САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа 2

Выполнил:

Никитин Павел

Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача
Продумать свою собственную модель пользователя

Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize

Написать запрос для получения пользователя по id/email

Ход работы

Реализовал задание на typescript с esm модулями, при компиляции файлы переводятся в cjs

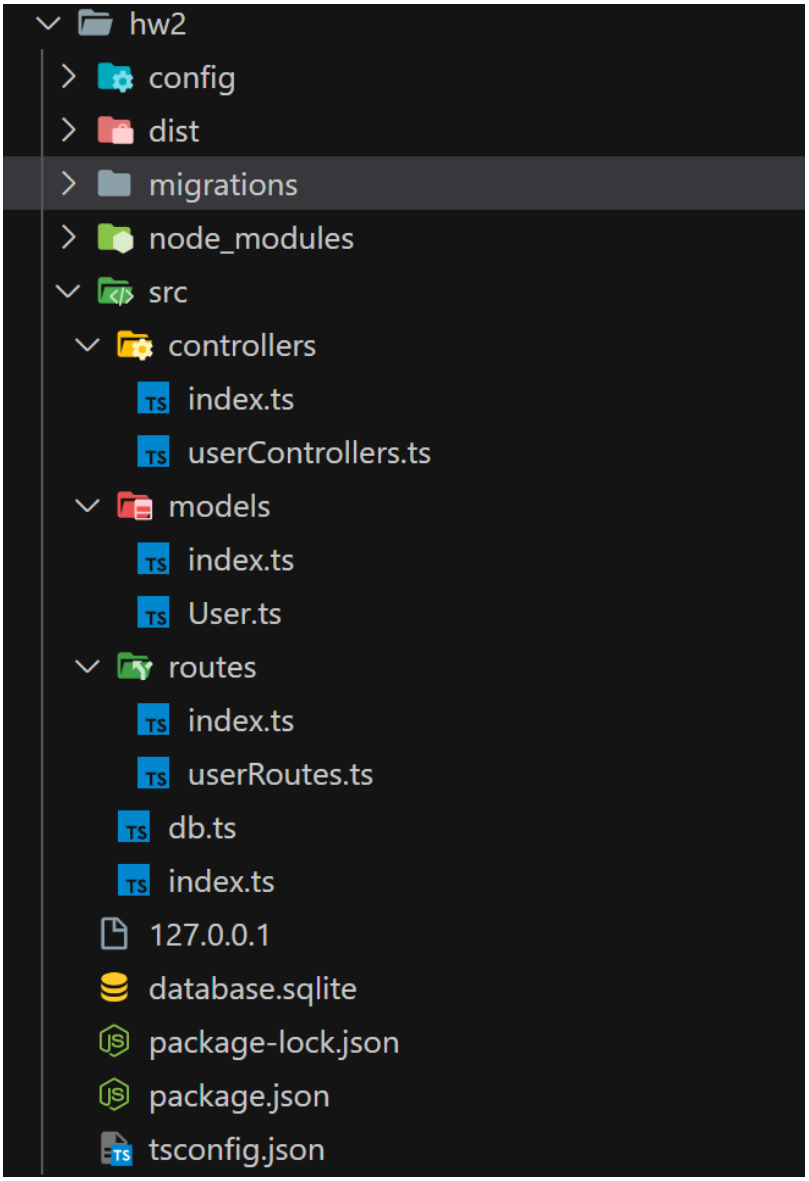

Рисунок 1 - файловая структура

В качестве базы данных использую sqlite3, прописал routes, для запросов к api

```ts
import express from "express";
import { UserController } from "../controllers";

const router = express.Router();

router.post("/", UserController.createUser);

router.get("/", UserController.getAllUsers);

router.get("/:id", UserController.getUserById);

router.get("/email/:email", UserController.getUserByEmail);

router.put("/:id", UserController.updateUser);

router.delete("/:id", UserController.deleteUser);

export { router };
```

Рисунок 2 - routes

Для каждого из файлов создается index.ts файл, откуда будет импортироваться весь функционал:

Nikitin Pavel > hw2 > src > routes > index.ts

```ts
export { router as UserRoutes } from "./userRoutes";

```

Рисунок 3 - пример index файла

Так выглядят обработчики api запросов

```ts
import { Request, Response } from "express";
import { User } from "../models/User";

type ControllerFunction = (req: Request, res: Response) => Promise<void>;

export const createUser: ControllerFunction = async (req, res) => {
  try {
    console.log(req.body);
    const user = await User.create(req.body);
    res.status(200).json(user);
  } catch (err: any) {
    res.status(400).json({ error: err.message });
  }
};

export const getAllUsers: ControllerFunction = async (req, res) => {
  try {
    const users = await User.findAll();
    res.status(200).json(users);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};

export const getUserById: ControllerFunction = async (req, res) => {
  try {
    const user = await User.findByPk(req.params.id);
    if (!user) {
      res.status(404).json({ error: "User not found" });
      return;
    }
    res.status(200).json(user);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};

export const updateUser: ControllerFunction = async (req, res) => {
  try {
    const user = await User.findByPk(req.params.id);
    if (!user) {
      res.status(404).json({ error: "User not found" });
      return;
    }
    await user.update(req.body);
    res.status(200).json(user);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};

export const deleteUser: ControllerFunction = async (req, res) => {
  try {
    const user = await User.findByPk(req.params.id);
    if (!user) {
      res.status(404).json({ error: "User not found" });
      return;
    }
    await user.destroy();
    res.status(200).end();
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};

export const getUserByEmail: ControllerFunction = async (req, res) => {
  try {
    const userEmail = req.params.email;
    const user = await User.findOne({ where: { email: userEmail } });
    if (!user) {
      res.status(404).json({ error: "User not found" });
      return;
    }
    res.status(200).json(user);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
};
```

Рисунок 4 - контроллеры

Так выглядит модель пользователя

```ts
import { DataTypes, Model, ValidationError } from 'sequelize';
import { sequelize } from '../db';

interface UserAttributes {
  id: string;
  email: string;
  username: string;
  password: string;
}

export class User extends Model<UserAttributes> implements UserAttributes {
  public id!: string;
  public email!: string;
  public username!: string;
  public password!: string;
}

User.init(
  {
    id: {
      type: DataTypes.UUID,
      defaultValue: DataTypes.UUIDV4,
      primaryKey: true
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
    username: {
      type: DataTypes.STRING,
      allowNull: false
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false
    }
  },
  {
    sequelize,
    modelName: 'User'
  }
);
```

Рисунок 5 - пример модели

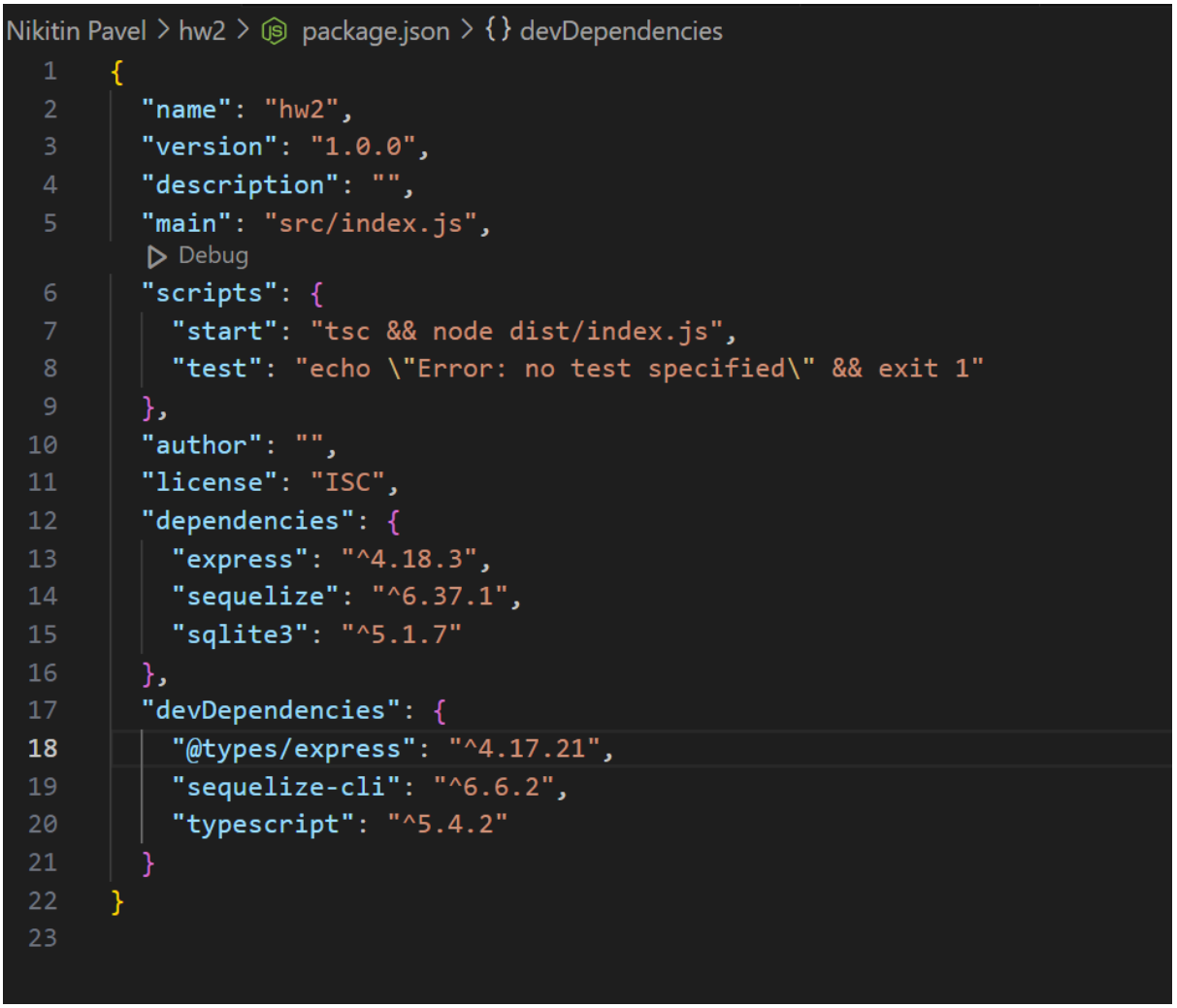

Рисунок 6 - package.json

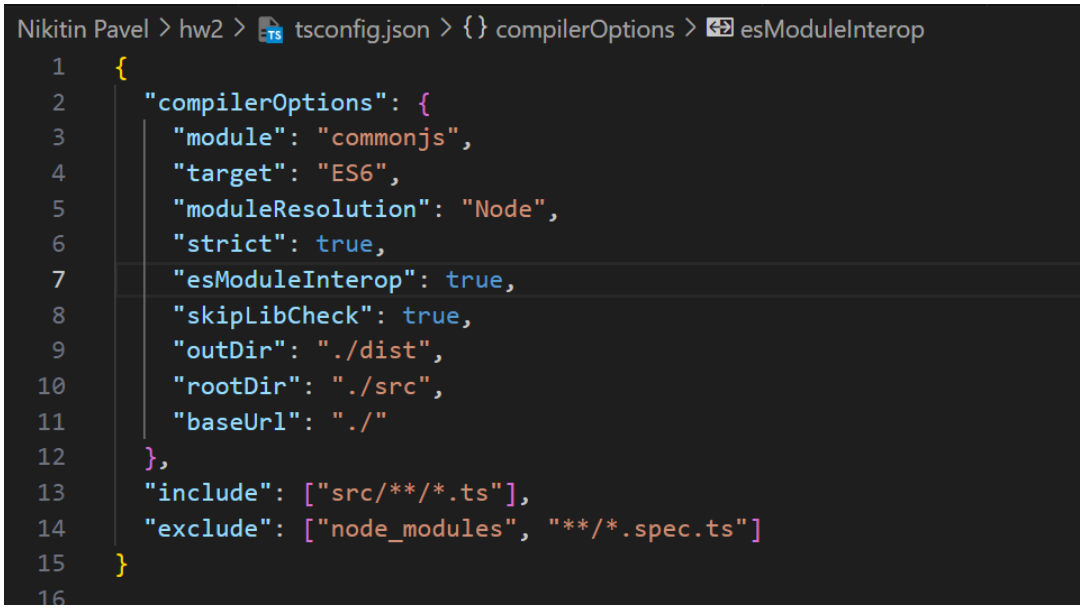

Рисунок 7 - tsconfig.json

После запуска миграций и создания бд можем проверить работоспособность системы

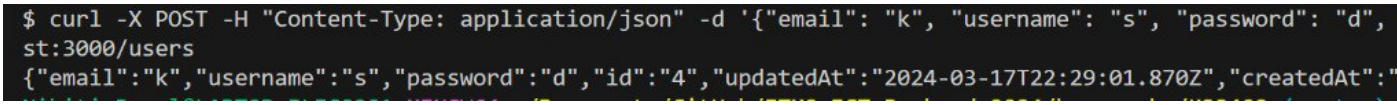

Рисунок 8 - создание пользователя

Рисунок 9 - получение пользователя по id

Рисунок 10 - получение пользователя по email

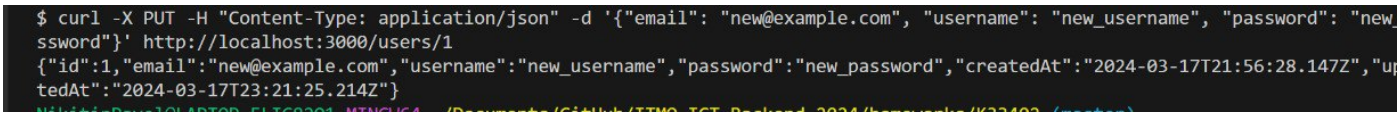

Рисунок 11 - обновление пользователя по id

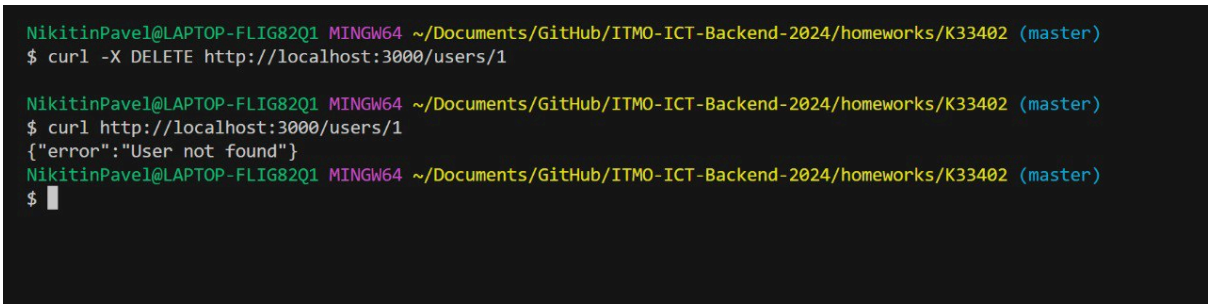

Рисунок 12 = удаление по id

Вывод

В ходе работы я научился создавать routes в express и работать с базой данных sqlite3, проводить миграции, писать запросы в терминале при помощи curl.