

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнила:

Красюк Карина

К3341

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

## Задача

- 1) Реализовать все модели данных, спроектированные в рамках ДЗ1
- 2) Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
- 3) Реализовать API-эндпоинт для получения пользователя по id/email

## Ход работы

Вариант: сайт для поиска работы

### 1 Реализация моделей

Для начала были реализованы Entity модели данных, исходя из схемы базы данных в ДЗ 1 с помощью TypeORM. На рисунке 1 представлена модель образования (Education) в резюме. Все оставшиеся модели были созданы по аналогии с показанной сущностью.

```
@Entity( name: 'educations')
export class Education {
    @PrimaryGeneratedColumn( strategy: 'uuid')
    id: string;

    @ManyToOne( typeFunctionOrTarget: () => Resume, inverseSide: resume : Resume => resume.educations)
    @JoinColumn( options: { name: 'resume_id' })
    resume: Resume;

    @Column( type: 'varchar', options: { length: 255 })
    institute: string;

    @Column( type: 'varchar', options: { length: 255, nullable: true })
    degree?: string;

    @Column( type: 'varchar', options: { length: 255, nullable: true })
    specialization?: string;

    @Column( options: { type: 'date' })
    start_date: string;

    @Column( options: { type: 'date', nullable: true })
    end_date?: string;
}
```

Рисунок 1 – Модель “Education” в резюме

### 2 Реализация CRUD методов

Для каждой модели разработан отдельный сервис, содержащий CRUD-методы и взаимодействующий напрямую с репозиториями. Затем, в соответствии с

бизнес-логикой приложения, для необходимых моделей были реализованы контроллеры и роутеры.

Пример связки service + controller + router для сущности users представлен на рисунках 2, 3, 4

```
export class UserService {
  private repository : Repository<User> = AppDataSource.getRepository(User);

  Show usages
  async getById(id: string) : Promise<User> {
    const user : User = await this.repository.findOneBy( where: {id});
    if (!user) return null;
    return user;
  }

  Show usages
  async getEmail(email: string) : Promise<User> {
    const user : User = await this.repository.findOneBy( where: {email});
    if (!user) return null;
    return user;
  }

  no usages
  async create(data: Partial<User>) : Promise<User> {
    const user : User = this.repository.create(data);
    return this.repository.save(user);
  }

  no usages
  async update(id: string, data: Partial<User>) : Promise<UpdateResult> {
    await this.getById(id);
    return this.repository.update(id, data);
  }

  no usages
  async delete(id: string) : Promise<DeleteResult> {
    await this.getById(id);
    return this.repository.delete(id);
  }

  no usages
  async getAll() : Promise<User[]> {
    return this.repository.find();
  }
}
```

Рисунок 2 – UserService

```

export class UserController {
  private userService: UserService;

  Show usages
  constructor() {
    this.userService = new UserService();
  }

  Show usages
  getUserById: RequestHandler = async (request : Request<ParamsDictionary, any,... , response : Response<any, Record<string, a... ) : Promise<void> => {
    const id : string = request.params.id;
    const user : User = await this.userService.getById(id);
    if (!user) {
      response.status( code: 404).json( body: {message: "User not found"});
    }
    response.status( code: 200).json(user);
  };

  Show usages
  getUserByMail: RequestHandler = async (request : Request<ParamsDictionary, any,... , response : Response<any, Record<string, a... ) : Promise<void> => {
    const mail : string = request.params.mail;
    const user : User = await this.userService.getByEmail(mail);
    if (!user) {
      response.status( code: 404).json( body: {message: "User not found"});
    }
    response.status( code: 200).json(user);
  };
}

```

Рисунок 3 – UserController

```

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

router.get( path: "/:id", controller.getUserById);
router.get( path: '/by-mail/:mail', controller.getUserByMail);

Show usages
export default router;

```

Рисунок 4 – UserRouter

### 3 Реализация API-эндпоинтов

В качестве заключительного задания были реализован отдельный API-эндпоинт для модели пользователя. Он позволяет найти пользователя по id или email. На рисунке 5 представлен этот эндпоинт.

```

export class UserController {
    private userService: UserService;

    Show usages
    constructor() {
        this.userService = new UserService();
    }

    Show usages
    getUserById: RequestHandler = async (request : Request<ParamsDictionary, any,... , response : Response<any, Record<string, a... > : Promise<void> => {
        const id : string = request.params.id;
        const user : User = await this.userService.getById(id);
        if (!user) {
            response.status( code: 404).json( body: {message: "User not found"});
        }
        response.status( code: 200).json(user);
    });

    Show usages
    getUserByEmail: RequestHandler = async (request : Request<ParamsDictionary, any,... , response : Response<any, Record<string, a... > : Promise<void> => {
        const mail : string = request.params.mail;
        const user : User = await this.userService.getByEmail(mail);
        if (!user) {
            response.status( code: 404).json( body: {message: "User not found"});
        }
        response.status( code: 200).json(user);
    });
}

```

Рисунок 5 – Контроллер для поиска пользователя по id или email

Инициализацию приложения можно увидеть на рисунке 6.

```

const app : Express = express();

AppDataSource
    .initialize() Promise<DataSource>
    .then(() : void => {
        console.log('Data Source has been initialized!');
    }) Promise<void>
    .catch((err) : void => {
        console.error('Error during Data Source initialization:', err);
    });

app.use(express.json());
app.use('/users', userRouter);

app.listen( port: 3000, callback: () => console.log("Server running at http://localhost:3000"));

```

Рисунок 6 – Инициализация приложения

## Вывод

В ходе работы были успешно реализованы все модели данных из ДЗ1, после чего для каждой из них разработаны CRUD-методы с использованием Express и TypeScript. Для работы с данными созданы сервисы, взаимодействующие с репозиториями, а также контроллеры и роутеры там, где это требовала бизнес-логика. В рамках задачи был реализован API-эндпоинт, позволяющий

получать пользователя по id или email. Также отсутствует валидация входных данных, централизованная обработка ошибок и логирование.