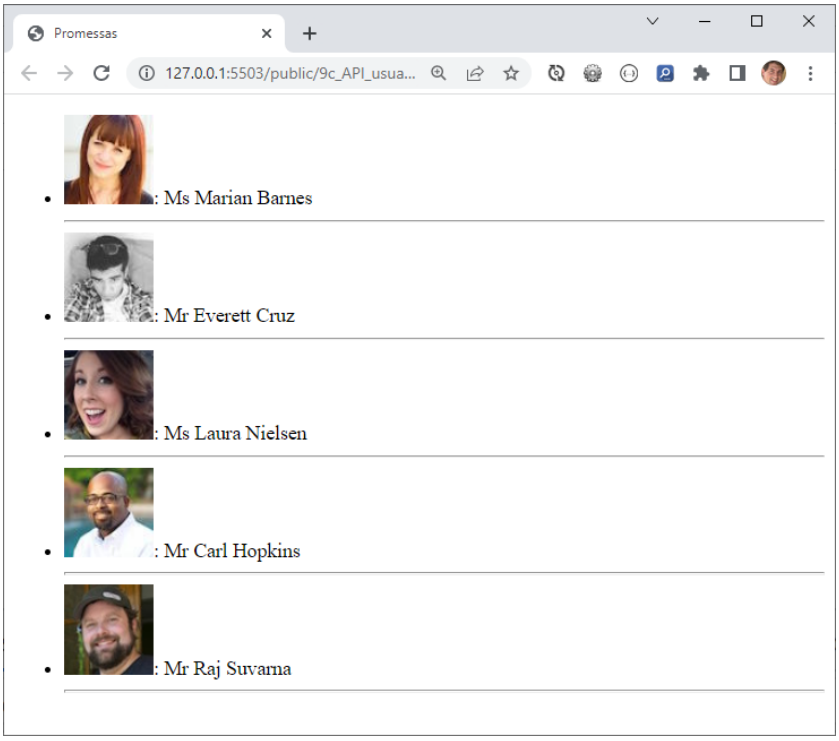
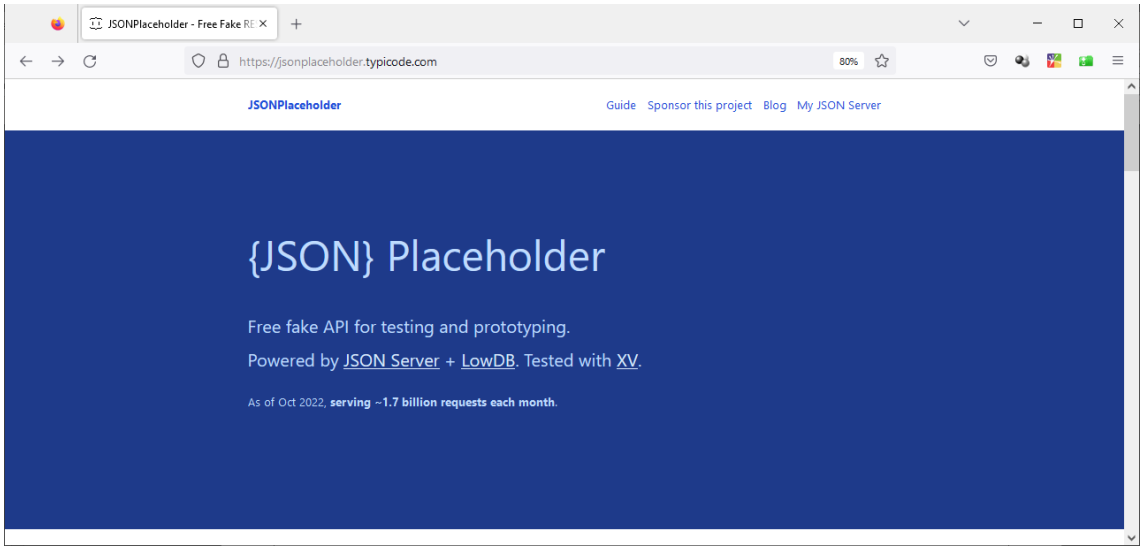
Lista 13 – JavaScript Assíncrono

1. Primeiramente baixe a versão atualizada da apresentação '*aula 32a - Fetch API (02.12.2022).pptx*'.
2. Baixe também o arquivo *aula32.rar* com os códigos feitos na aula de ontem.
3. Modifique o arquivo *1a_ws_ceps.html* de forma que ele fique como a versão 2.0 da apresentação (*slides* 9 a 11).
4. Modifique o arquivo *1b_random_people.html* de forma a mostrar também as imagens das pessoas, como mostrado no exemplo mais abaixo. Para conseguir isso, algumas dicas.
   - Dica 1: Crie dinamicamente, com `createElement()`, um objeto do tipo `span` e outro do tipo `img`.
   - Dica 2: O endereço da foto da pessoa pode ser encontrado dentro da propriedade `medium` que por sua vez está dentro da propriedade `picture` de cada usuário.
   - Dica 3: Use `appendChild()` para adicionar o `img` ao `span` e o `span` ao `li`.
   - Dica 4: Os dados dos usuários (pronome de tratamento, primeiro nome e sobrenome) deverão ser definidos no `innerHTML` do `span`.



   - Como desafio, para depois que terminar o restante dos exercícios desta lista, tente melhorar a visualização dos dados colocando-os em uma tabela responsiva com o Bootstrap em vez de uma lista simples como feito acima.
5. O exemplo a seguir ilustra o acesso a uma API externa (JSON Placeholder).



Como pode ser visto na figura abaixo, a API é baseada em promessas.

Front-end:

lista12 - index.html

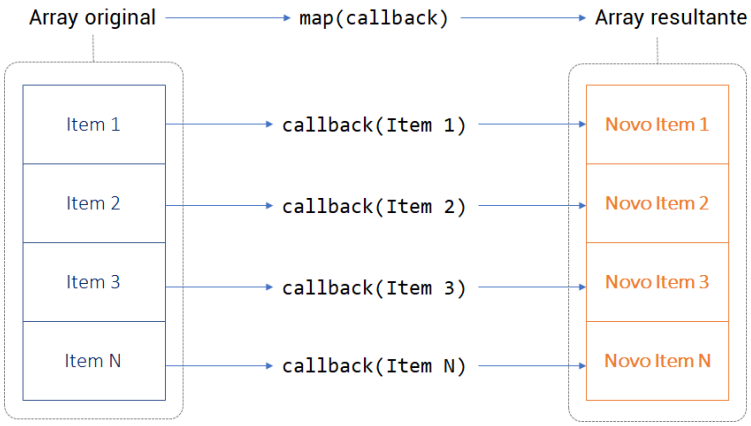
```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Promessas com Fetch API</title>
</head>
<body>
    <ul id="usuarios"></ul>

    <script src="./script.js"></script>
</body>
</html>
```

Back-end:

lista12 - script.js

```javascript
const UL = document.getElementById('usuarios');
const URL = 'https://jsonplaceholder.typicode.com/users';

fetch(URL)
    .then( (resposta) => resposta.json() )
    .then( function (dados) {
        console.log(dados);

        return dados.map(function (usuario) {
            let li = document.createElement('li');
            li.innerHTML = `${usuario.name} (${usuario.username})`;
            UL.appendChild(li);
        })
    })
    .catch( (erro) => {
        console.log('Erro!' + erro);
    })
```

A linha 9 usa o método `map()`. `map()` é um recurso da programação funcional que aplica uma função *callback* anônima a cada elemento de um *array*. `map()` foi utilizada em substituição ao laço de repetição `for()` convencional.



6. Crie uma nova pasta que irá conter o próximo projeto.

```
npm init -y
npm install express cors express-form-data
npm install --save-dev nodemon
```

Acrescentar no *package.json*

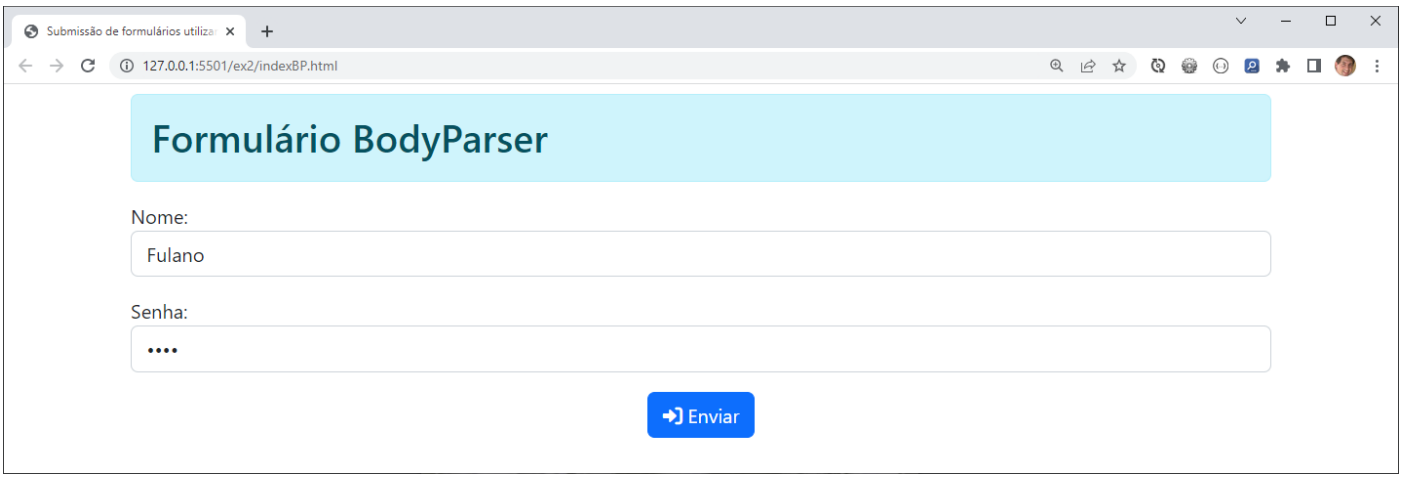
```
"scripts": {
  "devBP": "nodemon serverBP.js",
  "devM": "nodemon serverM.js",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

a) A primeira parte deste projeto usará a classe `FormData` do JavaScript para armazenar os dados do formulário. A seguir o código irá converter os itens do formulário em um objeto JSON e irá enviá-lo através do método POST via API Fetch para o *back-end*. O *back-end* irá receber a requisição, transformar o nome do usuário para letras maiúsculas e devolvê-lo de volta ao *front-end* em formato JSON. Ao receber a resposta da requisição o *front-end* irá mostrá-la no *console*.

b) Código do *front-end* (arquivo *indexBP.html*):

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Submissão de formulários utilizando a API Fetch</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.2.0/css/all.min.css"
    crossorigin="anonymous">
</head>
<body>
    <div class="container">
        <h2 class="alert alert-info mt-2">Formulário BodyParser</h2>

        <form id="formulario">
            <div class="form-group">
                Nome: <input type="text" name="usuario" class="form-control" value="Fulano">
            </div>

            <div class="form-group mt-3">
                Senha: <input type="password" name="senha" class="form-control" value="1234">
            </div>

            <div class="form-group mt-3 text-center">
                <button type="submit" class="btn btn-primary">
                    <i class="fa-solid fa-right-to-bracket"></i>
                    Enviar
                </button>
            </div>
        </form>
    </div>
```

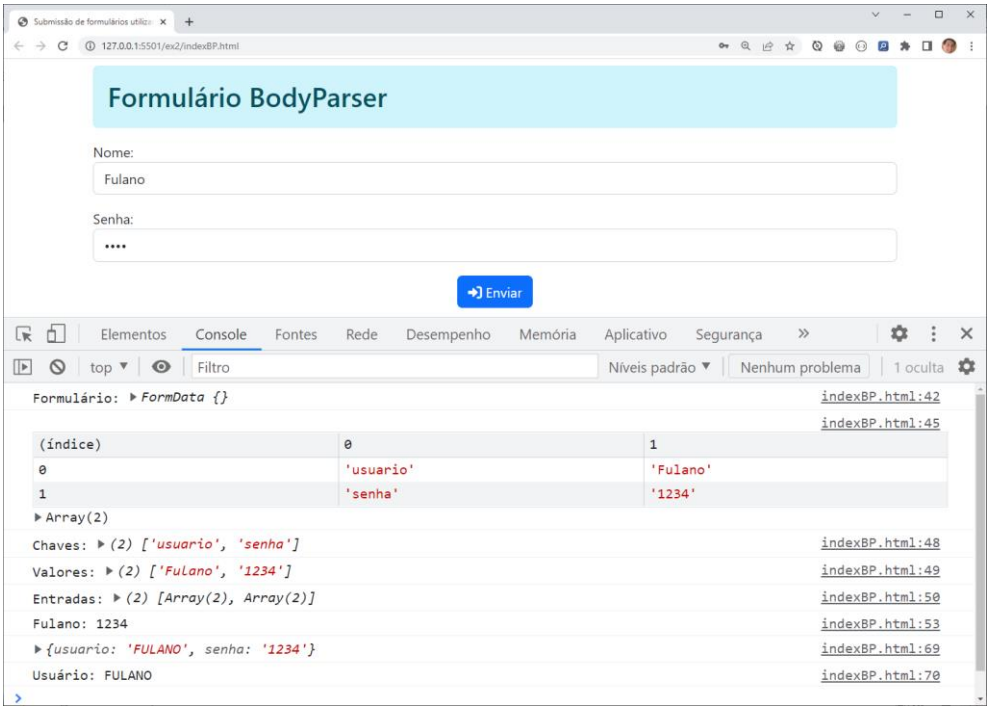
```
    <script>
        const formulario = document.getElementById('formulario');

        formulario.addEventListener('submit', function(e) {
            e.preventDefault();

            const dadosForm = new FormData(formulario);
            console.log('Formulário:', dadosForm);

            // Utilização do operador spread (...) para expandir o array
            console.table([...dadosForm]);

            // Transforma um iterador em um array
            console.log('Chaves:', Array.from(dadosForm.keys()));
            console.log('Valores:', Array.from(dadosForm.values()));
            console.log('Entradas:', Array.from(dadosForm.entries()));

            // Recupera campos individuais
            console.log(dadosForm.get('usuario') + ': ' + dadosForm.get('senha'));

            fetch('http://localhost:8080/postBP', {
                    method: 'POST',
                    body: JSON.stringify(Object.fromEntries(dadosForm)),
                    headers: {
                        'Content-Type': 'application/json; charset=UTF-8'
                    }
                })
                .then(res => {
                    if (!res.ok) {
                        console.log('Página não encontrada!');
                    }
                    return res.json();
                })
                .then(data => {
                    console.log(data);
                    console.log('Usuário:', data.usuario);
                })
                .catch(() => console.log('Erro!'));
        })
    </script>
</body>
</html>
```



c) Código do *back-end* (arquivo $serverBP.html$). A biblioteca $body\text{-}parser$ foi incorporada à biblioteca Express.js, portanto não é mais preciso baixá-la e utilizá-la separadamente. Para rodar o projeto execute o comando `npm run devBP`

lista12 - serverBP.js

```
const express = require('express');
const cors = require('cors');

const app = express();
const porta = 8080;

// Configura o CORS
app.use(cors({ origin: '*' }));

// Configura o tratamento das requisições POST
app.use(express.urlencoded({extended: true}));
app.use(express.json());

// Instancia o servidor
app.listen(porta,
    () => console.log(`Servidor iniciado na porta: ${porta}`)
);

// Responde a requisição no endereço http://localhost:8080/
app.get('/', (request, response) => {
    response.status(200).send('<h2>Servidor node.js :-)</h2>');
});

// Tratamento da requisição via POST
app.post('/postBP', function(req, res) {
    console.log('Servidor:', req.body);
    console.log('Servidor:', req.body.usuario);

    req.body.usuario = req.body.usuario.toUpperCase();

    if (req.body.usuario === '') {
        res.status(404).send({});
    } else {
        res.status(200).send(req.body);
    }
});
```

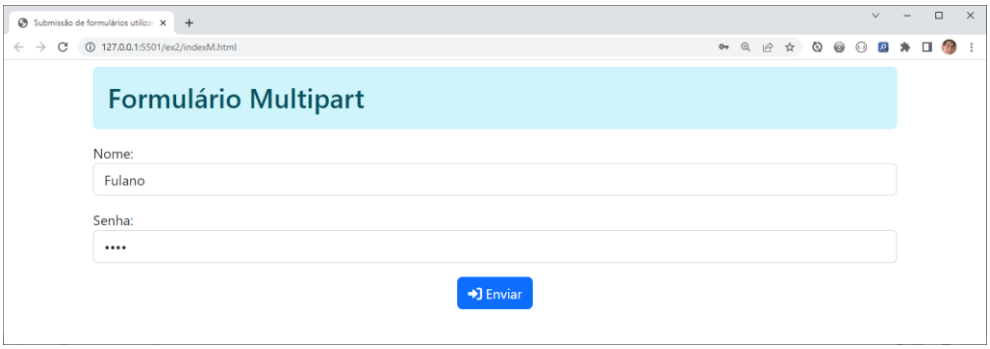


d) Interrompa a execução do servidor e clique novamente no botão $Enviar$. Veja a mensagem no console do dev-tools.

e) Apague o nome Fulano do campo do formulário e o envie novamente. Verifique o que vai acontecer no console.

f) A segunda parte deste projeto irá enviar os dados no formato `multipart/form-data`. Para isso, faça uma cópia do arquivo $indexBP.html$ para o arquivo $indexM.html$ e altere a parte do código que usa o `fetch()` para fazer a requisição, como mostrado abaixo.

```
56          fetch('http://localhost:8081/postM', {
57              method: 'POST',
58              body: dadosForm
59          })
60          .then(res => {
61              if (!res.ok) {
62                  console.log('Página não encontrada!');
63              }
64              return res.json();
65          })
66          .then(data => {
67              console.log(data)
68          })
69          .catch(() => console.log('Erro!'));
70      })
```



g) Código do *back-end* (arquivo $serverM.js$). Este arquivo usa a biblioteca express-form-data para ler os dados enviados pelo *front-end* no formato *multipart*. Para rodar o projeto execute o comando `npm run devM`

lista12 - serverM.js

```
const express = require('express');
const cors = require('cors');
const formData = require('express-form-data');

const app = express();
const porta = 8081;

// Configura o CORS
app.use(cors({ origin: '*' }));

// Faz o parsing no formato multipart
app.use(formData.parse());

// Instancia o servidor
app.listen(porta,
    () => console.log(`Servidor iniciado na porta: ${porta}`)
);

// Responde a requisição no endereço http://localhost:8080/
app.get('/', (request, response) => {
    response.status(200).send('<h2>Servidor node.js :-)</h2>');
});

// Tratamento da requisição via POST
app.post('/postM', function(req, res) {
    console.log('Servidor:', req.body);
    console.log('Servidor:', req.body.usuario);

    req.body.usuario = req.body.usuario.toUpperCase();

    if (req.body.usuario === '') {
        res.status(404).send({});
    } else {
        res.status(200).send(req.body);
    }
});
```

h) Interrompa a execução do servidor e clique novamente no botão $Enviar$. Veja a mensagem no console do dev-tools.

i) Apague o nome Fulano do campo do formulário e o envie novamente. Verifique o que vai acontecer no console.