САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа

Выполнил:

Кузнецов Артур

Группа К3340

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

В ходе выполнения домашней работы были реализованы все модели данных, в соответствии со схемой базы данных (рисунок 1).

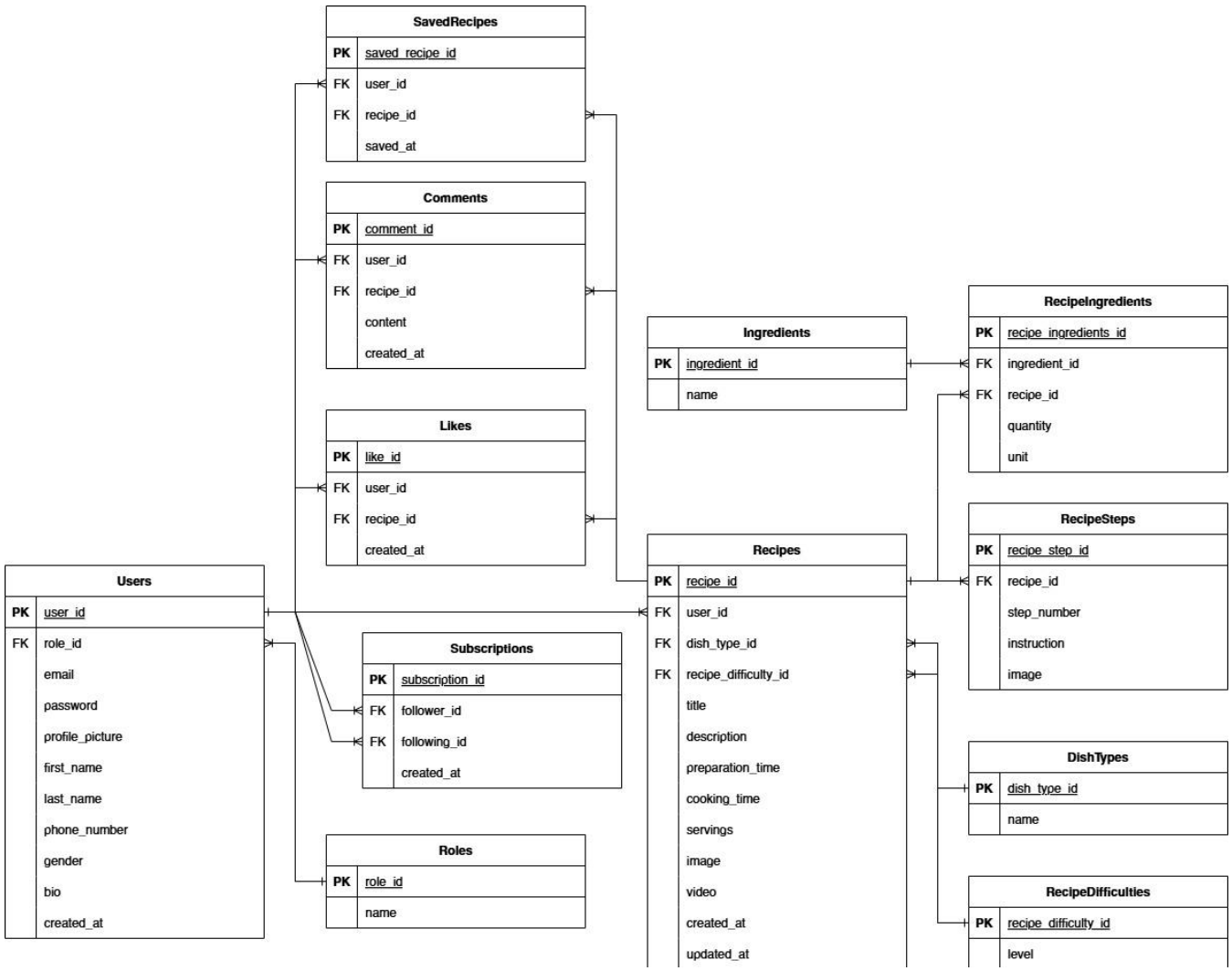

Рисунок 1 – Схема базы данных

Для каждой сущности была создана модель с использованием TypeORM. Пример реализации модели Users представлен в листинге 1.

Листинг 1 – Код модели Users

```
@Entity()
export class Users {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({type: "varchar", unique: true})
    email: string;

    @Column({type: "varchar"})
    password: string;

    @ManyToOne(() => Roles, (role) => role.users)
    role: Roles;

    @Column({type: "varchar"})
    first_name: string;

    @Column({type: "varchar"})
    last_name: string;

    @Column({type: "varchar", nullable: true})
    profile_picture: string;

    @Column({type: "varchar", nullable: true})
    phone_number: string;

    @Column({type: "varchar", nullable: true})
    gender: string;

    @Column({type: "varchar", nullable: true})
    bio: string;

    @Column({type: "timestamp", default: () => "CURRENT_TIMESTAMP"})
    created_at: Date;

    @OneToMany(() => Recipes, (recipe) => recipe.user)
    recipes: Recipes[]

    @OneToMany(() => Comments, (comment) => comment.user)
    comments: Comments[];

    @OneToMany(() => Likes, (like) => like.user)
    likes: Likes[];

    @OneToMany(() => SavedRecipes, (savedRecipe) => savedRecipe.user)
    savedRecipes: SavedRecipes[];

    @OneToMany(() => Subscriptions, (subscription) =>
subscription.follower)
    followingSubscriptions: Subscriptions[];

    @OneToMany(() => Subscriptions, (subscription) =>
subscription.following)
    followerSubscriptions: Subscriptions[];
}
```

Аналогичным образом были реализованы остальные модели: Roles, Subscriptions, Recipes, RecipeIngredients, Ingredients, SavedRecipes, Likes, Comments, RecipeSteps, RecipeDifficulties, и DishTypes.

Для каждой модели был создан контроллер с набором CRUD-методов и соответствующий роутер. Пример для модели Users представлен в листингах 2 и 3.

Листинг 2 – Код контроллера UserController

```
import { Request, Response } from "express";
import { AppDataSource } from "../config/AppDataSource";
import { Users } from "../models/Users";

const userRepository = AppDataSource.getRepository(Users);

// Создание нового пользователя
export const createUser = async function (req: Request, res: Response) {
    try {
        const userData = req.body;
        const user = userRepository.create(userData);
        const savedUser = await userRepository.save(user);
        res.status(201).json(savedUser);
    } catch (error: any) {
        res.status(500).json({message: error.message});
    }
};

// Получение всех пользователей
export const getUsers = async function (req: Request, res: Response) {
    try {
        const users = await userRepository.find();
        res.json(users);
    } catch (error: any) {
        res.status(500).json({message: error.message});
    }
};

// Получение одного пользователя по id
export const getUser = async function (req: Request, res: Response) {
    try {
        const id = Number(req.params.id);
        const user = await userRepository.findOneBy({id: id});
        if (!user) {
            return res.status(404).json({message: "User not found"});
        }
        res.json(user);
    } catch (error: any) {
        res.status(500).json({message: error.message});
    }
};

// Обновление пользователя
export const updateUser = async function (req: Request, res: Response) {
    try {
        const id = Number(req.params.id);
        const updatedData = req.body;
        const result = await userRepository.update(id, updatedData);
```

```
        if (result.affected === 0) {
            return res.status(404).json({message: "User not found"});
        }
        res.json({message: "User updated successfully"});
    } catch (error: any) {
        res.status(500).json({message: error.message});
    }
};

// Удаление пользователя
export const deleteUser = async function (req: Request, res: Response) {
    try {
        const id = Number(req.params.id);
        const result = await userRepository.delete(id);
        if (result.affected === 0) {
            return res.status(404).json({message: "User not found"});
        }
        res.status(204).send();
    } catch (error: any) {
        res.status(500).json({message: error.message});
    }
};
```

Листинг 3 – Код роутера UserRouter

```
import { Router } from "express";
import { createUser, getUsers, getUser, updateUser, deleteUser } from
"../controllers/UserController";

const router = Router();

router.post("/", createUser);
router.get("/", getUsers);
router.get("/:id", getUser);
router.put("/:id", updateUser);
router.delete("/:id", deleteUser);

export default router;
```

Аналогичные контроллеры и роутеры были созданы для всех остальных моделей: Roles, Subscriptions, Recipes, RecipeIngredients, Ingredients, SavedRecipes, Likes, Comments, RecipeSteps, RecipeDifficulties, и DishTypes.

Вывод

В ходе выполнения домашней работы успешно реализованы все модели данных, CRUD-методы для работы с ними, а также API-эндпоинт для получения пользователя по id.