ВВЕДЕНИЕ

Информационный портал — крупный веб-сайт, организованный как многоуровневое объединение различных ресурсов и сервисов, обновление которых происходит в реальном времени.

Информационный портал содержит большое количество контента, как правило, уникального и рассчитан на большие нагрузки по посещаемости. Сайты такого типа имеют сложную структуру и навигацию и содержат различные интерактивные сервисы. Информационные порталы предназначены для большого количества посетителей.

Целью курсовой работы является применение на практике знаний, полученных в процессе изучения курса «Современные технологии интернет-программирования».

Задачами курсовой работы являются:

* анализ предметной области;
* провести анализ целевой аудитории;
* формирование понимания назначения информационных порталов;
* умения производить обоснованный выбор программного обеспечения для разработки портала;
* разработать техническое задание;
* разрабатывать структуру веб-ресурса
* разработать дизайн-макет веб-ресурса,
* разработать модули веб ресурса;
* разработать приложение;
* провести тестирование приложения.

В курсовой работе студент должен показать степень освоения теоретическим материалом, степень понимания назначения информационных порталов, умение производить обоснованный выбор программного обеспечения для разработки портала.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Поликлиника - это многопрофильное или специализированное лечебно-профилактическое учреждение для оказания амбулаторной медицинской помощи больным на приёме и на дому.

Это учреждение первичной медико-санитарной помощи.

На территории России поликлиники распределены по территориальному признаку и являются базовым уровнем оказания медицинской помощи населению.

Поликлиника является самостоятельной медицинской организацией или структурным подразделением медицинской организации (ее структурного подразделения), оказывающей первичную медико-санитарную помощь, и организуется для оказания первичной доврачебной медико-санитарной помощи, первичной врачебной медико-санитарной помощи, первичной специализированной медико-санитарной помощи, неотложной, а также паллиативной медицинской помощи населению.

Основными задачами и функциями являются:

* оказание первичной (доврачебной, врачебной), специализированной медико- санитарной помощи, в том числе в неотложной форме, больным, проживающим на территории обслуживания и (или) прикрепленным на обслуживание, при острых заболеваниях, травмах, отравлениях и других неотложных состояниях;
* проведение профилактических мероприятий по предупреждению и снижению заболеваемости, выявление ранних и скрытых форм заболеваний, социально значимых заболеваний и факторов риска;
* проведение диспансеризации населения;
* диагностика и лечение различных заболеваний и состояний;
* восстановительное лечение и реабилитация;
* клинико-экспертная деятельность по оценке качества и эффективности лечебных и диагностических мероприятий;
* экспертиза временной нетрудоспособности, выдача и продление листков нетрудоспособности и направление граждан на медико-социальную экспертизу;
* диспансерное наблюдение за состоянием здоровья лиц. страдающих хроническими заболеваниями, в том числе отдельных категорий граждан, имеющих право на получение набора социальных услуг, функциональными расстройствами, иными состояниями, с целью своевременного выявления осложнений, обострений заболеваний, иных патологических состояний, их профилактики и осуществления медицинской реабилитации;
* организация дополнительной бесплатной медицинской помощи, в том числе необходимыми лекарственными средствами, отдельным категориям граждан;
* установление медицинских показаний и направление в медицинские организации для получения специализированных видов медицинской помощи;
* организация и оказание паллиативной помощи больным, в том числе больным онкологическими заболеваниями, нуждающимся в наркотических и сильнодействующих лекарственных средствах в соответствии с рекомендациями врачей-специалистов;
* проведение всех видов медицинских.

В рамках данной под главы был проведен анализ предметной области, определены цели и задачи компании.

* 1. Классификация веб-ресурса

Разрабатываемый веб-ресурс относится к типу динамических одностраничных веб-приложений, обновление которых происходит в реальном времени. Выбранный подход позволяет создать полноценный опыт приложения, в котором переходы между экранами происходят мгновенно, а у пользователя нет необходимости обновлять страницу для получения ответа.

1.2 Анализ целевая аудитория

Целевая аудитория медицинского центра может включать различные группы людей, в зависимости от предоставляемых услуг. К ним могут относиться:

* взрослые мужчины и женщины, нуждающиеся в общей медицинской помощи;
* пациенты с конкретными заболеваниями или хроническими состояниями;
* женщины, ожидающие ребенка или нуждающиеся в акушерско-гинекологической помощи;
* дети, требующие педиатрического обслуживания;
* пациенты, нуждающиеся в стоматологической помощи;
* спортсмены и активные люди, ищущие услуги по физиотерапии или спортивной медицине.

Социально-демографические характеристики целевой аудитории медицинского центра могут варьироваться в зависимости от местоположения и специализации центра. Среди них могут быть следующие:

* возраст: от молодых взрослых до пожилых людей;
* пол: мужчины и женщины;
* образование: от неполного среднего до высшего образования;
* доход: различные социально-экономические группы;
* местоположение: жители города или пригорода;
* семейный статус: одинокие, замужние/женатые, семьи с детьми;
* предпочтения в области здоровья и медицины.

Психологический анализ целевой аудитории медицинского центра позволяет понять, какие мотивации и потребности могут привлечь клиентов. Некоторые общие характеристики психологического анализа могут включать:

* стремление к сохранению и улучшению своего здоровья;
* опасения и тревожность по поводу состояния здоровья;
* желание получить профессиональную и квалифицированную медицинскую помощь;
* потребность в доброжелательном, эмпатическом и внимательном отношении со стороны медицинского персонала;
* приверженность к натуральным и альтернативным методам лечения;
* потребность в комфорте и удобстве при посещении медицинского центра.

Поведенческий анализ целевой аудитории медицинского центра помогает определить, как клиенты ищут и выбирают медицинские услуги, а также какие факторы влияют на их принятие решения о посещении центра. Некоторые характерные аспекты поведенческого анализа могут быть следующими:

* использование интернета для поиска медицинских услуг и отзывов о медицинских центрах;
* предпочтение рекомендаций и рецензий от друзей и семьи;
* заключение медицинской страховки и возможность ее использования в данном центре;
* расположение и доступность практики;
* внешний вид и качество услуги;
* цена и наличие акций или скидок.

В рамках данной под главы были определены представлены результаты маркетингового исследования, целью которого являлось составление «портрета» целевой аудитории (социально-демографические, поведенческие, психологические характеристики), выявление потребительских предпочтений.

1.3 Техническое задание

1.3.1 Введение

1.3.1.1 Назначение

Разрабатываемый продукт предназначен для целевой аудитории предоставляя возможность взаимодействия с врачами. Предоставляя возможность записи пациента к врачу, просмотр истории болезни, просмотр истории записей, и так далее. в любой момент времени.

1.3.1.2 Обзор продукта

Продукт представляет собой 2 веб приложении front-end и back-end, которые будут общаться между собой по протоколу https, front-end будет передать на back-end данные что делает пользователей, после чего back-end будет обрабатывать эти данные и возвращать front-end ответ.

Продукт должен предоставлять врачам и клиентов возможность удобнее взаимодействовать между собой, давай возможность пациентом в любое время записаться на прием или получить интересующую информацию.

1.3.1.3 Функции продукта

Продукт предоставляет возможности:

* просмотреть свою историю болезни;
* создать пользователя на основе полиса;
* просмотреть историю приемов;
* просмотреть список врачей в поликлинике;
* возможность записаться к врачу на конкретную дату и время;
* добавить врачу запись в историю болезни.

1.3.1.4 Термины и определения

Back-end –работает с компонентами на стороне сервера и манипулирует данными, обеспечивая их хранение, обработку и получение.

front-end – это презентационная часть web-приложений, информационной или программной системы, её пользовательский интерфейс и связанные с ним компоненты.

Протокол https - протокол, который обеспечивает целостность и конфиденциальность данных при их передаче между сайтом и устройством пользователя.

Веб приложение - это компьютерная программа, которая для выполнения своих функций использует веб-браузеры и веб-технологии в сети Интернет.

1.3.2 Детальные требования

1.3.2.1 Требование к юзабилити

Существует 3 ключевых критериев, характеризующих уровень юзабилити ресурса:

* простота. Насколько легко новому посетителю ориентироваться в пространстве и выполнять элементарные действия;
* эффективность. Сколько времени требуется на достижение цели;
* запоминаемость. Как быстро пользователь адаптируется к нюансам работы с сайтом, сможет ли оперативно разобраться при повторных посещениях.

1.3.2.2 Требование к производительности

Требования к производительности:

* быстродействие приложения;
* минимальное ожидание до 1 клика.

1.3.2.3 Требование к логической структуре БД

Требование к логической структуре БД:

* база данных должна включать совокупность информационных объектов различного типа;
* каждый информационный объект может иметь набор атрибутов и связей с другими информационными объектами;
* информационные объекты связаны друг с другом иерархически;
* при внесении изменений структура БД должна обеспечивать автоматическое изменение всех объектов и связей, затрагиваемых таким изменением.

1.3.2.4 Ограничения проектирования

Ограничения проектирования:

* работает только в браузере;
* требуется доступа к интернету;
* работает по http протоколу;
* срок 1-2 месяца разработки.

В данной главе были достигнуты следующие задачи произведен анализ предметной области «Поликлиника», произведен анализ целевой аудитории и составлен портрет пользователей, разработано техническое задание на разрабатываемый продукт и определены требования к разрабатываемому продукту.

1. ВЫБОР ИНСТРУМЕНТАЛЬНЫХ И ПРОГРАММНЫХ СРЕДСТВ

РАЗРАБОТКИ

2.1 Выбор языка и технологии программирования

На начальных этапах процесса проектирования программного продукта необходимо принять принципиальные решения, во многом определяющие этот процесс, а также качество и трудоемкость разработки. К таким решениям относят:

– выбор архитектуры программного обеспечения;

– выбор типа пользовательского интерфейса;

– выбор подхода к разработке;

– выбор языка и среды программирования.

Языки веб-разработки - это языки программирования и технологии, используемые для создания веб-сайтов, веб-приложений и компонентов серверной части. Они составляют основу веб-разработки и определяют характер взаимодействия пользователей с веб-контентом. Существует два основных аспекта веб-разработки:

Front-End Development: Включает в себя проектирование и реализацию визуальных элементов, которые видят пользователи и с которыми они взаимодействуют в веб-приложении. Разработчики фронт-энда используют комбинацию HTML, CSS и JavaScript для создания визуально привлекательного и функционального пользовательского интерфейса.

Back-End Development: Работает с компонентами на стороне сервера и манипулирует данными, обеспечивая их хранение, обработку и получение. Это обеспечивает бесперебойную работу веб-приложения и его масштабирование при необходимости. К распространенным языкам back-end относятся Python, PHP, Ruby, Java и C#.

Для Front-End Development в качестве языка будет использоваться JavaScript как один из самый лидирующих языков в Front-End разработке.

Кроме языка программирования требуется выбрать используемые фреймворк.

Фреймворк (англ. framework — «каркас, структура») — это готовый набор инструментов, который помогает разработчику быстро создать продукт: сайт, приложение, интернет-магазин, CMS-систему.

В таблице 1 представлено сравнение Фреймворк для Front-End разработки.

Таблица 1 – Сравнение Фреймворк для Front-End разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Фреймворк | Производительность | Сложность | Комьюнити |
| Angular | - | Сложный | + |
| Vue | + | Простой | - |
| React | + | Средней | + |
| Svelte | + | Простой | - |

В качестве выбора это не будет Angular из-за своей сложности, не Vue из-за не большого комьюнити что может привести к проблеме отсутствию информации о какой-либо проблеме, не Svelte из-за аналогичной проблемы и так же данный фреймворк не прошел проверку временем.

Остается React, проверенный временем и наибольшим комьюнити что дает возможность найди ответ на любой вопрос, который так же прост в понимании.

2.2 Выбор программных средств для создания веб-приложения

Существует множество способов писать код для веб-приложений: от текстовых редакторов до облачных сред разработки. Трудно сразу решить, какая среда лучше подходит для поставленных задач. Чтобы сэкономить вам время, выбрали наиболее популярные это IDE и редактор кода.

IDE (Integrated Development Environment) — это набор программных инструментов, которые используются для создания ПО. Второе название — интегрированная среда разработки.

Редактор кода - это программное обеспечение, которое позволяет программистам и разработчикам создавать, редактировать и отлаживать исходный код программ.

В таблице 2 представлено сравнение средств для создания веб-приложения.

Таблица 2 – Сравнение средств для создания веб-приложения

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Платность | Удобность | Быстрота |
| VS code | - | + | + |
| WebStorm | + | + | - |
| Sublime Text | - | - | + |

Из выбранных вариантов предпочтителен VS code.

Visual Studio Code сочетает в себе простоту редактора исходного кода с мощными инструментами разработчика, такими как доработка и отладка кода IntelliSense.

В основе Visual Studio Code лежит молниеносный редактор исходного кода, идеально подходящий для повседневного использования. Благодаря поддержке сотен языков VS Code позволяет мгновенно повысить производительность при работе с подсветкой синтаксиса, подбором скобок, автоматическим отступом, выделением блоков, фрагментами и многим другим.

2.3 Выбор системы управления базами данных

Система управления базами данных (СУБД) — это программное обеспечение, предназначенное для создания, управления, обновления и анализа баз данных. Она обеспечивает интерфейс для взаимодействия пользователя или приложения с данными, хранящимися в базе данных. СУБД позволяют структурировать данные таким образом, чтобы обеспечить их легкий доступ, безопасность и эффективное использование.

СУБД — комплекс программ, позволяющих создать базу данных и манипулировать данными (вставлять, обновлять, удалять и выбирать). Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД.

В таблице 3 представлено сравнение СУБД.

Таблица 3 – Сравнение СУБД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| СУБД | Бесплатность | Размер базы | Размер таблицы | Число пользователей |
| HSQLDB | нет | 28 ТB | 120 GB | Не ограничено |
| Microsoft SQL Server | нет | 16 ТВ | 532 GB | Не ограничено |
| MySQL | да | 256 TB | 256 ТB | Не ограничено |
| PostgreSQL | да | Неограничен | 32 TB | Не ограничено |

Так как количество данных всегда будет увеличивать и их требуется хранить то лучшей СУБД является PostgreSQL который не имеет ограничений на размере базы данных, так же является бесплатной СУБД, так же имеет возможность быстрого чтения среди множество данных и неограниченный размер хранение индексов.

PostgreSQL заработал прочную репутацию благодаря своей проверенной архитектуре, надежности, целостности данных, надежному набору функций, расширяемости и преданности сообщества открытого исходного кода, стоящего за программным обеспечением, для последовательного предоставления производительных и инновационных решений. PostgreSQL работает во всех основных операционных системах, совместим с ACID с 2001 года и имеет мощные надстройки, такие как популярный расширитель геопространственных баз данных PostGIS. Неудивительно, что PostgreSQL стала предпочитаемой реляционной базой данных с открытым исходным кодом для многих людей и организаций.

2.4 Выбор средств разработки веб сервера

Backend (бэкенд) – серверная часть сайта. Отвечает за быструю загрузку страниц, обработку данных, безопасность, интеграцию с другими системами. Пользователь не видит всего этого, но благодаря backend веб-продукт или ПО нормально функционируют.

Backend-разработчик пишет код, организует хранение и передачу данных. Создает и поддерживает механизмы и алгоритмы, которые позволяют посетителям взаимодействовать со страницей. Например, делает так, чтобы платежи в интернет-магазине были безопасными.

В таблице 4 представлено сравнение фрейморков для разработки веб сервиса.

Таблица 4 – Сравнение фрейморков для разработки веб сервиса

|  |  |  |  |
| --- | --- | --- | --- |
| Фреймворк | Сложность | Документация | Производительность |
| Nest(js) | Средний | + | + |
| Django(python) | Легкий | - | - |
| Spring(java) | Сложный | - | + |

Из представленных Фреймворков выбран Nest(js) из-за своей простоты и хорошо написанной документации в сравнение с др. фреймворками, так же используется тот же язык что и в frond-end разработке что упрощает разработку.

В рамках этой главы были достигнуты следующие задачи выбран язык и фреймворк разработки front-end приложения, выбран язык и фреймворк разработки back-end приложения, выбрана система управления базой данных и выбраны программные средства для разработки веб приложения.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ВЕБ ПРИЛОЖЕНИЯ

3.1 Структура веб приложения

Для разработки веб приложение требуется определить компоненты, из которых будет состоять это приложение.

Компонент меню - предназначен для основной навигации по сайту.

Компонент главная страница - самая первая страница, которая появляется при заходе на сайт. Главная страница есть у всех сайтов, от солидных интернет-магазинов до популярных форумов. Основная функция главной страницы – это привлечение внимания целевой аудитории и конвертации пользователей в реальных клиентов.

Компонент врачи – представляет список врачей.

Компонент конкретный врач – представляет подробную информацию о конкретном враче.

Компонент форма запись к врачу – представляет возможно записаться к врачу.

Для начала пользователь должен ввести полис и нажать кнопку проверить полис, если полис не найден в базе данных, то пользователю нужно будет ввести свои данные ФИО и номер телефона, после чего нажать на кнопку «сохранить», после чего пользователь должен выбрать врача, к которому он хочет записаться и дату записи. Далее из ниже представленного списка Компонента «время записи к врачу» выбрать свободное время, когда к врачу можно будет записаться.

Компонент время записи к врачу – представляет список времени, когда возможно записаться к врачу, на элемент списка требуется нажать что бы записаться на прием, если текущие время занято, то нечего не произойдет.

Компонент создание пользователя – форма для создания пользователя.

Компонент история болезни – предоставляет возможность пользователю просмотреть свою историю болезни, для этого требуется ввести свой полис и нажать на кнопку.

Компонент форма заполнение истории болезни – предоставляет возможность врачу заполнять историю болезни пациента.

Компонент история приемов - предоставляет возможность пользователю просматривать свою историю приемов для этого требуется ввести свой полис и нажать на кнопку «поиск».

В рамках данной под главы были определены компоненты, требуемые для разработки программного продукта.

3.2 Разработка дизайн-макета веб приложения

Дизайн-макет – это эскиз или образец того, как будут выглядеть ваши печатные материалы (например, буклеты, визитки, баннеры) до того, как они отправятся на печать. Это важная стадия в процессе разработки дизайна, так как позволяет вам увидеть, как будут располагаться элементы (текст, изображения, цвета), а также предотвратить возможные ошибки и определить нужные изменения для получения лучшего результата.

На рисунке 1 представлен дизайн макета главной страницы.

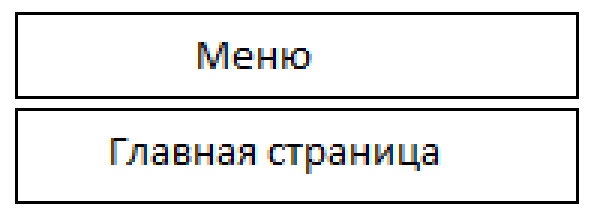


Рисунок 1 – Дизайн макета главной страницы

На рисунке 2 представлен дизайн макета страницы врачей.

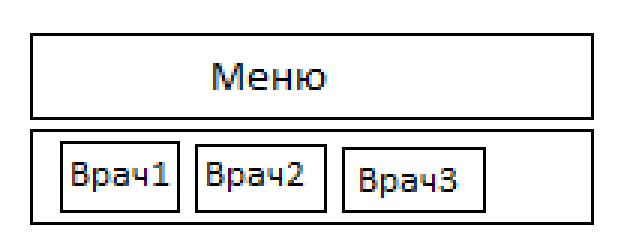


Рисунок 2 – Дизайн макета страницы врачей

На рисунке 3 представлен дизайн макета конкретного врача.

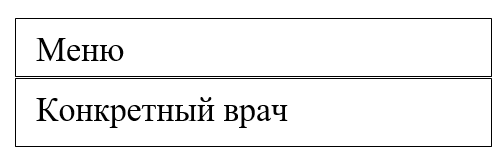


Рисунок 3 – Дизайн макета страница конкретного врача

На рисунке 4 представлен дизайн макета страницы истории болезни.

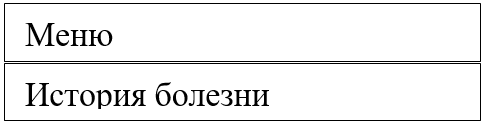


Рисунок 4 – Дизайн макета страница история болезни

На рисунке 5 представлен дизайн макета страницы заполнения истории болезни.

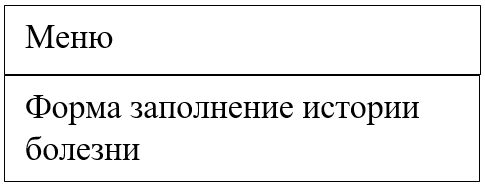


Рисунок 5 – Дизайн макета страница заполнения история болезни

На рисунке 6 представлен дизайн макета страницы запись к врачу.

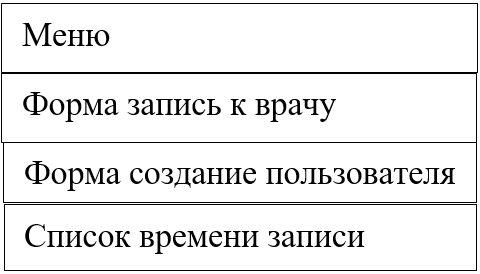


Рисунок 6 – Дизайн макета страница запись к врачу

В рамках данной под главы были разработан дизайн макета разрабатываемого приложения.

3.3 Разработка компонентов веб-приложения

Веб приложение состоит из 2 модулей frond-end и back-end приложения, которые общаются между собой при помощи сетевого протокола http/ https.

Backend и frontend взаимодействуют между собой по следующему принципу. Пользователь попадает на Web-сайт и взаимодействует с пользовательским интерфейсом. Нажав на какую-либо кнопку frontend отправляет запрос на серверную сторону сервиса. Backend принимает, обрабатывает запрос и придает запросу определенный вид.

В backend используется ORM typeORM с помощью которого осуществляется взаимодействие с базой данных и swagger для автоматического документирование проекта.

В frontend используется библиотека React Router – для реализации навигации по приложению.

Схема базы данных представлена на рисунке 7.

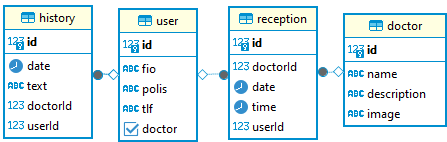


Рисунок 7 - Схема базы данных

На рисунке 8 представлена сущность user.

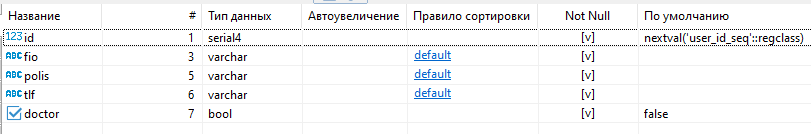


Рисунок 8 - Сущность user

На рисунке 9 представлена сущность doctor

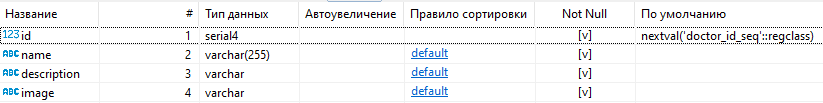


Рисунок 9 - Сущность user

На рисунке 10 представлена сущность history.

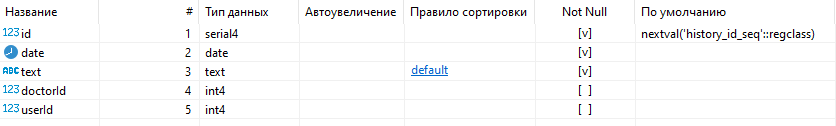


Рисунок 10 – Сущность history

На рисунке 11 представлена сущность reception.

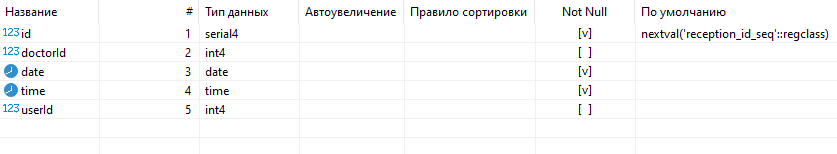


Рисунок 11 – Сущность reception

Ниже представлены конечные точки с помощью swagger.

На рисунке 12 представлено API doctor.

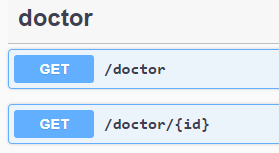


Рисунок 12 – API doctor

На рисунке 13 представлены API reception.

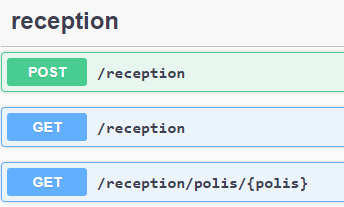


Рисунок 13 – API reception

На рисунке 14 представлены API user.

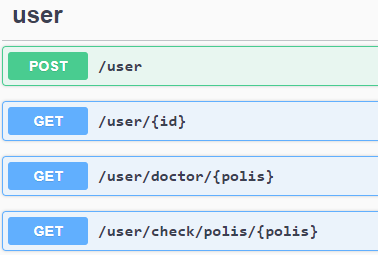


Рисунок 14 – API user

На рисунке 15 представлены API history.

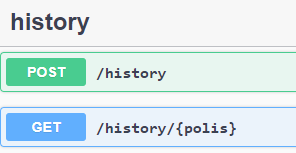


Рисунок 15 – API history

В данной главе были выполнены следующие задачи разработаны структура приложения, разработан макет-дизайн приложения, разработаны веб компоненты приложения, разработана схема базы данных и разработан API веб сервиса.

4 ТЕСТИРОВАНИЕ И ОТЛАДКА ВЕБ-ПРИЛОЖЕНИЯ

Проверим функционал работы нашего приложения.

Для начала просмотрим из меню все страницы на работоспособность.

Начнем с главной.

На рисунке 17 представлена страницы главная.

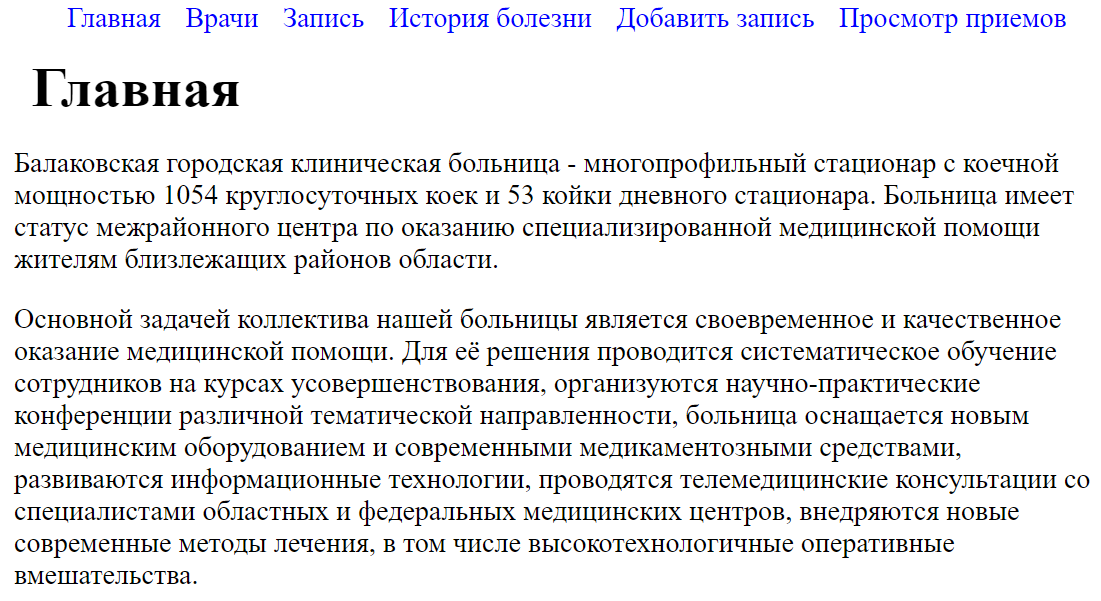


Рисунок 17 – Страница главная

Страница выводит общую информацию о компании.

После чего переходит на страницу врачи.

На рисунке 18 представлена страница врачи.

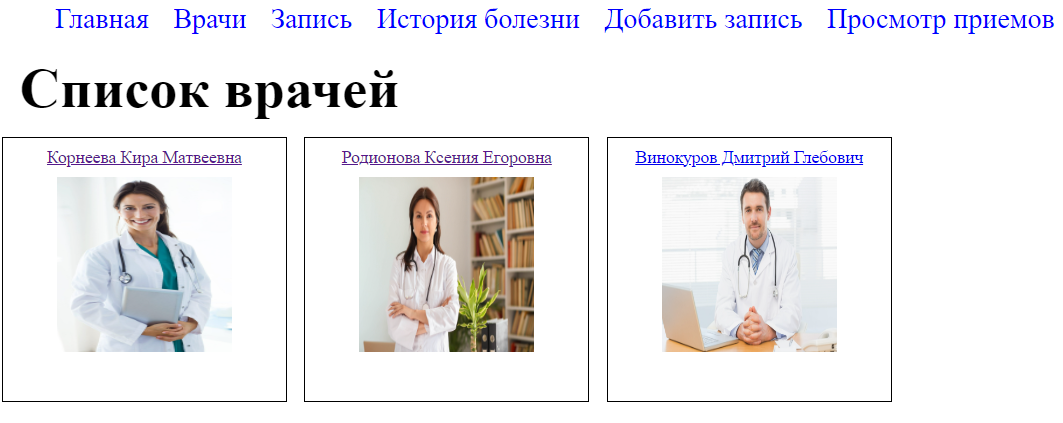


Рисунок 18 – Страница врачи

Front-end отправляет запрос на получение врачей на back-end после чего front-end получает ответ и отображает список врачей с ссылками переходами к странице к конкретному врачу.

На рисунке 19 представлена страница врач.

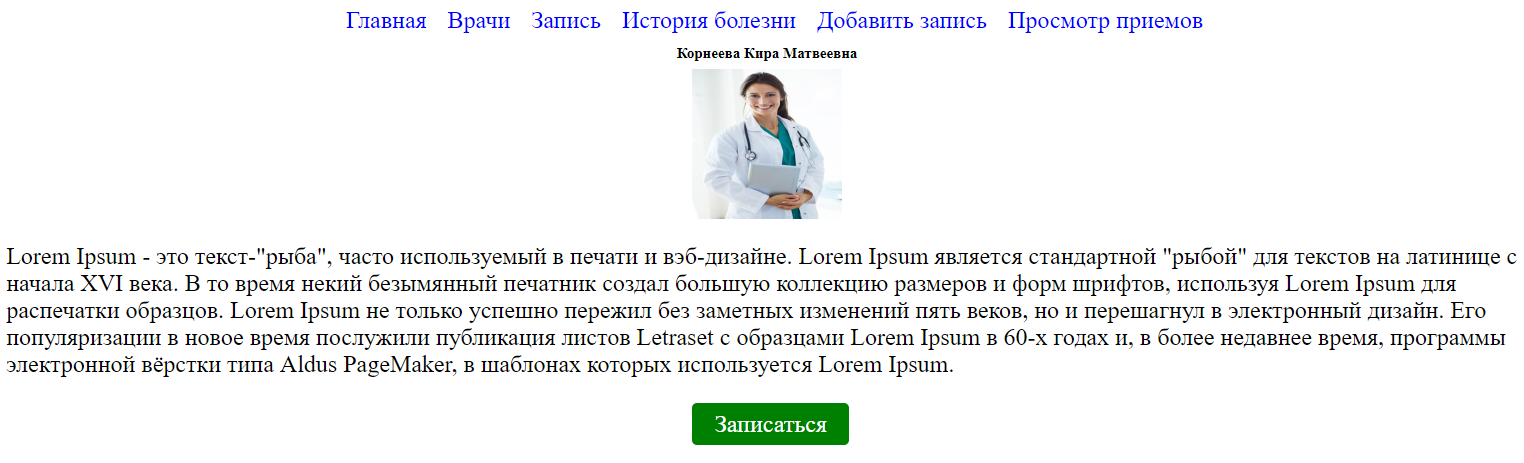


Рисунок 19 – Страница врач

Было проверено что front-end отправляет запрос на получение данных о конкретном враче передай уникальный идентификатор на back-end после чего front-end получает ответ и отображает полученную информацию.

На рисунке 20 представлена реализация страницы запись к врачу.

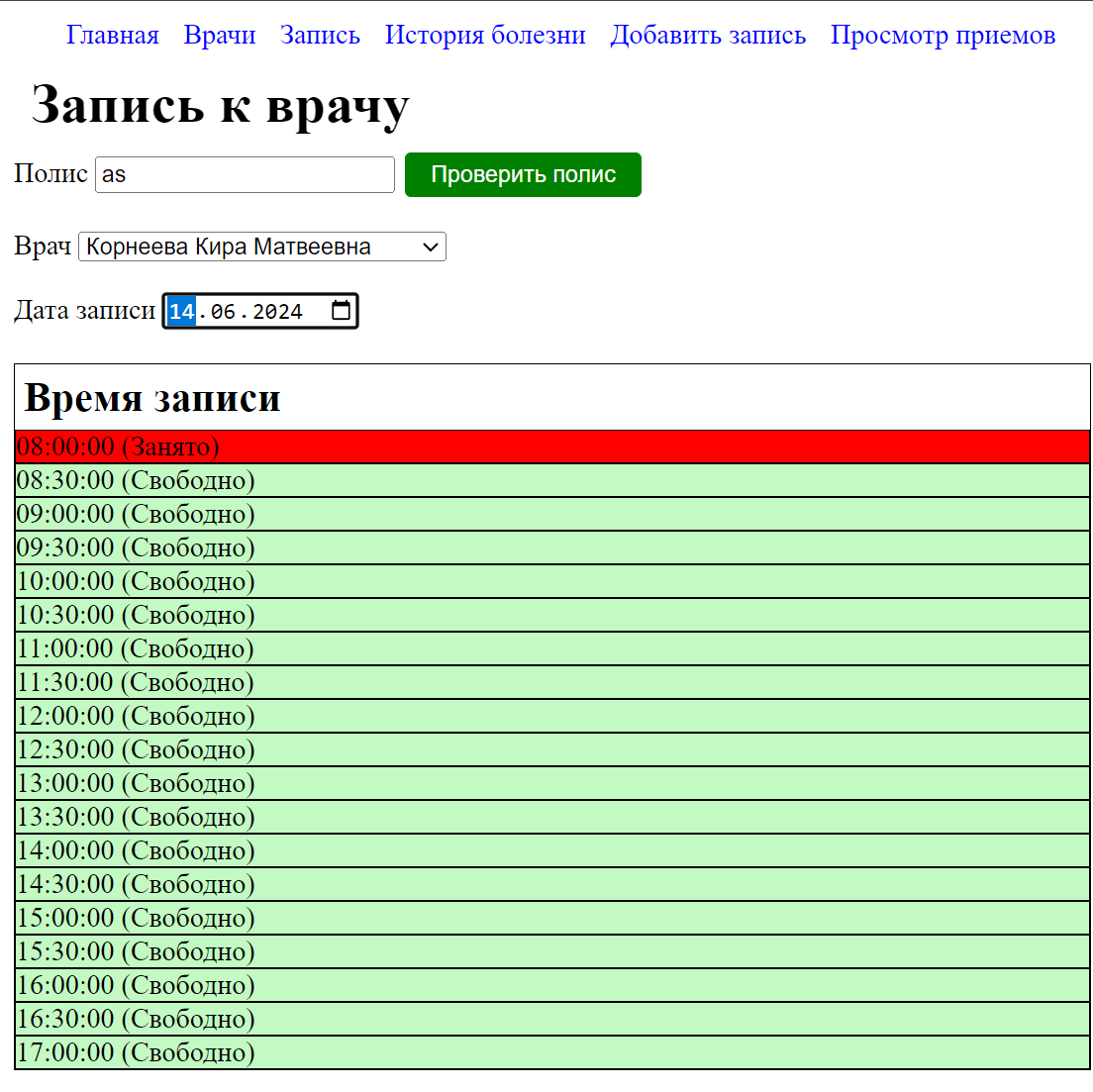


Рисунок 20 – Страница запись к врачу

Было проверено что если пользователь введет свой полис, укажет дату и врача то появится время записей, если пользователь нажмет на свободное время, то появится запись в базе данных, а если попробует нажать на время которое уже занято, то нечего не произойдет, как и ожидалось.

На рисунке 21 представлена реализация страницы история болезни.

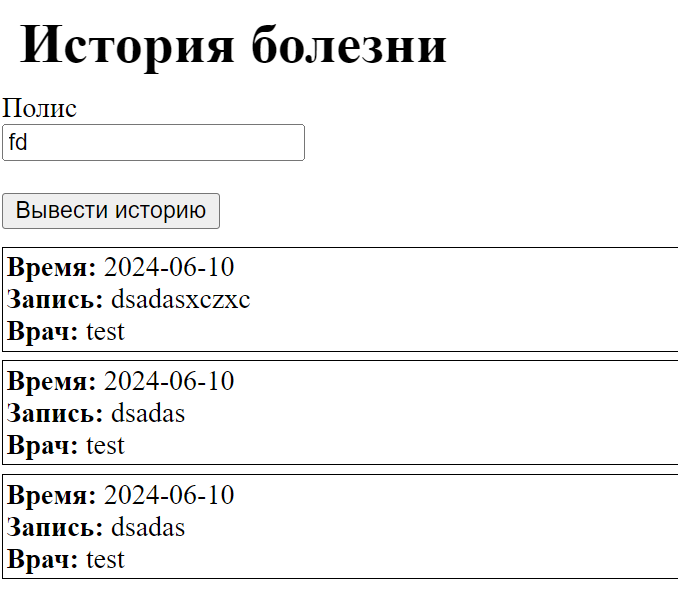


Рисунок 21 – Страница история болезни

На рисунке 22 представлена реализация страницы добавить запись.

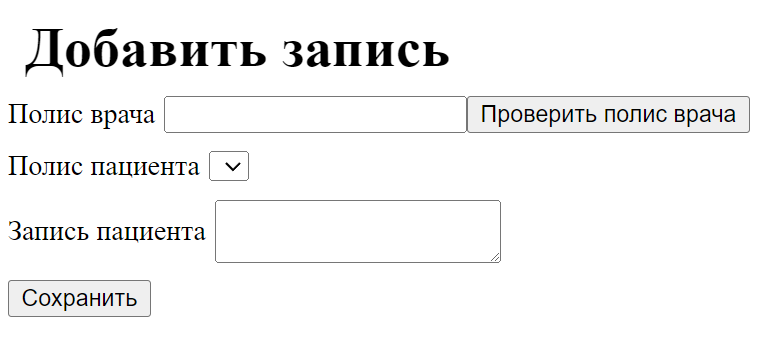


Рисунок 22 - Страницы добавить запись

Было проверено что запись, не может добавить пользователь не являющимся доктором, если проверка полиса не было вызвано или не прошла проверку, то кнопка сохранить не вызывает запрос на back-end.

На рисунке 23 представлена реализация страницы добавить просмотр приемов.

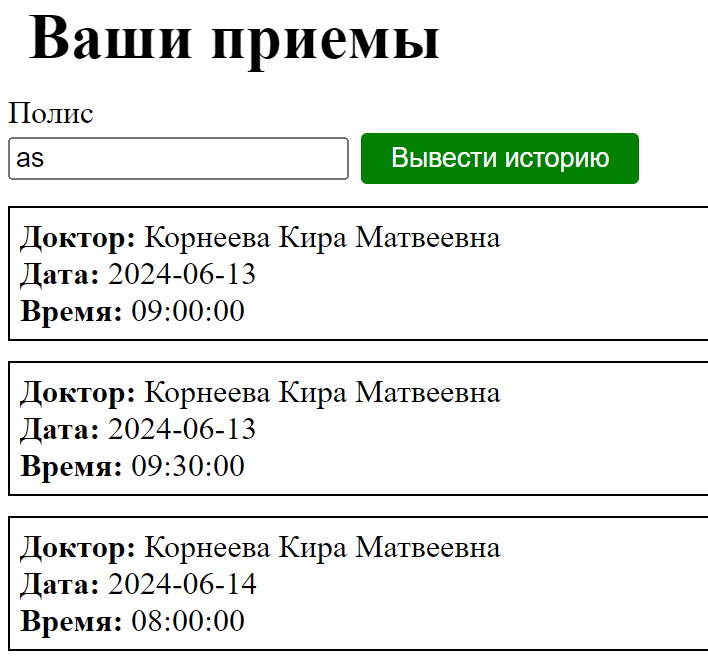


Рисунок 23 - Страницы просмотр приемов

В данной главе было проведено тестирование приложение, и было выявлено что приложение работает стабильно.

ЗАКЛЮЧЕНИЕ

Ниже представлены выполненные задачи в ходе выполнения курсовой работы.

В первой главе были выполнены следующие задачи произведен анализ предметной области «Поликлиника», произведен анализ целевой аудитории, определены цели компании «Поликлиника», составлен портрет пользователей, разработано техническое задание на разрабатываемый продукт, определить требования и ограничения к разрабатываемому продукту.

В второй главе были выполнены следующие задачи произведен анализ и выбор языка и фреймворка для разработки front-end приложения - язык программирование JavaScript и фреймворк React, произведен анализ и выбор языка и фреймворка разработки back-end приложения - язык программирование JavaScript, фреймворк Nest, произведен анализ и выбор системы управления базой данных - Postgressql, произведен анализ и выбор программное средства для разработки веб приложения. – VS code.

В третьей главе были выполнены следующие задачи разработана структура приложения, определены способы общение front-end с back-end, определены требуемые библиотеки для разработки приложения, разработан макет-дизайн приложения для каждой страницы, разработаны веб компоненты приложения, разработана схема базы данных, разработаны сущности базы данных, разработан API веб сервиса.

В четвертой главе было проведено тестирование приложения, была проверена работоспособность каждого модуля приложения, проверена функциональность front-end и back-end и проверена целостность работы приложения в целом.

Цель курсовой работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лион, У. Разработка веб-приложений GraphQL с React, Node.js и Neo4j : практическое руководство / У. Лион ; пер. с англ. А. Н. Киселева. - Москва : ДМК Пресс, 2023. - 262 с. - ISBN 978-5-93700-185-6. - Текст : электронный. - URL: https://znanium.com/catalog/product/2109522;

2. Солодушкин, С. И. Разработка программных комплексов на языке JavaScript : учебное пособие / С. И. Солодушкин, И. Ф. Юманова ; под общ. ред. В. Г. Пименова ; Министерство науки и высшего образования Россйской Федерации, Уральский федеральный университет. - Екатеринбург : Изд-во Уральского ун-та, 2020. - 132 с. - ISBN 978-5-7996-3034- 8. - Текст : электронный. - URL: https://znanium.com/catalog/product/1936353;

3. Стоянович, С. Бессерверные приложения на JavaScript : практическое руководство / С. Стоянович, А. Симович ; пер. с анг. А. Н. Киселева. - Москва : ДМК Пресс, 2020. - 394 с. - ISBN 978-5-97060-782-4. - Текст : электронный. - URL: https://znanium.com/catalog/product/1094948;

4. Хорстманн, К. С. Современный JavaScript для нетерпеливых : практическое руководство / К. С. Хорстманн ; пер. с англ. А. А. Слинкина. - Москва : ДМК Пресс, 2021. - 288 с. - ISBN 978-5-97060-177-8. - Текст : электронный. - URL: https://znanium.com/catalog/product/2109482;

5. Хортон, А. Разработка веб-приложений в ReactJS : практическое руководство / А. Хортон, Р. Вайс ; пер. с англ. Р. Н. Рагимова. - 2-е изд. - Москва : ДМК Пресс, 2023. - 255 с. - ISBN 978-5-89818-503-9. - Текст : электронный. - URL: https://znanium.com/catalog/product/2107174.

ПРИЛОЖЕНИЕ А

Ниже представлен код front-end приложения файла App.js

import React from 'react';

import { BrowserRouter as Router, Route, Link, Routes } from 'react-router-dom';

import { MainPage } from "./pages/mainPage"

import { DoctorsPage } from "./pages/doctorsPage"

import { RecordPage } from './pages/recordPage';

import { DoctorPage } from './pages/doctorPage';

import { HistoryPage } from './pages/historyPage';

import { AddHistoryPage } from './pages/addHistoryPage';

import { RecordUserIdPage } from './pages/recordUserIdPage';

export default function App() {

return (

<Router>

<div>

<nav className="nav">

<Link className="link" to="/">Главная</Link>

<Link className="link" to="/doctors">Врачи</Link>

<Link className="link" to="/record">Запись</Link>

<Link className="link" to="/history">История болезни</Link>

<Link className="link" to="/history/add">Добавить запись</Link>

<Link className="link" to="/record/get">Просмотр приемов</Link>

</nav>

<Routes>

<Route path="/" element={<MainPage />} />

<Route path="/doctors" element={<DoctorsPage />} />

<Route path="/record" element={<RecordPage />} />

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

<Route path="/history" element={<HistoryPage />} />

<Route path="/history/add" element={<AddHistoryPage />} />

<Route path="/record/get" element={<RecordUserIdPage />} />

</Routes>

</div>

</Router>

);}

Ниже представлен код front-end приложения файла user.js – который отправляет запросы на back-end

import axios from "axios";

import { baseApi } from "./base";

export function apiUser() {

const { MainApi } = baseApi();

const checkPolisUser = async (polis) => {

const { data } = await axios.get(`${MainApi}/user/check/polis/${polis}`);

return data;

}

const saveUser = async (body) => {

const { data } = await axios.post(`${MainApi}/user`, body);

return data;

}

const recordUser = async (polis) => {

const { data } = await axios.get(`${MainApi}/reception/polis/${polis}`);

return data;

}

return {

checkPolisUser,

saveUser,

recordUser

}

}

Ниже представлен код файла recordPage.jsx

import { useEffect, useState } from "react"

import { apiDoctors } from "../api/doctors"

import { useSearchParams } from "react-router-dom";

import { TimeListRecord } from "../components/timeListRecord";

import { apiUser } from "../api/user";

export function RecordPage() {

const { checkPolisUser, saveUser } = apiUser()

const options = [];

const [searchParams] = useSearchParams();

const [activeDoctor, activeDoctorSave] = useState("");

const [activeDate, activeDateSave] = useState("");

const [polis, polisSave] = useState("");

const [polisCheckApi, polisCheckApiSave] = useState(undefined);

const [checkPolis, checkPolisSave] = useState(false);

const [doctors, doctorsSave] = useState([]);

const [userFIO, userFIOSave] = useState("");

const [userTLF, userTLFSave] = useState("");

// состояние активного доктора

const selectDoctorChange = (e) => {

activeDoctorSave(e.target.value);

}

// состояние фио пользователя

const selectUserFIOChange = (e) => {

userFIOSave(e.target.value);

}

// состояние телефона пользователя

const selectUserTLFChange = (e) => {

userTLFSave(e.target.value);

}

// состояние полиса пользователя

const polisSaveChange = (e) => {

checkPolisSave(false);

polisCheckApiSave(undefined);

polisSave(e.target.value);

}

// состояние активной даты

const activeDateSaveChange = (e) => {

activeDateSave(e.target.value);

}

// создать пользователя

const saveUserButton = async () => {

if (userFIO && userTLF && polis) {

await saveUser({ polis: polis, fio: userFIO, tlf: userTLF })

polisCheckApiSave(true);

checkPolisSave(true)

}

}

// проверить полис

const checkPolisRun = async () => {

if (polis) {

const status = await checkPolisUser(polis);

polisCheckApiSave(status);

checkPolisSave(status);

}

}

const htmlUser = () => {

if (polisCheckApi == false) {

return <div className="formUser">

<p className="errors">данный полис не найден</p>

<label htmlFor="">Введите ФИО </label>

<input value={userFIO} onChange={selectUserFIOChange}></input>

<br />

<label htmlFor="">Введите номер телефона </label>

<input value={userTLF} onChange={selectUserTLFChange}></input>

<br />

<button onClick={saveUserButton} className="buttonMain">

Сохранить ваши данные</button>

<br />

</div>

}

return <></>

}

useEffect(() => {

(async () => {

doctorsSave(await apiDoctors().doctorsAll());

const searchDoctor = searchParams.get("doctor");

if (+searchDoctor) {

activeDoctorSave(+searchDoctor);

} else if (doctors.length) {

activeDoctorSave(doctors[0].id);

}

})()

}, [])

if (doctors.length) {

for (const doctor of doctors) {

options.push(<option value={doctor.id} key={doctor.id}>{doctor.name}</option>)

}

}

if (doctors.length && !activeDoctor) {

activeDoctorSave(doctors[0].id);

}

const timeListRecord = () => {

if (activeDate && activeDoctor && polis && checkPolis) {

return <TimeListRecord date={activeDate} doctor={activeDoctor} polis={polis} />

}

return <></>

}

return (

<div className="recordPage">

<h1>Запись к врачу</h1>

<div className="formFilter">

<div>

<label htmlFor=""> Полис </label>

<input value={polis} onChange={polisSaveChange} />

<button className="buttonMain" onClick={checkPolisRun}>

Проверить полис</button>

</div>

{htmlUser()}

<br />

<div>

<label htmlFor="">Врач </label>

<select value={activeDoctor} onChange={selectDoctorChange}>{options}</select>

</div>

<br />

<div>

<label htmlFor="">Дата записи </label>

<input value={activeDate} onChange={activeDateSaveChange} type="date" />

{timeListRecord()}

</div>

</div>

</div>)}

Ниже представлен код файла timeListRecord.jsx

import { useEffect, useState } from "react";

import { apiDoctors } from "../api/doctors";

export function TimeListRecord(props) {

const timeHtml = [];

const [timeList, timeListSave] = useState([]);

useEffect(() => {

(async () => {

timeListSave(await apiDoctors().doctorsGetTime(props.doctor, props.date));

})()

}, [props.date, props.doctor])

if (timeList?.length) {

for (const [index, timeElem] of timeList.entries()) {

const classNameActive = timeElem.status == 1 ? "timeElement closeTime" : "timeElement";

const textTime = timeElem.status == 1 ? "Занято" : "Свободно";

const click = async () => {

if (timeElem.status != 1) {

console.log(props);

try {

await apiDoctors().receptionSave({ idDoctor: props.doctor, time: timeElem.time, date: props.date, polis: props.polis });

let copy = Object.assign([], timeList);

copy[index].status = 1;

timeListSave(copy);

} catch (e) {

console.error(e);

}

}

}

timeHtml.push(<div onClick={click} key={timeElem.time} className={classNameActive}>{timeElem.time} ({textTime}) </div>);

}

}

return (

<div className="timeElementList">

<h2>Время записи</h2>

{timeHtml}

</div>

)

}

Ниже представлен код файла app.module.ts

import { Module } from '@nestjs/common';

import { DoctorModule } from './doctor/doctor.module';

import { TypeOrmModule } from '@nestjs/typeorm';

import { bdMain } from './database/bdMain';

import { ReceptionModule } from './reception/reception.module';

import { UserModule } from './user/user.module';

import { HistoryModule } from './history/history.module';

@Module({

imports: [TypeOrmModule.forRoot(bdMain), DoctorModule, ReceptionModule, UserModule, HistoryModule],

controllers: [],

providers: [],

})

export class AppModule { }

Ниже представлен код файла bdMain.ts

import { TypeOrmModuleOptions } from "@nestjs/typeorm";

export const bdMain:TypeOrmModuleOptions = {

type: 'postgres',

host: "localhost",

port: 5432,

username: "postgres",

password: "1",

database: "web",

synchronize: true,

autoLoadEntities: true,

}

Ниже представлен код файла user.controller.ts

import { Controller, Get, Post, Body, Patch, Param, Delete } from '@nestjs/common';

import { UserService } from './user.service';

import { CreateUserDto } from './dto/create-user.dto';

import { ApiTags } from '@nestjs/swagger';

@ApiTags('user')

@Controller('user')

export class UserController {

constructor(private readonly userService: UserService) { }

@Post()

create(@Body() createUserDto: CreateUserDto) {

return this.userService.create(createUserDto);

}

@Get(':id')

findOne(@Param('id') id: string) {

return this.userService.findOne(+id);

}

@Get('doctor/:polis')

async checkDoctor(@Param('polis') polis: string) {

return await this.userService.checkPolisAndDoctor(polis);

}

@Get('/check/polis/:polis')

checkPolis(@Param('polis') polis: string) {

return this.userService.checkPolis(polis);

}

}

Ниже представлен код файла user.service.ts

import { Injectable } from '@nestjs/common';

import { CreateUserDto } from './dto/create-user.dto';

import { Repository } from 'typeorm';

import { InjectRepository } from '@nestjs/typeorm';

import { UserEntity } from './entities/user.entity';

@Injectable()

export class UserService {

constructor(

@InjectRepository(UserEntity)

private userRepository: Repository<UserEntity>,

) {}

create(createUserDto: CreateUserDto) {

return this.userRepository.save(createUserDto);

}

async checkPolisAndDoctor(polis: string) {

const polisFind = await this.findPolis(polis);

if (polisFind.doctor) {

return await this.userRepository.find();

}

return [];

}

async checkPolis(polis: string) {

const polisFind = await this.findPolis(polis);

return !!polisFind;

}

async findPolis(polis: string) {

return await this.userRepository.findOne({ where: { polis: polis } });

}

findOne(id: number) {

return this.userRepository.findOne({ where: { id: id } });

}

}

Ниже представлен код файла user.entity.ts

import { ReceptionEntity } from "src/reception/entities/reception.entity";

import { Column, Entity, OneToMany, PrimaryGeneratedColumn } from "typeorm";

@Entity("user")

export class UserEntity {

@PrimaryGeneratedColumn()

id: number;

@Column()

polis: string;

@Column()

fio: string;

@Column()

tlf: string;

@Column({ default: false })

doctor: boolean;

@OneToMany(() => ReceptionEntity, (reception) => reception.id)

reception: ReceptionEntity[]

}