

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнила:
Хисаметдинова Д.Н.

Группа
К3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- ☐ Реализовать все модели данных, спроектированные в рамках ДЗ1
- ☐ Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
- ☐ Реализовать API-эндпоинт для получения пользователя по id/email
- ☐ Необходимо сделать отчёт по шаблону

Ход работы

Проект реализован по принципу MVC (Model–View–Controller) с разделением на директории:

- models: сущности базы данных (TypeORM Entity)
- dto: классы для валидации и сериализации данных
- services: бизнес-логика и работа с БД
- controllers: обработка запросов
- routes: модули NestJS для сборки логики

В соответствии с ER-диаграммой были реализованы следующие модели:

- User
- Appointment
- Psychologist
- Message
- Chat
- Review
- Specialization
- Schedule
- PsychologistSpecialization

На сниппете приведён пример одной из сущностей:

```

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  }

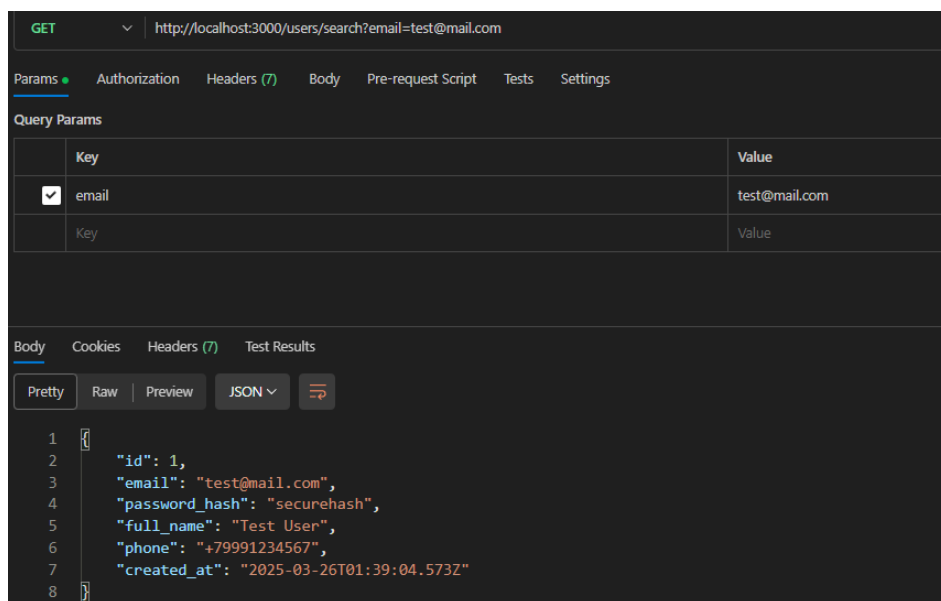
```

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:

```
[Post] 2025-03-26T01:39:10.100 [PostApplication] test application successfully started v0.0.1
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]
query: START TRANSACTION
query: 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]
query: 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.