САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа

Выполнил:

Фирсов Илья

K3341

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

Ход работы

В качестве основы и вдохновения для файловой структуры взял шаблон (https://github.com/kantegory/express-typeorm-boilerplate). Так как тема работы — «Работа с TypeORM», начнем с data source.

```
import 'reflect-metadata';
import { DataSource } from 'typeorm';
import SETTINGS from './settings';
const dataSource = new DataSource({
    type: 'postgres',
    host: SETTINGS.DB_HOST,
    port: SETTINGS.DB_PORT,
    username: SETTINGS.DB_USER,
    password: SETTINGS.DB_PASSWORD,
    database: SETTINGS.DB_NAME,
    synchronize: false,
    logging: true,
    entities: [SETTINGS.DB_ENTITIES],
    subscribers: [SETTINGS.DB_SUBSCRIBERS],
    migrations: [SETTINGS.DB_MIGRATIONS],
});
export default dataSource;
```

Чтобы поменьше бойлерплейтить написал дженерики: контроллер и сервис.

```
import { Repository, ObjectLiteral, DeepPartial } from 'typeorm';
export abstract class BaseService<T extends ObjectLiteral> {
    protected constructor(protected readonly repo: Repository<T>) {}
    public async list(): Promise<T[]> {
        return this.repo.find();
    }
    public async getById(id: string): Promise<T> {
        const e = await this.repo.findOneBy({ id } as any);
        if (!e) throw new Error(`${this.constructor.name}(${id}) not found`);
        return e;
    }
    public async create(data: DeepPartial<T>): Promise<T> {
        const ent = this.repo.create(data);
        return this.repo.save(ent);
    }
    public async update(id: string, data: Partial<T>): Promise<T> {
        await this.repo.update(id, data);
        return this.getById(id);
    }
    public async delete(id: string): Promise<void> {
        const result = await this.repo.delete(id);
        if (result.affected === 0) throw new Error(`${this.constructor.name}(${id}) not
found`);
    }
}
```

```
import {
    Get,
    Post,
    Put,
```

```
    Delete,
    Path,
    Body,
    SuccessResponse, Controller,
} from 'tsoa';
import { BaseService } from '../services/base';
import {DeepPartial, ObjectLiteral} from "typeorm";
export abstract class BaseCrudController<T extends ObjectLiteral> extends Controller {
    protected abstract service: BaseService<T>;
    @Get()
    public async list(): Promise<T[]> {
        return this.service.list();
    }
    @Get('{id}')
    public async detail(@Path() id: string): Promise<T> {
        return this.service.getById(id);
    }
    @Post()
    @SuccessResponse('201')
    public async create(@Body() dto: DeepPartial<T>): Promise<T> {
        return this.service.create(dto);
    }
    @Put('{id}')
    public async update(
        @Path() id: string,
        @Body() dto: Partial<T>,
    ): Promise<T> {
        return this.service.update(id, dto);
    }
    @Delete('{id}')
    @SuccessResponse('204')
    public async delete(@Path() id: string): Promise<void> {
        await this.service.delete(id);
    }
}
```

Было неожиданно много проблем с настройкой Swagger UI — tsoa, почему-то, игнорирует securitySchemas из tsoa.json, так что пришлось расширить хелпер

```
import { Application } from 'express';
import swaggerUi from 'swagger-ui-express';
import rawDocument from '../../swagger.json';
import SETTINGS from '../config/settings';
export function useSwagger(app: Application): void {
    const spec = JSON.parse(JSON.stringify(rawDocument));
    spec.info = {
        title:       SETTINGS.APP_NAME,
        version:     SETTINGS.APP_VERSION,
        description: SETTINGS.APP_DESCRIPTION,
    };
    spec.servers = [
        {
            url: `${SETTINGS.APP_PROTOCOL}://${SETTINGS.APP_HOST}:${SETTINGS.APP_PORT}${SETTINGS.APP_API_PREFIX}`,
            description: 'API server',
        },
    ];
    spec.components = spec.components || {};
    spec.components.securitySchemes = {
        bearerAuth: {
            type:         'http',
            scheme:       'bearer',
            bearerFormat: 'JWT',
            description:  'Enter: `<token>`',
        },
        ...spec.components.securitySchemes,
    };
    const uiOptions = {
        explorer: true,
```

```
        swaggerOptions: {
            persistAuthorization: true,
        },
    };
    app.use('/docs', swaggerUi.serve, swaggerUi.setup(spec, uiOptions));
}
```

Для сущностей тоже использовал абстрактный класс, тип pk — uuid, + created_at/updated_at с помощью встроенные в TypeORM декораторов

```
import {
    PrimaryGeneratedColumn,
    CreateDateColumn,
    UpdateDateColumn,
    BaseEntity as TypeORMBase,
} from 'typeorm';
export abstract class BaseEntity extends TypeORMBase {
    @PrimaryGeneratedColumn('uuid')
    id!: string;
    @CreateDateColumn({ name: 'created_at' })
    createdAt!: Date;
    @UpdateDateColumn({ name: 'updated_at' })
    updatedAt!: Date;
}
```

В целом, больше ничего интересного тут нет, ниже представлены сущность, сервис и контроллер для одной сущности (User)

```
import {
  Entity,
  Column,
  ManyToOne,
  OneToMany,
  JoinColumn,
} from 'typeorm';
import { BaseEntity } from './BaseEntity';
import { Company } from './Company';
import { Resume } from './Resume';
import { Application } from './Application';
import { UserRole, UserActivity } from '../common/enums';
import {Vacancy} from "./Vacancy";
@Entity({ name: 'users' })
export class User extends BaseEntity {
  @Column({ unique: true })
  email!: string;
  @Column({ unique: true })
  username!: string;
  @Column()
  passwordHash!: string;
  @Column()
  name!: string;
  @Column({
    type: 'enum',
    enum: UserRole,
    enumName: 'user_role',
    default: UserRole.JobSeeker,
  })
  role!: UserRole;
  @Column({
```

```
    type: 'enum',
    enum: UserActivity,
    enumName: 'user_activity',
    default: UserActivity.LookingForJob,
  })
  activity!: UserActivity;
  @OneToMany(() => Resume, (r) => r.user)
  resumes!: Resume[];
  @ManyToOne(() => Resume, { nullable: true })
  @JoinColumn({ name: 'current_resume_id' })
  currentResume?: Resume;
  @OneToMany(() => Application, (app) => app.user)
  applications!: Application[];
  @ManyToOne(() => Company, (c) => c.users, { nullable: true })
  company?: Company;
  @OneToMany(() => Vacancy, (vacancy) => vacancy.employer)
  vacancies!: Vacancy[];
}
```

```
import { BaseService } from './base';
import dataSource from '../config/data-source';
import { User } from '../models/User';
export class UserService extends BaseService<User> {
    constructor() {
        super(dataSource.getRepository(User));
    }
    public async getByEmail(email: string): Promise<User | null> {
        return this.repo.findOneBy({ email });
    }
}
```

```
import {
    Route,
    Get,
```

```
    Post,
    Put,
    Delete,
    Body,
    Path,
    Query,
    SuccessResponse,
    Tags,
    Response,
    Security
} from 'tsoa';
import { UserService } from '../services/user';
import { User } from '../models/User';
import {BaseCrudController} from "./base-controller";
@Security('bearerAuth')
@Route('users')
@Tags('User')
export class UserController extends BaseCrudController<User> {
    protected service = new UserService();
    @Get('search')
    @Response<Error>(400, 'Bad Request')
    public async getOneByEmail(@Query() email?: string): Promise<User> {
        if (!email) {
            this.setStatus(400);
            throw new Error('Email query parameter is required');
        }
        const user = await this.service.getByEmail(email);
        if (!user) {
            this.setStatus(404);
            throw new Error('User not found');
        }
        return user;
    }
    @Get()
    public async list(): Promise<User[]> {
```

```
        return super.list();
    }
    @Get('{id}')
    public async detail(@Path() id: string): Promise<User> {
        return super.detail(id);
    }
    @Post()
    @SuccessResponse('201')
    public async create(@Body() dto: Partial<User>): Promise<User> {
        return super.create(dto);
    }
    @Put('{id}')
    public async update(
        @Path() id: string,
        @Body() dto: Partial<User>
    ): Promise<User> {
        return super.update(id, dto);
    }
    @Delete('{id}')
    @SuccessResponse('204')
    public async delete(@Path() id: string): Promise<void> {
        return super.delete(id);
    }
}
```

Вывод

Научился работать с TypeORM + tsoa + express