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
«GraphQL»

По дисциплине: «Архитектура клиент-серверных приложений»

Выполнил студент группы ИКБО-10-19 Дараган Ф.А.

Принял преподаватель Степанов П.В.

Практическая работы выполнена «___»_____2021 г.
(подпись студента)

«Зачтено» «___»_____2021 г.
(подпись руководителя)

Москва 2021

Оглавление

Практическая работа № 6 GraphQL

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.

Задание

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 (по выбору):

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.

Выполнение практической работы

На рисунке 1 представлена иерархия файлов проекта.

Рис. 1. Скриншот иерархии файлов проекта

Проект был реализован на фреймворке Spring, приложение представляло собой микросервис в другом моем проекте и может случайно содержать неиспользуемый код.

На листинге 1 показана GraphQL схема сервера.

Листинг 1. GraphQL схема

```
schema {
    query: Query
    mutation: Mutation
}

type UserBookDetails {
    username: String!
    books: [Book!]
    tagLists: [TagList!]
}

type TagList {
    userBookDetails: UserBookDetails!
    book: Book!
    tags: [Tag!]
}

type Author {
    id: ID!
    name: String!
    surname: String!
    middlename: String
    description: String
    birthDate: Date
    deathDate: Date
    books: [Book!]
}

type Book {
    id: ID!
    name: String!
    description: String
    publicationDate: Date
    authors: [Author!]
    usersBookDetails: [UserBookDetails!]!
    tagLists: [TagList!]
    tags: [Tag!]
}

type Tag {
    id: ID!
```

```
    name: String!
}

input TagCreateInput {
    name: String!
}

input BookCreateInput {
    username: String!
    name: String!
    description: String
    publicationDate: Date
    authorIds: [ID!]
    tagIds: [ID!]
}

input BookAddToUserInput {
    username: String!
    bookId: ID!
}

input BookRemoveFromUserInput {
    username: String!
    bookId: ID!
}

input BookAuthorAddInput {
    bookId: ID!
    authorId: ID!
}

input BookAuthorRemoveInput {
    bookId: ID!
    authorId: ID!
}

input BookTagAddInput {
    bookId: ID!
    tagId: ID!
}

input BookTagRemoveInput {
    bookId: ID!
    tagId: ID!
}

input AuthorCreateInput {
    name: String!
    surname: String!
```

```
    middlename: String
    description: String
    birthDate: Date
    deathDate: Date
    bookIds: [ID!]
}

input TagListAddInput {
    username: String!
    bookId: ID!
    tagId: ID!
}

input TagListRemoveInput {
    username: String!
    bookId: ID!
    tagId: ID!
}

type Query {
    # TAG
    getTag(tagId: ID!): Tag
    getAllTags: [Tag!]

    # BOOK
    getBook(bookId: ID!): Book
    getAllBooks: [Book!]

    # AUTHOR
    getAuthor(authorId: ID!): Author
    getAllAuthors: [Author!]

    # USER_BOOK_DETAILS
    getUserBookDetails(username: String!): UserBookDetails
    getAllUserBookDetails: [UserBookDetails!]
}

type Mutation {
    # TAG
    createTag(input: TagCreateInput!): Tag
    deleteTag(tagId: ID!): Tag
    # changeTagName

    # BOOK
    createBook(input: BookCreateInput!): Book
    deleteBook(bookId: ID!): Book
    addAuthorToBook(input: BookAuthorAddInput!): Book
    removeAuthorFromBook(input: BookAuthorRemoveInput!): Book
    addTagToBook(input: BookTagAddInput!): Book
```

```
    removeTagFromBook(input: BookTagRemoveInput!): Book
    # TODO add book changes

    # AUTHOR
    createAuthor(input: AuthorCreateInput!): Author
    deleteAuthor(authorId: ID!): Author
    # TODO add author changes

    # USER_BOOK_DETAILS
    deleteUserBookDetails(username: String!): UserBookDetails
    addBookToUser(input: BookAddToUserInput!): UserBookDetails
    removeBookFromUser(input: BookRemoveFromUserInput!):
UserBookDetails

    # USER_TAG_LIST
    addUserTagToBook(input: TagListAddInput!): TagList
    removeUserTagFromBook(input: TagListRemoveInput!): TagList
}

scalar Date
```

Резолверы для сущностей и полей Spring GraphQL генерирует автоматически, остается только написать резолверы для мутаций и запросов.

Рассмотрим класс, содержащий резолверы для всех операций, связанных с книгой, показанный на листинге 2.

Листинг 2. Класс BookResolver

```
@Controller
public class BookResolver {

    @Autowired
    BookService bookService;

    @Autowired
    UserBookDetailsService userBookDetailsService;

    @Autowired
    TagService tagService;

    @Autowired
    AuthorService authorService;

    @QueryMapping
    public List<Book> getAllBooks() {
        return bookService.getAll();
    }
```

```
    @QueryMapping
    public Book getBook(@Argument long bookId) {
        return bookService.getBookById(bookId);
    }

    @MutationMapping
    public Book createBook(@Argument BookCreateInput input) {

        // TODO replace to service ??

        // TODO catch error
        UserBookDetails userBookDetails =

userBookDetailsService.getOrCreateByUsername(input.getUsername()); //
TODO replace to only get

        if (userBookDetails == null)
            return null; // TODO add error

        Set<Author> authors = new HashSet<>();

        if (input.getAuthorIds() != null)
            authors = input.getAuthorIds().stream()
                    .map(authorService::getAuthorById) //TODO add
checks ?
                    .filter(Objects::nonNull)
                    .collect(Collectors.toSet());

        Set<Tag> tags = new HashSet<>();

        if (input.getTagIds() != null)
            tags = input.getTagIds().stream()
                    .map(tagService::getTagById) //TODO add checks ?
                    .filter(Objects::nonNull)
                    .collect(Collectors.toSet());

        Book book = Book.builder()
                        .name(input.getName())
                        .description(input.getDescription())
                        .users(
                            Set.of(userBookDetails)
                        )
                        .publicationDate(input.getPublicationDate())
                        .tags(tags)
                        .authors(authors)
                        .tagsMap(new HashMap<>())
                        .build();

        return bookService.saveBook(book);
```

```
    }

    @MutationMapping
    public Book deleteBook(@Argument long bookId) {
        return bookService.deleteBookById(bookId);
    }

    @MutationMapping
    public Book addAuthorToBook(@Argument BookAuthorAddInput input) {
        return bookService.addAuthorToBookById(
            input.getBookId(),
            input.getAuthorId()
        );
    }

    @MutationMapping
    public Book removeAuthorFromBook(@Argument BookAuthorRemoveInput
input) {
        return bookService.removeAuthorFromBookById(
            input.getBookId(),
            input.getAuthorId()
        );
    }

    @MutationMapping
    public Book addTagToBook(@Argument BookTagAddInput input) {
        return bookService.addTagToBookTagList(
            input.getBookId(),
            input.getTagId()
        );
    }

    @MutationMapping
    public Book removeTagFromBook(@Argument BookTagRemoveInput input)
{
        return bookService.removeTagFromBookTagList(
            input.getBookId(),
            input.getTagId()
        );
    }
}
```

Как видно из листинга 2, резолвер представляет собой почти что контроллер из MVC. Он осуществляет взаимодействие с сервисами, которые содержат бизнес-логику. В текущем примере только createBook метод содержит какую-то дополнительную логику. Но поскольку там осуществляется преобразование входных данных, то она имеет место быть в контроллере.

На рисунках 2 и 3 показаны несколько запросов к созданному приложению, с помощью консоли для тестирования запросов GraphiQL, встроенной в приложение.

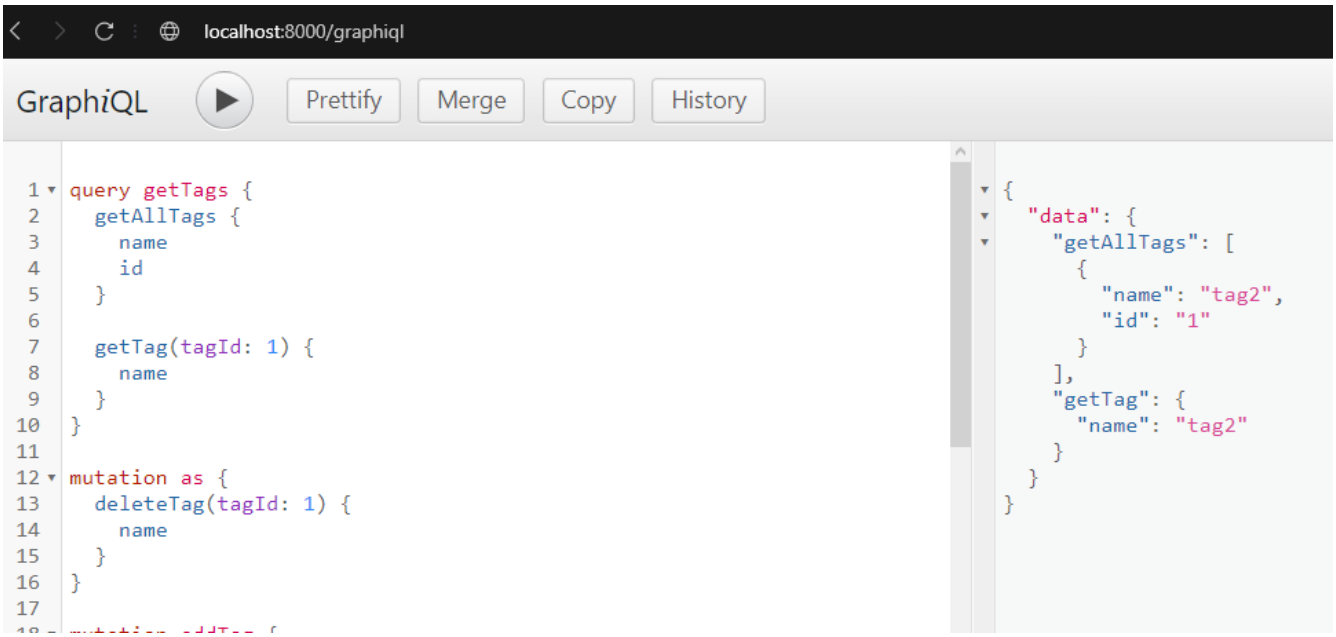

Рис. 2. Выполнение запроса getTags, после добавления тега

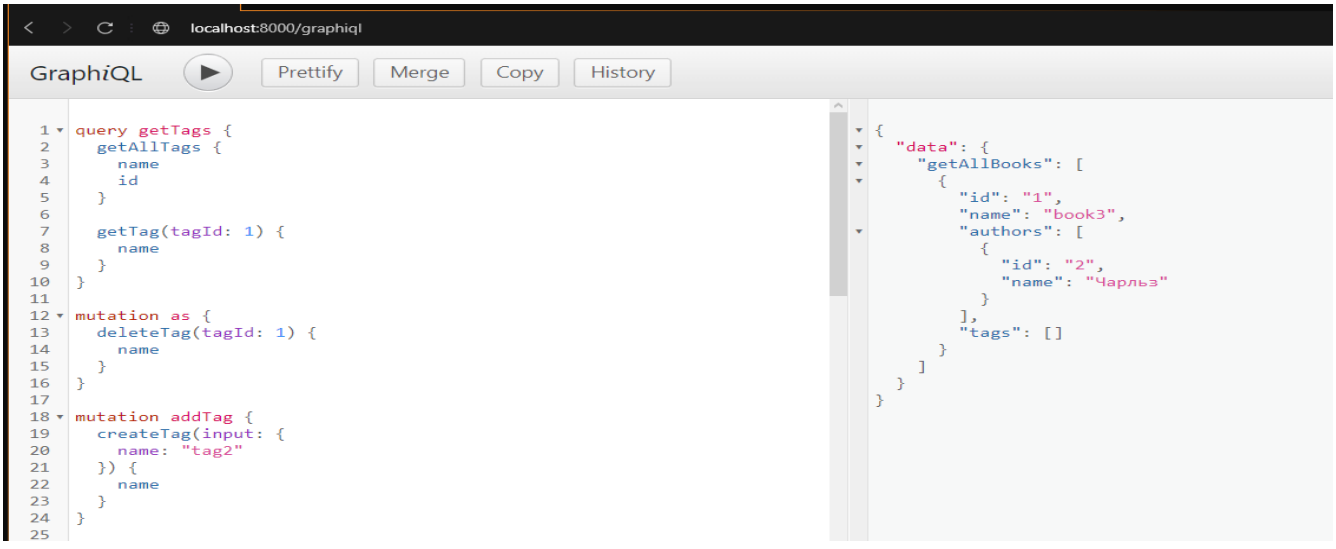

Рис. 3. Выполнение запроса books, после выполнения нескольких мутаций

Выводы по работе

В результате практической работы, мы познакомились GraphQL, научились создавать свои системы с использованием этого API, и писать запросы к данному API, а так же проверять его с помощью GraphiQL.

Используемая литература

1. Вязовик, Н. А. Программирование на Java : учебное пособие / Н. А. Вязовик. — 2-е изд. — Москва : ИНТУИТ, 2016. — 603 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100405 (дата обращения: 13.09.2021). — Режим доступа: для авториз. пользователей.
2. Наир, В. Предметно-ориентированное проектирование в Enterprise Java : руководство / В. Наир ; перевод с английского А. В. Снастина. — Москва : ДМК Пресс, 2020. — 306 с. — ISBN 978-5-97060-872-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/179503 (дата обращения: 13.09.2021). — Режим доступа: для авториз. пользователей.
3. Васильев, А. Н. Самоучитель Java с примерами и программами : учебное пособие / А. Н. Васильев. — 4-е, изд. — Санкт-Петербург : Наука и Техника, 2017. — 368 с. — ISBN 978-5-94387-745-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/101548 (дата обращения: 13.09.2021). — Режим доступа: для авториз. пользователей.