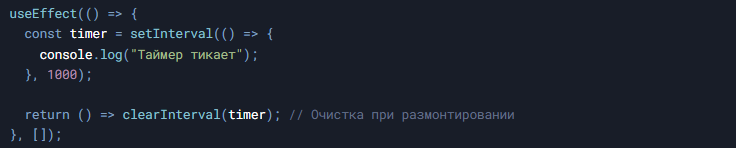
**useEffect** — выполнение побочных эффектов. Позволяет выполнять побочные эффекты (запросы к API, подписки, ручное изменение DOM) в функциональных компонентах.

Принимает функцию, которая выполняется после каждого рендера. Можно контролировать, когда эффект будет срабатывать, передавая массив зависимостей. Возвращает функцию очистки (если нужна).

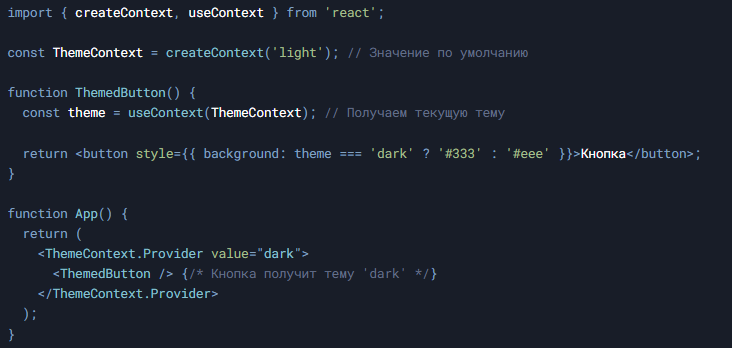








**useContext** — доступ к контексту. Позволяет получить доступ к значению контекста без использования Consumer. Принимает объект контекста (созданный через React.createContext).Возвращает текущее значение контекста.

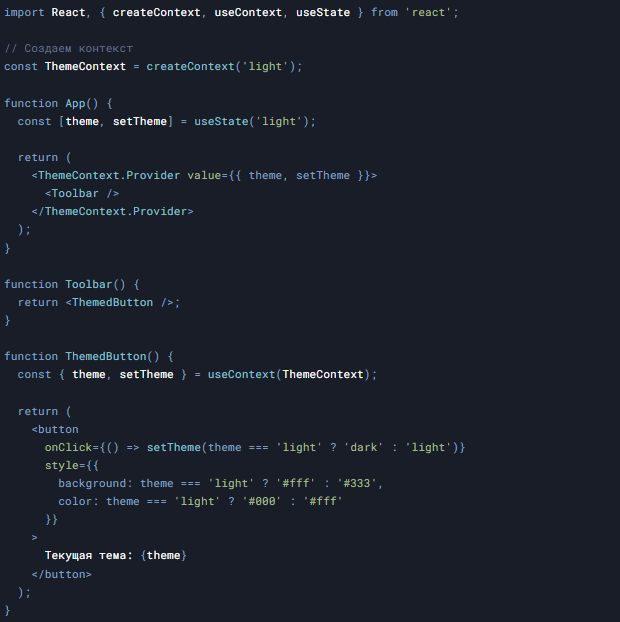


* Контекст

Это механизм для передачи данных через дерево компонентов без необходимости явно передавать пропсы на каждом уровне. Он позволяет делиться данными, которые можно считать "глобальными" для всего приложения или его части.

Проблема, которую решает контекст. Пропс-дриллинг (prop drilling) - когда нужно передавать данные через множество промежуточных компонентов, которые сами эти данные не используют. Глобальные данные (тема, настройки, аутентификация и т.д.)

Создается контекст с начальным значением. Провайдер (Provider) оборачивает часть дерева компонентов и предоставляет значение контекста. Компоненты-потребители (Consumer) или хуки могут читать это значение.



***Redux Toolkit.***

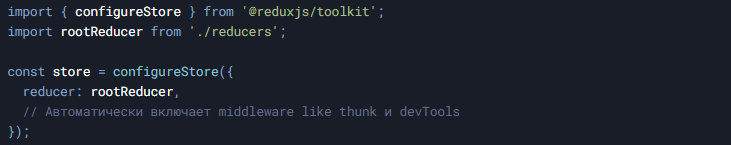
Redux Toolkit (RTK) - это официальный, рекомендуемый подход для работы с Redux в React-приложениях. Он был создан для решения трех основных проблем "классического" Redux:

* Слишком много шаблонного кода (boilerplate)
* Сложность настройки хранилища (store)
* Необходимость подключения множества дополнительных библиотек

Основные возможности Redux Toolkit.

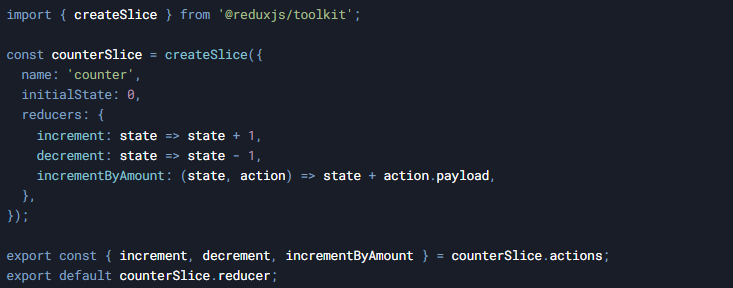
**configureStore()**

Упрощает создание Redux-хранилища с хорошими настройками по умолчанию.



**createSlice()**

Позволяет создавать редюсеры и действия (actions) вместе в одном месте, значительно сокращая boilerplate-код.



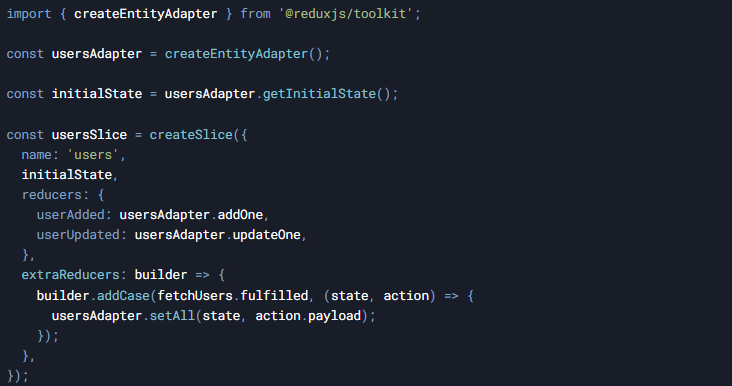
**createAsyncThunk()**

Упрощает работу с асинхронными операциями (например, запросами к API).



**createEntityAdapter**

Предоставляет стандартизированный способ хранения данных в нормализованном виде.



Преимущества Redux Toolkit

* Меньше кода: Сокращает количество шаблонного кода на 50-75%
* Лучшие практики "из коробки": Включает thunk middleware, DevTools и другие полезные инструменты
* Иммутабельные обновления: Использует Immer.js, позволяя писать код, который "мутирует" состояние, но на самом деле создает новые иммутабельные состояния
* Типизация: Хорошая поддержка TypeScript
* Модульность: Позволяет легко разделять логику на "слайсы" (slices)

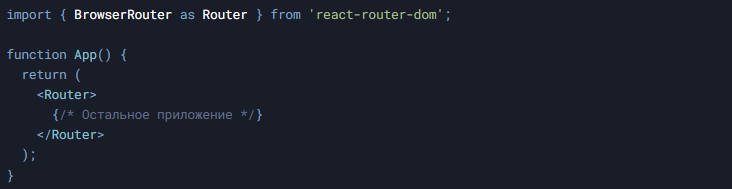
***React Router***

React Router - это стандартная библиотека для маршрутизации в React-приложениях. Она позволяет создавать одностраничные приложения (SPA) с навигацией между различными "страницами" без полной перезагрузки.

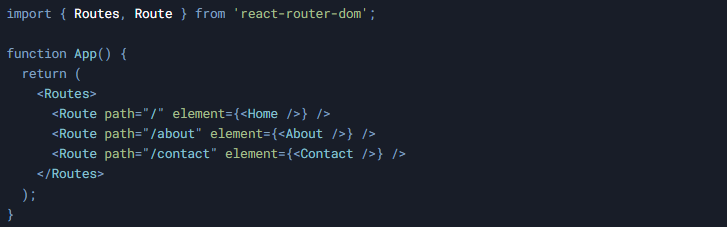
React Router предоставляет мощные инструменты для создания сложных навигационных систем в React-приложениях, сохраняя при этом высокую производительность и удобство разработки.

Основные компоненты.

**BrowserRouter.** Это роутер, который использует HTML5 History API для синхронизации UI с URL.



**Routes и Route.** Routes - это контейнер для маршрутов, Route определяет конкретный маршрут.

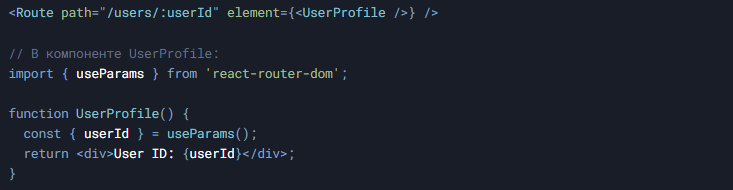
****

**Link и NavLink.** Компоненты для навигации вместо тега <a>.



Динамические маршруты.

Параметры в URL.

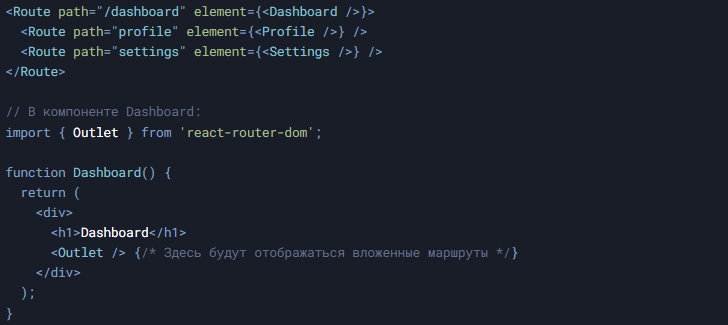


Необязательные параметры



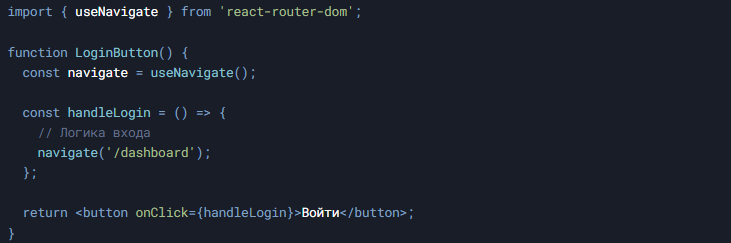
Вложенные маршруты

Позволяют создавать сложные иерархии маршрутов.

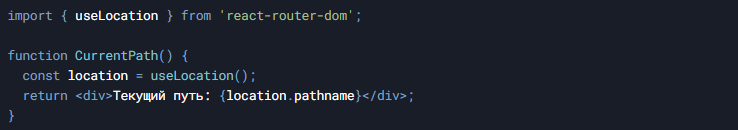


Хуки React Router

**useNavigate**. Для программной навигации.

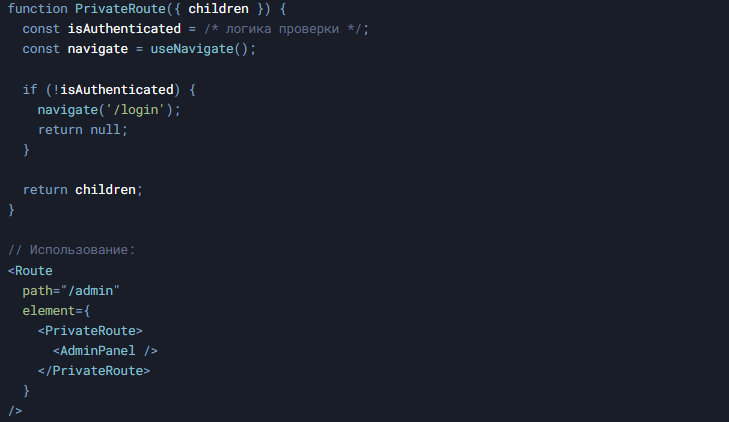


**useLocation.** Доступ к текущему местоположению (URL).



Защищённые маршруты.

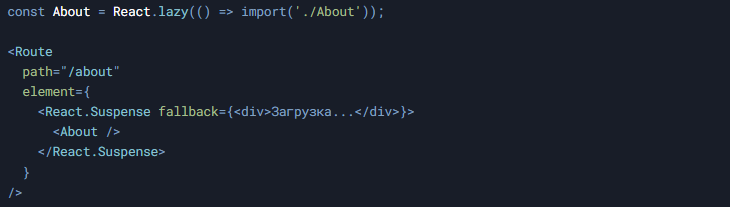
Пример реализации аутентификации:



Обработка 404 (Not Found).



Ленивая загрузка (Code Splitting).



Лучшие практики

* Организация маршрутов: Храните конфигурацию маршрутов в отдельном файле
* Анализ бандла: Используйте ленивую загрузку для больших компонентов
* Обработка ошибок: Создайте компонент для обработки ошибок маршрутизации
* Прокрутка: Настройте автоматическую прокрутку вверх при смене маршрута
* SEO: Используйте <Helmet> для управления мета-тегами на разных страницах

***SCSS***

SCSS - препроцессор CSS, который позволяет использовать переменные, вложенности, миксины и другие возможности для более удобной работы со стилями.

Зачем нужен SCSS? Обычный CSS имеет ряд ограничений:

* Повторяющийся код – одни и те же значения (цвета, отступы, шрифты) приходится прописывать многократно.
* Отсутствие модульности – большие файлы стилей сложно поддерживать.
* Нет возможности использовать логику – нельзя применять условия, циклы или функции.

SCSS решает эти проблемы, предлагая:

* Переменные – хранение часто используемых значений в одном месте.
* Вложенность – более понятная структура стилей.
* Миксины и функции – переиспользуемые блоки кода.
* Импорт файлов – разбиение стилей на логические модули.
* Математические операции – вычисления прямо в стилях.
* Логические конструкции – условия и циклы для динамических стилей.

Ключевые особенности SCSS.

*Переменные*. Позволяют задавать значения (цвета, размеры, шрифты) один раз и использовать их многократно. Это особенно полезно в больших проектах, где нужно быстро менять дизайн-систему.

*Вложенность селекторов*. Позволяет группировать стили, относящиеся к одному блоку, что делает код более читаемым. Например, стили для меню можно писать внутри одного родительского селектора nav, а не дублировать его каждый раз.

*Миксины (примеси)*. Это шаблоны стилей, которые можно переиспользовать с разными параметрами. Например, можно создать миксин для скругления углов и применять его к разным элементам, меняя радиус.

*Наследование (@extend)*. Позволяет одному классу наследовать стили другого, уменьшая дублирование кода. Например, можно создать базовый класс alert и расширять его для разных типов уведомлений (alert-success, alert-error).

*Импорт файлов*. SCSS поддерживает разбиение стилей на отдельные файлы (например, \_variables.scss, \_buttons.scss, \_layout.scss), которые затем компилируются в один CSS. Это улучшает организацию кода.

*Условные операторы и циклы*. Можно использовать @if, @else, @for, @each и @while для генерации динамических стилей. Например, можно автоматически создавать классы для сетки или менять стили в зависимости от условий.

*Математические операции*. SCSS позволяет выполнять вычисления прямо в стилях, что удобно для адаптивных размеров, отступов и других значений.

Преимущества SCSS перед CSS

* Снижение дублирования кода – за счёт переменных, миксинов и наследования.
* Лучшая читаемость – вложенность и модульность делают стили более понятными.
* Лёгкость поддержки – изменение одного значения в переменной обновит все связанные стили.
* Динамические стили – можно использовать логику, циклы и условия.
* Совместимость с CSS – любой валидный CSS-код является валидным SCSS.

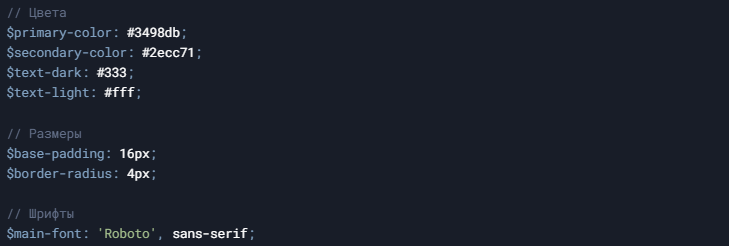
Где применяется SCSS.

* Фронтенд-разработка (React, Vue, Angular).
* Фреймворки (Bootstrap, Tailwind с плагинами).
* Большие проекты – где важна поддерживаемость кода.

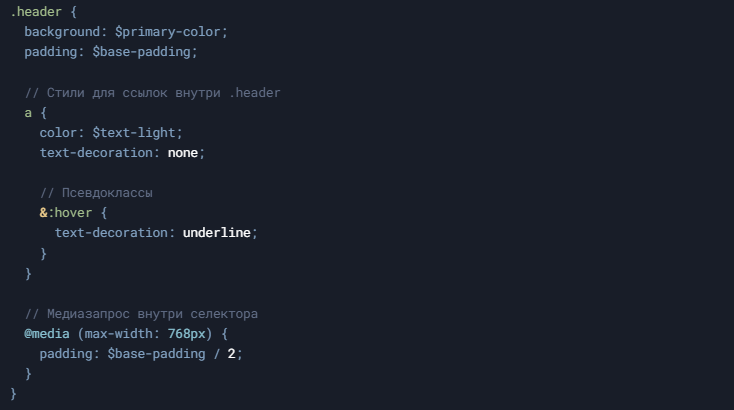
SCSS – это мощный инструмент, который делает вёрстку удобнее, а код – чище. Он подходит как для небольших проектов, так и для крупных приложений, где важна структура и поддерживаемость стилей. Если вы ещё не используете SCSS, стоит попробовать – после него верстать на чистом CSS будет менее комфортно!

Примеры кода SCSS.

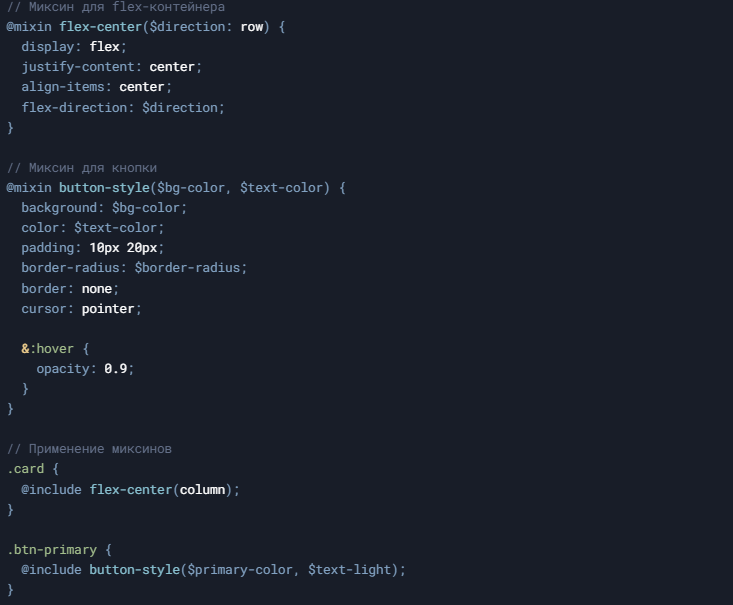
Переменные. Можно быстро изменить цветовую схему или отступы во всём проекте, правя всего 1 строку.



Вложенность (Nesting). Удобная группировка стилей для элементов внутри родителя. Медиазапросы можно писать прямо внутри селектора.



Миксины (Mixin). Переиспользуемые блоки стилей (как функции в JS). Можно передавать параметры (например, $direction или цвета кнопки).



Наследование (@extend). Избегаем дублирования кода (общие стили .message не повторяются).



***Custom Hooks***

Custom Hooks (пользовательские хуки) — это функции, которые позволяют выносить логику из компонентов в переиспользуемые модули. Они помогают избежать дублирования кода, делают его чище и удобнее для поддержки.

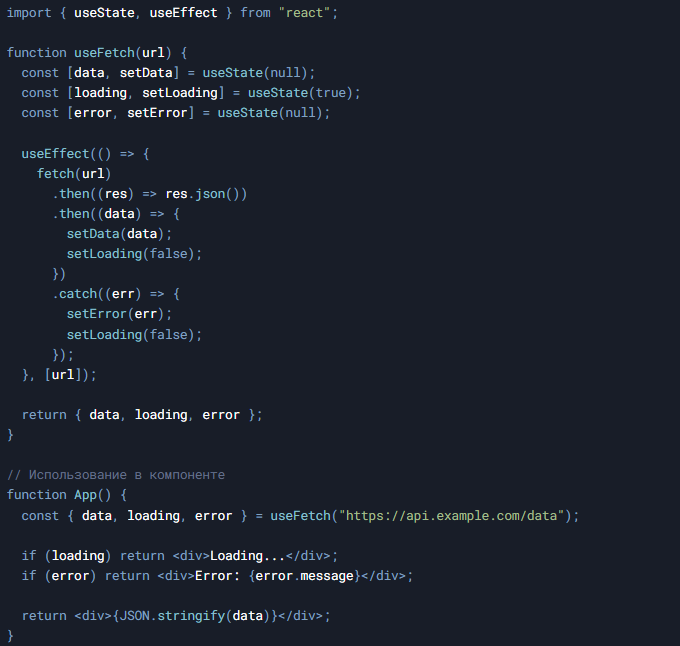
Основные правила Custom Hooks

* Имя должно начинаться с use (например, useFetch, useLocalStorage).
* Могут вызывать другие хуки (useState, useEffect, useContext и т. д.).
* Не могут использоваться в условиях или циклах (как и обычные хуки).

Зачем нужны Custom Hooks?

* Уменьшают дублирование кода – логика выносится в отдельный хук.
* Делают компоненты чище – бизнес-логика скрыта в хуке.
* Позволяют переиспользовать логику между разными компонентами.

**useFetch** – хук для работы с API



**useLocalStorage** – хук для работы с localStorage



**useToggle** – хук для булевых переключателей



Custom Hooks в React — мощный инструмент для организации логики и переиспользования кода. Они помогают:

* Избегать дублирования
* Делать компоненты чище
* Упрощать тестирование

***Адаптивный дизайн***

Адаптивный дизайн (англ. Responsive Web Design, RWD) — это подход к созданию веб-сайтов и интерфейсов, при котором страницы автоматически подстраиваются под размер экрана и устройство пользователя (смартфон, планшет, ноутбук, десктоп).

Основные принципы адаптивного дизайна.

**Гибкая сетка (Fluid Grid).** Элементы макета (колонки, блоки) меняют ширину пропорционально экрану, а не имеют фиксированные размеры (пиксели). Используются относительные единицы (%, vw, fr) вместо px.

**Гибкие изображения и медиа (Flexible Media**). Изображения и видео масштабируются так, чтобы не выходить за границы контейнера. Пример: max-width: 100%; height: auto;

**Медиазапросы (Media Queries**). CSS-правила применяются в зависимости от характеристик устройства (ширина экрана, ориентация, плотность пикселей).



**Мобильный-first подход.** Сначала проектируется версия для мобильных, а затем добавляются стили для более широких экранов.

Почему это важно?

* Удобство пользователей: сайт хорошо выглядит на любом устройстве.
* SEO-оптимизация: Google ранжирует адаптивные сайты выше.
* Экономия: один сайт вместо отдельных версий для ПК и мобильных.

Адаптивный дизайн — стандарт современной веб-разработки, без него не обходится ни один профессиональный проект.

1. **Архитектура проекта**

Проект следует принципам Feature-Sliced Design с элементами модульной структуры:

* Компоненты разделены по функциональности (общие, врачебные, пациентские)
* Страницы представляют собой контейнеры для компонентов
* Redux-хранилище организовано по модульному принципу с отдельными слайсами
* Утилиты и хуки вынесены в отдельные директории для повторного использования.

Feature-Sliced Design (FSD) — это методология проектирования фронтенд-приложений, основанная на разделении кода по бизнес-логике (фичам) и техническим слоям. Она помогает создавать масштабируемые, поддерживаемые и предсказуемые приложения.

Основные принципы FSD

Разделение по бизнес-логике (фичам), а не по типам файлов. В отличие от классического подхода (где папки components, pages, styles лежат вперемешку), FSD группирует код по функциональности.

Слоистая архитектура. Каждый модуль делится на слои, что делает зависимости предсказуемыми.

Однонаправленные зависимости. Зависимости идут только сверху вниз.

Изолированность модулей. Каждая фича/сущность самодостаточна (имеет свой API, сторибук, тесты). Можно вынести модуль в отдельный пакет (микросервисная идеология).

Гибкость и масштабируемость. Легко добавлять/удалять фичи без побочных эффектов. Подходит для монолитов и микрофронтендов.

Преимущества FSD.

* Масштабируемость — кодовая база растёт без хаоса
* Поддерживаемость — фичи изолированы, легко рефакторить
* Тестируемость — модули независимы, можно тестировать по слоям
* Повторное использование — shared и entities доступны везде
* Чёткие зависимости — нет циклических импортов

Feature-Sliced Design — это современный способ структурирования фронтенда, который:

* Делает код понятнее за счёт разделения по бизнес-логике
* Уменьшает связанность модулей
* Позволяет проекту расти без потери поддерживаемости

1. **Функциональные возможности**

Приложение может включать следующие возможности:

***Аутентификация и авторизация пользователей (врачей и пациентов)***

Регистрация пользователей

Пациенты:

* Форма регистрации с полями: имя, фамилия, email, телефон, дата рождения, пол, пароль (с подтверждением).
* Валидация email и телефона (уникальность, формат).
* Подтверждение email (отправка ссылки для активации аккаунта).
* Возможность регистрации через OAuth (Google, Facebook и т. д.).
* Создание профиля пациента с базовыми данными.

Врачи:

* Форма регистрации с дополнительными полями: специализация, квалификация, номер лицензии, место работы.
* Верификация данных администратором или через прикрепление документов (скан лицензии).
* Временная блокировка аккаунта до подтверждения.
* Автоматическое создание профиля врача после модерации.

Стандартная форма входа по email/телефону и паролю.

* Восстановление пароля через email/SMS (временная ссылка или код).
* Двухфакторная аутентификация (2FA) для врачей (SMS, Google Authenticator).
* Вход через OAuth-провайдеров.
* Защита от брутфорса (капча, временная блокировка после N попыток).

Авторизация и ролевая модель

Роли:

* Пациент (доступ к своему профилю, записи к врачам, истории приёмов).
* Врач (доступ к расписанию, пациентам, медицинским записям, назначениям).
* Администратор (управление пользователями, верификация врачей, статистика).
* Модератор (проверка контента, жалобы).

Права:

* Пациент может записываться к врачу, просматривать свои данные.
* Врач может редактировать расписание, назначать лечение, но не может изменять личные данные пациента.
* Администратор имеет полный доступ к управлению системой.
* Проверка прав при каждом запросе к API (например, через JWT или сессии).

Управление профилем

Пациент:

* Редактирование личных данных (кроме email без подтверждения).
* Загрузка аватара.
* Настройка уведомлений (email/SMS о записях, напоминания).

Врач:

* Редактирование профессиональной информации (специализация, график работы).
* Управление расписанием (слоты, выходные).
* Прикрепление документов (дипломы, сертификаты).

Безопасность

* Хранение паролей в хешированном виде (bcrypt/scrypt).
* HTTPS для всех запросов.
* Регулярный аудит логинов (отображение последних сессий).
* Возможность принудительного разлогина со всех устройств.
* Автоматический выход после периода неактивности (например, 30 минут).

API-эндпоинты (пример)

* `POST /api/auth/register` – регистрация.
* `POST /api/auth/login` – вход.
* `POST /api/auth/logout` – выход.
* `POST /api/auth/forgot-password` – запрос на сброс пароля.
* `POST /api/auth/reset-password` – установка нового пароля.
* `GET /api/profile` – получение данных профиля (разные данные для врача/пациента).
* `PUT /api/profile` – обновление профиля.

Дополнительные функции

* История действий пользователя (логирование критичных операций).
* Уведомления о подозрительной активности (например, вход с нового устройства).
* Интеграция с CRM для врачей (если есть).
* Возможность привязки нескольких пациентов к одному аккаунту (семейный доступ).

Технические детали

* Использование JWT для stateless-аутентификации или сессий.
* Redis для хранения активных сессий/токенов.
* Микросервисная архитектура (отдельный сервис аутентификации).
* Генерация PDF-отчётов для врачей (история авторизаций).

Сценарии использования

* Пациент регистрируется, подтверждает email, записывается к врачу.
* Врач входит в систему, проверяет расписание, получает уведомление о новой записи.
* Админ блокирует неверифицированного врача.

***Просмотр списка врачей с фильтрацией***

Данный блок позволяет пользователям просматривать список врачей, доступных в системе, с возможностью фильтрации по различным параметрам (специальность, отделение, рейтинг и др.).

Основные элементы интерфейса

* Список врачей – отображается в виде карточек или таблицы.
* Фильтры – панель с параметрами для сортировки и поиска нужных специалистов.
* Поисковая строка – для быстрого поиска по ФИО врача.
* Кнопки действий – запись на прием, просмотр профиля врача.

Параметры фильтрации

Пользователь может фильтровать список по:

* Специальности (терапевт, хирург, кардиолог и др.).
* Отделению (терапия, хирургия, педиатрия и т. д.).
* Рейтингу (сортировка по оценкам пациентов).
* Стажу работы (например, "более 10 лет").
* Доступности записи (только те, у кого есть свободные окна).
* Полу врача (мужчина/женщина – опционально).
* Местоположению (если клиник несколько).

Дополнительные возможности

* Сортировка по алфавиту, рейтингу, стажу.
* Просмотр подробной информации о враче (образование, опыт, отзывы).
* Быстрая запись на прием прямо из списка.
* Избранное – добавление врачей в личный список для быстрого доступа.

Логика работы

1. Пользователь открывает раздел "Врачи".

2. Система загружает полный список врачей (или только доступных).

3. Пользователь применяет фильтры – список обновляется в реальном времени.

4. При нажатии на карточку врача открывается его профиль.

Технические аспекты

* Адаптивность – корректное отображение на ПК, планшетах и смартфонах.
* Оптимизация – быстрая загрузка списка даже при большом количестве врачей.
* Интеграция с БД – актуальные данные о врачах и их расписании.

Этот блок упрощает поиск нужного специалиста и делает процесс записи на прием удобным и быстрым.

***Запись на прием к врачу с выбором даты и времени***

Выбор специализации врача

* Выпадающий список с возможностью поиска.
* При выборе специализации подгружаются доступные врачи.
* Дополнительно: краткое описание специализации (по клику на "?").

Выбор врача

Динамический список врачей выбранной специализации.

Отображение:

* ФИО врача
* Фото (если есть)
* Рейтинг (звезды + отзывы)
* Специализация, категория, стаж
* Кнопка "Подробнее" (полная информация о враче)

3. Выбор даты

* Интерактивный календарь с доступными датами.
* Недоступные даты (выходные, отпуск врача) заблокированы.
* Подсветка ближайших свободных дат.
* Возможность переключения между месяцами.

Выбор времени

* После выбора даты загружается список свободных временных слотов.
* Формат: HH:MM (например, 09:00, 09:30, 10:15).
* Если слот занят – он неактивен (или отмечен как "Нет мест").
* Возможность фильтрации (утро/день/вечер).

Ввод данных пациента

* ФИО (обязательное поле, проверка на корректность).
* Номер телефона (маска ввода, проверка формата).
* Email (проверка на валидность).
* Жалобы/симптомы (необязательное поле, ограничение по символам).
* Возможность прикрепить файлы (анализы, снимки).

Подтверждение записи

* Чекбокс согласия на обработку персональных данных (обязательный).
* Кнопка "Записаться" (после нажатия – проверка заполнения полей).
* После успешной записи:
* Показ подтверждения (дата, время, врач).
* Отправка уведомления на email/SMS.
* Ссылка на добавление события в календарь (Google Calendar, Outlook).
* Кнопка "Перенести/Отменить запись".

***Личные кабинеты для врачей и пациентов***

Функционал личных кабинетов для врачей и пациентов.

Личный кабинет пациента.

Личный кабинет пациента предоставляет удобный доступ к медицинским данным, записям на прием и коммуникации с врачами.

Основные функции:

* Регистрация и авторизация (через email, телефон или соцсети).
* Профиль пациента (личные данные, контакты, медицинская карта).
* Запись на прием (онлайн-расписание врачей, выбор специалиста, времени).
* Электронная медицинская карта (история болезней, диагнозы, назначения).
* Доступ к результатам анализов (лабораторные исследования, снимки).
* Онлайн-консультации (чат, видеозвонки с врачом).
* Напоминания о приемах (уведомления по email/SMS).
* Оплата услуг (онлайн-платежи за приемы, страховые выплаты).
* Обратная связь (оценка работы врачей, жалобы и предложения).

Личный кабинет врача.

Кабинет врача предназначен для управления приемами, пациентами и медицинской документацией.

Основные функции:

* Авторизация и верификация (доступ только для подтвержденных специалистов).
* Расписание приемов (график работы, запись пациентов).
* Управление пациентами (история обращений, медицинские записи).
* Электронные рецепты и направления (формирование и отправка пациенту).
* Доступ к медицинским данным (анализы, снимки, история болезни).
* Онлайн-консультации (чат, видеоконсультации).
* Интеграция с лабораториями и клиниками (получение результатов анализов).
* Отчетность и статистика (анализ приемов, нагрузка).
* Настройки профиля (личные данные, специализация, график работы).

Преимущества системы.

* Удобство – пациенты записываются онлайн, врачи экономят время.
* Безопасность – защита персональных данных (соответствие GDPR/ФЗ-152).
* Доступность – удаленные консультации, электронные документы.
* Автоматизация – напоминания, онлайн-оплата, цифровая медкарта.

Такой функционал повышает эффективность работы медучреждений и комфорт пациентов.

***Управление записями на прием***

Этот модуль позволяет администраторам, сотрудникам и клиентам управлять записями на прием в различных сферах: медицина, услуги, консультации и т. д.

Создание записи на прием

* Выбор услуги/специалиста – клиент или администратор выбирает нужную услугу, врача, мастера и т. д.
* Выбор даты и времени – доступны только свободные слоты (учитывается график работы, перерывы, праздники).
* Добавление клиента – если клиент новый, создается профиль; если уже есть в системе – выбирается из базы.
* Подтверждение записи – отправка уведомления клиенту (SMS, email, мессенджеры).

Просмотр и редактирование записей

* Календарь записей – отображение всех записей с фильтрами по дате, специалисту, статусу.
* Поиск записей – по ФИО клиента, номеру телефона, email.
* Изменение данных – перенос времени, замена специалиста, добавление комментариев.
* Отмена записи – с уведомлением клиента и освобождением слота.

Управление расписанием

* Настройка графика работы – рабочие дни, часы приема, перерывы.
* Блокировка времени – если специалист не доступен (отпуск, болезнь).
* Шаблоны расписания – для быстрой настройки повторяющихся интервалов.

Уведомления и напоминания

* Автоматические напоминания – клиенту (за день, за час до приема).
* Оповещения об отмене/переносе – сразу при изменении записи.
* Интеграция с мессенджерами/email/SMS – рассылка через выбранные каналы.

Аналитика и отчеты

* Статистика посещений – количество записей, отмен, загруженность специалистов.
* Финансовые отчеты – выручка по услугам, специалистам, периодам.
* Экспорт данных – в Excel, PDF для дальнейшей обработки.

Интеграции

* CRM/ERP-системы – синхронизация данных о клиентах.
* Онлайн-оплата – возможность предоплаты или оплаты при записи.
* Телефония – автоматический звонок, подтверждение записи через IVR.

Доступы и роли

* Администратор – полный доступ к управлению.
* Специалист (врач, мастер) – просмотр своих записей, отметка о выполнении.
* Клиент – запись через личный кабинет или мобильное приложение.

Этот функционал оптимизирует процесс записи, сокращает количество "ноу-шоу" (неявок) и повышает удобство для клиентов и сотрудников.

***Отображение информации о врачах и пациентах***

Данный функционал позволяет пользователям системы просматривать данные о врачах и пациентах в удобном и структурированном виде.

Отображение информации о врачах

Пользователь может просматривать список врачей с возможностью фильтрации и сортировки. Для каждого врача отображаются следующие данные:

* ФИО врача
* Специализация (терапевт, хирург, кардиолог и т. д.)
* Квалификация (категория, ученая степень)
* График работы (дни и часы приема)
* Контактные данные (номер телефона, email)
* Кабинет (номер или место приема)
* Стаж работы
* Фото (опционально)

Дополнительные возможности:

* Поиск врача по ФИО или специализации.
* Фильтрация по отделению или доступному времени.
* Возможность просмотра отзывов о враче (если предусмотрено системой).

Отображение информации о пациентах.

Доступ к данным пациентов обычно ограничен и предоставляется только авторизованным сотрудникам (врачам, администраторам). Отображаемая информация включает:

* ФИО пациента
* Дата рождения и возраст
* Пол
* Контактные данные (телефон, email, адрес)
* Медицинская карта (история болезней, диагнозы)
* Прикрепленный врач (если есть)
* Дата последнего визита
* Аллергии и противопоказания
* Страховой полис\*\* (номер, срок действия)

Дополнительные возможности:

* Поиск пациента по ФИО, номеру полиса или дате рождения.
* Фильтрация по возрасту, полу или лечащему врачу.
* Экспорт данных (в PDF или Excel, если разрешено политикой клиники).
* Быстрый доступ к истории посещений и назначенным препаратам.

Интерфейс и взаимодействие.

* Табличное представление – список врачей/пациентов с возможностью сортировки.
* Карточка профиля – детальная информация при выборе конкретного врача/пациента.
* Печать и экспорт – формирование документов для отчетности.
* Редактирование (для администраторов) – возможность обновлять данные (если есть права доступа).

Безопасность и доступ

* Данные пациентов доступны только авторизованному медперсоналу.
* Врачи видят только своих пациентов (если не имеют расширенных прав).
* Ведется журнал доступа к конфиденциальной информации.

Этот функционал обеспечивает удобное управление медицинскими данными и ускоряет работу сотрудников клиники.

**В рамках проекта по текущей дипломной работе будут реализованы не все возможности и функции.**

**Практическая часть.**

**Общая структура проекта**

```

medical-center/

├── public/                  # Статические файлы

│   ├── index.html           # Основной HTML файл

│   └── favicon.ico          # Иконка сайта

│

├── src/

│   │

│   ├── components/          # Компоненты

|   |   |\_\_ AppointmentsList/

|   |   |   |\_\_ AppointmentsList.jsx

|   |   |

│   │   ├── auth/            # Компоненты аутентификации

│   │   │   ├── AuthForm/    # Форма входа/регистрации

|   |   |   |   |\_\_AuthForm.jsx

|   |   |   |   |\_\_AuthForm.scss

│   │   │   └── AuthSwitch/  # Переключатель между формами

│   │   │       |\_\_AuthSwitch.jsx

|   |   |       |\_\_AuthSwitch.scss

│   │   ├── common/          # Общие компоненты

│   │   │   ├── Button/      # Кнопка

|   |   |   |   |\_\_Button.jsx

|   |   |   |   |\_\_Button.scss

│   │   │   ├── DatePicker/  #

|   |   |   |   |\_\_DataPicker.jsx

|   |   |   |   |\_\_DataPicker.scss

│   │   │   ├── Header/      # Шапка сайта

|   |   |   |   |\_\_Header.jsx

|   |   |   |   |\_\_Header.scss

│   │   │   ├── Footer/      # Подвал сайта

|   |   |   |   |\_\_Footer.jsx

|   |   |   |   |\_\_Footer.scss

│   │   │   ├── Input/       # Поле ввода

|   |   |   |   |\_\_Input.jsx

|   |   |   |   |\_\_Input.scss

|   |   |   |\_\_ LoadingScreen

|   |   |   |   |\_\_LoadingScreen.jsx

|   |   |   |   |\_\_LoadingScreen.scss

│   │   │   ├── Modal/       # Модальное окно

|   |   |   |   |\_\_Modal.jsx

|   |   |   |   |\_\_Modal.scss

│   │   │   ├── Navigation/  # Навигация

|   |   |   |   |\_\_Navigation.jsx

|   |   |   |   |\_\_Navigation.scss

│   │   │   ├── Notificatiion/  #

|   |   |   |   |\_\_Notification.jsx

|   |   |   |   |\_\_Notification.scss

│   │   │   ├── Select/      # Выпадающий список

|   |   |   |   |\_\_Select.jsx

|   |   |   |   |\_\_Select.scss

│   │   │   |── TimeSlotPicker

|   |   |   |   |\_\_TimeSlotPicker.jsx

|   |   |   |   |\_\_TimeSlotPicker.scss

│   │   │

│   │   ├── doctor/          # Компоненты врача

│   │   │   ├── DoctorCard/  # Карточка врача

|   |   |   |   |\_\_DoctorCard.jsx

|   |   |   |   |\_\_DoctorCard.scss

│   │   │   ├── DoctorInfo/  # Информация о враче

|   |   |       |\_\_DoctorInfo.jsx

|   |   |       |\_\_DoctorInfo.scss

│   │   │

│   │   |── patient/         # Компоненты пациента

│   │   |    ├── PatientInfo/ # Информация о пациенте

|   |   |        |\_\_PatientInfo.jsx

|   |   |        |\_\_PatientInfo.jsx

|   |   |\_\_PersistLoading.jsx

│   │

│   ├── hooks/               # Кастомные хуки

│   │   ├── useLocalStorage.js # Работа с localStorage

│   │   └── useAuth.js       # Логика аутентификации

│   │

│   ├── pages/               # Страницы приложения

│   │   ├── Home/            # Главная страница

|   |   |   |\_\_Home.jsx

|   |   |   |\_\_Home.scss

│   │   ├── Doctors/         # Список врачей

|   |   |   |\_\_Doctors.jsx

|   |   |   |\_\_Doctors.scss

│   │   ├── DoctorCabinet/   # Кабинет врача

|   |   |   |\_\_DoctorCabinet.jsx

|   |   |   |\_\_DoctorCabinet.scss

│   │   ├── PatientCabinet/  # Кабинет пациента

|   |   |   |\_\_PatientCabinet.jsx

|   |   |   |\_\_PatientCabinet.scss

│   │   ├── Auth/            # Страница аутентификации

|   |   |   |\_\_Auth.jsx

|   |   |   |\_\_Auth.scss

│   │   |── NotFound/        # Страница 404

|   |   |   |\_\_NotFound.jsx

|   |   |   |\_\_NotFound.scss

│   │

│   ├── store/               # Redux хранилище

│   │   ├── slices/          # Redux слайсы

│   │   │   ├── appointmentsSlice.js #

│   │   │   ├── authSlice.js # Аутентификация

│   │   │   ├── dataSlice.js # Тестовые данные

│   │   │   ├── doctorSlice.js #

│   │   │   ├── notificationSlice.js #

│   │   │   ├── patientSlice.js #

│   │   │   ├── specialitesSlice.js #

│   │   │

│   │   ├── selectors.js     # Селекторы для Redux

│   │   └── store.js         # Конфигурация хранилища

│   │

│   ├── utils/               # Вспомогательные функции

│   │   ├── constants.js     # Константы приложения

│   │   ├── helpers.js       # Вспомогательные функции

│   │   └── storage.js       # Работа с хранилищем

│   │

│   ├── App.js               # Корневой компонент

│   ├── AppRouter.js         # Маршрутизация

│   ├── index.js             # Точка входа

│   └── setupTests.js        # Настройка тестов

│

├── .gitignore               # Игнорируемые файлы Git

├── package.json             # Зависимости и скрипты

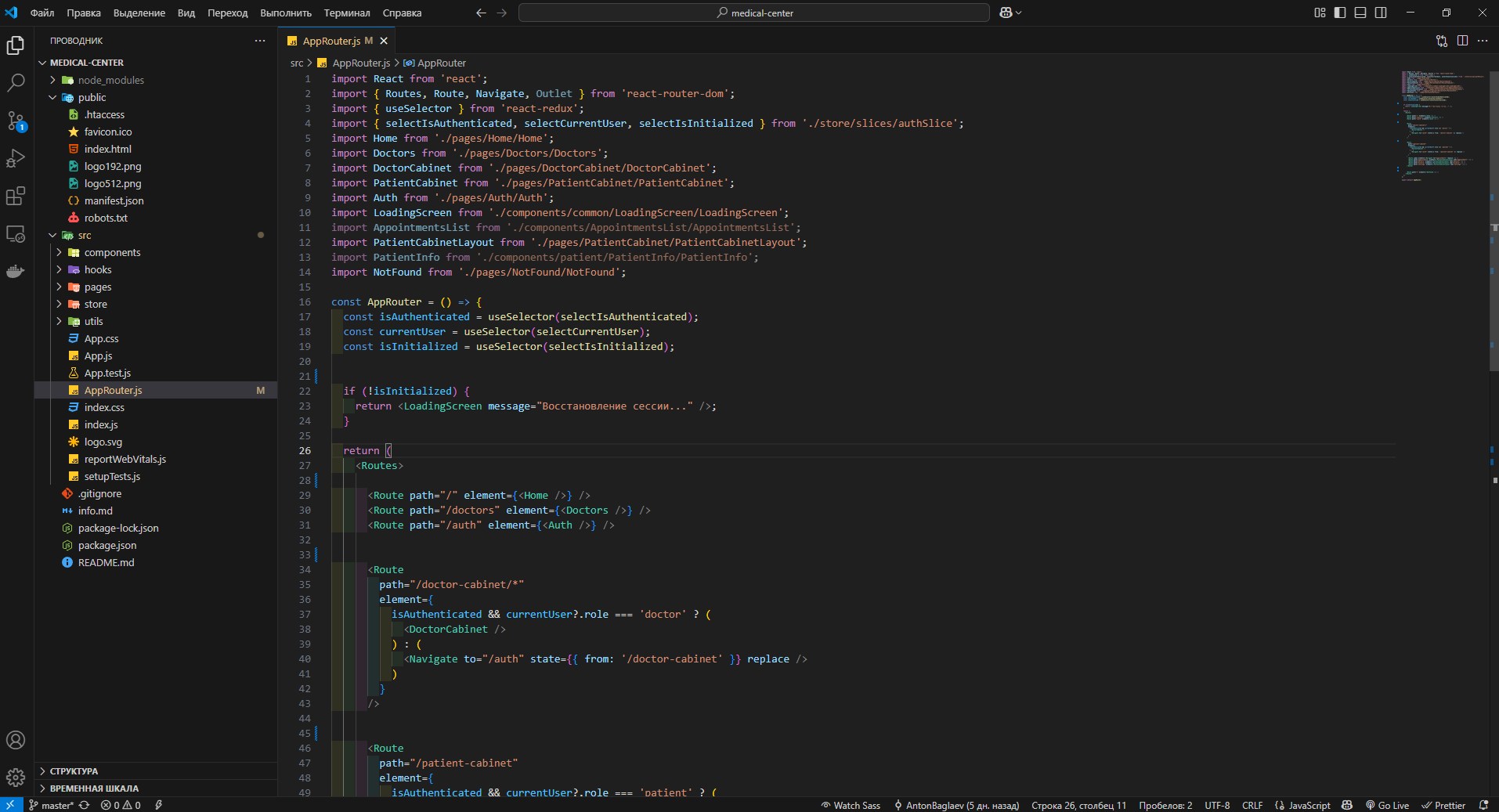
├── README.md                # Описание проекта

└── yarn.lock                # Версии зависимостей

```

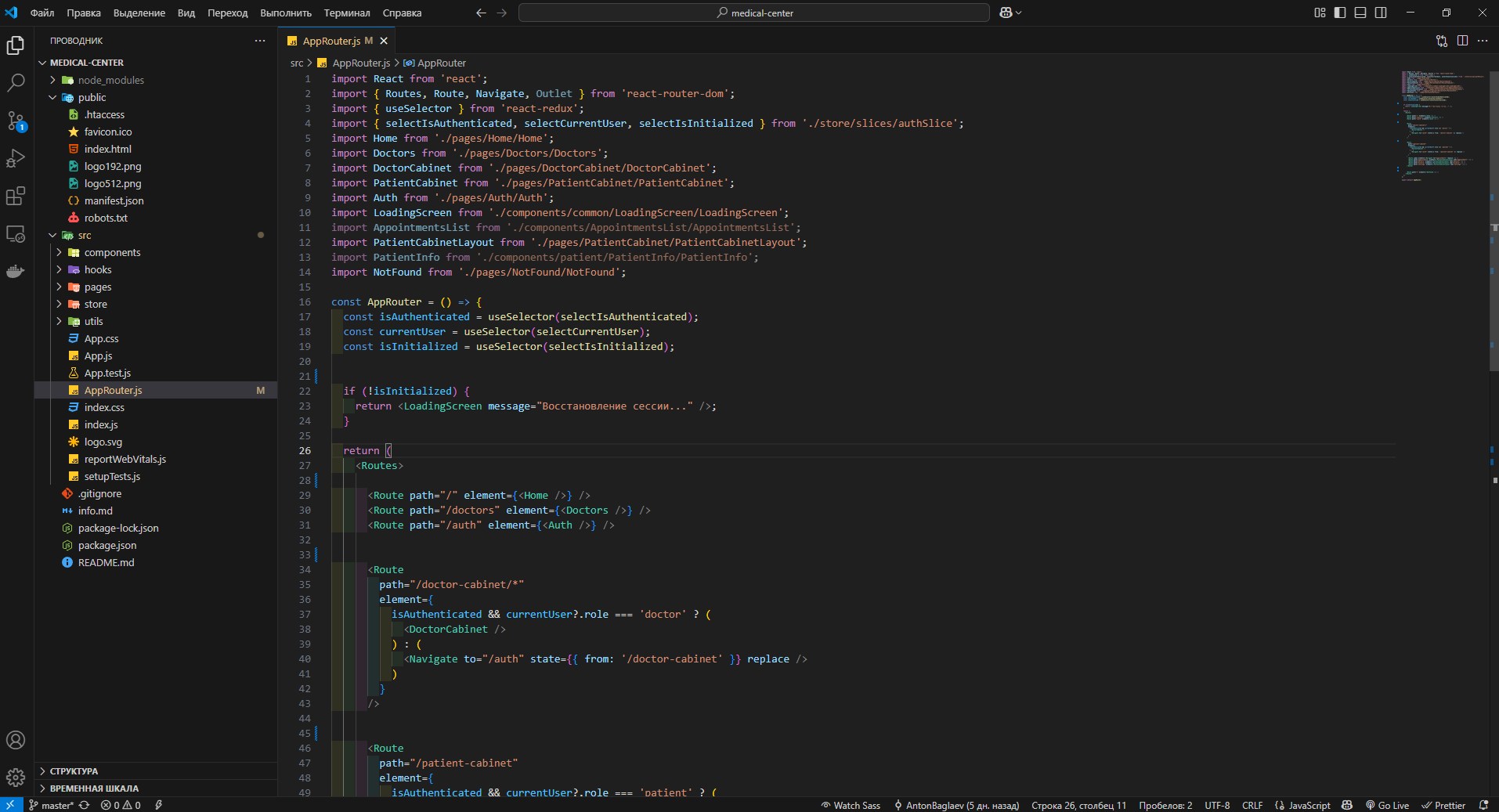
**2.1. Структура и взаимодействие модулей**

2.1.1. Маршрутизация (AppRouter.js)



Главный компонент AppRouter определяет маршруты приложения:

* Публичные маршруты (/auth, /doctors)
* Приватные маршруты (/doctor-cabinet, /patient-cabinet)
* Обработка 404 страницы



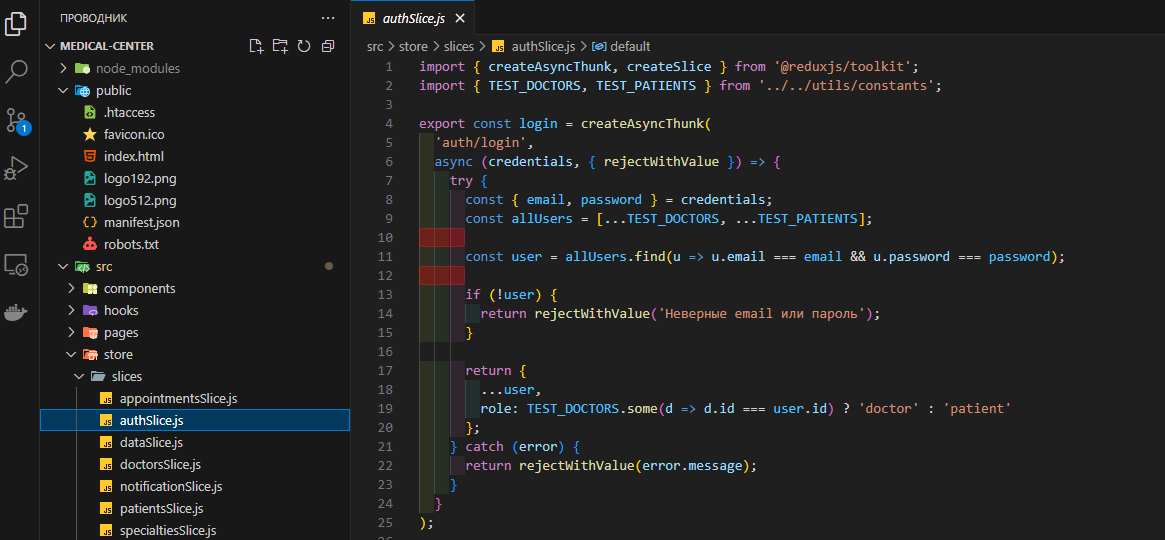
Маршрутизация зависит от состояния аутентификации (проверяется через Redux-селекторы из authSlice).

2.1.2. Управление состоянием (Redux store).

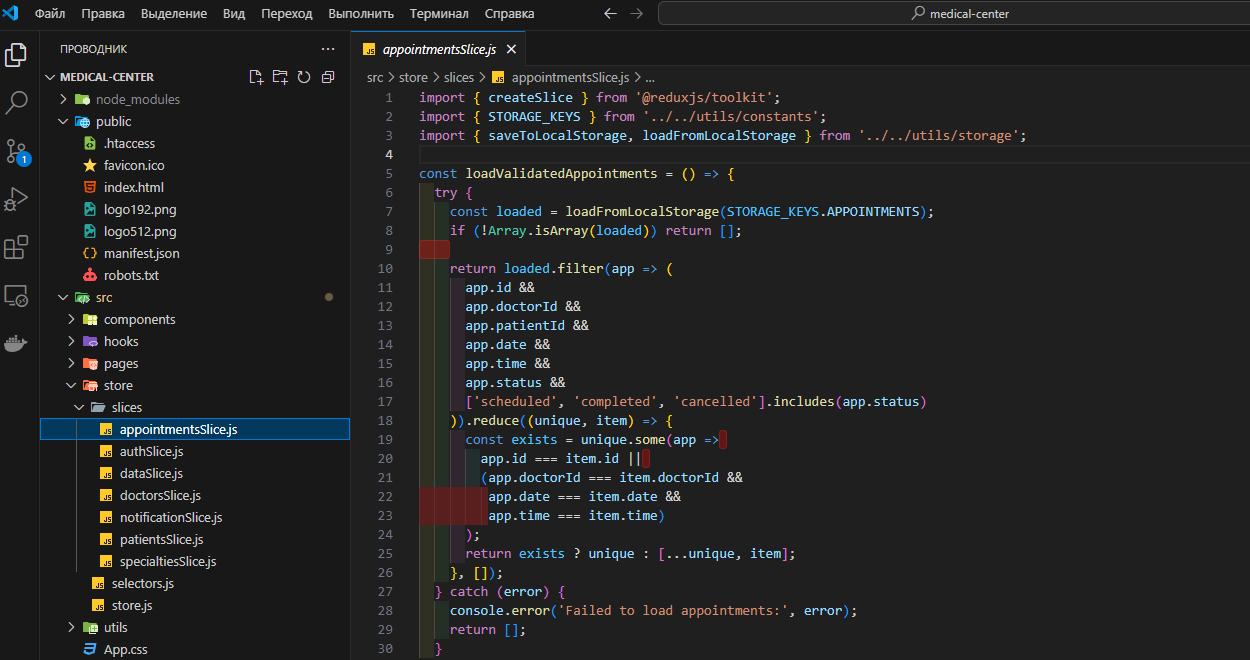


Хранилище состоит из нескольких слайсов:

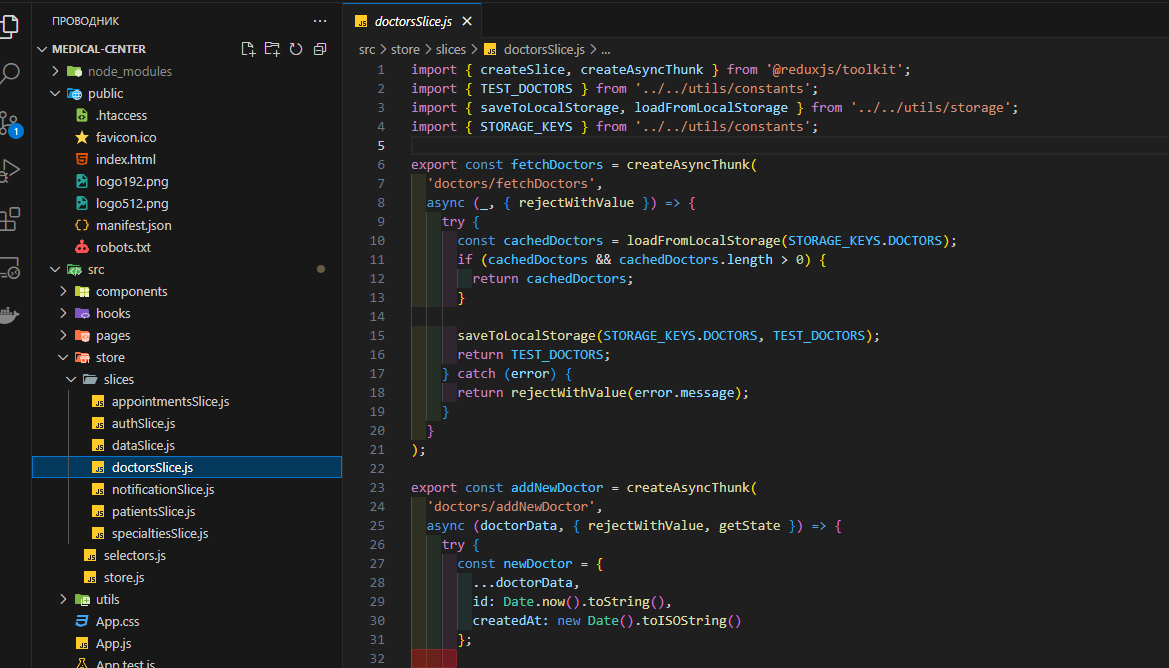
* authSlice - хранит данные аутентификации (токен, роль пользователя)

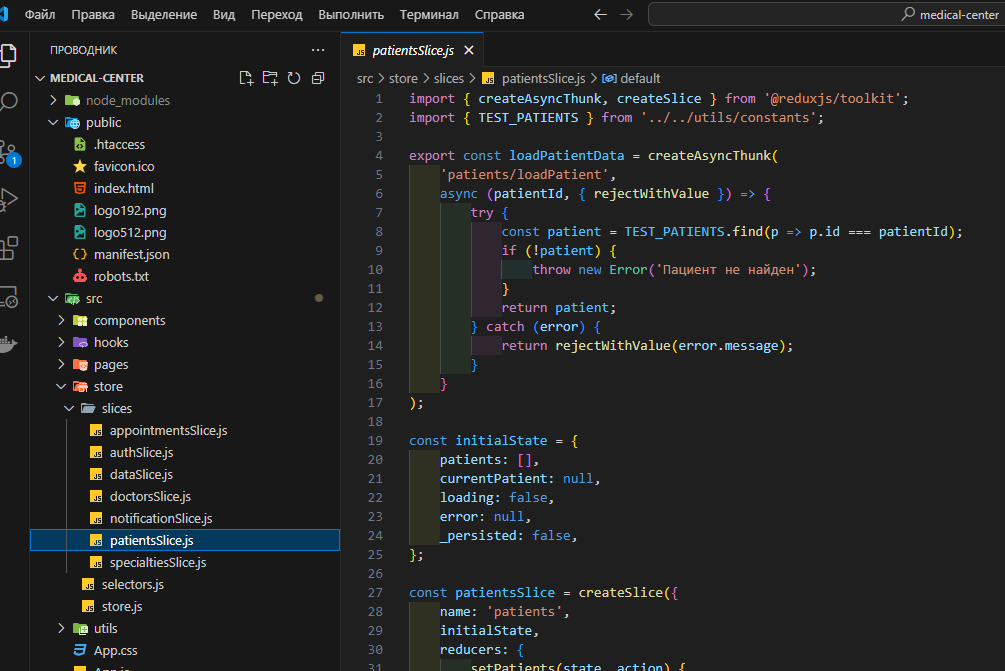


* appointmentsSlice - управляет записями на прием

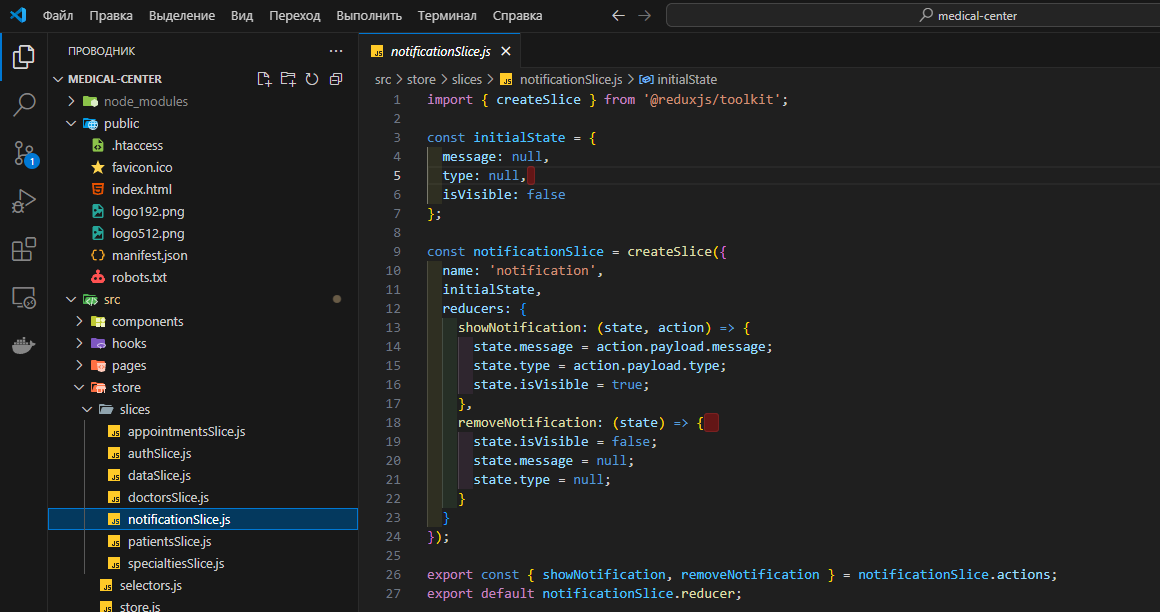


* doctorSlice/patientSlice - содержат профили пользователей

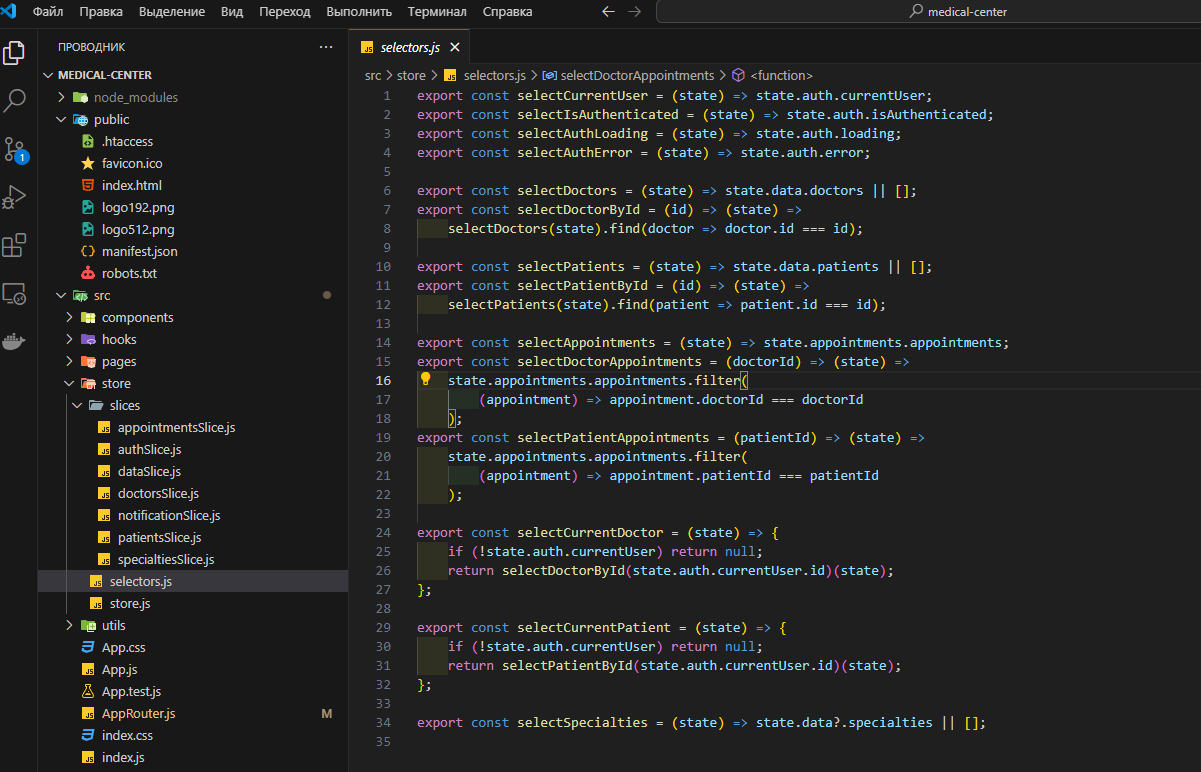




* notificationSlice - отвечает за отображение уведомлений



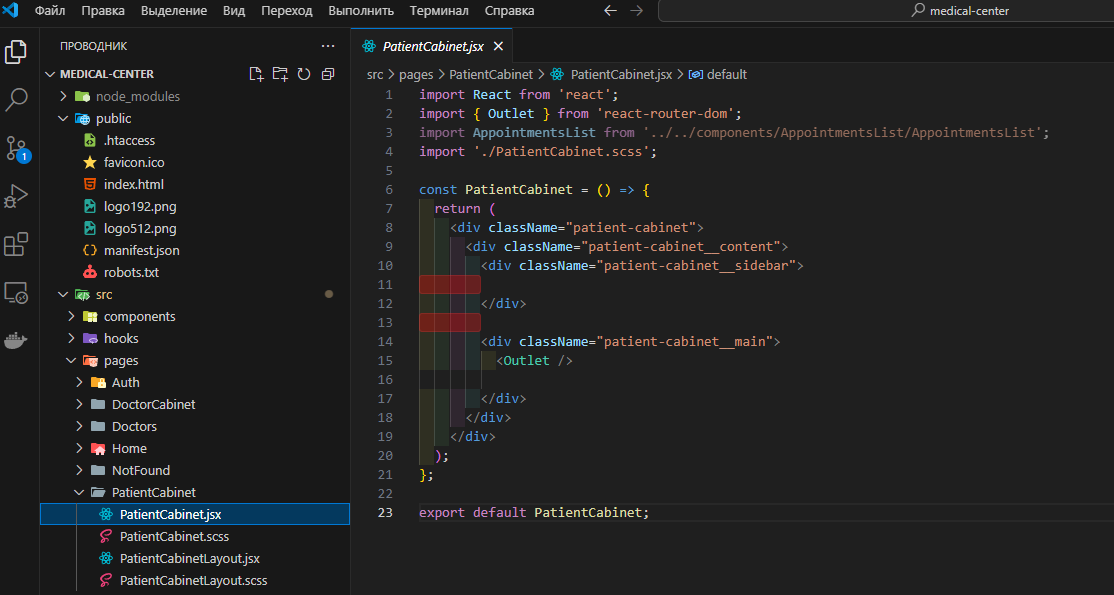
Селекторы (selectors.js) предоставляют доступ к данным из хранилища.

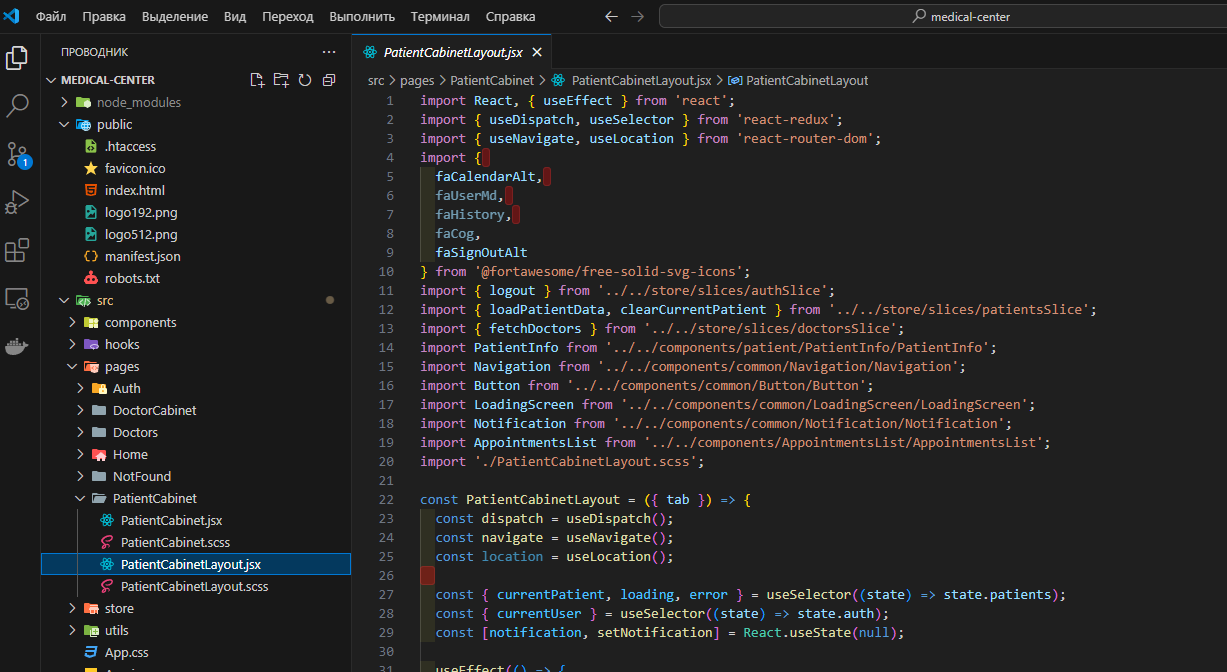


2.1.3. Работа компонентов.

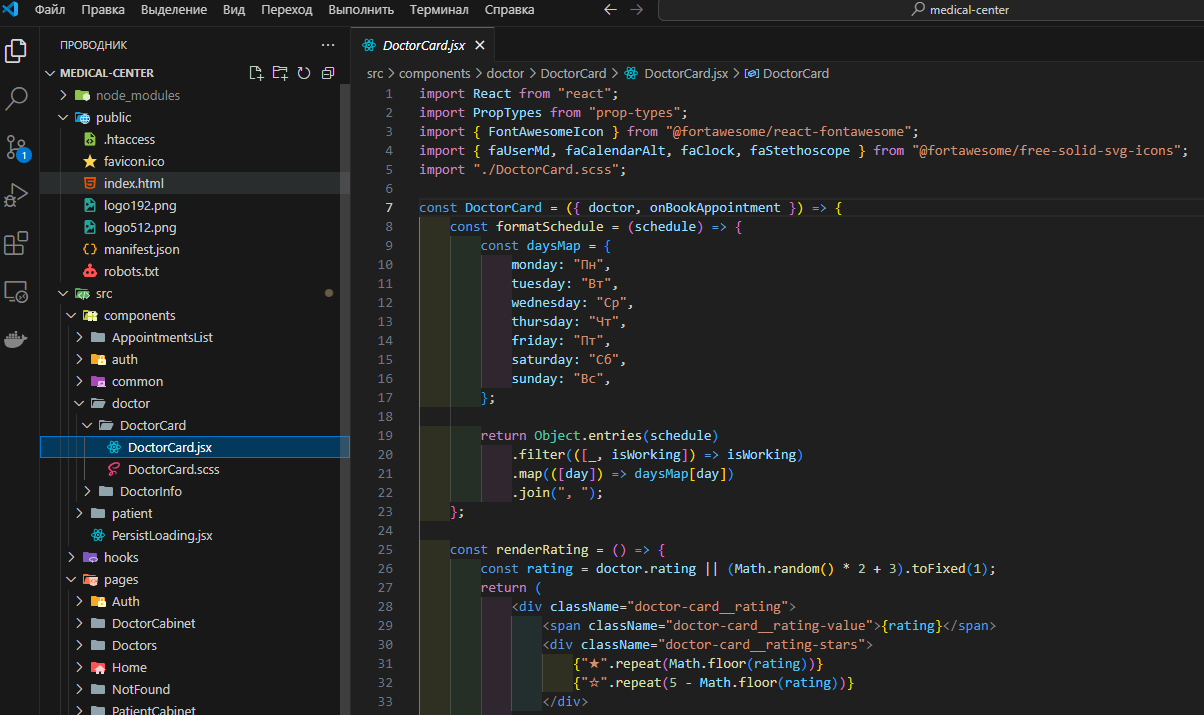
Пример взаимодействия при записи на прием:

* PatientCabinet загружает список врачей через useEffect и doctorSlice

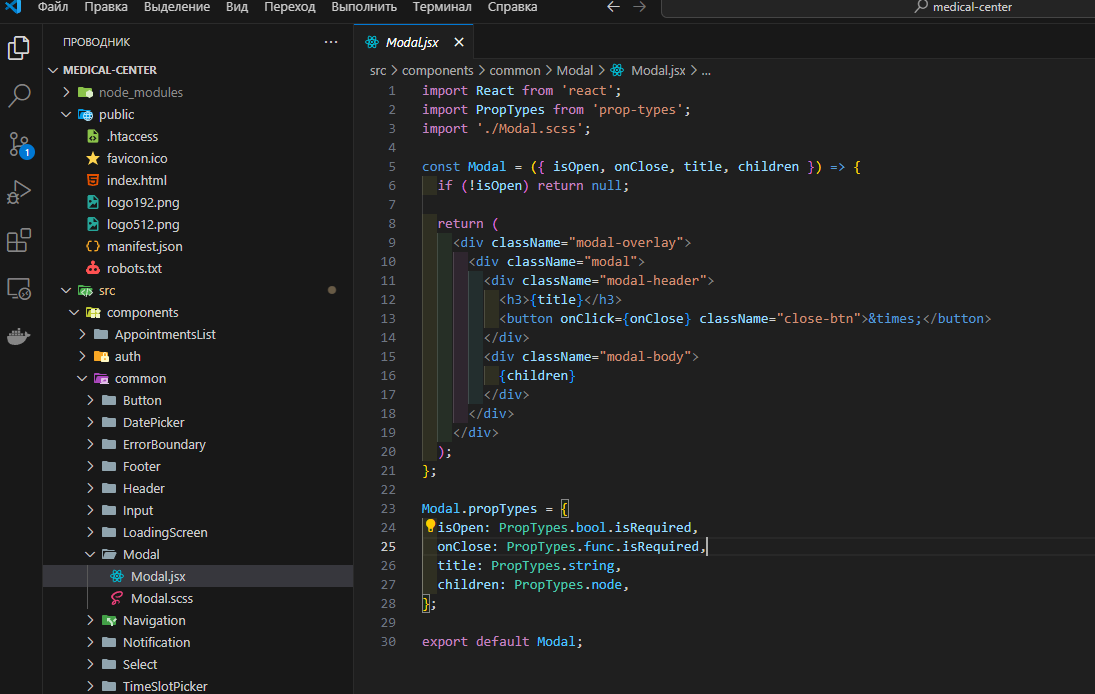




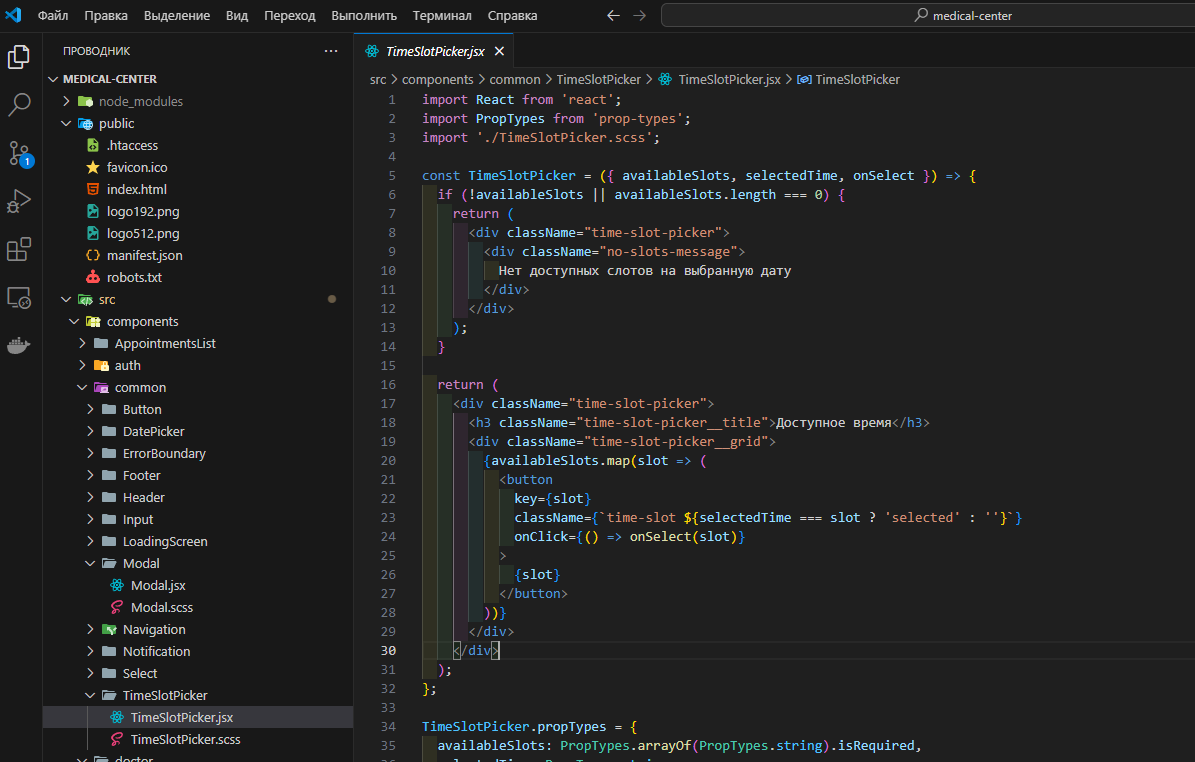
* DoctorCard отображает информацию о враче



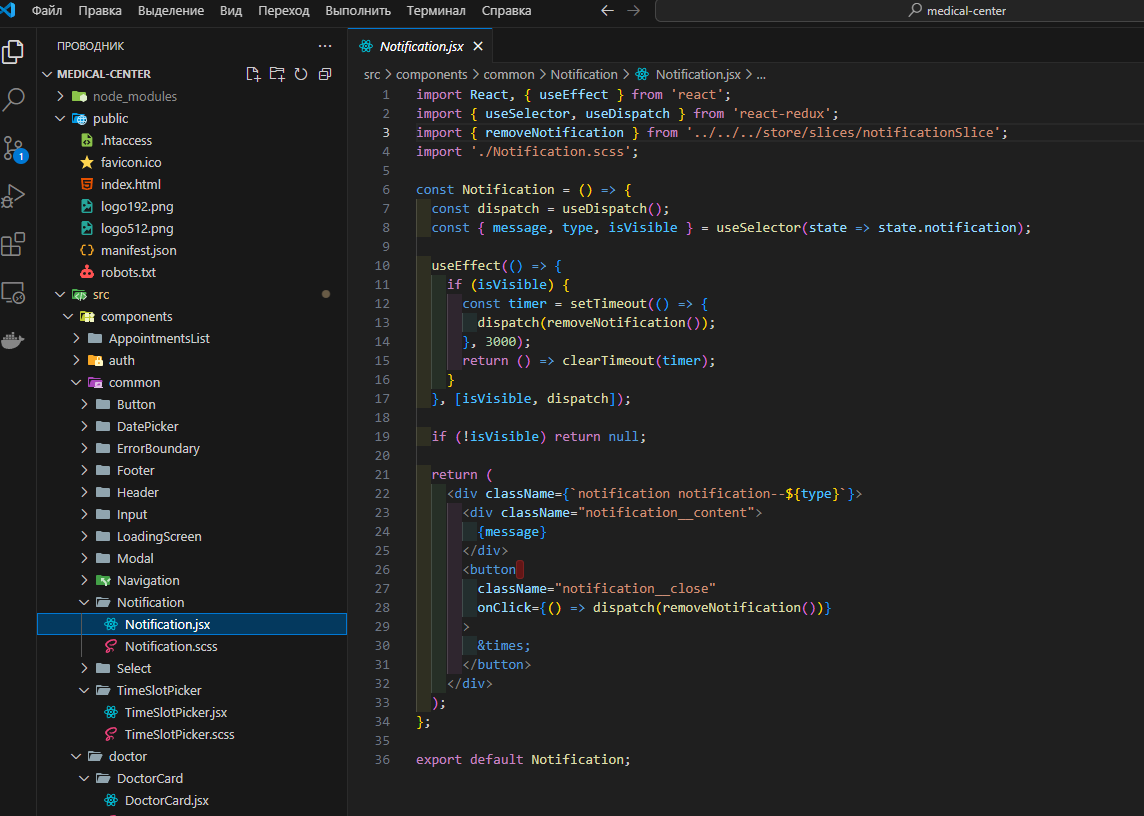
* При клике на "Записаться" открывается Modal с TimeSlotPicker



* TimeSlotPicker получает доступные слоты через appointmentsSlice



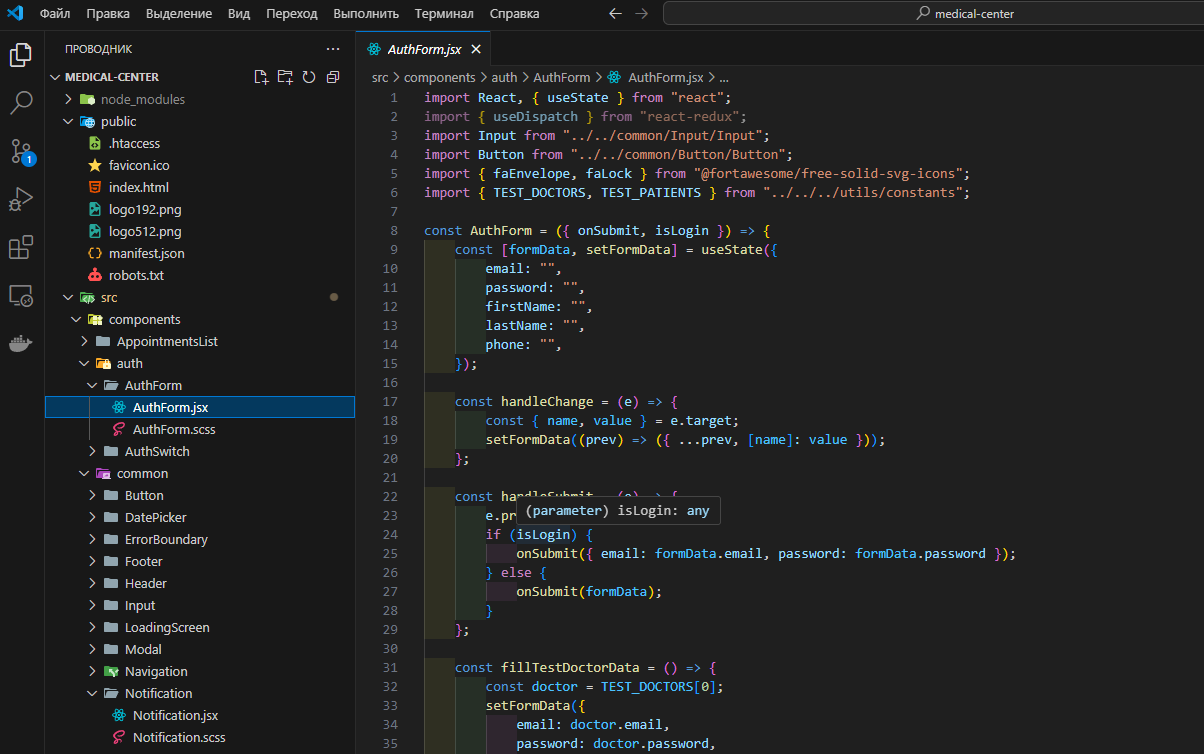
* После выбора даты/времени данные отправляются в appointmentsSlice
* Notification показывает результат операции



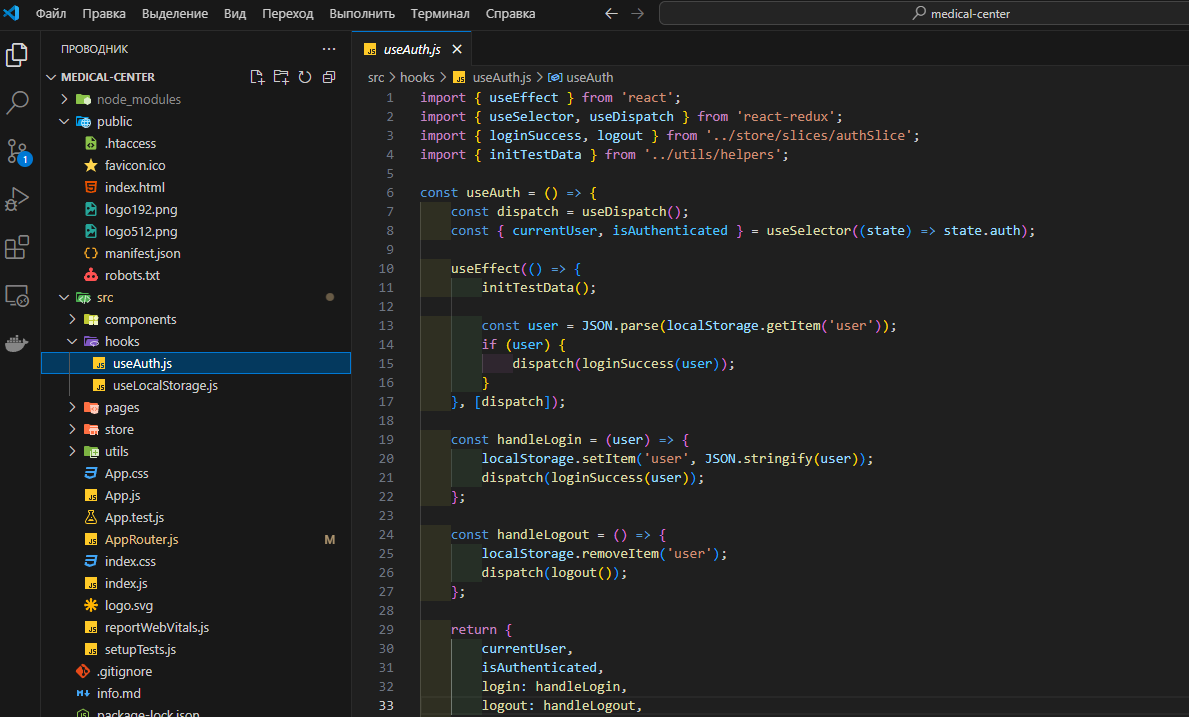
**2. Ключевые функциональные цепочки**

**2.2.1. Аутентификация**

* Пользователь вводит данные в AuthForm



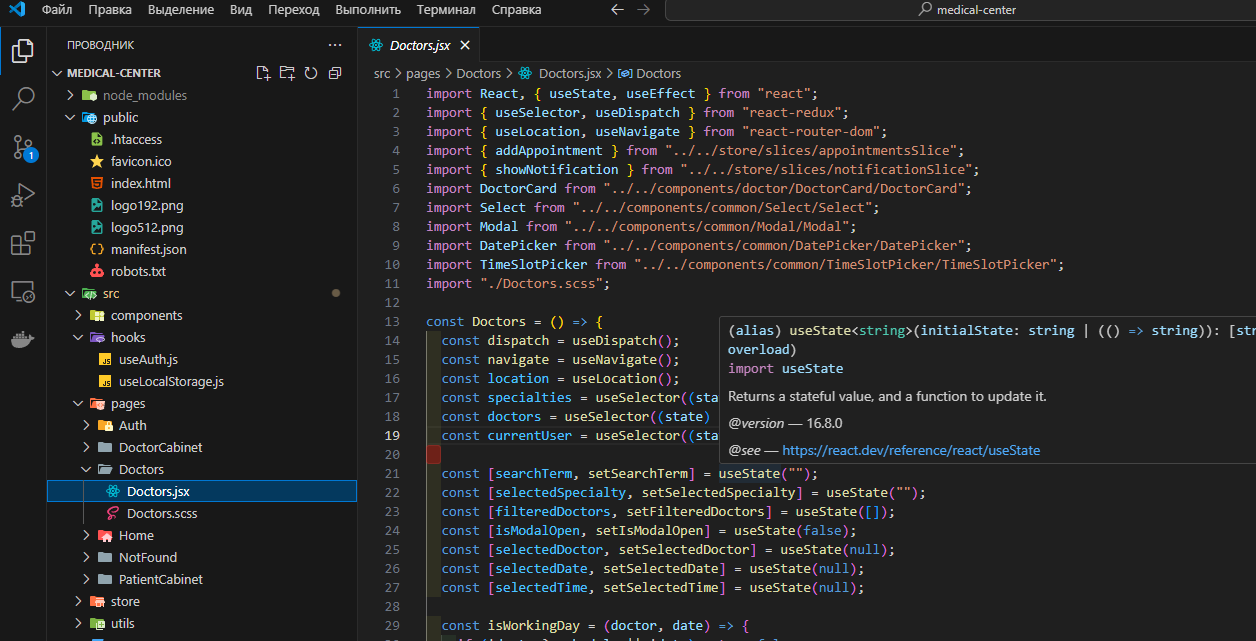
* Данные отправляются через кастомный хук useAuth

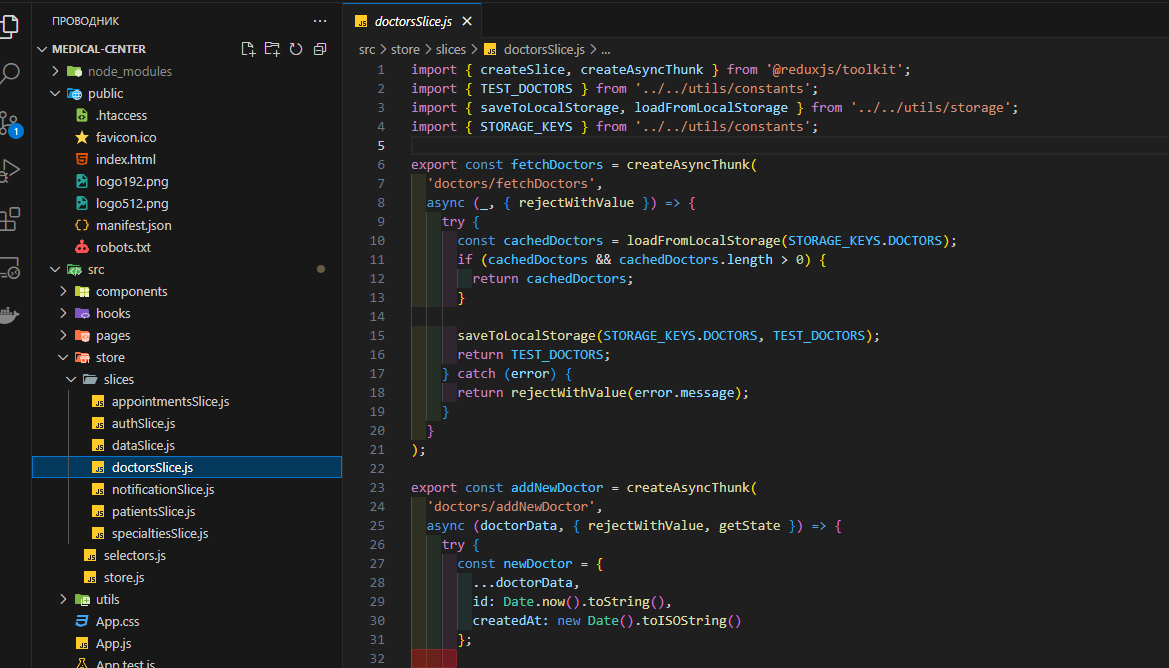


* Хук вызывает API, сохраняет токен в localStorage (через useLocalStorage)
* Результат записывается в authSlice
* AppRouter перенаправляет пользователя в соответствующий кабинет

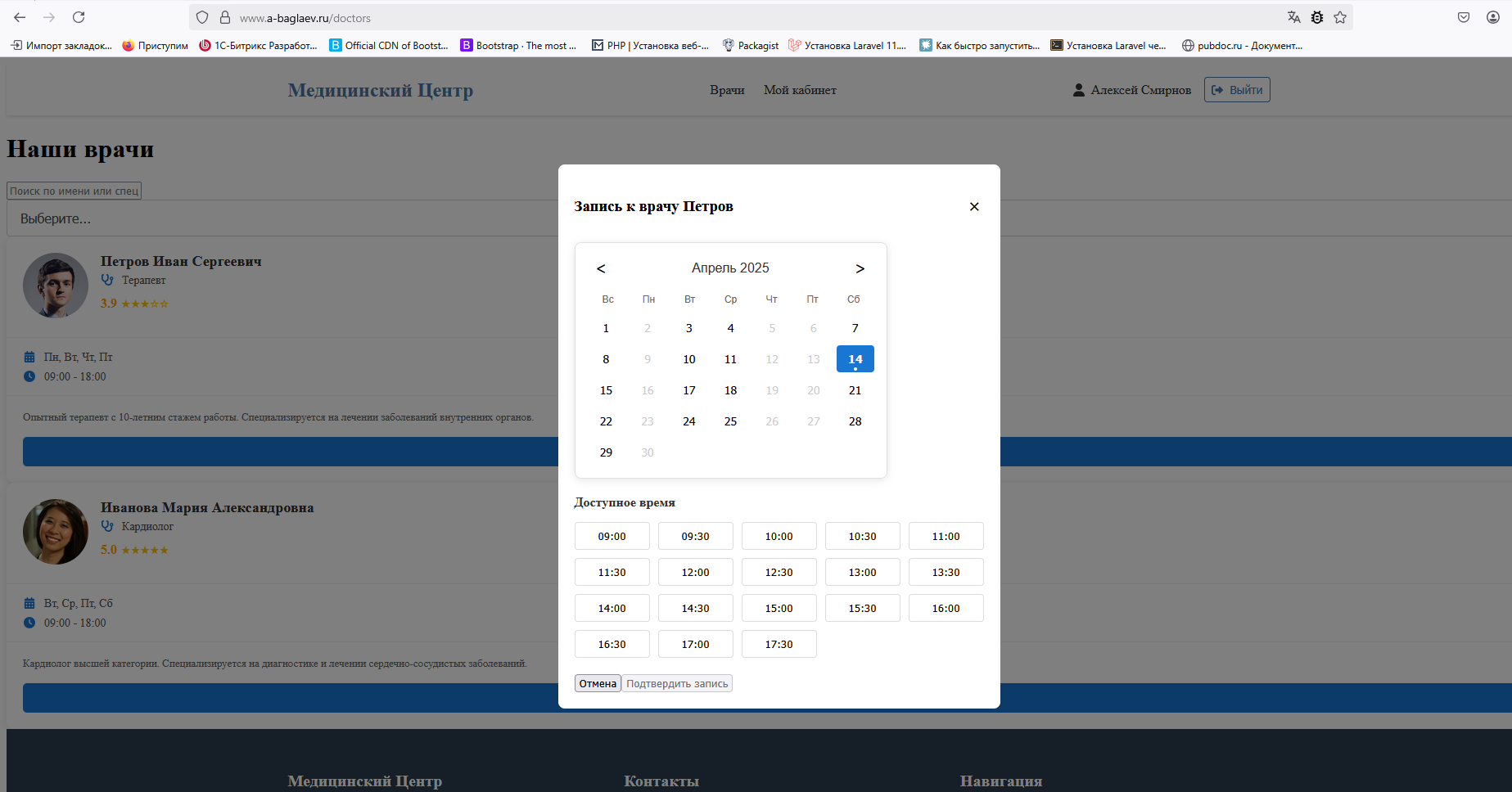
**2.2.2. Запись на прием**

* В Doctors странице загружается список врачей через doctorSlice

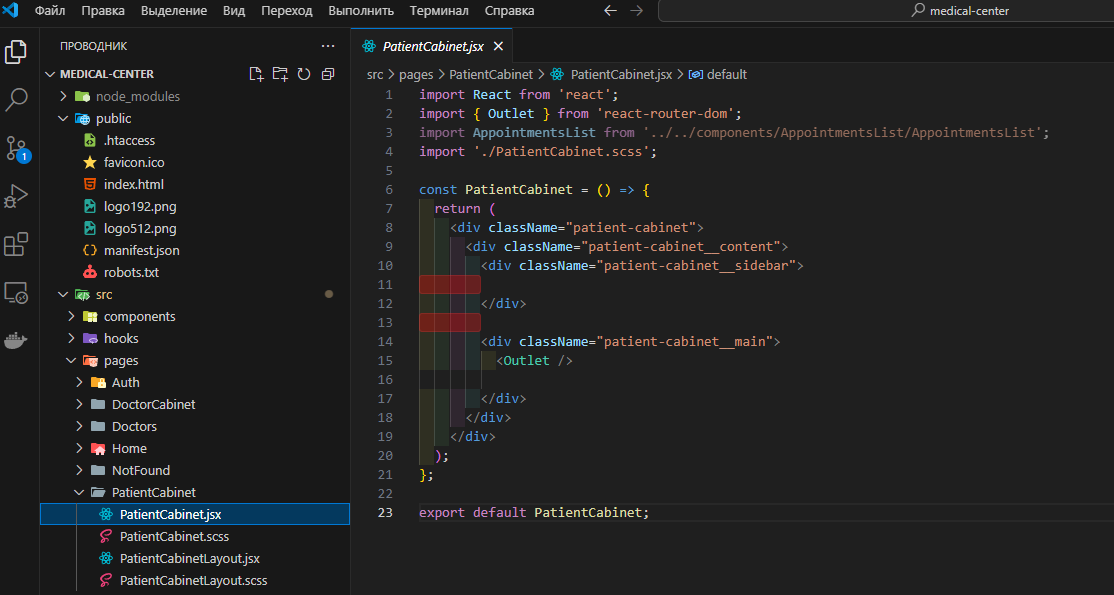




* При выборе врача открывается Modal с DatePicker и TimeSlotPicker

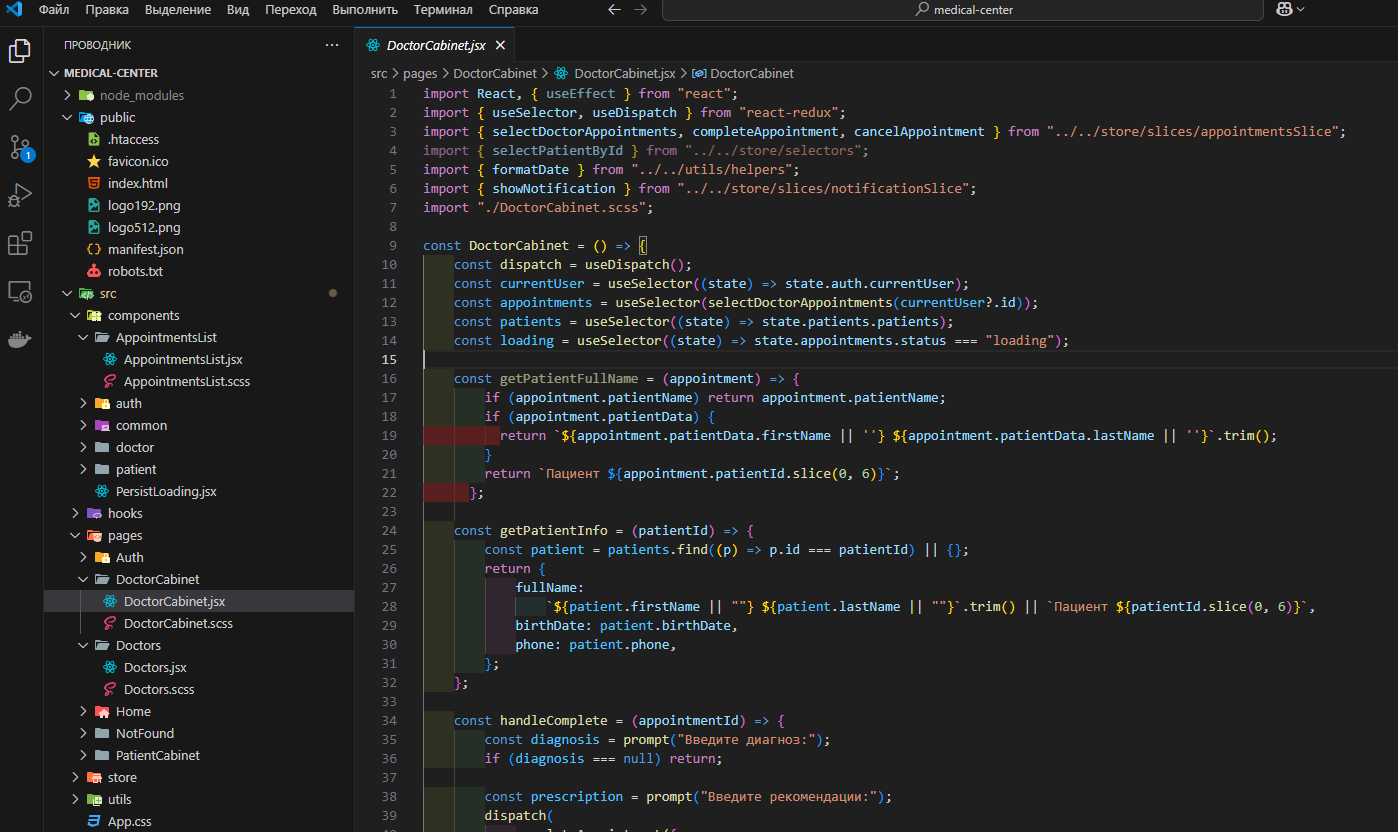


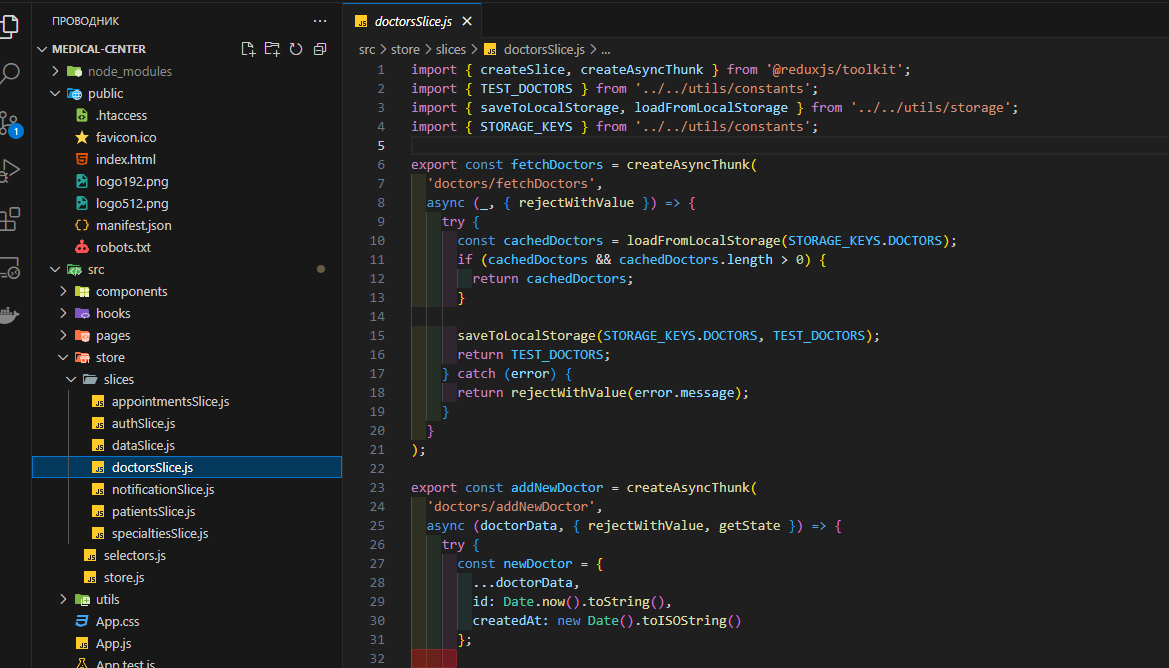
* После выбора слота данные отправляются в appointmentsSlice
* PatientCabinet получает обновленный список записей через селектор



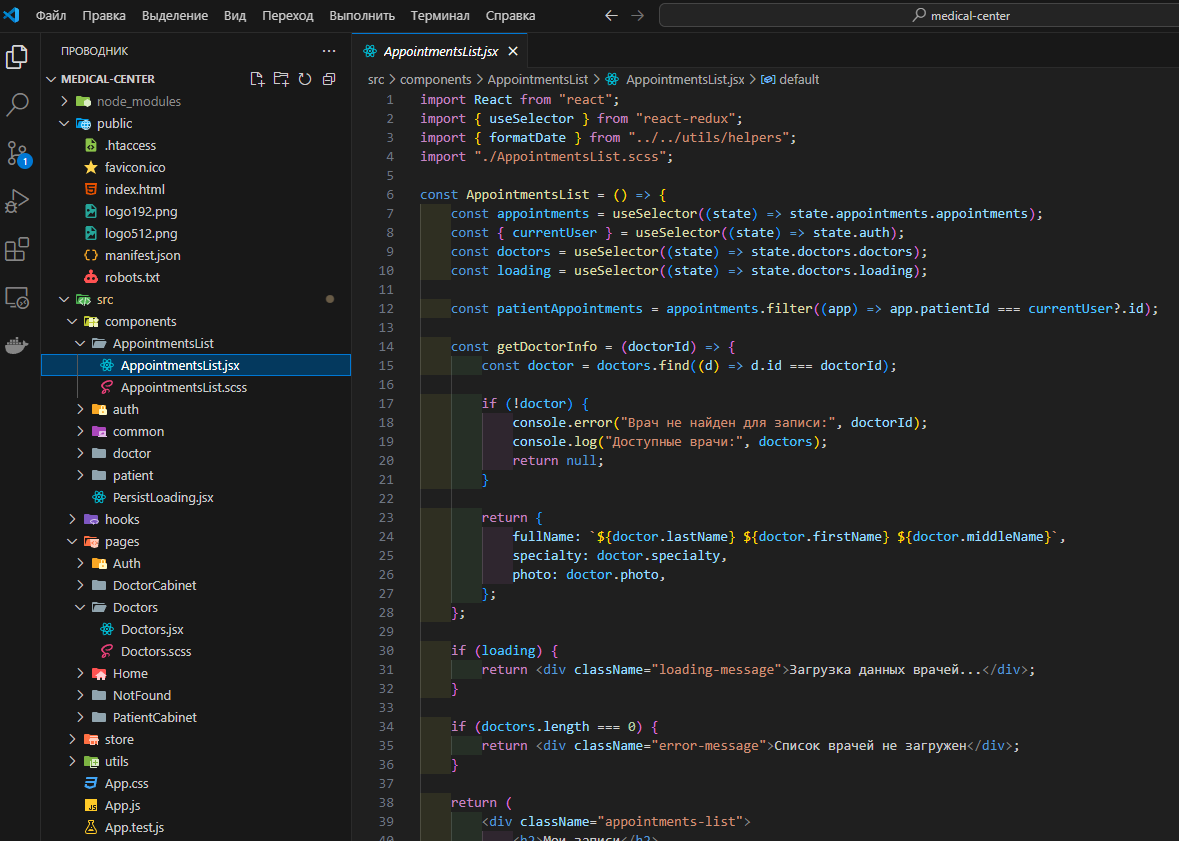
**2.2.3. Личный кабинет врача.**

* DoctorCabinet загружает данные через doctorSlice





* AppointmentsList получает записи пациента через appointmentsSlice



* Врач может изменять статус записи, что обновляет состояние в Redux

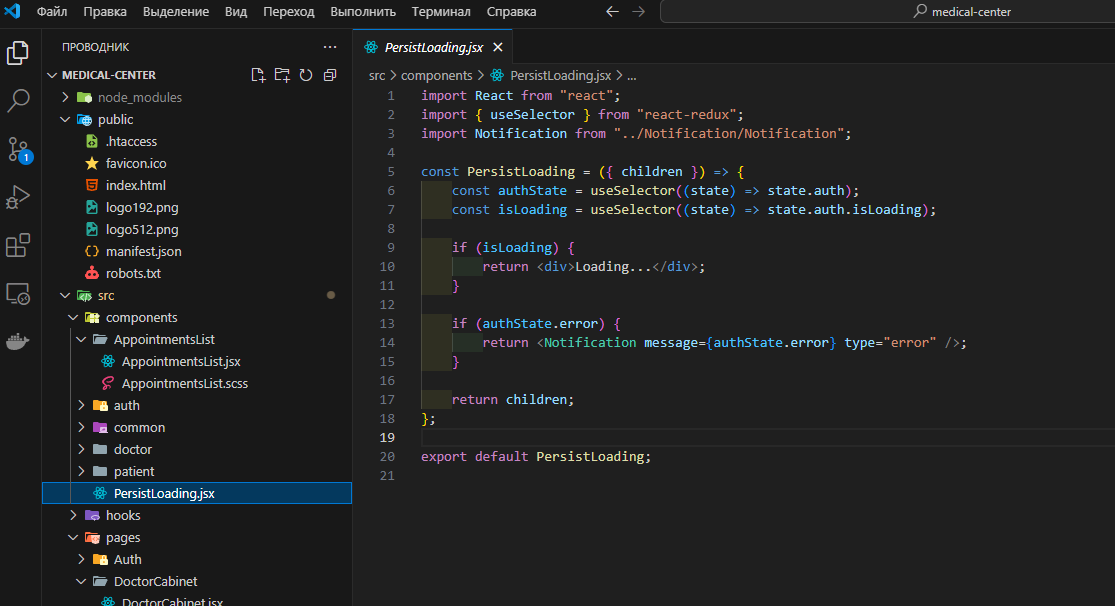
**2.3 Взаимодействие с API**

Проект использует mock-данные, но архитектура предусматривает:

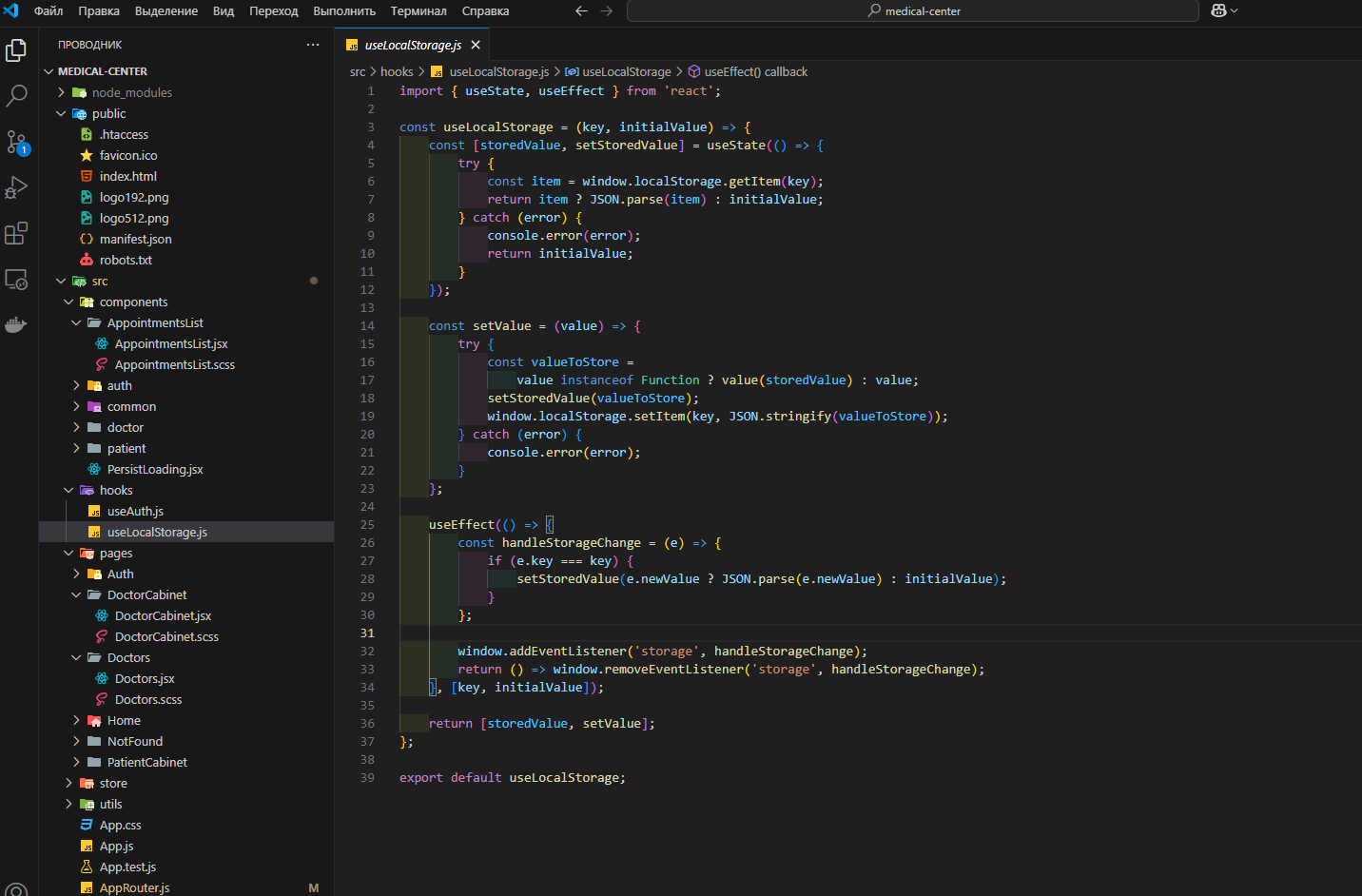
* API-вызовы в асинхронных thunk-функциях (в слайсах)
* Обработку состояний загрузки через LoadingScreen
* Отображение ошибок через Notification

**2.4 Особенности реализации.**

* PersistLoading - компонент для проверки аутентификации при перезагрузке



* useLocalStorage - хук для синхронизации Redux и localStorage



* Модульная структура SCSS позволяет легко изменять стили компонентов
* Все компоненты разделены на презентационные и контейнеры

# Приложения

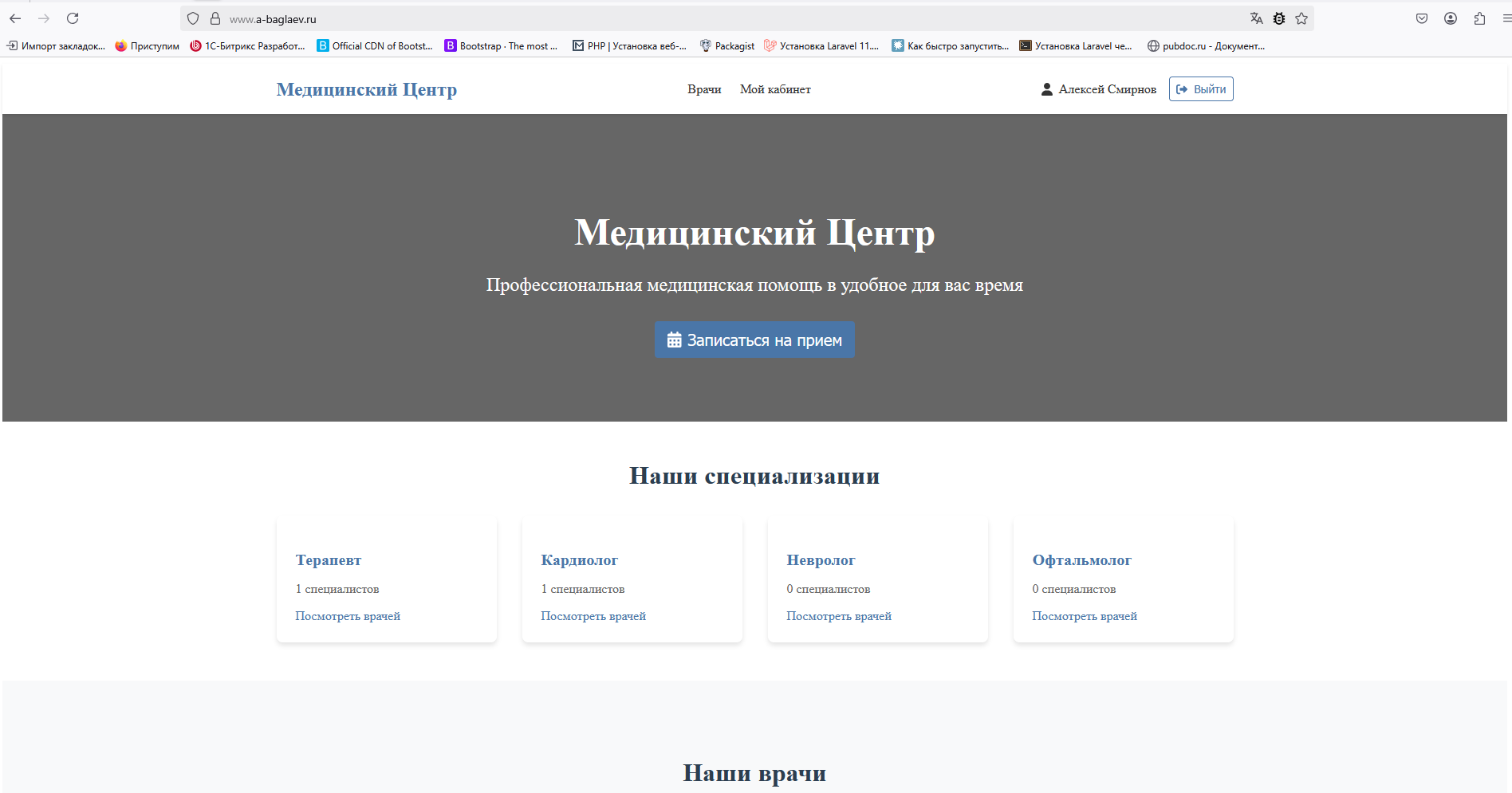
Основное содержание диплома по части практики:

Репозиторий с проектом: <https://github.com/AntonBaglaev/react-med>

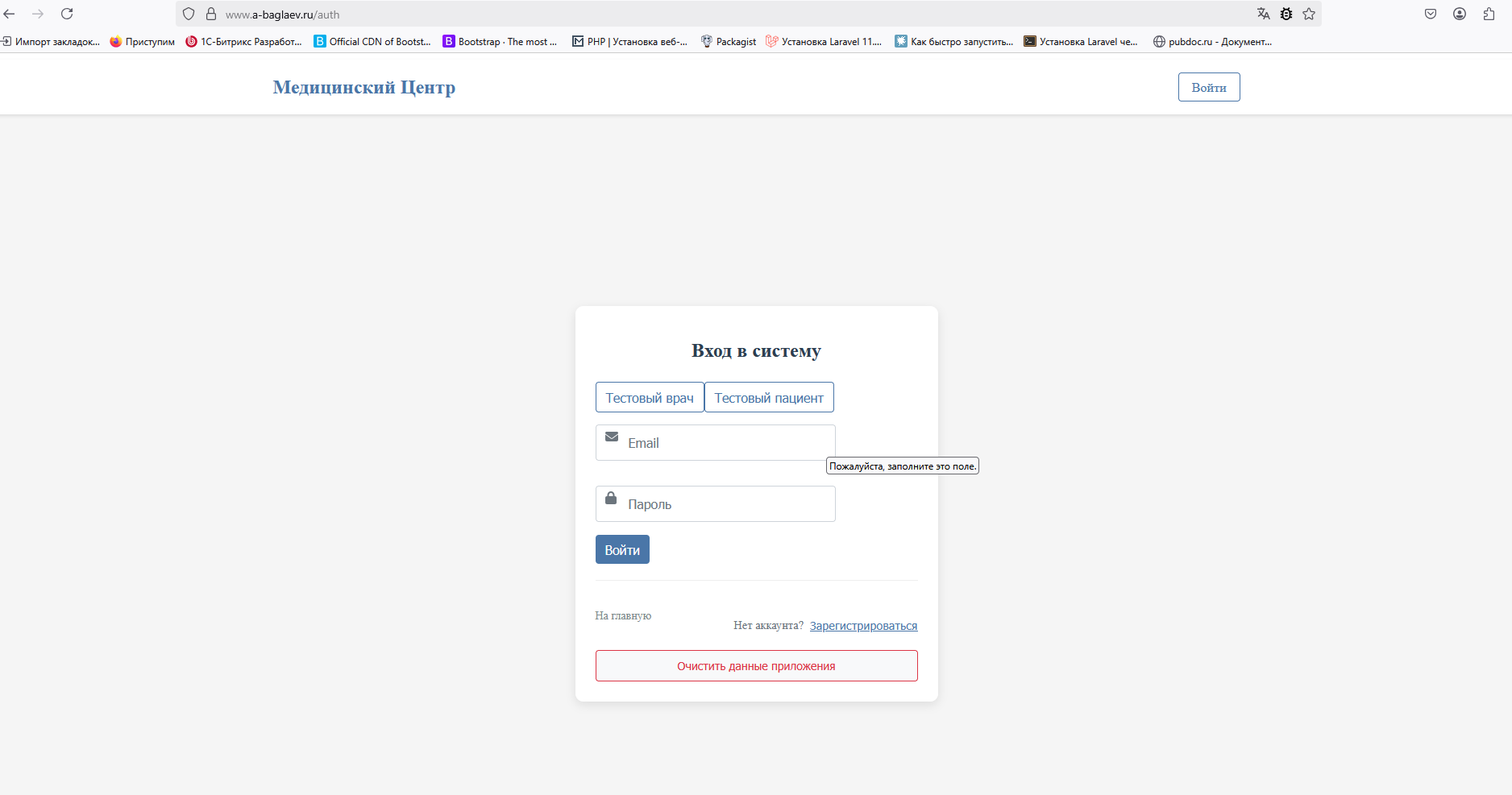
Основной домен: <https://www.a-baglaev.ru>

Скриншоты основных страниц

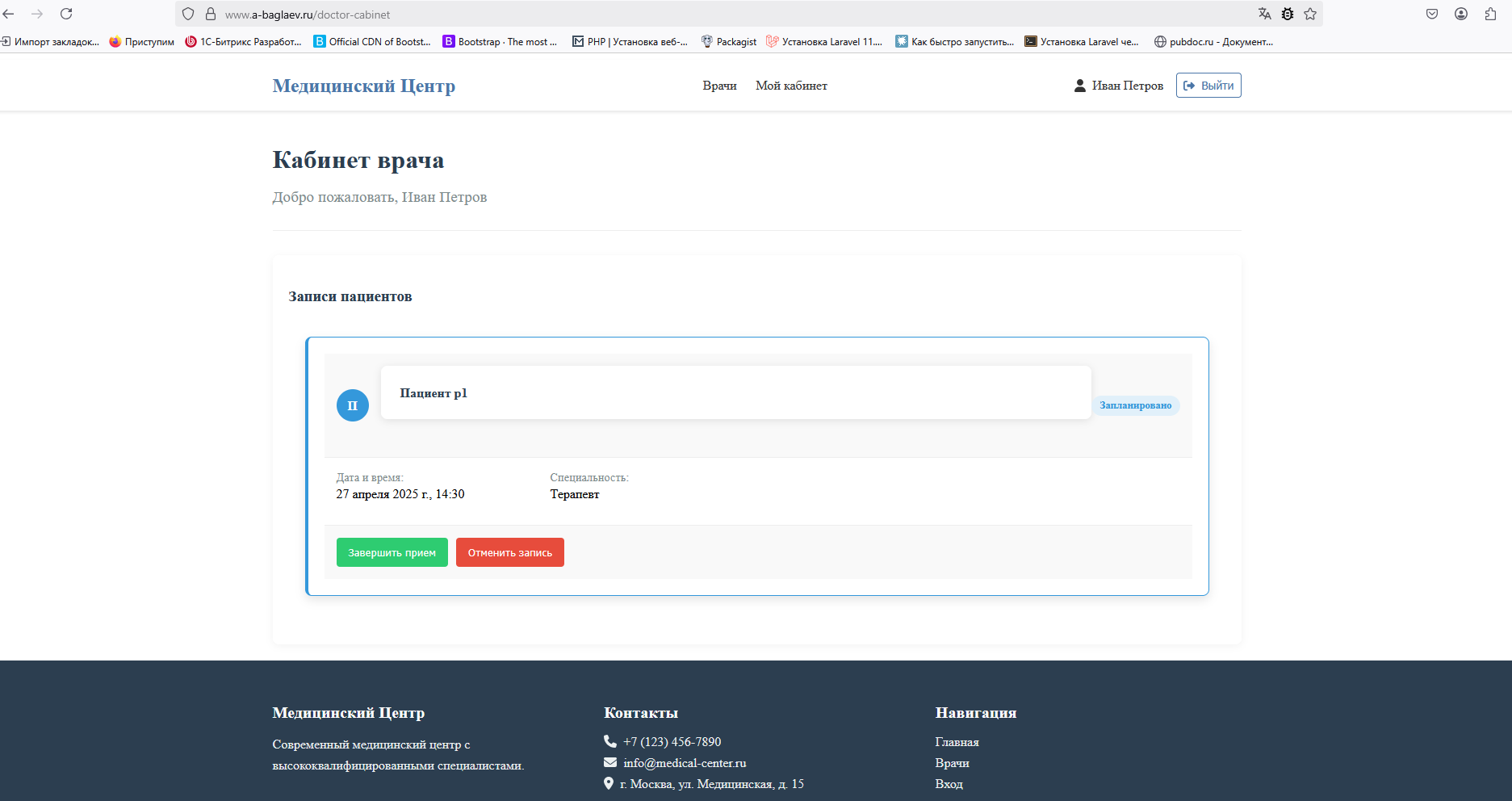
Главная страница



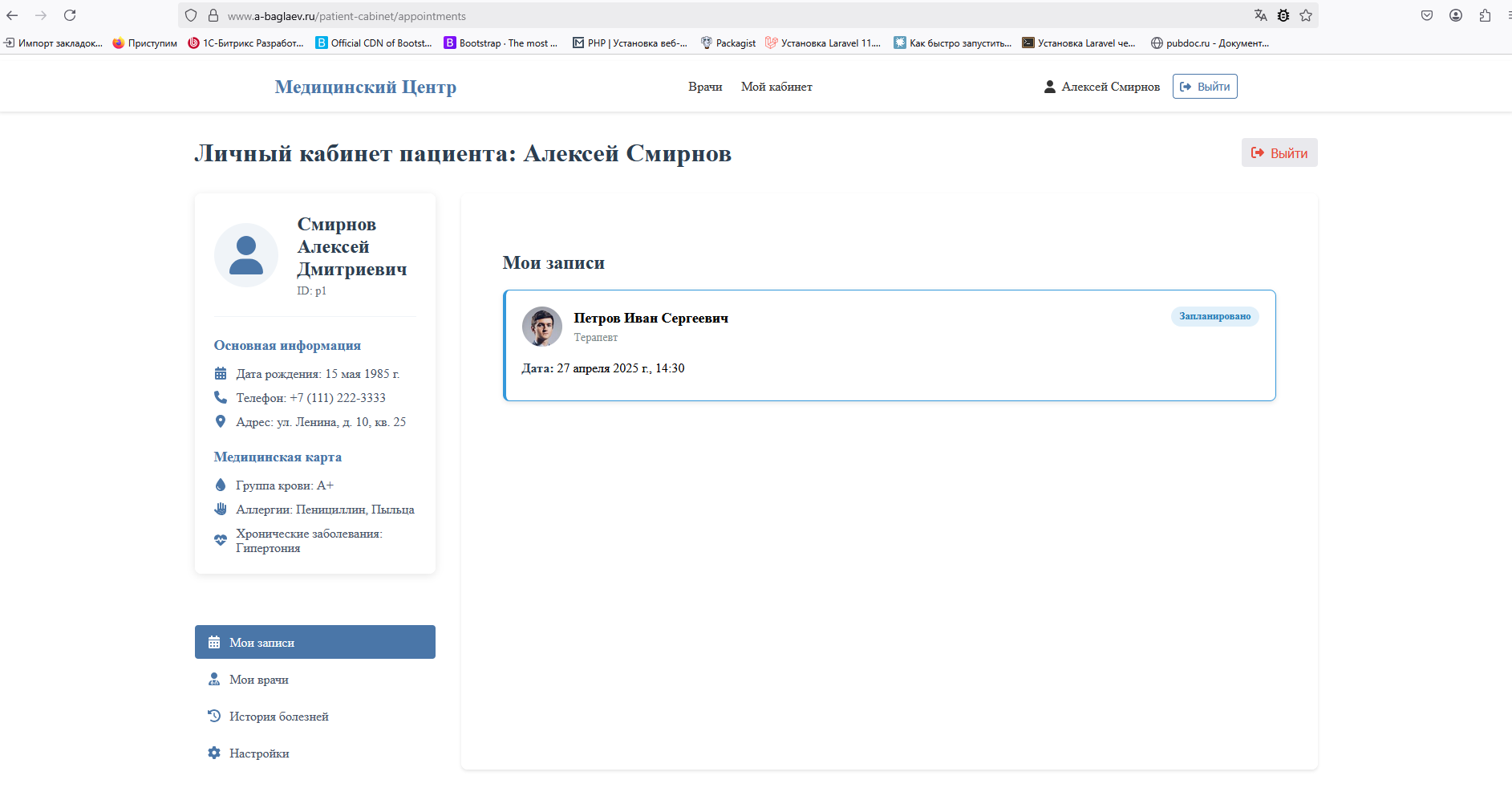
Страница авторизации



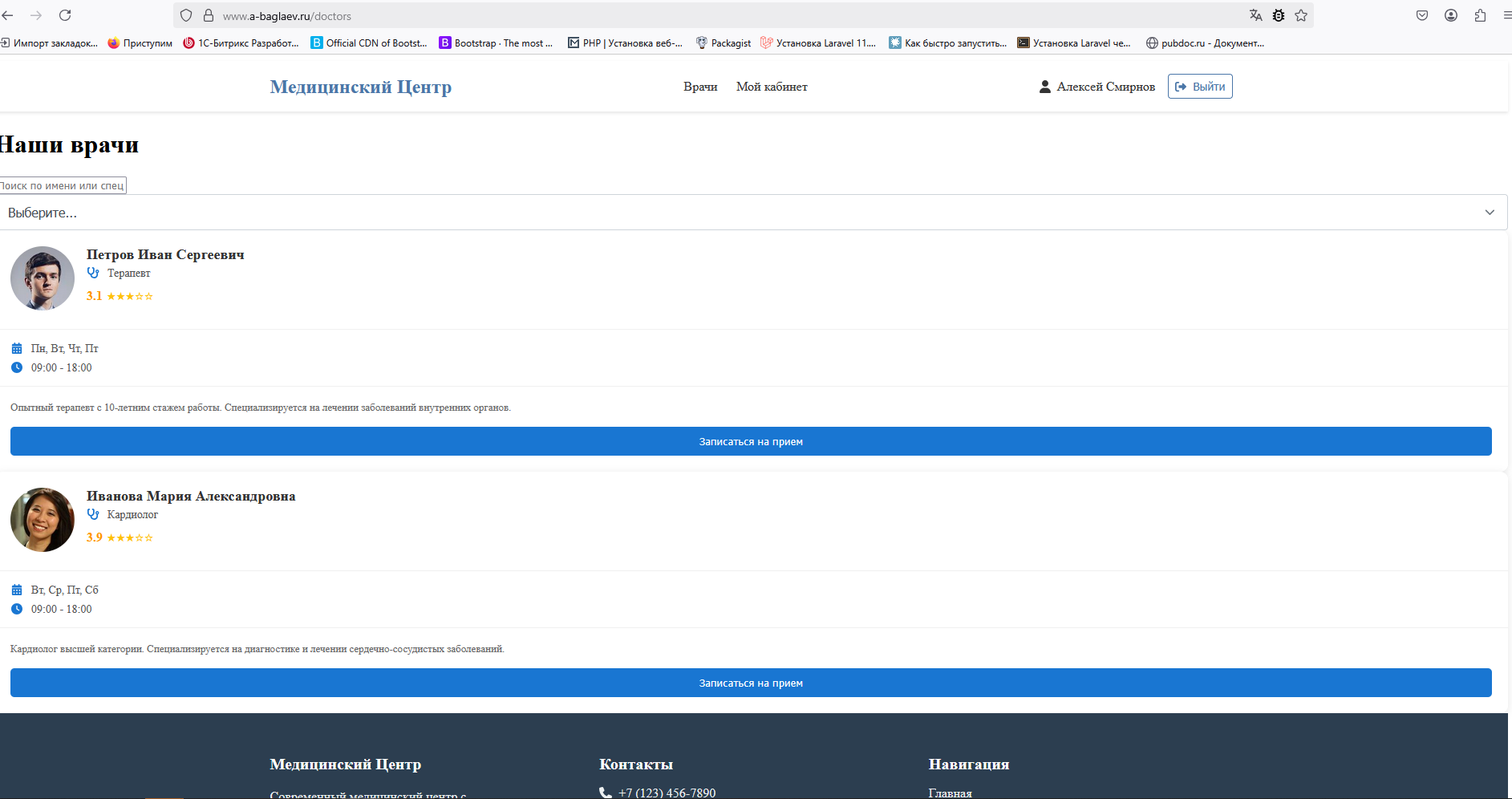
Личный кабинет врача



Кабинет пациента



Страница со списком врачей



Окно записи к врачу

