САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашнее задание № 5

Выполнила:
Хисаметдинова Д.Н.

Группа
K3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- подключить и настроить rabbitMQ/kafka;
- реализовать межсервисное взаимодействие посредством rabbitMQ/kafka.

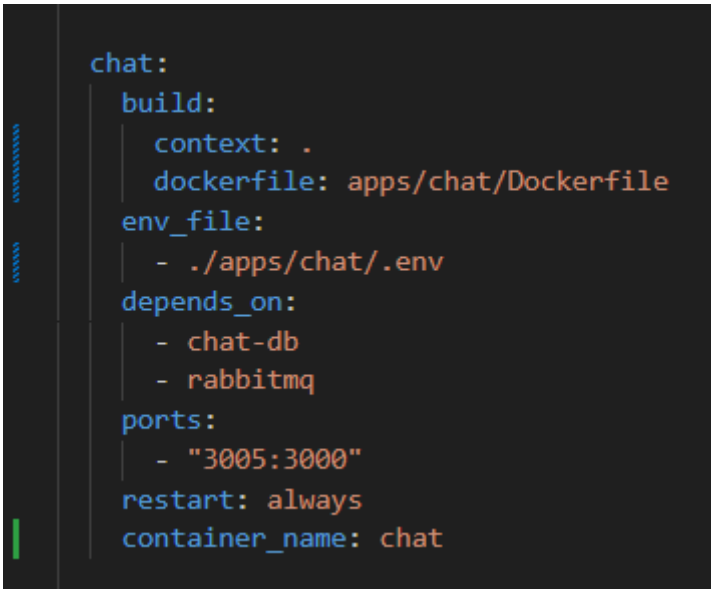
Ход работы

В рамках лабораторной работы реализована микросервисная архитектура на NestJS, состоящая из нескольких сервисов (user, psychologist, appointment, review, chat). Для организации межсервисного взаимодействия используется брокер сообщений RabbitMQ.

Docker Compose

В `docker-compose.yml` добавлен сервис RabbitMQ:

Каждый микросервис в docker-compose.yml получает зависимость от RabbitMQ через depends_on, например:



В .env каждого сервиса прописывается URL для RabbitMQ:
RABBITMQ_URL=amqp://rabbitmq:5672

```
pps > user > .env
    Import to Postman
1   DB_HOST=user-db
2   DB_PORT=5432
3   DB_USERNAME=postgres
4   DB_PASSWORD=postgres
5   DB_NAME=user
6   JWT_SECRET=supersecretkey
7   JWT_EXPIRES_IN=1h
8   RABBITMQ_URL=amqp://rabbitmq:5672
9
```

Установка зависимостей
В каждый сервис, где требуется работа с RabbitMQ, установлены необходимые пакеты:
npm install @nestjs/microservices amqplib amqp-connection-manager

Настройка подключения RabbitMQ в сервисе (пример: user)

4.1. main.ts

```
pps > user > src > TS main.ts > ...
2   import { AppModule } from './app.module';
3   import { MicroserviceOptions, Transport } from '@nestjs/microservices';
4
5   async function bootstrap() {
6     const app = await NestFactory.create(AppModule);
7
8     app.connectMicroservice<MicroserviceOptions>({
9       transport: Transport.RMQ,
10      options: {
11        urls: [process.env.RABBITMQ_URL || 'amqp://rabbitmq:5672'],
12        queue: 'users_queue',
13        queueOptions: { durable: false },
14      },
15    });
16
17    await app.startAllMicroservices();
18    await app.listen(3000);
19  }
20  void bootstrap();
21
```

После запуска docker-compose (docker-compose up), RabbitMQ доступен по адресу:
http://localhost:15672
(логин/пароль: guest/guest по умолчанию)

Итог

В результате работы:

- Настроен RabbitMQ в инфраструктуре проекта с помощью docker-compose.
- Во всех микросервисах прописано подключение к брокеру сообщений.
- Организовано межсервисное взаимодействие через команды и события RabbitMQ.

- Проверена работоспособность отправки и приёма сообщений между сервисами.

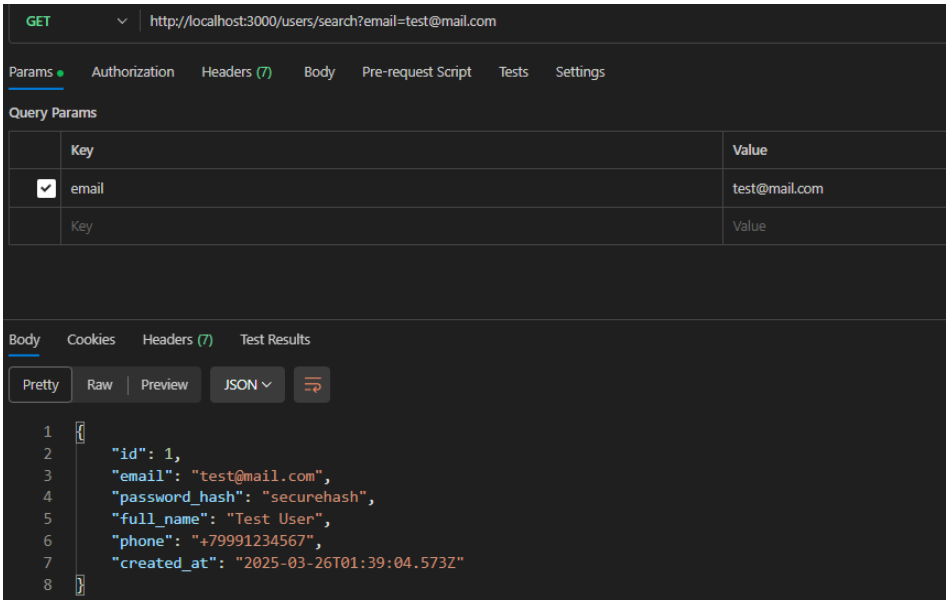
Структура

- docker-compose.yml (фрагмент с rabbitmq)
- main.ts (пример инициализации microservice через RMQ)
- package.json (раздел dependencies)

Пример реализации CRUD для модели Appointment

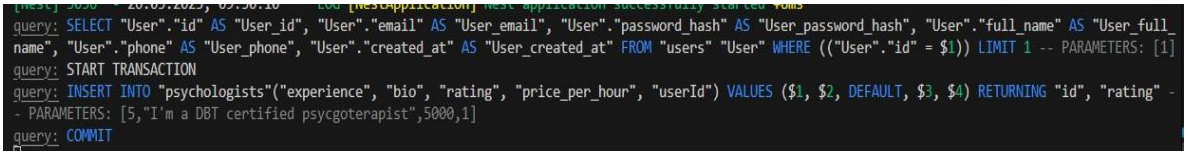
Была реализована сущность Appointment, DTO классы для создания и обновления записей, сервис и контроллер с методами:
- POST /appointments — создание встречи
- GET /appointments — получение всех
- GET /appointments/:id — получение по ID
- PUT /appointments/:id — обновление
- DELETE /appointments/:id — удаление

С помощью декораторов @ApiProperty и @ApiTags API документировано. Все эндпоинты и схемы запросов/ответов отображаются в Swagger UI по адресу http://localhost:3000/api.



Было выполнено задание реализовать API-эндпоинт для получения пользователя по id/email - показано на изображении выше.

Вот пример выполнения POST запроса в консоли:



Вывод

В ходе лабораторной работы были реализованы основные элементы серверной части приложения: модели, сервисы и контроллеры. Реализован полный набор CRUD-операций, обеспечивающий взаимодействие с базой данных. Работа API протестирована через Swagger.