

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа

Выполнила:

Исмагилова Карина

Группа: К3344

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Ход работы

1. Реализовать все модели данных, спроектированные в рамках ДЗ1.

Мы реализовали модели в папке entities. Пример реализации сущности Recipe:

```
import { Entity, PrimaryGeneratedColumn, Column,ManyToOne, OneToMany, ManyToMany } from "typeorm";
import { User } from "../User";
import { File } from "../File";
import { Category } from "../Category";
import { Comment } from "../Comment";
import { Favourite } from "../Favourite";
import { Rating } from "../Rating";
import { Collection } from "../Collection";

@Entity()
export class Recipe {
  @PrimaryGeneratedColumn()
  recipe_id!: number;

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

  @Column()
  title!: string;

  @Column("text")
  description!: string;

  @Column("text")
  ingredients!: string;

  @Column("text")
  instructions!: string;

  @Column()
  difficulty_level!: string;

  @Column()
  preparation_time!: number;

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

  @Column({ type: 'timestamp', default: () => 'CURRENT_TIMESTAMP', onUpdate: 'CURRENT_TIMESTAMP' })
  updated_at!: Date;

  @OneToMany(() => File, file => file.recipe)
  files!: File[];

  @OneToMany(() => Category, category => category.recipe)
```

```

categories!: Category[];

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

@OneToMany(() => Favourite, favourite => favourite.recipe)
favourites!: Favourite[];

@OneToMany(() => Rating, rating => rating.recipe)
ratings!: Rating[];

@ManyToMany(() => Collection, collection => collection.recipes)
collections!: Collection[];
}

```

2. Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript.

CRUD-методы у нас хранятся в папке services. Пример реализации CRUDов для recipeService:

```

import "reflect-metadata";
import express, { Request, Response } from "express";
import { AppDataSource } from "../app-data-source";
import { Recipe } from "../entities/Recipe";

AppDataSource.initialize()
  .then(() => {
    console.log("Data Source has been initialized!");
  })
  .catch((err) => {
    console.error("Error during Data Source initialization:",
err);
  });

const app = express();
app.use(express.json());

app.get("/recipes", async (req: Request, res: Response):
Promise<void> => {
  const recipes = await
AppDataSource.getRepository(Recipe).find();
  res.json(recipes);
});

app.get('/recipes/:id', async (req: Request, res: Response):
Promise<void> => {
  const recipe: Recipe | null = await
AppDataSource.getRepository(Recipe)
    .findOneBy({ recipe_id: Number(req.params.id) });

  if (!recipe) {
    res.status(404).json({ message: 'Recipe not found' });
    return;
  }
}

```

```

    res.json(recipe);
  });

app.post("/recipes", async (req: Request, res: Response):
Promise<void> => {
  const repo = AppDataSource.getRepository(Recipe);
  const newRecipe = repo.create(req.body);
  const result = await repo.save(newRecipe);
  res.status(201).json(result);
});

app.put("/recipes/:id", async (req: Request, res: Response):
Promise<void> => {
  const repo = AppDataSource.getRepository(Recipe);
  const recipe: Recipe | null = await repo.findOneBy({ recipe_id:
Number(req.params.id) });

  if (!recipe) {
    res.status(404).json({ message: "Recipe not found" });
    return;
  }

  repo.merge(recipe, req.body);
  const result = await repo.save(recipe);
  res.json(result);
});

app.delete("/recipes/:id", async (req: Request, res: Response):
Promise<void> => {
  const result = await
AppDataSource.getRepository(Recipe).delete({
    recipe_id: Number(req.params.id),
  });
  res.json(result);
});

app.listen(3000, () => {
  console.log("Server is running on http://localhost:3000");
});

```

3. Реализовать API-эндпоинт для получения пользователя по id/email.

Для User прописали аналогичные CRUDы и реализовали API-эндпоинты для получения пользователя по id/email:

```

app.get('/users/:id', async (req: Request, res: Response):
Promise<void> => {
  const user: User | null = await
AppDataSource.getRepository(User)
    .findOneBy({ user_id: Number(req.params.id) });

  if (!user) {

```

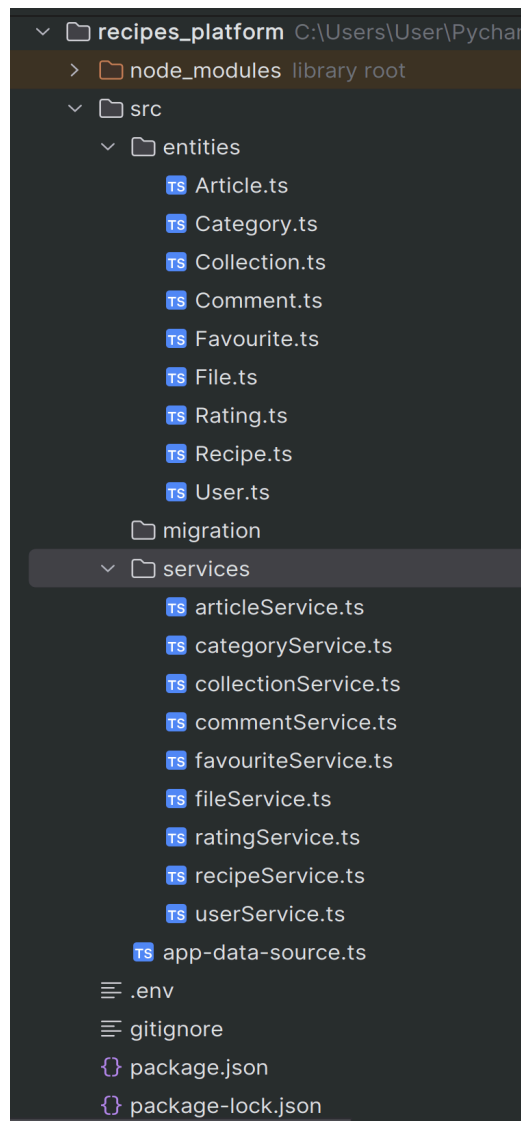
```
    res.status(404).json({ message: 'User not found' });
    return;
  }

  res.json(user);
});
app.get('/users/email/:email', async (req: Request, res:
Response): Promise<void> => {
  const user: User | null = await
AppDataSource.getRepository(User)
  .findOneBy({ email: req.params.email });

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

  res.json(user);
});
```

Наш проект на данный момент имеет такую структуру:



Вывод: В рамках задания были реализованы все модели данных в папке entities с использованием TypeORM. CRUD-методы для работы с данными были реализованы через Express и TypeScript. Отдельно реализованы API-эндпоинты для работы с пользователями, включая поиск по id и email. В ходе работы научилась работать с новыми средствами и освоила TypeORM.