САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Захарчук Александр

К3341

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

В рамках домашнего задания №2 были реализованы модели базы данных с использованием TypeORM для следующих сущностей:

- Comment
- Ingredient
- Like
- Recipe
- RecipeStep
- RecipeIgredient
- SavedRecipe
- Subscription
- User

Код для модели рецепта представлен в листинге 1.

Листинг 1 – модель рецепта

```
import { Entity, Column, JoinColumn, PrimaryGeneratedColumn, ManyToOne,
OneToMany } from "typeorm";
import { EntityWithMetadata } from "./EntitiyWithMetadata";
import { User } from "./User";
import { RecipeIngredient } from "./RecipeToIngredient";
import { RecipeStep } from "./RecipeStep";
import { Comment } from "./Comment";
import { Like } from "./Like";

export enum DishType {
  APPETIZER = "appetizer",
  MAIN_COURSE = "main_course",
  DESSERT = "dessert",
  SALAD = "salad",
  SOUP = "soup",
  SIDE_DISH = "side_dish",
  BREAKFAST = "breakfast",
  SNACK = "snack",
  BEVERAGE = "beverage"
}

export enum DifficultyLevel {
  BEGINNER = "beginner",
  EASY = "easy",
```

```
  MEDIUM = "medium",
  HARD = "hard",
  EXPERT = "xpert"
}

@Entity("recipe")
export class Recipe extends EntityWithMetadata {
    @PrimaryGeneratedColumn()
    recipe_id: number;

    @ManyToOne(() => User, user => user.recipes, {onDelete: "CASCADE"})
    @JoinColumn({ name: "username" })
    user: User;

    @Column()
    title: string;

    @Column("text")
    description: string;

    @Column({
        type: "enum",
        enum: DishType,
    })
    dish_type: DishType;

    @Column({
        type: "enum",
        enum: DifficultyLevel,
    })
    difficulty_level: DifficultyLevel;

    @Column("int")
    preparation_time_minutes: number;

    @Column()
    cooking_time_minutes: number;

    @OneToMany(() => RecipeIngredient, recipeIngredient =>
recipeIngredient.recipe)
    ingredients: RecipeIngredient[];

    @OneToMany(() => RecipeStep, step => step.recipe)
    steps: RecipeStep[];

    @OneToMany(() => Comment, comment => comment.recipe)
    comments: Comment[];

    @OneToMany(() => Like, like => like.recipe)
    likes: Like[];
}
```

Для всех сущностей базы данных были реализованы CRUD методы. Пример методов для ингредиентов представлен в листинге 2.

Листинг 2 – методы для работы с ингредиентами

```
import { Router } from "express";
import { Request, Response } from "express"
import { Ingredient } from "../entities/Ingredient";
import { dataSource } from "../dataSource"
import { checkJwt } from "../middleware/validateJWT";

const ingredientRouter = Router();
const ingredientRepository = dataSource.getRepository(Ingredient);

ingredientRouter.post("/", [checkJwt], async function (req: Request, res:
Response) {
    const ingredient = ingredientRepository.create(req.body);
    const results = await ingredientRepository.save(ingredient);
    res.send(results);
})

ingredientRouter.get("/", [checkJwt], async function (req: Request, res:
Response) {
    const ingredients = await ingredientRepository.find();
    res.json(ingredients);
})

ingredientRouter.get("/:id", [checkJwt], async function (req: Request, res:
Response) {
    const ingredientId = parseInt(req.params.id);
    const results = await ingredientRepository.findOneBy({
        ingredient_id: ingredientId,
    });
    if (!results) {
        res.status(404).json({ detail: `Ingredient with id ${ingredientId}
not found` });
        return;
    }
    res.send(results);
})

ingredientRouter.put("/:id", [checkJwt], async function (req: Request, res:
Response) {
    const ingredientId = parseInt(req.params.id);
    const ingredient = await ingredientRepository.findOneBy({
        ingredient_id: ingredientId,
    });
    if (!ingredient) {
        res.status(404).json({ detail: `Ingredient with id ${ingredientId}
not found` });
        return;
    }
    ingredientRepository.merge(ingredient, req.body);
```

```
    const results = await ingredientRepository.save(ingredient);
    res.send(results);
})

ingredientRouter.delete("/:id", [checkJwt], async function (req: Request,
res: Response) {
    const ingredientId = parseInt(req.params.id);
    const results = await ingredientRepository.delete(ingredientId);
    if (!results.affected || results.affected === 0) {
        res.status(404).json({ detail: `Ingredient with id ${ingredientId}
not found` });
        return;
    }
    res.send(results);
})

export default ingredientRouter;
```

Роутеры для всех сущностей подключаются к объекту приложения в главном файле. Код подключения представлен в листинге 3.

Листинг 3 – подключение роутеров

```
const app = express()

app.use(express.json())

app.use("/users", userRouter);
app.use("/recipes", recipeRouter);
app.use("/ingredients", ingredientRouter);
app.use("/recipes-ingredients", recipeIngredientRouter);
app.use("/recipe-steps", recipeStepRouter);
app.use("/saved-recipes", savedRecipeRouter);
app.use("/likes", likeRouter);
app.use("/subscriptions", subscriptionRouter);
app.use("/comments", commentRouter);

app.listen(3000)
```

Пример ответа сервиса при получении списка всех рецептов представлен в листинге 4.

Листинг 4 – Список всех рецептов

```
[
    {
        "created_at": "2025-04-06T10:56:59.948Z",
        "updated_at": "2025-04-06T10:56:59.948Z",
        "recipe_id": 1,
        "title": "Borsch",
        "description": "The red soup",
        "dish_type": "soup",
```

```
        "difficulty_level": "medium",
        "preparation_time_minutes": 90,
        "cooking_time_minutes": 70,
        "ingredients": [
            {
                "created_at": "2025-04-06T11:24:16.244Z",
                "updated_at": "2025-04-06T11:24:16.244Z",
                "id": 1,
                "quantity": 1,
                "unit": "kg",
                "ingredient": {
                    "created_at": "2025-04-06T11:07:28.879Z",
                    "updated_at": "2025-04-06T11:07:28.879Z",
                    "ingredient_id": 1,
                    "name": "Potato",
                    "description": "Basic vegetable"
                }
            },
            {
                "created_at": "2025-04-08T14:48:49.813Z",
                "updated_at": "2025-04-08T14:50:05.256Z",
                "id": 2,
                "quantity": 400,
                "unit": "g",
                "ingredient": {
                    "created_at": "2025-04-08T14:48:05.039Z",
                    "updated_at": "2025-04-08T14:48:05.039Z",
                    "ingredient_id": 3,
                    "name": "Tomato",
                    "description": "Basic vegetable"
                }
            }
        ],
        "steps": [
            {
                "created_at": "2025-04-08T15:08:59.860Z",
                "updated_at": "2025-04-08T15:08:59.860Z",
                "recipe_step_id": 1,
                "step_number": 1,
                "instruction": "Boil the water",
                "image": null,
                "video": null
            }
        ]
    }
]
```

Вывод

В ходе выполнения данной лабораторной работы был реализован RESTful API на основе имеющегося boilerplate. Для всех сущностей базы данных реализованы CRUD методы, которые соответствуют требованиям REST.