САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа

Выполнила:

Едигарева Дарья

Группа К3339

Проверил:
Добряков Д. И.

Санкт-Петербург
2025 г.

Задача:
Реализовать серверную часть сайта для поиска работы с использованием Express и TypeScript, а именно:

1. Описать все модели данных, спроектированные в рамках ДЗ1, с помощью TypeORM.

2. Реализовать для этих моделей набор CRUD-методов.

3. Сделать отдельный API-эндпоинт для поиска пользователя по id и по email.

Вариант: Сайт для поиска работы

регистрация, личный кабинет со своим резюме, поиск вакансий с фильтрацией (отрасль, зарплата, опыт), страница деталей вакансии, личный кабинет работодателя (управление вакансиями).

Ход работы

1. **Описание моделей**
   Все сущности спроектированы как классы-модели TypeORM, наследующие общий BaseEntity с полями id, createdAt, updatedAt.

   Листинг 1. Пример модели User

```
import {
  Entity,
  Column,
  OneToOne,
} from 'typeorm';
import { BaseEntity } from './BaseEntity';
import { UserRole } from '../common/enums';
import { EmployerProfile } from './EmployerProfile';
import { JobSeekerProfile } from './JobSeekerProfile';

@Entity({ name: 'users' })
export class User extends BaseEntity {
```

```
    @Column({ unique: true })
    email!: string;

    @Column()
    passwordHash!: string;

    @Column({
        type: 'enum',
        enum: UserRole,
        enumName: 'user_role',
        default: UserRole.JobSeeker,
    })
    role!: UserRole;

      @OneToOne(() => EmployerProfile, (ep) => ep.user,
{ cascade: true })
    employerProfile?: EmployerProfile;

      @OneToOne(() => JobSeekerProfile, (jp) => jp.user,
{ cascade: true })
    jobSeekerProfile?: JobSeekerProfile;
}
```

По аналогии определены модели Company, Vacancy, EmployerProfile, JobSeekerProfile, Resume, WorkExperience, Education, Skill, ResumeSkill, VacancySkill, Application.

2. **Миграции БД**
   На основе моделей сгенерированы миграции (TypeORM CLI), которые создают соответствующие таблицы и enum-типы в PostgreSQL.

3. **Настройка Express + TSOA**

Создан App в src/app.ts, подключены cors, JSON-парсер.

Настроен Swagger UI через tsoa и файл swagger.json (в src/common/swagger.ts).
Маршрутизация генерируется автоматически в src/routes.ts на основе декораторов в контроллерах .

4. **Сервисный слой (CRUD-методы)**
   Реализован UserService с методами list(), getById(), create(), update(), delete(), getByEmail().

Листинг 2. Пример CRUD-методов в UserService

```
import { Repository } from 'typeorm';

import dataSource from '../config/data-source';
import { User } from '../models/User';

export class UserService {
   private readonly repo: Repository<User>;

   constructor() {
       this.repo = dataSource.getRepository(User);
   }

   public async list(): Promise<User[]> {
       return this.repo.find();
   }

   public async getById(id: string): Promise<User> {
           const u = await this.repo.findOne({ where:
{ id } });
       if (!u) throw new Error(`User(${id}) not found`);
       return u;
   }

       public async create(data: Partial<User>):
Promise<User> {
       const ent = this.repo.create(data);
       return this.repo.save(ent);
   }

    public async update(id: string, data: Partial<User>):
Promise<User> {
       const result = await this.repo.update(id, data);
       if (result.affected === 0) throw new Error(`User($
{id}) not found`);
       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(`User($
{id}) not found`);
   }
```

```
    public async getByEmail(email: string): Promise<User |
null> {
        return this.repo.findOne({ where: { email } });
    }
}
```

5. **Контроллеры и эндпоинты**
   На основе TSOA созданы контроллеры в src/controllers.

UserController с маршрутами:

GET /api/users — список пользователей;
GET /api/users/{id} — детальная информация;
POST /api/users — создание;
PUT /api/users/{id} — обновление;
DELETE /api/users/{id} — удаление;
GET /api/users/byEmail?email=… — поиск по email.

Листинг 3. Эндпоинт поиска пользователя по email

```
@Get('byEmail')
@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;
}
```

Вывод

В ходе работы:

Спроектированы и реализованы все необходимые модели данных для сервиса для поиска работы.
Настроен процесс миграций БД и внесены соответствующие таблицы и перечисления.
Реализован сервисный слой с полным набором CRUD-методов.
С помощью TSOA и Express+TypeScript настроены контроллеры и маршруты.