САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:

Гнеушев Владислав

Группа К3339

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

1. Были реализованы *Entity* модели данных, исходя из схемы базы данных в ДЗ 1.

Пример *Entity* соискателя:

```
@Entity()
export class EmployeeCabinet {
    @PrimaryGeneratedColumn()
    id: number

    @OneToOne(() => User)
    @JoinColumn()
    user: User

    @Column({ length: 128 })
    first_name: string

    @Column({ length: 128 })
    last_name: string

    @Column({ length: 4096, nullable: true })
    resume_text: string

    @Column()
    updated_resume_at: Date

    @CreateDateColumn()
    created_at: Date

    @OneToMany(() => Resume, resume => resume.employee)
    resumes: Resume[]

    @OneToMany(() => EmployeeApplication, application => application.employee)
    applications: EmployeeApplication[]
}
```

2. Далее были реализованы контроллеры с CRUD-методами для каждого *Entity*.

Пример контроллера *EmployeeCabinetController*:

```
export class EmployeeCabinetController {
    private repository: Repository<EmployeeCabinet>

    constructor() {
        this.repository = myDataSource.getRepository(EmployeeCabinet)
    }

    async getAll(req: Request, res: Response) {
        try {
            const items = await this.repository.find()
            res.json(items)
        } catch (error) {
            res.status(500).json({ message: "Error fetching employee cabinets" })
        }
    }

    async getById(req: Request, res: Response) {
        try {
            const item = await this.repository.findOneBy({ id: parseInt(req.params.id) })
            if (!item) {
                return res.status(404).json({ message: "Employee cabinet not found" })
            }
            res.json(item)
        } catch (error) {
            res.status(500).json({ message: "Error fetching employee cabinet" })
        }
    }

    async create(req: Request, res: Response) {
```

Контроллеры будут обращаться к базе данных при помощи репозитория, управляющего нужным классом *Entity*. Репозиторий будет получен из *DataSource* (источника данных) TypeORM.

3. Далее были созданы эндпоинты, и к ним прикреплены соответствующие им контроллеры.

Пример CRUD-эндпоинтов для управления соискателями:

```
app.get("/employee-cabinets", function(req, res) { employeeCabinetController.getAll(req, res) })
app.get("/employee-cabinets/user/:userId", function(req, res) { employeeCabinetController.getByUserId(req, res) })
app.get("/employee-cabinets/:id", function(req, res) { employeeCabinetController.getById(req, res) })
app.post("/employee-cabinets", function(req, res) { employeeCabinetController.create(req, res) })
app.patch("/employee-cabinets/:id", function(req, res) { employeeCabinetController.update(req, res) })
app.delete("/employee-cabinets/:id", function(req, res) { employeeCabinetController.delete(req, res) })
```

Инициализация приложения выглядела так:

```
const app = express()
app.use(express.json())

myDataSource.initialize()
    .then(() => {
        console.log("Data Source has been initialized!")
    })
    .catch((err) => {
        console.error("Error during Data Source initialization:", err)
    })

const userController = new UserController()
const employeeCabinetController = new EmployeeCabinetController()
const employerCabinetController = new EmployerCabinetController()
const resumeController = new ResumeController()
const skillController = new SkillController()
const jobOfferController = new JobOfferController()
const jobCategoryController = new JobCategoryController()
const companyController = new CompanyController()
```

Вывод

В данной лабораторной работе были созданы базовые эндпоинты для управления всеми сущностями в информационной системе “Сайт для поиска работы” при помощи Express и TypeORM. В коде не было предусмотрено качественного разделения на слои, что может усложнить масштабирование и поддержку проекта в будущем. Также отсутствует валидация входных данных, централизованная обработка ошибок и логирование.