

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Лабораторная работа № 3,4

Выполнила:  
Хисаметдинова Д.Н.

Группа  
К3341

Проверил:  
Добряков Д. И.

Санкт-Петербург

2025 г.

## Цели работы

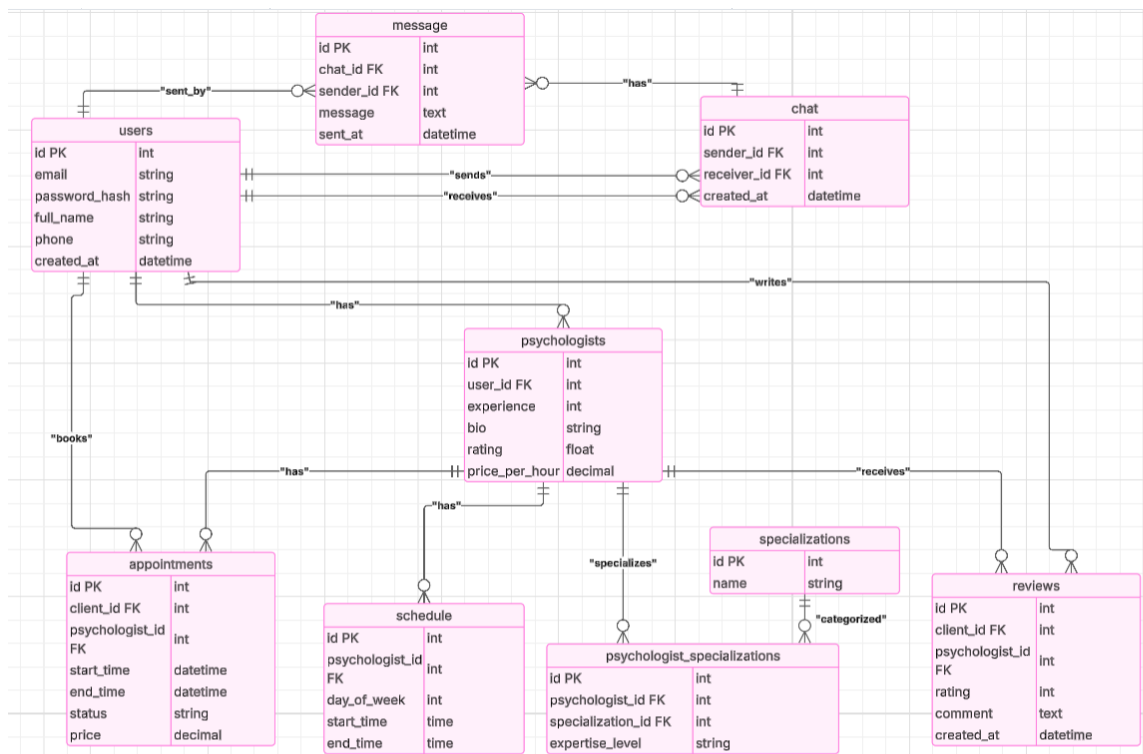
1. Выделить самостоятельные модули в приложении.
2. Провести разделение API на микросервисы (минимум 3).
3. Настроить сетевое взаимодействие между микросервисами.
4. Реализовать Dockerfile для каждого сервиса.
5. Написать общий docker-compose.yml.
6. Настроить сетевое взаимодействие между сервисами (RabbitMQ/Kafka)

## Структура проекта:

psychologist\_picking/

```
|
|
├── apps/
|   ├── user/
|   ├── psychologist/
|   ├── appointment/
|   ├── review/
|   └── chat/
├── libs/
|   └── shared/
├── docker-compose.yml
├── package.json
└── ...
```

- user — управление пользователями
- psychologist — работа с психологами
- appointment — записи на приём
- review — ОТЗЫВЫ
- chat — чат пользователей и психологов



## Разделение на самостоятельные модули

Каждый модуль (user, psychologist, appointment) — это отдельный NestJS-приложение со своей логикой, базой данных, контроллерами, сервисами и моделями.

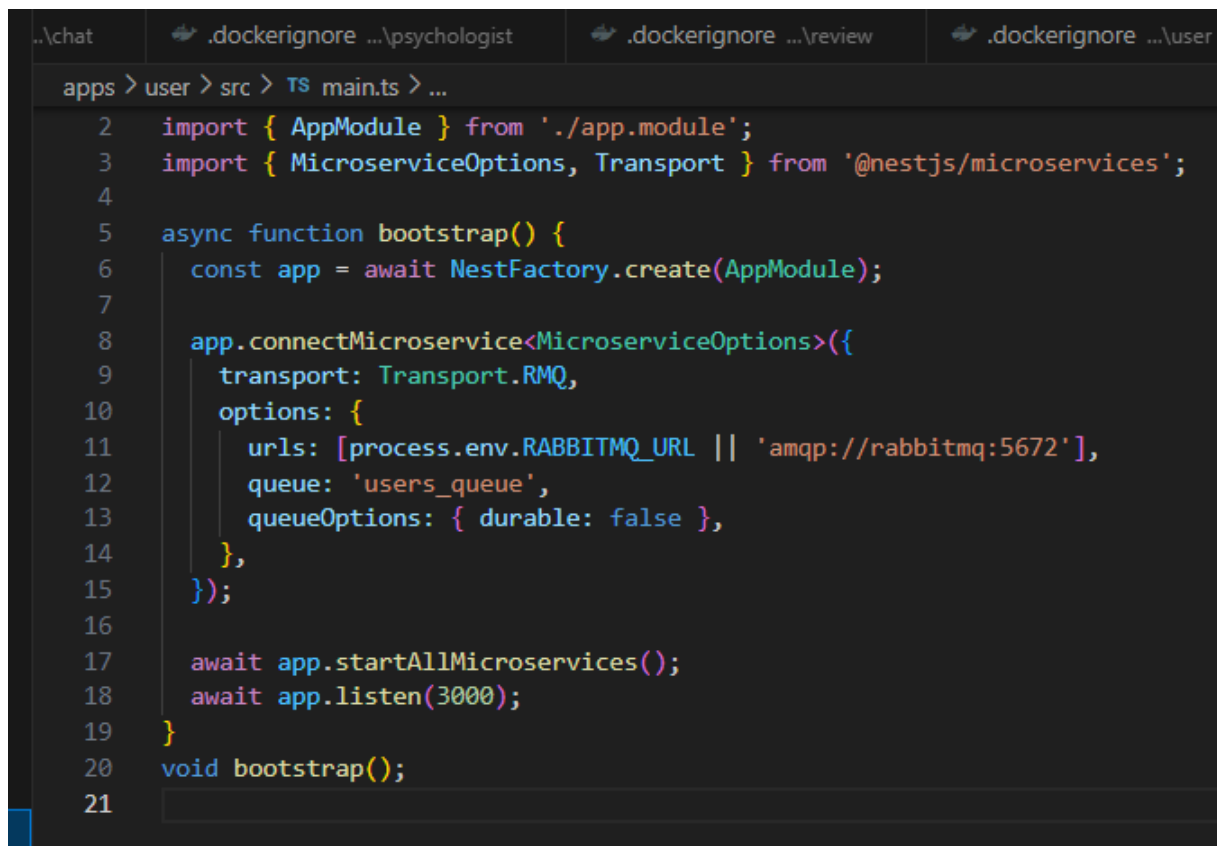
### Пример app.module.ts (user):

```

apps > user > src > TS app.module.ts > ...
1  import { Module } from '@nestjs/common';
2  import { ConfigModule, ConfigService } from '@nestjs/config';
3  import { TypeOrmModule } from '@nestjs/typeorm';
4  import { User } from './user/user.entity';
5  import { UserModule } from './user/user.module';
6  import { AuthModule } from './auth/auth.module';
7
8  @Module({
9    imports: [
10     ConfigModule.forRoot({ isGlobal: true }),
11     TypeOrmModule.forRootAsync({
12       inject: [ConfigService],
13       useFactory: (config: ConfigService) => ({
14         type: 'postgres',
15         host: config.get('DB_HOST'),
16         port: config.get<number>('DB_PORT'),
17         username: config.get('DB_USERNAME'),
18         password: config.get('DB_PASSWORD'),
19         database: config.get('DB_NAME'),
20         entities: [__dirname + '/*.entity{.ts,.js}'],
21         synchronize: true,
22         logging: true,
23       }),
24     ]),
25     TypeOrmModule.forFeature([User]),
26     UserModule,
27     AuthModule,
28   ],
29 })
30 export class AppModule {}
  
```

### 3. Реализация микросервисов и сетевого взаимодействия (RabbitMQ)

#### 3.1. Инициализация микросервиса (пример: user/main.ts):



```
..\chat  .dockerignore ...\psychologist  .dockerignore ...\review  .dockerignore ...\user
apps > 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
```

#### 3.2. Отправка и приём сообщений между сервисами:

**UserService отправляет событие:**

```
await this.clientProxy.emit('user_created', userData);
```

**Другой сервис (например, appointment) принимает:**

```
@MessagePattern('user_created')
```

```
handleUserCreated(data: any) {
```

```
  // обработка нового пользователя
```

```
}
```

#### 4. Dockerfile для каждого сервиса (универсальный пример)

```
apps > user > Dockerfile
1 FROM node:20-alpine
2
3 WORKDIR /app
4
5 COPY package*.json ./
6
7 COPY apps ./apps
8 COPY libs ./libs
9 COPY nest-cli.json ./
10 COPY tsconfig.json ./
11
12 RUN npm install --legacy-peer-deps
13
14 RUN npm run build
15
16 EXPOSE 3000
17
18 CMD ["node", "dist/apps/user/src/main.js"]
19
```

```
services:
  user-db:
    image: postgres:15
    container_name: user-db
    environment:
      POSTGRES_DB: user
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"
    volumes:
      - user_db_data:/var/lib/postgresql/data
    restart: always

  user:
    build:
      context: .
      dockerfile: apps/user/Dockerfile
    env_file:
      - ./apps/user/.env
    depends_on:
      - user-db
      - rabbitmq
    ports:
      - "3001:3000"
    restart: always
    container_name: user

  psychologist-db:
    image: postgres:15
    container_name: psychologist-db
    environment:
      POSTGRES_DB: psychologist
```

```
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
ports:
  - "5433:5432"
volumes:
  - psychologist_db_data:/var/lib/postgresql/data
restart: always

psychologist:
  build:
    context: .
    dockerfile: apps/psychologist/Dockerfile
  env_file:
    - ./apps/psychologist/.env
  depends_on:
    - psychologist-db
    - rabbitmq
  ports:
    - "3002:3000"
  restart: always
  container_name: psychologist

review-db:
  image: postgres:15
  container_name: review-db
  environment:
    POSTGRES_DB: review
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
  ports:
    - "5434:5432"
  volumes:
    - review_db_data:/var/lib/postgresql/data
  restart: always

review:
  build:
    context: .
    dockerfile: apps/review/Dockerfile
  env_file:
    - ./apps/review/.env
  depends_on:
    - review-db
    - rabbitmq
  ports:
    - "3003:3000"
  restart: always
  container_name: review

appointment-db:
  image: postgres:15
  container_name: appointment-db
  environment:
```

```
    POSTGRES_DB: appointment
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
  ports:
    - "5435:5432"
  volumes:
    - appointment_db_data:/var/lib/postgresql/data
  restart: always

appointment:
  build:
    context: .
    dockerfile: apps/appointment/Dockerfile
  env_file:
    - ./apps/appointment/.env
  depends_on:
    - appointment-db
    - rabbitmq
  ports:
    - "3004:3000"
  restart: always
  container_name: appointment

chat-db:
  image: postgres:15
  container_name: chat-db
  environment:
    POSTGRES_DB: chat
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
  ports:
    - "5436:5432"
  volumes:
    - chat_db_data:/var/lib/postgresql/data
  restart: always

chat:
  build:
    context: .
    dockerfile: apps/chat/Dockerfile
  env_file:
    - ./apps/chat/.env
  depends_on:
    - chat-db
    - rabbitmq
  ports:
    - "3005:3000"
  restart: always
  container_name: chat

rabbitmq:
  image: rabbitmq:3-management
  container_name: rabbitmq
```

```
ports:
  - "5672:5672"
  - "15672:15672"
restart: always

volumes:
  user_db_data:
  psychologist_db_data:
  review_db_data:
  appointment_db_data:
  chat_db_data:
```

## Выводы

- Выделены и реализованы самостоятельные микросервисы (user, psychologist, appointment и др).
- Каждый сервис имеет свой Dockerfile для сборки контейнера.
- Использован единый docker-compose.yml для запуска всей системы (базы данных, брокер сообщений и микросервисы).
- Для сетевого взаимодействия между сервисами использован брокер RabbitMQ, реализована отправка и обработка сообщений между сервисами.





```

src > models > TS review.entity.ts > Review > client
1  import {
2      Entity,
3      PrimaryGeneratedColumn,
4      Column,
5      CreateDateColumn,
6      ManyToOne,
7  } from 'typeorm';
8  import { User } from './user.entity';
9  import { Psychologist } from './psychologist.entity';
10
11  @Entity('reviews')
12  export class Review {
13      @PrimaryGeneratedColumn()
14      id: number;
15
16      @ManyToOne(() => User, (user) => user.reviews, { nullable: false })
17      client: User;
18
19      @ManyToOne(() => User)
20      psychologist: Psychologist;
21
22      @Column({ type: 'int' })
23      rating: number;
24
25      @Column({ type: 'text', nullable: true })
26      comment?: string;
27
28      @CreateDateColumn()
29      created_at: Date;
30  }

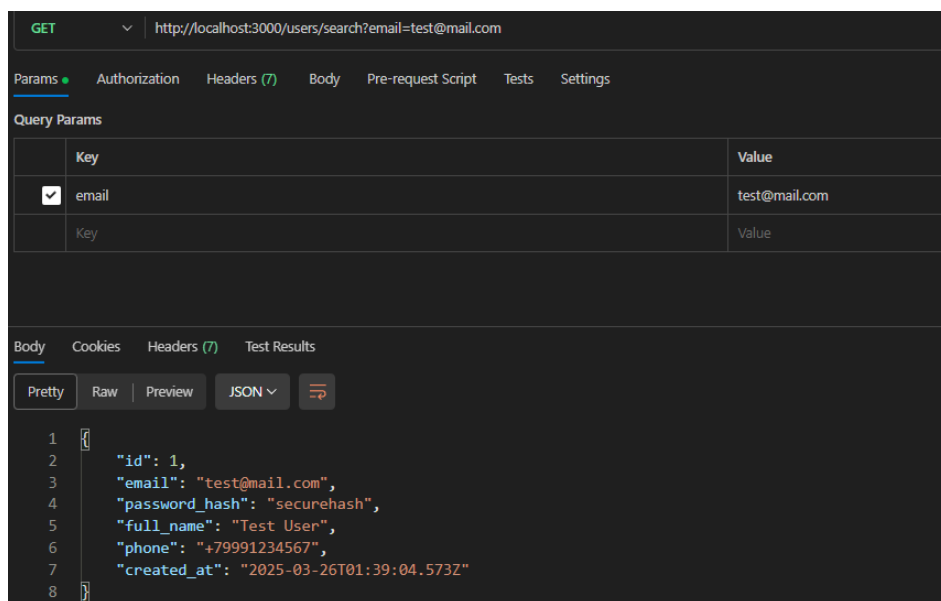
```

## 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:

```
[postgres@localhost ~]$ psql -h localhost -U postgres -d postgres
psql (16.1)
Type "help" for help.

postgres=# \i /home/postgres/.pg_service.conf
postgres=# \c test
test=# \i /home/postgres/.pg_service.conf
test=# SELECT "User"."id" AS "User_id", "User"."email" AS "User_email", "User"."password_hash" AS "User_password_hash", "User"."full_name" AS "User_full_name", "User"."phone" AS "User_phone", "User"."created_at" AS "User_created_at" FROM "users" "User" WHERE (("User"."id" = $1)) LIMIT 1 -- PARAMETERS: [1]
query: SELECT "User"."id" AS "User_id", "User"."email" AS "User_email", "User"."password_hash" AS "User_password_hash", "User"."full_name" AS "User_full_name", "User"."phone" AS "User_phone", "User"."created_at" AS "User_created_at" FROM "users" "User" WHERE (("User"."id" = $1)) LIMIT 1 -- PARAMETERS: [1]
test=# START TRANSACTION
test=# INSERT INTO "psychologists"("experience", "bio", "rating", "price_per_hour", "userId") VALUES ($1, $2, DEFAULT, $3, $4) RETURNING "id", "rating" -
- PARAMETERS: [5,"I'm a DBT certified psychoterapist",5000,1]
test=# COMMIT
```

## 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.