

Geekbrains

**Разработка высокоэффективного API для управления
медицинскими данными с использованием Java, и страничным
сайтом на JavaScript**

Программа:

Программист Java
Коновалов Алексей Владимирович

Ухта
2024

Содержание

| | |
|--|----|
| Введение | 5 |
| <i>Теоретическая часть</i> | |
| Выбор технологий: | |
| 1.1 Выбор фреймворка для Java..... | 7 |
| 1.2 Выбор фреймворка для JavaScript..... | 9 |
| 1.3 Выбор базы данных..... | 10 |
| API и инструменты: | |
| 2 Виды и выбор API..... | 12 |
| 3.1 Принципы разработки приложения на Spring Boot..... | 14 |
| 3.2 Необходимые инструменты для разработки на Spring Boot..... | 15 |
| 4.1 Принципы разработки приложения на React.. .. | 17 |
| 4.2 Необходимые инструменты для разработки на React. | 18 |
| Версионирование и управление кодом: | |
| 5.1 Принципы работы с Git и GitHub.. .. | 19 |
| Деплой и контейнеризация: | |
| 6.1 Принципы работы с Docker. | 22 |
| Базы данных: | |
| 7.1 Принципы создания моделей в MySQL.. .. | 24 |
| <i>Практическая часть</i> | |
| Frontend: | |
| 1.1 Создание приложения на React с использованием Npm. | 26 |
| 1.2 Категорирование файлов и папок. | 28 |

| | |
|---|----|
| 1.3 Использование JSX..... | 29 |
| 1.4 Запросы через Fetch. | 30 |
| 1.5 Использование Хуков. | 31 |
| 1.6 Исправление багов..... | 32 |
| Backend: | |
| 2.1 Создание приложения на Spring Boot. | 33 |
| 2.2 Категорирование файлов и папок..... | 34 |
| 2.3 Подключение к базе данных..... | 36 |
| 2.4 Использование JPA. | 37 |
| База данных: | |
| 3.1 Моделирование моделей базы данных..... | 39 |
| 3.2 Создание SQL файлов для инициализации первичных данных..... | 41 |
| Деплой и управление: | |
| 4.1 Создание Docker образов для каждого сервиса. | 43 |
| 4.2 Использование Docker-compose. | 44 |
| 4.3 Сокращение времени обновления. | 47 |
| Управление проектом: | |
| 5.1 Использование GitHub Project. | 48 |
| 5.2 Использование NGINX проху..... | 49 |
| Тестирование: | |
| 6.1 Тестирование..... | 50 |
| 6.2 Вход и регистрация. | 51 |
| 6.3 Регистратура..... | 53 |
| 6.4 Врачи | 55 |

| | |
|--------------------------------------|----|
| Заключение..... | 57 |
| Список используемой литературы. | 59 |

Введение

Тема проекта: Разработка высокоэффективного API для управления медицинскими данными с использованием Java, и страничным сайтом на JavaScript.

Цель: создать эффективное и надежное приложение для управления медицинскими данными через API, реализованное на языке программирования Java, а также разработать интуитивно понятный и функциональный веб-сайт на JavaScript для взаимодействия с этим API.

Какую проблему решает: Проблема, которую решает предложенный проект, заключается в отсутствии единой информационной базы в медицинской сфере. В современных условиях человек, временно покидая свое постоянное место пребывания (например, в командировке или на отдыхе), может столкнуться с необходимостью обратиться в медицинское учреждение для получения неотложной медицинской помощи или лечения. При этом ему создается новая медицинская карта, в которую вносятся данные о проведенных обследованиях, лечении и других медицинских манипуляциях. Такая ситуация повторяется при каждом обращении в новое медицинское учреждение.

Этот фрагментированный подход к хранению медицинской информации приводит к серьезным проблемам. Врачи не имеют доступа к полной медицинской истории пациента, так как данные разбросаны по различным медицинским учреждениям. Они вынуждены ориентироваться только на текущие симптомы и состояние пациента, что может затруднять точный диагноз и разработку эффективного лечебного плана.

Целью нашего проекта является создание единой информационной системы, которая позволит врачам получать доступ к общей и полной медицинской истории пациента независимо от его места пребывания. Это позволит врачам более точно диагностировать заболевания, предупреждать возможные осложнения, учитывать аллергии и прочие факторы, влияющие на состояние пациента.

С развитием цифровых технологий в медицине становится критически важно обеспечить удобство и безопасность управления медицинскими данными. Наш

проект направлен на создание универсального API (интерфейса программирования приложений), который обеспечит доступ к медицинским данным, а также на разработку веб-приложения, которое позволит пользователям комфортно и безопасно взаимодействовать с этой информацией через веб-интерфейс..

Задачи:

1. Изучить фреймворки для создания приложений на Java и JavaScript, а также Базы данных для использования.
2. Рассмотреть основные виды связи приложений и построение API.
3. Ознакомиться с основными принципами создания приложений на Java и JavaScript.
4. Составить план создания приложений с использованием Базы данных.
5. Найти и использовать один из видов упаковки сервисов.
6. Разработать приложения используя ранее изученные материалы, а также упаковать.

Инструменты: Postman, Npm, .Node, Docker Desktop, GitHub, Git.

Состав команды: ФИО (Тестировщик)

Теоретическая глава

1.1 Выбор фреймворка для Java

При разработке программного обеспечения ключевым фактором является выбор подходящего инструмента, который может определить успех проекта. В процессе исследования фреймворков для создания API на Java был проанализирован широкий спектр альтернатив.

Первым вариантом, рассмотренным в данном исследовании, стал Spark Framework. Его легковесность и простота использования оказались весьма привлекательными. Однако, дальнейший анализ показал, что для эффективной реализации поставленных задач может потребоваться более обширный набор функциональных возможностей.

Следующим вариантом, на который обратилось внимание, стал Dropwizard. Этот фреймворк предоставляет более комплексное решение, включая механизмы для создания RESTful API, метрики, логирование и прочее. Однако после проведения сравнительного анализа стало понятно, что для конкретного проекта Dropwizard может оказаться излишне сложным и монолитным, что не соответствует требуемому уровню гибкости.

Наконец, было принято решение обратить внимание на Spring Boot. Этот фреймворк привлек внимание своей широкой популярностью и разнообразием возможностей, а также активным сообществом разработчиков. Изучение его функциональности показало, что создание RESTful сервисов с использованием Spring Boot происходит легко и эффективно. Автоконфигурация, встроенные средства тестирования и поддержка различных баз данных подтвердили уверенность в том, что проект будет успешно развиваться в долгосрочной перспективе.

Еще одним преимуществом Spring Boot является наличие обширного набора готовых библиотек для решения типовых задач. Например, Spring Security для обеспечения безопасности API, Spring Data JPA для работы с базами данных, а также поддержка микросервисной архитектуры через Spring Cloud.

В итоге, выбор Spring Boot для разработки API на Java был сделан обоснованно и основан на его простоте, гибкости, обширной функциональности и поддержке разработчиков. Альтернативы, такие как Spark и Dropwizard, имеют свои преимущества, однако не смогли предложить тот же уровень удобства, гибкости и поддержки, что обеспечивает Spring Boot.

1.2 Выбор фреймворка для JavaScript

При выборе инструмента для создания одностраничных веб-приложений с учетом механизмов роутинга, управления состоянием и других важных аспектов, представлено множество альтернатив. Начав исследование Angular, я быстро обнаружил, что для освоения этого фреймворка потребуется значительное время из-за обилия понятий и концепций, которые необходимо освоить.

Затем мое внимание привлекла библиотека Vue.js. Ее легковесность и простота использования оказались удивительно привлекательными. Vue.js предлагает интуитивно понятный синтаксис, который позволяет быстро создавать пользовательские интерфейсы. Проведя время на создание прототипов с помощью Vue.js, я оценил его гибкость и простоту интеграции с другими инструментами. Однако, также было важно иметь поддержку от активного сообщества разработчиков и надежные инструменты для масштабирования проекта.

Когда я начал изучать React, меня поразили его подход к созданию пользовательских интерфейсов. Концепции JSX, компонентный подход, виртуальный DOM были понятны и легко применимы с самого начала. React предоставляет гибкие инструменты для создания динамических и масштабируемых интерфейсов, а также имеет обширное сообщество разработчиков, готовых делиться опытом и знаниями.

Для меня одним из крупных преимуществ React стала его экосистема. Библиотеки и инструменты, такие как Redux для управления состоянием, React Router для навигации в приложении, а также множество других плагинов и компонентов, делают процесс разработки веб-приложений на React более удобным и эффективным.

В итоге, я принял решение остановить свой выбор на React для разработки интерфейса моего веб-приложения. Его легковесность, понятный синтаксис, поддержка от активного сообщества разработчиков и обширная экосистема инструментов убедили меня в том, что это именно то, что необходимо для успешной реализации проекта.

1.3 Выбор Базы данных

При выборе базы данных для реализации моего проекта, я столкнулся с множеством вариантов, каждый из которых предлагал свои особенности и преимущества. Однако, среди всех доступных альтернатив, MySQL выделялся как наиболее подходящий выбор.

Мое знакомство с MySQL началось с осознания его широкой популярности и долгой истории успеха в сфере баз данных. Эта реляционная Система Управления Базами Данных (СУБД) с открытым исходным кодом широко используется в различных проектах, начиная от небольших веб-сайтов и заканчивая крупными корпоративными системами. Ее надежность и стабильность подтверждены временем, что сразу привлекло мое внимание.

Глубже погрузившись в изучение MySQL, я оценил его простоту использования и мощные возможности. Создание и управление базами данных в MySQL оказались относительно простыми благодаря понятному SQL-синтаксису и интуитивно понятному интерфейсу. Широкий спектр инструментов для администрирования, таких как phpMyAdmin или MySQL Workbench, делают работу с базой данных еще более удобной.

Одним из ключевых преимуществ MySQL для меня стала его масштабируемость. Он поддерживает работу с большими объемами данных, что делает его прекрасным выбором как для небольших проектов, так и для крупных систем. Это означает, что я могу начать с небольшой базы данных и масштабировать ее по мере роста моего проекта, не беспокоясь о переходе на другую СУБД.

Кроме того, MySQL предоставляет широкий набор возможностей для обеспечения безопасности данных. От различных уровней доступа до поддержки шифрования данных и аудита, MySQL предоставляет инструменты для защиты информации и соблюдения стандартов безопасности.

Несмотря на то, что на рынке существует множество альтернативных СУБД, таких как PostgreSQL, SQLite, MongoDB и другие, мой выбор остановился на MySQL из-

за его широкой распространенности, надежности, масштабируемости и удобства использования.

Таким образом, MySQL стал для меня идеальным выбором для хранения и управления данными в моем проекте. Его возможности позволяют мне эффективно работать с данными, обеспечивая надежность, безопасность и масштабируемость для моего приложения.

2 Виды и выбор API

API (Application Programming Interface) являются неотъемлемым элементом современного программирования, предоставляя способ взаимодействия между различными компонентами программного обеспечения. Существует несколько типов API, каждый из которых имеет свои особенности и преимущества.

SOAP — один из наиболее старых и широко используемых протоколов для создания веб-сервисов. Он основан на XML и предоставляет строгую спецификацию для обмена данными между клиентом и сервером. SOAP API обычно требует более многословного и строгого подхода к описанию интерфейсов и обмену данными.

REST — это архитектурный стиль, который использует стандартные протоколы HTTP для создания веб-сервисов. Он ориентирован на ресурсы и операции над ними (CRUD: Create, Read, Update, Delete). RESTful API использует удобочитаемые URL-адреса и стандартные методы HTTP (GET, POST, PUT, DELETE) для взаимодействия с сервером.

GraphQL — это относительно новый язык запросов для API, разработанный Facebook. Он позволяет клиентам запрашивать только те данные, которые им нужны, и возвращать их в определенном формате. GraphQL API обеспечивает более гибкий и точечный доступ к данным, позволяя клиентам определять структуру и формат ответов.

При выборе технологий для проекта, включающего в себя бэкенд на Spring Boot и фронтенд на React, я внимательно рассмотрел различные виды API. И вот почему REST оказался идеальным выбором:

RESTful API предоставляет простой и понятный способ организации взаимодействия между клиентом и сервером. Его основные принципы, такие как использование URL-адресов для ресурсов и стандартных HTTP методов, делают процесс разработки и поддержки API удобным и интуитивно понятным.

REST позволяет легко добавлять новые ресурсы и операции без изменения структуры API. Это означает, что проект может масштабироваться по мере роста требований, не требуя значительных изменений в коде.

Использование стандартных методов HTTP, таких как кэширование, делает RESTful API более эффективным и производительным. Это особенно важно для веб-приложений, где скорость ответа сервера играет важную роль в пользовательском опыте.

React, будучи интерфейсной библиотекой, исключительно хорошо работает с RESTful API благодаря своей природе обработки состояний и архитектуре, основанной на компонентах. API-интерфейсы RESTful предоставляют четкие конечные точки для компонентов React для извлечения данных, что делает процесс интеграции простым.

Spring Boot обладает встроенной поддержкой для создания RESTful сервисов, предоставляя разработчикам удобные инструменты для создания, тестирования и развертывания API. Благодаря этому, разработка бэкенда на Spring Boot с использованием REST становится быстрой и эффективной.

Выбор подходящего типа API для проекта зависит от множества факторов, включая тип приложения, требования к производительности, гибкость и удобство использования. В нашем случае, RESTful API оказался наилучшим выбором для сочетания Spring Boot и React, обеспечивая нам простоту, гибкость, производительность и легкость интеграции между бэкендом и фронтендом. Каждый проект уникален, и можно выбрать другой тип API в зависимости от требований и предпочтений. Но в случае, если вы работаете с комбинацией Spring Boot и React, RESTful API обеспечивает превосходное сочетание удобства, производительности и гибкости для успешной разработки приложения.

3.1 Принципы разработки приложения на Spring Boot

Spring Boot основан на принципах IoC и DI, что означает, что контейнер Spring управляет объектами приложения. IoC делегирует контроль за созданием объектов контейнеру, в то время как DI позволяет внедрять зависимости в классы, вместо того чтобы они создавали их самостоятельно.

Spring Boot использует множество аннотаций для конфигурирования приложения. Например, `@SpringBootApplication` объявляет основной класс приложения, `@Controller` используется для обозначения контроллеров, `@Autowired` позволяет внедрять зависимости и многое другое.

Spring Boot поставляется с встроенными веб-контейнерами, такими как Tomcat или Jetty. Это означает, что для разработки и тестирования приложения не нужно устанавливать и настраивать отдельный веб-сервер.

Maven — это инструмент управления зависимостями, который используется для создания и сборки проектов. Он позволяет определить зависимости проекта в файле `pom.xml` и автоматически загрузить их из удаленных репозиторий.

3.2 Необходимые инструменты

JDK является необходимым компонентом для работы с Java и Spring Boot. Мы можем скачать и установить JDK 21 с официального сайта Oracle или OpenJDK.

Скачав JDK 21 с официального сайта Oracle или OpenJDK. Запустим установочный файл и следуем инструкциям установщика. После установки убедимся, что переменные среды JAVA_HOME и PATH настроены правильно. Для этого добавим путь до установленной JDK в переменную JAVA_HOME, а затем добавим %JAVA_HOME%\bin в переменную PATH.

Maven — это инструмент управления зависимостями и сборки проекта, который поможет управлять библиотеками и зависимостями моего приложения.

Установка Apache Maven:

- Скачиваем Apache Maven с официального сайта.
- Распаковываем архив в удобное место.
- Устанавливаем переменную среды MAVEN_HOME, указав путь до папки с Maven.
- Добавим %MAVEN_HOME%\bin в переменную PATH.

Создание нового проекта Spring Boot с помощью Maven:

Теперь, когда у меня установлены JDK 21 и Apache Maven, я готов создать новый проект Spring Boot.

- Откроем терминал или командную строку.
- Создадим новый проект с помощью команды Maven:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=demo -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Эта команда создаст новый проект с groupId com.example и artifactId demo. Я могу изменить эти значения на свои.

Перейдем в директорию моего нового проекта:

cd demo

Добавим зависимость для Spring Boot в файл pom.xml проекта:

```
<dependency>  
  
    <groupId>org.springframework.boot</groupId>  
  
    <artifactId>spring-boot-starter-web</artifactId>  
  
</dependency>
```

Соберем и запустим проект:

```
mvn spring-boot:run
```

Теперь у меня есть базовый проект Spring Boot, который запускается на встроенном веб-сервере. Я могу начать добавлять контроллеры, сервисы и другие компоненты для разработки приложения.

4.1 Принципы разработки приложения на React

React основан на компонентном подходе, что означает, что пользовательский интерфейс разбивается на независимые и переиспользуемые компоненты. Это позволяет создавать чистый и организованный код, упрощает его тестирование и поддержку.

React использует виртуальный DOM для оптимизации производительности при обновлении пользовательского интерфейса. Вместо множественных манипуляций с реальным DOM, React обновляет только необходимые части виртуального DOM, после чего применяет изменения к реальному DOM.

В React данные передаются вниз по иерархии компонентов от родительских к дочерним. Это делает поток данных более предсказуемым и упрощает отслеживание изменений в приложении.

JSX (JavaScript XML) позволяет писать HTML-подобный код внутри JavaScript, что делает код React более читаемым и удобным для разработки интерфейсов.

4.2 Необходимые инструменты

Node.js — это среда выполнения JavaScript, которая позволяет запускать JavaScript на сервере. npm — это менеджер пакетов для установки и управления зависимостями в проекте React.

Установка Node.js и npm:

- Скачаем и установим Node.js.
- После установки Node.js, npm будет установлен автоматически.

Для создания нового проекта React я буду использовать инструмент create-react-app, который предоставляет готовую структуру проекта и настроенную среду разработки.

Установка create-react-app:

Откроем терминал или командную строку.

Установим create-react-app глобально с помощью npm:

- `npm install -g create-react-app`

Создадим новый проект React с помощью create-react-app:

- `npx create-react-app my-react-app`

my-react-app — это название моего нового проекта. Я могу выбрать любое другое имя.

Перейдем в директорию нового проекта:

```
cd my-react-app
```

Запустим проект:

```
npm start
```

Эта команда запустит новое приложение React на локальном сервере, и сможем начать разработку, открыв его в браузере по адресу <http://localhost:3000>.

5 Принципы работы с Git и GitHub

Git и GitHub — это два инструмента, которые играют важную роль в современной разработке программного обеспечения. Рассмотрим основные принципы работы с ними:

Git — это система управления версиями, позволяющая отслеживать изменения в исходном коде проекта.

Репозиторий:

Локальный репозиторий: когда начинаете работать над проектом, создаете локальный репозиторий на своем компьютере.

Удаленный репозиторий: Этот репозиторий находится на сервере, таком как GitHub, GitLab или Bitbucket. Он используется для совместной работы и обмена кодом с другими разработчиками.

`git commit`: это основной способ "сохранения" изменений в Git. Когда я вношу изменения в свой проект, создаю коммиты, которые представляют собой снимки текущего состояния проекта.

`git branch`: ветвление позволяет создавать отдельные "ветки" моего проекта. Это позволяет разрабатывать новую функциональность или исправлять ошибки, не затрагивая основной код.

`git merge`: после того как работа в ветке завершена, изменения можно включить обратно в основную ветку с помощью операции слияния (`merge`).

Основные команды:

`git init`: Инициализация нового репозитория.

`git add`: Добавление изменений в индекс (`staging area`).

`git commit`: Создание коммита с сохраненными изменениями.

`git push`: Отправка коммитов в удаленный репозиторий.

`git pull`: Получение изменений из удаленного репозитория и объединение их с локальным.

`git merge`: Слияние изменений из одной ветки в другую.

GitHub — это веб-платформа для хостинга проектов, основанных на Git. Он предоставляет дополнительные инструменты для совместной разработки и управления проектами.

Репозитории:

Публичные и частные репозитории: GitHub позволяет создавать как открытые (публичные), так и закрытые (частные) репозитории.

Issues: Проблемы (Issues) используются для отслеживания задач, ошибок и других вопросов в проекте.

Pull Requests: Запросы на включение (Pull Requests) используются для предложения изменений и их обсуждения перед включением в основной код проекта.

Actions и Workflows:

GitHub Actions: Это возможность автоматизировать различные процессы в проекте, такие как сборка, тестирование, развертывание и другие.

GitHub Workflows: это набор задач, которые выполняются при определенных событиях в репозитории (например, при создании Pull Request или при коммите кода).

Используйте ветвление: Разрабатывайте новую функциональность или исправления в отдельных ветках, чтобы избежать конфликтов и сохранить основной код стабильным.

Часто делайте коммиты: регулярно сохраняя свою работу в коммитах, чтобы иметь возможность вернуться к предыдущим версиям проекта, если что-то пойдет не так.

Используйте Pull Requests: перед внесением изменений в основной код проекта, открывайте Pull Request для обсуждения изменений и проверки кода другими разработчиками.

Активно используйте инструменты GitHub: Issues для отслеживания задач и ошибок, Actions и Workflows для автоматизации процессов, Pages для размещения документации и веб-сайтов.

Работа с Git и GitHub становится более эффективной с опытом, поэтому не стесняйтесь экспериментировать и использовать различные функции для улучшения процесса разработки.

6.1 Принципы работы с Docker

Docker — это платформа для разработки, доставки и запуска приложений в контейнерах. Контейнеры предоставляют стандартизированную единицу программного обеспечения, которая упаковывает код и все его зависимости, включая библиотеки и другие ресурсы, в изолированную среду.

Docker Image: это шаблон для создания контейнера. Образ содержит все необходимые файлы и зависимости для запуска приложения.

Dockerfile: это текстовый файл, в котором определяется шаг за шагом, как создать Docker Image. Определяя базовый образ, устанавливаете зависимости, копируете файлы и т. д.

Контейнеры (Containers):

Docker Container: это экземпляр Docker Image, который запущен и работает в изолированной среде. Контейнеры могут быть созданы, запущены, остановлены, перемещены и удалены.

Основные команды:

`docker build`: Создание Docker Image из Dockerfile.

`docker run`: Запуск контейнера на основе Docker Image.

`docker stop`: Остановка контейнера.

`docker rm`: Удаление контейнера.

`docker rmi`: Удаление Docker Image.

Docker Compose: Инструмент для определения и запуска многоконтейнерных Docker приложений. Это позволяет описывать структуру приложения в файле `docker-compose.yml` и управлять всеми контейнерами одной командой.

`docker-compose up`: Запуск приложения на основе файла `docker-compose.yml`.

`docker-compose down`: Остановка и удаление контейнеров, созданных `docker-compose up`.

Работа с Docker Hub:

Docker Hub: Это облачное хранилище образов Docker, аналогичное GitHub для кода.

`docker login`: Вход в Docker Hub из командной строки.

`docker push`: Отправка локального Docker Image в Docker Hub.

`docker pull`: Загрузка Docker Image из Docker Hub на локальную машину.

Основные принципы:

Используйте Dockerfile: Опишите свое приложение и его зависимости в Dockerfile, чтобы создать Docker Image.

Разделяйте сервисы на контейнеры: Используйте контейнеры для разделения сервисов и зависимостей приложения.

Используйте Docker Compose для локальной разработки: Опишите все сервисы приложения в `docker-compose.yml`, чтобы легко запускать и останавливать их вместе.

Используйте образы с официального репозитория: В большинстве случаев для приложения уже существуют официальные образы на Docker Hub. Это поможет избежать необходимости создания образа "с нуля".

Работа с Docker помогает упростить развертывание и управление вашими приложениями, обеспечивая консистентность и изоляцию. Это особенно полезно в средах разработки, тестирования и производства, где вы хотите быть уверены, что приложение будет работать одинаково независимо от окружения.

7.1 Принципы создания моделей в MySQL

При моделировании таблиц в базе данных необходимо учитывать множество аспектов, каждый из которых играет ключевую роль в обеспечении эффективности, надежности и гибкости базы данных.

Во-первых, следует определить структуру данных. Это включает в себя определение каждого поля, его типа данных, ограничений на значения и связей с другими таблицами. Важно правильно выбрать типы данных для каждого поля, учитывая размеры данных и необходимость точности. Например, для числовых значений можно использовать целочисленные или числа с плавающей точкой, а для дат - специализированные типы данных.

Следующий аспект - ключи. Ключи играют решающую роль в базах данных, обеспечивая связи между таблицами и эффективный доступ к данным. Первичный ключ идентифицирует уникальные записи в таблице, а внешний ключ устанавливает связь между двумя таблицами. Правильно выбранные ключи улучшают производительность запросов и обеспечат целостность данных.

Третий аспект - нормализация. Это процесс организации данных в базе данных таким образом, чтобы избежать избыточности и обеспечить целостность. Нормализация позволяет избежать проблем с обновлением и удалением данных, улучшает производительность и экономит место на диске. Однако следует помнить, что избыточная нормализация может привести к сложным запросам и медленной работе системы.

Еще одним важным аспектом является индексация. Индексы позволяют ускорить поиск данных в таблице, особенно при выполнении запросов с условиями. Однако избыточное использование индексов может привести к излишнему использованию ресурсов, таким как память и процессорное время. Поэтому необходимо балансировать между частотой использования запросов и созданием индексов.

Также важно учитывать потребности приложения при моделировании таблиц. Необходимо понимать, какие запросы будут чаще всего выполняться, какие данные чаще всего изменяются, и оптимизировать структуру таблиц под эти требования. Это позволит достичь оптимальной производительности и эффективности работы базы данных.

В конечном итоге, моделирование таблиц в базе данных - это сложный и ответственный процесс, который требует внимательного анализа требований приложения, осознанного выбора структуры данных, ключей и индексов. Правильно спроектированная база данных обеспечит стабильную и эффективную работу приложения, минимизируя возможность ошибок и повышая общую производительность системы.

Практическая глава

1.1 Создание приложение на React с использованием Npm

Для начала создадим проект MedicalHubClient с помощью инструмента Create React App, который позволяет быстро настроить приложение на React. Предполагается, что у вас уже установлен Node.js и npm.

Шаг 1: Установка Create React App

Откройте терминал (командную строку) и выполните следующую команду для установки Create React App глобально на вашем компьютере (если вы еще не установили его):

```
npm install -g create-react-app
```

Шаг 2: Создание проекта MedicalHubClient

Теперь создадим проект MedicalHubClient. В терминале перейдите в папку, где вы хотите создать проект, и выполните следующую команду:

```
npm create-react-app MedicalHubClient
```

Эта команда создаст новую папку "MedicalHubClient" и установит в неё все необходимые файлы и зависимости для приложения на React.

Шаг 3: Переход в папку проекта

Перейдите в только что созданную папку "MedicalHubClient" с помощью команды:

```
cd MedicalHubClient
```

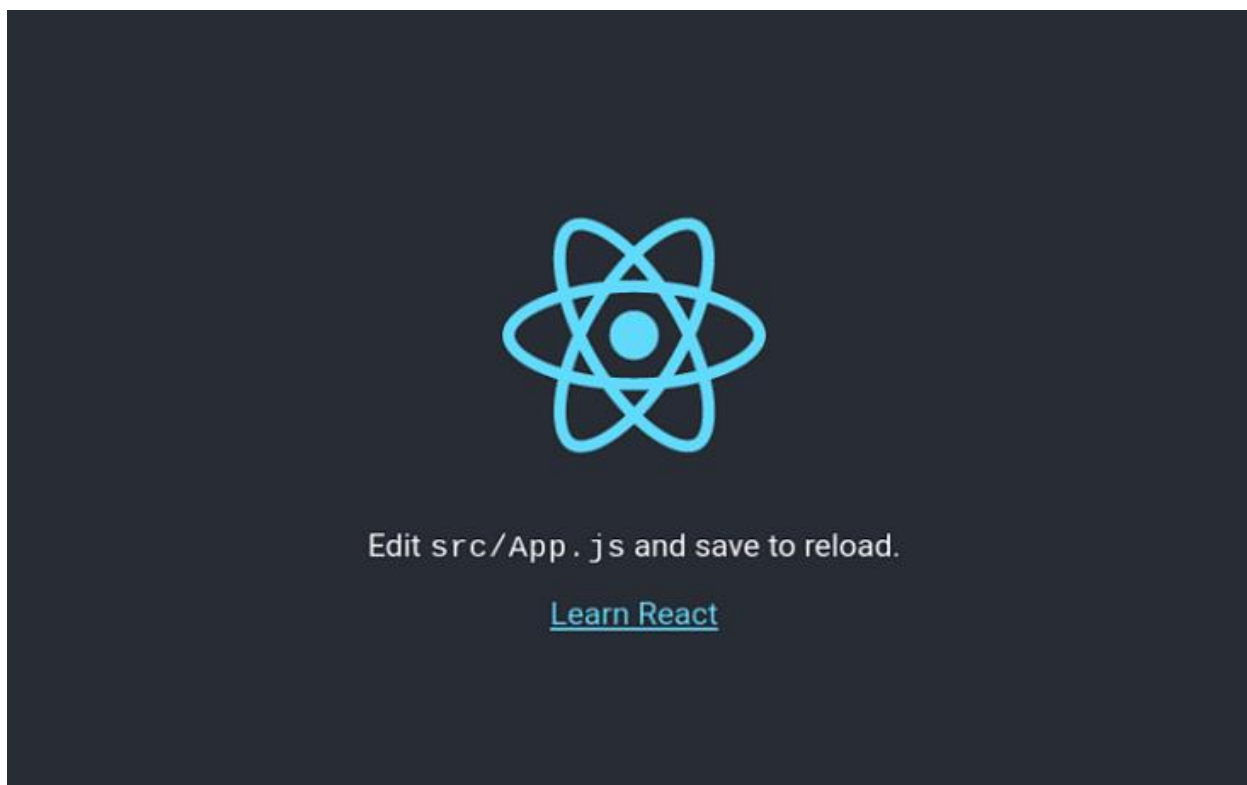
Шаг 4: Запуск проекта

Теперь мы можем запустить наше приложение React. В терминале выполните:

```
npm start
```

После этого ваш браузер автоматически откроет приложение по адресу `http://localhost:3000/`. Вы увидите стандартное окно с ссылкой на React документацию.

Теперь вы можно начать разработку приложения `MedicalHubClient`, редактируя файлы в папке проекта.



1.2 Категорирование файлов и папок

Давайте теперь поговорим о категоризации файлов в папке. Основные операции будут проводиться внутри каталога `src`. В этой директории мы создадим несколько ключевых подкаталогов:

`Core` - здесь будут содержаться основные элементы системы.

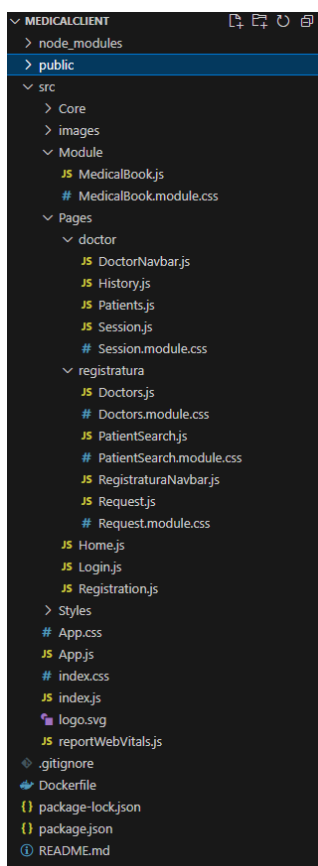
`Images` - этот каталог будет предназначен для хранения изображений.

`Modules` - здесь будут размещаться модули, которые можно использовать в различных частях проекта.

`Pages` - каталог для страниц сайта.

`Styles` - здесь будут находиться файлы, отвечающие за оформление и стили страницы.

Такая система категоризации позволит эффективно организовать структуру проекта, облегчит доступ к необходимым файлам и способствует удобству разработки и поддержки системы.



1.3 Использование JSX

Теперь поговорим о JSX — это расширение языка JavaScript, которое играет ключевую роль в разработке интерфейсов. С помощью JSX мы можем преобразовывать наши JavaScript функции в HTML-подобный код, даже указывая собственные атрибуты элементов.

Взглянем на пример использования JSX:

```
<MedicalBook id={userId} isFromUserId={true} />
```

В данном примере мы создаем элемент `<MedicalBook>`, передавая ему атрибуты `id` с значением `userId` и `isFromUserId` со значением `true`.

Такой подход позволяет нам более наглядно описывать структуру пользовательского интерфейса, упрощает работу с компонентами и делает код более читаемым и понятным.

1.4 Запросы через Fetch

Теперь перейдем к получению данных. В нашем проекте мы используем Fetch API для этой цели, однако не в каждой странице мы пишем запросы вручную. Мы придерживаемся объектно-ориентированного подхода, который позволяет нам экономить время и ресурсы, указывая нужные запросы в удобном виде по всему сайту.

Взглянем на пример сравнения стандартного использования Fetch

```
export async function HttpNoData(url, method, jwt) {  
  try {  
    var response = await fetch(  
      url, {  
        method: method, // *GET, POST, PUT, DELETE, etc.  
        headers: {  
          "Content-Type": "application/json",  
          "Authorization": 'Bearer ' + jwt  
        }  
      });  
    if (response.ok) {  
      try {  
        var responsejson = await response.json();  
        return { statusSuccessful: true, isEmpty: false, data: responsejson };  
      } catch (error) {  
        return { statusSuccessful: true, isEmpty: true, error: error };  
      }  
    }  
    else return { statusSuccessful: false, error: response.status };  
  } catch (error) {  
    return { statusSuccessful: false, error: error };  
  }  
}
```

и нашего подхода с объектно-ориентированным программированием.

```
const response = await HttpNoData('/api/doctor/my/patient', 'GET', jwt);  
  
if(response.statusSuccessful)  
{  
  setData(response.data);  
}else console.log(response.error);
```

Такой подход с классами и объектами позволяет нам структурировать код, делает его более читаемым и понятным. Эффективное использование объектно-ориентированного программирования во время разработки обеспечивает удобство и экономию времени, что, несомненно, способствует популярности нашего подхода.

1.5 Использование Хуков

Использование хуков является одним из ключевых инструментов при работе с данными в React. В нашем проекте мы в основном используем хук `useState` для хранения состояния компонентов и `useEffect` для выполнения побочных эффектов, таких как обновление страницы при изменении данных.

На картинке представлен пример использования `useState` и `useEffect`:

```
const [jwt, setJwt] = useState(props.jwt);
const [data, setData] = useState([]);
const [visibleMedicalBook, setVisibleMedicalBook] = useState(false);
const [currentMedicalBookId, setCurrentMedicalBookId] = useState(-1);

useEffect(() => {
  async function DidMounth(){
    const response = await HttpNoData('/api/doctor/my/patient', 'GET', jwt);
    if(response.statusSuccessful)
    {
      setData(response.data);
    }else console.log(response.error);
  }
  DidMounth();
}, []);
```

Как вы можете видеть, в `useEffect` вторым аргументом указан пустой массив `[]`. Это указывает на то, что эффект должен выполняться только один раз, при первом рендере компонента. Это особенно полезно, когда нам нужно выполнить запрос данных или другие побочные действия только один раз при загрузке компонента.

Такой подход с хуками `useState` и `useEffect` позволяет нам эффективно управлять состоянием компонентов и их жизненным циклом, делая наш код более чистым, читаемым и эффективным.

1.6 Исправление багов

При создании приложения в React мы столкнулись с ошибкой, вызванной двойным выполнением `useEffect` при пробуждении страницы. Эта проблема связана с режимом разработчика и, конкретно, с использованием `<StrictMode>` в файле `index.js`, где наше приложение `App` обернуто в JSX `<StrictMode>`.

Для решения этой проблемы было принято решение просто удалить `<StrictMode>`, поскольку на момент написания этого сообщения соответствующая задача остается открытой на GitHub и не решена.

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <App />
);
```

Это временное решение позволит избежать двойного выполнения `useEffect` и продолжить работу над проектом. Необходимо помнить, что `<StrictMode>` используется для выявления потенциальных проблем в приложении на этапе разработки, поэтому его удаление может потребовать более внимательного контроля за кодом в дальнейшем

2.1 Создание приложение на Spring Boot

Для начала, мы загрузили конфигурационный файл с официального сайта Spring, в котором указали следующие параметры: использование Java, версии JDK21, и сборка проекта с помощью Maven. Наш проект был назван MedicalHubApi, отражая его функциональное предназначение. Дополнительно, инструкции по использованию Spring Boot я нашёл прямо на этом же сайте, что значительно упростило процесс разработки. Модули и пакеты, необходимые для проекта, я скачивал из раздела обучения, что дало возможность быстро находить и интегрировать необходимые пакеты без долгих поисков и дополнительных усилий.

2.2 Категорирование файлов и папок

Корневым элементом нашего проекта является папка src. Уже в ней мы провели категоризацию файлов и папок, учитывая основные функциональные области нашего приложения:

Controller - здесь содержатся классы, отвечающие за обработку запросов от клиентов. Эти классы служат в качестве контроллеров, определяющих точки входа в наше API и логику обработки запросов.

Entity - данная папка содержит классы, которые представляют объекты в базе данных. Это могут быть классы, соответствующие таблицам базы данных или другим структурам данных.

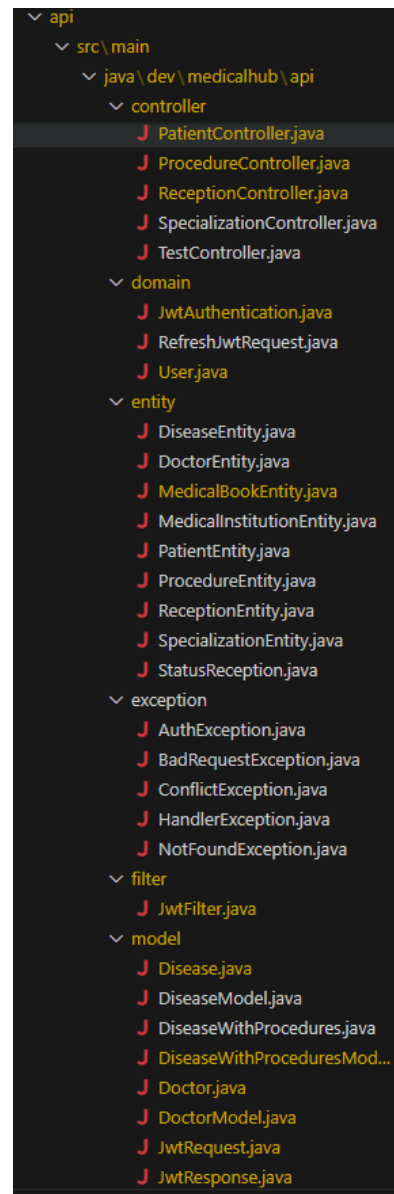
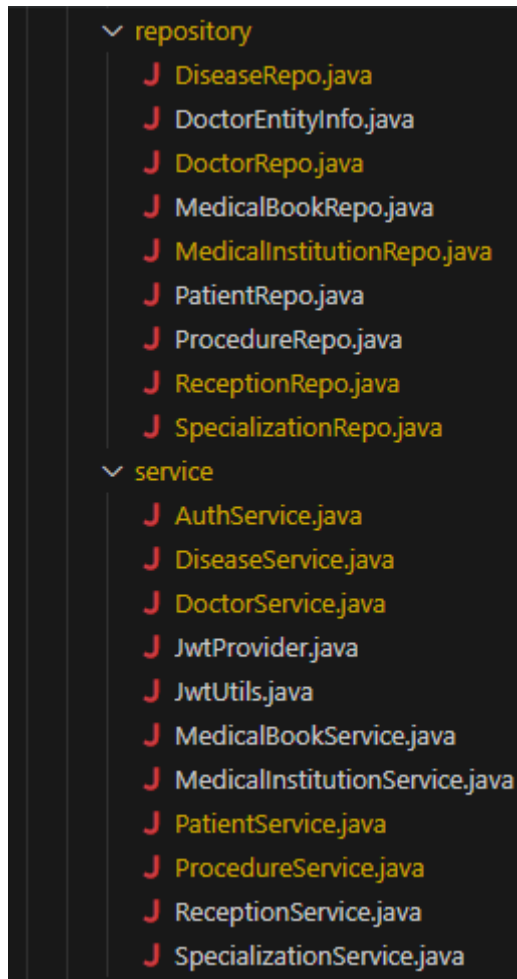
Model - здесь находятся классы, которые используются для представления данных, отправляемых клиенту. Это могут быть классы DTO (Data Transfer Object) или любые другие модели, используемые во взаимодействии с пользовательским интерфейсом.

Exception - в этой папке расположены классы, отвечающие за обработку исключений, которые могут возникнуть при работе с данными. Здесь можно определить собственные классы исключений или обработчики исключений для более точной и информативной обработки ошибок.

Repository - здесь содержатся классы, отвечающие за взаимодействие с базой данных. Эти классы являются частью слоя доступа к данным (Data Access Layer) и содержат логику для выполнения операций CRUD (Create, Read, Update, Delete) с объектами базы данных.

Service - в данной папке находятся классы, отвечающие за бизнес-логику приложения и обработку данных. Эти классы содержат методы и функции для выполнения операций над данными, взаимодействия с репозиториями и другие сервисные функции.

Такая организация папок и файлов в проекте позволяет легко найти нужные компоненты, поддерживать структуру проекта и облегчает совместную разработку, так как каждая функциональная область имеет свою четко определенную область ответственности.



2.3 Подключение к базе данных

Подключение к базе данных осуществляется из файла `application.properties`. В этом файле мы указываем путь к базе данных, а также `username` и `password`, необходимые для аутентификации:

```
#----- DB Connection Properties -----  
spring.datasource.url=jdbc:mysql://db:3306/medicalhub?useUnicode=true&serverTimezone=UTC&createDatabaseIfNotExist=true  
spring.datasource.username=root  
spring.datasource.password=1234
```

В данном примере мы используем имя хоста `db` вместо обычного `localhost`. Это связано с тем, что в нашем проекте мы планируем использовать контейнеризацию. Контейнеризация позволяет упаковать наше приложение в изолированные контейнеры, включая базу данных. Когда мы запустим наше приложение в контейнере, мы также запустим и базу данных в контейнере.

Таким образом, использование имени хоста `db` вместо `localhost` позволяет нам в будущем легко изменить конфигурацию для работы с контейнеризированной базой данных. При запуске контейнеров мы можем связать наше приложение и базу данных, используя внутреннюю сеть `Docker` или другие механизмы контейнеризации.

Подробнее об этом мы рассмотрим в разделе, посвященном контейнеризации нашего приложения.

2.4 Использование JPA

Когда мы начали использовать JPA и репозитории, мы внедрили в наше приложение мощный механизм для работы с базой данных. JPA предоставляет нам удобный способ отображения Java объектов на записи в базе данных, а также выполнения операций CRUD (Create, Read, Update, Delete) без необходимости писать многочисленные SQL запросы.

Репозитории (Repositories):

Репозитории в Spring Boot представляют собой интерфейсы, которые расширяют JpaRepository или другие интерфейсы из Spring Data JPA. Эти интерфейсы предоставляют нам уже реализованные методы для взаимодействия с базой данных. Мы можем создавать собственные методы в репозиториях, и Spring автоматически реализует их на основе именованных запросов или собственных SQL запросов.

Например, мы можем иметь репозиторий для сущности

```
@Repository
public interface MedicalBookRepo extends CrudRepository<MedicalBookEntity, Long> {
    Optional<MedicalBookEntity> findByPatient_Id(Long id);
}
```

Сервисы (Services):

Сервисы в Spring Boot являются компонентами, отвечающими за бизнес-логику нашего приложения. Они инкапсулируют логику работы с данными и предоставляют ее для контроллеров. Сервисы позволяют нам разделить наше приложение на отдельные слои ответственности (Service Layer).

Например, мы можем иметь сервис для работы с пользователями:

```
@Repository
public interface ReceptionRepo extends CrudRepository<ReceptionEntity, Long> {
    public Iterable<ReceptionEntity> findByDoctorDiplomaNumberAndStatus(long
doctorDiplomaNumber, StatusReception status);
    public Iterable<ReceptionEntity> findByDoctorDiplomaNumberAndStatusNot(long
doctorDiplomaNumber, StatusReception status);
}
```

```
List<ReceptionEntity> findByDoctor_DiplomaNumber(long diplomaNumber);  
  
long countByDoctor_DiplomaNumberAndPatient_Id(long diplomaNumber, Long id);  
}
```

Мы можем использовать методы репозитория внутри сервиса для выполнения операций с пользователями, таких как поиск по имени пользователя или сохранение нового пользователя.

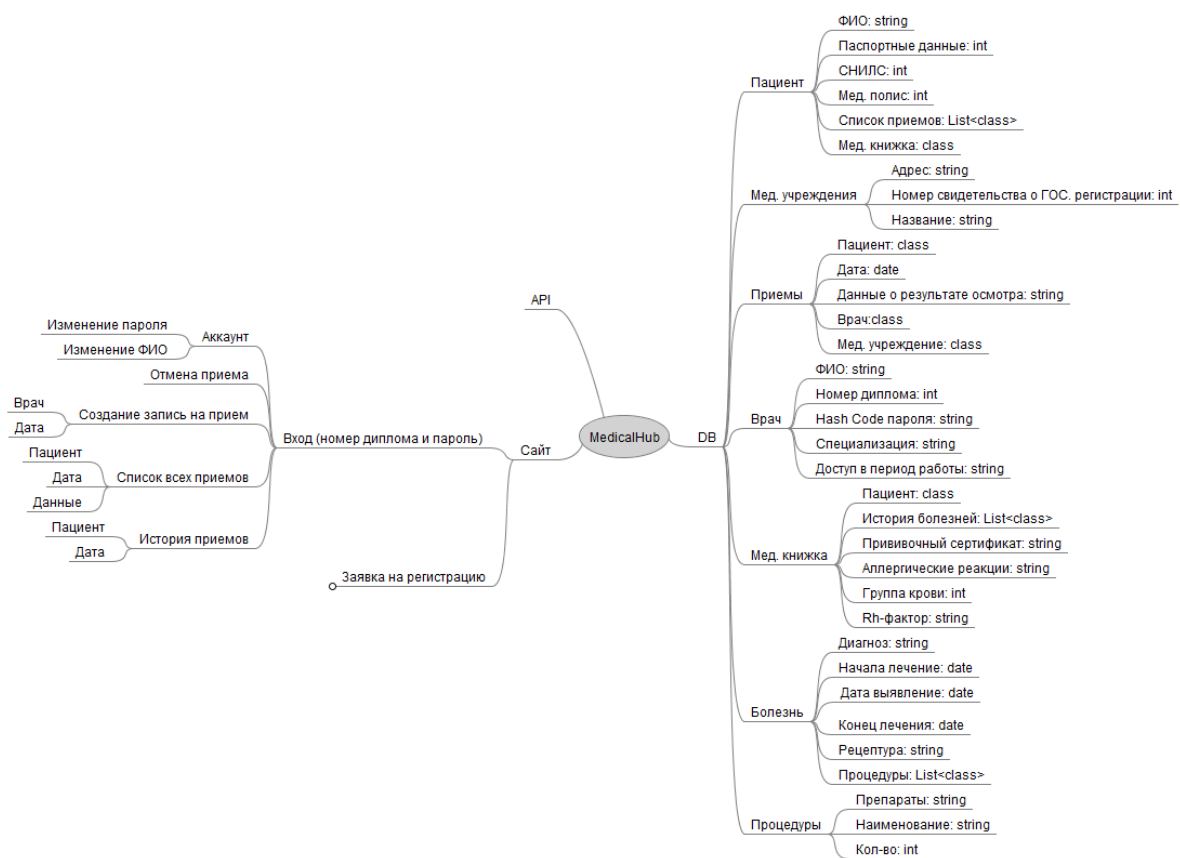
Таким образом, использование JPA и репозитория с сервисами позволило нам создать чистую и структурированную архитектуру приложения. Репозитории обеспечивают удобный доступ к данным, а сервисы инкапсулируют бизнес-логику, что делает наше приложение более гибким, расширяемым и легким для тестирования.

3.1 Моделирование моделей базы данных.

Одним из наиболее увлекательных этапов в нашем проекте было создание моделей базы данных. Для проектирования структуры проекта и его функциональных потребностей мы использовали инструменты, такие как FreeMind и Draw.io. Эти инструменты оказались весьма полезными для создания общей картины действий и визуализации структуры наших моделей.

FreeMind для проектирования:

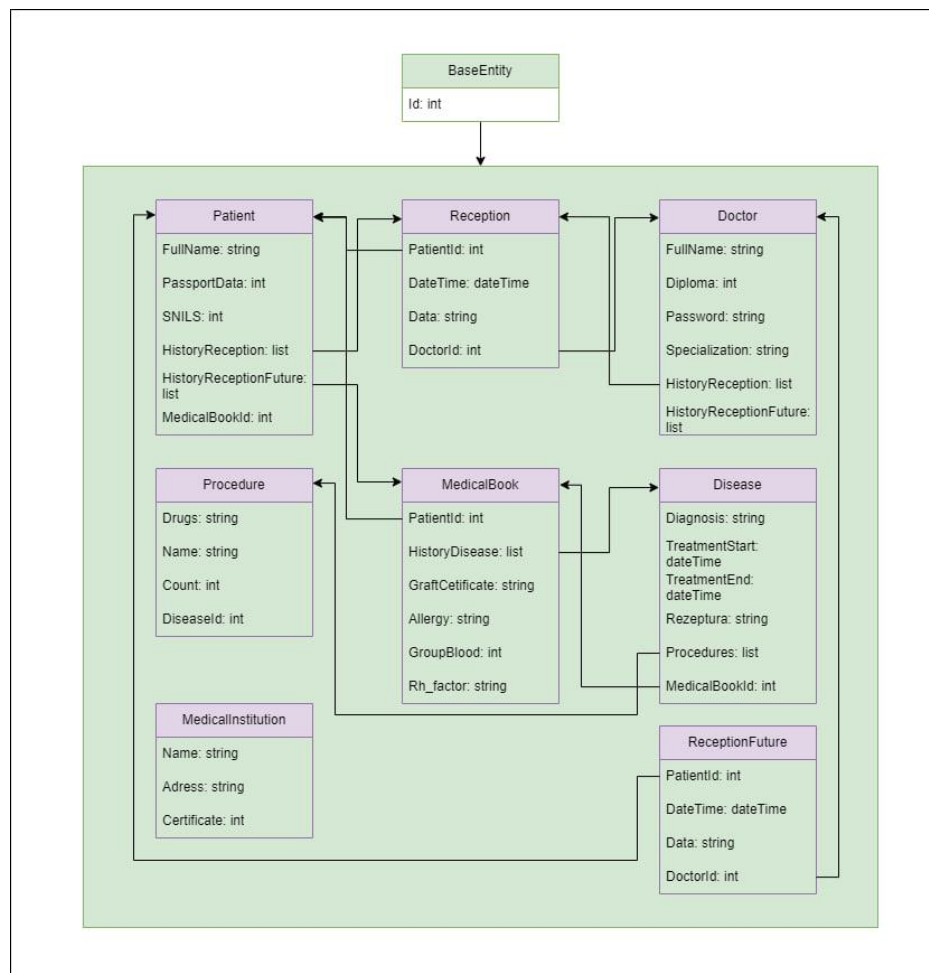
FreeMind позволил нам создать древовидную структуру наших идей, задач и требований. Мы использовали его для создания общего плана проекта, определения основных модулей приложения и их взаимосвязей. Этот инструмент стал своеобразным мозговым штурмом, позволяющим нам видеть общую картину разработки и понимать логику взаимодействия компонентов.



Draw.io для моделей данных:

Draw.io стал нашим инструментом выбора для визуализации самих моделей базы данных. Мы создавали схемы таблиц, определяли связи между сущностями, указывали атрибуты и типы данных. Изначальные версии моделей часто претерпевали изменения в процессе разработки, но благодаря Draw.io мы легко вносили коррективы и обновляли схемы в соответствии с новыми требованиями.

Конечно, исходные модели наших баз данных немного отличались от того, что мы имели в конечном итоге. Это вполне естественно в процессе разработки, когда становится ясно, чего не хватает или что, наоборот, излишне. Подход к проектированию и моделированию позволил нам не только создать эффективную структуру приложения, но и гибко реагировать на изменения в ходе работы над проектом.



3.2 Создание SQL файлов для инициализации первичных данных

Для управления базой данных и обеспечения ее начальной настройки, мы приняли решение использовать SQL файлы. Мы разделили этот процесс на два файла: один для создания таблиц и их инициализации, а другой для заполнения таблиц конкретными данными.

SQL файл для создания таблиц (create_tables.sql):

В этом SQL файле мы определили структуру наших таблиц, их колонки, типы данных и любые другие параметры. Этот файл содержит SQL запросы для создания всех необходимых таблиц в базе данных. Примерно так может выглядеть содержимое файла:

```
create table hibernate_sequence (
    next_val bigint
) engine=MyISAM;

insert into hibernate_sequence values ( 1 );
insert into hibernate_sequence values ( 1 );
insert into hibernate_sequence values ( 1 );

create table disease (
    id bigint not null auto_increment,
    diagnosis varchar(255),
    rezeptura varchar(255),
    treatment_end date,
    treatment_start date,
    medical_book_id bigint,
    primary key (id)
) engine=MyISAM;

CREATE TABLE doctor (
    diploma_number bigint not null,
    password varchar(64) not null,
    full_name varchar(64) not null,
    access_during_working_period varchar(64),
    hash_code_password varchar(64),
    specialization_name varchar(64) NOT NULL,
    medical_institution_id BIGINT NOT NULL,
    PRIMARY KEY (diploma_number)
) engine=MyISAM;

create table medical_book (
    id bigint not null auto_increment,
    rh_factor bit not null,
    allergy varchar(255),
    graft_certificate varchar(255),
    group_blood integer not null,
    patient_id bigint,
    primary key (id)
) engine=MyISAM;

CREATE TABLE medical_institution (
    id BIGINT not null AUTO_INCREMENT,
    address varchar(255) not null,
    name varchar(255),
    PRIMARY KEY (id)
) engine=MyISAM;

create table patient (
    id bigint not null auto_increment,
    full_name varchar(255) not null,
    passport_data varchar(16) not null,
    snils varchar(16) not null.
```

SQL файл для заполнения данными (insert_data.sql):

В этом SQL файле мы определили запросы для вставки начальных данных в таблицы. Этот файл содержит SQL запросы INSERT для добавления фиктивных данных, которые мы можем использовать для тестирования или демонстрации функциональности нашего приложения. Примерно так может выглядеть содержимое файла:

```
INSERT INTO doctor (diploma_number, medical_institution_id, password, full_name, specialization_name)
VALUES(1234, 1, '1234','test doctor','Регистратор'),
      (2, 2, 'password','Смирнов Марк Романович','Фтизиатр'),
      (3, 3, 'password','Козлова Амина Ивановна','Ревматолог');

INSERT INTO specialization (name)
VALUES('Регистратор'),
      ('Фтизиатр'),
      ('Ревматолог'),
      ('Медсестра (медбрат)'),
      ('ЛОР (отоларинголог)'),
      ('Травматолог (Травматолог-ортопед)');

INSERT INTO medical_institution (id,name,address)
VALUES(1, 'МедПроф', 'Новая улица, 9, Люберцы, Московская область, 140002'),
      (2, 'Институт здоровья', 'Комсомольский проспект, 11Б, Люберцы, Московская область, 140013'),
      (3, 'Ниармедик', 'проспект Маршала Жукова, 38к1, Москва');

INSERT INTO patient (full_name,passport_data,snils,medical_book_id)
VALUES('Филатова Виктория Давидовна', 123, 123,1),
      ('Тихонов Николай Данилович', 321, 321,2),
      ('Степанова Милана Матвеевна', 456, 456,3),
      ('Денисова Элина Львовна', 654,654,4);
```

4.1 Создание Docker образов для каждого сервиса

Мы подошли к одной из самых тяжелых задач - работе с Docker. Для того, чтобы поместить наши сервисы в контейнеры, мы использовали инструкции Dockerfile. К сожалению, на момент написания я не смог найти готовых решений, поэтому пришлось создавать свои собственные Dockerfile. Это представляло собой непростую задачу, как показала практика.

Вот примеры наших Dockerfile для сервисов на базе Spring Boot и React:

```
FROM maven:3.9.5-sapmachine-21
WORKDIR /app
COPY mvnw pom.xml ./
COPY src ./src
RUN mvn dependency:resolve
RUN mvn install
CMD mvn spring-boot:run
```

Этот Dockerfile начинается с базового образа Maven, который используется для сборки проекта. Мы копируем файлы mvnw и pom.xml для установки зависимостей, а затем копируем директорию src с исходным кодом. Затем происходит установка зависимостей и сборка проекта с помощью Maven. В конце мы указываем команду для запуска Spring Boot приложения.

```
FROM node:15.13-alpine
WORKDIR /app
COPY . .
RUN npm install
RUN npm run build
CMD ["npm", "start"]
```

Этот Dockerfile начинается с базового образа Node.js, который используется для сборки приложения React. Мы копируем все файлы из текущей директории в контейнер, устанавливаем зависимости с помощью npm, выполняем сборку приложения и указываем команду для запуска приложения.

Таким образом, создание Dockerfile для каждого из наших сервисов позволило нам упаковать их в контейнеры, обеспечивая изолированное и надежное выполнение приложений. Этот процесс представляет собой важную часть нашей работы над проектом, облегчая развертывание и управление нашим приложением.

4.2 Использование Docker-compose

Переходя к использованию Docker-Compose, мы начали использовать внутренние сети, создаваемые самим Docker-Compose для каждого сервиса. Это исключает понятие localhost для обращения к сервисам, вместо этого мы обращаемся к другим сервисам по их именам.

Примерно так выглядит наш файл docker-compose.yml, где мы определили наши сервисы и их конфигурации:

```
version: '3.4'

services:
  backend:
    image: api
    restart: always
    ports:
      - "8080:8080"
    build:
      context: ./api
      dockerfile: Dockerfile

  client:
    image: client
    ports:
      - "5001:3000"
    build:
      context: ./medicalclient
      dockerfile: Dockerfile
    environment:
      - WDS_SOCKET_PORT=0

  db:
    image: mysql:8
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ROOT_PASSWORD=1234
    volumes:
      - ./dbdata:/var/lib/mysql/

  proxy:
    image: nginx:mainline
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    ports:
      - "80:80"
```

Сервис backend:

Мы создаем сервис backend, используя образ api, который собирается из Dockerfile в директории ./api.

Определяем перенаправление порта 8080, чтобы обращаться к нашему API на localhost:8080.

Сервис client:

Создаем сервис client, используя образ client, который собирается из Dockerfile в директории ./medicalclient.

Определяем перенаправление порта 5001, чтобы обращаться к нашему клиентскому приложению на localhost:5001.

Сервис db:

Создаем сервис db, используя образ mysql:8.

Определяем перенаправление порта 3306, чтобы обращаться к базе данных на localhost:3306.

Задаем переменную окружения для пароля MYSQL_ROOT_PASSWORD и определяем том для сохранения данных базы в ./dbdata.

Сервис проху:

Создаем сервис проху, используя образ nginx:mainline.

Определяем перенаправление порта 80, чтобы обращаться к нашему прокси серверу на localhost:80.

Монтируем конфигурационный файл nginx.conf для настройки прокси сервера.

Таким образом, использование Docker-Compose позволяет нам легко управлять нашими сервисами, создавать внутренние сети для их взаимодействия и запускать

все приложение с помощью одной команды. Это значительно упрощает процесс разработки, развертывания и масштабирования нашего приложения.

4.3 Сокращение времени обновления

Когда стоит вопрос об обновлении сервиса в контейнере, так чтобы пользователи не замечали неполадок и не ждали длительных перезапусков, это становится одной из главных задач. Долгое время я провел в поиске решения этой проблемы, и часто встречал рекомендации использовать Kubernetes (K8S). Хотя я признаю его мощь и гибкость, мой проект не требовал столь обширной системы, и я продолжил искать другие решения.

Наконец, я обнаружил, что Docker предоставляет некоторые инструменты для снижения времени перезапуска сервисов. Если контейнер уже запущен, мы можем спокойно собирать новый образ с помощью `docker build`, а затем использовать команду `docker-compose up -d`. Это позволяет Docker начать перезапуск только тех контейнеров, для которых изменился хеш ключ нового образа. Этот подход позволяет нам обновлять наши сервисы по мере необходимости, минимизируя простой и временные затраты. Пользователи не замечают сбоев в работе приложения, поскольку новые версии сервисов запускаются по мере готовности и заменяют старые только в том случае, если произошли изменения.

Таким образом, использование комбинации Docker и Docker-Compose позволяет нам эффективно управлять обновлениями наших сервисов, делая перезапуск приложения быстрым и незаметным для конечных пользователей.

5.1 Использование GitHub Project

Мы подошли к одной из ключевых составляющих успешной работы в команде - использованию GitHub Project. Этот инструмент можно сравнить с идеальным Task Manager'ом для управления задачами в нашем проекте, учитывая репозитории, с которыми мы работаем. GitHub Project предоставляет нам возможность эффективно организовать работу, отслеживать задачи и управлять проектом в целом.

Мы можем создавать задачи и проблемы (issues) прямо в GitHub Project для каждой части проекта. Используя язык разметки Markdown, мы добавляем пояснения, описания и дополнительную информацию к каждой задаче, что делает их более понятными и структурированными.

Кроме того, мы открываем новые задачи при необходимости и закрываем их по мере выполнения, что позволяет нам легко отслеживать прогресс и состояние проекта. Устанавливаем уровни сложности и приоритеты для задач, чтобы определить, какие из них требуют больше внимания и приоритетного решения.

С помощью GitHub Project мы можем устанавливать дедлайны для задач, что помогает нам следить за временем выполнения и укладываться в график проекта. Мы фильтруем задачи по различным параметрам, таким как статус, приоритет, метки и другие, что делает поиск нужной информации быстрым и удобным.

Одним из самых интересных аспектов GitHub Project является возможность построения графиков и отчетов по выполненным задачам, прогрессу проекта, времени, затраченному на каждую задачу и многое другое. Это помогает нам анализировать эффективность работы команды и делать выводы для улучшения процессов.

Таким образом, GitHub Project стал неотъемлемой частью нашей работы в команде. Мы используем его для эффективной организации работы, управления задачами и проектом в целом, а также для анализа результатов и постоянного совершенствования нашей командной работы.

5.2 Использование NGINX proxy

Применение Proxy NGINX в нашем проекте стало необходимостью по нескольким причинам. Во-первых, основной причиной использования была проблема с политикой CORS (Cross-Origin Resource Sharing). NGINX позволяет нам обойти эти ограничения и обеспечить безопасное взаимодействие между различными доменами и источниками данных.

Однако, помимо этого, NGINX также помогает нам скрыть реальные порты и IP-адреса наших сервисов. Это делает наш проект более безопасным и обеспечивает дополнительный уровень защиты от несанкционированного доступа к нашим сервисам.

Таким образом, использование Proxy NGINX в нашем проекте не только решает проблему с политикой CORS, но и обеспечивает дополнительный слой безопасности, скрывая реальные порты и IP-адреса наших сервисов от внешнего мира.

```
worker_processes 4;

events { worker_connections 1024; }

http{
    server{
        listen 80;

        location / {
            proxy_pass http://client:3000;
        }

        location /ws {
            proxy_pass http://client:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
        }

        location /api/ {
            proxy_pass http://backend:8080/api/;
        }
    }
}
```

6.1 Тестирование

Пора перейти к тестированию нашего разработанного приложения. Начнем с основного окна, которое встречает пользователя при запуске приложения. Затем мы углубимся в логику работы, начиная с процесса входа или регистрации.

Поскольку у нас нет доступа к реальным государственным службам для получения настоящих данных, мы решили имитировать процесс проверки при регистрации. При регистрации пользователю требуется указать следующие данные: имя, номер диплома, город и желаемую специальность.

Одним из наших главных преимуществ является то, что у нас используется единая база данных для всех больниц. Это позволяет нам избежать необходимости привязываться к конкретной базе данных каждый раз при регистрации в разных медицинских учреждениях. Вместо этого мы можем запросить необходимую информацию по вашим данным из единой базы данных и даже записать вас на прием к врачу.

6.2 Вход и регистрация

На стартовой странице сайта вас приветствует возможность войти или зарегистрироваться. Для доступа к личному кабинету врача необходимо ввести уникальный номер диплома и соответствующий пароль. При желании создать новую учетную запись, потребуется указать номер диплома, полное имя, местоположение и предпочтительную специализацию.

После успешной регистрации вы попадаете на персонализированную страницу врача, однако в начальный момент ваш профиль останется пустым, готовым к наполнению информацией. Здесь вы сможете добавить свои квалификации,

The image shows a login form titled "Вход" (Login) centered on a light gray background. The form is a white rounded rectangle. It contains two input fields: "Номер диплома:" (Diploma number) and "Пароль:" (Password). Below the password field is a green button with the text "Войти" (Login). At the bottom of the form is a link that says "подать заявку." (submit application).

специализации, опыт работы, а также информацию о вашей практике и медицинские достижения.

ФИО:

Диплом:

Пароль:

Специализация:

Специализация:

ЛОР (отоларинголог)



Город:

Больница:

(МедПроф) Новая улица, 9, Люберцы, Московская



Отправить заявку

6.3 Регистратура

При выборе специальности 'Регистратура' нашего веб-приложения открывается удобное меню, предоставляющее широкий спектр возможностей для эффективной работы с пациентами и коллегами.


Первая опция меню - 'Запись пациента' - позволяет врачам быстро и просто создавать записи о приемах и обследованиях пациентов. Здесь можно указать дату и время приема, специализацию врача, тип обследования или процедуры, а также основные данные о пациенте. Это помогает поддерживать актуальную информацию о визитах и обеспечивать пациентам качественный сервис.


Запись к врачу

Филатова Виктория Давидовна


Поиск

Дата записи:





Выберите врача:



Комментарии:

Записать

Вторая опция - 'Список докторов из нашей больницы' - предоставляет доступ к полному перечню специалистов, работающих в нашей медицинской организации. Здесь вы можете найти контактные данные, специализации, расписание приемов и другую важную информацию о врачах. Это упрощает процесс назначения приемов, обращения к конкретному специалисту и получения необходимой консультации.

Faclok Faclok Faclok
Фтизиатр

суббота, 2 марта 2024 г. в 15:00
Пациент: Филатова Виктория Давидовна
Примечания: Болит ухо, время от времени идет кровь

Faclok Faclok Faclok
Фтизиатр

Удалить

Отмена

Третья опция - 'Поиск пациента по всему миру' - представляет собой уникальную функцию нашего приложения, которая позволяет быстро найти информацию о пациенте в базе данных медицинских учреждений по всему миру. Это особенно удобно в случаях экстренных ситуаций или необходимости доступа к истории болезни пациента, если он обратился в другой медицинский центр.

Создать заявку Список докторов Поиск пациента

Паспорт
123

СНИЛС
123

Поиск

123

123

История посещений

Мед. книжка

6.4 Врачи

Для врачей нашего веб-приложения доступны ключевые функции, позволяющие эффективно управлять приемами пациентов и обеспечивать высокий уровень медицинского обслуживания.

Первая вкладка - 'Список будущих записей' - предоставляет врачам удобный обзор предстоящих приемов. Здесь врач может увидеть расписание своих будущих консультаций, обследований или процедур. Это помогает грамотно планировать рабочее время, подготавливать необходимое оборудование и материалы к приемам.

суббота, 2 марта 2024 г. в 15:00
Пациент: Филатова Виктория Давидовна
Примечания: Болит ухо, время от времени идет кровь

Принять пациента Мед. книжка Отменить

Прививочный сертификат
121dfa23

Группа крови
1

Резус фактор
Положительная

Аллергия

Диагнозы

06.01.2024 - 27.01.2024
Диагноз: Пневмония, вызванная Str. pneumoniae, с локализацией в S 9,10 правого легкого, нетяжелая. ДН I
Рецептура: receptura

Просмотреть процедуры Удалить диагноз

06.01.2024 - 27.01.2024
Диагноз: Пневмония, вызванная Str. pneumoniae, с локализацией в S 9,10 правого легкого, нетяжелая. ДН I
Рецептура: receptura

Просмотреть процедуры Удалить диагноз

Добавить диагноз Закрыть

Вторая вкладка - 'История записей' - представляет собой ценный инструмент для врачей, позволяющий просматривать и анализировать историю всех предыдущих визитов пациентов. Здесь врач может получить доступ к информации о проведенных обследованиях, лечении, выписанных рецептах и рекомендациях. Это помогает врачу лучше понять медицинскую историю пациента и принимать более обоснованные решения.

Третья вкладка - 'Список всех пациентов' - предоставляет полный перечень всех пациентов, обратившихся к врачу в течение определенного периода времени. Здесь врач может видеть информацию о каждом пациенте, включая количество посещений и общую динамику здоровья. Это помогает врачу отслеживать состояние здоровья пациентов на протяжении времени и предоставлять индивидуализированное лечение.

Пациент: Филатова Виктория Давидовна

Кол-во: 1

Мед. книжка

Кроме того, в любой из этих вкладок врач может просматривать медицинские данные пациентов, такие как медицинская книжка с записями, информация об аллергиях, противопоказаниях и другие важные медицинские аспекты. Это делает наше веб-приложение уникальным инструментом для врачей, обеспечивая полный и структурированный доступ к информации о пациентах и их медицинской истории.

Паспорт

Прививочный сертификат
121dfa23

Группа крови
1

Резус фактор
Положительная

Аллергия

Диагнозы

06.01.2024 - 27.01.2024
Диагноз: Пневмония, вызванная Str. pneumoniae, с локализацией в S 9,10 правого легкого, нетяжелая. ДН I
Рецептура: rezeptura

Просмотреть процедуры

Удалить диагноз

06.01.2024 - 27.01.2024
Диагноз: Пневмония, вызванная Str. pneumoniae, с локализацией в S 9,10 правого легкого, нетяжелая. ДН I
Рецептура: rezeptura

Просмотреть процедуры

Удалить диагноз

Добавить диагноз

Закрыть

Заключение

В ходе данного дипломного проекта мы занялись разработкой и реализацией медицинского веб-приложения под названием MedicalHubApi. Этот проект был осуществлен с использованием современных инструментов и технологий, таких как Spring Boot, React, Docker, Docker-Compose, GitHub Project и NGINX Proxy.

Мы начали с создания конфигурационного файла на официальном сайте Spring, определив параметры использования Java с версией JDK 21 и сборку проекта с помощью Maven. Наш проект был назван MedicalHubApi, отражая его функциональное предназначение.

Для удобства организации файлов и структуры проекта мы категоризировали их, создавая папки для Controller, Entity, Model, Exception, Repository, и Service в корневой директории src.

Подключение к базе данных было осуществлено через файл application.properties, где мы указали путь к базе данных, а также учетные данные пользователя.

Далее мы перешли к использованию JPA и репозитория, создав сервисы для логики и работы с данными. Этот шаг позволил нам эффективно взаимодействовать с базой данных и обрабатывать запросы от клиентской стороны.

Создание моделей базы данных было совершено с помощью инструментов, таких как FreeMind для проектирования и Draw.io для визуализации структуры самих моделей. Это помогло нам четко представить требования проекта и его функциональность.

Для инициализации базы данных мы использовали SQL-файлы, один для создания таблиц и другой с конкретными данными. Это позволило нам эффективно начать работу с базой данных, обеспечивая необходимые структуры и информацию.

Одним из самых сложных и важных моментов в проекте стала контейнеризация с помощью Docker и написание своих Dockerfile. Мы создали Dockerfile для каждого сервиса, что позволило нам упаковать их в контейнеры, обеспечивая изолированное

и надежное выполнение приложений. Также мы использовали Docker-Compose для создания внутренних сетей и управления нашими сервисами.

Для управления задачами и организации работы в команде мы воспользовались GitHub Project. Этот инструмент стал идеальным Task Manager'ом, позволяя нам создавать, отслеживать, устанавливать приоритеты и дедлайны для задач, а также анализировать прогресс проекта через графики и отчеты.

И, наконец, использование NGINX Proxy в нашем проекте не только решило проблему с политикой CORS, но и обеспечило безопасность проекта, скрывая реальные порты и IP-адреса наших сервисов.

Данный дипломный проект позволил нам не только применить полученные знания и навыки в разработке программного обеспечения, но и лучше понять важность выбора правильных инструментов и методов при создании сложных веб-приложений.

Мы изучили и применили современные подходы к разработке, такие как использование Spring Boot для бэкэнд-разработки, React для фронтэнд-разработки, Docker для контейнеризации, GitHub Project для управления задачами и NGINX Proxy для обеспечения безопасности.

Этот проект подтвердил, что грамотно выбранные инструменты и методы помогают создать надежное, безопасное и масштабируемое веб-приложение. Мы приобрели ценный опыт в разработке и управлении проектами, который будет полезен в нашей дальнейшей карьере в сфере информационных технологий.

Список используемой литературы

Книги:

"Spring Boot in Action" by Craig Walls - Основы работы с Spring Boot, примеры использования и передовые практики.

"Pro Spring Boot 2" by Felipe Gutierrez - Расширенное руководство по Spring Boot, обзор основных функций и возможностей.

"Learning React: Modern Patterns for Developing React Apps" by Alex Banks and Eve Porcello - Основы работы с React, современные паттерны и лучшие практики.

"Pro React" by Cassio de Sousa Antonio - Расширенное руководство по разработке с использованием React, погружение в различные аспекты фронтенд-разработки.

Онлайн-ресурсы:

Официальная документация Spring Boot - Исчерпывающие руководства и примеры по использованию Spring Boot.

Официальная документация React - Полезные материалы, учебные пособия и примеры кода по React.

Baeldung - Популярный блог с множеством статей о Spring Boot и связанных технологиях.

React Training - Обучающие материалы и ресурсы для работы с React.

Журналы и статьи:

Johnson, C., & Hoeller, J. (2017). "Building Modern Web Applications with Spring Boot and React". Java Magazine, 21(6), 24-31.

Gupta, P., & Agrawal, A. (2018). "Enhancing Web Development with Spring Boot and React.js". Journal of Web Engineering, 17(2&3), 137-156.

Приложения