|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИППО)**

**ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ №1-8**

по дисциплине «Интерфейсы прикладного программирования»

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | ИКБО-20-21, Хитров Н.С. | (подпись) | |
| Принял ассистент | Зарипов Евгений Андреевич | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_2023 г. | |  | |

Москва 2023 г.

**ОГЛАВЛЕНИЕ**

[ПРАКТИЧЕСКАЯ РАБОТА №1 3](#_Toc151469280)

[ПРАКТИЧЕСКАЯ РАБОТА №2 6](#_Toc151469281)

[ПРАКТИЧЕСКАЯ РАБОТА №3 11](#_Toc151469282)

[ПРАКТИЧЕСКАЯ РАБОТА №4 13](#_Toc151469283)

[ПРАКТИЧЕСКАЯ РАБОТА №5 18](#_Toc151469284)

[ПРАКТИЧЕСКАЯ РАБОТА №6 26](#_Toc151469285)

[ПРАКТИЧЕСКАЯ РАБОТА №7 32](#_Toc151469286)

[ПРАКТИЧЕСКАЯ РАБОТА №8 37](#_Toc151469287)

[ВЫВОД ПО ПРАКТИЧЕСКИМ РАБОТАМ 45](#_Toc151469288)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 46](#_Toc151469289)

# ПРАКТИЧЕСКАЯ РАБОТА №1

**Постановка задачи**

Используя любой публичный WEB API (большой список API, например, можно взять на сайте https://any-api.com/), создать приложение, демонстрирующее возможности выбранного API. Ограничений по используемым языкам, готовым библиотекам, упрощающим работу с API нет, но желательно использовать языки JAVA или Python. Например, используя https://developers.google.com/gmail/api/guides, загрузить все входящие сообщения для конкретного пользователя.

**Результат работы**

На Рисунках 1.1 – 1.4 представлены результаты выполнения практической работы.

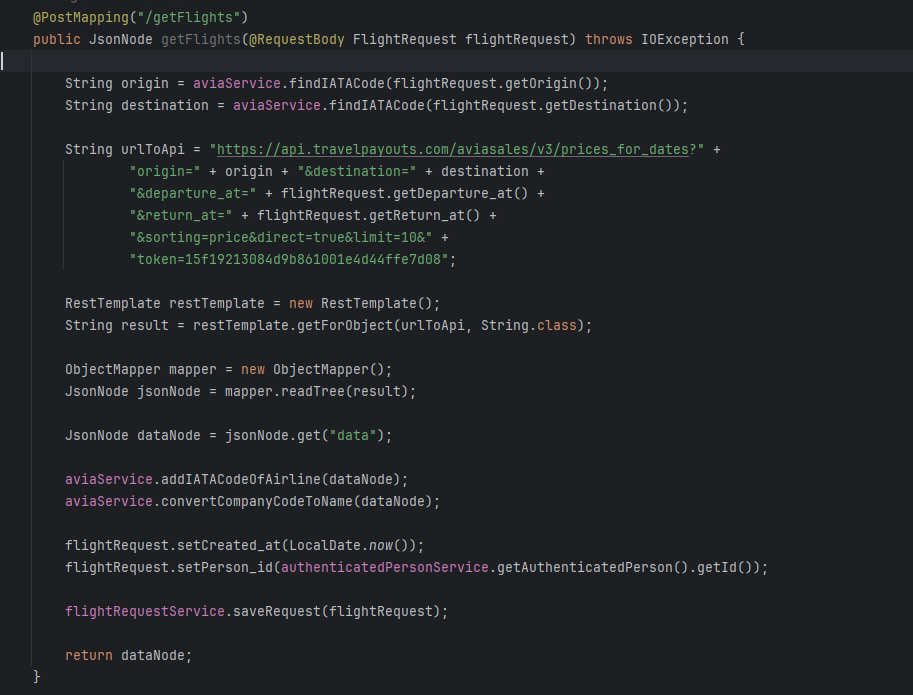


Рисунок 1.1 – Ендпоинт, принимающий запросы пользователя, запрашивающий данные у API Aviasales и возвращающий их пользователю

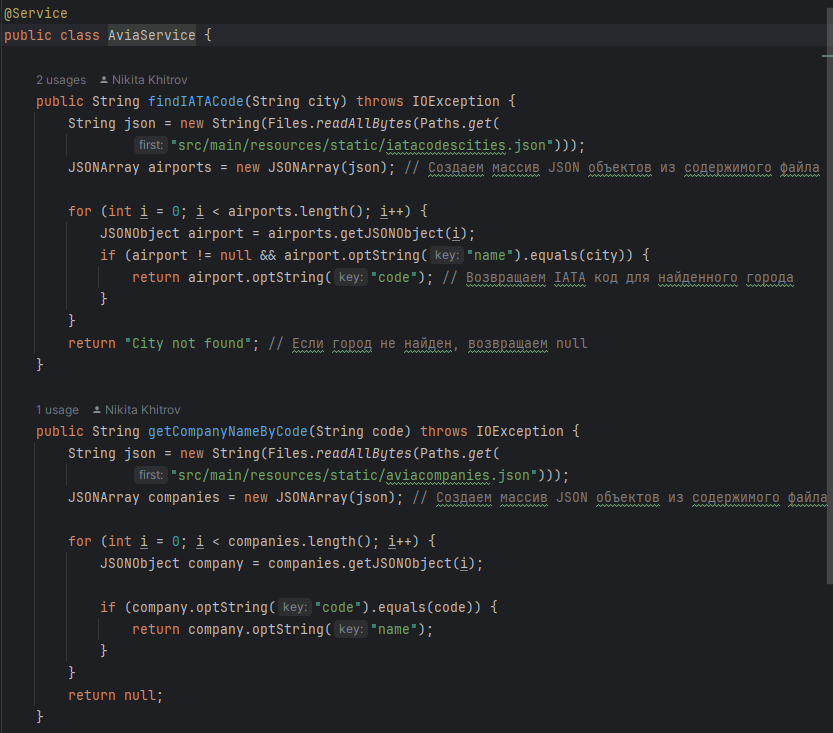


Рисунок 1.2 – Функции сервиса обработки «raw» данных, приходящих с API (часть1)

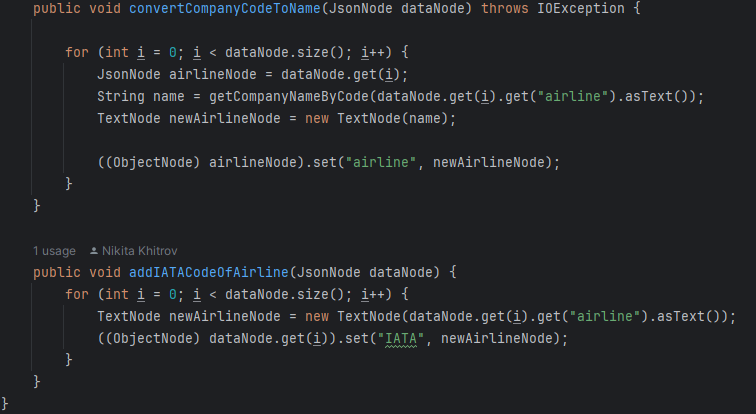


Рисунок 1.3 – Функции сервиса обработки «raw» данных, приходящих с API (часть2)

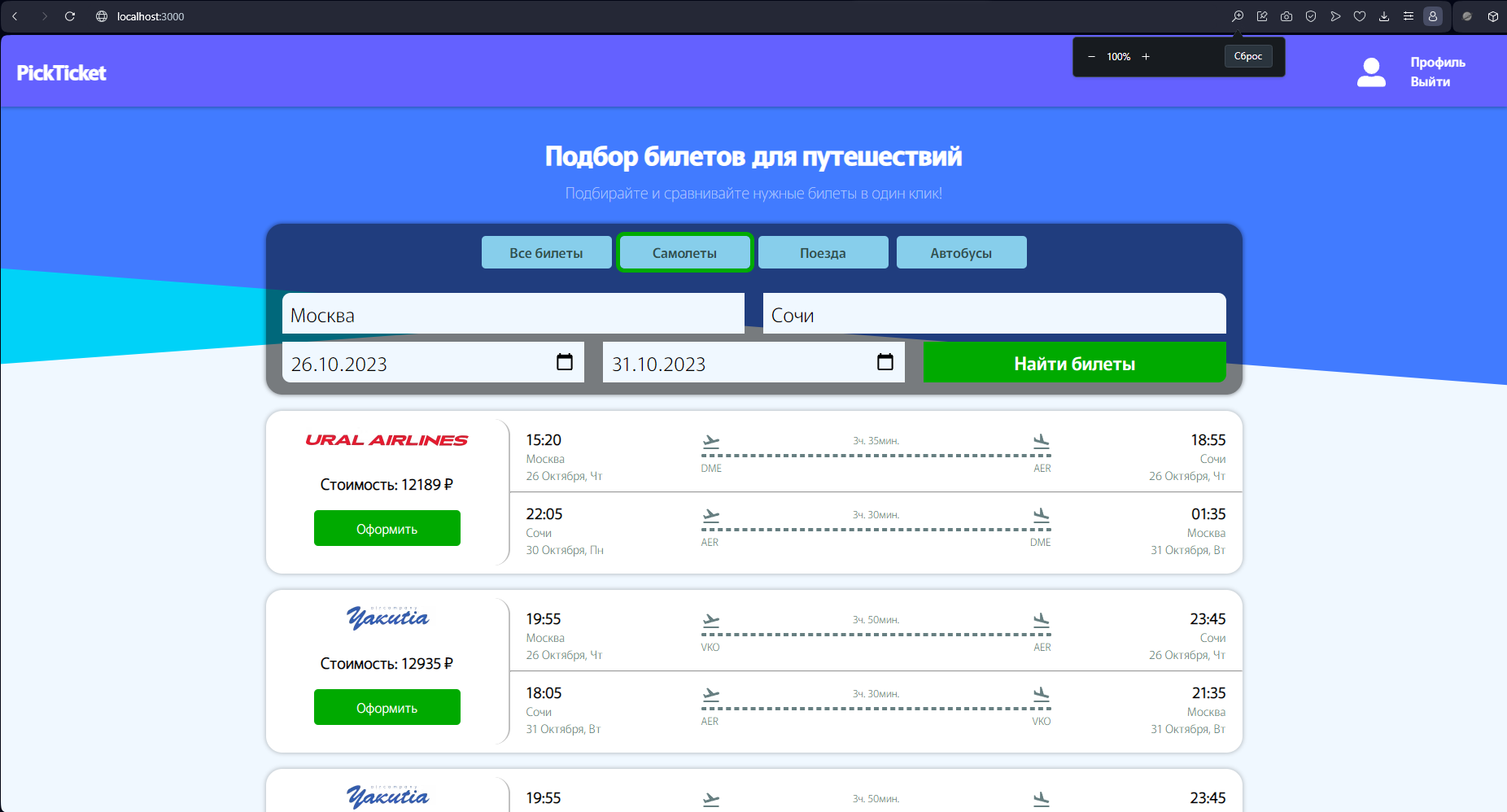


Рисунок 1.4 – Проверка работы сервиса на frontend части

**Вывод**

В результате выполнения работы были получены навыки реализации взаимодействия с публичным WEB API на языке Java. В работе был использован API Aviasales.

# ПРАКТИЧЕСКАЯ РАБОТА №2

**Постановка задачи**

1. Получить список поддерживаемых API Currency Layer валют.

2. Необходимо протестировать API Currency Layer, путём создания запроса об исторических курсах валют для 22 февраля 2018 года, для евро, фунтов стерлингов и иен с исходной валютой доллар США.

3. Получить исторические данные о курсе евро к доллару США, начиная с 25 февраля 2016 по 21 февраля 2017 года.

Дополнительным заданием требуется ознакомиться с API поставщика погоды openweathermap, получив текущие погодные данные для любого выбранного местоположения и протестировать запрос на 16 дневний прогноз для любого города (https://openweathermap.org/forecast16). Попробуйте протестировать API других поставщиков погоды (Gismeteo, Яндекс.Погоды и т. д.).

**Результат работы**

В процессе тестирования выяснилось, что часть конечных точек API OpenWeather недоступна без платной подписки. На Рисунках 2.1-2.4 показаны скриншоты тестирования.

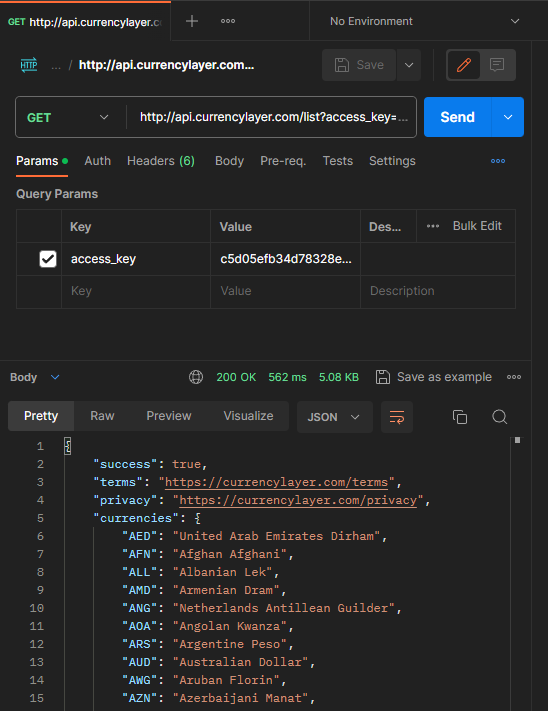


Рисунок 2.1 – Тестирование API Currency Layer для получения списка всех валют

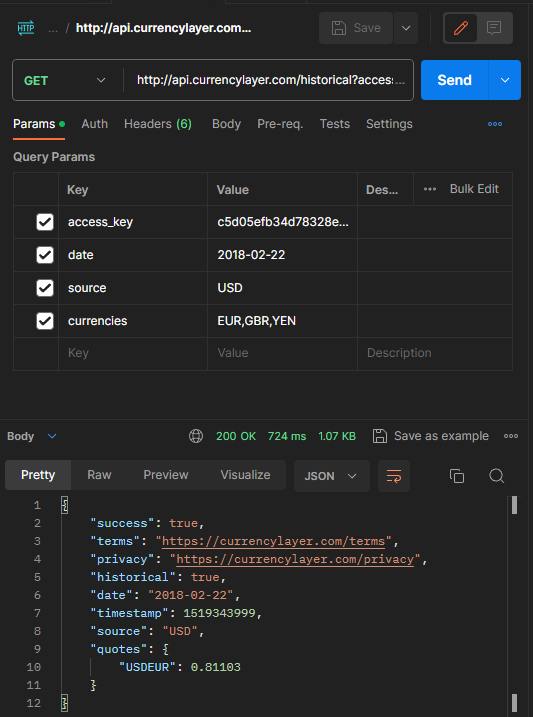


Рисунок 2.2 – Тестирование API Currency Layer для данных об исторических курсах валют для 22 февраля 2018 года, для евро, фунтов стерлингов и иен с исходной валютой доллар США

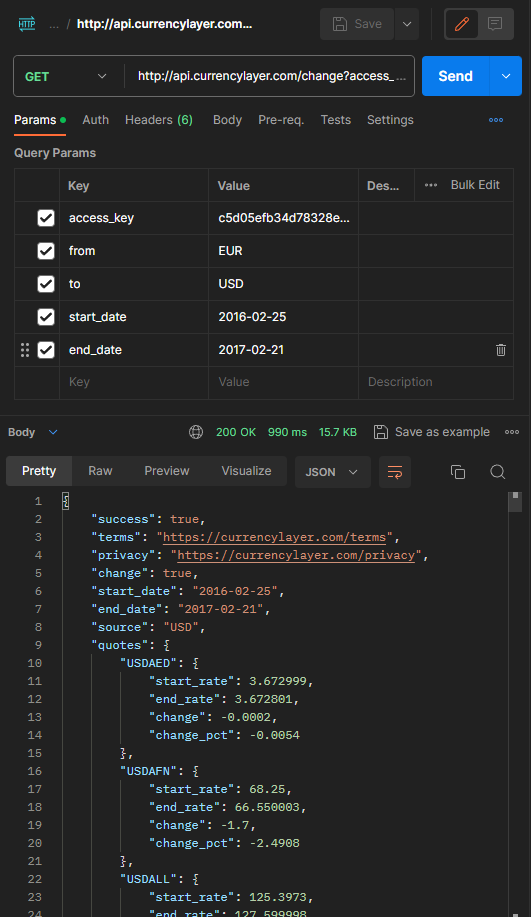


Рисунок 2.3 – API Currency Layer для данных об историческом курсе евро к доллару США, начиная с 25 февраля 2016 по 21 февраля 2017 года

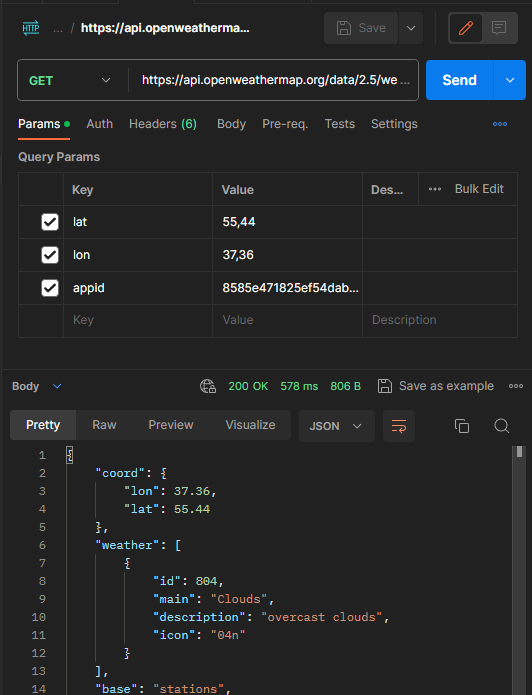


Рисунок 2.4 – Тестирование API OpenWeather

**Вывод**

В ходе работы были получены навыки работы с WEB API CurrencyLayer и OpenWeather.

# ПРАКТИЧЕСКАЯ РАБОТА №3

**Постановка задачи**

Вам необходимо создать веб-страницу, на которой делается запрос к API при помощи Asynchronous Javascript And Xml (AJAX). В качестве примера рассмотрим создание веб-страницы с информацией о текущем прогнозе погоды, полученной от поставщика openweathermap.org.

**Результат работы**

На Рисунках 3.1-3.2 показаны код страницы и результат отображения страницы в браузере.

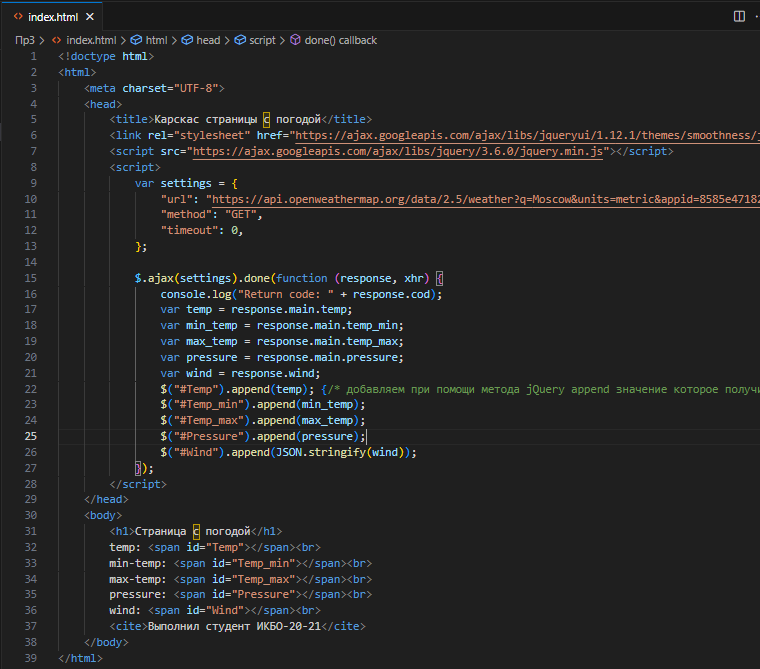


Рисунок 3.1 – Код страницы

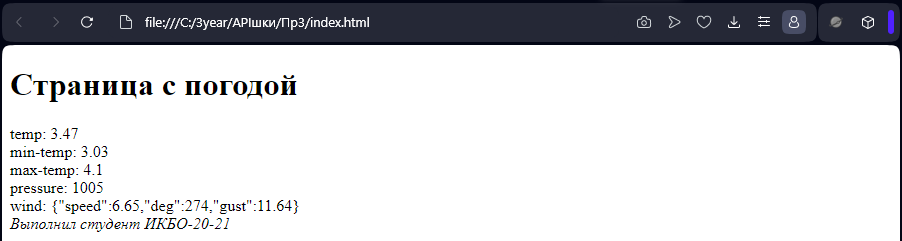


Рисунок 3.2 – Результат отображения страницы

**Вывод**

В результате выполнения работы были получены навыки отправки запросов к WEB API при помощи утилиты cURL и модуля AJAX в библиотеке JQuery, а также отображения полученных ответов на веб-странице при помощи JQuery.

# ПРАКТИЧЕСКАЯ РАБОТА №4

**Постановка задачи**

1. Проверить, установлен ли NodeJS. В случае отсутствия установить с официального сайта разработчика.

2. По аналогии с вышеуказанным примером, необходимо, с помощью jsonserver, создать имитационный API для сервиса поиска книг в библиотеке. Для этого необходимо создать, как минимум три конечных точки: авторы, книги и библиотеки.

3. Протестировать работу с локальным сервисом, путём соответствующих запросов. Дать объяснение каждому запросу и возвращаемому ответу. Для этого желательно использовать curl или Postman.

**Результат работы**

На Рисунке 4.1 показан созданный JSON для имитационного API. На Рисунке 4.2 показан вывод json-server в консоли. На Рисунке 4.3 показано получение всех книг, на 4.4 получение книги с идентификатором 3, на 4.5 все писатели.

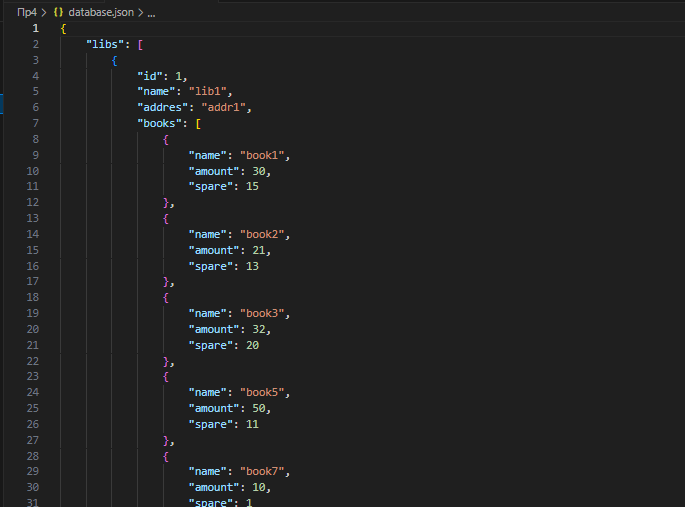


Рисунок 4.1 – Созданный файл базы-данных database.json

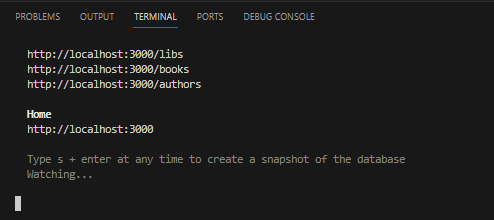


Рисунок 4.2 – Вывод json-server

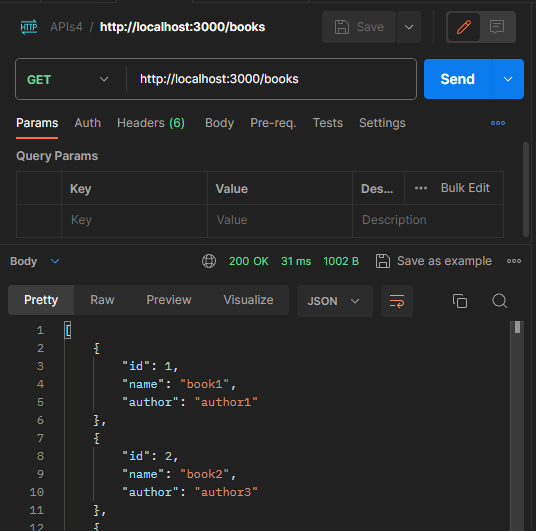


Рисунок 4.3 – Получение всех книг

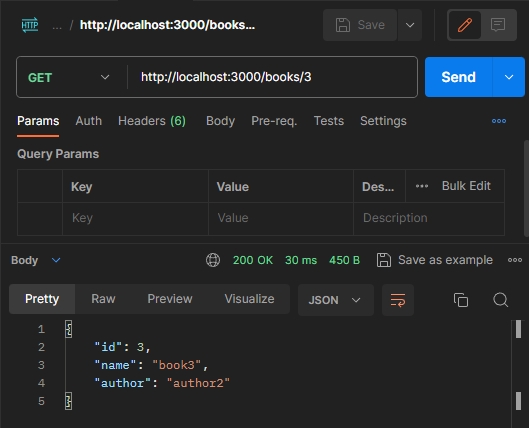


Рисунок 4.4 – Получение книги с id = 3

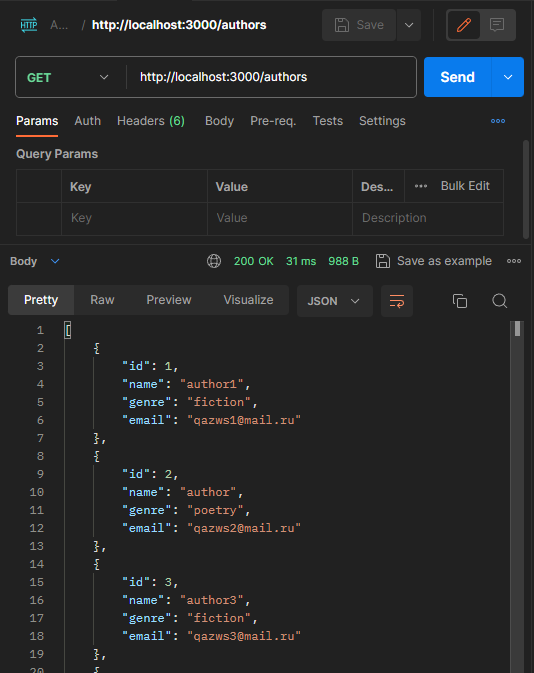


Рисунок 4.5 – Получение всех писателей

**Вывод**

В результате выполнения работы был реализован имитационный API на json-server. API поддерживает СRUD операции и благодаря простоте реализации позволяет проводить тестирование на импровизированной базе данных.

# ПРАКТИЧЕСКАЯ РАБОТА №5

**Постановка задачи**

Используя материалы практической работы и предложенный стек технологий, необходимо создать и развернуть на сервере Heroku приложение для управления списком телефонных контактов. Стоит обратить внимание на то, что используемый стек технологий очень активно развивается, а также сама облачная PaaS система Heroku может вводить определённые ограничения на поддержку используемых в данном проекте технологий, поэтому могут возникать определённые трудности со сборкой проекта, при попытке простого копирования исходников кода из данной практической работы. По согласованию с преподавателем, можно выбрать и другой стек технологий, а также и другую облачную систему.

**Результат работы**

В результате выполнения работы было разработано RESTful приложение, состоящее из серверной и клиентской части, а также базы данных MongoDB (стек MERN). На рисунках 5.1-5.2 представлен код серверной части приложения. На рисунках 5.3-5.9 представлено тестирование разработанного приложения. Приложение было развернуто на клауд-хостинге Render. База данных хранится и работает на сервере MongoDB Atlas.

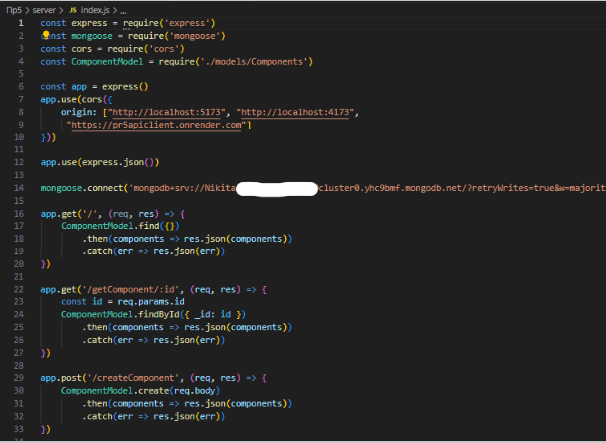


Рисунок 5.1 – Код серверной части (часть 1)



Рисунок 5.2 – Код серверной части (часть 2)

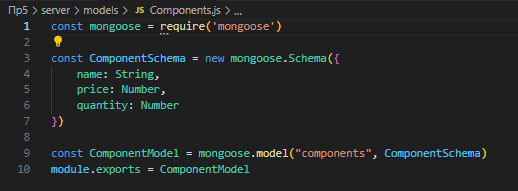


Рисунок 5.3 – Код описывающий схему таблицы MongoDB

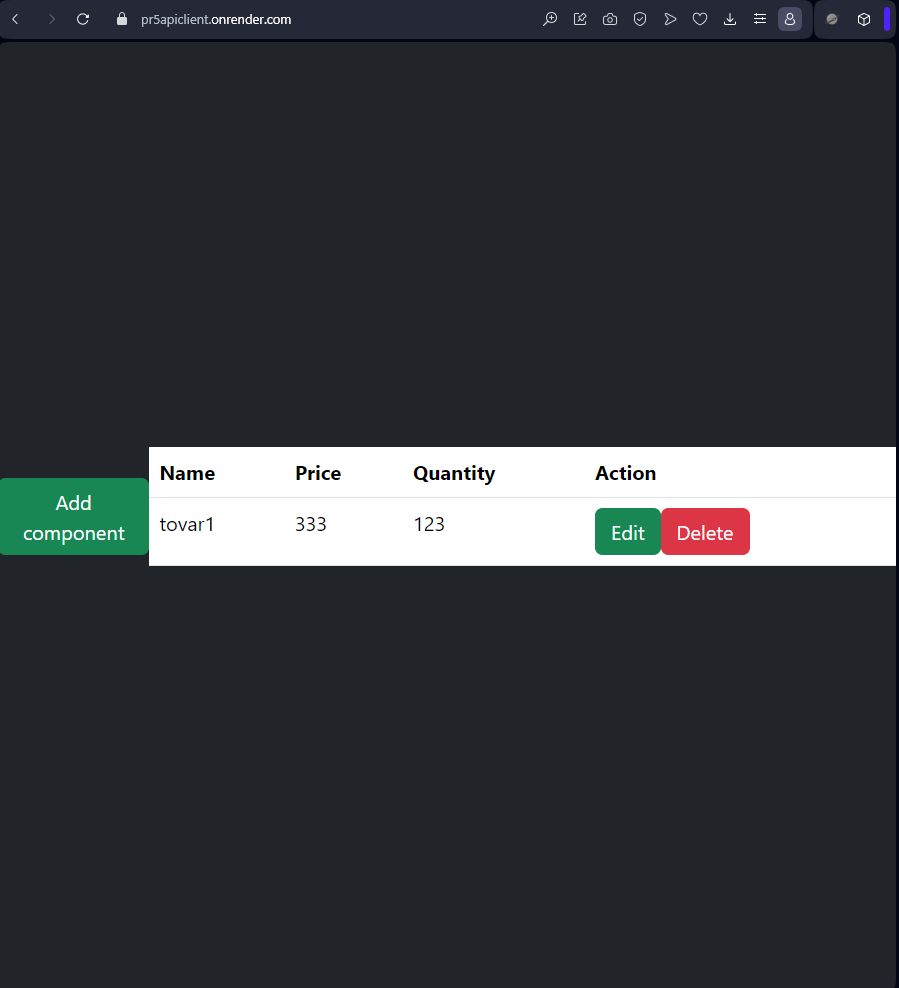


Рисунок 5.4 – Страница получения всех записей

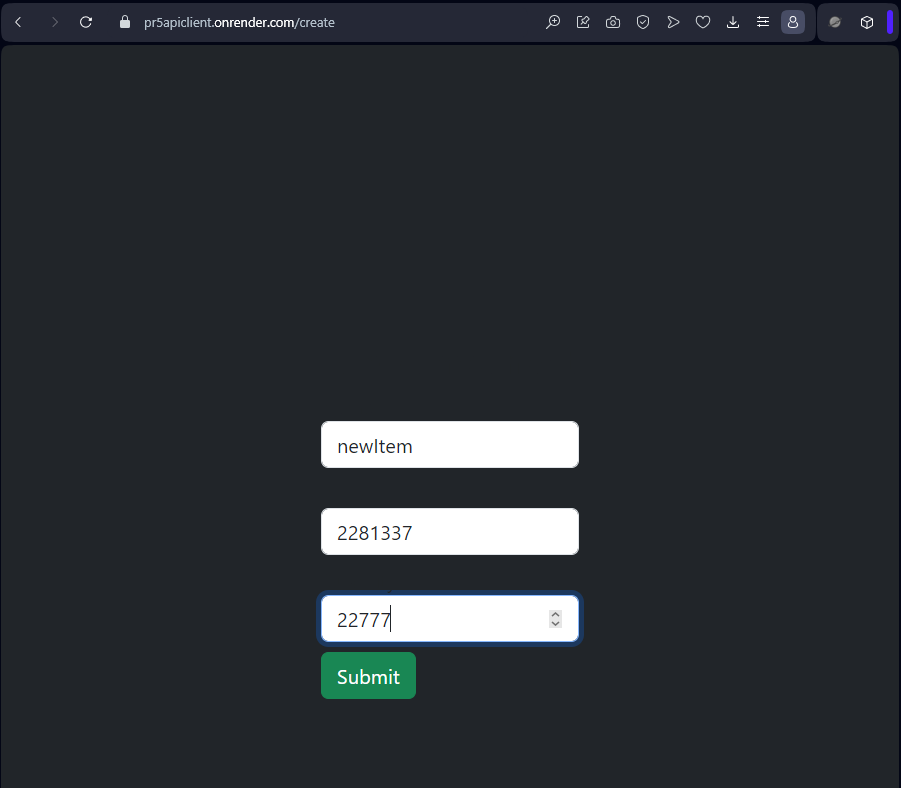


Рисунок 5.5 – Страница добавления записи

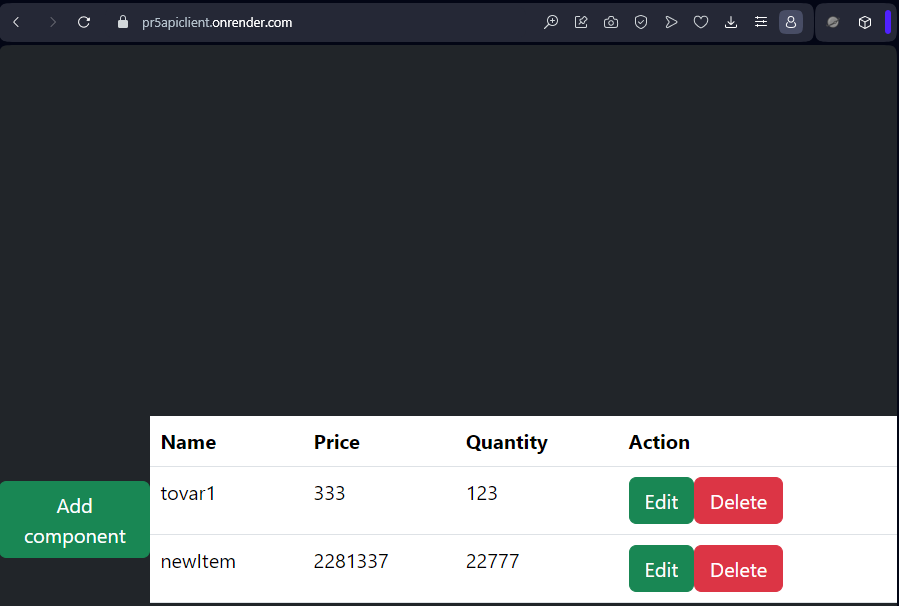


Рисунок 5.6 – Результат добавления записи

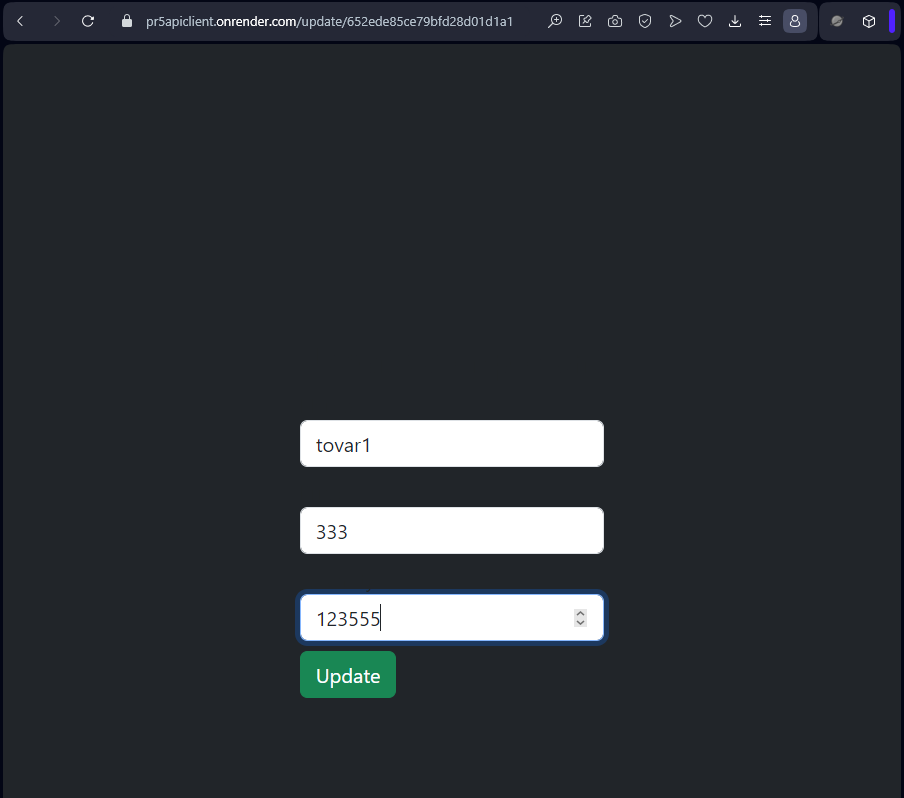


Рисунок 5.7 – Страница обновления записи

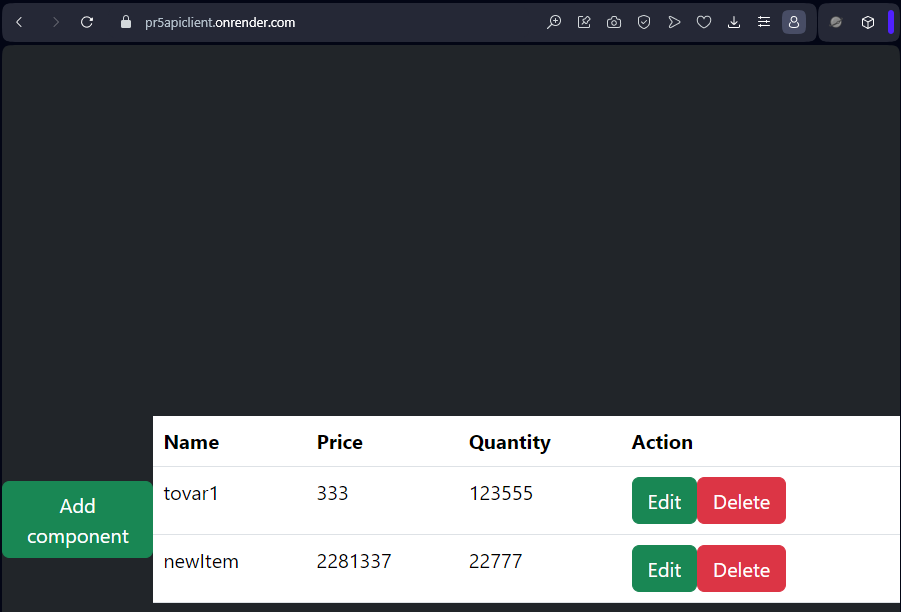


Рисунок 5.8 – Результат обновления записи

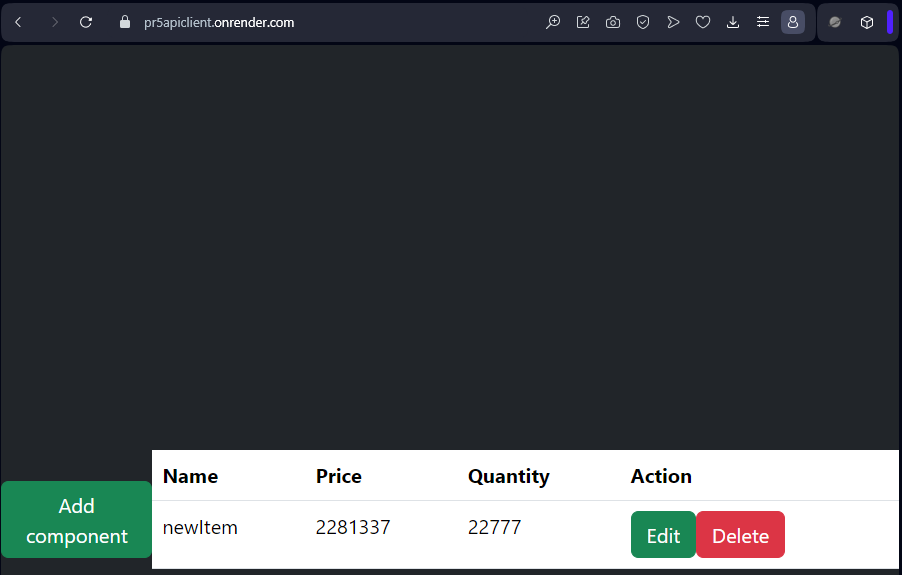


Рисунок 5.9 – Результат удаления записи

API расположен по ссылке <https://pr5apiclient.onrender.com>.

**Вывод**

В ходе выполнения работы были получены навыки реализации веб-приложения на стеке MERN (M – MongoDB, E – Express, R – React, N - NodeJS), а также его развертки на клауд-хостинге Render.

# ПРАКТИЧЕСКАЯ РАБОТА №6

**Постановка задачи**

В данной практической работе необходимо ознакомиться с техникой разработки документации и создания программных интерфейсов на основе спецификации OpenAPI. В качестве задания необходимо выбрать любую прикладную задачу, например связанной с тематикой компьютерных игр, компьютерного интернет магазина и т. д. и создать интерфейс прикладного программирования для данной задачи, используя для этого OpenAPI, ReDOC, prism. Для создания спецификации OpenAPI на языке YAML можно воспользоваться Swagger Editor (https://editor.swagger.io/) Для проверки правильности работы решения необходимо отправить соответствующий запрос, например, в Postman и сверить запрос с ReDoc в правой части html-страницы.

**Результат работы**

В результате выполнения данной работы был создан файл swagger.yaml для API маркетплейса компьютеров. На рисунках 6.1-6.3 привед код этого файла. На рисунке 6.5 показана сгенерированная ReDoc страница с описанием API. На рисунке 6.6 показан вывод Prism в консоль, а на рисунках 6.7-6.8 тестирование вызовов Prism в Postman.

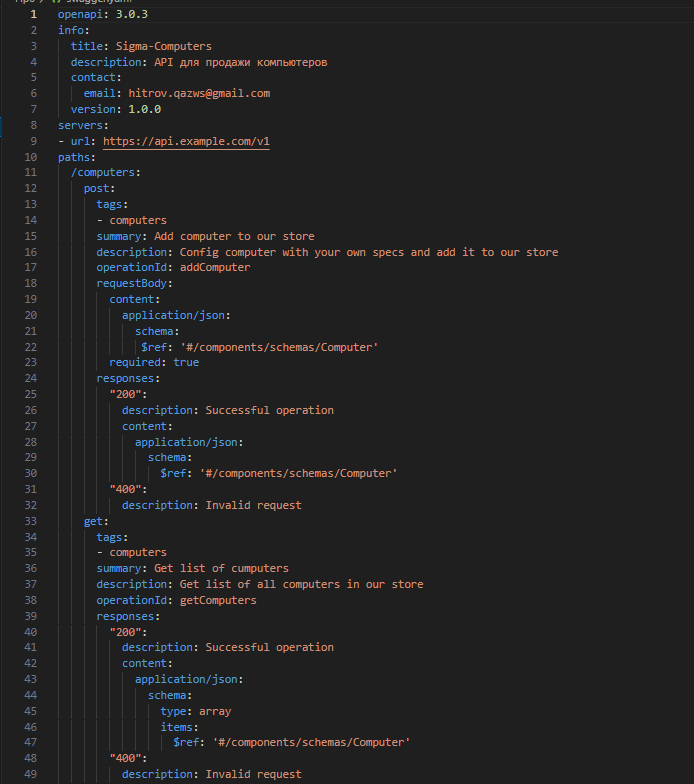


Рисунок 6.1 – Первая часть файла swagger.yaml

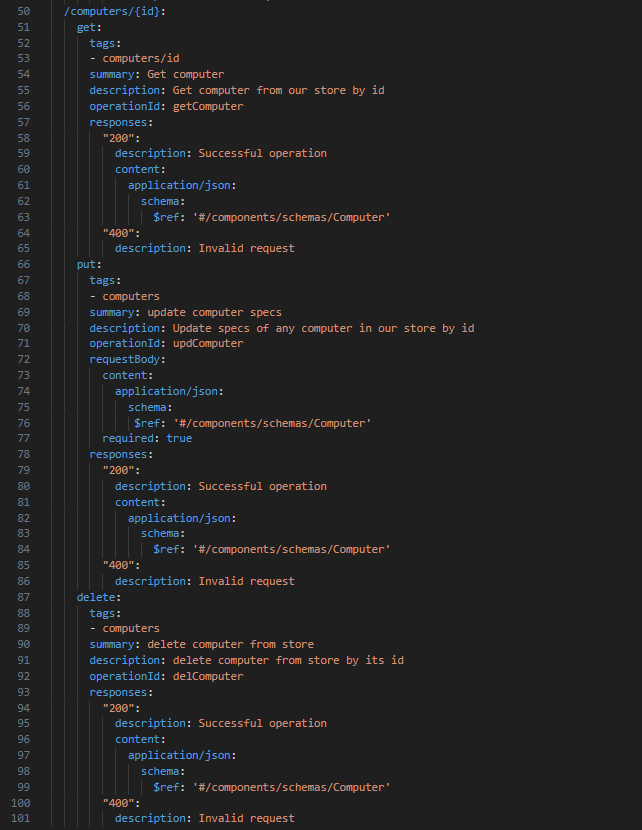


Рисунок 6.2 – Вторая часть файла swagger.yaml

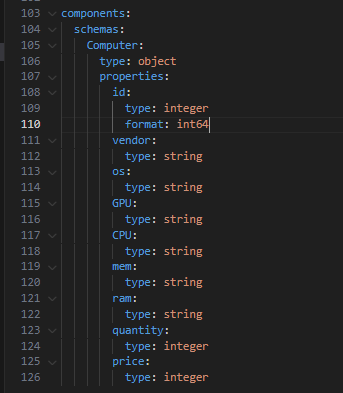


Рисунок 6.4 – Третья часть файла swagger.yaml

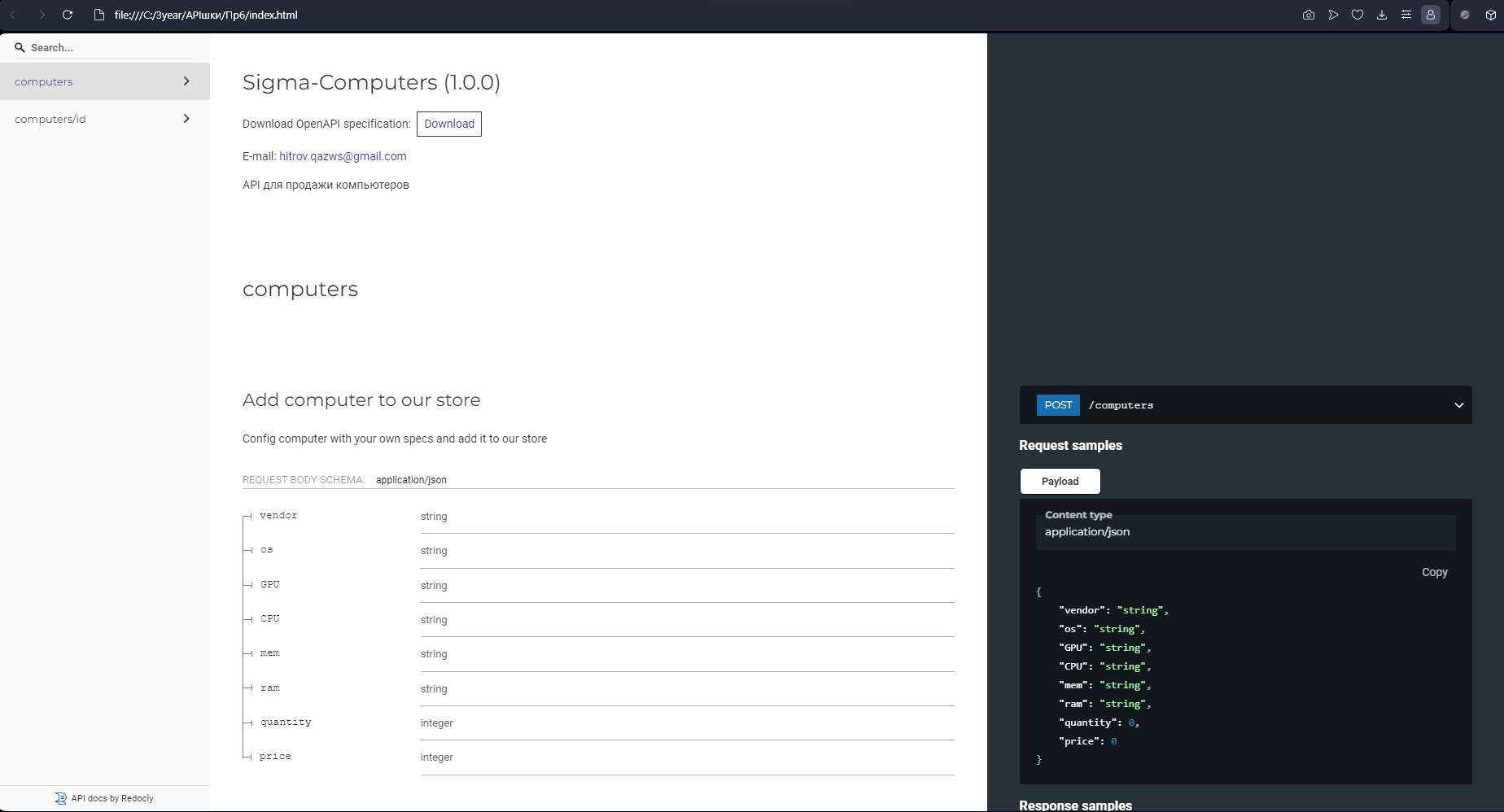


Рисунок 6.5 – Сгенерированная страница ReDoc

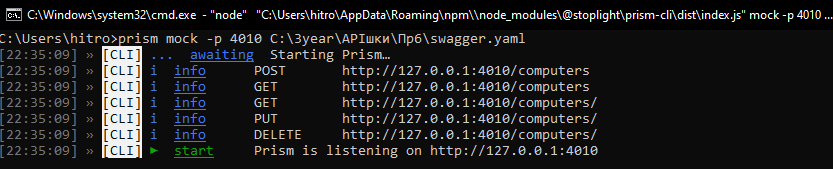


Рисунок 6.6 – Вывод prism в консоли

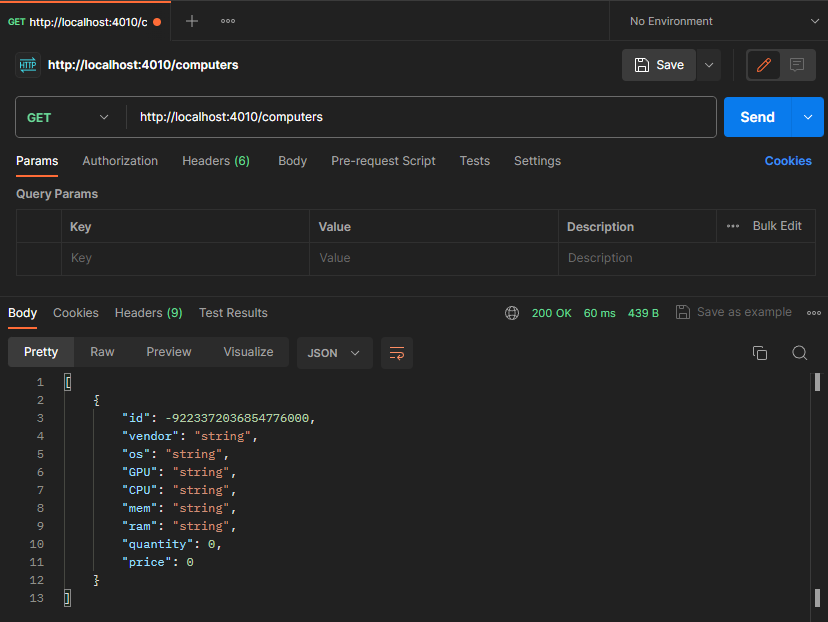


Рисунок 6.7 – Тестирование вызова к Prism

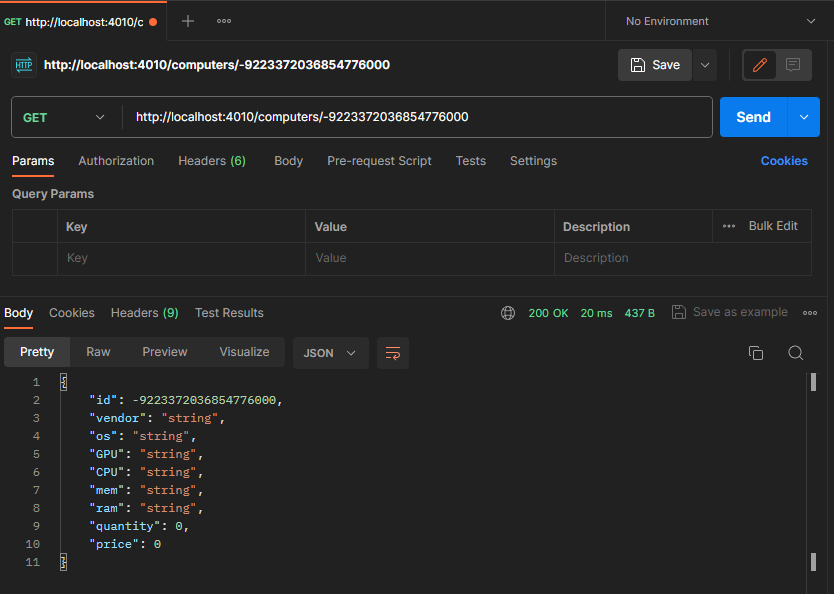


Рисунок 6. – Тестирование получения записи по id из Prism

**Вывод**

В ходе выполнения работы были получены навыки написания спецификации API с помощью OpenAPI, ее документации в документе формата html при помощи утилиты ReDoc, а также навыки использования prism для реализации интерфейса прикладного программирования по ранее созданной спецификации.

# ПРАКТИЧЕСКАЯ РАБОТА №7

**Постановка задачи**

Используя теоретические сведения из данной практической работы, открытые интернет-источники, официальную документацию по GraphQL необходимо, с использованием SDL создать схему, реализовать сервер и клиента GraphQL для следующих бизнес-задач. В качестве варианта был выбран сервис для хранения списка покупок. Схема должна описывать следующий полный набор данных: уникальный идентификатор товара для покупки (поле id), краткое именование товара (поле text), количество требуемого товара (поле qty), поле флага покупки, определяемое через true/false (поле completed), а также уникальный идентификатор пользователя. Если требуются дополнительные поля, то они определяются обучающимися самостоятельно.

**Результат работы**

В результате выполнения работы был создан сервис, использующий GraphQL. На рисунке 7.1 показан код запуска сервера, на рисунках 7.2-7.4 показан код схемы с настроенными запросами и мутациями.

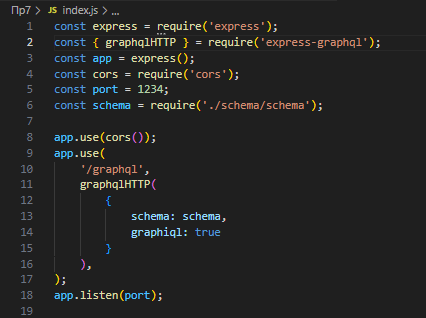


Рисунок 7.1 – Код index.js

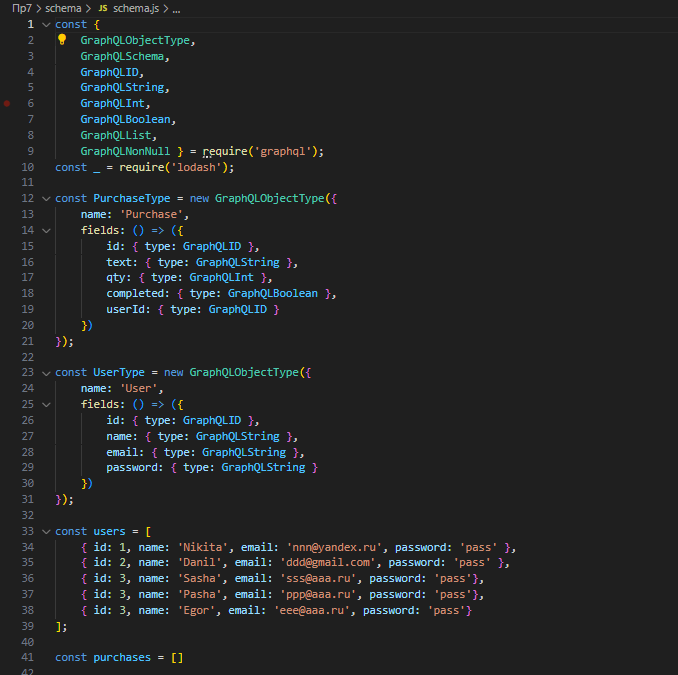


Рисунок 7.2 – Первая часть листинга кода схемы



Рисунок 7.3 – Вторая часть листинга кода схемы

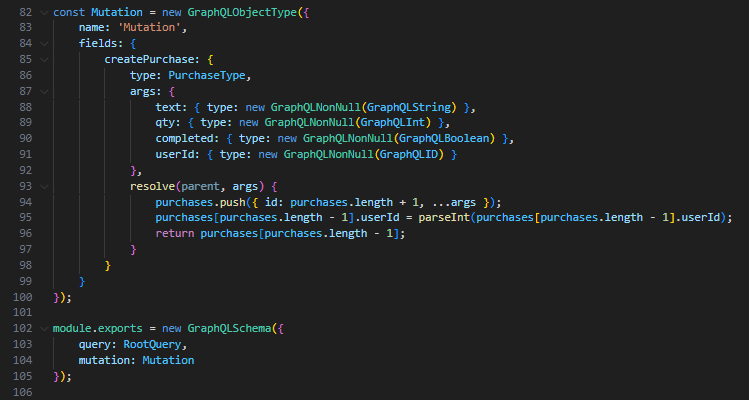


Рисунок 7.4 – Третья часть листинга кода схемы

На рисунках 7.5-7.7 показано тестирование созданного сервиса при помощи graphiql.

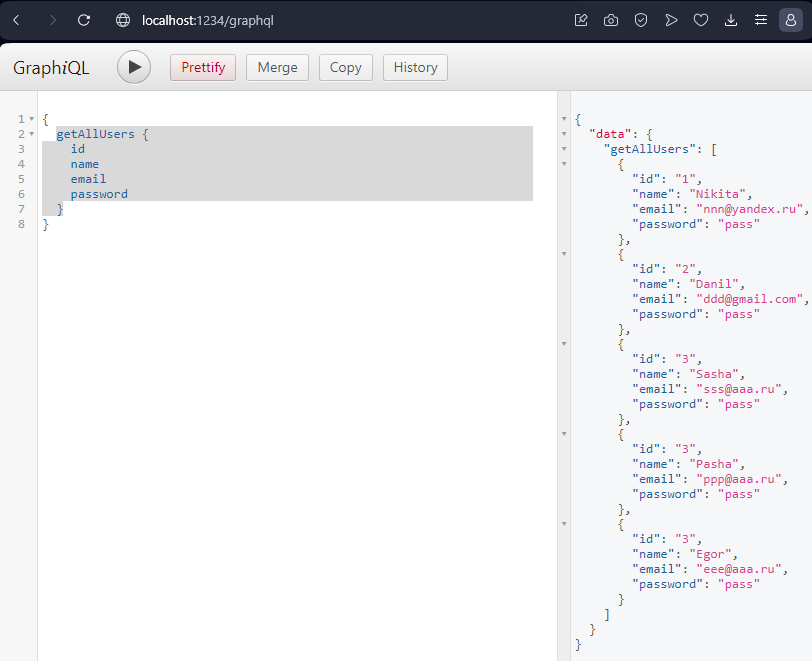


Рисунок 7.5 – Результат получения всех пользователей

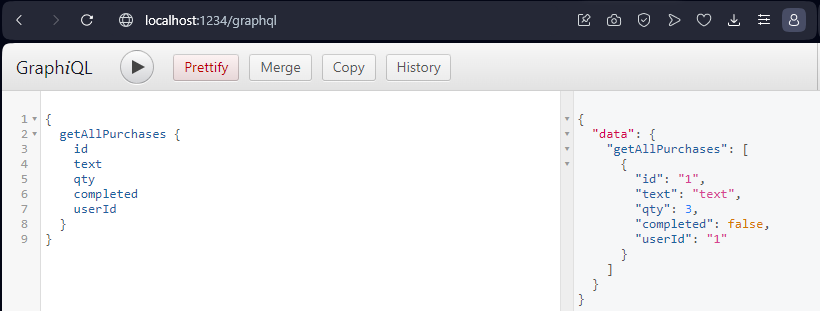


Рисунок 7.6 – Результат получения всех покупок

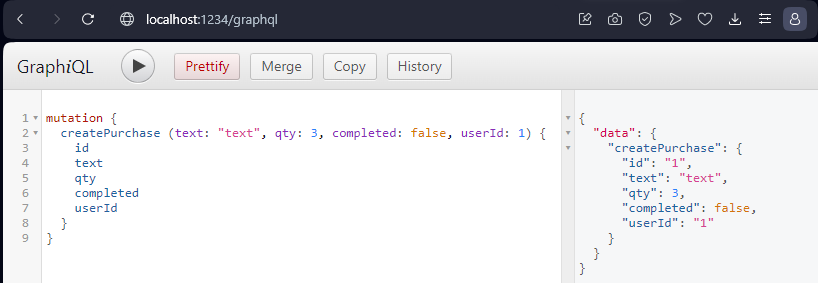


Рисунок 7.7 – Результат создания новой покупки

**Вывод**

В ходе выполнения работы были получены навыки написания GraphQL схемы с помощью SDL, а также навыки реализации сервера и клиента GraphQL для решения поставленных бизнес задач.

# ПРАКТИЧЕСКАЯ РАБОТА №8

**Постановка задачи**

Используя теоретические сведения из данной практической работы, открытые интернет-источники, официальную документацию по gRPC необходимо реализовать минимально жизнеспособный продукт (MVP) клиент-серверное приложение с API и базовыми операциями create, read, update, delete (CRUD) для следующих бизнес-задач: сервис для формирования списка личных задач на день. Приложение должно позволять создавать задачу, редактировать её, просматривать список задач, удалять задачу из списка. Дополнительный функционал, которые вы хотели бы реализовать в данном приложении, необходимо обсудить и согласовать с преподавателем.

**Результат работы**

В результате выполнения данной работы был реализован gRPC-сервис для операций входа, регистрации, изменения и удаления пользователей. На рисунке 8.1 показан protobuf-файл todo.proto, согласно которому описаны сообщения и методы сервиса.

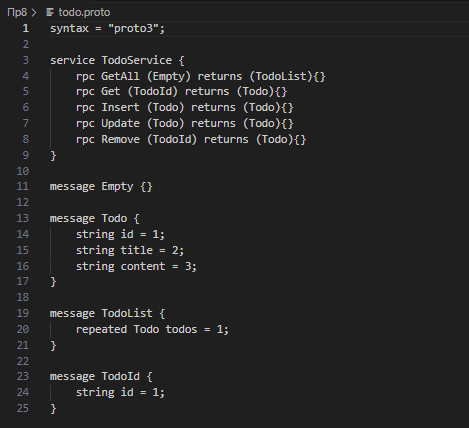


Рисунок 8.1 – Код файла спецификации - todo.proto

Далее были созданы реализации методов в server.js. На рисунке 8.3 показан код данного файла.

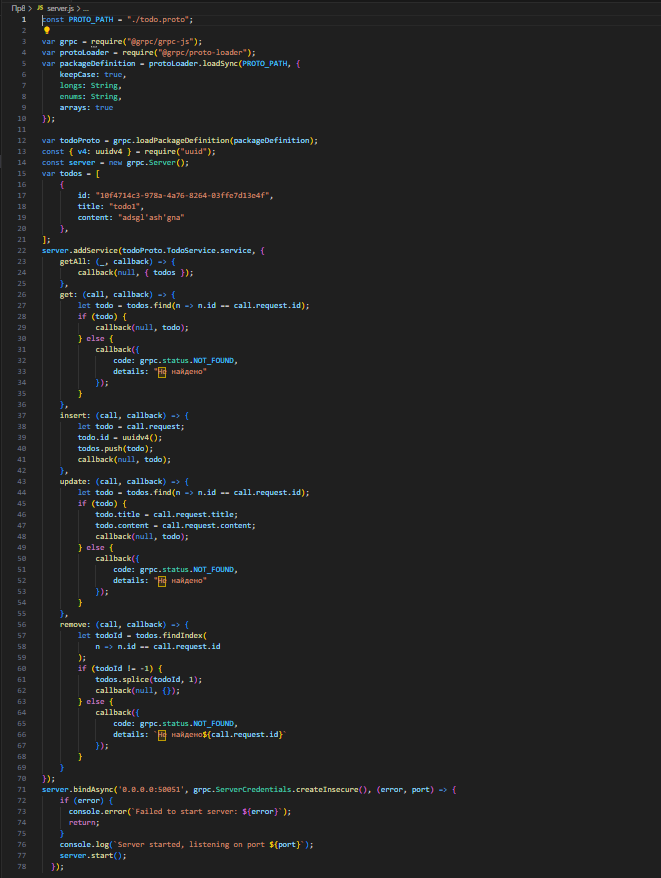


Рисунок 8.2 – Код файла, реализующего сервер - server.js

На рисунке 8.3 показан листинг кода client.js, используемого для отправки запросов к gRPC-серверу.

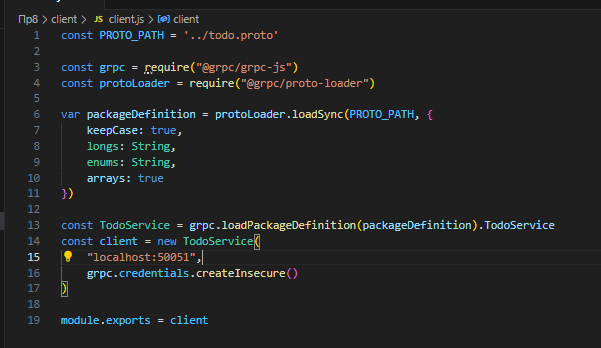


Рисунок 8.3 – код файла client.js

На рисунке 8.4 показан код express-сервера, через который пользователь взаимодействует с сервером, посредством gRPC.

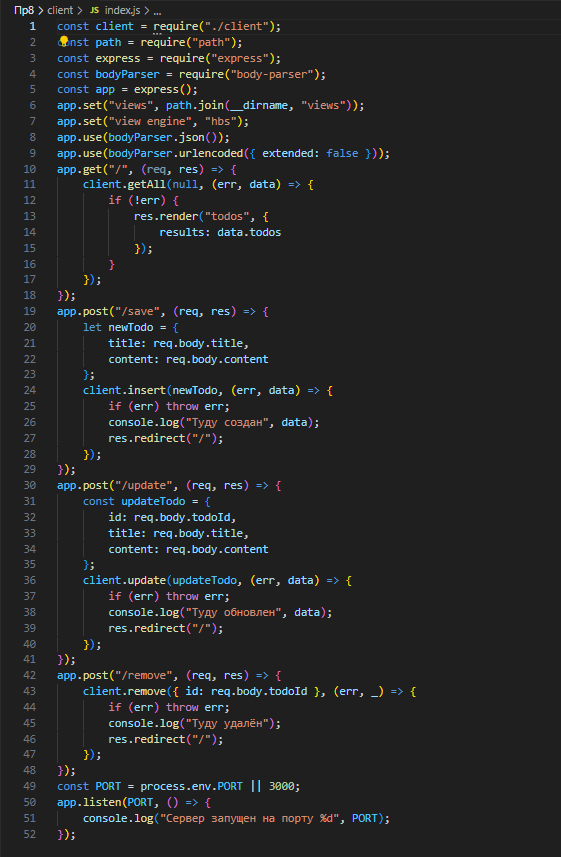


Рисунок 8.4 – Код файла веб-сервера - index.js

Далее была создана фронтенд часть приложения (todos.hbs). На рисунках 8.5-8.6 изображен код реализующий html-страницу с помощью шаблонизатора Handlebars.



Рисунок 8.5 – Код файла html-страницы - todos.hbs (часть 1)

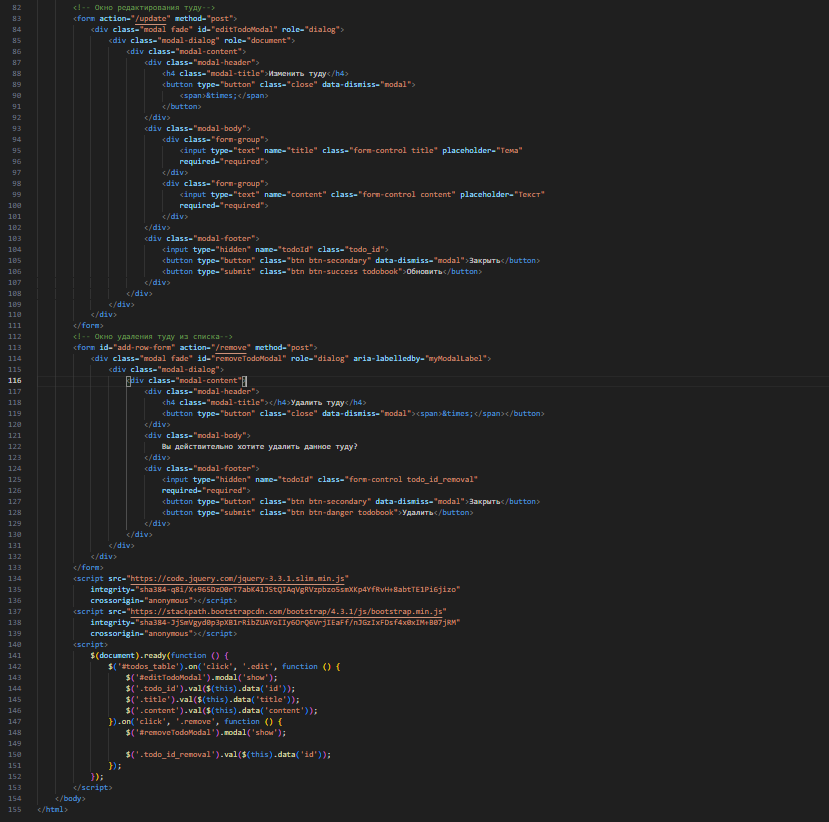


Рисунок 8.6 – Код файла html-страницы - todos.hbs (часть 2)

Результат запуска и использования сервиса показан на рисунках 8.7-8.8.

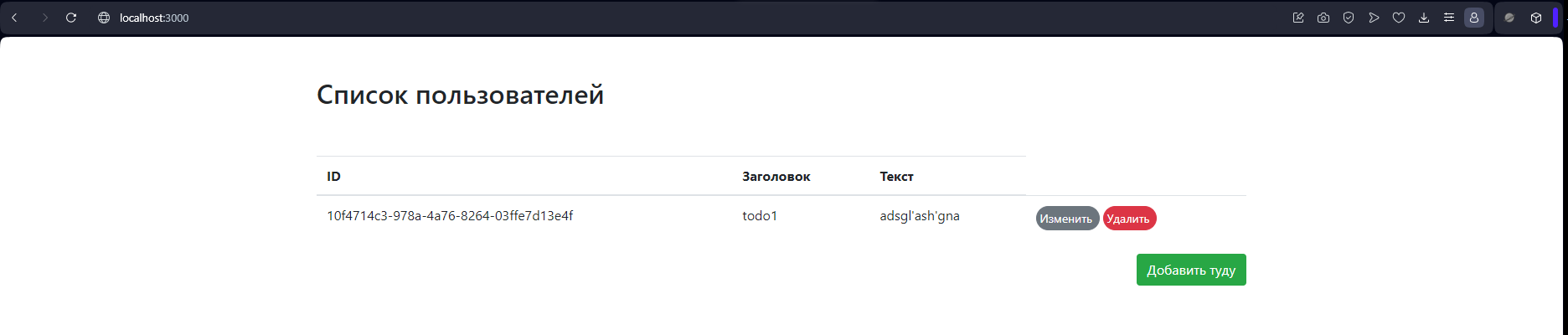


Рисунок 8.7 – Список всех задач

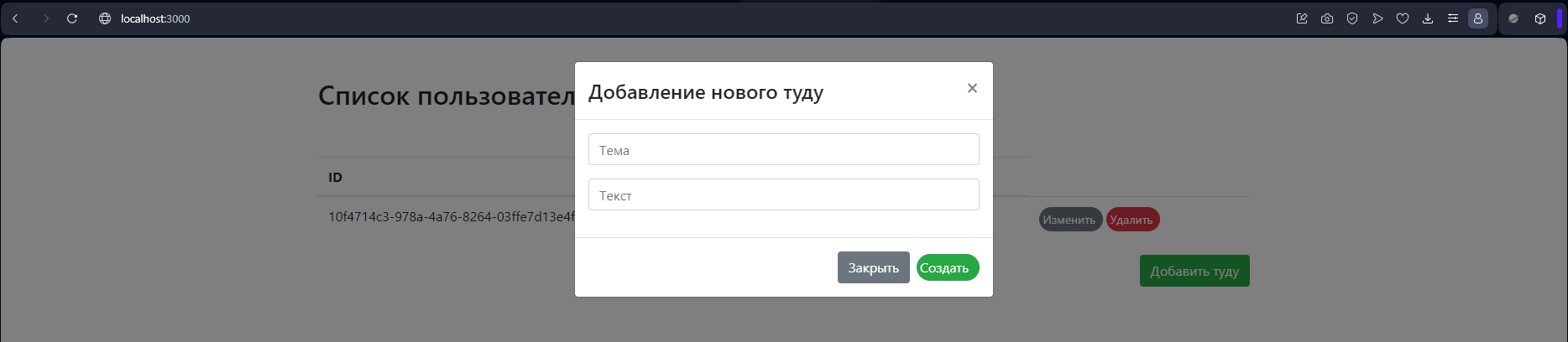


Рисунок 8.8 – Модальная форма добавления задач

**Вывод**

В ходе выполнения работы были получены навыки написания MVP CRUD клиент-серверных приложений на архитектуре gRPC.

# ВЫВОД ПО ПРАКТИЧЕСКИМ РАБОТАМ

В ходе выполнения восьми практических работ были освоены различные аспекты работы с публичными WEB API. Каждая из задач предоставила ценный опыт в создании приложений, демонстрирующих функциональность выбранных API. Были созданы веб-страницы с использованием AJAX для запросов к openweathermap.org, что позволило динамически обновлять данные о прогнозе погоды.

Далее, была проведена работа с jsonserver для имитации API сервиса поиска книг в библиотеке, созданы соответствующие конечные точки и протестированы запросы с использованием curl и Postman. Затем, с использованием предоставленного стека технологий, было успешно развернуто приложение для управления списком телефонных контактов на сервере.

Также был освоен процесс разработки документации и создания программных интерфейсов с использованием спецификации OpenAPI, ReDOC и prism. Этот этап позволил создать удобный интерфейс программирования приложения, а также проверить его работу с помощью Postman.

Наконец, с применением знаний о GraphQL и gRPC были реализованы клиент-серверные приложения с базовыми операциями CRUD, демонстрируя понимание принципов работы этих технологий. В результате, данные практические работы позволили овладеть важными навыками работы с различными API и их интеграцией в приложения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справочная информация JavaRush – URL: https://javarush.com/groups/posts/1094-ajax-i-drugie-jazihki (дата обращения: 07.09.2023) – текст: электронный.
2. Статья Baeldung – URL: https://www.baeldung.com/angular-js-rest-api (дата обращения: 10.09.2023) – текст: электронный.
3. Статья Habr – URL: https://habr.com/ru/articles/590679/ (дата обращения: 12.09.2023) – текст: электронный.
4. Методические указания для выполнения правктических работ – URL: https://online-edu.mirea.ru/pluginfile.php?file=%2F970821%2Fmod\_resour ce%2Fcontent%2F0%2FПрактикум\_по\_проектированию\_и\_разработке\_программных\_интерфейсов.pdf (дата обращения: 28.09.2023) – текст: электронный
5. Статья Habr – URL: https://habr.com/ru/companies/ruvds/articles/445268 (дата обращения: 10.10.2023) – текст: электронный.
6. Статья Medium – URL: https://medium.com/maddevs-io/introduction-to-grpc-6de0d9c0fe61 (дата обращения: 15.10.2023) – текст: электронный.