САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа 2

Выполнил:

Борисова Элина

Группа К3341

Проверил:
Добряков Д. И.

Санкт-Петербург

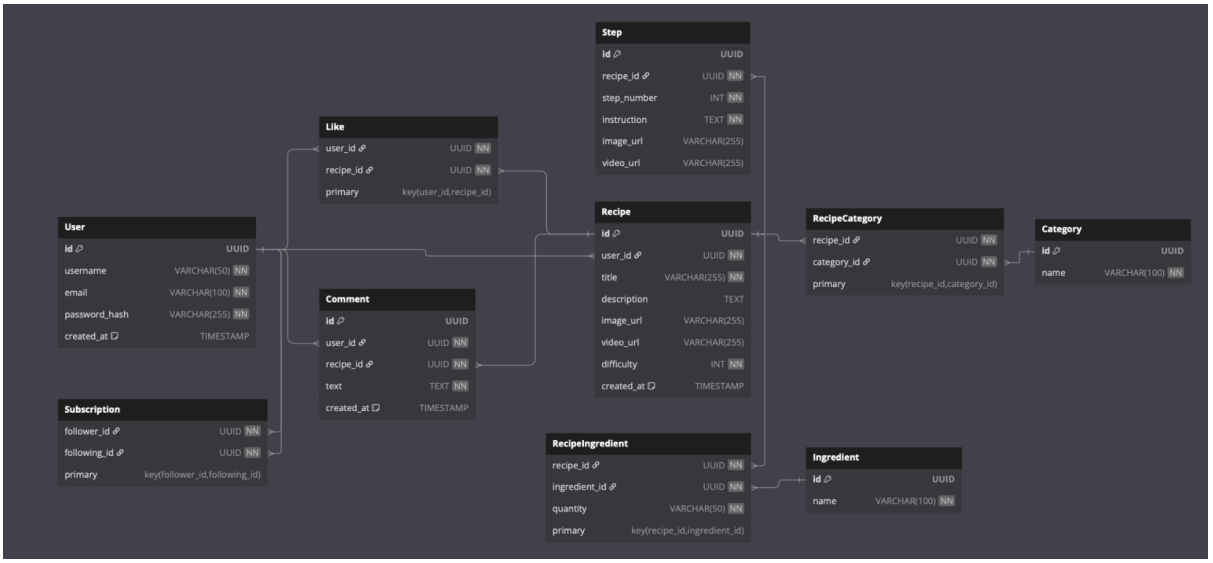
2025 г.

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

Вариант 4. Сервис для обмена рецептами и кулинарных блогов. Схема бд:



User:

```
import { Entity, PrimaryGeneratedColumn, Column,
OneToMany, ManyToMany, JoinTable, Index } from
"typeorm";
import { Recipe } from "./Recipe";
import { Comment } from "./Comment";
import { Like } from "./Like";
import { Subscription } from "./Subscription";

@Entity()
export class User {
   @PrimaryGeneratedColumn("uuid")
   id: string;

   @Column({ type: "varchar", length: 50, unique:
true })
   username: string;
```

```
    @Index()
    @Column({ type: "varchar", length: 100, unique:
true })
    email: string;

    @Column({ type: "varchar", length: 255 })
    password_hash: string;

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

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

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

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

    @OneToMany(() => Subscription, (subscription) =>
subscription.follower)
    followers: Subscription[];

    @OneToMany(() => Subscription, (subscription) =>
subscription.following)
    followings: Subscription[];

    @ManyToMany(() => Recipe, (recipe) =>
recipe.saved_by_users)
    @JoinTable()
    saved_recipes: Recipe[];
}
```

Recipe:

```
import { Entity, PrimaryGeneratedColumn, Column,
ManyToOne, OneToMany, ManyToMany, JoinTable } from
"typeorm";
import { User } from "./User";
import { RecipeCategory } from "./RecipeCategory";
import { RecipeIngredient } from
"./RecipeIngredient";
import { Comment } from "./Comment";
import { Like } from "./Like";
import { RecipeStep } from "./RecipeStep";

@Entity()
export class Recipe {
   @PrimaryGeneratedColumn("uuid")
   id: string;

   @ManyToOne(() => User, (user) => user.recipes)
   user: User;

   @Column({ type: "varchar", length: 255 })
   title: string;

   @Column({ type: "text", nullable: true })
   description: string;

   @Column({ type: "varchar", length: 255, nullable:
true })
   image_url: string;

   @Column({ type: "varchar", length: 255, nullable:
true })
   video_url: string;

   @Column({ type: "integer" })
   difficulty: number;

   @Column({ type: "timestamp", default: () =>
"CURRENT_TIMESTAMP" })
```

```
    created_at: Date;

    @OneToMany(() => RecipeCategory, (recipeCategory)
=> recipeCategory.recipe)
    categories: RecipeCategory[];

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

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

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

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

    @ManyToMany(() => User, (user) =>
user.saved_recipes)
    saved_by_users: User[];
}
```

Ingredient:

```
import { Entity, PrimaryGeneratedColumn, Column,
OneToMany } from "typeorm";
import { RecipeIngredient } from
"./RecipeIngredient";

@Entity()
export class Ingredient {
    @PrimaryGeneratedColumn("uuid")
    id: string;
```

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

    @OneToMany(() => RecipeIngredient,
(recipeIngredient) => recipeIngredient.ingredient)
    recipes: RecipeIngredient[];
}
```

Остальные модели были реализованы в проекте.
Далее представлены листинги реализованных CRUD - методов для user, для остальных моделей реализация представлена в проекте.

```
import { Router } from "express";
import { AppDataSource } from "../dataSource";
import { User } from "../entities/User";

const router = Router();

router.post("/", async (req, res) => {
    try {
        const userRepository =
AppDataSource.getRepository(User);
        const user = userRepository.create(req.body);
        const results = await
userRepository.save(user);
        return res.send(results);
    } catch (error) {
        return res.status(500).json({ error:
error.message });
    }
});

router.get("/", async (req, res) => {
    try {
        const userRepository =
AppDataSource.getRepository(User);
        const users = await userRepository.find();
```

```
        return res.send(users);
    } catch (error) {
        return res.status(500).json({ error:
error.message });
    }
});

// Get user by id
router.get("/:id", async (req, res) => {
    try {
        const userRepository =
AppDataSource.getRepository(User);
        const user = await userRepository.findOne({
            where: { id: req.params.id },
            select: ["id", "username", "email",
"created_at"]
        });

        if (!user) {
            return res.status(404).json({ error:
"User not found" });
        }

        return res.json(user);
    } catch (error) {
        return res.status(500).json({ error:
error.message });
    }
});

// Get user by email
router.get("/email/:email", async (req, res) => {
    try {
        const userRepository =
AppDataSource.getRepository(User);
        const user = await userRepository.findOne({
            where: { email: req.params.email },
```

```
            select: ["id", "username", "email",
"created_at"]
        });

        if (!user) {
            return res.status(404).json({ error:
"User not found" });
        }

        return res.json(user);
    } catch (error) {
        return res.status(500).json({ error:
error.message });
    }
});

router.put("/:id", async (req, res) => {
    try {
        const userRepository =
AppDataSource.getRepository(User);
        const user = await userRepository.findOneBy({
id: req.params.id });
        if (!user) return res.status(404).json({
error: "User not found" });

        userRepository.merge(user, req.body);
        const results = await
userRepository.save(user);
        return res.send(results);
    } catch (error) {
        return res.status(500).json({ error:
error.message });
    }
});

router.delete("/:id", async (req, res) => {
    try {
```

```
        const userRepository =
AppDataSource.getRepository(User);
        const results = await
userRepository.delete(req.params.id);
        return res.send(results);
    } catch (error) {
        return res.status(500).json({ error:
error.message });
    }
});

export default router;
```

В файле app.ts пропишем все эндпоинты:

```
import "reflect-metadata";
import express from "express";
import { AppDataSource } from "./dataSource";
import userRoutes from "./routes/users";
import recipeRoutes from "./routes/recipes";
import ingredientRoutes from "./routes/ingredients";
import commentRoutes from "./routes/comments";
import likeRoutes from "./routes/likes";
import subscriptionRoutes from
"./routes/subscriptions";
import recipeCategoryRoutes from
"./routes/recipeCategories";
import recipeIngredientRoutes from
"./routes/recipeIngredients";
import recipeStepRoutes from "./routes/recipeSteps";

const app = express();
const PORT = 3000;

app.use(express.json());

app.use("/users", userRoutes);
app.use("/recipes", recipeRoutes);
app.use("/ingredients", ingredientRoutes);
```

```
app.use("/comments", commentRoutes);
app.use("/likes", likeRoutes);
app.use("/subscriptions", subscriptionRoutes);
app.use("/recipe-categories", recipeCategoryRoutes);
app.use("/recipe-ingredients",
recipeIngredientRoutes);
app.use("/recipe-steps", recipeStepRoutes);

AppDataSource.initialize()
    .then(() => {
        console.log("Data Source has been
initialized!");
        app.listen(PORT, () => {
            console.log(`Server is running on
http://localhost:${PORT}`);
        });
    })
    .catch((err) => {
        console.error("Error during Data Source
initialization", err);
    });
```

Выше представлена реализация эндпоинтов для поиска пользователя по email и id, проверим их работу в postman. Для начала заполним базу данных. Для этого создадим файл seed.ts и добавим его в package.json:
`"seed": "ts-node src/seed.ts"`

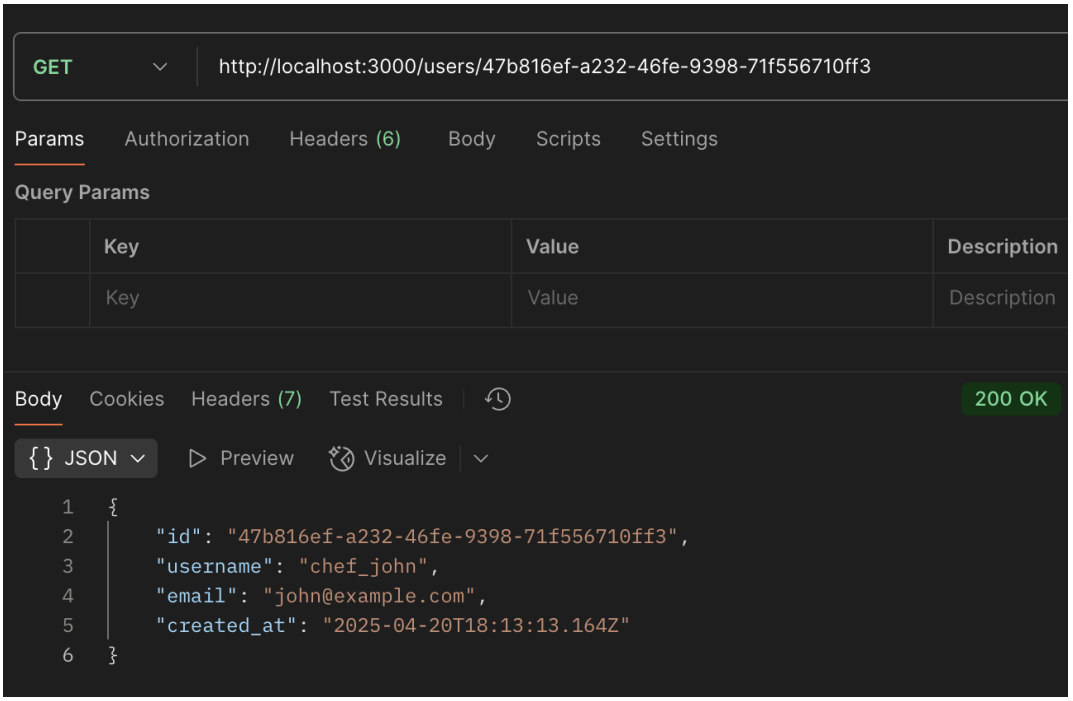
```
import { AppDataSource } from "./dataSource";
import { User } from "./entities/User";
import { Recipe } from "./entities/Recipe";
import { Ingredient } from "./entities/Ingredient";
import { RecipeIngredient } from
"./entities/RecipeIngredient";

async function seedDatabase() {
    await AppDataSource.initialize();

    const user1 = new User();
```

```
  user1.username = "chef_john";
  user1.email = "john@example.com";
  user1.password_hash = "hashed_password_123";

  const user2 = new User();
  user2.username = "baker_mary";
  user2.email = "mary@example.com";
  user2.password_hash = "hashed_password_456";

  await AppDataSource.manager.save([user1, user2]);

  const flour = new Ingredient();
  flour.name = "Мука";

  const sugar = new Ingredient();
  sugar.name = "Сахар";

  await AppDataSource.manager.save([flour, sugar]);

  const cake = new Recipe();
  cake.title = "Шоколадный торт";
  cake.description = "Простейший рецепт торта";
  cake.difficulty = 3;
  cake.user = user1;

  await AppDataSource.manager.save(cake);

  const ri1 = new RecipeIngredient();
  ri1.recipe = cake;
  ri1.ingredient = flour;
  ri1.quantity = "300 г";

  const ri2 = new RecipeIngredient();
  ri2.recipe = cake;
  ri2.ingredient = sugar;
  ri2.quantity = "200 г";

  await AppDataSource.manager.save([ri1, ri2]);
```

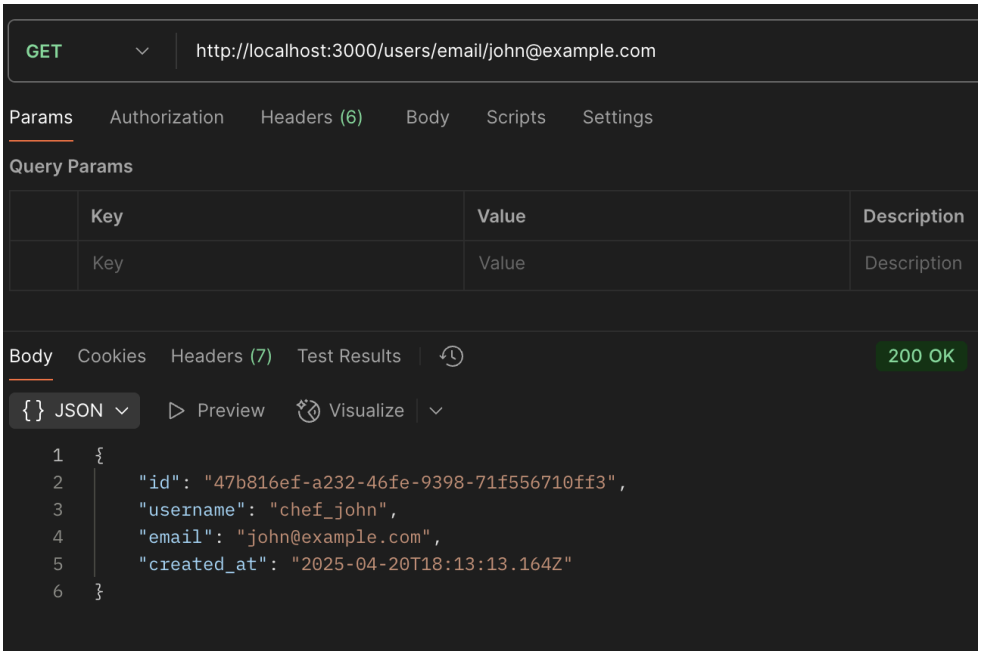
```
    console.log("База данных успешно заполнена!");
    process.exit(0);
}

seedDatabase().catch(error => {
    console.error("Ошибка при заполнении базы:",
error);
    process.exit(1);
});
```

Выполним команды:
npm run seed
npm run dev

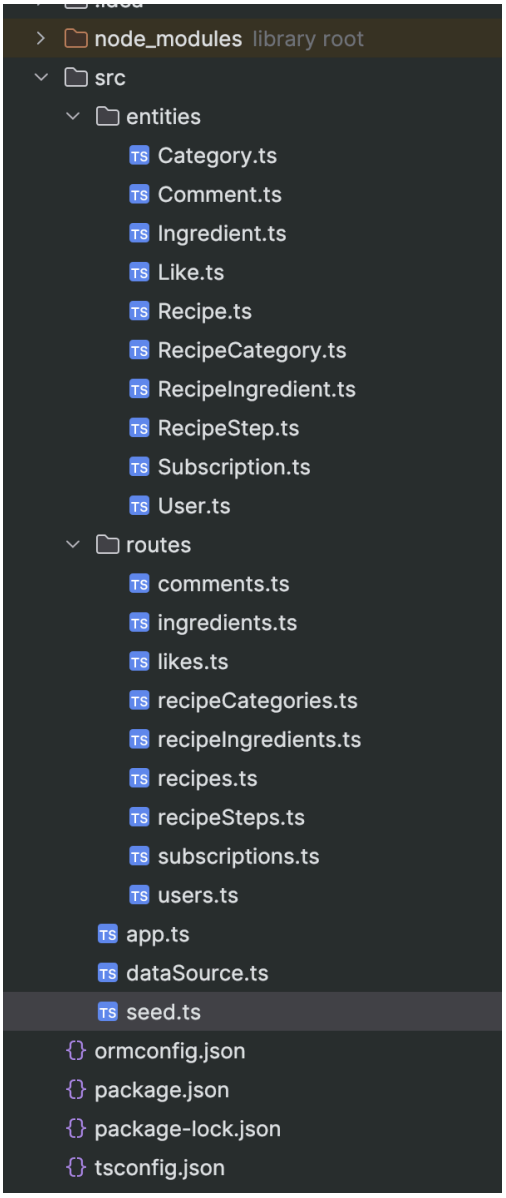
Теперь перейдем в postman. Для поиска по id в строке GET запроса вводим http://localhost:3000/users/47b816ef-a232-46fe-9398-71f556710ff3, получаем ответ с нашими данными:



Проверим поиск по email:



Итоговая структура проекта:



Вывод

В ходе данной работы API для сервиса рецептов реализован на Express и TypeORM, с корректной структурой БД и основными CRUD-операциями. Эндпоинты проверены через Postman — поиск по email, добавление рецептов и другие функции работают без ошибок.