САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа 2

Выполнил:

Захарчук Александр
К3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- Реализовать все модели данных, спроектированные в рамках ДЗ1
- Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
- Реализовать API-эндпоинт для получения пользователя по id/email

Ход работы

В ходе работы были реализованы модели базы данных в соответствии со схемой, представленной на рисунке 1.

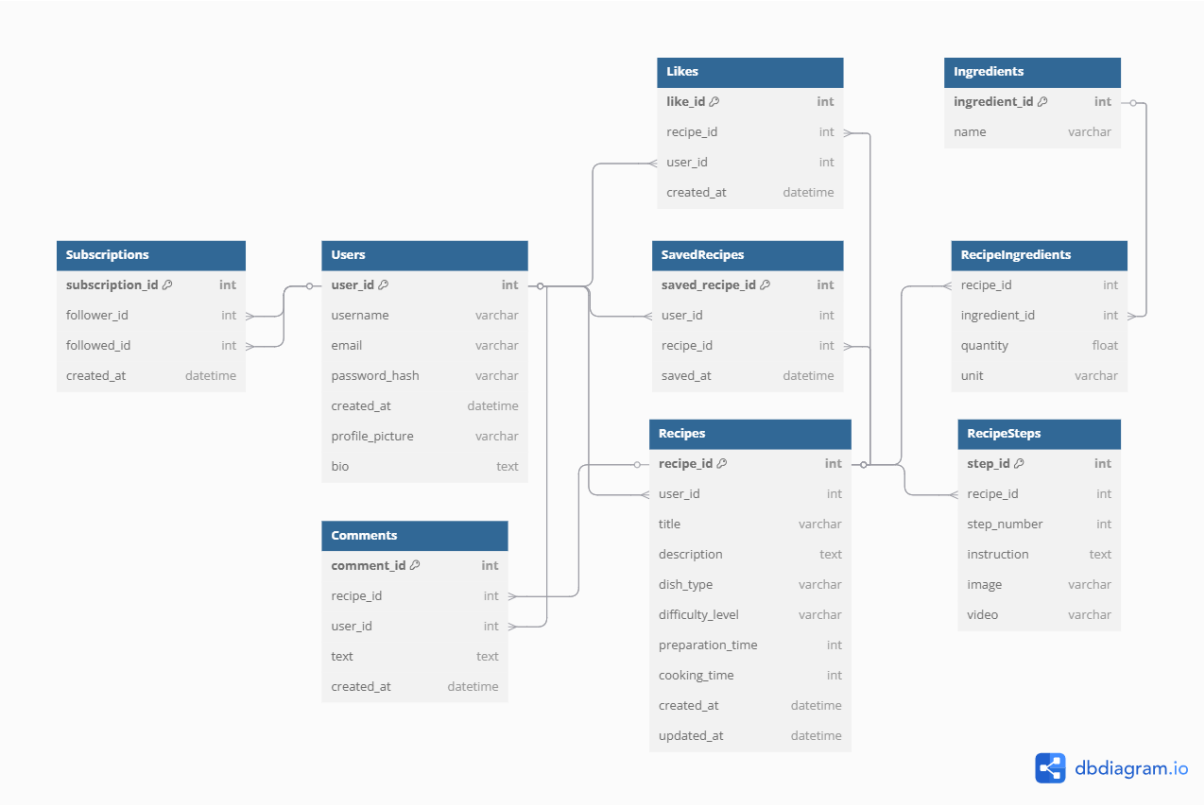

Рисунок 1 - Схема базы данных

Код, описывающий модель 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[];
}
```

Остальные модели были реализованы аналогичным образом.
Для каждой модели базы данных были также реализованы CRUD-методы.
Реализация методов для сущности рецепта представлена в листинге 2.

Листинг 2 – методы для рецепта

```
import { Router } from "express";
import { Request, Response } from "express"
import { Recipe } from "../entities/Recipe";
import { User } from "../entities/User";
import { dataSource } from "../dataSource"

const recipeRouter = Router();

const recipeRepository = dataSource.getRepository(Recipe);
const userRepository = dataSource.getRepository(User);
```

```
recipeRouter.post("/", async function (req: Request, res:
Response) {
    const username = req.get("Authorization");
    if (!username) {
        res.status(401).json({detail: "No authorization
provided"});
    }
    const user = await userRepository.findOneBy({username})

    if (!user) {
        res.status(404).json({detail: `User with username
${username} not found`});
        return;
    }

    const recipe = recipeRepository.create({...req.body,
user});
    const results = await recipeRepository.save(recipe);
    res.send(results);
})

recipeRouter.get("/", async function (req: Request, res:
Response) {
    const results = await recipeRepository
        .createQueryBuilder("recipe")
        .leftJoinAndSelect("recipe.ingredients",
"recipeIngredient")
        .leftJoinAndSelect("recipeIngredient.ingredient",
"ingredient")
        .leftJoinAndSelect("recipe.steps", "steps")
        .orderBy("steps.step_number", "ASC")
        .getMany();
    res.json(results);
})

recipeRouter.get("/:id", async function (req: Request, res:
Response) {
    const recipeId = parseInt(req.params.id);
    const results = await recipeRepository
        .createQueryBuilder("recipe")
        .leftJoinAndSelect("recipe.ingredients",
"recipeIngredient")
        .leftJoinAndSelect("recipeIngredient.ingredient",
"ingredient")
```

```
        .leftJoinAndSelect("recipe.steps", "steps")
        .where("recipe.recipe_id = :recipeId", { recipeId })
        .orderBy("steps.step_number", "ASC")
        .getOne();
    if (!results) {
        res.status(404).json({ detail: `Recipe with id
${recipeId} not found` });
        return;
    }
    res.send(results);
})

recipeRouter.put("/:id", async function (req: Request, res:
Response) {
    const recipeId = parseInt(req.params.id);
    const recipe = await recipeRepository.findOneBy({
        recipe_id: recipeId,
    });
    if (!recipe) {
        res.status(404).json({ detail: `Recipe with id
${recipeId} not found` });
        return;
    }
    recipeRepository.merge(recipe, req.body);
    const results = await recipeRepository.save(recipe);
    res.send(results);
})

recipeRouter.delete("/:id", async function (req: Request, res:
Response) {
    const recipeId = parseInt(req.params.id);
    const results = await recipeRepository.delete(recipeId);
    if (!results.affected || results.affected === 0) {
        res.status(404).json({ detail: `Recipe with id
${recipeId} not found` });
        return;
    }
    res.send(results);
})

export default recipeRouter;
```

Пример ответа от сервиса при получении списка всех рецептов представлен в листинге 3.

Листинг 3 – 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 работы были изучены и отработаны на практике инструменты Express и TypeORM, а также выполнены следующие задания:

- реализованы все модели данных, спроектированные в рамках ДЗ1,
- реализован набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript,
- реализован API-эндпоинт для получения пользователя по username.