САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:
Жаров Александр
К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript. Должно быть явное разделение на:

- модели

- контроллеры

- роуты

- сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы

1. Инициализируем проект и устанавливаем зависимости

```
{
  "name": "lab1",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "dependencies": {
        "dotenv": "^16.4.5",
        "express": "^4.19.2",
        "sequelize-typescript": "^2.1.6",
        "sqlite3": "^5.1.7"
      },
      "devDependencies": {
        "typescript": "^5.5.0-dev.20240402"
      }
    },
```

2. Создаем tsconfig файл

```
{
  "compilerOptions": {
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "target": "ES2020",
    "sourceMap": true,
    "outDir": "dist",
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  },
  "include": ["src/**/*"]
}

```

3. Пишем index файл с инициализацией приложения express

```
import express from "express";
import unicornsRouter from "./routes/";
import sequelize from "./providers/db.js";
import dotenv from "dotenv";  6.8k (gzipped: 3k)

dotenv.config();

const app = express();

app.use(express.json());
app.use("/unicorns", unicornsRouter);

app.listen(process.env.PORT, () => {
  sequelize;
  console.log(`Listening on port ${process.env.PORT}`);
});

```

4. Создаем тестовую модель для базы данных

```
import {
  Table,
  Column,
  Model,
  Unique,
  PrimaryKey,
  AutoIncrement,
} from "sequelize-typescript";  597 (gzipped: 374)

@Table
export class Unicorn extends Model {
  @Unique
  @PrimaryKey
  @AutoIncrement
  @Column
  id: number;

  @Column
  name: string;

  @Column
  age: number;
}

```

5. Подключаемся к базе данных и передаем модели

```ts
import { Sequelize, SequelizeOptions } from "sequelize-typescript";  490 (gzipped: 293)
import { settings } from "./settings";
import { Unicorn } from "../models/unicorns";

const sequelize = new Sequelize(settings);

const models = [Unicorn];

sequelize.addModels(models);

sequelize
  .sync()
  .then(() => {
    console.log("synced models");
  })
  .catch((e) => console.log(e));

async function testConnection() {
  try {
    await sequelize.authenticate();
    console.log("Connection has been established successfully.");
  } catch (error) {
    console.error("Unable to connect to the database:", error);
  }
}

testConnection();

export default sequelize;
```

6. Создаем controller для стандартных операций с бд, куда передаем функции из services

```ts
export class UnicornService {
  async getAll(): Promise<Unicorn[]> {
    try {
      return await Unicorn.findAll();
    } catch {
      throw new Error("failed to get unicorns");
    }
  }

  async create(data: any): Promise<Unicorn> {
    try {
      return (await Unicorn.create(data)).toJSON();
    } catch {
      throw new Error("failed to create unicorn");
    }
  }

  async update(id: number, newData: Partial<Unicorn>): Promise<Unicorn> {
    try {
      const unicornToUpdate = await Unicorn.findByPk(id);
      if (!unicornToUpdate) throw new Error(`Unicorn with id ${id} not found`);

      await unicornToUpdate.update(newData);
      return unicornToUpdate;
    } catch {
      throw new Error(`failed to update unicorn with id ${id}`);
    }
  }
}
```

```ts
export class UnicornController {
  service: UnicornService;

  constructor() {
    this.service = new UnicornService();
  }

  getAll = async (req: Request, res: Response) => {
    try {
      const unicorns = await this.service.getAll();
      res.send(unicorns);
    } catch {
      res.status(404).send({ error: "unicorns not found" });
    }
  };

  post = async (req: Request, res: Response) => {
    try {
      res.send(await this.service.create(req.body as Unicorn));
    } catch {
      res.status(400).send({ error: "invalid data specified" });
    }
};

  update = async (req: Request, res: Response) => {
    try {
      const id = req.params.id;
      const updatedUnicorn = await this.service.update(id, req.body as Unicorn);
      res.send(updatedUnicorn);
    } catch {
      res.status(400).send({ error: "failed to update unicorn" });
    }
  };
}
```

7. Создаем routes

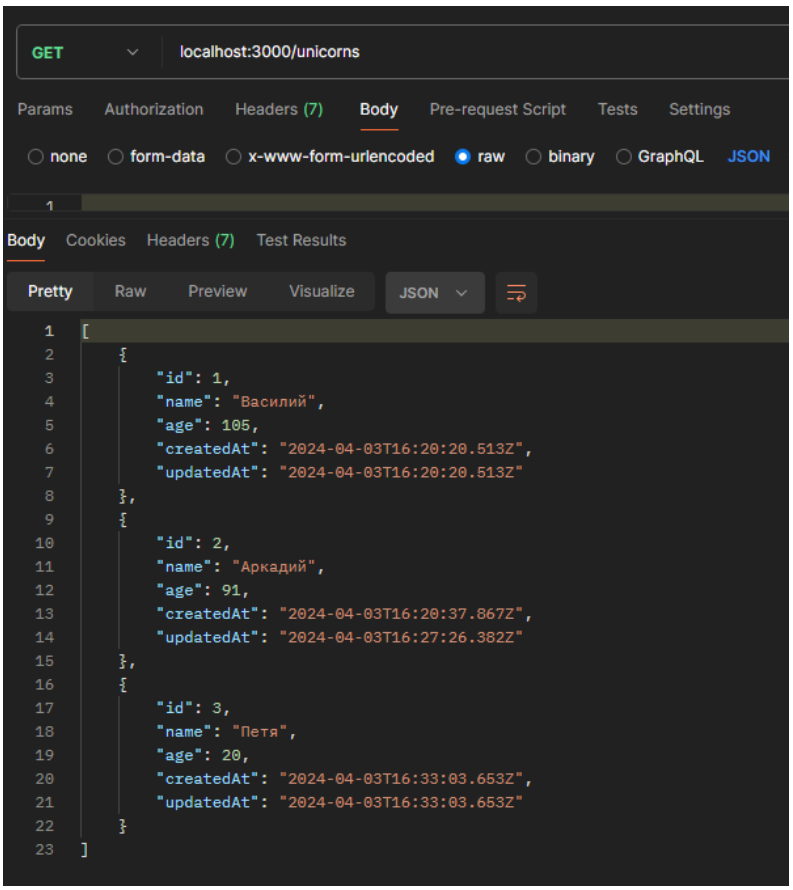
```ts
import { Router } from "express";
import { UnicornController } from "../controllers";

const router = Router();
const controller = new UnicornController();

router.get("/", controller.getAll);
router.post("/", controller.post);
router.put("/:id", controller.update);

export default router;
```

8. Проверяем работу api



Вывод

В ходе работы был написан boilerplate для express приложения