САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1
Реализация boilerplate

Выполнила:

Едигарева Дарья

Группа
К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

Реализовать boilerplate-проект на основе Express, TypeScript и TypeORM, обеспечив явное разделение на: модели (TypeORM entities), контроллеры (TSOA + Express), роуты (генерируемые автоматически).

Ход работы

1. Модели данных (TypeORM Entities)

В каталоге src/models/ реализованы все основные сущности:
User.ts, Company.ts, Vacancy.ts, EmployerProfile.ts, JobSeekerProfile.ts, Resume.ts, WorkExperience.ts, Education.ts, Skill.ts, VacancySkill.ts, ResumeSkill.ts, Application.ts и общий BaseEntity.ts.

Пример — модель User:

```
@Entity({ name: 'vacancies' })

export class Vacancy extends BaseEntity {

   @ManyToOne(() => Company, (c) => c.vacancies, {

       nullable: false,

       onDelete: 'CASCADE',

   })

   @JoinColumn({ name: 'company_id' })

   company!: Company;



   @ManyToOne(() => EmployerProfile, (ep) => ep.id, {

       nullable: false,

       onDelete: 'CASCADE',

   })

   @JoinColumn({ name: 'employer_profile_id' })

   employerProfile!: EmployerProfile;
```

```
@Column()
title!: string;

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

@Column({ type: 'text', nullable: true })
requirements?: string;

@Column({ name: 'salary_min', type: 'numeric', nullable: true })
salaryMin?: number;

@Column({ name: 'salary_max', type: 'numeric', nullable: true })
salaryMax?: number;

@Column({
    type: 'enum',
    enum: Industry,
    enumName: 'industry',
    default: Industry.Other,
})
industry!: Industry;

@Column({ name: 'experience_required', type: 'integer', nullable: true })
experienceRequired?: number;
```

```
  @Column({ name: 'posted_date', type: 'timestamptz' })
  postedDate!: Date;

  @Column({ name: 'expire_date', type: 'timestamptz', nullable: true })
  expireDate?: Date;

  @OneToMany(() => VacancySkill, (vs) => vs.vacancy, { cascade: true })
  vacancySkills!: VacancySkill[];

  @OneToMany(() => Application, (app) => app.vacancy, { cascade: true })
  applications!: Application[];
}
```

Для инициализации схемы создана миграция:
src/migrations/1745436695167-Init.ts.

2. Контроллеры (TSOA + Express)

В src/controllers/ находятся контроллеры:
auth-controller.ts, user-controller.ts, vacancy-controller.ts, resume-controller.ts, company-controller.ts, ...

Пример — контроллер вакансий (VacancyController):

```
import { Controller, Get, Post, Put, Delete, Route, Tags, Query, Path, Body, Security, SuccessResponse, Response } from 'tsoa';

import { VacancyService, VacancySearchParams, VacancyCreateDto, VacancyUpdateDto } from '../services/vacancy';

import { Vacancy } from '../models/Vacancy';

import { Industry } from '../common/enums';
```

```
interface VacancySearchQuery {
   industry?: Industry;
   salaryMin?: number;
   salaryMax?: number;
   experienceMin?: number;
   experienceMax?: number;
}

@Route('vacancies')
@Tags('Vacancy')
export class VacancyController extends Controller {
   private service = new VacancyService();

   @Get()
   public async search(
       @Query() industry?: Industry,
       @Query() salaryMin?: number,
       @Query() salaryMax?: number,
       @Query() experienceMin?: number,
       @Query() experienceMax?: number,
   ): Promise<Vacancy[]> {
       const params: VacancySearchParams = {
           industry,
           salaryMin: typeof salaryMin === 'number' ? salaryMin : undefined,
           salaryMax: typeof salaryMax === 'number' ? salaryMax : undefined,
```

```
            experienceMin: typeof experienceMin === 'number' ? experienceMin :
undefined,
            experienceMax: typeof experienceMax === 'number' ? experienceMax :
undefined,
        };
        return this.service.search(params);
    }

    @Get('{id}')
    public async detail(@Path() id: string): Promise<Vacancy> {
        return this.service.getById(id);
    }

    @Security('bearerAuth')
    @Post()
    @SuccessResponse('201')
    @Response<Error>(403, 'Forbidden')
    public async create(@Body() dto: VacancyCreateDto): Promise<Vacancy> {
        const user = (this as any).request?.user as { id: string } | undefined;
        if (!user?.id) {
            this.setStatus(401);
            throw new Error('Unauthorized');
        }
        const v = await this.service.createForEmployer(user.id, dto);
        this.setStatus(201);
        return v;
    }
```

```
  @Security('bearerAuth')
  @Put('{id}')
  @Response<Error>(403, 'Forbidden')
     public async update(@Path() id: string, @Body() dto: VacancyUpdateDto): Promise<Vacancy> {
      const user = (this as any).request?.user as { id: string } | undefined;
      if (!user?.id) {
          this.setStatus(401);
          throw new Error('Unauthorized');
      }
      return this.service.updateForEmployer(user.id, id, dto);
  }

  @Security('bearerAuth')
  @Delete('{id}')
  @SuccessResponse('204')
  @Response<Error>(403, 'Forbidden')
  public async remove(@Path() id: string): Promise<void> {
      const user = (this as any).request?.user as { id: string } | undefined;
      if (!user?.id) {
          this.setStatus(401);
          throw new Error('Unauthorized');
      }
      await this.service.deleteForEmployer(user.id, id);
      this.setStatus(204);
```

```
    }

}
```

Контроллеры принимают параметры и делегируют в сервисы.

3. Роуты

Роуты генерируются TSOA в src/routes.ts и подключаются в src/app.ts через:

RegisterRoutes(this.app);

Базовый путь (/api) и схема авторизации (bearerAuth) задаются в tsoa.json.

4. Точка входа (App)

Файл src/app.ts отвечает за настройку Express:

```
export class App {

   public readonly app: Application;

   constructor() {

       this.app = express();



       // middlewares

       this.app.use(cors());

       this.app.use(express.json());



       // swagger

       useSwagger(this.app);



       // routes

       RegisterRoutes(this.app);

       this.configureErrorHandling();
```

```
    }


    private configureErrorHandling(): void {
        const errorHandler: ErrorRequestHandler = (err, req, res, next) => {
            if (err && (err as any).status && (err as any).fields) {
                const e = err as any;
                   res.status(e.status).json({ message: e.message, details: e.fields
});
            }
            else if (err instanceof Error) {
                console.error(err);
                res.status(500).json({ message: err.message });
            }
            else {
                next();
            }
        };
        this.app.use(errorHandler);
    }
    public async start(): Promise<void> {
        await dataSource.initialize();
        this.app.listen(SETTINGS.APP_PORT, SETTINGS.APP_HOST, () => {
            console.log(
                                                        `Server    running    at
${SETTINGS.APP_PROTOCOL}://${SETTINGS.APP_HOST}:${SETTINGS.APP_PORT}`
            );
        });
```

```
    }
}


new App()
    .start()
    .catch(err => {
        console.error('Failed to start server', err);
        process.exit(1);
    });
```

Обработчик ошибок возвращает JSON-ответы с message и details.

5. Подключение БД и миграции

Конфигурация вынесена в src/config/settings.ts и src/config/data-source.ts:

```
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;
```

Скрипты в package.json позволяют выполнять миграции:

"migration:run": "typeorm-ts-node-commonjs migration:run -d src/config/data-source.ts"

6. Авторизация

Реализована через JWT (middleware src/middlewares/auth.ts).
TSOA вызывает expressAuthentication при использовании security-схемы bearerAuth.

```
const token = auth.slice(7);

try {

            const    payload    =    jwt.verify(token,
SETTINGS.JWT_SECRET_KEY) as JwtPayload;

   (request as any).user = payload;

   return payload;

} catch {

      throw  new  Unauthorized('Invalid  or  expired
token');

}
```

Вывод

В рамках ЛР1 реализован boilerplate-проект на Express + TypeScript + TypeORM. Определены все модели данных и миграции. Настроена структура проекта с явным разделением: модели, сервисы, контроллеры, роуты. Подключена база данных PostgreSQL и система миграций. Реализована авторизация через JWT. Организована централизованная обработка ошибок и удобные скрипты запуска.