

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Шурубова Прасковья

Группа: К3343

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Нужно написать свой boilerplate на express + TypeORM + typescript.

Должно быть явное разделение на:

- модели
- контроллеры
- роуты

Ход работы

Модели и сервисы были написаны в рамках Д32, поэтому мы начали прописывать контроллеры. Создали папку Controllers, на основе сервисов написали контроллеры.

Пример userController:

```
import { Request, Response } from "express";
import { AppDataSource } from "../app-data-source";
import { User } from "../entities/User";

export const userController = {
  async getAllUsers(req: Request, res: Response): Promise<void>
  {
    const users = await
AppDataSource.getRepository(User).find();
    res.json(users);
  },

  async getUserById(req: Request, res: Response): Promise<void>
  {
    const user = 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);
  },

  async getUserByEmail(req: Request, res: Response):
Promise<void> {
    const user = await
AppDataSource.getRepository(User).findOneBy({
      email: req.params.email,
    });

    if (!user) {
```

```

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

    res.json(user);
},

async createUser(req: Request, res: Response): Promise<void>
{
    const data = req.body;

    if (data.user_id && isNaN(Number(data.user_id))) {
        res.status(400).json({ message: "Invalid user_id" });
        return;
    }

    const repo = AppDataSource.getRepository(User);
    const newUser = repo.create(data);
    const result = await repo.save(newUser);
    res.status(201).json(result);
},

async updateUser(req: Request, res: Response): Promise<void>
{
    const repo = AppDataSource.getRepository(User);
    const user = await repo.findOneBy({ user_id:
Number(req.params.id) });

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

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

async deleteUser(req: Request, res: Response): Promise<void>
{
    const result = await
AppDataSource.getRepository(User).delete({
        user_id: Number(req.params.id),
    });
    res.json(result);
}
};

```

Также прописали все маршруты, решили разместить их в одном файле:

```
// Users
router.get("/users", userController.getAllUsers);
router.get("/users/:id", userController.getUserById);
router.get("/users/email/:email", userController.getUserByEmail);
router.post("/users/create", userController.createUser);
router.put("/users/update/:id", userController.updateUser);
router.delete("/users/delete/:id", userController.deleteUser);
// Recipes
router.get("/recipes", recipeController.getAllRecipes);
router.get("/recipes/:id", recipeController.getRecipeById);
router.post("/recipes/create", recipeController.createRecipe);
router.put("/recipes/update/:id", recipeController.updateRecipe);
router.delete("/recipes/delete/:id", recipeController.deleteRecipe);
```

В package.json добавили следующие параметры:

```
"start": "nodemon --watch 'src/**/*.ts' --exec './node_modules/.bin/ts-node src/server.ts',
"build": "tsc",
"typeorm": "typeorm-ts-node-commonjs --config src/ormconfig.ts",
"typeorm:generate": "typeorm-ts-node-commonjs migration:generate ./src/migration/InitialMigration -d .",
"migration:create": "typeorm-ts-node-commonjs migration:create",
"typeorm:run": "typeorm-ts-node-commonjs migration:run -d ./src/app-data-source.ts"
```

start - для запуска сервера с автоперезапуском.

build - для компиляции TypeScript в JavaScript.

typeorm - для запуска любой команды TypeORM с кастомным конфигом.

typeorm:generate - для генерации миграции по моделям.

migration:create - создается пустая миграция.

typeorm:run - для применения миграций к базе.

Также был создан ormconfig.ts:

```
import { DataSourceOptions } from "typeorm";
import * as dotenv from "dotenv";

dotenv.config();

const config: DataSourceOptions = {
  type: process.env.DB_TYPE as "postgres",
  host: process.env.DB_HOST,
  port: parseInt(process.env.DB_PORT!, 10),
  username: process.env.DB_USERNAME,
  password: process.env.DB_PASSWORD,
```

```
database: process.env.DB_NAME,  
synchronize: false,  
logging: process.env.NODE_ENV === "development",  
entities: ["src/entities/**/*.ts"],  
migrations: ["src/migration/**/*.ts"],  
subscribers: ["src/subscriber/**/*.ts"],  
};  
  
export default config;
```

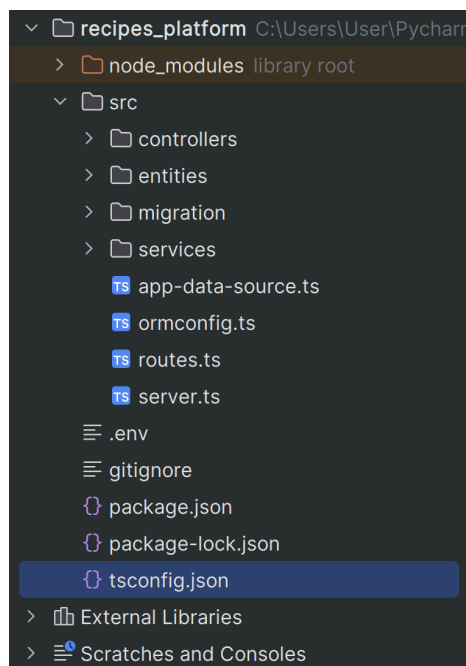
const config: DataSourceOptions = { ... }- заполняются все поля для подключения к БД.

synchronize: для синхронизации схемы БД.

logging: логирование SQL-запросов.

Информация о параметрах записана в .env.

Структура нашего проекта:



Вывод: в ходе работы была разделена бизнес-логика нашего проекта: были созданы контроллеры, маршруты, параметры для подключения к базе данных. Проект готов к дальнейшему расширению.