САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №4

Выполнил:
Ребров С. А.

Группа:
K3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- реализовать Dockerfile для каждого сервиса;
- написать общий docker-compose.yml;
- настроить сетевое взаимодействие между сервисами.

Ход работы

В ходе лабораторной работы была реализована микросервисная архитектура из трёх независимых сервисов: user-service, job-service и app-service. Для каждого из них был прописан Dockerfile:

```
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
CMD ["node", "dist/index.js"]
```

А также был написан общий docker-compose.yml:

```
services:
  postgres:
    image: postgres:17
    container_name: postgres
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: microservices_db
```

```
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 3s
      timeout: 5s
      retries: 5

  app-service:
    build: ./app-service
    container_name: app-service
    ports:
      - "5000:5000"
    depends_on:
      postgres:
        condition: service_healthy
    env_file:
      - /.env

  job-service:
    build: ./job-service
    container_name: job-service
    ports:
      - "5001:5001"
    depends_on:
```

```
    postgres:
      condition: service_healthy
  env_file:
   - /.env

 user-service:
  build: ./user-service
  container_name: user-service
  ports:
   - "5002:5002"
  depends_on:
   postgres:
     condition: service_healthy
  env_file:
   - /.env

volumes:
 postgres_data:
```

Было настроено сетевое взаимодействие между сервисами. В частности, app-service обращается к user-service и job-service для проверки существования пользователя и вакансии перед созданием заявки. Для этого используются HTTP-запросы с помощью axios. Ниже приведён фрагмент кода из application.controller.ts:

```
const userServiceUrl = process.env.USER_SERVICE_URL;
const jobServiceUrl = process.env.JOB_SERVICE_URL;

let userExists = false;
```

```
    try {
      const userRes = await
axios.get(`${userServiceUrl}/api/users/${userId}`);
      userExists = userRes.status === 200;
    } catch {
      userExists = false;
    }

    if (!userExists) {
      res.status(400).json({ message: "Invalid userId — user not found" });
      return;
    }

    let jobExists = false;
    try {
      const jobRes = await axios.get(`${jobServiceUrl}/api/jobs/${jobId}`);
      jobExists = jobRes.status === 200;
    } catch {
      jobExists = false;
    }
```

После чего приложение было протестировано:

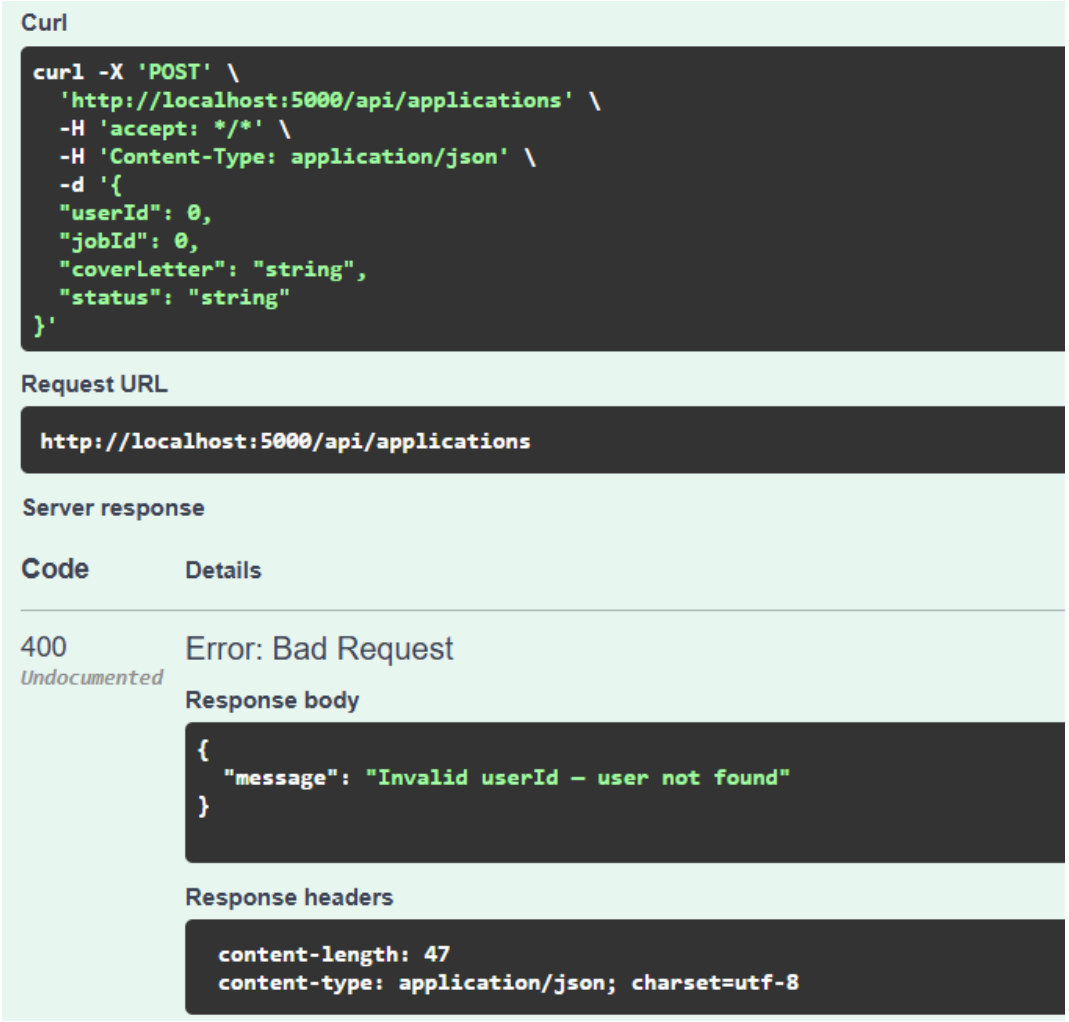

Рисунок 1 – передача userId = 0

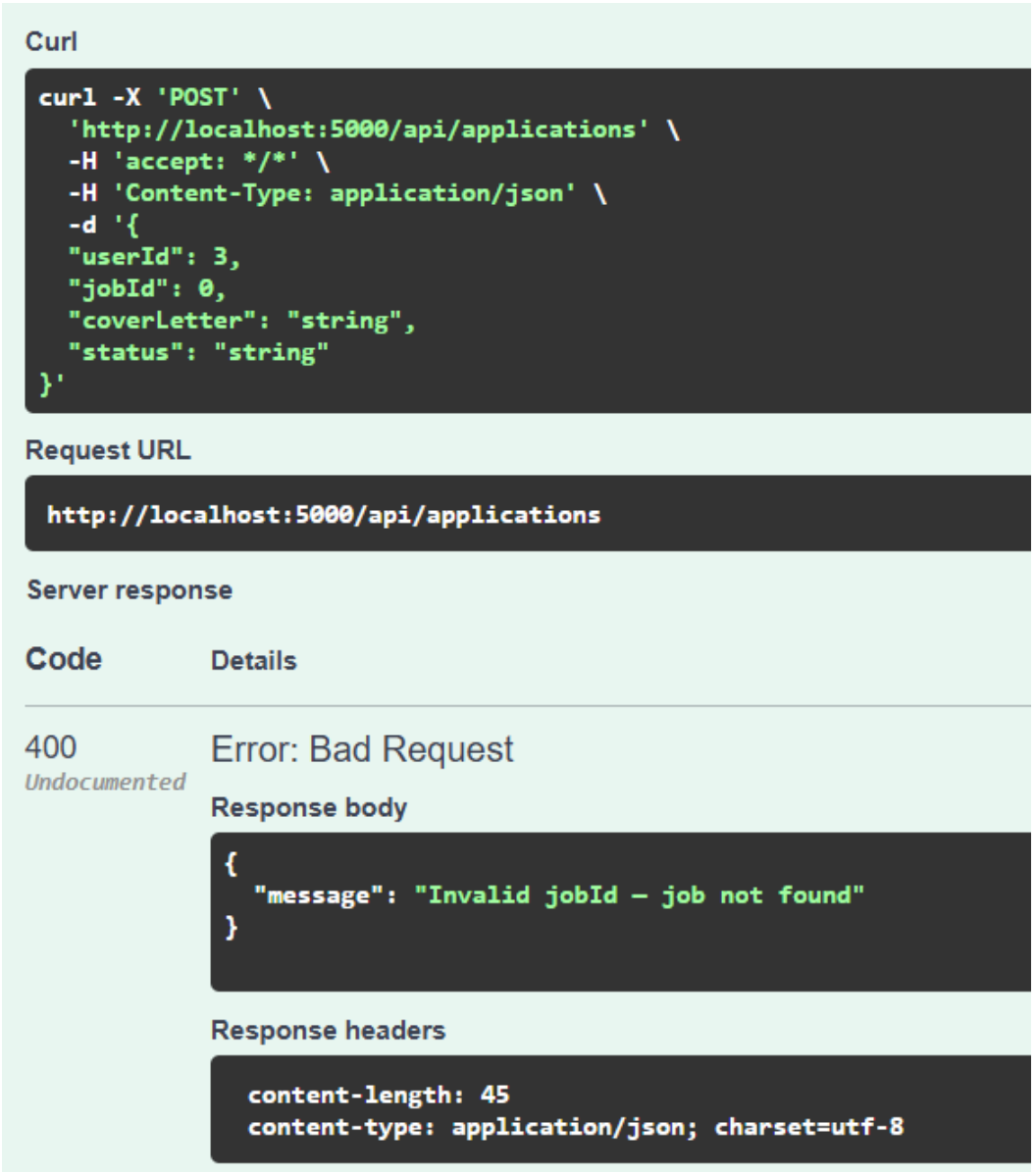

Рисунок 2 – передача jobId = 0

Curl

```
curl -X 'POST' \
  'http://localhost:5000/api/applications' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "userId": 3,
  "jobId": 2,
  "coverLetter": "string",
  "status": "string"
}'
```

Request URL

```
http://localhost:5000/api/applications
```

Server response

Code	Details
201	Response body

```
{
  "user": 3,
  "job": 2,
  "coverLetter": "string",
  "status": "string",
  "id": 1,
  "createdAt": "2025-06-23T09:37:46.438Z",
  "updatedAt": "2025-06-23T09:37:46.438Z"
}
```

Response headers

```
content-length: 144
content-type: application/json; charset=utf-8
```

Рисунок 3 – успешно обработанный запрос

Вывод

В ходе работы была успешно реализована микросервисная архитектура, включающая три независимых сервиса: для пользователей, вакансий и откликов. Каждый из сервисов имеет собственную структуру, конфигурацию и маршруты, а взаимодействие между ними осуществляется через HTTP-запросы. Все сервисы были упакованы в Docker и объединены с помощью docker-compose, что позволило упростить развертывание системы. Работа показала преимущества разделения ответственности между сервисами и повысила гибкость и масштабируемость приложения.