

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

ДЗ 1 - Проектирование базы данных

Выполнил:

Сергеев Виктор

К3341

Проверил:

Добряков Д. И.

Санкт-Петербург

2022 г.

## Задача

Требуется реализовать модели спроектированные в ДЗ1 с помощью TypeORM и подготовить проект к дальнейшей работы с Express.

Конкретные требования:

- Реализовать все модели данных
- Реализовать набор из CRUD-методов для дальнейшей работы Express
- Реализовать API-эндпоинт для получения пользователя по id

## Ход работы

Данный проект будет реализован по архитектуре MVCS. Текущая файловая структура проекта изображена на рисунке 1

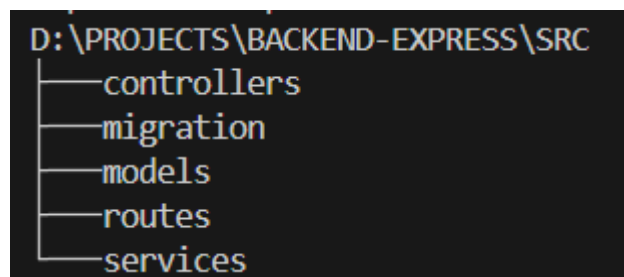


Рисунок 1 - файловая структура проекта

Помимо этого, в директории src находятся файлы index.ts - содержит инициализацию приложения - и data-source.ts - содержит инициализацию настроек базы данных. Назначение каждой директории следующее:

- controllers - содержит контроллеры моделей
- migration - содержит файлы миграции базы данных
- models - содержит описания моделей
- routes - содержит роутеры для каждой модели
- services - содержит сервисы моделей

В этой структуре сервис отвечает за обращения к бд, контроллер для принятие данных, их валидации и взаимодействие с сервисом.

Модели были описаны с помощью TypeORM. На рисунке 2 изображен пример описания модели User - пользователя. Все остальные модели были спроектированы аналогичным образом.

```

src > models > TS Users.ts > ...
1  import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from "typeorm"
2  import { Recipe } from "../Recipe"
3  import { Comment } from "../Comment"
4
5  @Entity()
6  export class User {
7
8      @PrimaryGeneratedColumn()
9      id: number
10
11      @Column({type: "varchar", length: 50, unique: true})
12      username: string
13
14      @Column({type: "varchar", length: 100})
15      password: string
16
17      @Column({type: "varchar", length: 200, nullable: true})
18      avatar_url: string | null
19
20      @Column({type: "varchar", length: 800, nullable: true})
21      bio: string | null
22
23      @Column({type: "timestamp", update: false, default: () => "CURRENT_TIMESTAMP"})
24      created_at: Date
25
26      @OneToMany(() => Recipe, (recipe) => recipe.author)
27      recipes: Array<Recipe>
28
29      @OneToMany(() => Comment, (comment) => comment.user)
30      comments: Array<Comment>
31
32      @OneToMany(() => Recipe, (recipe) => recipe.users_liked)
33      recipes_liked: Array<Recipe>
34
35      @OneToMany(() => Recipe, (recipe) => recipe.users_saved)
36      recipes_saved: Array<Recipe>
37  }
38

```

Рисунок 2 - файл User.ts

Далее следовало реализовать CRUD-методы - способ взаимодействовать с базой данных и выполнять основные операции над моделями. Для этого надо было реализовать написать UserService и UserController. На рисунках 3 и 4 представлены примеры UserService и UserController соответственно. Написания сервисов и контроллеров для остальных моделей было выполнено аналогично.

```

src > services > TS UserService.ts > UserService > deleteUser
1  import { Repository } from "typeorm";
2  import { AppDataSource } from "../data-source";
3  import { User } from "../models/User";
4
5  export class UserService {
6      private repository: Repository<User>
7
8      constructor() {
9          this.repository = AppDataSource.getRepository(User);
10     }
11
12     getAllUsers = async(): Promise<Array<User>> => {
13         return this.repository.find();
14     }
15
16     getUserById = async (id: number): Promise<User | null> => {
17         return this.repository.findOneBy({id: id});
18     }
19
20     createUser = async (data: Partial<User>): Promise<User> => {
21         const entity = this.repository.create(data);
22         return this.repository.save(entity);
23     }
24
25     updateUser = async (id: number, data: Partial<User>): Promise<User | null> => {
26         const entity = await this.getUserById(id);
27         if (!entity) {
28             return null;
29         }
30         this.repository.merge(entity, data);
31         return this.repository.save(entity);
32     }
33
34     deleteUser = async (id: number): Promise<boolean> => {
35         const result = await this.repository.delete(id);
36         return result.affected > 0;
37     }
38 }

```

Рисунок 3 - файл UserService.ts

```

src > controllers > TS UserController.ts > UserController > updateUser
3
4 export class UserController {
5     private service: UserService;
6
7     constructor () {
8         this.service = new UserService();
9     }
10
11     getUsers = async (request: Request, response: Response) => {
12         const entities = await this.service.getAllUsers();
13         response.json(entities);
14     }
15
16     getUserById = async (request: Request, response: Response) => {
17         const entity = await this.service.getUserById(Number(request.params.id));
18         if (!entity) {
19             response.status(404).json({message: "User not found"});
20         }
21         response.json(entity);
22     }
23
24     createUser = async (request: Request, response: Response) => {
25         try {
26             const entity = await this.service.createUser(request.body);
27             response.status(201).json(entity);
28         } catch(error) {
29             response.status(400).json({error: error.message});
30         }
31     }
32
33     updateUser = async (request: Request, response: Response) => {
34         try {
35             const updated = await this.service.updateUser(Number(request.params.id), request.body);
36             if (!updated) {
37                 response.status(404).json({message: "User not found"});
38             } else {
39                 response.json(updated);
40             }
41         } catch (error) {
42             response.status(400).json({error: error.message});
43         }
44     }
45
46     deleteUser = async (request: Request, response: Response) => {
47         try {
48             const deleted = await this.service.deleteUser(Number(request.params.id));
49             if (!deleted) {
50                 response.status(404).json({message: "User not found"});
51             } else {
52                 response.json({message: "User deleted"});
53             }
54         } catch (error) {
55             response.status(400).json({error: error.message});
56         }
57     }
58 }

```

Рисунок 4 - файл UserController.ts

Наконец, был реализован роутер для модели User - UserRouter.ts. Он отвечает за сопоставление методов контроллера URL адресам, по которым идут обращения к серверу. Его код представлен на рисунке 5.

```

src > routes > TS UserRouter.ts > ...
1  import { Router } from "express";
2  import { UserController } from "../controllers/UserController";
3
4  const router = Router();
5  const controller = new UserController();
6
7  router.get("/", controller.getUsers);
8  router.get("/:id", controller.getUserById);
9  router.post("/", controller.createUser);
10 router.put("/:id", controller.updateUser);
11 router.delete("/:id", controller.deleteUser);
12
13 export default router;
14

```

Рисунок 5 - файл UserRouter.ts

Наконец, был инициализирован весь проект в файле index.ts. Код представлен на рисунке 6 и на рисунках 7-9 представлена работа api для модели пользователя.

```

src > TS index.ts > ...
1  import "reflect-metadata"
2  import { AppDataSource } from "../data-source"
3  import express = require("express");
4  import bodyParser = require("body-parser");
5  import userRouter from "../routes/UserRouter"
6
7  AppDataSource.initialize()
8  .then(async () => {
9    console.log("db initiated")
10  }).catch(error => console.log(error))
11
12 const app = express();
13 app.use(bodyParser.json())
14 app.use("/api/user", userRouter)
15
16 app.listen(3000);

```

Рисунок 6 - файл index.ts

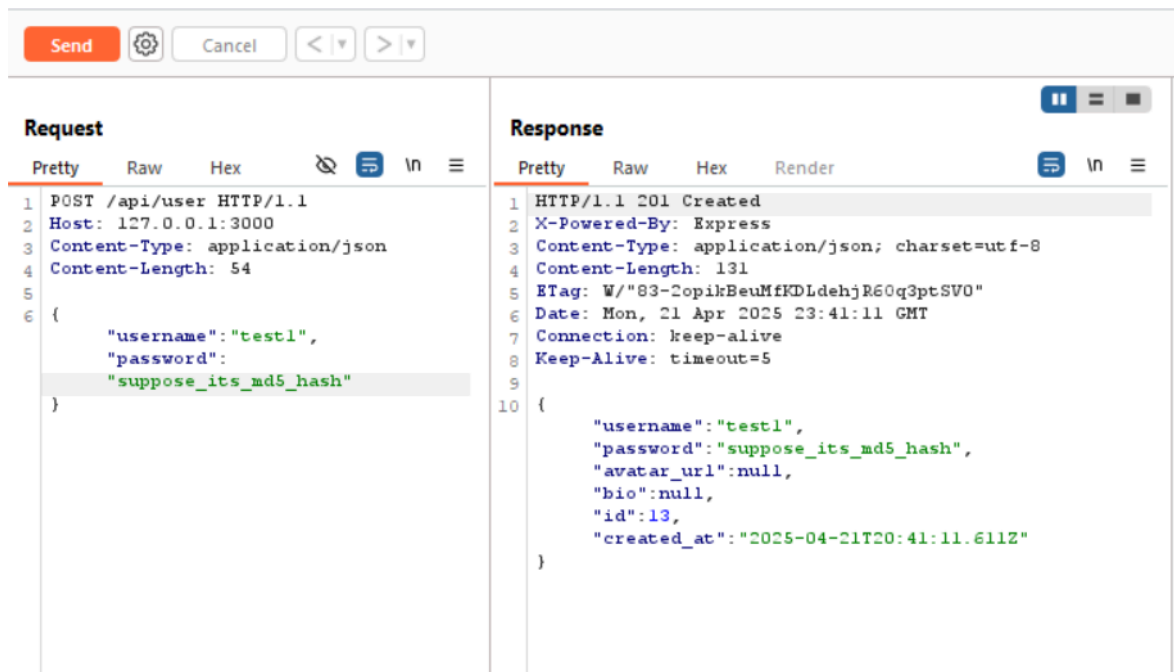


Рисунок 7 - создание пользователя

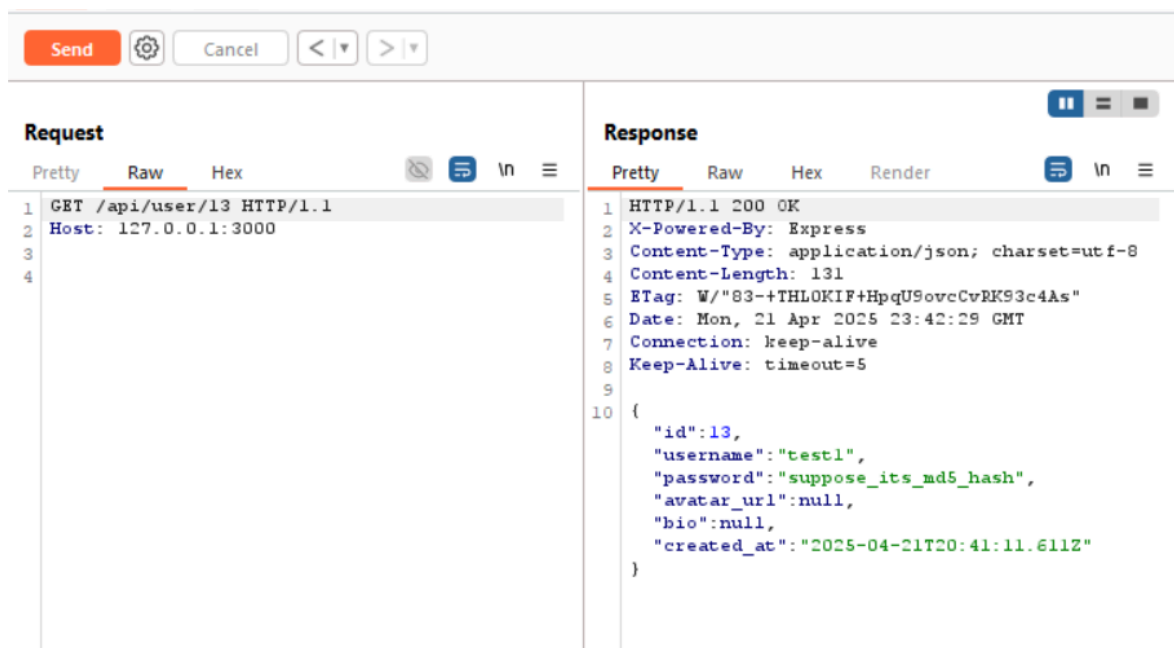


Рисунок 8 - получение пользователя по id



Рисунок 9 - удаление пользователя

## Вывод

В процессе работы были реализованы модели базы данных по архитектуре MVCS реализованы сервисы и контроллеры, которые отвечают за взаимодействие с базой данных. Был реализован API для управление моделью User.