

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Практическая работа 2

Выполнил:

Пиотуховский Александр

Группа  
К3341

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

## Задача

1. Реализовать все модели данных, спроектированные в рамках ДЗ1
2. Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
3. Реализовать API-эндпоинт для получения пользователя по id/email

## Ход работы

### 1. Реализация моделей данных

Для начала все сущности (таблицы), спроектированные в рамках ДЗ1, были реализованы в виде классов с помощью TypeORM. На рисунке 1 представлена модель записи (поста) в блоге.

```
17 @Entity('posts') Show usages
18 export class Post {
19   @PrimaryGeneratedColumn()
20   id: number;
21
22   @ManyToOne(() => User, user => user.posts, { onDelete: 'CASCADE' })
23   @JoinColumn({ name: 'author_id' })
24   author: User;
25
26   @Column({ type: 'varchar', length: 100 })
27   title: string;
28
29   @Column({ type: 'text' })
30   content: string;
31
32   @CreateDateColumn({ name: 'created_at' })
33   createdAt: Date;
34
35   @UpdateDateColumn({ name: 'updated_at', nullable: true })
36   updatedAt: Date;
37
38   @OneToMany(() => PostImage, image => image.post)
39   images: PostImage[];
40
41   @OneToMany(() => PostFavorite, fav => fav.post)
42   favorites: PostFavorite[];
43
44   @OneToMany(() => PostToRecipe, ptr => ptr.post)
45   postRecipes: PostToRecipe[];
46
47   @OneToMany(() => Comment, comment => comment.post)
48   comments: Comment[];
49 }
```

Рисунок 1 – Модель “Запись” в блоге

С помощью декоратора Entity указывается, что объявленный класс является ORM моделью. В качестве аргумента передаётся имя таблицы в базе данных. Декоратор PrimaryGeneratedColumn указывает, что атрибут является первичным ключом с автоматической генерацией. Декоратор Column выполняет маппинг неключевых атрибутов таблицы с полями класса. В качестве аргумента передаются название атрибута в таблице, его

тип в базе данных, свойство nullable, значение по умолчанию. Для связи моделей друг с другом используются декораторы OneToMany, ManyToMany, ManyToOne.

## 2. Реализация CRUD методов.

Для созданных моделей были написаны CRUD методы с помощью микрофреймворка express. На рисунке 2 изображены контроллеры для модели “Users”.

```
1 import { RequestHandler } from 'express';
2 import * as userService from '../services/userService';
3 import * as postService from '../services/postService';
4 import * as recipeService from '../services/recipeService';
5 import { toUserResponseDTO } from '../utils/user.mapper';
6 import { CreateUserDTO } from '../dtos/user.dto';
7 import { ValidationException } from '../utils/validation';
8 import { updateUser } from '../services/userService';
9 import { toPostResponseDTO } from '../utils/post.mapper';
10 import { toRecipeResponseDTO } from '../utils/recipe.mapper';
11
12 export const listUsersController: RequestHandler = async (req: Request<ParamsDictionary, any, any, Parse...>, res: Response<any, Record<string, any>, number...>, next: NextFunction ): Promise<void> => {
13   try {
14     const users: User[] = await userService.getAllUsers();
15     res.json(users.map(toUserResponseDTO));
16     return;
17   } catch (err) {
18     next(err);
19   }
20 };
21
22 export const getUserController: RequestHandler = async (req: Request<ParamsDictionary, any, any, Parse...>, res: Response<any, Record<string, any>, number...>, next: NextFunction ): Promise<void> => {
23   try {
24     const id: number = Number(req.params.id);
25     const user: User = await userService.findUser({ id });
26     if (!user) {
27       res.status(404).json({ message: 'User not found' });
28       return;
29     }
30     res.json(toUserResponseDTO(user));
31   } catch (err) {
32     next(err);
33   }
34 };
35
36 export const findUserController: RequestHandler = async (req: Request<ParamsDictionary, any, any, Parse...>, res: Response<any, Record<string, any>, number...>, next: NextFunction ): Promise<void> => {
37   try {
38     const { id, email } = req.body as { id?: number; email?: string };
39     if (id == null && !email) {
40       res.status(400).json({ message: 'Either id or email must be provided' });
41       return;
42     }
43   }
44 }
```

Рисунок 2 – CRUD методы модели “Users”

Для каждой модели было написано минимум по 5 контроллеров: получить список моделей, получить модель по Id, создать модель, обновить существующую модель, удалить модель.

## 3. API-эндпоинт для получения пользователя по id/email

В качестве заключительного задания были реализован отдельный API-эндпоинт для модели пользователя. Он позволяет найти пользователя по id или email. На рисунке 3 представлен этот эндпоинт.

```

46 export const findUserController: RequestHandler = async (req: Request<ParamsDictionary, any, any, Parse...>, res: Response<any, Record<string, any>, number...>, next: NextFunction): Promise<void> => {
47   try {
48     const { id, email } = req.body as { id?: number; email?: string };
49     if (id == null && !email) {
50       res.status(400).json({ message: 'Either id or email must be provided' });
51       return;
52     }
53     const user: User = await userService.findUser({ id, email });
54     if (!user) {
55       res.status(404).json({ message: 'User not found' });
56       return;
57     }
58     res.json(toUserResponseDTO(user));
59     return;
60   } catch (err) {
61     next(err);
62   }
63 };

```

Рисунок 3 – контроллер для поиска пользователя по id или email

Для поиска по id его можно взять как из форм-даты запроса, так и передать в пути запроса. Для поиска по email доступен только вариант с форм-датой.

## Вывод

В ходе выполнения работы все таблицы, описанные в задании ДЗ1, были интегрированы в приложение с использованием TypeORM. Для каждой модели были реализованы стандартные CRUD-операции и API-эндпоинты с применением фреймворка express.