

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №4

Выполнил:

Корчагин Вадим

Группа
К3341

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Тема

Платформа для фитнес-тренировок и здоровья.

Задача

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

Ход работы

Dockerfile

Для каждого из следующих сервисов был создан собственный Dockerfile:

- auth-service
- blog-service
- order-service
- progress-service
- workout-service

Каждый Dockerfile:

- использует образ node:23;
- копирует исходный код внутрь контейнера;
- устанавливает зависимости через pnpm;

```
1 FROM node:23-alpine
2
3 WORKDIR /app
4
5 COPY package*.json ./
6 COPY pnpm-lock.yaml ./
7 RUN npm install -g pnpm && pnpm install
8
9 COPY . .
10
11 ENV NODE_ENV=development
12 CMD ["pnpm", "dev"]
```

docker-compose.yml

Создан единый файл docker-compose.yml, который:

- поднимает базу данных PostgreSQL с указанием имени пользователя, пароля и тома для хранения данных;
- запускает все микросервисы как отдельные контейнеры;
- задаёт зависимости (depends_on) от PostgreSQL;
- задаёт порты, переменные окружения, монтирует конфигурации .env файлов;
- использует единый bridge-сеть по умолчанию для связи контейнеров по их container_name.
- запускает миграции с помощью скриптов из package.json;
- стартует сервис с помощью скриптов из package.json.


```
docker-compose.yml ×
labs > lab4 > docker-compose.yml > ...
  ▷ Run All Services | You, 2 hours ago | 1 author (You)
1  services:
  ▷ Run Service
2  blog-service:
3    build:
4      context: ../lab3/src/microservices/blog-service
5    container_name: blog-service
6    ports:
7      - "4004:4004"
8    volumes:
9      - ../lab3/src/microservices/blog-service:/app
10   env_file:
11     - ../lab3/src/microservices/blog-service/.env
12   depends_on:
13     - postgres
14   command: sh -c "pnpm migration:run && pnpm start"
74  postgres:
75    image: postgres:15
76    container_name: postgres
77    restart: always
78    environment:
79      POSTGRES_USER: postgres
80      POSTGRES_PASSWORD: postgres
81    volumes:
82      - pgdata:/var/lib/postgresql/data
83      - ./init.sql:/docker-entrypoint-initdb.d/init.sql:ro
84    ports:
85      - "5432:5432"
86
87  volumes:
88    pgdata:
```


Сетевое взаимодействие

Для настройки связи между сервисами:

- Каждый сервис получает подключение к базе PostgreSQL по host: postgres (а не localhost), благодаря Docker-сети.
- В переменные окружения (.env) вынесены HOST и PORT каждого сервиса.

```
TS orderService.ts ×
labs > lab3 > src > microservices > order-service > src > services > TS orderService.ts > OrderService
You, 2 hours ago | 1 author (You)
1 import { BaseService } from "../common/baseService";
2 import { Order } from "../entities/Order";
3 import axios from "axios";
4 import { config } from "dotenv";
5
6 config();
7
8 const host = process.env.AUTH_HOST;
9 const port = parseInt(process.env.AUTH_PORT);
You, 2 hours ago | 1 author (You)
10 export class OrderService extends BaseService<Order> {
11     constructor() {
12         super(Order);
13     }
14
15     async findAllWithRelations() {
16         const orders = await this.repository.find();
17         const enriched = await Promise.all(
18             orders.map(async (order) => {
19                 const user = await this.getUser(order.user_id);
20                 return { ...order, user };
21             })
22         );
23         return enriched;
24     }
25
26     You, 7 days ago • feat: lab3 is done
27     async findOneWithRelations(id: number) {
28         const order = await this.repository.findOne({ where: { id } });
29         if (!order) return null;
30         const user = await this.getUser(order.user_id);
31         return { ...order, user };
32     }
33
34     private async getUser(userId: number) {
35         const { data } = await axios.get(`http://${host}:${port}/users/id/${userId}`);
36         return data;
37     }
38 }
```

 .env.example M ✕

labs > lab3 > src > microservices > order-service >  .env.example

You, 13 seconds ago | 1 author (You)

```
1 DB_PASSWORD = "postgres"
2 DB_HOST = "postgres"
3 DB_PORT = "5432"
4 DB_USERNAME = "postgres"
5 DB_NAME = "order_service_db"
6
7 JWT_SECRET = "verystrongsecretstring2025"
8
9 POSTMAN_API_KEY = "фвфв4фвф"
10 COLLECTION_UID = "2вфцв"
11
12 HOST=order-service
13 PORT=4003
14 AUTH_HOST=auth-service
15 AUTH_PORT=4000
```

You, 1 second ago • Uncommitted changes

Пример поднятия контейнеров

При помощи команды ниже собираем контейнеры и поднимаем их:

```
docker compose up --build
```

```
✓ auth-service Built
✓ blog-service Built
✓ order-service Built
✓ progress-service Built
✓ workout-service Built
✓ Network lab4_default Created
✓ Volume "lab4_pgdata" Created
✓ Container postgres Created
✓ Container order-service Created
✓ Container auth-service Created
✓ Container blog-service Created
✓ Container progress-service Created
✓ Container workout-service Created
```

```
blog-service | Blog DB connected
progress-service | Progress DB connected
blog-service | 🚀 Blog service running at http://blog-service:4004
order-service | Order DB connected
workout-service | Workout DB connected
progress-service | 🚀 Progress service running at http://progress-service:4002
order-service | 🚀 Order service running at http://order-service:4003
workout-service | 🚀 Workout service running at http://workout-service:4001
auth-service | Auth DB connected
auth-service | 🚀 Auth service running at http://auth-service:4000
```

Выводы

В ходе работы была реализована полная контейнеризация микросервисного проекта. Каждый сервис получил собственный Dockerfile, а взаимодействие между ними организовано через единый docker-compose.yml. Все сервисы успешно подключаются к общей базе данных PostgreSQL и обмениваются данными через настроенную Docker-сеть.

Проект стал удобен для масштабирования, тестирования и развёртывания. Запуск всей системы теперь выполняется одной командой — быстро, изолированно и предсказуемо.