САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:
Ребров С. А.

Группа:
K3339

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

Для начала был подготовлен проект для работы с базой данных, установлены все необходимые зависимости для работы, а также для компиляции TypeScript.

В процессе разработки системы были спроектированы модели данных для сущностей User, Role, Resume, Job, Application, Experience, Education, которые отражают структуру базы данных и взаимоотношения между сущностями. Для каждой модели был создан файл в директории src/entity, который включает аннотации TypeORM для работы с таблицами базы данных.

Для каждой сущности были реализованы функции контроллеров для выполнения CRUD-операций.

Был реализован REST API с использованием маршрутов в Express для работы с моделями данных. Для каждой сущности создан свой набор маршрутов.

Пример кода для Role, остальные entity были написаны подобным образом:

```
import {
  Entity,
  BaseEntity,
  PrimaryGeneratedColumn,
  Column,
```

```
  OneToMany,
} from "typeorm";
import { User } from "./user";
import { Request, Response, NextFunction, Router } from "express";

@Entity({ name: "Role" })
export class Role extends BaseEntity {
  @PrimaryGeneratedColumn({ name: "role_id" })
  id: number;

  @Column({ type: "varchar", length: 256 })
  name: string;

  @OneToMany(() => User, (user) => user.role)
  users: User[];
}

export const createRole = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const { name } = req.body;
    const role = new Role();
    role.name = name;
    await role.save();
    res.status(201).json(role);
  } catch (error) {
    next(error);
  }
```

```
};

export const getRoles = async (_req: Request, res: Response, next: NextFunction) => {
  try {
    const roles = await Role.find();
    res.json(roles);
  } catch (error) {
    next(error);
  }
};

export const getRoleById = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const id = parseInt(req.params.id, 10);
    if (isNaN(id)) {
      res.status(400).json({ message: "Invalid ID format" });
      return;
    }

    const role = await Role.findOne({ where: { id } });
    if (!role) {
      res.status(404).json({ message: "Role not found" });
      return;
    }
    res.json(role);
  } catch (error) {
```

```
    next(error);
  }
};

export const updateRole = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const id = parseInt(req.params.id, 10);
    const { name } = req.body;

    const role = await Role.findOne({ where: { id } });
    if (!role) {
      res.status(404).json({ message: "Role not found" });
      return;
    }

    role.name = name;
    await role.save();
    res.json(role);
  } catch (error) {
    next(error);
  }
};

export const deleteRole = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const id = parseInt(req.params.id, 10);
```

```
    const role = await Role.findOne({ where: { id } });
    if (!role) {
      res.status(404).json({ message: "Role not found" });
      return;
    }

    await role.remove();
    res.json({ message: "Role deleted successfully" });
  } catch (error) {
    next(error);
  }
};

const router = Router();
router.post("/roles", createRole);
router.get("/roles", getRoles);
router.get("/roles/:id", getRoleById);
router.put("/roles/:id", updateRole);
router.delete("/roles/:id", deleteRole);
export { router };
```

Далее был написан код для data-source.ts:

```
import { DataSource } from "typeorm";

import { User } from "./entity/user";
import { Role } from "./entity/role";
import { Resume } from "./entity/resume";
```

```
import { Job } from "./entity/job";
import { Application } from "./entity/application";
import { Education } from "./entity/education";
import { Experience } from "./entity/experience";

export const AppDataSource = new DataSource({
  type: "postgres",
  host: "localhost",
  port: 5432,
  username: "postgres",
  password: “<PASSWORD>”
  database: "find_job",
  synchronize: false,
  logging: true,
  entities: [User, Role, Resume, Job, Application, Education, Experience],
  migrations: ["src/migration/**/*.ts"],
  subscribers: []
});
```

И был написан код для index.ts для запуска приложения:

```
import "reflect-metadata";
import { AppDataSource } from "./data-source";
import express from "express";
import { router as userRouter } from "./entity/user";
import { router as roleRouter } from "./entity/role";
import { router as resumeRouter } from "./entity/resume";
import { router as jobRouter } from "./entity/job";
```

```
import { router as applicationRouter } from "./entity/application";
import { router as experienceRouter } from "./entity/experience";
import { router as educationRouter } from "./entity/education";

const app = express();

app.use(express.json());

app.use("/api", userRouter);
app.use("/api", roleRouter);
app.use("/api", resumeRouter);
app.use("/api", jobRouter);
app.use("/api", applicationRouter);
app.use("/api", experienceRouter);
app.use("/api", educationRouter);

const startServer = async () => {
  try {
    await AppDataSource.initialize();
    console.log("Data Source has been initialized!");
    app.listen(5000, () => {
      console.log("Server running on port 5000");
    });
  } catch (error) {
    console.error("Error during Data Source initialization:", error);
  }
};
```

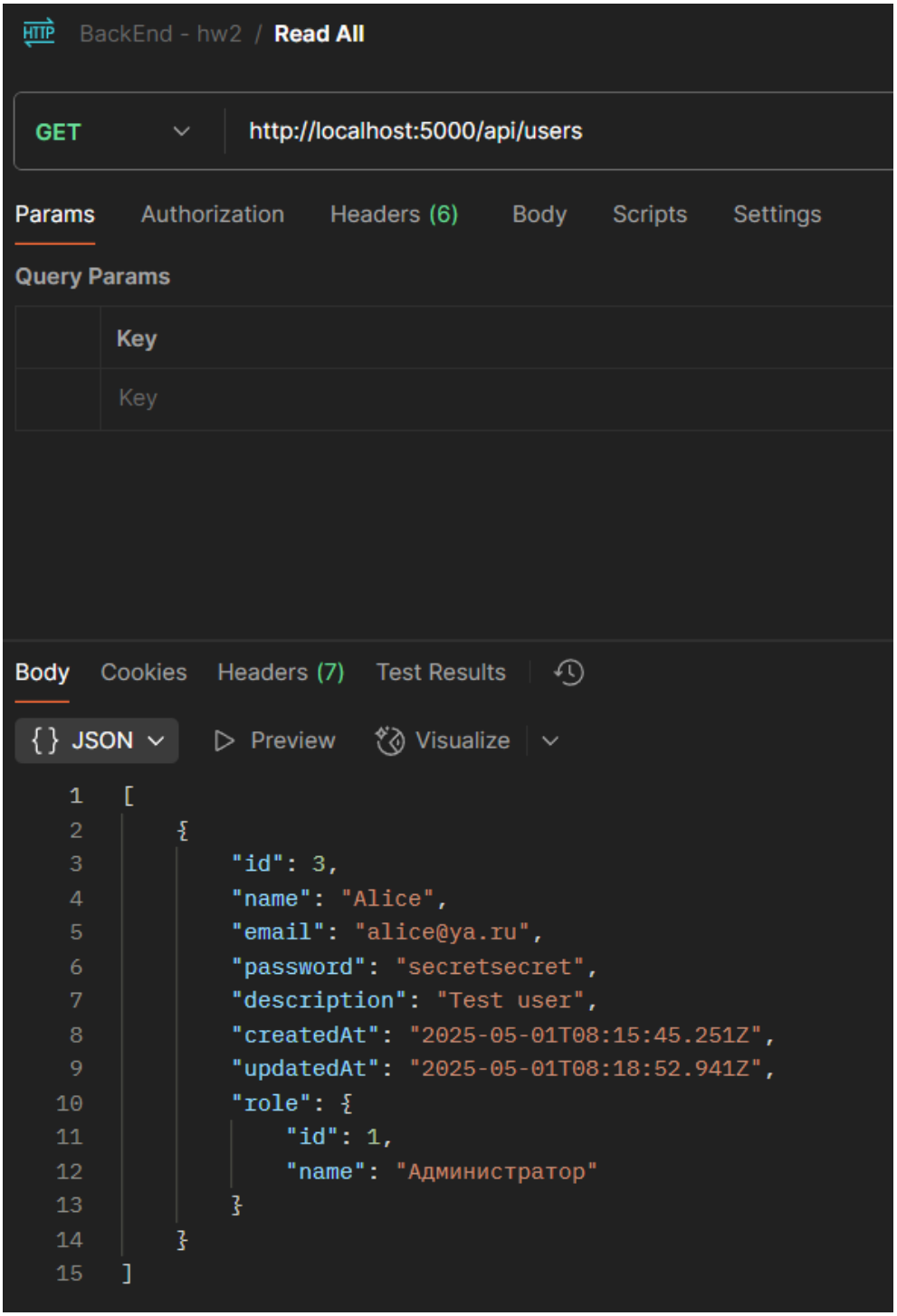
startServer();

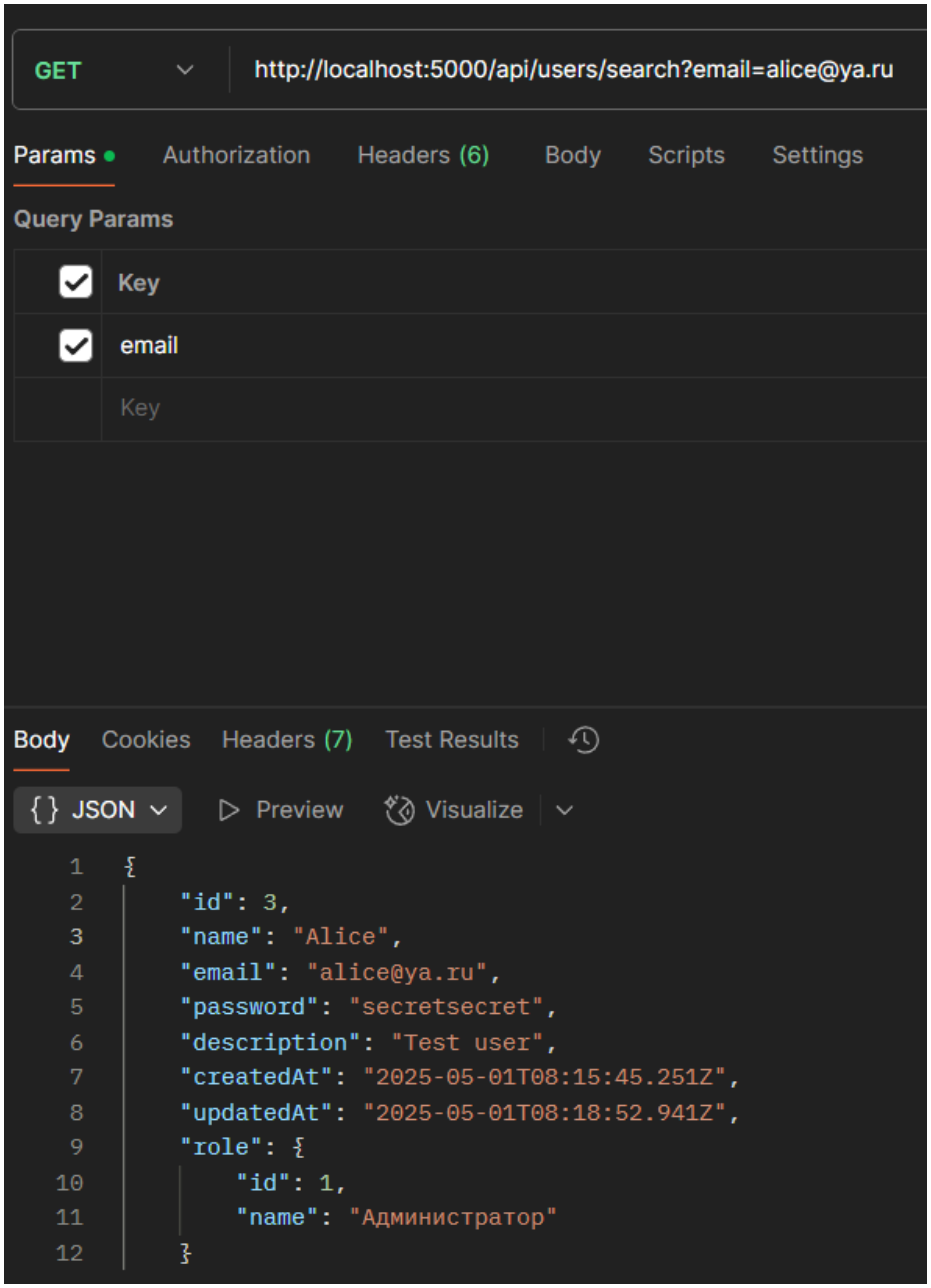
Также для таблицы Users был написан отдельный метод getUserByIdOrEmail согласно тексту задания:

```
export const getUserByIdOrEmail = async (
  req: Request,
  res: Response,
  next: NextFunction
): Promise<void> => {
  try {
    const { id, email } = req.query;

    if (id) {
      const userId = parseInt(id as string, 10);
      if (isNaN(userId)) {
        res.status(400).json({ message: "Invalid id" });
        return;
      }
      const user = await User.findOne({
        where: { id: userId },
        relations: ["role"],
      });
      if (!user) {
        res.status(404).json({ message: "User not found" });
        return;
      }
      res.json(user);
      return;
```

```
    }

    if (email) {
      const user = await User.findOne({
        where: { email: email as string },
        relations: ["role"],
      });
      if (!user) {
        res.status(404).json({ message: "User not found" });
        return;
      }
      res.json(user);
      return;
    }

    res.status(400).json({ message: "Please provide either id or email" });
  } catch (err) {
    next(err);
  }
};
```

После чего приложение было протестировано с помощью запросов через Postman:





Вывод

В ходе выполнения работы была разработана система на основе Express.js с использованием TypeScript и TypeORM для работы с базой данных. Были реализованы модели данных для сущностей, таких как User, Role, Resume, Job, Application, Experience, Education, и созданы соответствующие CRUD-операции для работы с ними через REST API. API было протестировано с использованием Postman, что позволило убедиться в

корректности всех функций, включая поиск по id и email. Проект продемонстрировал успешное взаимодействие между сервером, базой данных и API, а также продвигает понятие структурированного подхода к проектированию с разделением на модели, контроллеры и маршруты.