САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэкенд разработка

Отчет

Лабораторная работа №1

Выполнил:

Фамилия Имя
Тишалович Леонид

Группа
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача: написать свой boilerplate на express + sequelize / TypeORM + typescript.

Ход выполнения:

В ходе выполнения лабораторной работы, в папке src проект был сформирован следующим образом:

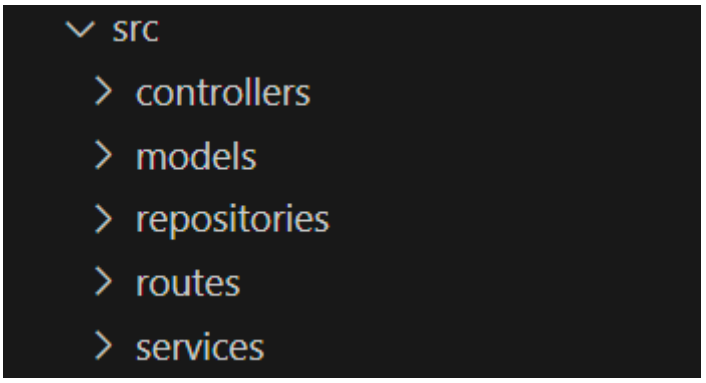

Рисунок 1 - Структура проекта

Всё начинается с index.ts в папке routes, где инициализируются все руты

```ts
import express from "express";
import userRoutes from "./userRoutes";

const router = express.Router();

router.use("/users", userRoutes);

export default router;
```

Рисунок 2 - index.ts в папке routes

В файле конкретного рута идут уже обращения к контроллеру. Например, userRoutes. Именно на этом уровне задается тип запроса GET, POST, PUT, DELETE.

```
import express from "express";
import UserController from "../controllers/UserControllers";

const router = express.Router();

router.get("/", UserController.getAllUsers);
router.get("/:id", UserController.getUserById);
router.post("/", UserController.createUser);
router.put("/:id", UserController.updateUser);
router.delete("/:id", UserController.deleteUser);

export default router;
```

Рисунок 3 - userRoutes

В контроллере уже происходит начало логики, например, файл UserController, где функции идут асинхронным способом и с помощью try/catch. В случае же удачного завершения функции, то идёт запрос на service

```
import { Request, Response } from "express";
const UserService = require("../services/UserServices");

export default {
  async getAllUsers(req: Request, res: Response) {
    try {
      const users = await UserService.getAllUsers();
      res.json(users);
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  },

  async getUserById(req: Request, res: Response) {
    try {
      const user = await UserService.getUserById(req.params.id);
      res.json(user);
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  },
```

Рисунок 4 - часть UserController

В UserService идёт вызов UserController.

```
export default class UserService {
  static async getAllUsers() {
    return UserRepository.getAllUsers();
  }

  static async getUserById(id: number) {
    return UserRepository.getUserById(id);
  }

  static async createUser(userData: any) {
    return UserRepository.createUser(userData);
  }

  static async updateUser(id: number, userData: any) {
    return UserRepository.updateUser(id, userData);
  }

  static async deleteUser(id: number) {
    return UserRepository.deleteUser(id);
  }
}
```

Рисунок 5 - UserService

В UserController происходят уже все манипуляции с моделями в бд.

```
export default class UserRepository {
  static getAllUsers() {
    return User.findAll();
  }

  static getUserById(id: number) {
    return User.findByPk(id);
  }

  static createUser(userData: any) {
    return User.create(userData);
  }

  static async updateUser(id: number, userData: any) {
    const user = await User.findByPk(id);
    if (!user) {
      throw new Error("User not found");
    }
    await user.update(userData);
    return user;
  }
```

Рисунок 6 - часть UserRepository

Вывод: В ходе данной лабораторной работы были получены навыки проектирования бэкенд-проекта, составления структуры проекта. На

защите были выявлены некоторые недостатки моего подхода (лишний слой в виде repository и обращение непосредственно к самим моделям), которые были исправлены в последующей лабораторной работе.