**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

# Лабораторная работа 3
# Контейнеризация написанного приложения средствами docker

Выполнила:
Едигарева Дарья

Группа
К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

**Задача**

Реализовать Dockerfile для каждого сервиса;

Написать общий docker-compose.yml;

Настроить сетевое взаимодействие между сервисами.

**Ход работы**

1. Для каждого сервиса создан двухэтапный Dockerfile: установка зависимостей, генерация TSOA и сборка TypeScript, затем минимальный runtime с артефактами и Swagger. Здесь сборочный этап подготавливает все сгенерированные файлы до упаковки в финальный образ.
   Фрагмент /services/auth-service/Dockerfile:

```
FROM node:18-alpine as build

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run tsoa:routes && npm run tsoa:spec && npm run build
```

2. Сформирован docker-compose.micro.yml, который поднимает PostgreSQL, три сервиса и Nginx gateway; переменные окружения задают доступ к БД и JWT-секрет, контейнеры общаются по DNS-именам.
   Фрагмент docker-compose.micro.yml:

```
auth-service:
  build:
    context: ./services/auth-service
    dockerfile: Dockerfile
  container_name: auth_service
  environment:
    NODE_ENV: production
    APP_HOST: 0.0.0.0
    APP_PORT: 8001
    APP_PROTOCOL: http
    DB_HOST: db
```

```
    DB_PORT: 5432
    DB_USER: postgres
    DB_PASSWORD: postgres
    DB_NAME: postgres
    JWT_SECRET_KEY: secret
  depends_on:
    - db
  expose:
    - '8001'
```

Настройка указывает docker-compose использовать локальный Dockerfile и прокидывает параметры подключения к общей базе db.

3. Gateway маршрутизирует /api/<service>/… к нужным контейнерам, пробрасывая клиентские заголовки; через него доступны Swagger UI и защищённые эндпоинты.
   Фрагмент gateway/nginx.conf:1:

```
location /api/auth/ {

  proxy_pass http://auth_service:8001/;

  proxy_set_header Host $host;

  proxy_set_header X-Real-IP $remote_addr;

  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

}
```

Прокси перенаправляет трафик к сервису авторизации и сохраняет данные о клиенте в заголовках.

4. JWT-проверка организована на middleware-уровне: Bearer-токен из заголовка декодируется и прикрепляется к запросу.
   Фрагмент services/profiles-service/src/middlewares/auth.ts:

```
const auth = request.headers.authorization;

if (!auth || !auth.startsWith('Bearer ')) {

  throw new Unauthorized('No Bearer token provided');

}

const token = auth.slice(7);
```

```
try {

  const payload = jwt.verify(token, SETTINGS.JWT_SECRET_KEY) as JwtPayload;

  (request as any).user = payload;
```

Этот код блокирует доступ без валидного токена и передаёт в контроллер идентификатор пользователя.

**Вывод:**
Контейнеризация всех сервисов REST API: каждый Node.js-сервис собирается собственным Dockerfile, запускается через docker-compose.micro.yml, обращается к единой PostgreSQL и маршрутизируется через Nginx gateway. JWT-аутентификация отрабатывает на middleware-уровне, это гарантирует защищённый доступ к приватным эндпоинтам при работе через общий gateway.