МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий

КАФЕДРА ИНСТРУМЕНТАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиППО)

ПРАКТИЧЕСКАЯ РАБОТА №6
ПО ДИСЦИПЛИНЕ «Архитектура клиент-серверных приложений»

Выполнил студент группы ИКБО-13-19 Малютина В.А.

Принял Степанов П.В.

Практическая работа выполнена «___»_____2021г. *подпись студента*

«Зачтено» «___»_____2021г. *подпись преподавателя*

Москва 2021

Содержание

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.

Вышеизложенный пример демонстрирует существенный недостаток базовой архитектуры REST, а именно недостаточность выборки данных. Недостаточная выборка, точно также как избыточная выборка данных являются существенными недостатками данного архитектурного стиля. Так, при недостаточной выборке конечная точка не может предоставить все данные, необходимые для реализации пользовательского сценария, что было и продемонстрировано в вышеизложенном примере. Это обстоятельство приводит к необходимости повторных запросов к API. Для получения всей недостающей информации. Но. в то же время, избыточность выборки данных также является существенной проблемой, приводящей к лишним накладным расходам на передачу ненужных клиенту API данных.

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.

Одним из самых распространённых и широко используемых примеров спецификации графовой модели является GraphQL. Согласно официальному определения с сайта https://graphql.org/ [6] Graphql - «это язык запросов для API и среда выполнения для выполнения этих запросов с вашими существующими данными». Многими разработчиками GraphQL позиционируется как технология разработки прикладных программных интерфейсов, ориентированная на реализацию, а не на описание. GraphQL был разработан в рамках работы над социальной сетью Facebook и в

настоящее время представлен сообществу разработчиков, как отдельный шаблон взаимодействия в рамках фонда GraphQL Foundation.

В современых реалиях, GraphQL стоит рассматривать именно как технологию, включающую в себя, как спецификацию GraphQL Foundation, так и язык запросов с набором соответствующего инструментария.

Шаблон взаимодействия, по своей природе, описывает поведение сервера API, при этом сами данные и их структура, в отличии от подхода определённого архитектруным шаблоном REST, определяется не самим сервером, а непосредственно клиентом. Так в рамках полноценной REST архитектуры, требуется обеспечить принцип единственного унифицированного интерфейса, в соответствии с требованиями, изложенными в диссертационной работе Роя Филдинга. Унификация пользовательского интерфейса, отчасти, обеспечивается использованием в качестве механизма взаимодействия транспортного протокола HTTP. Современные, полноценные REST сервисы третьего уровня модели зрелости Ричардсона используют практически все возможности инфраструктуры HTTP. Использование 4 основных базовых глаголов HTTP, таких как GET, POST, PUT, и DELETE избавляет разработчика от необходимости придумывать свои методы для реализации базовых операций CRUD для каждого нового приложения, что также влияет на унификацию программного интерфейса и упрощает взаимодействие со множеством программных интерфейсов, реализованных в рамках архитектуры REST. В случае REST архитектуры идентификация ресурса происходит посредством использования URL, что позволяет, с совместным использованием протокола HTTP, реализовать полноценные механизмы кеширования. В случае же GraphQL, таких соглашений нет и это может накладывать определённые сложности, в том числе и на кеширование (обойти это ограничение возможно с помощью внедрения пакетной технологии обработки и кеширования, например

DataLoader, или документоориентированных СУБД, использующих NoSQL, таких как Redis).

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 Netflix[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. На рисунке 6.1 показана возможная схема организации данного программного интерфейса, как слоя интеграции.

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:

1. **Создание приложения для хранения списка книг в библиотеке.** Схема должна определять требуемые типы данных с полями, возвращающими определённые данные. Должны быть определены поля: name (название книги), genre(жанр книги), id (уникальный идентификатор книги), name (ФИО автора книги).

Дополнительные поля и соответствующие типы, если они будут нужны для решения данной задачи, определяются самостоятельно.

Цель работы:

Целью данной практической работы является 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.

Выполнение работы

Установлен Node js, инициализирован новый проект в папке graphql. Добавлены следующие библиотеки:

- Express – это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений;
- Graphql – сервер graphql, необходим для работы graphql
- Express-graphql – является одним из популярных серверных промежуточных программных средств для интеграции GraphQL с Node. Js;
- Lodash – упрощает работу с коллекциями.

Создан файл index.js со следующем кодом (Листинг 1).

Листинг 1 - index.js

```
const express = require('express');
const { graphqlHTTP } = require('express-graphql');
const port = 8888; // номер порта
const schema=require('./schema/schema');
const app = express(); // инициализация объекта приложения
```

```
    //выполнение серий функций req,res при совпадении
корневого пути
    app. use(
        '/graphql',
        graphqlHTTP({
            schema: schema,
            graphiql: true,
        })
    )
    app.listen(port);
```

Схема graphql реализована в файле schema.js в директории schema (Листинг 2).

Листинг 2 - schema.js

```
    const graphql=require('graphql');
    const _ = require('lodash');
    const{
        GraphQLID,
        GraphQLString,
        GraphQLList,
        GraphQLInt,
        GraphQLNonNull,
        GraphQLObjectType,
        GraphQLSchema
    }=graphql;
    const books = [
        { id: "1", name: "Книга", author:"1", year: "1923",
genre: "Ужасы"},
    ]

    const authors = [
        {
          id: "1",
          first_name: "Ivan1",
          last_name: "Иван"
        },
    ]


    const BookType = new GraphQLObjectType({
        name: "Book",
        fields: ()=>({
            id: {type: GraphQLID},
            name: {type: GraphQLString},
            author: {type: AuthorType,
                resolve(parent, args){
                    return
_.find(authors,{id:parent.authorId});
```

```
                }
            },
            year:{type: GraphQLString},
            genre:{type: GraphQLString}
        })
    });
    const AuthorType = new GraphQLObjectType({
        name:"Author",
        fields:()=>({
            id: {type: GraphQLID},
            first_name: {type: GraphQLString},
            last_name: {type: GraphQLString},
            book: {
                type : BookType,
                resolve(parent, args){
                    return
_.find(authors,{authorId:parent.id});
                }
            }
        })
    });
    const RootQuery = new GraphQLObjectType({
        name: 'RootQueryType',
        fields:{
            info:{
                type:GraphQLString,
                resolve(parent, args){
                    return "Сервер запущен"
                }
            },
            book:{
                type: BookType,
                args: {id:{type: GraphQLID}},
                resolve(parent, args){
                    return _.find(books, {id: args.id});
                }
            },
            books:{
                type: new GraphQLList(BookType),
                resolve(parent, args){
                    return books;
                }
            },
            authors:{
                type: AuthorType,
                args: {id: {type: GraphQLID}},
                resolve(parent, args){
                    return _.find(authors, {id: args.id});
                }
            }
        }
```

```
});
module.exports = new GraphQLSchema({
    query: RootQuery
});
```

Вывод программы

Выполнен запуск сервера, после чего выполнено обращение к конечной точке graphql (Рисунок 1).

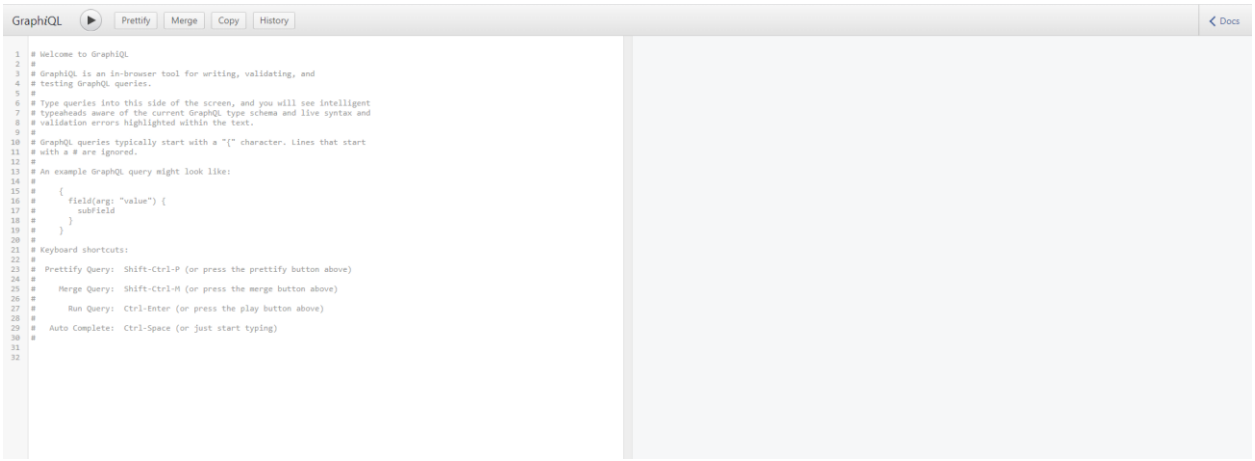

Рисунок 1 – Конечная точка /librarium

Выполнено тестирования доступных запросов к данным (Рисунок 2, Рисунок 3, Рисунок 4, Рисунок 5).

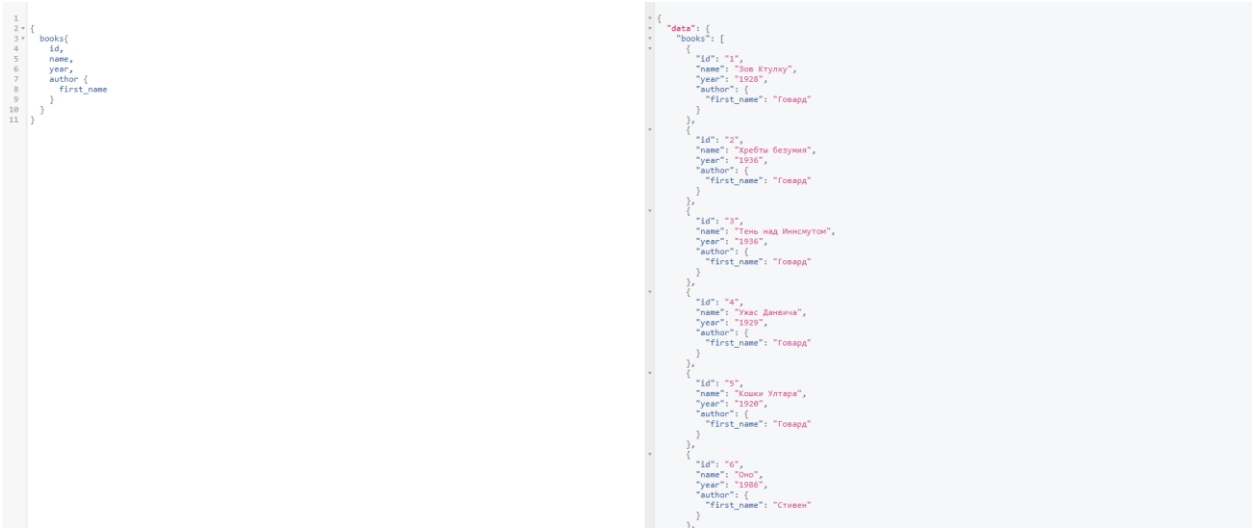

Рисунок 2 - books

Рисунок 3 - book()

Рисунок 4 – authors

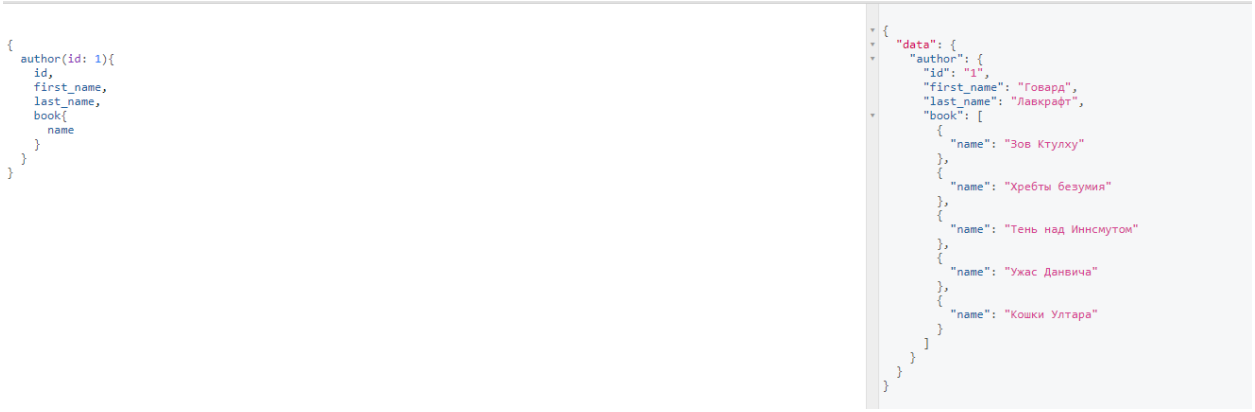

Рисунок 5 - author()

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы было создано клиент-серверное приложение с использованием технологии GraphQl. Добавлено две сущности и соответствующие запросы к ним. Доступ ко всем элементам происходит из общей конечной точки /librarium. Использован рекомендованный стек технологий.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Конспект лекций по дисциплине «АРХИТЕКТУРА КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ», РТУ МИРЭА, лектор – Алпатов А.Н.
2. GraphQl официальная документация [Электронный ресурс] – URL: https://graphql.org/ (Дата обращения 05.12.21)
3. Node js официальная документация [Электронный ресурс] – URL: https://nodejs.org/en/ (Дата обращения 05.12.21)