Microsserviços – Arquitetura de Aplicações

Thiago Rodrigues

Desenvolvimento

- Iniciando aplicação
  - npm init -y

- Utilizando o tipo de importação de pacotes module.
  - No arquivo package.json, adicione: "type": "module“.

- Instalando dependências
  - npm i ...
  - npm i -D ...

Desenvolvimento

- **Estrutura de pacotes**

  src
  - model
  - controller
  - routes
  - db
    - instances
  - middlewares

Desenvolvimento

- Criando arquivo .env
  - O arquivo .env deve ser criado na raiz do projeto.
  - Por padrão o nome das propriedades de um arquivo .env deve ser todo em caixa alta.

- Criando arquivo app.js
  - O arquivo app.js deve ficar dentro da pasta ./src

- Criando atalho para execução do projeto
  - No arquivo package.json adicione no objeto “scripts”
    - “start”:”node -r dotenv/config ./src/app.js”
    - “dev”:”nodemon -r dotenv/config ./src/app.js”

- Executando projeto
  - npm start
  - npm run dev

Desenvolvimento

- Adicionando propriedades no arquivo .env
  - Adicione as propriedades de porta e ambiente
    - PORT=3000
    - NODE_ENV=development

USER-MS

Serviço users-ms

- Dependências
  - axios
  - bcrypt
  - dotenv
  - express
  - jsonwebtoken
  - mysql2
  - sequelize
  - nodemon (dev)

.env

- PORT=5000

- MYSQL_DB_NAME=users_db
- MYSQL_USER=root
- MYSQL_PASSWORD=root
- MYSQL_PORT=3306

- SERVICE_NAME=users-ms
- SERVICE_REGISTRY_URL='http://localhost:4000'

app.js

```
import express from 'express';
import { connectDB } from './db/mysql_db.js';
import UserRoutes from './routes/UserRoutes.js';
import axios from 'axios';

const app = express();

app.use(express.json());

app.use(UserRoutes);

connectDB().then(()=>{
   app.listen(process.env.PORT, ()=>{
      console.log(`Server listening on port: ${process.env.PORT}`);
   });
   registerService();
});
```

```
const registerService = async () => {
   try{
      await
axios.post(`${process.env.SERVICE_REGISTRY_URL}/register`, {
         name: process.env.SERVICE_NAME,
         url: `http://localhost:${process.env.PORT}`
      });
      console.log('Service registered successfully');
   }catch(error){
      console.error(`Error registering service: ${error.message}`);
   }
}
```

mysql

```
import {Sequelize} from 'sequelize';

export const sequelize = new Sequelize(
    process.env.MYSQL_DB_NAME,
    process.env.MYSQL_USER,
    process.env.MYSQL_PASSWORD,
    {
        dialect: 'mysql',
        port: parseInt(process.env.MYSQL_PORT)
    }
);
```

mysql_db

```
import { sequelize } from "./instances/mysql.js";

export const connectDB = async ()=>{
    try{
        await sequelize.sync();
        console.log(`Connected in database
${process.env.MYSQL_DB_NAME}`);
    }catch(error){
        console.log(`Error: ${error}`);
        process.exit(1);
    }
}
```

User (model)

```
import { sequelize } from "../db/instances/mysql.js";
import { DataTypes } from "sequelize";

export const User = sequelize.define("User",{
   id:{
      type: DataTypes.BIGINT,
      primaryKey: true,
      autoIncrement: true,
      allowNull: false
   },
   username: {
      type: DataTypes.STRING(80),
      allowNull: false
   },
   password:{
      type: DataTypes.STRING(100),
      allowNull: false
   }
},{tableName: "users"});
```

UserController

- findAll

```
static async findAll(req, res){
    try{
        const users = await User.findAll({attributes:{exclude:['password']}});
        res.json({users});
    }catch(error){
        res.status(500).json({error: error.message});
    }
}
```

UserController

- findAll

```
static async findAll(req, res) {
    try {
      const users = await User.findAll({ attributes: { exclude: ['password'] } });

      const usersWithPosts = await Promise.all(users.map(async (user) => {
        try {
          const response = await axios.get(`http://localhost:5100/users/${user.id}/posts`);
          return {
            user,
            posts: response.data.posts,
          };
        } catch (error) {
          console.error(`Error fetching posts for user ${user.id}:`, error.message);
          return {
            user,
            posts: [],
          };
        }
      }));

      res.json({ users: usersWithPosts });
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  }
```

UserController

- findByUserName

```
static async findByUserName(req, res){
    try{
        const {username} = req.params;
        const user = await User.findOne({where: {username}});
        if(!user){
            return res.status(404).json({message: "User not found."});
        }
        res.json({user});
    }catch(error){
        res.status(500).json({error: error.message});
    }
}
```

UserController

- save

```
static async save(req, res){
    try{
        const {username, password} = req.body;
        const hashPassword = await bcrypt.hash(password, 10);
        const savedUser = await User.create({username, password: hashPassword});
        const result = savedUser.get({plain: true});
        delete result.password;
        res.status(201).json({user: result});
    }catch(error){
        res.status(500).json({error: error.message});
    }
}
```

UserRoutes

```
import { Router } from "express";
import {UserController} from '../controller/UserController.js';

const router = Router();

router.get("/users", UserController.findAll);
router.get("/users/:username", UserController.findByUserName);
router.post("/users", UserController.save);

export default router;
```

SERVICE-REGISTRY

Serviço service-registry

- Dependências
  - dotenv
  - express
  - nodemon (dev)

.env

- PORT=4000

app.js

```
import express from 'express';

const app = express();

app.use(express.json());

const services = {};

app.post("/register", (req, res)...
app.get("/services/:name", (req, res)...
app.get("/",(req, res)...

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

/register

```
app.post("/register", (req, res) => {
    const {name, url} = req.body;

    if(!services[name]){
        services[name] = [];
    }

    const index = services[name].indexOf(url);

    if(index === -1){
        services[name].push(url);
        console.log(`Service registered: ${name} at ${url}`);
        return res.status(200).json({message: `Service registered: ${name}`});
    }

    console.log(`Url already exists for service: ${name}`);
    res.status(200).json({message: `Url already exists for service: ${name}`});
});
```

/services/:name

```
app.get("/services/:name", (req, res) => {
    const {name} = req.params;
    const urls = services[name];
    if(urls && urls.length > 0){
        res.status(200).json({urls});
    }else{
        res.status(404).json({message: `Service not found: ${name}`});
    }
});
```

/

```
app.get("/",(req, res)=>{
    res.status(200).json({services});
});
```

GATEWAY

Serviço gateway

- Dependências
  - axios
  - dotenv
  - express
  - http-proxy-middleware
  - nodemon (dev)

.env

- PORT=3000
- SERVICE_REGISTRY_URL="http://localhost:4000"

app.js

```
import express from 'express';
import axios from 'axios';
import {createProxyMiddleware} from 'http-proxy-middleware';

const app = express();

app.use(express.json());

const serviceInstances = {};
const roudRobinIndex = {};

const getServiceUrls = async (serviceName)...
const getNextServiceUrl = (serviceName)...
app.use('/:serviceName/*', async (req, res, next)…

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

getServiceUrls

```
const getServiceUrls = async (serviceName) => {
    try{
        const response = await axios.get(`${process.env.SERVICE_REGISTRY_URL}/services/${serviceName}`);
        return response.data.urls;
    }catch(error){
        console.error(`Error fetching service URLs: ${serviceName}: ${error.message}`);
        return [];
    }
}
```

getNextServiceUrl

```
const getNextServiceUrl = (serviceName) => {
   if(!serviceInstances[serviceName] || serviceInstances[serviceName].length === 0){
      return null;
   }
   const urls = serviceInstances[serviceName];
   roudRobinIndex[serviceName] = (roudRobinIndex[serviceName] + 1) % urls.length;
   return urls[roudRobinIndex[serviceName]];
}
```

/:serviceName/*

```
app.use('/:serviceName/*', async (req, res, next)=>{
   const {serviceName} = req.params;
   if(!serviceInstances[serviceName] || serviceInstances[serviceName].length === 0){
      const urls = await getServiceUrls(serviceName);
      if(urls.length > 0){
         serviceInstances[serviceName] = urls;
         roudRobinIndex[serviceName] = 0;
      }else{
         return res.status(404).json({message: `Service not found ${serviceName}`});
      }
   }

   const serviceUrl = getNextServiceUrl(serviceName);
   console.log(`Proxying to: ${serviceUrl}`);

   if(serviceUrl){
      const endpoint = req.originalUrl.replace(`/${serviceName}/`, '');
      createProxyMiddleware({
         target: `${serviceUrl}/${endpoint}`,
         on: {
            proxyReq: (proxyReq, req, res) => {
               if(req.body){
                  const bodyData = JSON.stringify(req.body);
                  proxyReq.setHeader('Content-Length', Buffer.byteLength(bodyData));
                  proxyReq.write(bodyData);
               }
            }
         }
      })(req, res, next);
   }else{
      return res.status(404).json({message: `Service not found ${serviceName}`});
   }

});
```