

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

**ЛР 5 - Миграция написанного API на микросервисную  
архитектуру**

Выполнил:  
Сергеев Виктор  
К3441

Проверил:  
Добряков Д. И.

Санкт-Петербург

2026 г.

## Задача

- выделить самостоятельные модули в вашем приложении;
- провести разделение своего API на микросервисы (минимум, их должно быть 3);
- настроить сетевое взаимодействие между микросервисами;
- реализовать Dockerfile для каждого сервиса;
- написать общий docker-compose.yml.

## Ход работы

В приложении были выделены следующие самостоятельные модули:

- сервис работы с пользователем - информация о нём + авторизация;
- сервис работы с рецептами - хранение рецептов, ингредиентов, тегов и шагов;
- социальный сервис - лайки и комментарии;
- gateway - точка входа, перенаправляя запросы от клиента в разные сервисы.

Был создан новый проект, в котором будет содержаться миросервисное приложение. В проекте было создано 4 директории для каждого сервиса и создано окружения со всеми библиотеками. Также, каждый сервис должен собственную базу данных, поэтому в был создан докер-композ, в котором поднимается 3 контейнера с postgres и в каждой создаётся своя база данных.

```
services:
  ▷ Run Service
  user_db:
    container_name: user-db
    image: postgres:15
    restart: always
    shm_size: 128mb
    env_file:
      - "user_service/.env.postgres"
    networks:
      - backend-microservices-network
    ports:
      - 5433:5432
    volumes:
      - pgdata-user:/var/lib/postgresql@15/data
```

Рисунок 1 - Пример контейнера с базой данных для сервиса пользователей

Поскольку теперь приложение разделено, как и хранилище данных, требуется поменять модели и дто, чтобы модели не хралини другие модели, о которых в рамках своего сервиса они не знают. Вместо этого в моделях будут хранится ключи на эти модели в явном виде.

```
social_service > src > models > ts Comment.ts > Comment
1   import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";
2
3   @Entity()
4   export class Comment {
5
6     @PrimaryGeneratedColumn()
7     id: number
8
9     @Column({ type: "integer" })
10    user_id: number
11
12    @Column({ type: "integer" })
13    recipe_id: number
14
15    @Column({type: "varchar", length: 200})
16    comment: string
17
18    @Column({type: "timestamp", update: false, default: () => "CURRENT_TIMESTAMP"})
19    created_at: Date
20 }
```

Рисунок 2 - Модель комментария без зависимостей

Остальные модули остались без изменений - проект просто был разделён на несколько частей, но сохранил практически всю свою функциональность.

Чтобы упаковать каждый сервис в контейнер, были написаны Dockerfile-ы для каждого из них. Логика Dockerfile-а следующая:

- берётся образ ноды, который будет выступать загрузчиком зависимостей, в который загружаются package\*.json файлы и запускается загрузка библиотек
- берётся образ ноды, в котором уже будет поднимать приложение, в него копируются из загрузчика установленные зависимости, копируются исходники приложения и оно запускается

```
user_service > 🐳 Dockerfile > ...
1  FROM node:20 AS dependencies
2
3  ENV NODE_ENV=production
4
5  WORKDIR /app
6  COPY package.json package-lock.json ./
7  RUN npm ci --omit=optional
8
9  FROM node:20 AS prod
10
11 ENV NODE_ENV=production
12
13 WORKDIR /app
14 COPY . .
15 COPY --from=dependencies /app/node_modules ./node_modules
16
17 EXPOSE 3000
18
19 CMD ["npm", "run", "prod"]
20
```

Рисунок 3 - Пример Dockerfile для сервиса пользователей

Для каждого сервиса Dockerfile практически идентичны.

Далее требовалось подключить эти сервисы через docker-compose.

Сперва была создана внутренняя сеть docker-a:

```
87
88 networks:
89   | backend-microservices-network:
90   |   | driver: bridge
```

Рисунок 4 - Сеть докера для приложения

Теперь подключаются контейнеризированные сервисы:

```
▶ Run Service
user_service:
  container_name: user-service
  build: ./user_service
  restart: always
  ports:
    - 3001:3000
  env_file:
    - ./user_service/.env.prod
  depends_on:
    - user_db
  networks:
    - backend-microservices-network
```

Рисунок 5 - Пример подключения сервиса в компоузе

| CONTAINER ID | IMAGE   | COMMAND   | C    |
|--------------|---|---|------|
| CREATED      | STATUS  | PORTS   | NAME |
| de4db6613f21 | backend-express-microservices-gateway_service | "docker-entrypoint.s..." 4                          | gate |
| minutes ago  | Up 47 minutes                                 | 0.0.0.0:3000→3000/tcp, [::]:3000→3000/tcp           |      |
| 9894f2fd9532 | backend-express-microservices-social_service  | "docker-entrypoint.s..." 5                          | soci |
| minutes ago  | Up 53 minutes                                 | 3000/tcp  |      |
| l-service    |   |   |      |
| 45ac8fbce513 | backend-express-microservices-user_service    | "docker-entrypoint.s..." 5                          | user |
| minutes ago  | Up 53 minutes                                 | 3000/tcp  |      |
| service      |   |   |      |
| 3dde4a979bb8 | backend-express-microservices-recipe_service  | "docker-entrypoint.s..." 5                          | reci |
| minutes ago  | Up 53 minutes                                 | 3000/tcp  |      |
| e-service    |   |   |      |
| 63c535b08bb8 | rabbitmq                                      | "docker-entrypoint.s..." 3                          | mess |
| hours ago    | Up 3 hours                                    | 4369/tcp, 5671-5672/tcp, 15691-15692/tcp, 25672/tcp |      |
| ge-broker    |   |   |      |
| 83870a863961 | postgres:15                                   | "docker-entrypoint.s..." 2                          | soci |
| hours ago    | Up 3 hours                                    | 5432/tcp  |      |
| l-db         |   |   |      |
| 9c71b8138725 | postgres:15                                   | "docker-entrypoint.s..." 2                          | reci |
| hours ago    | Up 3 hours                                    | 5432/tcp  |      |
| e-db         |   |   |      |
| 5b5927531f63 | postgres:15                                   | "docker-entrypoint.s..." 2                          | user |
| hours ago    | Up 3 hours                                    | 5432/tcp  |      |
| db           |   |   |      |

Рисунок 6 - Список запущенных контейнеров

## Вывод

В процессе работы ранее написанное приложение было разделено на самостоятельные модули, которые были разделены в отдельные микросервисы. Каждый микросервис представляет собой отдельное приложение на node JS.

В процессе работы ранее написанные микросервисы приложения были упакованы в контейнеры с помощью средств docker. Был написан общий компоуз файл, который поднимает всё приложение вместе с базами данных. В компоуз файле была настроена внутренняя сеть докера для осуществления сетевого взаимодействия между сервисами.