**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

# Домашняя работа №2

Выполнил:

Русинов Василий

Группа К3340

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**

Вариант: сайт для поиска работы.

1. Реализация моделей

Были реализовани Entity модели данных по схеме базы данных из ДЗ1.

Пример Entity пользователя (остальные по аналогии):

```
@Entity()
export class User {
    @PrimaryGeneratedColumn()
    id!: number;

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

    @Column()
    name!: string;

    @Column()
    password!: string;

    @OneToMany(() => Resume, (resume) => resume.user)
    resumes!: Resume[];

    @OneToMany(() => Employer, (employer) => employer.user)
    employers!: Employer[];

    @OneToMany(() => Application, (application) => application.user)
    applications!: Application[];
}
```

2. Реализация CRUD-методов

Для каждой модели был реализован сервис, содержащий CRUD-методы, также, реализованы контроллеры и роутеры.

Пример сервиса для пользователя:

```
export class UserService {
    private userRepo = AppDataSource.getRepository(User);

    async create(data: Partial<User>): Promise<User> {
```

```
        const user = this.userRepo.create(data);
        return this.userRepo.save(user);
    }

    async getAll(): Promise<User[]> {
        return this.userRepo.find();
    }

    async getById(id: number): Promise<User | null> {
        return this.userRepo.findOneBy({ id });
    }

    async getByEmail(email: string): Promise<User | null> {
        return this.userRepo.findOneBy({ email });
    }

    async update(id: number, data: Partial<User>): Promise<User | null> {
        const user = await this.getById(id);
        if (!user) return null;
        Object.assign(user, data);
        return this.userRepo.save(user);
    }

    async delete(id: number): Promise<boolean> {
        const res = await this.userRepo.delete(id);
        return res.affected !== 0;
    }
}
```

Пример контроллера для пользователя:

```
const service = new UserService();

export class UserController {
    async create(req: Request, res: Response) {
        const user = await service.create(req.body);
        res.json(user);
    }

    async getAll(req: Request, res: Response) {
        const users = await service.getAll();
        res.json(users);
    }

    async getById(req: Request, res: Response) {
        const user = await service.getById(Number(req.params.id));
        if (!user) return res.status(404).send("User not found");
        res.json(user);
    }

    async getByEmail(req: Request, res: Response) {
        const user = await service.getByEmail(req.params.email);
        if (!user) return res.status(404).send("User not found");
        res.json(user);
    }

    async update(req: Request, res: Response) {
        const user = await service.update(Number(req.params.id), req.body);
        if (!user) return res.status(404).send("User not found");
        res.json(user);
    }
```

```
    async delete(req: Request, res: Response) {
        const ok = await service.delete(Number(req.params.id));
        if (!ok) return res.status(404).send("User not found");
        res.status(204).send();
    }
}
```

Пример роутера для пользователя (API-эндпоинт для получения пользователя по id/email):

```
const router = Router();
const controller = new UserController();

router.post("/", controller.create.bind(controller));
router.get("/", controller.getAll.bind(controller));
router.get("/email/:email", controller.getByEmail.bind(controller));
router.get("/:id", controller.getById.bind(controller));
router.put("/:id", controller.update.bind(controller));
router.delete("/:id", controller.delete.bind(controller));

export default router;
```

3. Инициализация приложения

```
AppDataSource.initialize()
    .then(() => {
        console.log("Data Source has been initialized!");
        app.listen(PORT, () => console.log(`Server running on port
${PORT}`));
    })
    .catch((err) => {
        console.error("Error during Data Source initialization", err);
    });
```

```
const app = express();

app.use(express.json());
app.use("/users", userRoutes);
app.use("/resumes", resumeRoutes);
app.use("/employers", employerRoutes);
app.use("/jobs", jobRoutes);
app.use("/industries", industryRoutes);
app.use("/applications", applicationRoutes);

export default app;
```

**Вывод**

В данной лабораторной работе были созданы все модели данных, спроектированные в рамках Д31, и для работы с ними реализованы CRUD-методы средствами Express + TypeScript. Также реализован API-эндпоинт для получения пользователя по id/email.