САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 3
Микросервисы

Выполнила:

Хорошкеева Ксения

Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

Необходимо реализовать отдельный микросервис, выполняющий какую-либо содержательную функцию из всего арсенала функций вашего приложения.

Ход работы

От основного приложения был отделен микросервис бронирования

Обновленная модель бронирования:

```
@DefaultScope( scopeGetter: () => ({
    attributes: ['id', 'startDate', 'endDate', 'roomId'],
}))
@Table
class Booking extends Model {
    @IsDate
    @AllowNull( allowNull: false)
    @Column(DataTypes.DATEONLY)
    startDate: any;

    @IsDate
    @AllowNull( allowNull: false)
    @Column(DataTypes.DATEONLY)
    endDate: any;

    @AllowNull( allowNull: false)
    @Column
    userId: number;

    @AllowNull( allowNull: false)
    @Column
    roomId: number;
```

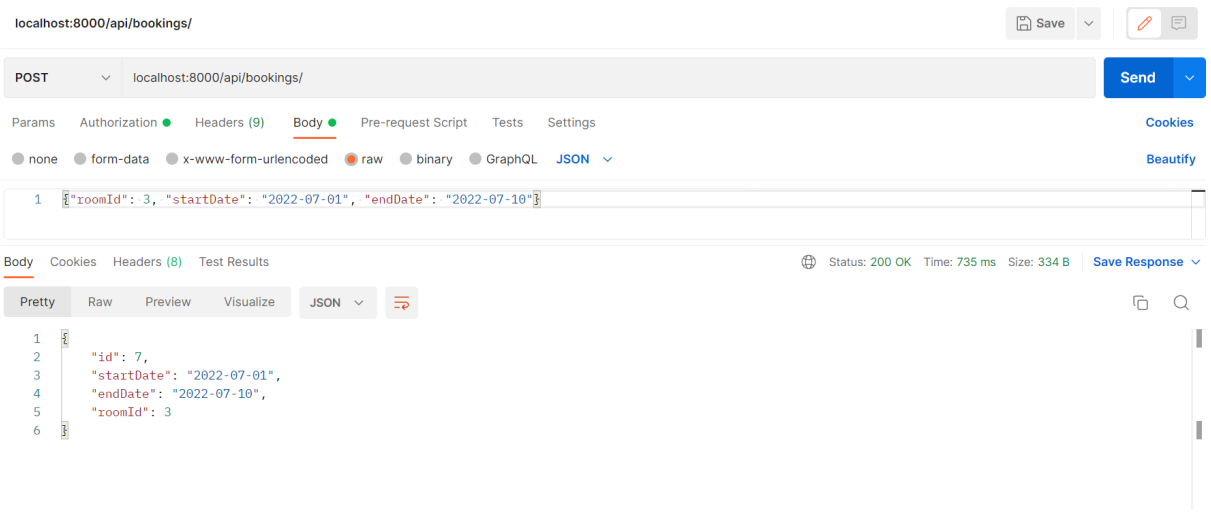
```
34
35          @BeforeValidate
36          static validateDates(instance: Booking) {
37              // Проверка правильности дат
38              if (instance.startDate >= instance.endDate) {
39                  throw new Error('Дата начала должна быть раньше даты конца');
40              }
41          }
42      }
43
44      export default Booking;
```

Контроллер для бронирования в главном микросервисе, который проксирует запросы к микросервису бронирования:

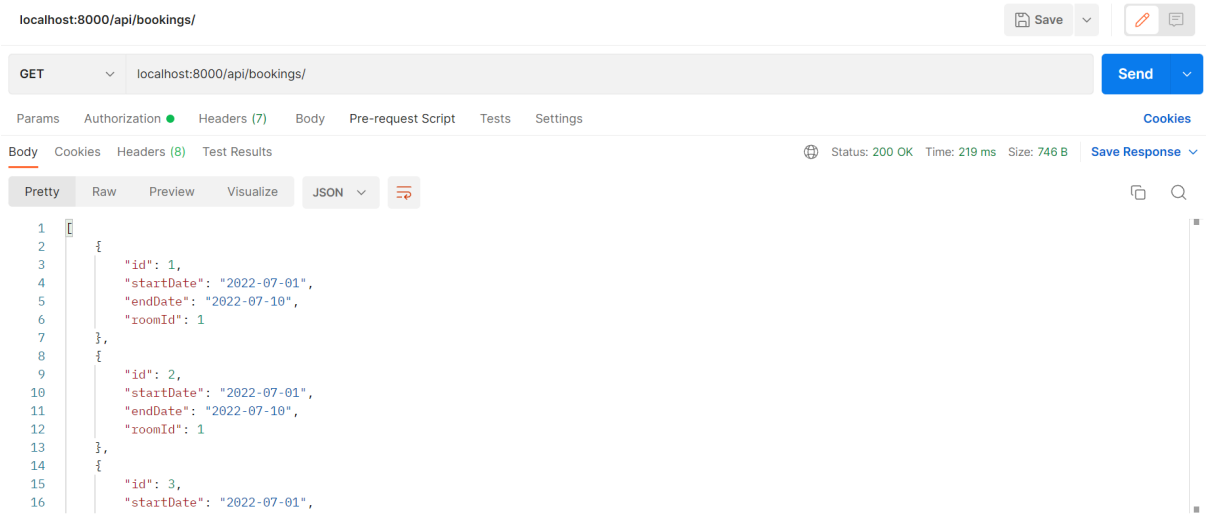
```
1   import axios from "axios";
2
3   class ProxyBookingController {
4       private async requestProxy(url: string, method: any, response: any, body: any = null) {
5           // Функция для отправки запросов другому сервису
6           axios( config: {
7               method: method,
8               url: `http://localhost:8001${url}`,
9               data: body
10          }).then((proxyResponse : AxiosResponse<any> ) =>{
11              response.status(proxyResponse.status).send(proxyResponse.data);
12          }).catch((proxyError) => {
13              if (proxyError.response) {
14                  response.status(proxyError.response.status).send(proxyError.response.data);
15              } else {
16                  response.status(500).send({"error": proxyError.message});
17              }
18          })
19      }
20
```

```
    get = async (request: any, response: any) => {
        // Получение конкретного бронирования пользователя
        const id = request.params.id;
        const userId = request.user.id;
        await this.requestProxy( url: `/api/bookings/${id}?userId=${userId}`, method: 'get', response);
    }

    list = async (request: any, response: any) => {
        // Список бронирований пользователя
        const userId = request.user.id;
        await this.requestProxy( url: `/api/bookings?userId=${userId}`, method: 'get', response);
    }

    create = async (request: any, response: any) => {
        // Создание бронирования
        const body = request.body;
        const userId = request.user.id;
        await this.requestProxy( url: `/api/bookings?userId=${userId}`, method: 'post', response, body);
    }
}

export default ProxyBookingController;
```

Проверка создания бронирования:



Проверка списка бронирований:



Вывод

Я познакомилась с микросервисной архитектурой веб-приложений и выделила часть функционала API сервиса поиска отелей в отдельный микросервис.