САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

ЛР 2 - реализация REST API на основе boilerplate

Выполнил:

Сергеев Виктор

K3341

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 boilerplate, реализованного в лабораторной работе 1. С помощью базовых CRUD-контроллеров были прописаны контроллеры для каждой модели. Для этих контроллеров были созданы роутеры с помощью встроенных в фреймворк Express. На рисунке 1 представлен пример роутера.

src > routes > TS UserRouter.ts > ...

```ts
import { Router } from "express";
import { UserController } from "../controllers/UserController";
import { authMiddleware } from "../middlewares/AuthMiddleware";

const userRouter = Router();
const userController = new UserController();

userRouter.get("/", userController.getAllEntities);
userRouter.get("/me", authMiddleware, userController.getMe);
userRouter.get("/:id", userController.getEntityById);
userRouter.put("/:id", userController.updateEntity);
userRouter.delete("/:id", userController.deleteEntity);

export default userRouter;
```

Рисунок 1 - UserRouter.ts

Роутер использует контроллер модели для направления в него http-запроса от клиента, и контроллер его обрабатывает, валидирует и возвращает ответ. Аналогичным образом были созданы роутеры для всех остальных моделей. На рисунке 2 представлен набор из роутеров.

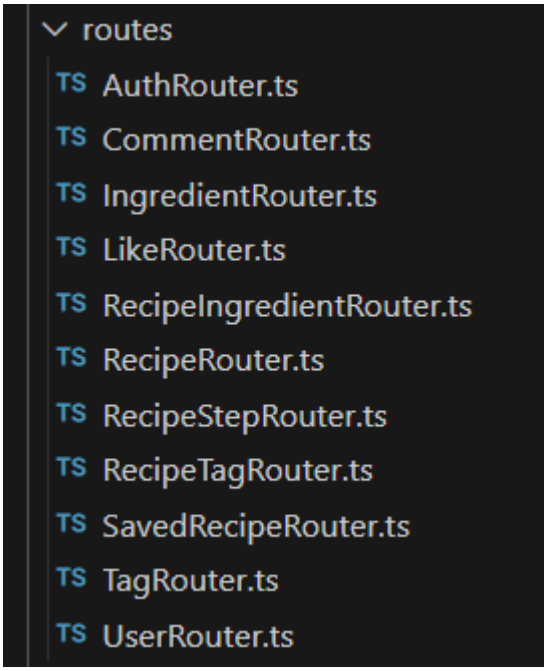

Рисунок 2 - директория routes

Использование базовых контроллеров позволяет наследовать от них контроллеры и добавлять в методы дополнительный функционал помимо основного. Так, в контроллеры были добавлены пункты валидации данных, а также добавлена возможность фильтровать некоторые модели по ключам. На рисунке 3 представлен пример контроллера с дополнительной функциональностью.

```typescript
export class RecipeController extends BaseCRUDController<Recipe>{
    protected service: RecipeService;

    constructor() {
        super(Recipe);
        this.service = new RecipeService();
    }

    async createEntity(
        req: Request,
        res: Response
    ): Promise<void> {
        const errors = new ValidationErrors();
        validateRecipe(req.body, errors);
        if (errors.isError) {
            res.status(422).json({errors: errors.errors});
            return;
        }
        super.createEntity(req, res);
    }

    async getAllEntities (
        req: Request,
        res: Response
    ): Promise<void> {
        const authorId = req.query.author;
        if (!authorId) {
            super.getAllEntities(req, res);
            return;
        }

        try {
            const entities = await this.service.getAllByAuthorId(Number(authorId));
            res.json(entities);
        } catch (e) {
            res.status(400).json({message: "Invalid param"});
        }
    }
```

Рисунок 3 - RecipeController.ts

Таким образом, были созданы все контроллеры и роутеры, что позволило создать полный API, который удовлетворяет принципам REST.

Вывод

В процессе работы был реализован REST API веб-приложения на стеке express + typescript.