САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №5

Выполнил:

Ананьев Никита

К3440

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

Миграция написанного API на микросервисную архитектуру

Ход работы

Было решено разбить монолитное приложение на 3 компоненты (см. рисунок 1):

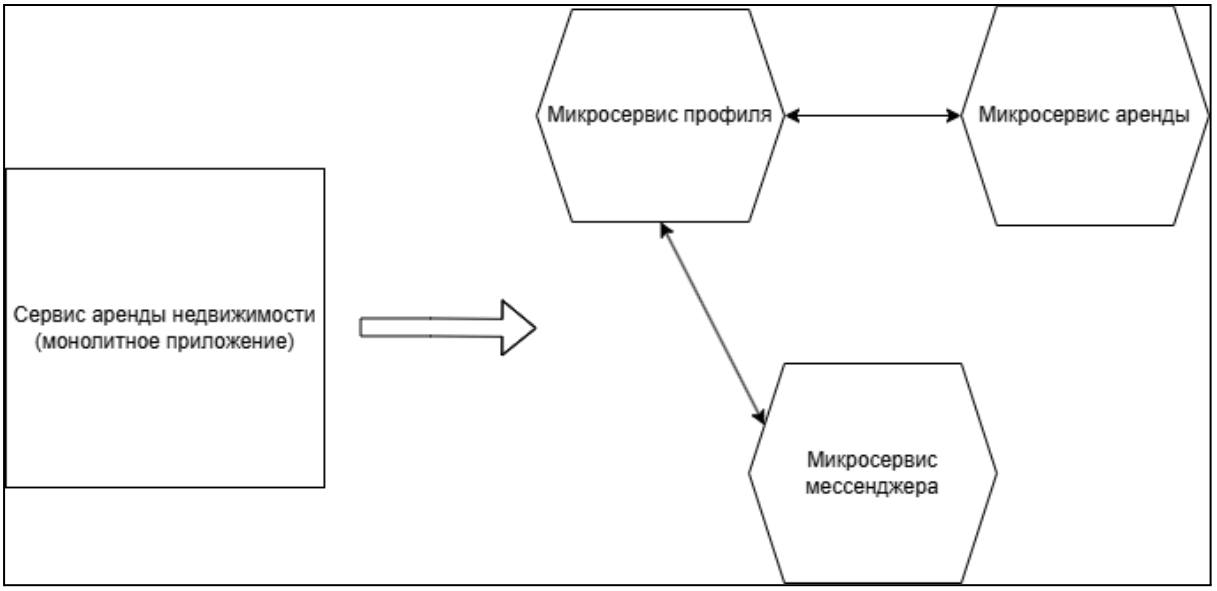


Рисунок 1 - Схема разбиения монолита

Файлы исходного проекта были разделены на 3 каталога, каждый из которых отвечает за свой сервис (см. рисунок 2):

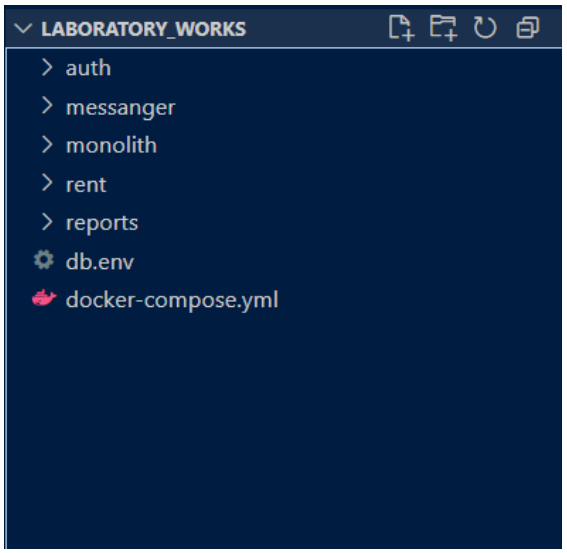


Рисунок 2 - структура папок

Так как мы имеем дело с распределенной системой, для каждого из ее узлов нам требуется собственное хранилище данных и свой сервер, обрабатывающий подключения. Поэтому создаем для всех сервисов по DataSource и ExpressServer (см. рисунок 3 и 4):

```
8   const envFile = path.resolve(__dirname, '../auth.env');
9   dotenv.config({path: envFile})
10
11
12  export const AppDataSource = new DataSource({
13      type: "postgres",
14      host: process.env.DB_HOST,
15      port: parseInt(process.env.DB_PORT as string),
16      username: process.env.DB_USER,
17      password: process.env.DB_PASSWORD,
18      database: process.env.DB_NAME,
19      entities: [User],
20      synchronize: true,
21      logging: true,
22  })
23
```

Рисунок 3 - Пример DataSource для сервиса профиля

```
auth > src > TS index.ts > ...
1   import { App } from "./app";
2
3   const app = new App();
4   const port = Number(process.env.APP_PORT) ?? 8001;
5
6   (async () => {
7       await app.init();
8       app.listen(port);
9   })();
10
```

Рисунок 4 - Пример запуска сервера для сервиса профиля

Для запуска сервиса в докере, необходимо написать файл создания образа или же Dockerfile (см. рисунок 1):

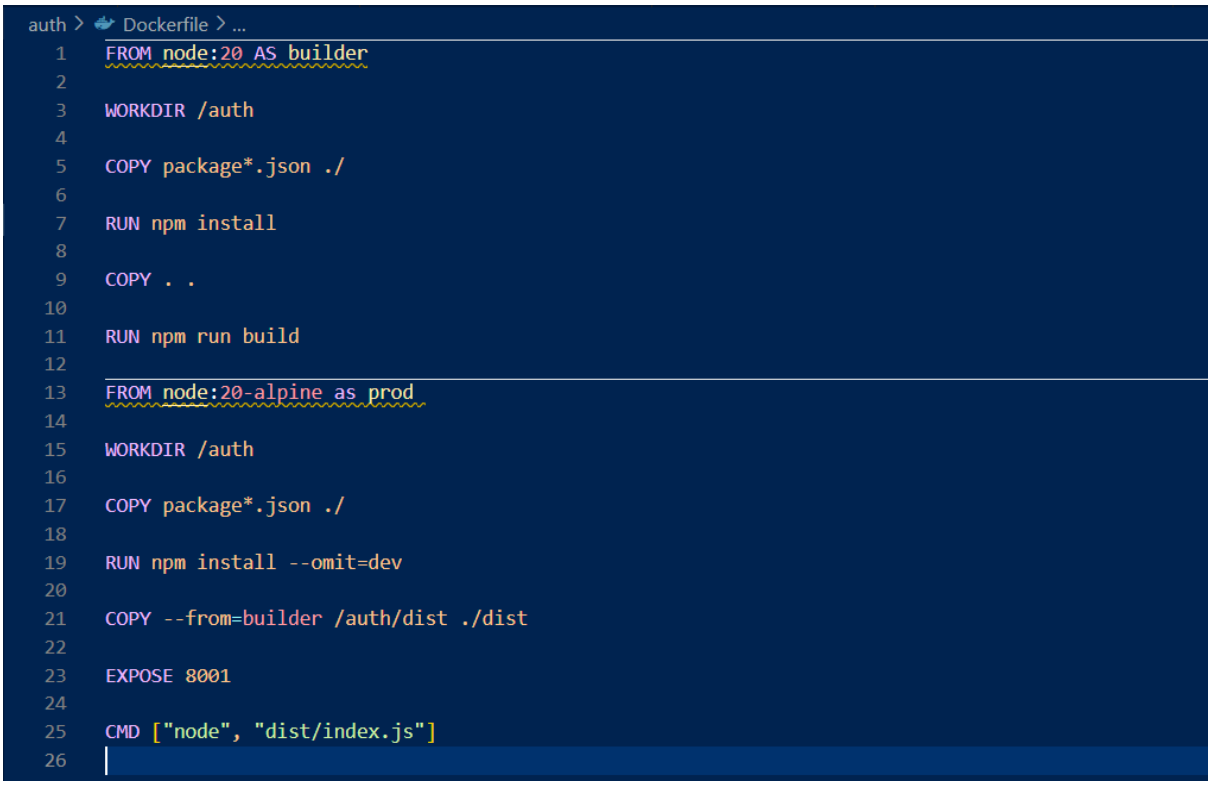
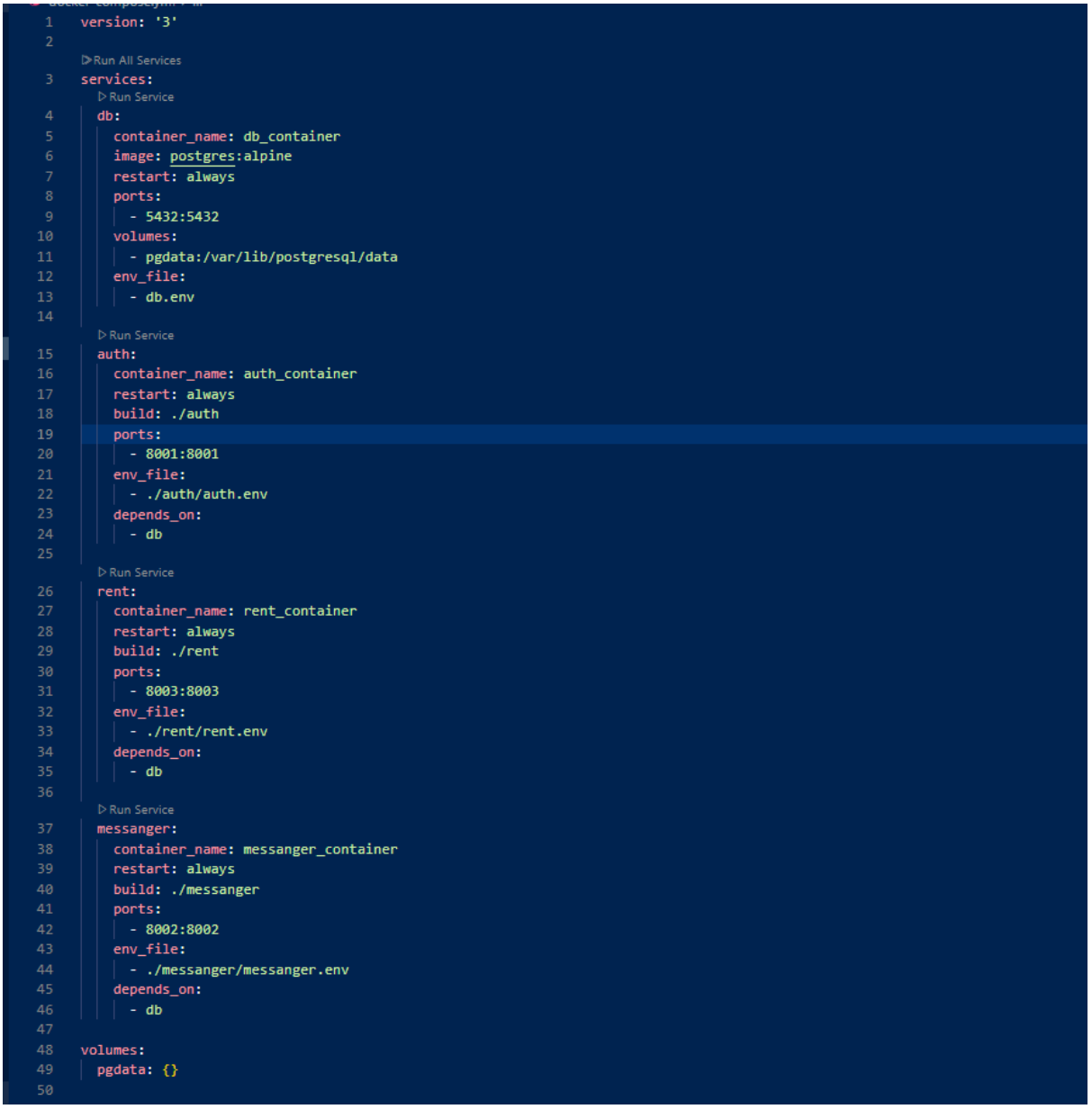


Рисунок 1 - пример Dockerfile для сервиса профиля

Используем multistage сборку - на первом этапе собираем наше приложение в единый компактный js файл, на втором берем только этот файл и нужные для работы приложения зависимости. Открываем нужный нам порт, чтобы в дальнейшем можно было общаться с контейнером по сети, запускаем наше приложение.

Примерно такие же действия необходимо повторить и для остальных сервисов

Для удобного запуска и настройки сетевого взаимодействия между несколькими контейнерами используется инструмент docker compose. Чтобы воспользоваться им, необходимо создать и описать наши контейнеры в файле с названием docker-compose.yml (см. рисунок 2):



```yaml
version: '3'

Run All Services
services:
  Run Service
  db:
    container_name: db_container
    image: postgres:alpine
    restart: always
    ports:
      - 5432:5432
    volumes:
      - pgdata:/var/lib/postgresql/data
    env_file:
      - db.env

  Run Service
  auth:
    container_name: auth_container
    restart: always
    build: ./auth
    ports:
      - 8001:8001
    env_file:
      - ./auth/auth.env
    depends_on:
      - db

  Run Service
  rent:
    container_name: rent_container
    restart: always
    build: ./rent
    ports:
      - 8003:8003
    env_file:
      - ./rent/rent.env
    depends_on:
      - db

  Run Service
  messanger:
    container_name: messanger_container
    restart: always
    build: ./messanger
    ports:
      - 8002:8002
    env_file:
      - ./messanger/messanger.env
    depends_on:
      - db

volumes:
  pgdata: {}
```

Рисунок 2 - docker compose файл

Для упрощения был создан один контейнер postgres (но с разными бд). В настоящей системе с микросервисной архитектурой это был бы отдельный (или даже несколько) контейнер на каждый сервис.

Вывод

Разбиение монолитного приложения на микросервисы позволяет удобно сегментировать работу над проектом (особенно если идет работа в команде), естественным образом защищает от высокой связности компонент, что позволяет строить более чистую архитектуру, а также способствует масштабированию всей системы.