

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Лабораторные работы 3,4

Миграция написанного API на микросервисную  
архитектуру, Контейнеризация приложения средствами  
Docker

Выполнили:

Жижилева Арина  
Строганова Елизавета

К3342

Проверил:  
Добряков Д. И.

Санкт-Петербург

2025 г.

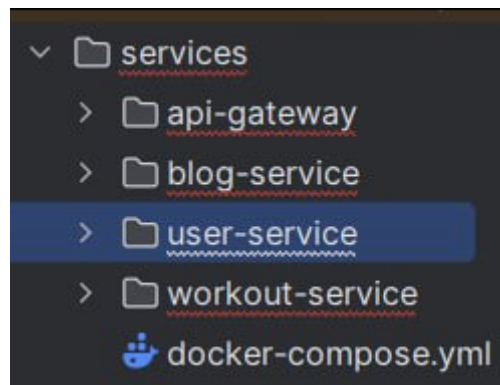
## ЛР3: Миграция написанного API на микросервисную архитектуру

### Задание:

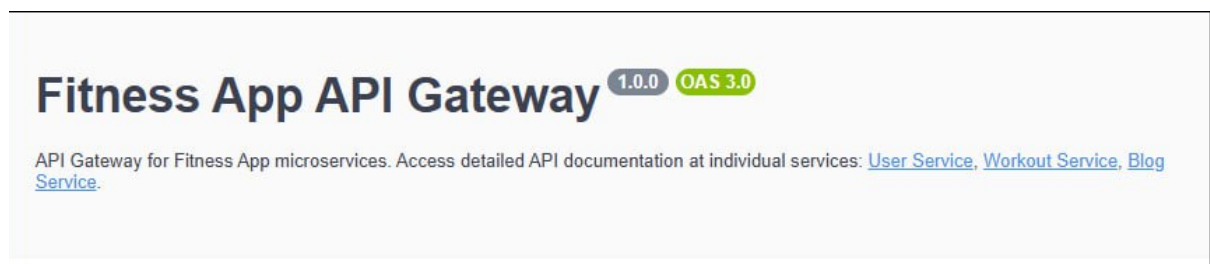
- выделить самостоятельные модули в вашем приложении;
- провести разделение своего API на микросервисы (минимум, их должно быть 3);
- настроить сетевое взаимодействие между микросервисами.

### Ход работы

#### 1. структура микросервисов



#### 2. основной порт с указанием маршрутов на документации остальных микросервисов



### 3. юзер сервис

# User Service API

1.0.0 OAS 3.0

API documentation for the User Service

ervers

http://localhost:3001 - User Service

Authorize

Users

POST /users Create a new user

GET /users Get all users

PUT /users/{id} Update a user by ID

DELETE /users/{id} Delete a user by ID


Auth

POST /auth/login Authenticate a user and return a JWT token

GET /auth/verify-token Verify a JWT token

UserMeasurementsProgress

## 4. воркаут сервис

 **Swagger**  
Supported by SMARTBEAR

# Fitness App API 1.0.0 OAS 3.0

API documentation for the Fitness App backend

**Servers**  
http://localhost:3000 - Development server

## Workouts

POST	/workouts	Create a new workout	⌵
GET	/workouts	Get all workouts	⌵
GET	/workouts/{id}	Get a workout by ID	⌵
PUT	/workouts/{id}	Update a workout by ID	⌵
DELETE	/workouts/{id}	Delete a workout by ID	⌵

## WorkoutPlans

POST	/plans	Create a new workout plan	⌵
GET	/plans	Get all workout plans	⌵
GET	/plans/{id}	Get a workout plan by ID	⌵
PUT	/plans/{id}	Update a workout plan by ID	⌵
DELETE	/plans/{id}	Delete a workout plan by ID	⌵

## 5. пост сервис

# Fitness App API

1.0.0 OAS 3.0

API documentation for the Fitness App backend

Servers

http://localhost:3000 - Development server

## PostComments

POST

/comment

Create a new comment

GET

/comment

Get all comments

GET

/comment/{id}

Get a comment by ID

PUT

/comment/{id}

Update a comment by ID

DELETE

/comment/{id}

Delete a comment by ID

## PostLikes

POST

/like

Create a new like

GET

/like

Get all likes

GET

/like/{id}

Get a like by ID

PUT

/like/{id}

Update a like by ID

DELETE

/like/{id}

Delete a like by ID

## PostTags

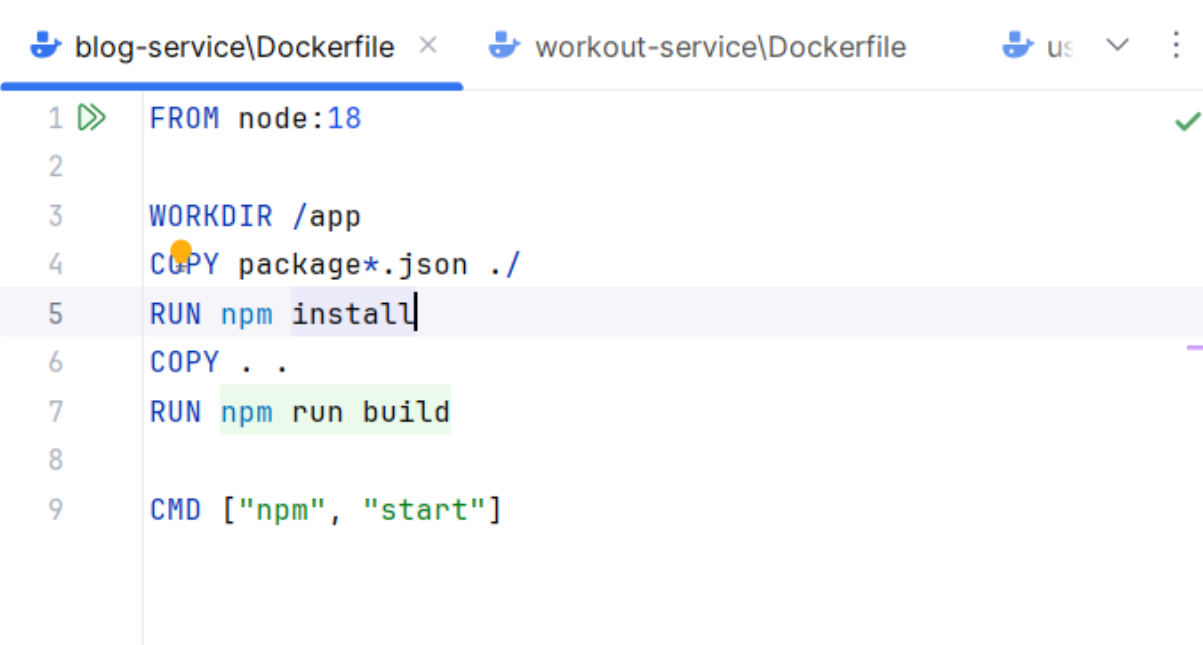
## ЛР4: Контейнеризация написанного приложения средствами docker

Задание:

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

### Ход работы:

Dockerfile



```
1 FROM node:18
2
3 WORKDIR /app
4 COPY package*.json ./
5 RUN npm install
6 COPY . .
7 RUN npm run build
8
9 CMD ["npm", "start"]
```

docker-compose.yml

```
version: "3.8"
services:
  user-service:
    build: ./user-service
    ports:
      - "3001:3001"
    environment:
      - DB_HOST=postgres
      - DB_PORT=5432
      - DB_USERNAME=postgres
      - DB_PASSWORD=admin
```

```

    - DB_NAME=user_service_db
    -
JWT_SECRET=7f9a8b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f
4
    - PORT=3001
    - RABBITMQ_URL=amqp://rabbitmq:5672
depends_on:
    - postgres
    - rabbitmq

workout-service:
  build: ./workout-service
  ports:
    - "3002:3002"
  environment:
    - DB_HOST=postgres
    - DB_PORT=5432
    - DB_USERNAME=postgres
    - DB_PASSWORD=admin
    - DB_NAME=workout_service_db
    - USER_SERVICE_URL=http://user-service:3001
    - PORT=3002
    - RABBITMQ_URL=amqp://rabbitmq:5672
  depends_on:
    - postgres
    - rabbitmq

blog-service:
  build: ./blog-service
  ports:
    - "3003:3003"
  environment:
    - DB_HOST=postgres
    - DB_PORT=5432
    - DB_USERNAME=postgres
    - DB_PASSWORD=admin
    - DB_NAME=blog_service_db
    - USER_SERVICE_URL=http://user-service:3001
    - PORT=3003
    - RABBITMQ_URL=amqp://rabbitmq:5672
  depends_on:
    - postgres
    - rabbitmq

api-gateway:
  build: ./api-gateway
  ports:
    - "3000:3000"
  environment:
    - PORT=3000

```

```

    - USER_SERVICE_URL=http://user-service:3001
    - WORKOUT_SERVICE_URL=http://workout-service:3002
    - BLOG_SERVICE_URL=http://blog-service:3003
    -
JWT_SECRET=7f9a8b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f
4
    - RABBITMQ_URL=amqp://rabbitmq:5672
  depends_on:
    - user-service
    - workout-service
    - blog-service

  postgres:
    image: postgres:13
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=admin
      - POSTGRES_DB=user_service_db
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"

  volumes:
    postgres_data:

```

Запускаем Docker Desktop

В терминале прописываем `docker-compose up --build`

```

[+] Building 8.9s (17/24)
=> CACHED [workout-service 2/6] WORKDIR /app
=> CACHED [blog-service 3/6] COPY package*.json ./
=> [blog-service 4/6] RUN npm install
=> CACHED [user-service 3/6] COPY package*.json ./
=> [user-service 4/6] RUN npm install
=> CACHED [workout-service 3/6] COPY package*.json ./
=> CACHED [workout-service 4/6] RUN npm install
=> [workout-service 5/6] COPY . .
=> [workout-service 6/6] RUN npm run build

```

В результате у нас три микросервиса `user-service`, `workout-service`, `blog-service`, доступные по портам 3001, 3002, 3003