

# Node e CRUD

Usabilidade, desenvolvimento web, mobile e jogos

Prof. Me. Gustavo Torres Custódio  
gustavo.custodio@ulife.com.br

# Conteúdo

**Consumindo uma API**

**CRUD - UPDATE**

**CRUD - DELETE**



Node e CRUD

## Consumindo uma API

## Consumindo uma API

- Vamos criar uma API utilizando NodeJS que retorna uma lista de funcionários.
  - Essa lista de funcionários é devolvida em **formato JSON**.
- Vamos criar o arquivo `backend_funcionarios.js`.

## Consumindo uma API

```
var http = require('http');
var sqlite3 = require('sqlite3');
var bodyParser = require('body-parser');
var path = require('path');
var express = require('express');
var app = express();

var server = http.createServer(app);
var db = new sqlite3.Database('./database/Empresa.db');

app.use(bodyParser.urlencoded({extended: false}));
// Isso é utilizado para que o servidor em node saiba
// a localização dos arquivos estáticos (css, js, imagens etc).
// Esses arquivos ficarão dentro de uma pasta chamada public.
app.use(express.static(path.join(__dirname, 'public')));
```

## Consumindo uma API

```
app.get('/listartodos', function(req,res){
  db.serialize(()=>{
    //db.all() inclui todos os funcionarios
    db.all('SELECT id, nome FROM Funcionario', function(err, rows){
      if(err){
        res.send("Erro ao encontrar funcionarios.");
        return console.error(err.message);
      }
      // Adiciona todos os funcionários em um array list
      let nomesFuncionarios = [];
      for (const row of rows) {
        nomesFuncionarios.push(row.nome);
      }
      res.send(nomesFuncionarios);
      console.log("Funcionarios encontrados");
    });
  });
});
```

## Consumindo uma API

```
app.get('/', function(req, res) {  
    res.sendFile(path.join(__dirname, 'listagem.html'));  
});  
  
server.listen(3333, function() {  
    console.log("Servidor ouvindo na porta 3333.");  
})
```

## Consumindo uma API

- O *backend* retorna apenas o texto necessário.
  - A apresentação desse texto é
  - Utilizar essa abordagem permite que criemos múltiplos *frontends* diferentes para o mesmo *backend*.
    - Como, por exemplo uma aplicação *mobile*.



## Consumindo uma API

- No *frontend*, a função **fetch** é responsável por “puxar” os dados do *backend*.
  - Esses dados devem obrigatoriamente ser devolvidos em formato JSON.

## Consumindo uma API

- Crie um arquivo chamado `busca.js` (coloque ele em uma pasta chamada `public`).
  - Esses dados devem obrigatoriamente ser devolvidos em formato JSON.

## Consumindo uma API

```
function preencherDivFuncionarios(funcionarios) {  
  // Coloca cada funcionário em um parágrafo  
  funcionariosParagrafos = "<p>" + funcionarios.join("</p> <p>") + "</p>";  
  divResultados = document.getElementById("divResultados");  
  divResultados.innerHTML = funcionariosParagrafos;  
}
```

```
function listarFuncionarios() {  
  const url = '/listartodos';  
  // A função fetch recupera alguma informação do back-end  
  // que deve ser fornecida em formato json  
  fetch(url  
  ).then( function(response) {  
    return response.json();  
  }).then( function(funcionarios) {  
    preencherDivFuncionarios(funcionarios);  
  });  
}
```

## Consumindo uma API

- Finalmente, crie um arquivo listagem.html:

```
<!DOCTYPE html>
<html>
  <head> <meta charset="utf-8">
    <title>Busca</title>
    <script src="js/busca.js"></script>
  </head>
  <body>
    <div id="divResultados">
      </div> <br>
      <button type="button" onclick="listarFuncionarios();">
        Listar Funcionários</button>
    </body>
</html>
```

## Consumindo uma API

- Rode o backend com o comando:
  - `node backend.js` e abra a página `listagem.html`.
  - Clique no botão de listas funcionários.
- O que aconteceu:
  - O *frontend* enviou uma requisição para o *backend*.
  - O *backend* respondeu com um *array*.
  - Após a resposta ser devolvida pelo servidor, o conteúdo do *array* preencheu uma *div*.



Node e CRUD

## CRUD - UPDATE

## Continuação CRUD

- Vamos continuar desenvolvendo nosso *backend* com operações CRUD.
  - Baixe os arquivos com o código e com o banco de dados no site da disciplina.

# Update

- A operação de *update* (atualização) modifica uma ou mais linhas de uma tabela.
  - O registro já precisa existir na tabela.
  - A **cláusula WHERE** determina quais linha serão afetadas.



# Update

- Forma geral:

```
UPDATE <Tabela>  
SET <atributo1>='<valor1>'  
    <atributo2>='<valor2>'  
    ...  
    <atributoN>='<valorN>'  
WHERE <condição>;
```

## Update

- Vamos criar um formulário com 4 campos:
  - id;
  - nome;
  - idade;
  - endereço.
- O id será utilizado para **determinar qual linha será atualizada.**
- Os demais campos serão atualizados.

## Update

- Adicione um formulário novo no arquivo formulario.html.

```
<form action="/update" method="POST">  
  <input type="number" name="id" placeholder="id">  
    <br> <br>  
  <input type="text" name="nome" placeholder="Nome">  
    <br> <br>  
  <input type="number" name="idade" placeholder="Idade">  
    <br> <br>  
  <input type="text" name="endereco" placeholder="Endereço">  
    <br> <br>  
  <input type="submit" value="Atualizar">  
</form>
```

# Update

- Crie um *endpoint* que processa os dados enviados pelo formulário.

```
app.post('/update', function(req,res){
  db.serialize(()=>{
    db.run(
      // Atualizar um funcionário no banco
      "UPDATE Funcionario (id, nome, idade, endereco) " +
        "SET id=?, nome=?, idade=?, endereco=? " +
        "WHERE id=?",

      // Aqui são os valores dos ?, que correspondem ao formulário
      [req.body.id, req.body.nome, req.body.idade, req.body.endereco,
        req.body.id],
    )
  })
})
```

# UPDATE

```
// Depois de rodar a o sql, essa função é executada
function(err) {
  // Se houver algum problema no update, mostre um erro
  if (err) {
    return console.log(err.message);
  }
  console.log("Registro atualizado.");
  res.send( "Funcionário " + req.body.id +
    " atualizado com sucesso.");
}
);
});
});
```

## UPDATE

- No final, será buscado um funcionário por id e suas informações (exceto o id) serão atualizadas.
  - De acordo com os valores dos campos no formulário.
- Listamos os funcionários novamente para ver o resultado.



Node e CRUD

## CRUD - DELETE

# DELETE

- A operação de *delete* (remover) remove uma ou mais linhas de uma tabela.
  - Assim como o *update*, a **cláusula WHERE** determina quais linha serão afetadas.
    - Sem ela, todas as linhas da tabela são removidas.



# DELETE

- Forma geral:

```
DELETE  
FROM <Tabela>  
WHERE <condição>;
```

# DELETE

- Adicionamos mais um formulário dentro de formulario.html:
  - Ele possui apenas o campo id.
  - O funcionário será buscado pelo id e sua linha na tabela será removida.

```
<h2>Remover</h2>  
<form action="/delete" method="POST">  
  <input type="number" name="id" id="id" placeholder="id">  
  <input type="submit" value="Remover Funcionário">  
</form>
```

# DELETE

- E um *endpoint* que recebe o id.
  - e aplica a operação SQL de remover o funcionário com o id correspondente.

```
app.post('/delete', function(req,res){  
  db.serialize(()=>{  
    db.run(  
      // Remover um funcionário com id específico  
      "DELETE From Funcionario WHERE id=?",  
      [req.body.id],  
    )  
  })  
})
```

# DELETE

```
// Depois de rodar a o sql, essa função é executada
function(err) {
  // Se houver algum problema no delete, mostre um erro
  if (err) {
    return console.log(err.message);
  }
  console.log("Registro removido.");
  res.send( "Funcionário " + req.body.id +
    " removido com sucesso.");
}
);
});
});
```

## DELETE

- No final, será buscado um funcionário por id e ele será removido da tabela.
- Listamos os funcionários novamente para ver o resultado.

## Exercício

- Adicione uma tela inicial com 4 *links*.
- Cada link direciona a página para um dos formulários:
  - Busca;
  - Inserir;
  - Atualizar;
  - Deletar.

## Referências

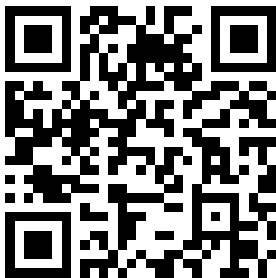


Paul, S. (2020).

Read html form data using get and post method in node.js.

<https://medium.com/swlh/read-html-form-data-using-get-and-post-method-in-node-js-8d2c7880adbf>.

## Conteúdo



<https://gustavotcustodio.github.io/usabilidade.html>



Obrigado

[gustavo.custodio@ulife.com.br](mailto:gustavo.custodio@ulife.com.br)