

Objetivos:

- I. Servidor de exemplo
- II. Fetch API;
- III. Enviar dados com Fetch: parâmetros no Body.

I. Servidor de exemplo

Antes de começar será necessário criar um servidor Node.js com Express. Siga os passos a seguir para configurar o servidor de exemplo:

Passo 1 – Crie uma pasta de nome `server` ou qualquer outro nome no local de sua preferência do computador;

Passo 2 – Use o comando a seguir para criar o arquivo de configuração de um projeto Node.js:

```
npm init -y
```

Passo 3 – Use o comando a seguir para instalar as dependências:

```
npm i express dotenv cors
```

O pacote CORS (Cross-Origin Resource Sharing) é usado para configurar o servidor para aceitar requisições provenientes de outros domínios. Como exemplo, se o nosso servidor estiver rodando em `localhost:3010`. Ele não poderá ser acessado por um cliente que estiver rodando em `localhost:3011`.

Passo 4 – Crie o arquivo `.env` na raiz do projeto e coloque a variável:

```
PORT = 3010
```

Passo 5 – Crie a pasta `src` e o arquivo `index.js` com o seguinte código:

```
const express = require("express");
const dotenv = require("dotenv");
const cors = require("cors");

dotenv.config();

const app = express();
const PORT = process.env.PORT || 3000;

app.use(express.json());
app.use(cors()); // Configura o servidor para aceitar requisições de qualquer domínio

app.listen(PORT, function() {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});

// Exemplo 1
app.get("/exemplo1", function(req, res) {
  res.json({ message: "Boa noite!" });
});
```

```
// Exemplo 2
```

```
app.post("/exemplo2", function(req, res) {  
  const { x, y } = req.body;  
  const resultado = parseFloat(x) + parseFloat(y);  
  res.json({ resultado });  
});
```

```
// Exercício 3
```

```
app.get("/multiplicacao/:x/:y", function(req, res) {  
  const { x, y } = req.params;  
  const resultado = parseFloat(x) * parseFloat(y);  
  res.json({ resultado });  
});
```

```
// Exercício 4
```

```
app.get("/potencia", function(req, res) {  
  const base = req.query.base;  
  const expoente = req.query.expoente;  
  const resultado = parseFloat(base) ** parseFloat(expoente);  
  res.json({ resultado });  
});
```

```
// Exercício 5
```

```
app.get("/lista", function(req, res) {  
  const dias = [  
    "domingo",  
    "segunda-feira",  
    "terça-feira",  
    "quarta-feira",  
    "quinta-feira",  
    "sexta-feira",  
    "sábado",  
  ];  
  res.json({ dias });  
});
```

```
// Exercício 6
```

```
app.get("/detalhes", function(req, res) {  
  const dias = [  
    { dia: "domingo", tipo: "final de semana" },  
    { dia: "segunda-feira", tipo: "normal" },  
    { dia: "terça-feira", tipo: "normal" },  
    { dia: "quarta-feira", tipo: "normal" },  
    { dia: "quinta-feira", tipo: "normal" },  
    { dia: "sexta-feira", tipo: "normal" },  
    { dia: "sábado", tipo: "final de semana" },  
  ];  
  res.json({ dias });  
});
```

Passo 6 – Inclua na propriedade `scripts` do arquivo `package.json` a propriedade para rodarmos o projeto:

```
"scripts": {  
  "start": "node ./src"  
},
```

Passo 7 – Use o comando `npm run start` para executar o projeto no terminal do VS Code.

II. Fetch API

Uma requisição HTTP é uma comunicação entre um cliente (navegador) e um servidor (como um servidor Node.js), onde:

- O cliente solicita dados usando o protocolo HTTP (Hypertext Transfer Protocol), que é o protocolo padrão para comunicação entre cliente e servidor na web.

Principais métodos HTTP:

- GET: buscar informações;
- POST: enviar dados;
- PUT: atualizar dados;
- DELETE: remover dados.

O servidor responde com os dados solicitados ou uma confirmação, acompanhada de um código de status.

Principais códigos de status HTTP:

- 200 OK: requisição bem-sucedida;
- 201 Created: registro criado com sucesso;
- 400 Bad Request: erro na requisição;
- 404 Not Found: recurso não encontrado;
- 500 Internal Server Error: erro no servidor.

Toda requisição é composta por:

- Request: dados enviados pelo cliente.
- Response: dados retornados pelo servidor.

Toda requisição é formada pelo Request (dados enviados pelo cliente) e pelo Response (dados retornados pelo servidor).

A Fetch API é uma interface que permite fazer requisições HTTP de forma assíncrona. Mais informações podem ser encontradas em https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API.

Uma operação assíncrona é aquela que não possui tempo determinado para ser concluída. O JavaScript usa *promises* (promessas) para lidar com essas operações.

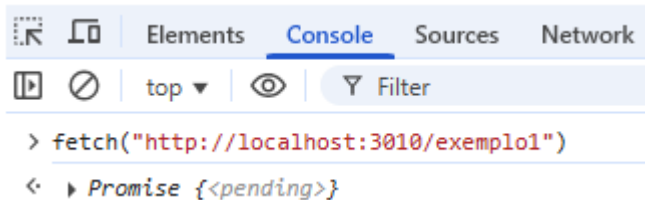
No JavaScript, um objeto *Promise* será executado somente quando a operação assíncrona for finalizada.

A Fetch API usa a função global `fetch()` com os seguintes parâmetros:

```
fetch(url, options);
```

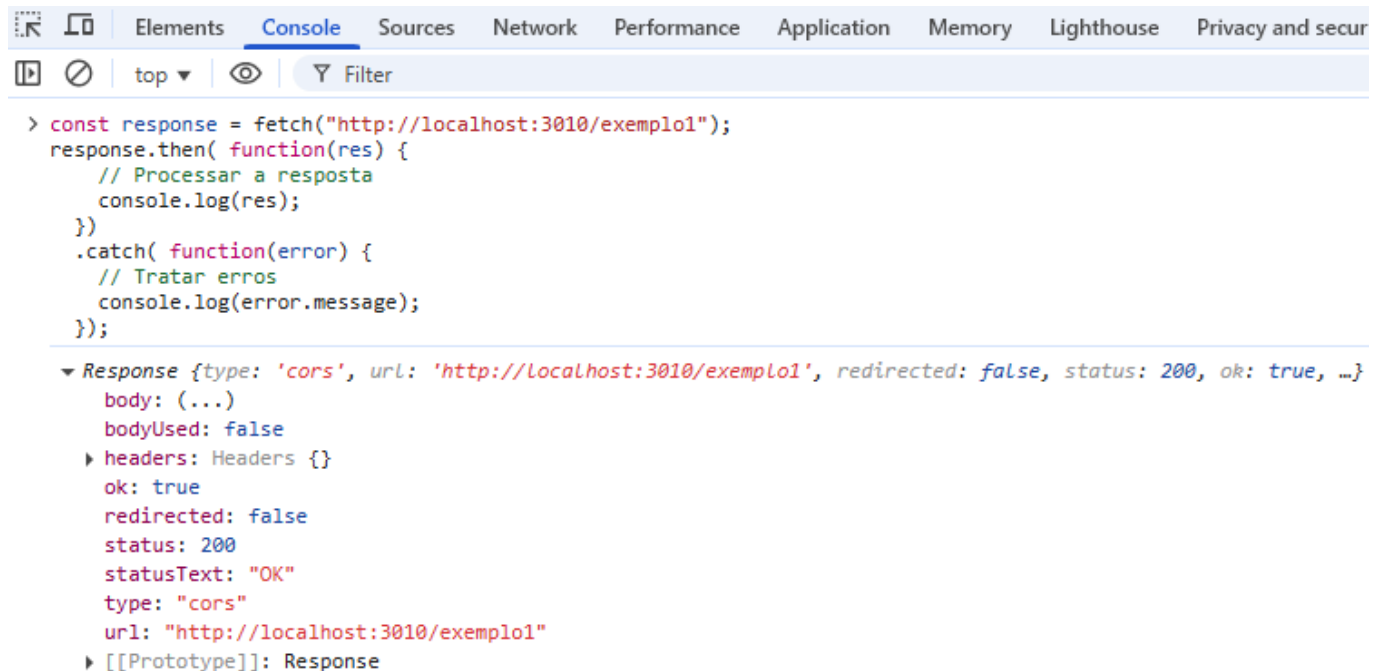
- url: endereço do servidor ou API que será acessada;
- options (opcional): configurações da requisição (método, cabeçalhos, corpo etc.).

A função `fetch` retorna uma promise, ou seja, a execução da função fica pendente até que a operação seja concluída.



Para lidar com a resposta da promise, utilizamos os métodos `.then` (para sucesso) e `.catch` (para erro):

```
const response = fetch("http://localhost:3010/exemplo1");
response.then( function(res) {
    // Processar a resposta
    console.log(res);
})
.catch( function(error) {
    // Tratar erros
    console.log(error.message);
});
```

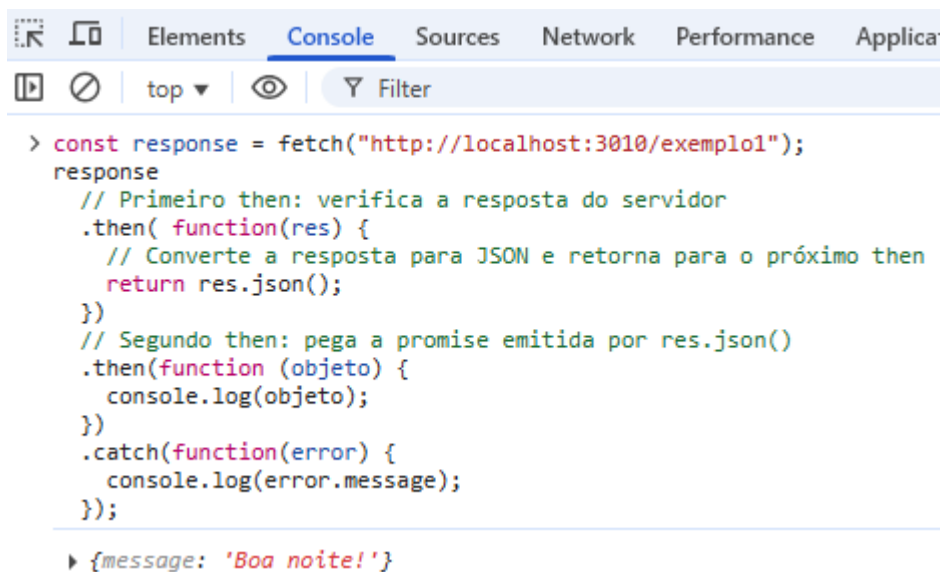


O objeto `res` retornado pelo `fetch()` contém a resposta, mas ainda não está no formato utilizável. Para extrair os dados, usamos os métodos `.json()` ou `.text()`. O método `.json()` retorna outra promise, por esse motivo tivemos de encadear o segundo método `.then()`.

```
const response = fetch("http://localhost:3010/exemplo1");
response
    // Primeiro then: verifica a resposta do servidor
    .then( function(res) {
```

```
// Converte a resposta para JSON e retorna para o próximo then
return res.json();
})
// Segundo then: pega a promise emitida por res.json()
.then( function (objeto) {
  console.log(objeto);
})
.catch( function(error) {
  console.log(error.message);
});
```

Substitua o método `json` por `text` e veja o resultado.



```
> const response = fetch("http://localhost:3010/exemplo1");
response
  // Primeiro then: verifica a resposta do servidor
  .then( function(res) {
    // Converte a resposta para JSON e retorna para o próximo then
    return res.json();
  })
  // Segundo then: pega a promise emitida por res.json()
  .then(function (objeto) {
    console.log(objeto);
  })
  .catch(function(error) {
    console.log(error.message);
  });

▶ {message: 'Boa noite!'}
```

III. Enviar dados com Fetch: parâmetros no Body

Para enviar informações ao servidor, usamos os métodos POST, PUT ou PATCH. Esses métodos exigem que os dados sejam enviados no body da requisição, geralmente no formato JSON.

Estrutura básica com body

Para enviar dados ao servidor utilizando Fetch API, devemos configurar os seguintes parâmetros:

- `method`: especifica o método HTTP a ser utilizado (POST, PUT, PATCH etc.);
- `headers`: define os cabeçalhos da requisição, como `"Content-Type": "application/json"`, para indicar que estamos enviando um corpo no formato JSON;
- `body`: contém os dados a serem enviados, convertidos para JSON utilizando `JSON.stringify()`.

A seguir, um exemplo de requisição POST enviando um objeto com dois valores (`x` e `y`) para um servidor na URL `http://localhost:3010/exemplo2`:

```
const url = "http://localhost:3010/exemplo2";
const dados = { x: 8, y: 12 };
const response = fetch(url, {
  method: "POST", // ou "PUT", "PATCH"
  headers: {
```

```

    "Content-Type": "application/json", // Indica que estamos enviando JSON
  },
  body: JSON.stringify(dados), // Converte o objeto JavaScript para JSON
});

response
  .then( function(res) {
    return res.json();
  })
  .then( function(objeto) {
    console.log(objeto);
  })
  .catch( function(error) {
    console.log(error.message);
  });

const url = "http://localhost:3010/exemplo2";
const dados = { x: 8, y: 12 };
const response = fetch(url, {
  method: "POST", // ou "PUT", "PATCH"
  headers: {
    "Content-Type": "application/json", // Indica que estamos enviando JSON
  },
  body: JSON.stringify(dados), // Converte o objeto JavaScript para JSON
});

response
  .then( function(res) {
    return res.json();
  })
  .then( function(objeto) {
    console.log(objeto);
  })
  .catch( function(error) {
    console.log(error.message);
  });
});

▶ {resultado: 20} ← Após processar a resposta do servidor

```

Exercícios

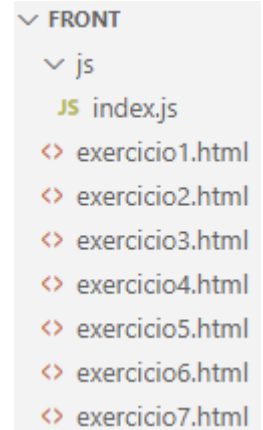
Veja os vídeos se tiver dúvidas nos exercícios:

Exercício 1 - <https://youtu.be/8piqBGeVTbQ>

Exercícios 2 a 7 - <https://youtu.be/X2wIXH2G28w>

Para fazer os exercícios você precisará ter:

- A pasta server com o servidor Node.js rodando na porta 3010 e o código fornecido no capítulo 1 dessa aula;
- A pasta front com a estrutura de pastas e arquivos mostrada ao lado. Nesses arquivos serão codificados os exercícios.



Exercício 1 – Complete a função `exer01` para processar a promise e exibir o resultado da requisição na página.

Resultado no navegador

Exibir

Boa noite!

Requisitos:

- Ao clicar no botão Exibir, deve ser realizada uma requisição ao endpoint `/exemplo1` do servidor;
- O resultado da requisição HTTP deve ser exibido no elemento `<h3 id="saida"></h3>`.

Dicas:

- Use o método `getElementById` para obter o elemento no documento HTML;
- Use a propriedade `innerText` para alterar o texto do elemento `<h3 id="saida"></h3>`.

Código do arquivo `exercicio1.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Exercício 1</title>
    <script src="./js/index.js"></script>
  </head>
  <body>
    <div>
      <button onclick="exer01()">
        Exibir
      </button>
      <h3 id="saida"></h3>
    </div>
  </body>
</html>
```

Função a ser colocada no arquivo `src/index.js`

```
function exer01() {
  const url = "http://localhost:3010/exemplo1";
  const response = fetch(url);
}
```

Exercício 2 – Complete a função `exer02` para processar a promise e exibir o resultado da requisição no campo de resultado da página.

Requisitos:

- Ao clicar no botão Somar, deve ser realizada uma requisição ao endpoint `/exemplo2`, do servidor, enviando pelo corpo da requisição os valores dos campos de entrada;
- O resultado da requisição HTTP deve ser exibido no elemento `<input value="0" id="resultado" readonly />`.

Dicas:

- Use o método `getElementById` para obter o elemento no documento HTML;
- Use a propriedade `value` para alterar o conteúdo do campo de entrada.

Resultado no navegador

Número

Número

Resultado

Código do arquivo `exercicio2.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Exercício 2</title>
    <script src="./js/index.js"></script>
  </head>
  <body>
    <div>
      <div>
        <label for="nro1">Número</label>
        <input value="0" id="nro1" />
      </div>
      <div>
        <label for="nro2">Número</label>
        <input value="0" id="nro2" />
      </div>
      <div>
        <label>Resultado</label>
        <input value="0"
          id="resultado" readonly />
      </div>
      <button
        onclick="exer02()">Somar</button>
      </div>
    </body>
  </html>
```

Função a ser colocada no arquivo `src/index.js`

```
function exer02() {
  const x = document.getElementById("nro1").value;
  const y = document.getElementById("nro2").value;
  const url = "http://localhost:3010/exemplo2";
  const response = fetch(url, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({x,y}),
  });
}
```

Exercício 3 – Codificar a função `exer03` no arquivo `src/index.js` para fazer uma

Resultado no navegador

requisição no endpoint `/multiplicacao` e exibir o resultado no campo de resultado da página.

Dica:

- Use a mesma estrutura de documento HTML do arquivo `exercicio2.html`.

Número

Número

Resultado

Exercício 4 – Codificar a função `exer04` no arquivo `src/index.js` para fazer uma requisição no endpoint `/potencia` e exibir o resultado no campo de resultado da página.

Dica:

- Use a mesma estrutura de documento HTML do arquivo `exercicio2.html`.

Resultado no navegador

Número

Número

Resultado

Exercício 5 – Codificar a função `exer05` no arquivo `src/index.js` para fazer uma requisição no endpoint `/lista` e exibir cada elemento do array em uma lista ordenada da página.

Dicas:

- Use a estrutura de repetição `for` para percorrer o array que está na propriedade `dias` do objeto enviado pelo servidor;
- Use o método `createElement` para criar um elemento ``;
- Use a propriedade `textContent` para atribuir o conteúdo do elemento ``;
- Use o método `appendChild` para adicionar o elemento `` no elemento ``.

Resultado no navegador

1. domingo
2. segunda-feira
3. terça-feira
4. quarta-feira
5. quinta-feira
6. sexta-feira
7. sábado

Código do arquivo `exercicio5.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Exercício 5</title>
    <script src="./js/index.js"></script>
  </head>
  <body>
    <div>
      <button onclick="exer05()">Listar</button>
    </div>
    <ol id="saida"></ol>
  </body>
```

</html>

Exercício 6 – Codificar a função `exer06` no arquivo `src/index.js` para fazer uma requisição no endpoint `/detalhes` e exibir cada elemento do array em uma linha da tabela da página.

Resultado no navegador

Dicas:

- Use a estrutura de repetição `for` para percorrer o array que está na propriedade `dias` do objeto enviado pelo servidor;
- Use o método `createElement` para criar um elemento `<tr>`;
- Use o método `createElement` para criar um elemento `<td>` para colocar o valor do dia;
- Use a propriedade `textContent` para atribuir o conteúdo do elemento `<td>` que recebe o dia;
- Use o método `createElement` para criar um elemento `<td>` para colocar o valor do tipo;
- Use a propriedade `textContent` para atribuir o conteúdo do elemento `<td>` que recebe o tipo;
- Use o método `appendChild` para adicionar o elemento `<td>` no elemento `<tr>`;
- Use o método `appendChild` para adicionar o elemento `<tr>` no elemento `<tbody>`.

Listar

Dia	Tipo
domingo	final de semana
segunda-feira	normal
terça-feira	normal
quarta-feira	normal
quinta-feira	normal
sexta-feira	normal
sábado	final de semana

Código do arquivo `exercicio6.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Exercício 6</title>
    <script src="./js/index.js"></script>
  </head>
  <body>
    <div>
      <button onclick="exer06()">Listar</button>
    </div>
    <table>
      <thead>
        <th>Dia</th>
        <th>Tipo</th>
      </thead>
      <tbody id="saida"></tbody>
    </table>
  </body>
</html>
```

Exercício 7 – O ViaCEP é uma API para a consulta de CEPs

Resultado no navegador

(<https://viacep.com.br>). A URL de acesso possui o seguinte formato

<https://viacep.com.br/ws/12328070/json>, em amarelo está o CEP da

Fatec Jacareí.

Como Codificar a função `exer07` no arquivo `src/index.js` para fazer uma requisição no Web Service da ViaCEP para obter o CEP fornecido pelo usuário.

Código do arquivo `exercicio7.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Exercício 7</title>
    <script src="./js/index.js"></script>
  </head>
  <body>
    <div>
      <label for="entrada">CEP</label>
      <input value="12328070" id="entrada" />
      <button onclick="exer07()">Obter</button>
    </div>
    <div>
      <p>Logradouro: <span id="logradouro"></span></p>
      <p>Bairro: <span id="bairro"></span></p>
      <p>Município: <span id="municipio"></span></p>
      <p>UF: <span id="uf"></span></p>
      <p>CEP: <span id="cep"></span></p>
    </div>
  </body>
</html>
```

CEP

Logradouro: Avenida Faria Lima

Bairro: Jardim Santa Maria

Município: Jacareí

UF: SP

CEP: 12328-070