**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

# **Дисциплина:** Бэк-энд разработка

Отчет
Лабораторная работа №4

Выполнил:

Екушев Владислав

К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

## Задача

Необходимо упаковать ваше приложение в docker-контейнеры и обеспечить сетевое взаимодействие между различными частями вашего приложения, а также настроить общение микросервисов между собой посредством RabbitMQ.

## Ход работы

Реализовано в предыдущей лабораторной работе средствами NestJS.

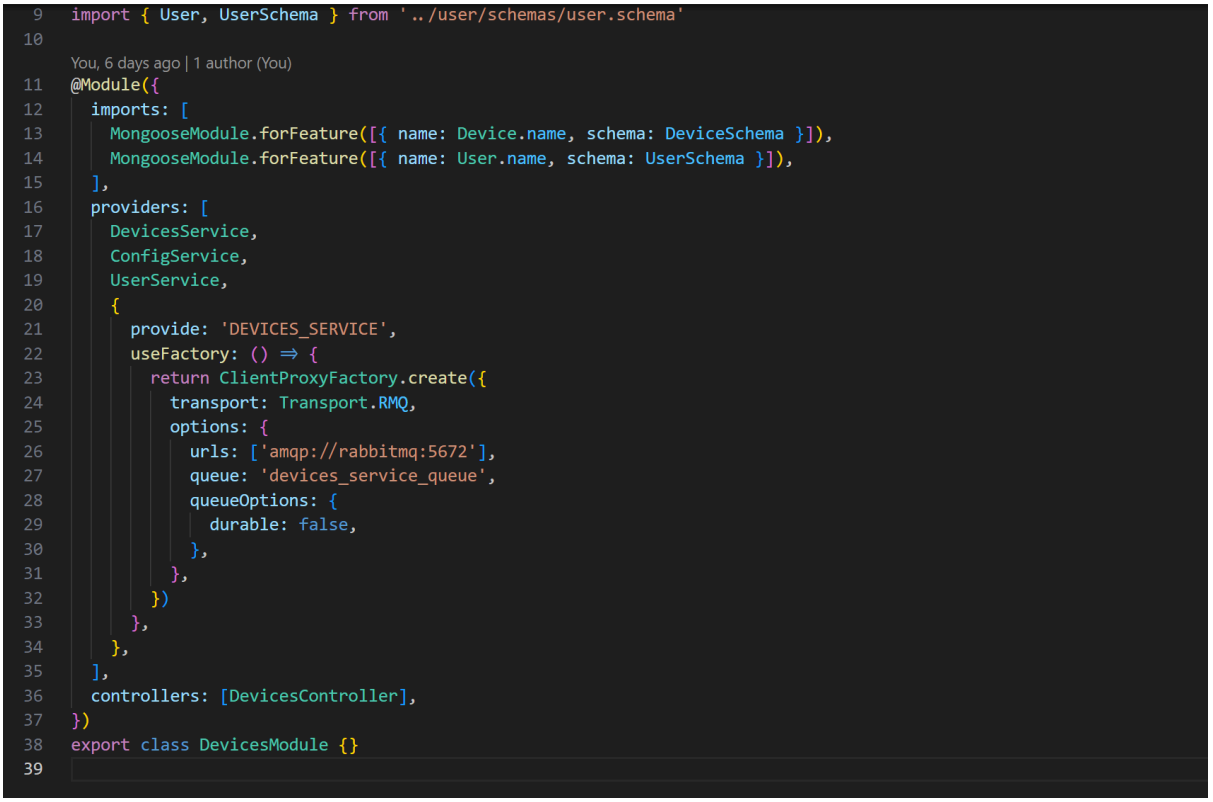
apps > devices > src > main.ts > ...

You, 6 days ago | 1 author (You)

```ts
import { Logger } from '@nestjs/common'
import { NestFactory } from '@nestjs/core'
import { AppModule } from './app/app.module'
import { MicroserviceOptions, Transport } from '@nestjs/microservices'

async function bootstrap() {
  const app = await NestFactory.createMicroservice<MicroserviceOptions>(
    AppModule,
    {
      transport: Transport.RMQ,
      options: {
        urls: ['amqp://rabbitmq:5672'],
        queue: 'devices_service_queue',
        queueOptions: {
          durable: false,
        },
      },
    }
  )

  await app.listen()
  Logger.log('🚀 Devices service is running')
}

bootstrap()
```

```ts
  app.connectMicroservice<MicroserviceOptions>({
    transport: Transport.RMQ,
    options: {
      urls: ['amqp://rabbitmq:5672'],
      queue: 'main_app_queue',
      queueOptions: {
        durable: false,
      },
    },
  })
```

Для того, чтобы использовать микросервис девайсов на бекенде, нужно включить его в провайдеры в модуле девайсов:



```
import { User, UserSchema } from '../user/schemas/user.schema'

You, 6 days ago | 1 author (You)
@Module({
  imports: [
    MongooseModule.forFeature([{ name: Device.name, schema: DeviceSchema }]),
    MongooseModule.forFeature([{ name: User.name, schema: UserSchema }]),
  ],
  providers: [
    DevicesService,
    ConfigService,
    UserService,
    {
      provide: 'DEVICES_SERVICE',
      useFactory: () => {
        return ClientProxyFactory.create({
          transport: Transport.RMQ,
          options: {
            urls: ['amqp://rabbitmq:5672'],
            queue: 'devices_service_queue',
            queueOptions: {
              durable: false,
            },
          },
        })
      },
    },
  ],
  controllers: [DevicesController],
})
export class DevicesModule {}

```

После этого можно обращаться к функциям сервиса внутри кода:

```
@Injectable()
export class DevicesService {
  constructor(
    @Inject('DEVICES_SERVICE') private readonly client: ClientProxy,
    private userService: UserService
  ) {}

  async getDevices(userId: string): Promise<IDevice[]> {
    return await firstValueFrom(
      this.client.send({ cmd: 'getDevices' }, { userId })
    )
  }
      You, 5 months ago • Add toggle on-off capability …
```

Dockerfile для бекенда и микросервиса девайсов (в нем меняется только последняя команда запуска сервиса на “devices:serve”):

```
You, 6 days ago | 1 author (You)
1  FROM node:20-alpine        You, 6 days ago • Add microservices
2
3  RUN apk update && apk add --no-cache openssl
4
5  WORKDIR /app
6
7  COPY ../../package.json ../../yarn.lock ./
8
9  RUN yarn install --frozen-lockfile
10
11 COPY ../../.env ./
12
13 COPY . .
14
15 CMD ["yarn", "nx", "run", "backend:serve"]
```

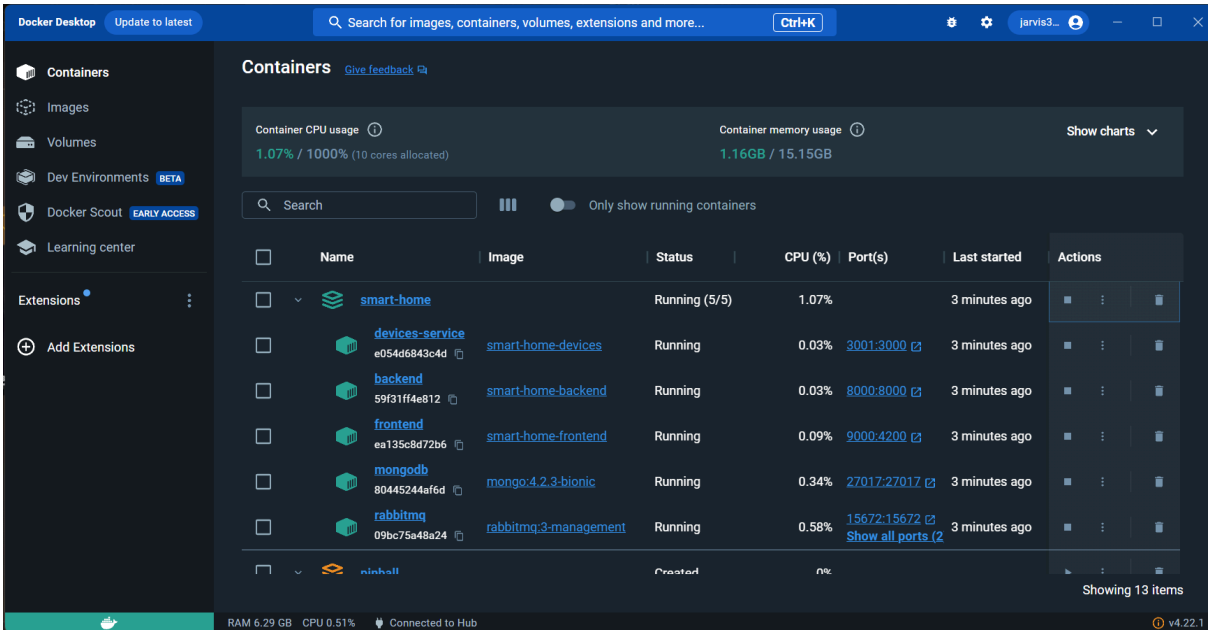
docker-compose.yml файл:

```
docker-compose.yml > {} services > {} frontend > [ ] networks
docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json) |
1  version: '3.8'
2  services:
3    devices:
4      container_name: devices-service
5      build:
6        context: .
7        dockerfile: apps/devices/Dockerfile
8      ports:
9        - '3001:3000'
10     networks:
11       - mongo_net
12
13   frontend:
14     container_name: frontend
15     build:
16       context: .
17       dockerfile: apps/frontend/Dockerfile
18     ports:
19       - '127.0.0.1:9000:4200'
20     networks:
21       - mongo_net
22       You, 6 days ago • Add microservices
23   backend:
24     container_name: backend
25     build:
26       context: .
27       dockerfile: apps/backend/Dockerfile
28     ports:
29       - '127.0.0.1:8000:8000'
30     networks:
31       - mongo_net
32     command: sh -c "yarn nx run backend:serve"
33
```

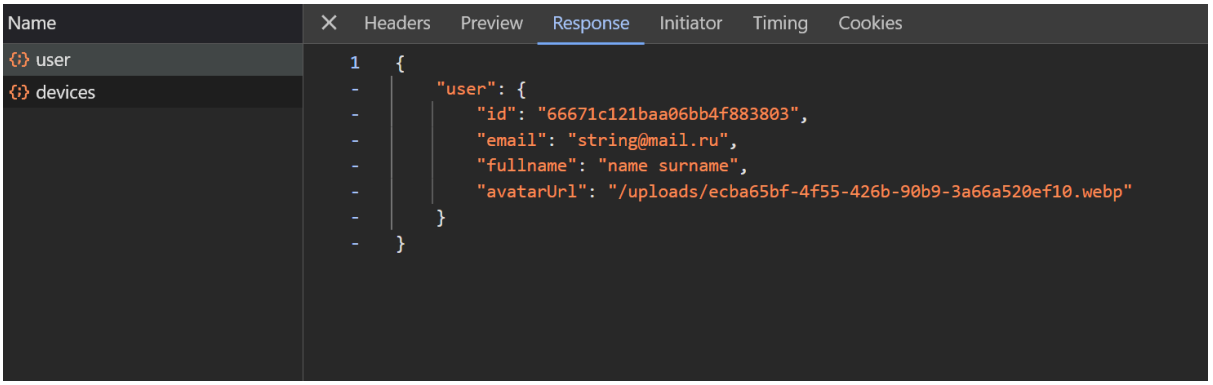
В этой части конфига мы поднимаем микросервисы и фронтенд приложения, не забывая пробросить порты наружу, чтобы заходить на сайт.

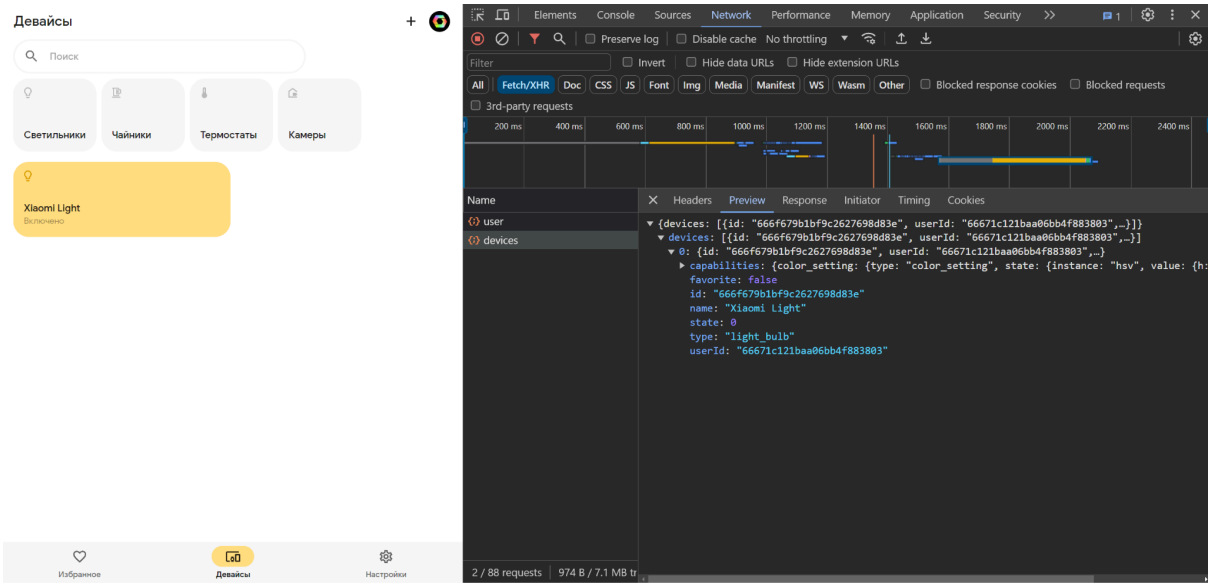
```yaml
  mongodb:
    image: mongo:4.2.3-bionic
    container_name: mongodb
    ports:
      - 27017:27017
    volumes:
      # seeding scripts
      - ./mongo-entrypoint:/docker-entrypoint-initdb.d
      # named volumes
      - mongodb:/data/db
      - mongoconfig:/data/configdb
    networks:
      - mongo_net

  rabbitmq:
    container_name: rabbitmq
    image: rabbitmq:3-management
    ports:
      - '5672:5672'
      - '15672:15672'
    networks:
      - mongo_net

volumes:
  # default dir on Ubuntu: /var/lib/docker/volumes
  mongodb:
  mongoconfig:

networks:
  mongo_net:
    driver: bridge
```

Здесь мы поднимаем RabbitMQ сервис и MongoDB, нужные для работы бекенда. Также мы задаем место сохранения данных для БД и задаем сеть, по которой общаются сервисы внутри Docker.



Проверим работоспособность сервисов:




Видим, что клиент получает список девайсов от бекенда – следовательно, микросервис успешно общается и передает данные главному приложению внутри Docker.

## Вывод

В ходе лабораторной работы микросервисы стали общаться по очереди RabbitMQ и запускаться через docker compose.

Результат представлен в репозитории – https://github.com/jarvis394/smart-home/tree/%40jarvis394/microservicesDocker