САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа

Выполнил:

Пиотуховский Александр

Группа
K3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

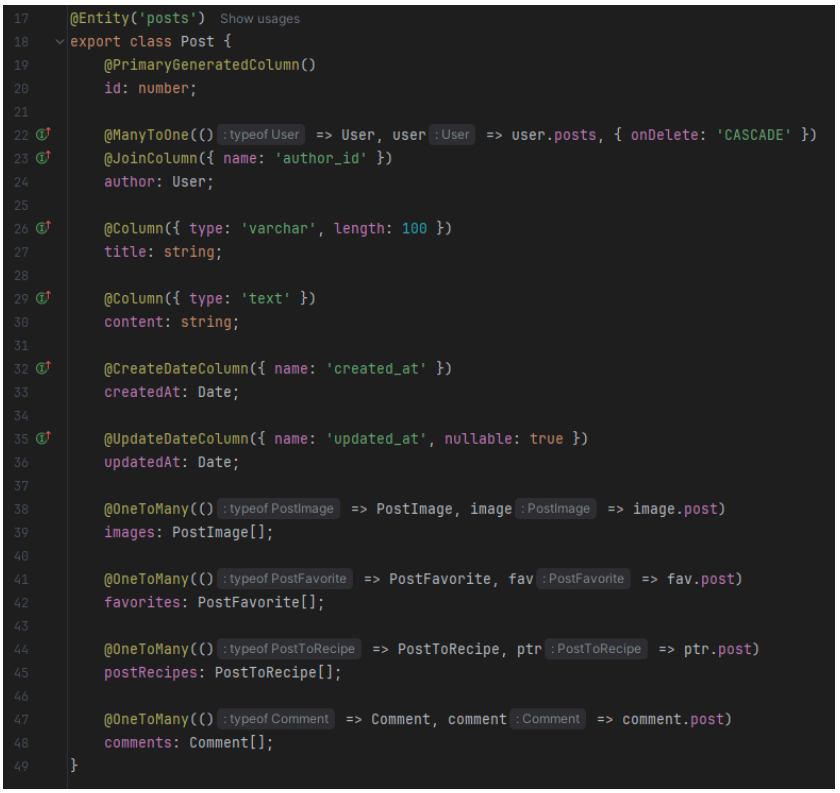
Задача

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.



```
@Entity('posts') Show usages
export class Post {
    @PrimaryGeneratedColumn()
    id: number;

    @ManyToOne(() : typeof User => User, user : User => user.posts, { onDelete: 'CASCADE' })
    @JoinColumn({ name: 'author_id' })
    author: User;

    @Column({ type: 'varchar', length: 100 })
    title: string;

    @Column({ type: 'text' })
    content: string;

    @CreateDateColumn({ name: 'created_at' })
    createdAt: Date;

    @UpdateDateColumn({ name: 'updated_at', nullable: true })
    updatedAt: Date;

    @OneToMany(() : typeof PostImage => PostImage, image : PostImage => image.post)
    images: PostImage[];

    @OneToMany(() : typeof PostFavorite => PostFavorite, fav : PostFavorite => fav.post)
    favorites: PostFavorite[];

    @OneToMany(() : typeof PostToRecipe => PostToRecipe, ptr : PostToRecipe => ptr.post)
    postRecipes: PostToRecipe[];

    @OneToMany(() : typeof Comment => Comment, comment : Comment => comment.post)
    comments: Comment[];
}
```

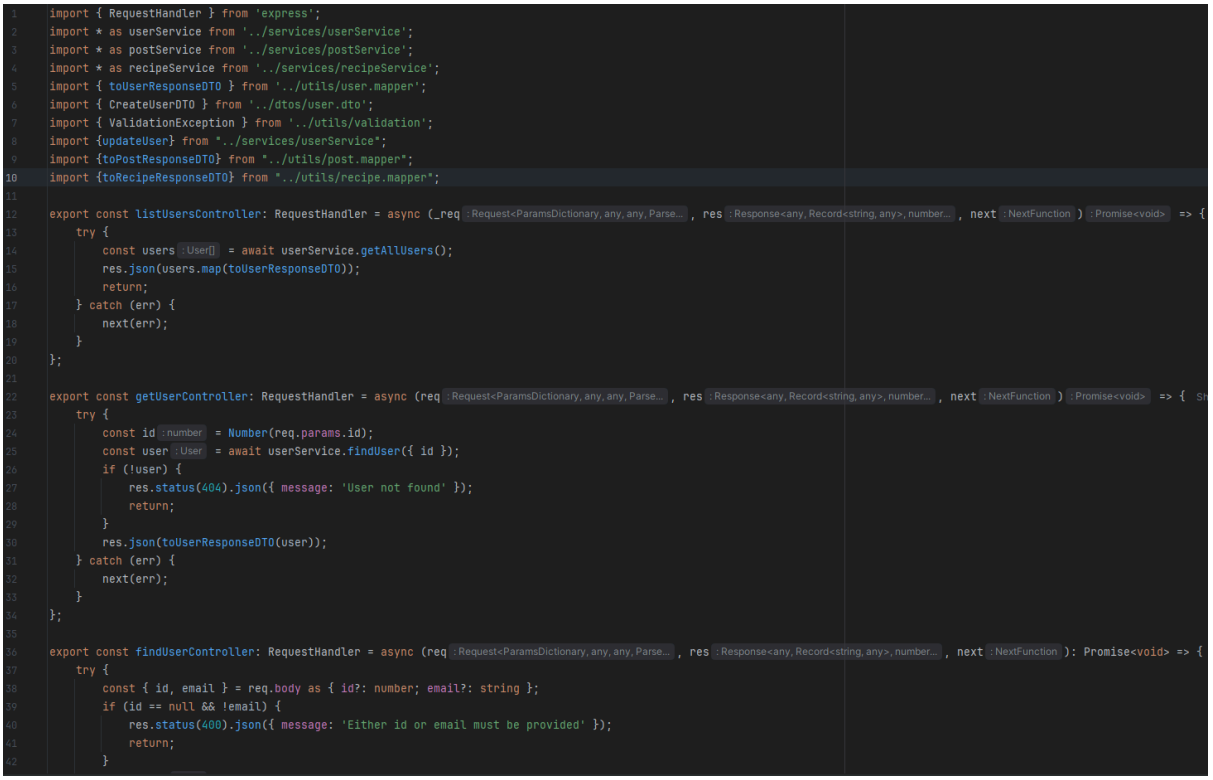
Рисунок 1 – Модель “Запись” в блоге

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”.



```
import { RequestHandler } from 'express';
import * as userService from '../services/userService';
import * as postService from '../services/postService';
import * as recipeService from '../services/recipeService';
import { toUserResponseDTO } from '../utils/user.mapper';
import { CreateUserDTO } from '../dtos/user.dto';
import { ValidationException } from '../utils/validation';
import {updateUser} from "../services/userService";
import {toPostResponseDTO} from "../utils/post.mapper";
import {toRecipeResponseDTO} from "../utils/recipe.mapper";

export const listUsersController: RequestHandler = async (_req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction ) : Promise<void> => {
    try {
        const users : User[] = await userService.getAllUsers();
        res.json(users.map(toUserResponseDTO));
        return;
    } catch (err) {
        next(err);
    }
};

export const getUserController: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction ) : Promise<void> => {
    try {
        const id : number = Number(req.params.id);
        const user : User = await userService.findUser({ id });
        if (!user) {
            res.status(404).json({ message: 'User not found' });
            return;
        }
        res.json(toUserResponseDTO(user));
    } catch (err) {
        next(err);
    }
};

export const findUserController: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction ): Promise<void> => {
    try {
        const { id, email } = req.body as { id?: number; email?: string };
        if (id == null && !email) {
            res.status(400).json({ message: 'Either id or email must be provided' });
            return;
        }
```

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.

```
export const findUserController: RequestHandler = async (req :Request<ParamsDictionary, any, any, Parse... , res :Response<any, Record<string, any>, number... , next :NextFunction ): Promise<void> => {
    try {
        const { id, email } = req.body as { id?: number; email?: string };
        if (id == null && !email) {
            res.status(400).json({ message: 'Either id or email must be provided' });
            return;
        }
        const user :User  = await userService.findUser({ id, email });
        if (!user) {
            res.status(404).json({ message: 'User not found' });
            return;
        }
        res.json(toUserResponseDTO(user));
        return;
    } catch (err) {
        next(err);
    }
};
```

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 использование TypeORM. Для каждой модели были реализованы стандартные CRUD-операции и API-эндпоинты с применением фреймворка express.