BOSCH
Invented for life

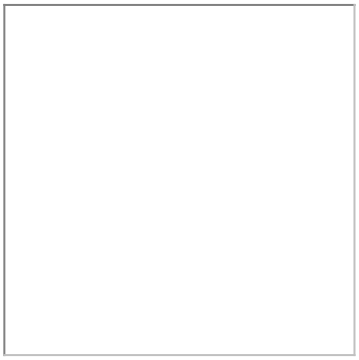
Aula 5 - API REST

Docupedia Export

Author: Siqueira Joao (CtP/ETS)
Date: 29-Jun-2022 15:30

Table of Contents

1 O que é uma API?



Uma API é um conjunto de definições e protocolos usado no desenvolvimento e na integração de aplicações. Às vezes, as APIs são descritas como um contrato entre um provedor e um usuário de informações, estabelecendo o conteúdo exigido pelo consumidor (a chamada) e o conteúdo exigido pelo produtor (a resposta). Por exemplo, o design da API de um serviço meteorológico pode especificar que o usuário forneça um CEP e o produtor responda em duas partes, a primeira contendo a temperatura mais elevada e a segunda com a temperatura mais baixa.

Em outras palavras, ao interagir com um computador ou sistema para recuperar informações ou executar uma função, a API ajudará a comunicar o que você quer ao sistema para que ele entenda e realize o que foi solicitado.

Pense nas APIs como um mediador entre os usuários ou clientes e os recursos ou serviços web que eles querem obter. As APIs também servem para que organizações compartilhem recursos e informações e, ao mesmo tempo, mantenham a segurança, o controle e a obrigatoriedade de autenticação, pois permitem determinar quem tem acesso e o que pode ser acessado.

Outra vantagem de usar APIs é que não é necessário saber todos os detalhes sobre o armazenamento em cache, como os recursos são recuperados ou qual é a origem deles.

2 O que é uma API REST?

API REST, também chamada de API RESTful, é uma interface de programação de aplicações (API ou API web) que está em conformidade com as restrições do estilo de arquitetura REST, permitindo a interação com serviços web RESTful. REST é a sigla em inglês para "Representational State Transfer", que em português significa tansferência de estado representacional. Essa arquitetura foi criada pelo cientista da computação Roy Fielding.

3 REST

REST não é um protocolo ou padrão, mas sim um conjunto de restrições de arquitetura. Os desenvolvedores de API podem implementar a arquitetura REST de maneiras variadas.

Quando um cliente faz uma solicitação usando uma API RESTful, essa API transfere uma representação do estado do recurso ao solicitante ou endpoint. Essa informação (ou representação) é entregue via HTTP utilizando um dos vários formatos possíveis: Javascript Object Notation (JSON), HTML, XLT, Python, PHP ou texto sem formatação. O formato JSON é a linguagem de programação mais usada porque, apesar de seu nome, é independente de qualquer outra linguagem e pode ser lido por máquinas e humanos.

Lembre-se também de que cabeçalhos e parâmetros são importantes nos métodos HTTP de uma solicitação HTTP de API RESTful porque contêm informações relevantes sobre o identificador, bem como metadados, autorização, Uniform Resource Identifier (URI), cache, cookies e outras informações da solicitação. Há os cabeçalhos da solicitação e os cabeçalhos da resposta, cada um contendo as informações de suas respectivas conexões HTTP e códigos de status.

Para que uma API seja considerada do tipo RESTful, ela precisa está em conformidade com os seguintes critérios:

- Ter uma arquitetura cliente/servidor formada por clientes, servidores e recursos, com solicitações gerenciadas por HTTP.
- Estabelecer uma comunicação stateless entre cliente e servidor. Isso significa que nenhuma informação do cliente é armazenada entre solicitações GET e toda as solicitações são separadas e desconectadas.
- Armazenar dados em cache para otimizar as interações entre cliente e servidor.
- Ter uma interface uniforme entre os componentes para que as informações sejam transferidas em um formato padronizado. Para tanto, é necessário que:
    - os recursos solicitados sejam identificáveis e estejam separados das representações enviadas ao cliente;
    - os recursos possam ser manipulados pelo cliente por meio da representação recebida com informações suficientes para tais ações;
    - as mensagens autodescritivas retornadas ao cliente contenham informações suficientes para descrever como processá-las;
    - hipertexto e hipermídia estão disponíveis. Isso significa que após acessar um recurso, o cliente pode usar hiperlinks para encontrar as demais ações disponíveis para ele no momento.
- Ter um sistema em camadas que organiza os tipos de servidores (responsáveis pela segurança, pelo carregamento de carga e assim por diante) envolvidos na recuperação das informações solicitadas em hierarquias que o cliente não pode ver.
- Possibilitar código sob demanda (opcional): a capacidade de enviar um código executável do servidor para o cliente quando solicitado para ampliar a funcionalidade disponível ao cliente.

Embora uma API REST precise estar em conformidade com os critérios acima, ela é considerada mais fácil de usar do que um protocolo prescrito, como o Protocolo Simples de Acesso a Objetos (SOAP). Esse tipo de protocolo tem requisitos específicos, como o sistema de mensageria XML, além de precisar cumprir com exigências de segurança incorporada e transações, o que o torna mais lento e pesado.

Em comparação, a arquitetura REST é composta de um conjunto de diretrizes que podem ser implementadas conforme necessário. Isso faz com que as APIs REST sejam mais rápidas, leves e escaláveis, o que é ideal para a Internet das Coisas (IoT) e o desenvolvimento de aplicativos mobile.

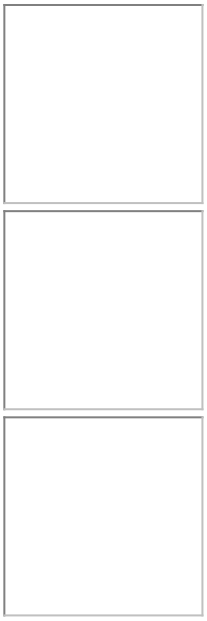
4 Como criar uma API?

Nessa aula vamos desenvolver uma API baseada no framework Express para tratar as requisições e enviar respostas aos endpoints, como banco de dados iremos utilizar o Microsoft SQL Server com o ORM Sequelize. Além disso vamos usar o framework Express e por último a extensão Thunder Client para testarmos nossa API.

Para começar a criação da nossa API será necessário ter instalado em nossa máquina:

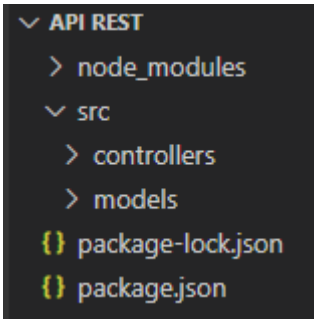
- Node.js
- Thunder Client
- Microsoft SQL Server

Antes de começar é necessário deixar claro que vamos criar uma API nos padrões RESTful com um CRUD (*create, read, update, delete*), nossos endpoints serão criados com verbos HTTP com base em sua ação e as respostas serão baseadas em JSON, contendo também o status correto, Após todo o processo aplicaremos algumas validações.



Para começar a criar nossa API vamos inicializar o npm com o comando *npm init -y* e em seguida instalar todos os pacotes necessários de uma só vez com o comando *npm install express mssql sequelize*, após a instalação eles deverão aparecer no arquivo package.json.

Antes de criarmos nossos arquivos, criaremos as pastas para dividir as camadas model e controller da seguinte maneira:



Dentro da pasta src será criado um arquivo db.js, pelo qual iremos começar.

Nele será realizada a configuração do nosso banco de dados da seguinte maneira:

db.js

```
1  const sequelize = require('sequelize');
2
3  //configurações da base de dados
4  const database = new sequelize('master',<nome_do_usuário>, <senha>,
5      {
6      dialect: 'mssql', host:<nome_do_host>, port: 1433
7      });
8
9  //sincroniza com uma tabela já existente
10 database.sync();
```

```
11
12  //
13  module.exports = database;
```

O próximo passo será criar na model um arquivo produto.js, que irá conter uma classe e seus atributos.

produto.js

```
const sequelize = require('sequelize');
const database = require('../db');
const schema = '';

//Model é uma abstração que representa uma tabela
class Produto extends sequelize.Model{}


Produto.init(
    {
     //Campos da nossa tabela
        Codigo :
        {
            type : sequelize.INTEGER,
            autoIncrement : true,
            allowNull : false,
            primaryKey : true
        },
        Descricao :
        {
            type : sequelize.STRING,
            allowNull : true
        },
        DataCriacao :
        {
            type : sequelize.DATE,
            allowNull : false
        },
        DataAtualizacao :
        {
            type : sequelize.DATE,
            allowNull : true
        }
    },

//Definição da tabela tbProduto
    {
        sequelize : database, modelName : 'tbProduto', schema
    }
)

module.exports = Produto;
```

Já possuindo a model pronta, vamos partir para a criação da controller, criando o arquivo ProdutoController.js, onde estarão nossos métodos de Create, Update, Delete, List e etc...

Começaremos pelo Create e Delete.

ProdutoController.js

```
const ModelProduto = require('../models/produto');

module.exports =
{

    async Create(req, res)
    {

        try {
            const prod = await ModelProduto.create(
                {
                    Descricao : req.body.Descricao,
                    DataCriacao : req.body.DataCriacao,
                    DataAtualizacao: null
                }
            );

            return res.json(prod);
        }

        catch (erro) {
            return console.error("Error na Create :", erro);
        }
    },

    async Delete(req, res)
    {
        try {
            const prod = await ModelProduto.findByPk(req.body.Codigo);
            await prod.destroy();
        }

        catch (erro) {
            return console.error("Error na Update :", erro);
        }
    }
}
```

Ainda faltam as rotas para acessar os métodos criados, então iremos criar mais um arquivo na pasta src chamado router.js

Ver: https://expressjs.com/pt-br/guide/routing.html

router.js

```
const express = require('express');
const controllerProduto = require('./controllers/ProdutoController');

//Criando uma instância router da classe express
const routes = express.Router();

//Definindo rotas para os métodos dentro da controller
```

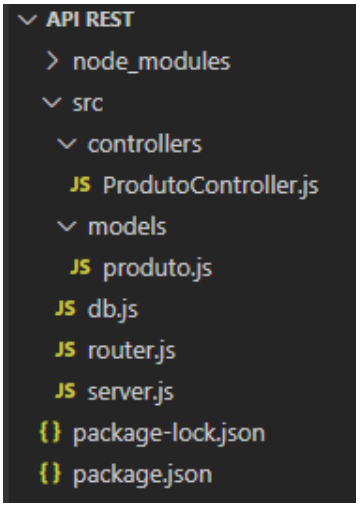
```
routes.get('/List', controllerProduto.List);

routes.post('/Create', controllerProduto.Create);

routes.post('/Update', controllerProduto.Update);

routes.get('/GetOne', controllerProduto.GetOne);

routes.post('/Delete', controllerProduto.Delete);

module.exports = routes;
```

Por último vamos criar nosso servidor para poder executar e testar nossa API.

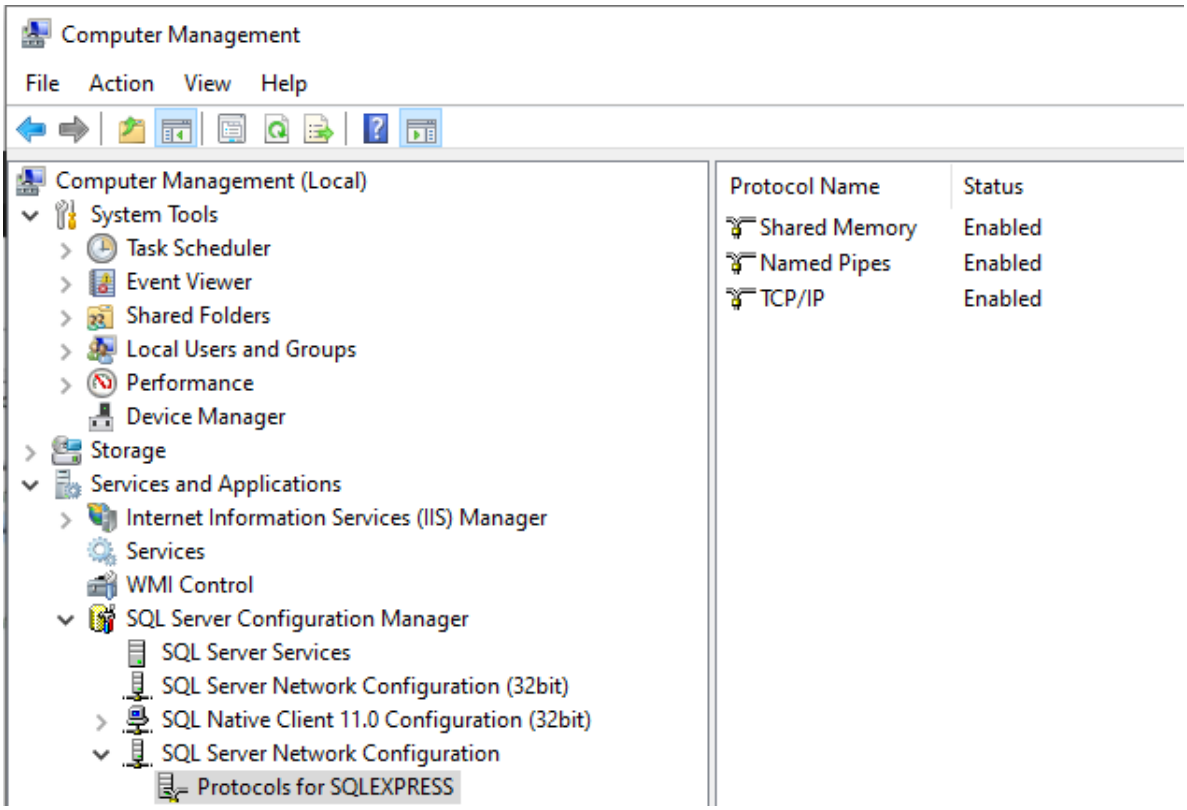
router.js

```
//Criando servidor
const express = require('express');
const api = express();

const routes = require('./router');
const cors = require('cors');

api.use(cors());
api.use(express.json());
api.use(routes);


api.listen(4200);
```

No final, nosso projeto deve estar organizados da seguinte maneira:

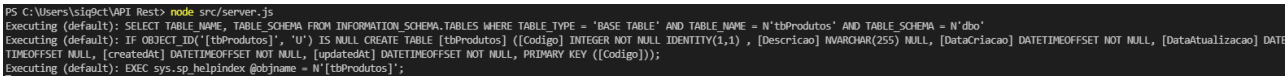


Antes de rodarmos o projeto devemos habilitar a configuração de rede TCP/IP do SQL Server instalado em nossa máquina. Isso pode ser acessado diretamente pelo SQL Server Configuration Manager ou pelo Computer Management (será necessário executar como administrador), como na imagem abaixo:

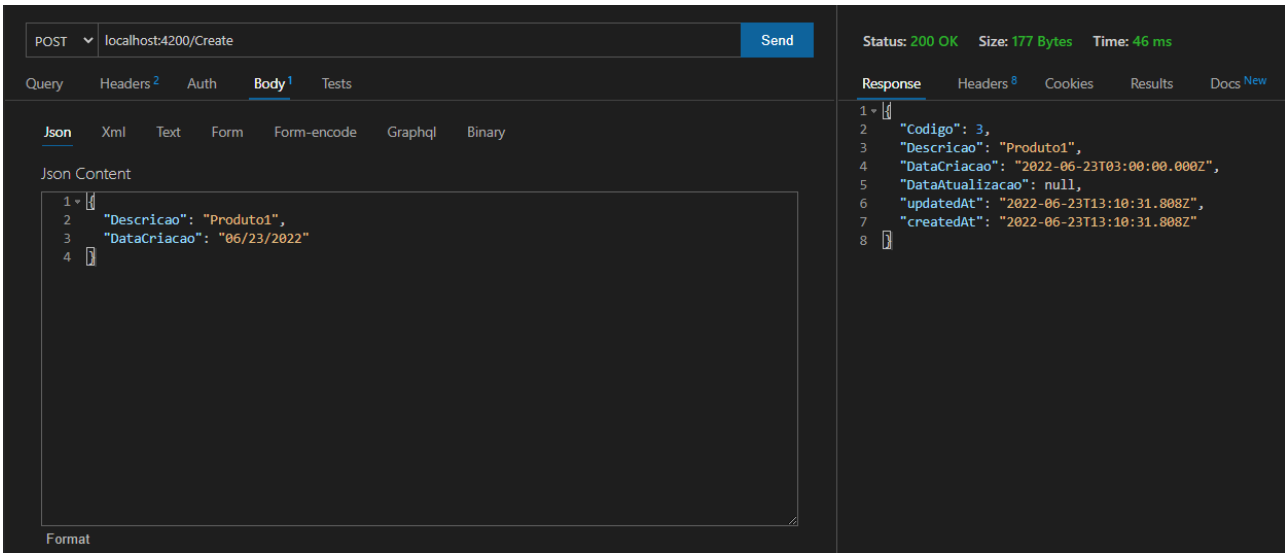


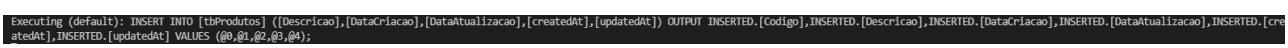
Após habilitar a configuração de rede TCP/IP será necessário reiniciar a máquina para que o servidor do SQL Server reinicie.

Para rodar nossa aplicação atráves do node iremos inserir o comando *node src/server.js* no terminal



Agora, já serão possíveis simular requisições usando o Thunder Client da seguinte maneira:





5 Exercício

Agora que você aprendeu a criar rotas e métodos, inclua as operações de Update (atualizar um único dado), List (lista todos os dados) e Get One (lista um único dado específico)