**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

# **Дисциплина:** Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:
Ребров С. А.

Группа:
K3339

Проверил:
Добряков Д. И.

Санкт-Петербург

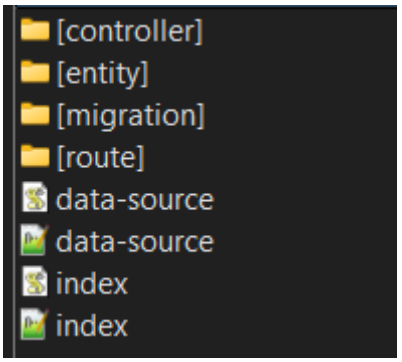
2025 г.

**Задача**

1. Нужно написать свой boilerplate на express + TypeORM + typescript.
2. Должно быть явное разделение на:
   - модели
   - контроллеры
   - роуты

**Ход работы**

За основу были взяты файлы из прошлой работы, и каждый был разделен на три файла entity, controller и router.




Пример разделения для Role:

```
// role entity
import {
  Entity,
  BaseEntity,
  PrimaryGeneratedColumn,
  Column,
  OneToMany
} from "typeorm";
import { User } from "./user";

@Entity({ name: "Role" })
```

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

  @Column({ type: "varchar", length: 100 })
  name: string;

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

// role controller
import { Request, Response, NextFunction } from "express";
import { Role } from "../entity/role";

export const createRole = async (
  req: Request,
  res: Response,
  next: NextFunction
): Promise<void> => {
  try {
    const { name } = req.body;
    if (!name) {
      res.status(400).json({ message: "Missing 'name'" });
      return;
    }
    const role = new Role();
    role.name = name;
```

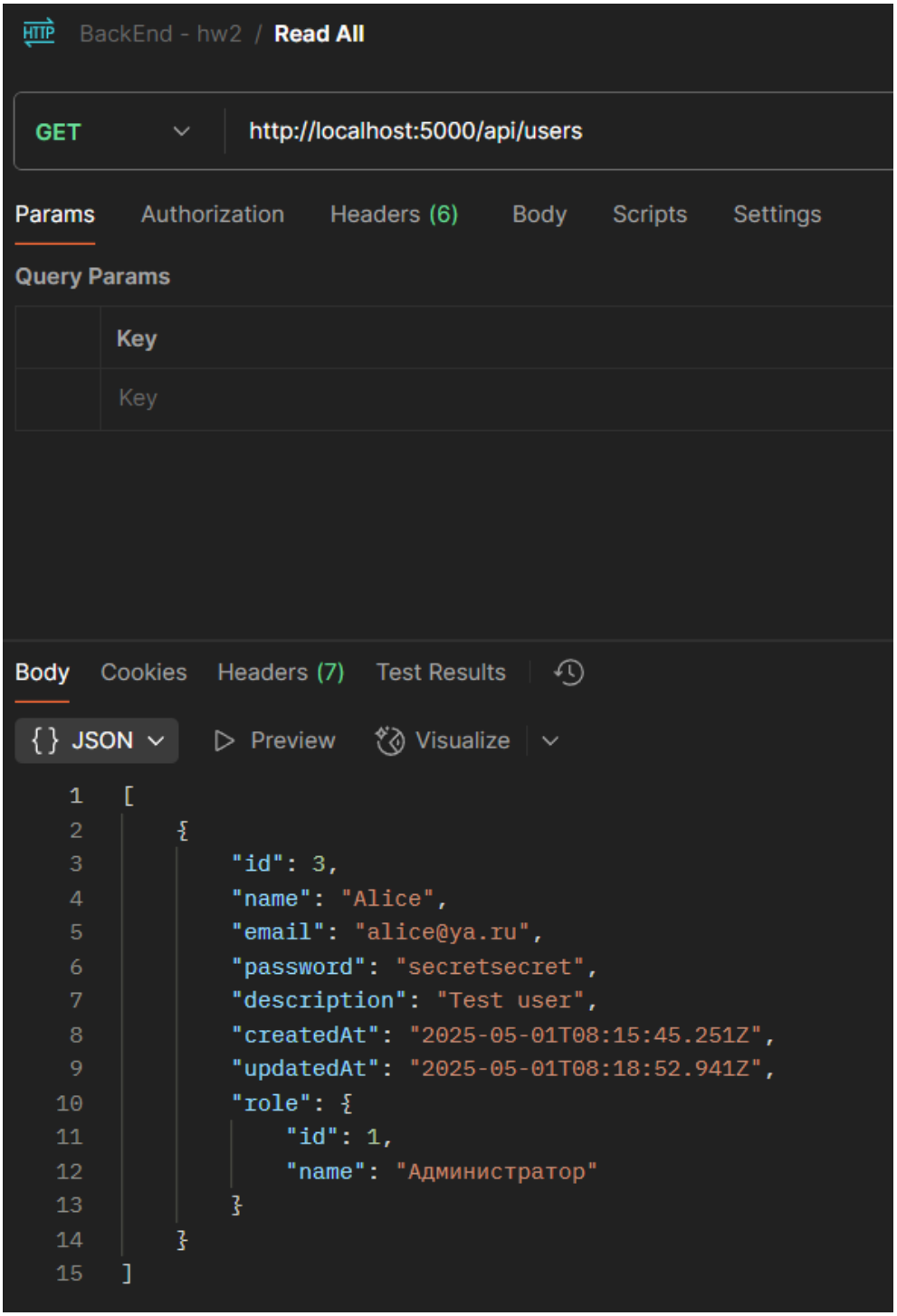
```
    await role.save();
    res.status(201).json(role);
  } catch (error) {
    next(error);
  }
};

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

export const getRoleById = async (
  req: Request,
  res: Response,
  next: NextFunction
): Promise<void> => {
  try {
    const id = parseInt(req.params.id, 10);
    if (isNaN(id)) {
```

```
      res.status(400).json({ message: "Invalid ID" });
      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
): Promise<void> => {
  try {
    const id = parseInt(req.params.id, 10);
    const { name } = req.body;
    if (isNaN(id) || !name) {
      res.status(400).json({ message: "Invalid data" });
      return;
    }
    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
): Promise<void> => {
  try {
    const id = parseInt(req.params.id, 10);
    if (isNaN(id)) {
      res.status(400).json({ message: "Invalid ID" });
      return;
    }
    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);
  }
};

// role route
import { Router } from "express";
import * as roleController from "../controller/role.controller";

const router = Router();

router.post("/roles", roleController.createRole);
router.get("/roles", roleController.getRoles);
router.get("/roles/:id", roleController.getRoleById);
router.put("/roles/:id", roleController.updateRole);
router.delete("/roles/:id", roleController.deleteRole);

export default router;
```

После этого все приложение было еще раз успешно протестировано через Postman.



**Вывод**

В рамках выполнения работы была проведена реорганизация исходного проекта с разделением кода на отдельные файлы для entity, controller и route, что позволило улучшить структуру и читабельность кода. Каждая сущность была вынесена в отдельный файл, где определены соответствующие модели с аннотациями для работы с TypeORM. Контроллеры были выделены в отдельные модули, что обеспечило обработку бизнес-логики и взаимодействие с данными, а маршруты были вынесены в отдельные файлы для лучшего управления API-эндпоинтами. Это улучшение структуры проекта повысило гибкость, удобство тестирования и поддержку кода, а также соответствовало лучшим практикам разработки приложений на Express.js с использованием TypeScript.