МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

КУРСОВАЯ РАБОТА

РАЗРАБОТКА СИСТЕМЫ ПОИСКА СПЕЦИАЛИСТА НА ОСНОВАНИИ СИМПТОМОВ БОЛЕЗНИ

Работу выполнил _____ Б.А. Михаенко

(подпись)

Направление подготовки 09.03.03 Прикладная информатика

Направленность Прикладная информатика в экономике

Научный руководитель
ст. преп. _____ А.В. Уварова

(подпись)

Нормоконтролер
канд. пед. наук, доц. _____ А.В. Харченко

(подпись)

Краснодар
2023

РЕФЕРАТ

Курсовая работа 28 с., 19 рис., 8 таб., 7 источников.

API, NODE.JS, EXPRESS.JS, REACT, RTK QUERY, AXIOS, БАЗЫ ДАННЫХ, POSTGRESQL, ВЕБ-ПРИЛОЖЕНИЕ.

Объектом исследования являются разработка собственного API для веб-ресурса классификации заболеваний.

Цель работы: изучение основных технологий разработки API.

В курсовой работе рассмотрены основы кластерного анализа, метрики и методы разработки веб-приложения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сегодня многие люди обращаются к интернету для поиска информации о своем здоровье. Однако, часто бывает трудно определить, каким заболеванием вы страдаете, основываясь только на симптомах. В этом контексте, создание веб-ресурса, который предоставляет быстрый и точный диагноз на основе симптомов, может быть крайне полезным.

Забота о здоровье и поиск способов сохранения его – это всегда актуальная тема. В современном мире люди сталкиваются с большим количеством заболеваний, и нередко становится трудно определить, что именно вызвало симптомы и каким заболеванием они соответствуют. В этой ситуации важным ресурсом становится информация о заболеваниях и их симптомах.

API (Application Programming Interface) — это набор инструментов и правил, которые позволяют разным программам взаимодействовать друг с другом. API позволяет программам использовать функциональность других программ без необходимости знать, как эти программы работают внутри.

Данная курсовая работа посвящена разработке собственного API для веб-ресурса классификации заболеваний по симптомам. API представляет собой набор программных интерфейсов, которые позволяют взаимодействовать с веб-ресурсом и получать от него информацию.

В случае нашего веб-ресурса API позволяет пользователям взаимодействовать с базой данных, содержащей информацию о заболеваниях и их симптомах.

Первая глава курсовой работы содержит описание основных инструментов разработки API и ее структуру.

Вторая глава посвящена реализации собственного API.

Третья глава содержит описание взаимодействия пользователя с разработанным API.

1 Сравнительный анализ основных технологий API

API (Application Programming Interface) — это интерфейс, который позволяет программам взаимодействовать между собой. Он представляет собой набор правил и структур данных, которые определяют, как программы могут общаться друг с другом и использовать функциональность других приложений.

API используются для обмена данными между различными программами и платформами, что позволяет разработчикам создавать более сложные и функциональные приложения. Они также помогают ускорить процесс разработки.

1.1 RESTful API

RESTful API (Representational State Transfer) — это стиль архитектуры программного обеспечения, который используется для создания веб-сервисов. Он базируется на протоколе HTTP и представляет собой набор правил и ограничений, которые определяют, как клиентские приложения могут взаимодействовать с сервером.

RESTful API использует стандартные методы HTTP-запросов, такие как GET, POST, PUT и DELETE, чтобы управлять ресурсами на сервере. Он использует URL-адреса для идентификации ресурсов и форматы данных, такие как JSON или XML, для обмена информацией между клиентом и сервером.

CRUD — это акроним, который описывает четыре основных операции, которые могут выполняться с данными: Create (создание), Read (чтение), Update (обновление) и Delete (удаление). RESTful API использует эти операции для управления ресурсами на сервере.

HTTP-запросы — это способ, с помощью которого клиентские приложения взаимодействуют с веб-сервисами. Они используют стандартные

методы HTTP, такие как GET, POST, PUT и DELETE, чтобы выполнить определенные действия с ресурсами на сервере.

Метод GET используется для получения данных с сервера. Он не изменяет данные на сервере и используется для чтения информации.

Метод POST используется для создания новых ресурсов на сервере. Он отправляет данные на сервер для сохранения.

Метод PUT используется для обновления существующих ресурсов на сервере. Он отправляет данные на сервер для обновления информации.

Метод DELETE используется для удаления существующих ресурсов на сервере. Он отправляет запрос на сервер для удаления информации.

RESTful API использует эти методы для обработки запросов клиентских приложений и управления ресурсами на сервере. Он также предоставляет механизмы для работы с авторизацией, обработки ошибок и другими аспектами, связанными с веб-сервисами.

RESTful API позволяет упростить разработку и интеграцию различных приложений и сервисов, поскольку использует стандартные протоколы и методы взаимодействия. Кроме того, RESTful API обеспечивает высокую производительность и масштабируемость, что позволяет ему обрабатывать большие объемы запросов от клиентов.

1.2 GraphQL

GraphQL это язык запросов и средство разработки API, который был разработан Facebook. Он обеспечивает более гибкий, эффективный и интуитивно понятный способ обмена данными между клиентом и сервером по сравнению с RESTful API.

Основное преимущество GraphQL заключается в том, что он позволяет клиентам запрашивать только необходимые данные и получать ответы только на эти запросы. Это значит, что клиенты не получают избыточных данных и не тратят время и ресурсы на обработку лишних данных. Таким образом,

GraphQL обеспечивает более быстрый и эффективный обмен данными между клиентом и сервером.

GraphQL также обеспечивает более гибкий и расширяемый способ разработки API. Он позволяет разработчикам быстро и легко изменять и добавлять новые функции в API без необходимости изменения клиентского кода.

Однако, GraphQL также имеет свои ограничения. Он не является подходящим выбором для приложений, которые требуют передачи больших объемов статических данных или имеют высокий трафик. Также, GraphQL требует более тщательного проектирования и разработки, поскольку он позволяет клиентам запрашивать любые данные, что может повлечь за собой определенные проблемы без должной осторожности в проектировании API.

В целом, GraphQL является отличным выбором для приложений, которые требуют гибкости и эффективности в обмене данными между клиентом и сервером, и которые нуждаются в быстрой и легкой масштабируемости.

1.3 WebSocket

WebSocket является протоколом связи, который позволяет установить двустороннее соединение между клиентом и сервером в режиме реального времени. В отличие от RESTful API, который работает с HTTP-запросами и ориентирован на передачу статических данных, WebSocket поддерживает динамические потоковые данные, такие как чаты, игровые события и другие вещи, которые требуют быстрого и реального времени обмена данными между клиентом и сервером.

Основное преимущество WebSocket заключается в том, что он обеспечивает более быстрый и эффективный способ обмена данными в режиме реального времени по сравнению с RESTful API. Он использует меньше ресурсов, потому что он устанавливает постоянное соединение между

клиентом и сервером, и не нуждается в повторном установлении соединения при каждом запросе, как это делает RESTful API.

WebSocket также обеспечивает более высокую производительность в ситуациях, когда много клиентов подключены к серверу одновременно и передают данные в режиме реального времени. Это может быть особенно важным для приложений, таких как онлайн-игры, чаты и трансляции медиа контента.

Однако, WebSocket имеет свои ограничения, он не может использоваться для передачи больших объемов статических данных, которые обычно передаются через RESTful API. Также, WebSocket требует дополнительной настройки сервера и клиента, поэтому его использование может потребовать больше времени и усилий в разработке, чем простые RESTful API.

2 Описание реализации и технологий создания API

В рамках курсовой работы ставится задача разработки веб-приложения для установки диагноза болезни по выбранным симптомам основными функциями которого являются:

– расширение существующей базы данных,

– создание маршрутов API для обработки запросов клиента,

– создание скриптов заполнения базы данных и обновления данных,

– создание страниц администрирования БД.

2.1 Описание базы данных

Разработанное REST API взаимодействует с Базой Данных, используя определенные CRUD операции.

База данных была разработана с помощью СУБД PostgreSQL [3].

Она состоит из 8 таблиц (рисунок 2.1).

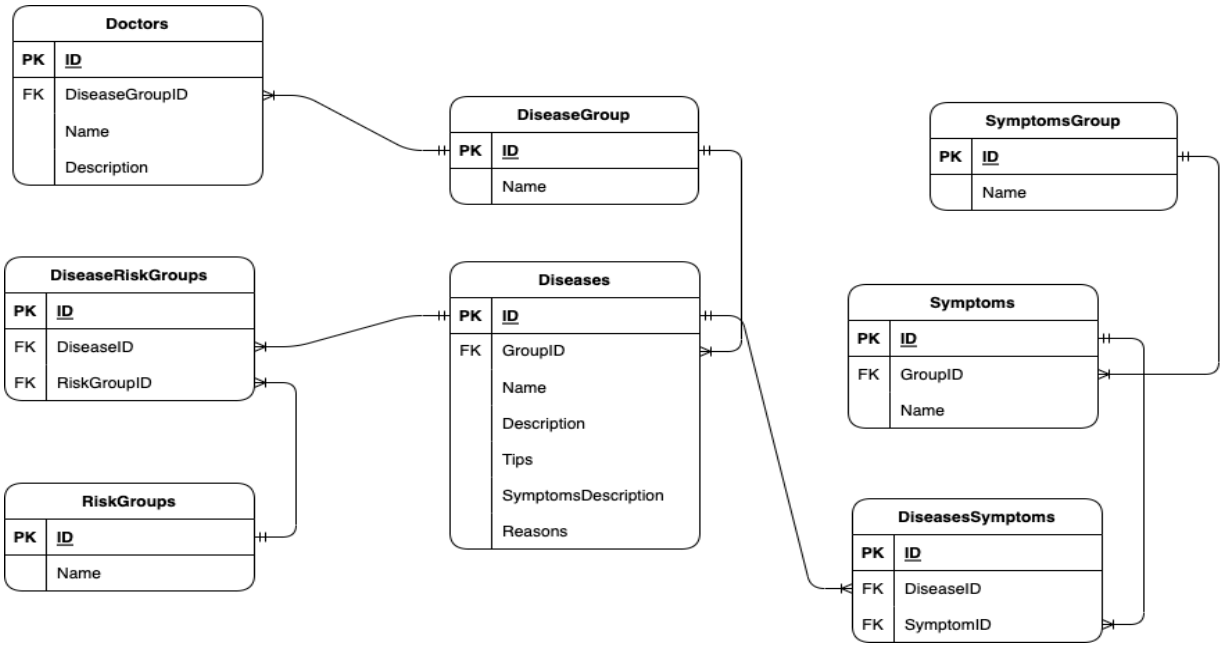

Рисунок 2.1 – Схема структуры базы данных

Таблица Diseases предназначена для хранения информации о существующих заболеваниях их наименование, описание, симптомы, группа, к которой относится заболевание, возможные причины возникновения, советы по лечению и описание появления и развития симптоматики. Её структура приведена в таблице 1. Данная таблица имеет связь Many-to-Many с таблицей Symptoms через промежуточную таблицу DiseasesSymptoms. А также связь Many-to-One с таблицей DiseaseGroup.

Таблица 1 - Структура таблицы Diseases

Наименование	Тип	Описание
ID	Primary Key	Уникальный идентификатор заболевания
GroupID	Foreign Key	Внешний ключ, ссылающийся на таблицу DiseaseGroup
Name	Unique String	Уникальное наименование заболевания
Description	Text	Описание заболевания
Tips	Text	Строка советов по лечению заболевания
Reasons	Text	Строка возможных причин возникновения заболевания
SymptomsDescription	Text	Описание появления и развития симптоматики заболевания

Таблица DiseaseGroup предназначена для хранения информации о группах заболеваний. Её структура представлена в таблице 2. Данная таблица имеет свзяь One-to-Many с таблицей Diseases устанавливая свой идентификатор группы, как внешний ключ для таблицы Diseases.

Таблица 2 – Структура таблицы DiseaseGroup

Наименование	Тип	Описание
ID	Primary Key	Уникальный идентификатор группы заболеваний
Name	Unique String	Уникальное наименование группы

Таблица Doctors предназначена для хранения информации о докторах их специализация и ее описание, а также уникальный номер группы заболеваний, в которой они специализируются. Её структура приведена в таблице 3. Данная таблица имеет связь Many-to-One с таблицей DiseaseGroup.

Таблица 3 – Структура таблицы Doctors

Наименование	Тип	Описание
ID	Primary Key	Уникальный идентификатор доктора
DiseaseGroupID	Foreign Key	Уникальный идентификатор группы заболеваний
Name	Unique String	Уникальное наименование специализации
Description	Text	Описание специализации

Таблица RiskGroups предназначена для хранения информации о группах риска их наименования. Её структура приведена в таблице 4. Данная таблица имеет связь Many-to-Many с таблицей Diseases через промежуточную таблицу DiseaseRiskGroups.

Таблица 4 – Структура таблицы RiskGroups

Наименование	Тип	Описание
ID	Primary Key	Уникальный идентификатор группы риска
Name	Unique String	Уникальное наименование группы риска

Таблица DiseaseRiskGroups представляет собой промежуточную таблицу, предназначенную для хранения пары ключей вида: уникальный идентификатор заболевания - уникальный идентификатор группы риска. Её структура представлена в таблице 5.

Таблица 5 – Структура таблицы DiseaseRiskGroups

Наименование	Тип	Описание
ID	Primary Key	Уникальный идентификатор записи
DiseaseID	Foreign Key	Уникальный идентификатор заболевания
RiskGroupID	Foreign Key	Уникальный идентификатор группы риска

Таблица Symptoms предназначена для хранения информации о симптомах заболеваний их наименование, а также уникальный номер группы, к которой они принадлежат. Её структура приведена в таблице 6. Данная таблица имеет связь Many-to-Many с таблицей Diseases через промежуточную таблицу DiseasesSymptoms, а также связь Many-to-One с таблицей SymptomsGroup, через внешний ключ GroupID.

Таблица 6 - Структура таблицы Symptoms

Наименование	Тип	Описание
ID	Primary Key	Уникальный идентификатор симптома
GroupID	Foreign Key	Уникальный идентификатор группы симптомов
Name	Unique String	Наименование симптома

Таблица SymptomsGroup предназначена для хранения информации о группах симптомов. Её структура представлена в таблице 7. Данная таблица имеет связь One-to-Many с таблицей Symptoms устанавливая свой уникальный идентификатор группы, как внешний ключ для таблицы Symptoms.

Таблица 7 - Структура таблицы SymptomsGroup

Наименование	Тип	Описание
ID	Primary Key	Уникальный идентификатор группы
Name	Unique String	Уникальное наименование группы

Таблица DiseaseSymptoms представляет собой промежуточную таблицу, предназначенную для хранения пары ключей вида: уникальный идентификатор заболевания - уникальный идентификатор симптома. Её структура представлена в таблице 8.

Таблица 8 - Структура таблицы DiseasesSymtptoms

Наименование	Тип	Описание
ID	Primary Key	Уникальный идентификатор пары
DiseaseID	Foreign Key	Уникальный внешний идентификатор заболевания
SymptomID	Foreign Key	Уникальный внешний идентификатор симптома

2.2 Создание маршрутов API для обработки запросов клиента

В данной главе рассмотрено создание маршрутов API, которые обрабатывают запросы клиента на сервере. Они позволяют клиентским приложениям получать доступ к данным и функциональности, которые предоставляются сервером. В качестве основы для создания маршрутов были использованы технологии Node.js и Express.js, которые позволяют создавать высокопроизводительные и масштабируемые приложения на сервере.

Node.js — это среда выполнения JavaScript, которая позволяет выполнять код на стороне сервера. Node.js обеспечивает высокую производительность благодаря использованию асинхронных операций ввода-вывода и многопоточности.

Express.js — это фреймворк для Node.js, который позволяет создавать мощные веб-приложения на сервере. Он предоставляет удобные инструменты для обработки запросов и ответов, маршрутизации запросов, управления состоянием сеанса и многое другое.

Далее кратко рассмотрим создание составной части основного маршрутизатора:

Импортируем фреймворк Express.js и модуль контролера, который позволяет обрабатывать HTTP-запросы, создаем экземпляр класса Router для возможности настройки маршрутизации (рисунок 2.2)

```
const express = require('express')
const diseaseController = require('../controllers/diseaseController')
const router = express.Router()
```

Рисунок 2.2 – Импорт фреймоворка, контроллера и создание экземпляра

После создания экземпляра класса Router, назначаем обработчики из импортированного контролера на HTTP-запросы по соответствующим путям

и экспортируем модуль для дальнейшего использования в основном маршрутизаторе (рисунок 2.3).

```
router.get( path: '/', diseaseController.all)
router.get( path: '/:id', diseaseController.one)
router.post( path: '/', diseaseController.create)
router.put( path: '/', diseaseController.update)
router.delete( path: '/:id', diseaseController.delete)


module.exports = router
```

Рисунок 2.3 – Назначение обработчиков и экспорт модуля

Далее кратко рассмотрим создание основного маршрутизатора:

Импортируем фреймворк Express.js, а также модули, отвечающие за составные части маршрутизатора, создаем экземпляра класса Router для дальнейшей настройки маршрутизации (рисунок 2.4).

```
const express = require('express');
const router = express.Router();
const symptomsRouter = require('./symptomsRouter');
const diseasesRouter = require('./diseaseRouter');
const diagnoseRouter = require('./diagnoseRouter');
const doctorsRouter = require('./doctorsRouter');
const groupRouter = require('./groupRouter');
```

Рисунок 2.4 – Импорт фреймоворка, модулей и создание экземпляра

После подключения всех необходимых составляющих, с помощью встроенной в Express.js функции use устанавливаем промежуточные обработчики по соответствующему пути и экспортируем модуль (рисунок 2.5).

```
router.use('/symptoms', symptomsRouter);
router.use('/diseases', diseasesRouter);
router.use('/diagnose', diagnoseRouter);
router.use('/doctors', doctorsRouter);
router.use('/groups', groupRouter);

module.exports = router
```

Рисунок 2.5 – Установка обработчиков и экспорт основного модуля

2.3 Описание технологий взаимодействия клиента с API

Для реализации взаимодействия клиентской части с созданным API использовались следующие технологии:

React – это библиотека JavaScript с открытым кодом для создания внешних пользовательских интерфейсов. В отличие от других библиотек JavaScript, предоставляющих полноценную платформу приложений, React ориентируется исключительно на создание представлений приложений через инкапсулированные единицы (называются компонентами), которые сохраняют состояние и генерируют элементы пользовательского интерфейса.

Redux – библиотека для JavaScript с открытым исходным кодом, предназначенная для управления состоянием приложения. Чаще всего используется в связке с React или Angular для разработки клиентской части. Содержит ряд инструментов, позволяющих значительно упростить передачу данных хранилища через контекст.

RTK Query (или Redux Toolkit Query) – это мощный инструмент выборки и кэширования данных. Он предназначен для упрощения распространенных случаев загрузки данных в веб-приложение, устраняя необходимость самостоятельного написания логики выборки и кэширования данных [7].

TypeScript – язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript.

TypeScript является обратно совместимым с JavaScript и компилируется в последний. Фактически, после компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой Node.js. Код экспериментального компилятора, транслирующего TypeScript в JavaScript, распространяется под лицензией Apache. Его разработка ведётся в публичном репозитории через сервис GitHub

Схема взаимодействия между веб-приложением, сервером и БД (рисунок 2.6).

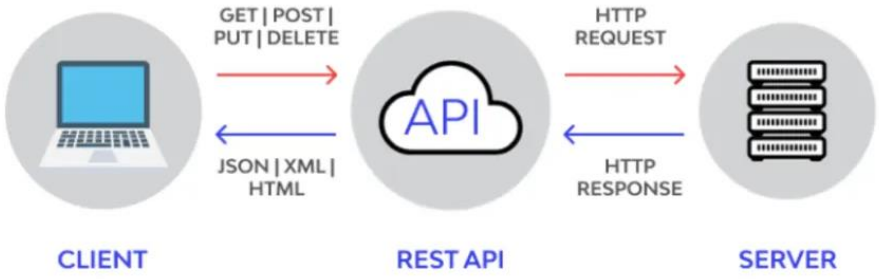

Рисунок 2.6 – Схема взаимодействия между клиентом, сервером и API

Для отправки запросов с клиентской части использовался RTK Query [7].

Рассмотрим пример некоторых запросов на сервер:

```
export const symptomsApi = createApi({
    reducerPath: 'symptoms',
    baseQuery: fetchBaseQuery({ baseUrl:
`${process.env.REACT_APP_HOST_LINK}` }),
    tagTypes: ['SYMPTOM', 'GROUP', 'DISEASE'],
    endpoints: (builder) => ({
        getSymptoms: builder.query<Symptom[], void>({
            query: () => '/symptoms',
            providesTags: ['SYMPTOM']
        }),
        addSymptom: builder.mutation<Symptom, SymptomPreview>({
            query: (symptom) => ({
                url: '/symptoms',
```

```
                method: 'POST',
                body: symptom,
            }),
            invalidatesTags: ['SYMPTOM']})
```

`export const symptomsApi = createApi` – инициализация и экспорт переменной

`reducerPath: 'symptoms'` – название редуктора

`baseQuery: fetchBaseQuery({ baseUrl: `${process.env.REACT_APP_HOST_LINK}` })` – инициализация основного пути для создания запроса

`tagTypes: ['SYMPTOM', 'GROUP', 'DISEASE']` – тэги для отслеживания изменений

```
endpoints: (builder) => ({
        getSymptoms: builder.query<Symptom[], void>({
            query: () => '/symptoms',
            providesTags: ['SYMPTOM']
        }),
        addSymptom: builder.mutation<Symptom, SymptomPreview>({
            query: (symptom) => ({
                url: '/symptoms',
                method: 'POST',
                body: symptom,
            }),
            invalidatesTags: ['SYMPTOM']}) – создание конкретных
```

типов запросов используя объект builder и его встроенную функцию query.

3. Описание взаимодействия клиентов с API

Глава разбита на две версии: версию для администратора и версию для клиента. Версия для администратора предоставляет описание процессов добавления, обновления и удаления данных. Версия для клиента фокусируется на использовании API для получения данных, отправки запросов и обработки ответов.

3.2 Описание возможностей администрирования

При запуске приложения, администратора встречает главная страница администрирования, содержащая меню для возможности перехода по страницам редактирования или добавления Основных Моделей или Моделей Групп (рисунок 3.1).

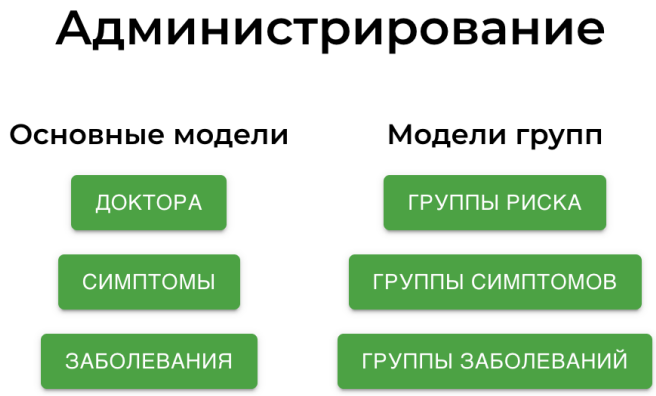

Рисунок 3.1 – Главная страница администрирования

При нажатии на кнопку Симптомы администратора перенаправит на страницу добавления или редактирования Симптомов (рисунок 3.2).

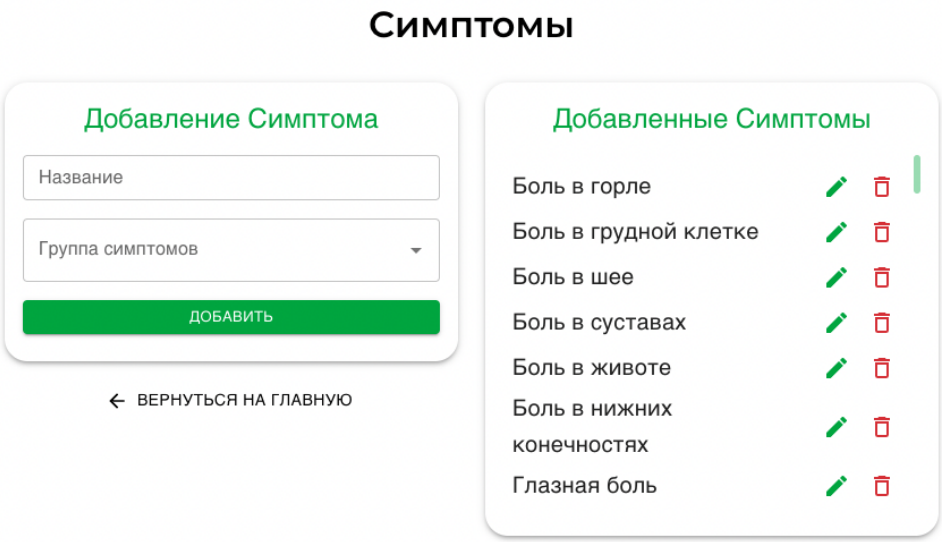

Рисунок 3.2 – Страница администрирования симптомов

При нажатии на компонент Группа симптомов откроется выпадающий список с существующими в базе данных группа симптомов, с возможностью выбора и поиска конкретной группы (рисунок 3.3).

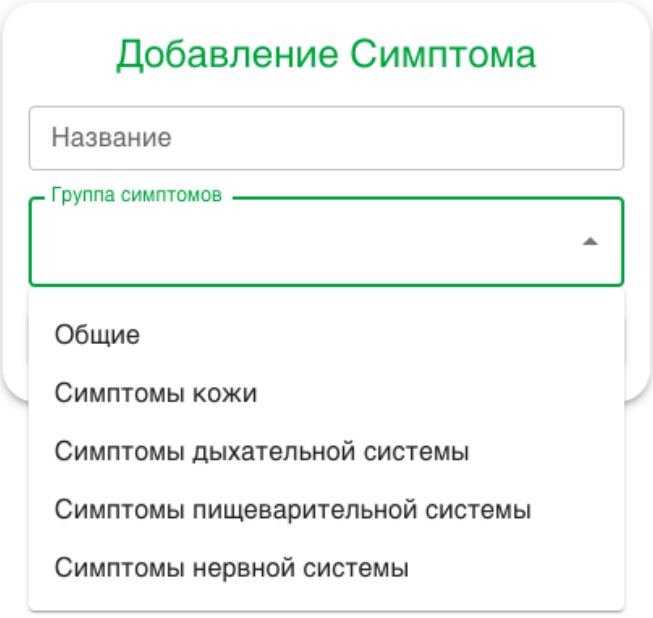

Рисунок 3.3 – Список существующих в БД симптомов

Для того чтобы выбрать нужную группу симптомов необходимо нажать на подходящую. Выбранная группа появится в строке поиска, а выпадающий список будет закрыт (рисунок 3.4).

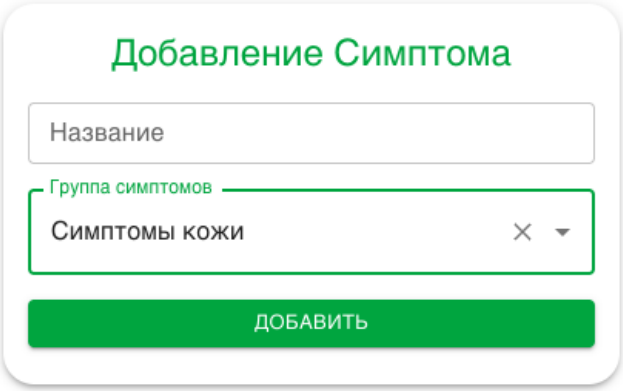

Рисунок 3.4 – Выбранные симптомы

Также администратору доступны возможности редактирования конкретного из симптомов или его удаления. Для этого в компоненте Добавленные Симптомы у каждого из симптомов присутствую две кнопки действий, редактировать и удалить соответственно (рисунок 3.5).

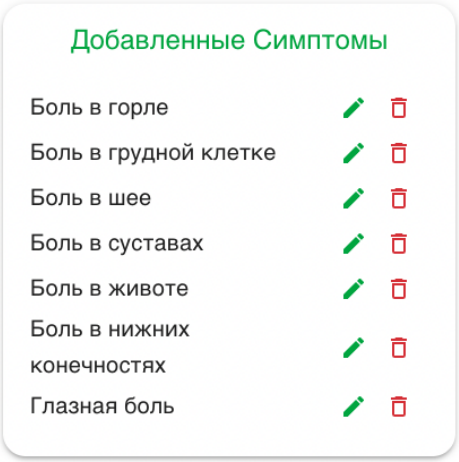

Рисунок 3.5 – Доступные действия с добавленными данным

При нажатии на кнопку редактирования (знак карандаша) в Компонент Добавления Симптома измениться на Изменение Симптома, при этом

появиться кнопка в виде креста рядом с заголовком компонента, позволяющая досрочно завершить изменение данных. В компонентах Название и Группа Симптомов появятся данные о выбранном для редактирования Симптоме (рисунок 3.6).

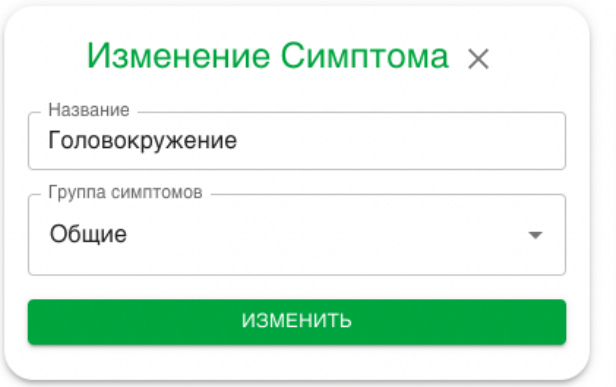

Рисунок 3.6 – Редактируемые данные

При нажатии кнопки изменить данные для выбранного симптома будут обновлены, а компонент Изменения Симптома очистится и вернется в состояние Добавления Симптома.

3.2 Описание взаимодействия пользователя

При запуске приложения, пользователя встречает страница запуска диагностирования (рисунок 3.7).

Рисунок 3.7 – Стартовая страница приложения

При нажатии на кнопку старта пользователя перенаправит на страницу выбора симптомов (рисунок 3.8).

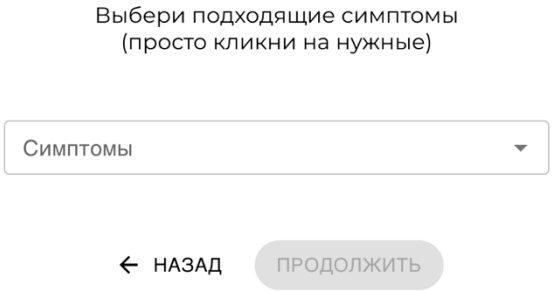

Рисунок 3.8 – Страница выбора симптомов

При нажатии на зеленую галочку откроется выпадающий список с существующими в базе данных симптомами, разделенными на конкретные группы (рисунок 3.9).

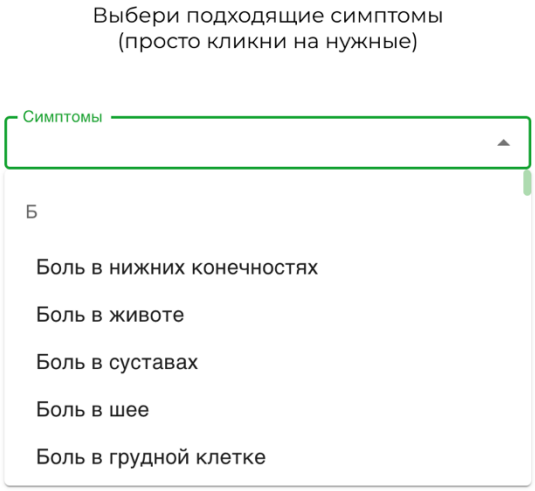

Рисунок 3.9 – Список существующих в БД симптомов

Для того чтобы выбрать нужный симптом необходимо нажать на него, тогда его цвет изменится на зеленый, а рядом появится индикатор того, что он выбран. Для получения диагноза необходимо выбрать не менее 3-х симптомов (рисунок 3.10).

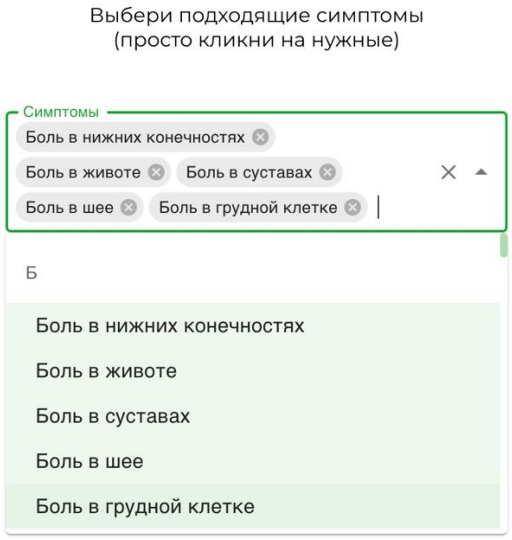

Рисунок 3.10 – Выбранные симптомы

Также для пользователя доступна возможность поиска конкретных симптомов, используя строку ввода данных (рисунок 3.11).

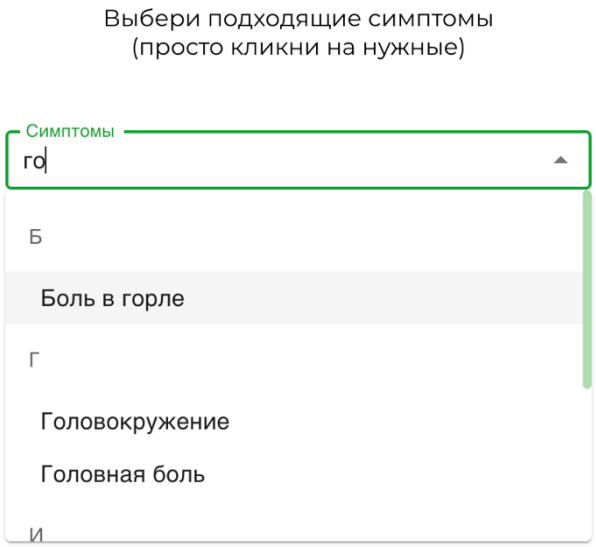

Рисунок 3.11 – Поиск симптомов

После выбора минимум 3-х симптомов пользователь может узнать вероятный диагноз нажав на кнопку вперед. Страница с диагнозом представляет из себя список из наиболее вероятных болезней (рисунок 3.12).

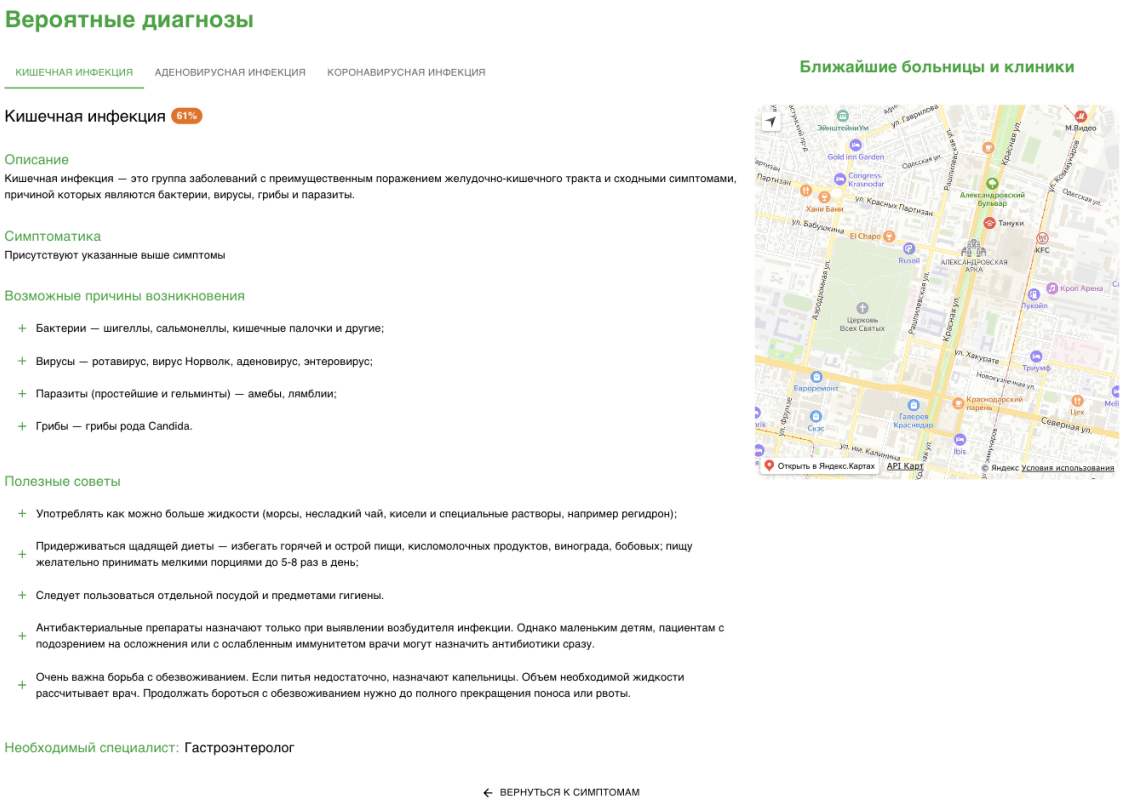

Рисунок 3.12 – Страница вероятностных диагнозов

Для каждого из представленных диагнозов есть возможность узнать его название, описание, некоторые советы которых стоит придерживаться во время болезни, а также необходимый специалист. Внизу страницы располагается карта с ближайшими поликлиниками (рисунок 3.13).

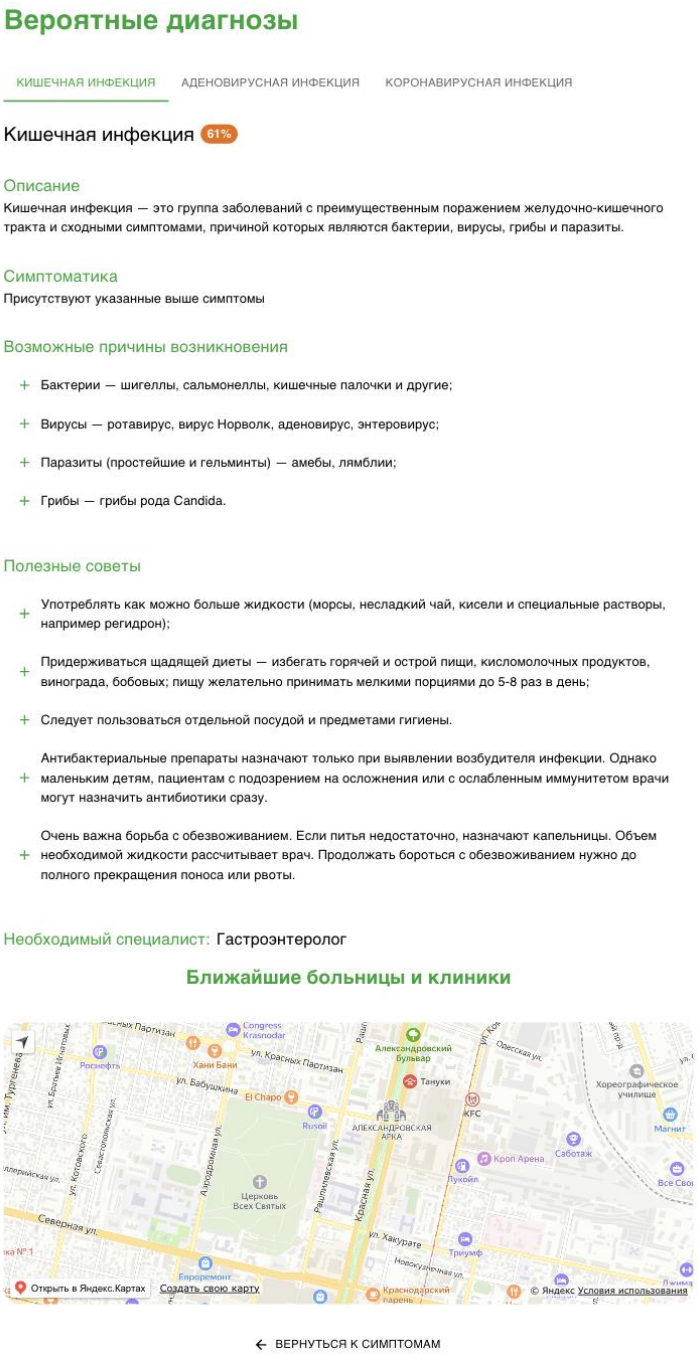

Рисунок 3.13 – Информация о конкретном диагнозе

ЗАКЛЮЧЕНИЕ

В рамках выполнения курсовой работы были изучены основные технологии, использующиеся в разработке API.

Было реализовано собственное API основными функциями которого являются: создание, чтение, обновление и удаление данных в существующей информационной базе, а также страница администрирования для более эффективного использования этих функций.

В дальнейшем планируется расширение функциональных возможностей данного проекта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы кластерного анализа // Учебные материалы онлайн: [сайт] - URL: https://studwood.net/870242/marketing/znachenie_tseli_zadachi_klasternogo_analiza (дата обращения: 05.11.2022).

2. Основные метрики, сходства между объектами // Файловый архив: [сайт] - URL: https://studfile.net/preview/1582407/page:3/ (дата обращения: 10.11.2022).

3. Что такое PostgreSQL // Википедия – свободная энциклопедия: [сайт] - URL: https://ru.wikipedia.org/wiki/PostgreSQL (дата обращения: 17.11.2022).

4. Что такое Node.js // Node.js: сайт // URL: https://nodejs.org/ru/ – (дата обращения: 21.11.2022).

5. Что такое Sequilize // Блог SkillFactory: [сайт] - URL: https://blog.skillfactory.ru/glossary/sequelize/ (дата обращения: 28.12.2022).

6. Что такое Express.js // Express – Node.js web application framework: сайт // URL: https://expressjs.com/ru/ – (дата обращения: 03.12.2022).

7. Что такое Redux Toolkit Query (RTK Query) // Redux Toolkit: [сайт] - URL: https://redux-toolkit.js.org/rtk-query/overview (дата обращения: 06.11.2022).