

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Реализация REST API на основе boilerplate

Выполнил:

Петухов Семён

Группа
K3339

Проверил:
Добряков Д. И.

Санкт-Петербург

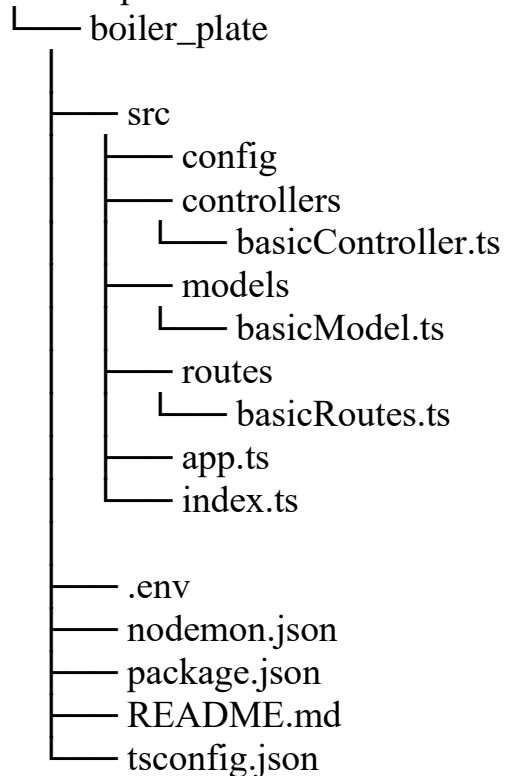
2025 г.

Задача

По выбранному варианту необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Ход работы

На основе реализованного ранее boilerplate`а был сделана реализация REST API с моделями базы данных из варианта на тему разработки сайта для поиска работы



На основе базовой модели была разработана модель пользователя

```
import {
  Entity,
  PrimaryGeneratedColumn,
  Column,
  ManyToOne,
  OneToMany,
} from "typeorm";
import { Resume } from "../resumeModel";
import { Company } from "../companyModel";
import { Application } from "../applicationModel";
import { MotivationLetter } from "../motivation_letterModel";

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

  @Column()
  username: string;

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

```

@Column()
password: string;

@Column()
role: "соискатель" | "работодатель";

@ManyToOne(() => Company, (company) => company.users, { nullable: true })
company: Company | null;

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

@OneToMany(() => Application, (app) => app.user)
applications?: Application[];

@OneToMany(() => MotivationLetter, (ml) => ml.user)
motivationLetters?: MotivationLetter[];

constructor(
  username: string,
  email: string,
  password: string,
  role: "соискатель" | "работодатель",
  company?: Company
) {
  this.username = username;
  this.email = email;
  this.password = password;
  this.role = role;
  this.company = company || null;
}
}

```

Далее была описан контроллер, с описанием CRUD запросов к модели выше. В каждой функции описан конкретный запрос со структурой нашей модели

```

import { Request, Response } from "express";
import { AppDataSource } from "../config/data-source";
import { User } from "../models/userModel";

const userRepo = AppDataSource.getRepository(User);

export const getAllUsers = async (req: Request, res: Response): Promise<void> => {
  const users = await userRepo.find({ relations: ["company"] });
  res.json(users);
};

export const getUserById = async (req: Request, res: Response): Promise<void> => {
  const user = await userRepo.findOne({
    where: { id: parseInt(req.params.id) },
    relations: ["company"],
  });
  if (!user) {
    res.status(404).json({ message: "User not found" });
    return;
  }
  res.json(user);
};

export const createUser = async (req: Request, res: Response) => {
  const user = userRepo.create(req.body);
  await userRepo.save(user);
  res.status(201).json(user);
};

```

```

};

export const updateUser = async (req: Request, res: Response): Promise<void>
=> {
  const user = await userRepo.findOneBy({ id: parseInt(req.params.id) });
  if (!user) {
    res.status(404).json({ message: "User not found" });
    return;
  }

  userRepo.merge(user, req.body);
  await userRepo.save(user);
  res.json(user);
};

export const deleteUser = async (req: Request, res: Response) => {
  const result = await userRepo.delete(parseInt(req.params.id));
  res.json({ deleted: result.affected });
};

```

Далее был описан файл с эндпоинтами, которые ссылаются на функции из контроллера

```

import { Router } from "express";
import {
  getAllUsers,
  getUserById,
  createUser,
  updateUser,
  deleteUser,
} from "../controllers/userController";

const router = Router();
router.get("/", getAllUsers);
router.get("/:id", getUserById);
router.post("/", createUser);
router.put("/:id", updateUser);
router.delete("/:id", deleteUser);

export default router;

```

Остальные модели реализуются аналогичным образом

Вывод

По результатам работы был разработана модель базы данных и CRUD-запросы к ней в нотации REST API