|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий (ИТ) |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
| **ОТЧЕТ**  **ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 6** | |
| **по дисциплине** |  |
| **«Архитектура клиент-серверных приложений»**  **Тема: GraphQL** | |
| Выполнил студент группы ИКБО-20-22 | Шумахер М.Е. |
| Принял преподаватель кафедры ИиППО | Волков М.Ю. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | «04» сентября 2024 г. |  |
| «Зачтено» | «04» сентября 2024 г. |  |

Москва 2024

**Цель работы:** знакомство обучающихся с набирающим популярность современным подходом к проектированию и реализации API на основе графовых моделей и с реализующей данный подход технологией на основе спецификации GraphQL.

**Теоретическое введение:**

В предыдущих работах, нами было рассмотрен один из самых распространённых, на сегодняшний момент, подходов к проектированию и разработке программных интерфейсов в вебе, который основан на архитектуре REST. Используемый ресурсно-ориентированный подход предполагает организацию данных поверх веб-ресурсов, определяемых посредством URI, а для реализации базовых операций, таких как чтение/запись, удаление, обновления использовались базовые глаголы протокола HTTP. Однако, несмотря на все преимущества данного подхода, у REST подхода имеется ряд существенных недостатков, которые могут влиять, при определённых сценариях использования, в том числе, на производительность программного интерфейса.

Самый типичный сценарий взаимодействия с API, демонстрирующий ограничения REST по сравнению с архитектурой на основе графовой модели, является сложная выборка данных. Например, имеется веб- приложение, реализующую функционал социальной сети. Клиентская часть этого приложения (например, веб-клиент) должна отображать множество данных, таких как количество просмотров страницы, количество друзей, подписчиков и т.д. В случае использования REST API, для извлечения всех требуемых данных, обычно, необходимо обратиться к нескольким конечным точкам. Таким образом, количество запросов для отображения полноценной страницы пользователя социальной сети может потребовать несколько десятков, а то и сотен запросов к API. В случае большого количества числа единовременных пользователей такой системы, количество реальных запросов к API может достигать нескольких миллионов запросов в минуту, снижая общую производительность системы в целом. Для решение этой задачи, в рамках REST архитектуры, необходимо обеспечить, либо оптимизацию соответствующей структуры конечных точек, либо оптимизировать количество запросов на сервер API.

В случае использования графовых моделей извлечение всех требуемых данных происходит одновременно, посредством обращения к единой точке входа API. Таким образом графовая модель позволяет оптимизировать процесс извлечения данных. Графовая модель предоставляет набор методов создания, извлечения данных, которые имплементированы в виде графа, а службы являются узлами этого графа. Данный граф также может включать в себя дополнительные элементы, такие как метаданные, элементы конфигурации, дополнительные службы и т.д. Структура и способы формирования данного графа определяется соответствующей спецификацией, поддерживающую соответствующую графовую модель.

В настоящее время многими технологическими компаниями, такими как Facebook, Coursera, PayPal, Twitter, GitHub используется GraphQL. Многие компании, с целью преодоления технологических недостатков архитектуры REST, разработали собственные, чем-то схожие, но в тоже время сильно различающиеся решения, такие как, например, falcor от Netfliх[7]. Данный пример очень показателен тем, что в дальнейшем Netflix отказалась от использования falcor, отдав предпочтение GraphQL[8]. При этом, простота миграции с одной технологии на другую стала возможной, благодаря тому, что контракт между клиентом и сервером в рамках технологии GraphQL определяется, непосредственно, схемой SDL(англ. Schema Definition Language), которая и определяет соответствующие бизнес-процессы. Таким образом технология GraphQL можно использовать для унификации всех систем в единый API.

**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.

**Программный код:**

from typing import List, Optional

from fastapi import FastAPI

from strawberry.fastapi import GraphQLRouter

import strawberry

@strawberry.type

class Book:

    id: int

    name: str

    genre: str

    author: str

books = [

    Book(id=1, name="Journey to the Unknown", genre="Adventure", author="Mark Shumakher"),

    Book(id=2, name="Whispers of the Night", genre="Romance", author="Kate Johnson"),

    Book(id=3, name="The Silent Forest", genre="Thriller", author="Anna Taylor"),

    Book(id=4, name="Cosmos Unveiled", genre="Science", author="Carl Vane"),

    Book(id=5, name="The Art of Code", genre="Programming", author="John Doe"),

    Book(id=6, name="Shadows of the Past", genre="Historical Fiction", author="Emily Green"),

    Book(id=7, name="Adventures in AI", genre="Sci-Fi", author="Sophia K. Brown"),

    Book(id=8, name="The Hidden Truth", genre="Mystery", author="Robert Lane"),

]

@strawberry.type

class Query:

    @strawberry.field

    def book\_by\_id(self, id: int) -> Optional[Book]:

        return next((book for book in books if book.id == id), None)

    @strawberry.field

    def books(self) -> List[Book]:

        return books

@strawberry.type

class Mutation:

    @strawberry.mutation

    def create\_book(self, name: str, genre: str, author: str) -> Book:

        new\_book = Book(id=len(books) + 1, name=name, genre=genre, author=author)

        books.append(new\_book)

        return new\_book

    @strawberry.mutation

    def update\_book(self, id: int, name: Optional[str] = None, genre: Optional[str] = None, author: Optional[str] = None) -> Optional[Book]:

        book = next((book for book in books if book.id == id), None)

        if book:

            if name is not None:

                book.name = name

            if genre is not None:

                book.genre = genre

            if author is not None:

                book.author = author

        return book

    @strawberry.mutation

    def delete\_book(self, id: int) -> str:

        global books

        # Найти книгу перед изменением списка

        book\_exists = any(book.id == id for book in books)

        if book\_exists:

            books = [book for book in books if book.id != id]

            return f"Book with ID {id} was successfully deleted"

        else:

            return "Book not found"

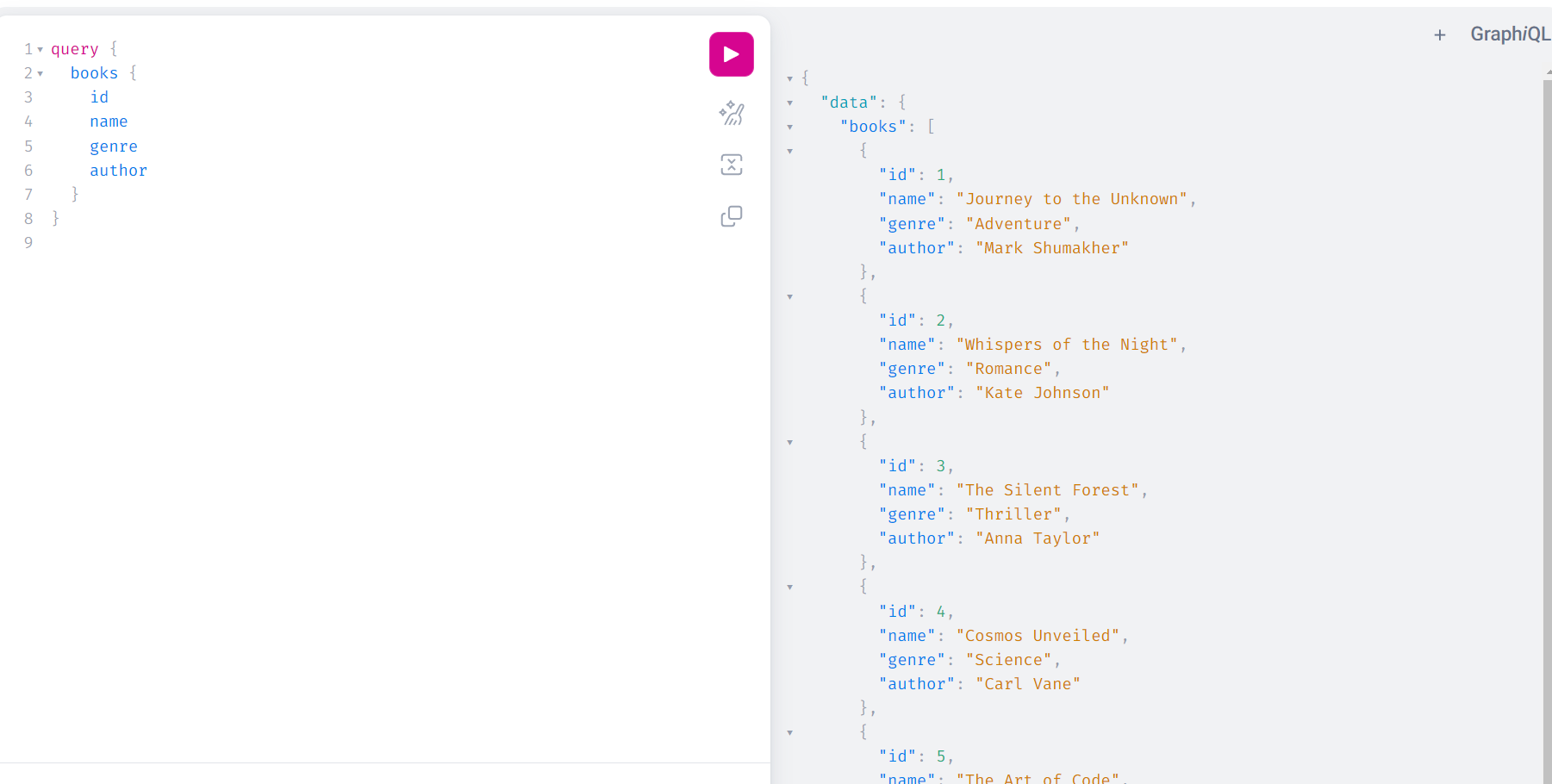
schema = strawberry.Schema(query=Query, mutation=Mutation)

app = FastAPI()

graphql\_app = GraphQLRouter(schema)

app.include\_router(graphql\_app, prefix="/graphql")

Вывод программы:



**Вывод**: в результате выполнения практической работы выполнено ознакомление с современным подходом к проектированию и реализации API на основе графовых моделей и с реализующей данный подход технологией на основе спецификации GraphQL.