

Node.js e SQL Server

Primeiramente vamos criar uma aplicação Node.js para testarmos o acesso ao banco de dados SQL Server. Para criar uma aplicação no Node, vamos criar uma pasta para todos os arquivos da aplicação (tanto os criados pelo node como os criados pelo desenvolvedor).

Por linha de comando (CMD) vamos criar a pasta e criar o projeto Node.

```
mkdir nodeSQL  
  
cd nodeSQL  
  
npm init
```

O comando **npm init** vai fazer várias perguntas para criar um arquivo de configuração para o projeto. Para esse primeiro exemplo, não vamos inserir nenhuma informação adicional a não ser o nome do projeto. Todas as informações do projeto serão gravadas no arquivo **package.json** na pasta do projeto.

Para acessarmos o banco de dados precisamos instalar o módulo para a comunicação com o banco que vamos usar. Como vamos usar o SQL Server da Microsoft, devemos instalar o módulo **mssql**. Aproveitaremos também para instalar também os módulos **express** e **body-parser** que utilizaremos nesse exemplo.

```
nodesqlserver> npm install -S mssql express body-parser
```

Na sequência, vamos criar um arquivo **index.js** na pasta do projeto onde vamos criar o nosso servidor da API para tratar as requisições que chegarão em breve. Vamos começar apenas definindo as constantes locais que serão usadas mais pra frente:

```
const express = require('express');  
  
const app = express();  
  
const bodyParser = require('body-parser');  
  
const porta = 3000; //porta padrão  
  
const sql = require('mssql');  
  
const conexaoStr = "Server=regulus;Database=patricia;User Id=patricia;Password=senha;"
```

Vamos adicionar as seguintes linhas logo abaixo do bloco anterior, para conectar no banco de dados assim que a aplicação iniciar:

```
//conexao com BD  
sql.connect(conexaoStr)  
  
  .then(conexao => global.conexao = conexao)  
  
  .catch(erro => console.log(erro));
```

Agora vamos configurar nossa aplicação (app) Express para usar o body parser que carregamos da biblioteca body-parser, permitindo que recebamos mais tarde POSTs nos formatos URLEncoded e JSON:

```
// configurando o body parser para pegar POSTS mais tarde

app.use(bodyParser.urlencoded({ extended: true}));

app.use(bodyParser.json());
```

Na sequência, vamos criar um roteador e dentro dele definir uma regra inicial que apenas exibe uma mensagem de sucesso quando o usuário requisitar um GET na raiz da API (/) para ver se está funcionando.

```
//definindo as rotas

const rota = express.Router();

rota.get('/', (requisicao, resposta) => resposta.json({ mensagem: 'Funcionando!'}));
```

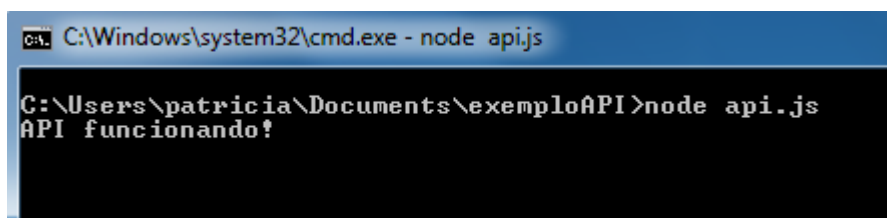
Por enquanto definimos apenas uma rota, ou seja, depois na URL que identifica nosso servidor será tratada apenas a barra "/" que significa dizer que é a raiz do endereço. Por fim, adicionamos as linhas abaixo no final do arquivo que dão o start no servidor da API:

```
//inicia servidor

app.listen(porta);

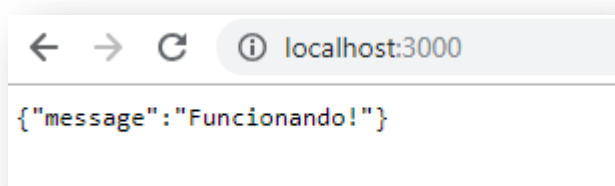
console.log('API Funcionando!');
```

Vamos testar a API executando via console o index.js com o comando **node index.js**. Você deve ver a mensagem de 'API funcionando!' no console, e se acessar no navegador localhost:3000 deve ver o JSON default que deixamos na rota raiz!



```
C:\Windows\system32\cmd.exe - node api.js

C:\Users\patricia\Documents\exemploAPI>node api.js
API funcionando!
```



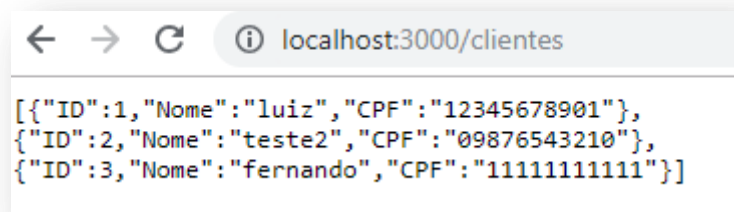
Listando Dados

Vamos adicionar uma rota `/clientes` que listará todos os clientes do banco de dados. Para fazer isso, primeiro vamos criar uma função que executará consultas SQL no banco usando o pool de conexões global (`conexao`), como abaixo:

```
function execSQL(sql, resposta) {
    global.conexao.request()
        .query(sql)
        .then(resultado => resposta.json(resultado.recordset))
        .catch(erro => resposta.json(erro));
}

rota.get('/clientes', (requisicao, resposta) => {
    execSQL('SELECT * FROM Clientes', res);
})
```

Ao executarmos novamente nosso projeto e acessarmos a URL `localhost:3000/clientes`, veremos todos os clientes cadastrados no banco de dados:



Criando a pesquisa de um cliente

Se o usuário quiser ver apenas um cliente, ele deverá passar o ID do mesmo na URL, logo após `/clientes`. Para fazer isso, vamos modificar nossa rota criada no passo anterior, `/clientes`, para aceitar um parâmetro opcional ID. Além disso, dentro do processamento da rota, se vier o ID, devemos fazer uma consulta diferente da anterior, como mostra o código abaixo, com os ajustes na mesma rota do passo anterior:

```
//o simbolo ? indica que id na rota abaixo é opcional
rota.get('/clientes/:id?', (requisicao, resposta) => {
    let filtro = '';
    if (requisicao.params.id)
        filtro = ' WHERE ID=' + parseInt(requisicao.params.id);
    execSQL('SELECT * from Clientes' + filtro, resposta);
})
```

