JavaScript Assíncrono (Promessas / API Fetch / Async - Await)

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

Herculano De Biasi
herculano.debiasi@unoesc.edu.br

Tópicos

- Promessas (*Promises*)
- API Fetch
- *Async / Await*

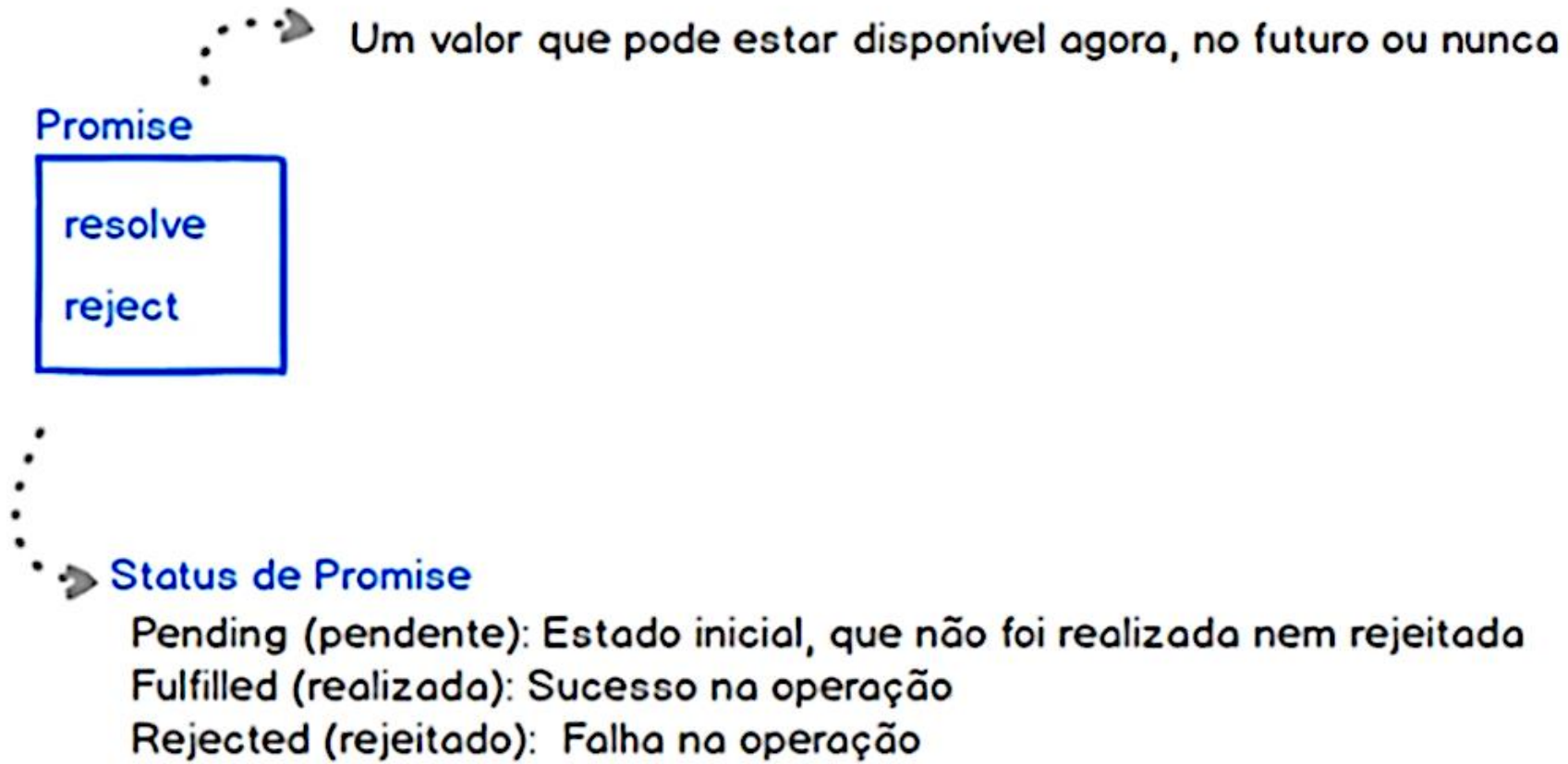
PROMESSAS (*PROMISES*)

- *Promise* (promessa) é um padrão de desenvolvimento muito usado no JavaScript que representa fluxos de ações em operações assíncronas, mais especificamente, o tratamento da conclusão destas operações
  - Um objeto *promise* guarda a promessa de que a função que o gerou irá, eventualmente, em algum momento desconhecido no futuro terminar e retornar um resposta
  - Uma *promise* representa a eventual conclusão (ou falha) de uma operação assíncrona e seu valor resultante
  - Elas foram implementadas no ES6, antes disso a maioria das funções utilizava *callbacks* (funções de retorno de chamada) assíncronos
  - Antes do ES6 elas apareceram em bibliotecas como a Dojo *Toolkit* e jQuery

- Uma promessa define uma ação que será executada no futuro, ou seja, ela pode ser resolvida (com sucesso) ou rejeitada (com erro)

PROMESSAS (*PROMISES*)

- Ideia geral, estados (*pending*, *fulfilled* e *rejected*) e parâmetros (`resolve` e `reject`)
  - Quando uma promessa foi cumprida (*fulfilled*) ou rejeitada (*rejected*) é dito que ela passou para o estado *settled* (resolvido decidido, assentado, determinado)

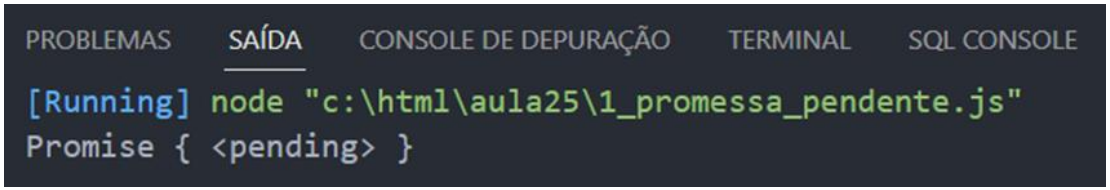


PROMESSAS (*PROMISES*)

- Verificação de estado: Pendente

aula25 - 1_promessa_pendente.js

```
let promessa = new Promise((resolve, reject) => {
    try {
        // Promise executada com sucesso
        setTimeout(() => {
            // Ponto de solução da promessa
            resolve('Promessa concluída com sucesso!');
        }, 3000);
    } catch (e) {
        // Erro na execução da promessa
        setTimeout(() => {
            // Promessa rejeitada
            reject(e);
        }, 3000);
    }
});

console.log(promessa);
```

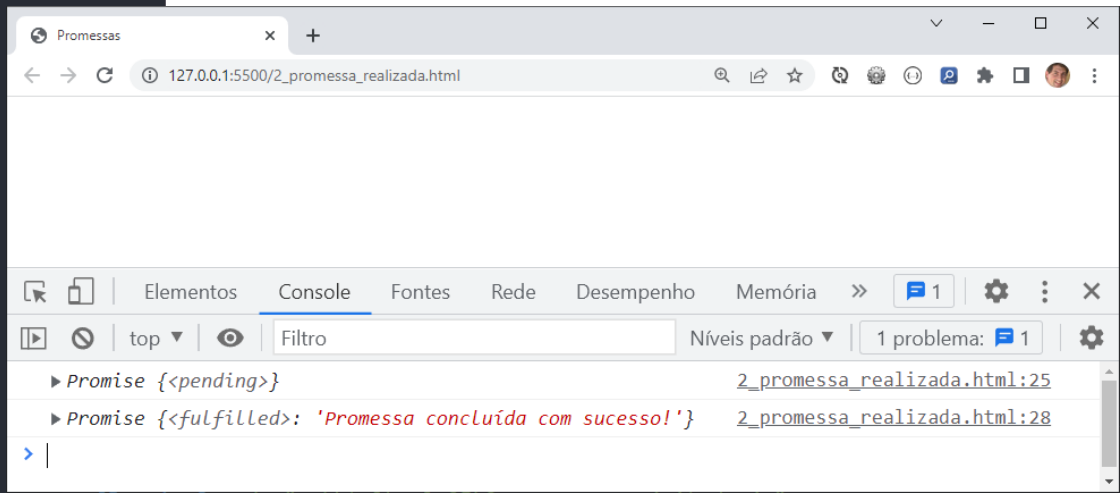


PROMESSAS (*PROMISES*)

- Verificação de estado: Realizada com sucesso

aula25 - 2_promessa_realizada.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</title>

    <script>
        let promessa = new Promise((resolve, reject) => {
            try {
                // Promise executada com sucesso
                setTimeout(() => {
                    resolve('Promessa concluída com sucesso!');
                }, 3000);
            } catch (e) {
                // Erro na execução da promessa
                setTimeout(() => {
                    reject(e);
                }, 3000);
            }
        });

        console.log(promessa);

        setTimeout(() => {
            console.log(promessa);
        }, 5000);
    </script>
</head>

<body>
</body>

</html>
```

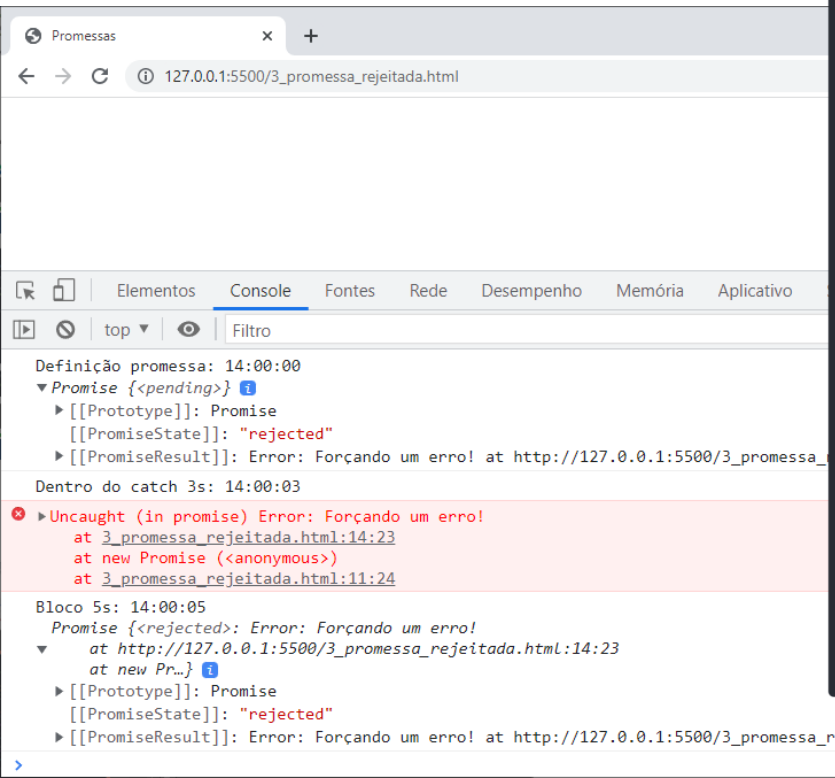


PROMESSAS (*PROMISES*)

- Verificação de estado: Rejeitada

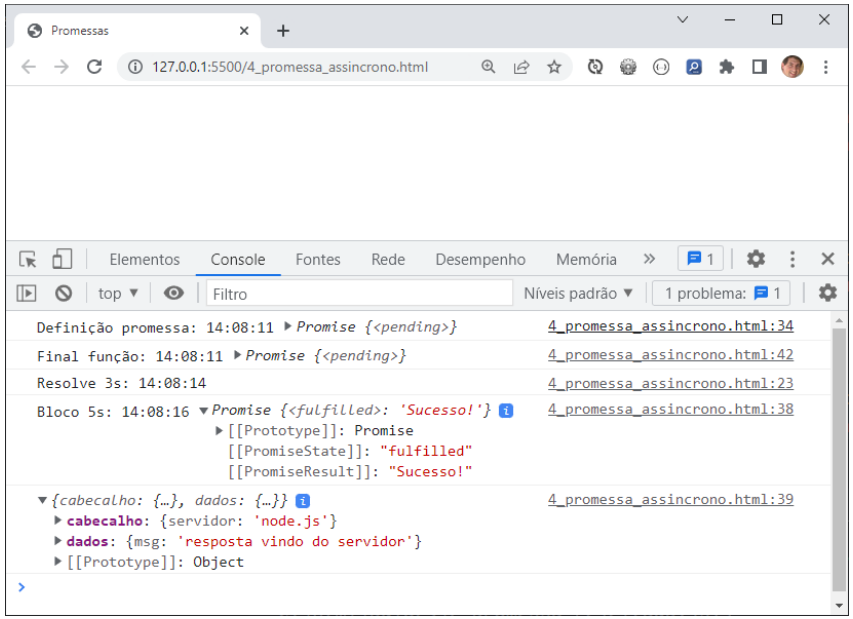
aula25 - 3_promessa_rejeitada.html

```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</title>

    <script>
        let promessa = new Promise((resolve, reject) => {
            try {
                // Forçando uma condição de erro
                throw new Error('Forçando um erro!');

                // Promise executada com sucesso (não será executado)
                setTimeout(() => {
                    resolve('Promessa concluída com sucesso!');
                }, 3000);
            } catch (e) {
                // Erro na execução da promessa
                setTimeout(() => {
                    console.log('Dentro do catch 3s:', new Date().toISOString().substr(11,8));
                    reject(e);
                }, 3000);
            }
        });

        console.log('Definição promessa:', new Date().toISOString().substr(11,8), promessa);

        setTimeout(() => {
            console.log('Bloco 5s:', new Date().toISOString().substr(11,8), promessa);
        }, 5000);
    </script>
</head>

<body>
</body>

</html>
```



PROMESSAS (*PROMISES*)

- Exemplo de simulação de uma requisição assíncrona



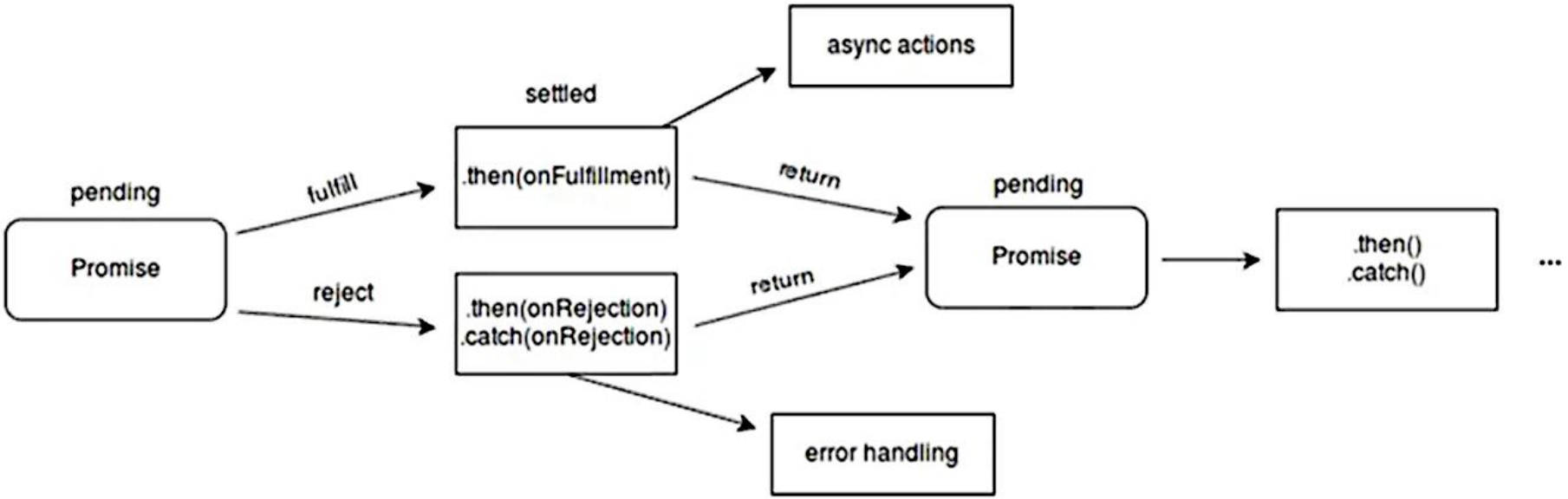
aula25 - 4_promessa_assincrono.html

```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</title>

    <script>
        // Simulando uma resposta de uma requisição HTTP
        let resposta = {};

        let promessa = new Promise((resolve, reject) => {
            try {
                // Promise executada com sucesso (não será executado)
                setTimeout(() => {
                    resposta = {
                        cabecalho: { servidor: 'node.js' },
                        dados: { msg: 'resposta vindo do servidor' }
                    };

                    console.log('Resolve 3s:', new Date().toISOString().substr(11,8));
                    resolve('Sucesso!');
                }, 3000);
            } catch (e) {
                // Erro na execução da promessa
                setTimeout(() => {
                    reject(e);
                }, 3000);
            }
        });

        console.log('Definição promessa:',
                    new Date().toISOString().substr(11,8),
                    promessa);

        // Bloco 5s
        setTimeout(() => {
            console.log('Bloco 5s:', new Date().toISOString().substr(11,8), promessa);
            console.log(resposta);
        }, 5000);

        console.log('Final função:', new Date().toISOString().substr(11,8), promessa);
    </script>
</head>

<body>
</body>

</html>
```

UNOESC

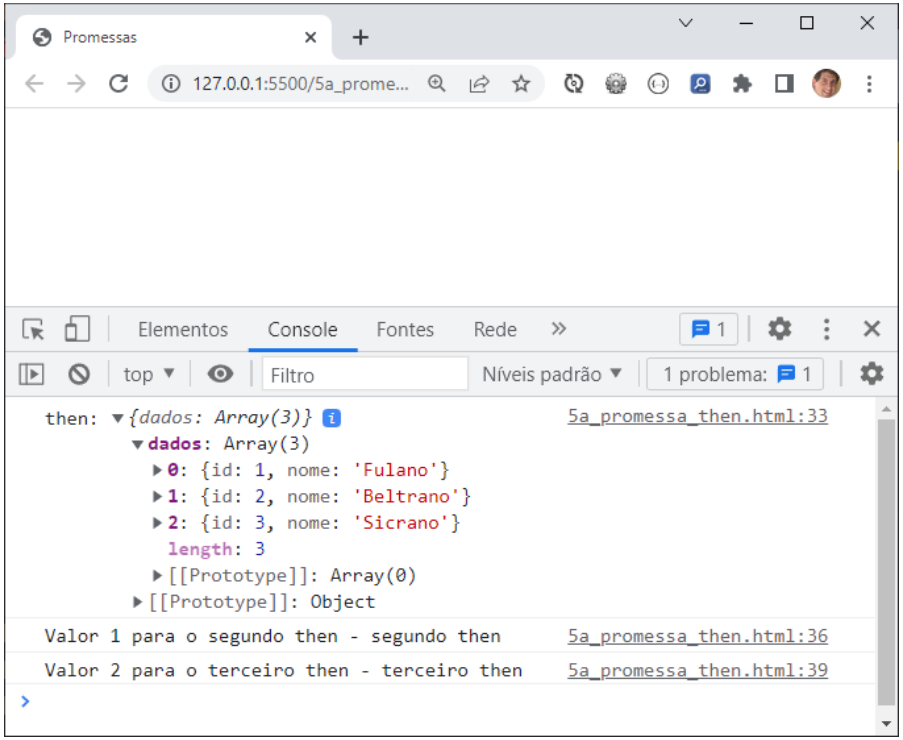
PROMESSAS (*PROMISES*)

- Funcionamento
  - Um resultado bem sucedido, gerado pela chamada da função `resolve`, desencadeia a chamada do método `then`, ou seja, o método `then` executa uma função *callback* mediante a resolução da promessa
  - Um resultado mal sucedido, gerado pela função `reject`, desencadeia a chamada do método `catch`, ou seja, o método `catch` executa uma função *callback* em caso de rejeição da promessa
  - Qualquer valor passado para a função `resolve` será acessível como parâmetro da função `then`, podendo ser uma `String`, `Number`, `Booleano`, `Function`, `Object`, etc – o mesmo ocorre entre `reject` e `catch`



PROMESSAS (*PROMISES*)

- Exemplo: `then()`



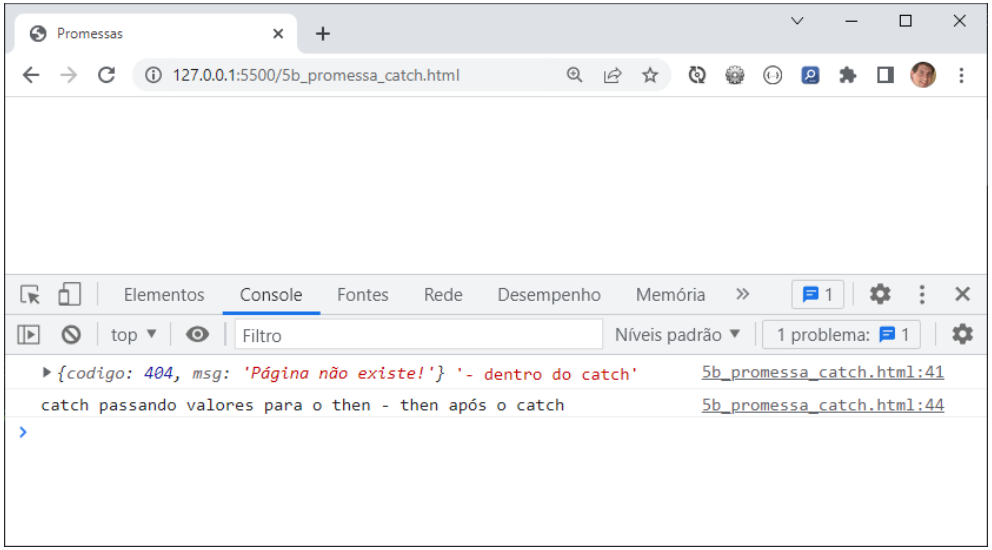
aula25 - 5a_promessa_then.html

```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</title>

    <script>
        let promessa = new Promise((resolve, reject) => {
            // Simulação de uma requisição HTTP
            setTimeout(() => {
                // Requisição foi realizada
                let resposta = {};
                const erro_requisicao = false;

                // Erro na requisição
                if (erro_requisicao) {
                    resposta = { codigo: 404, msg: 'Página não existe!'}
                    reject(resposta);
                }

                resposta = { dados: [
                    { id: 1, nome: 'Fulano' },
                    { id: 2, nome: 'Beltrano' },
                    { id: 3, nome: 'Sicrano' }
                ]};

                resolve(resposta);
            }, 4000);
        }).then(dados => {
            console.log('then:', dados);
            return('Valor 1 para o segundo then')
        }).then(dados => {
            console.log(dados, '- segundo then');
            return('Valor 2 para o terceiro then')
        }).then(dados => {
            console.log(dados, '- terceiro then');
        });
    </script>
</head>

<body>
</body>

</html>
```

UNOESC

PROMESSAS (*PROMISES*)

- Exemplo: `catch()`



aula25 - 5b_promessa_catch.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</title>

    <script>
        let promessa = new Promise((resolve, reject) => {
            // Simulação de uma requisição HTTP
            setTimeout(() => {
                // Requisição foi realizada
                let resposta = {};
                const erro_requisicao = true;

                // Erro na requisição
                if (erro_requisicao) {
                    resposta = { codigo: 404, msg: 'Página não existe!'}
                    reject(resposta);
                }

                resposta = { dados: [
                    { id: 1, nome: 'Fulano' },
                    { id: 2, nome: 'Beltrano' },
                    { id: 3, nome: 'Sicrano' }
                ]};

                resolve(resposta);
            }, 4000);
        }).then(dados => {
            console.log('then:', dados);
            return('Valor 1 para o segundo then')
        }).then(dados => {
            console.log(dados, '- segundo then');
            return('Valor 2 para o terceiro then')
        }).then(dados => {
            console.log(dados, '- terceiro then');
        }).catch(erro => {
            console.log(erro, '- dentro do catch');
            return('catch passando valores para o then')
        }).then(dados => {
            console.log(dados, '- then após o catch');
        });
    </script>
</head>

<body>
</body>

</html>
```

UNOESC

PROMESSAS (*PROMISES*)

- Exemplo: Divisão

```
[Running] node "c:\html\aula25\6_promessa_divisao.js"
Divisão normal de 10 por 2 = 5
Divisão normal de 10 por 0 = Infinity
Divisão normal de  0 por 0 = NaN

Sucesso: 5
```

aula25 - 6_promessa_divisao.js

```
let log = console.log;

function dividir(a, b) {
    return a / b;
}

log('Divisão normal de 10 por 2 = ' + dividir(10, 2));
log('Divisão normal de 10 por 0 = ' + dividir(10, 0));
log('Divisão normal de  0 por 0 = ' + dividir(0, 0));
log('');

//-------

function promessaDivisao(a, b) {
    return new Promise(function (resolve, reject) {
        if (b === 0) {
            reject(new Error('Não é possível dividir por 0 !'));
            return;
        }

        resolve(a / b);
    });
}

promessaDivisao(10, 2).then(function (resultado) {
    log(`Sucesso: ${resultado}`);
}).catch(function (erro) {
    log('Erro na divisão');
    log(erro);
});
```

UNOESC

PROMESSAS (*PROMISES*)

- Exemplo: Divisão

```
[Running] node "c:\html\aula25\6_promessa_divisao.js"
Divisão normal de 10 por 2 = 5
Divisão normal de 10 por 0 = Infinity
Divisão normal de  0 por 0 = NaN

Erro na divisão
Error: Não é possível dividir por 0 !
    at c:\html\aula25\6_promessa_divisao.js:17:20
    at new Promise (<anonymous>)
    at promessaDivisao (c:\html\aula25\6_promessa_divisao.js:15:12)
    at Object.<anonymous> (c:\html\aula25\6_promessa_divisao.js:25:1)
    at Module._compile (node:internal/modules/cjs/loader:1126:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1180:10)
    at Module.load (node:internal/modules/cjs/loader:1004:32)
    at Function.Module._load (node:internal/modules/cjs/loader:839:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:81:12)
    at node:internal/main/run_main_module:17:47
```

aula25 - 6_promessa_divisao.js

```
1  let log = console.log;
2
3  function dividir(a, b) {
4      return a / b;
5  }
6
7  log('Divisão normal de 10 por 2 = ' + dividir(10, 2));
8  log('Divisão normal de 10 por 0 = ' + dividir(10, 0));
9  log('Divisão normal de  0 por 0 = ' + dividir(0, 0));
10 log('');
11
12 //-------
13
14 function promessaDivisao(a, b) {
15     return new Promise(function (resolve, reject) {
16         if (b === 0) {
17             reject(new Error('Não é possível dividir por 0 !'));
18             return;
19         }
20
21         resolve(a / b);
22     });
23 }
24
25 promessaDivisao(10, 0).then(function (resultado) {
26     log(`Sucesso: ${resultado}`);
27 }).catch(function (erro) {
28     log('Erro na divisão');
29     log(erro);
30 });
```

UNOESC

PROMESSAS (*PROMISES*)

- Exemplo

aula25 - 7a_limpar_quartojs

```
let promessaDeLimparQuarto = new Promise((resolve, reject) => {
    /* Processo: Limpando o quarto
        .
        .
        .    */
    let estaLimpo = true;

    if (estaLimpo) {
        resolve('está limpo');
    } else {
        reject('não está limpo');
    }
});

promessaDeLimparQuarto.then(function(doResolve) {
    console.log('O quarto ' + doResolve)
}).catch(function(doReject) {
    console.log('O quarto ' + doReject)
});
```

UNOESC

PROMESSAS (*PROMISES*)

- Dependências, execução sequencial e paralelismo de promessas com `all` e `race`

aula25 - 7b_limpar_casa.js

```
let limparQuarto = new Promise((resolve, reject) => {
    resolve('Limpei o quarto! ');
});

let retirarLixo = function (mensagem) {
    return new Promise((resolve, reject) => {
        resolve(mensagem + '-> Retirei o lixo! ')
    });
};

let ganharSorvete = function (mensagem) {
    return new Promise((resolve, reject) => {
        resolve(mensagem + '-> Ganhei sorvete! ')
    });
};

limparQuarto.then(resultado => {
    return retirarLixo(resultado);
}).then(resultado => {
    return ganharSorvete(resultado);
}).then(resultado => {
    console.log(resultado + '-> Fim!');
});

Promise.all([limparQuarto, retirarLixo(), ganharSorvete()])
    .then(() => console.log('Todas as tarefas foram finalizadas!'));

Promise.any([limparQuarto, retirarLixo(''), ganharSorvete('')])
    .then(tarefa => console.log(`Uma das tarefas - ${tarefa}- encerrada!\n`));
```

UNOESC

API Fetch

- Em 2015 foi lançada a Fetch API, que possui funcionalidades semelhante ao objeto XHR (`XMLHttpRequest`) mas de mais alto nível e baseado em promessas

```
fetch('send-ajax-data.php')
    .then(data => console.log(data))
    .catch (error => console.log('Error:' + error));
```

API Fetch

- Consumindo a API (*webservice*) de consulta de CEPs da ViaCEP

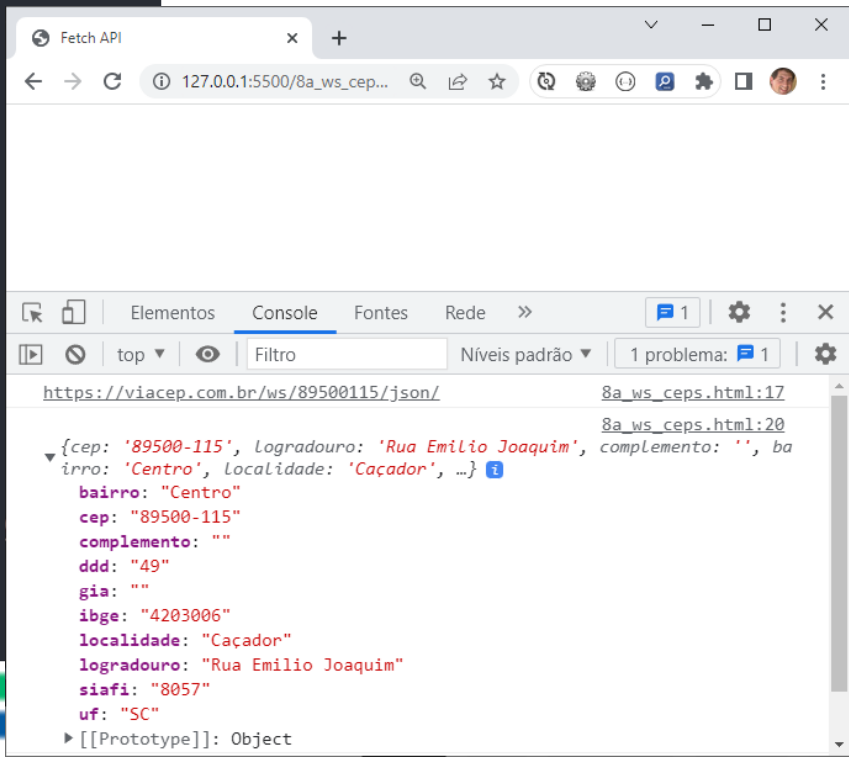
UNOESC

API Fetch

- Versão mais simples

aula25 - 8a_ws_ceps.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>Fetch API</title>
</head>

<body>
    <ul id="usuarios"></ul>

    <script>
        const CEP = '89500115';
        const URL = `https://viacep.com.br/ws/${CEP}/json/`;

        console.log(URL);

        fetch(URL)
            .then(response => response.json())
            .then(dados => console.log(dados))
    </script>
</body>

</html>
```

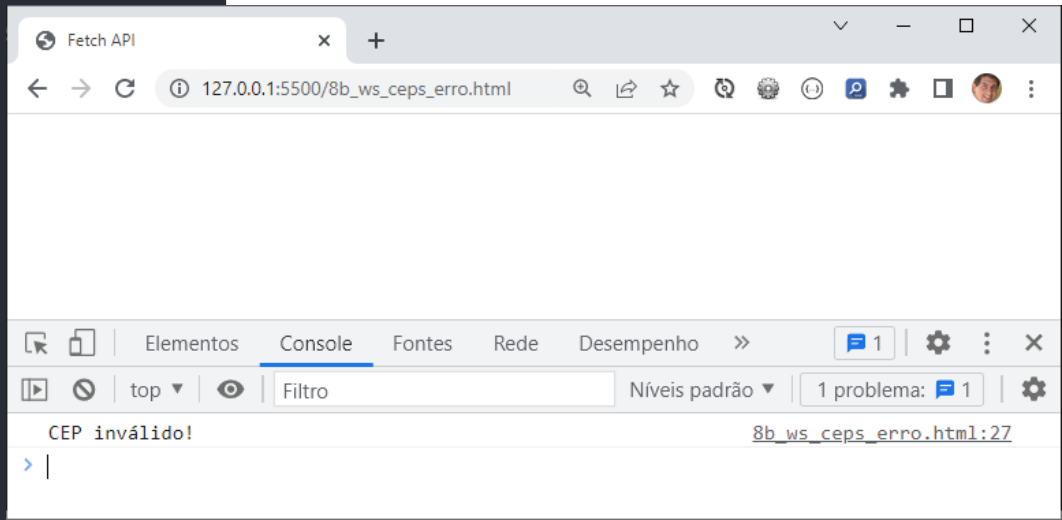


API Fetch

- Versão mais completa, com opções e tratamento de erro

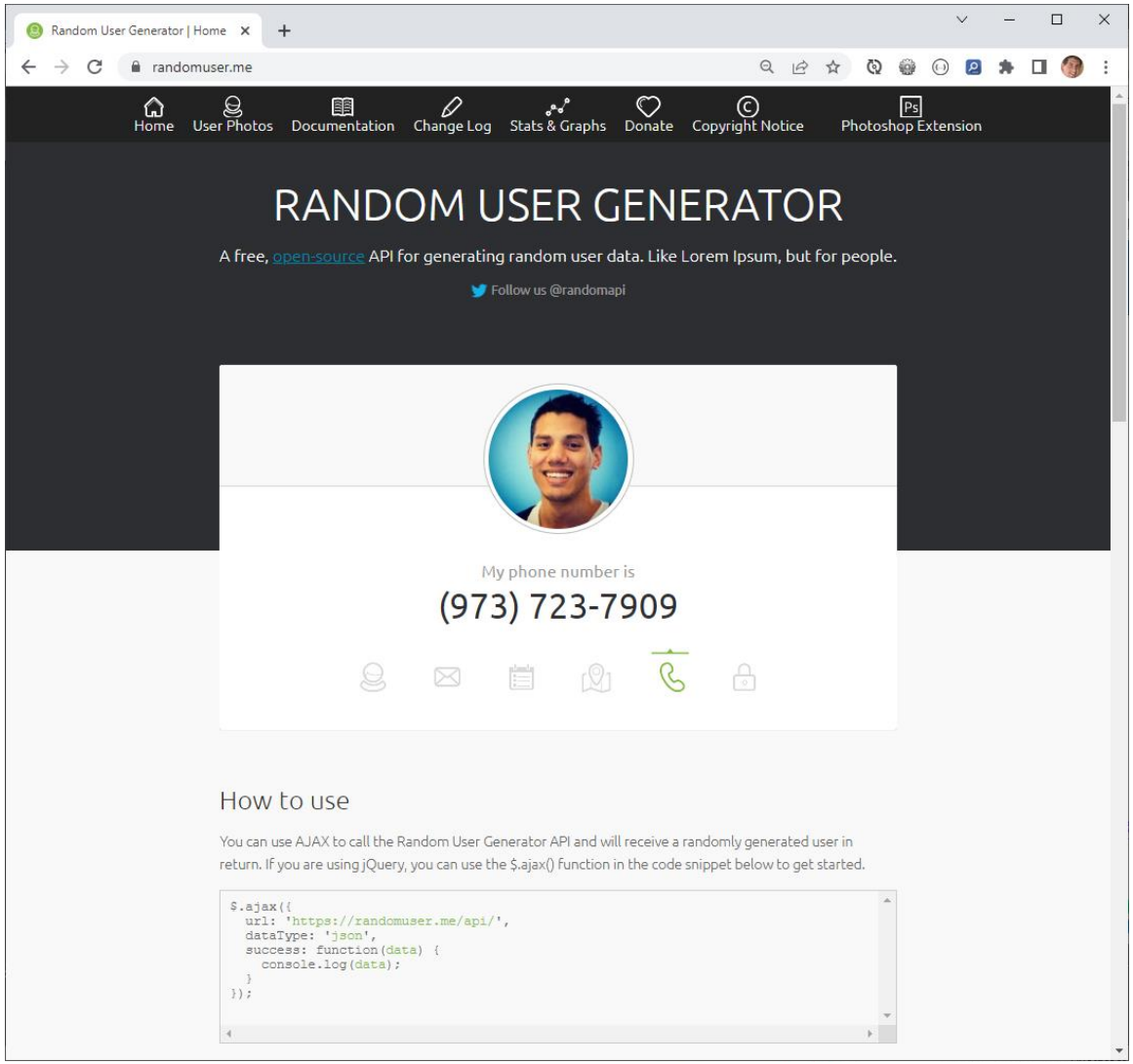
aula25 - 8b_ws_ceps_erro.html

```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>Fetch API</title>
</head>

<body>
    <ul id="usuarios"></ul>

    <script>
        const CEP = '89500000';
        const URL = `https://viacep.com.br/ws/${CEP}/json/`;

        const opcoes = {
            method: 'GET',
            mode: 'cors',
        };

        fetch(URL, opcoes)
            .then(response => response.json())
            .then(dados => {
                if (dados.erro) {
                    console.log('CEP inválido!');
                }
            })
    </script>
</body>

</html>
```



API Fetch

- Acessando uma API que gera dados de usuários aleatórios



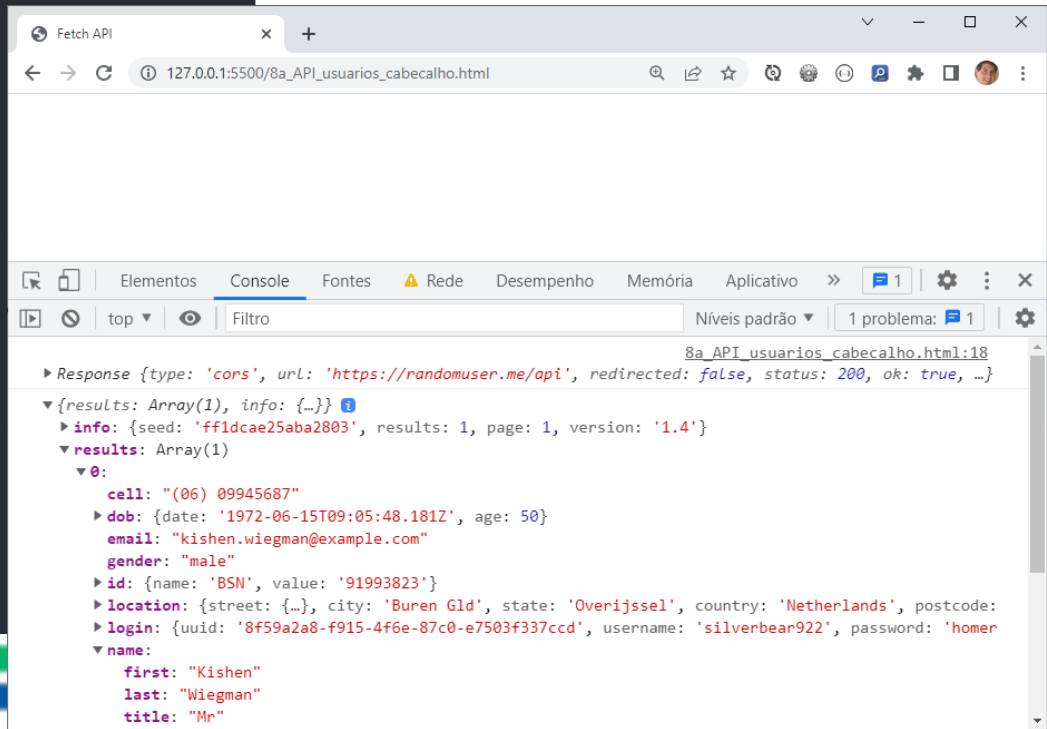
UNOESC

API Fetch

- Mostrando a resposta com seu cabeçalho

aula25 - 8a_API_usuarios_cabecalho.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>Fetch API</title>
</head>

<body>
    <ul id="usuarios"></ul>

    <script>
        const ul = document.getElementById('usuarios');

        fetch('https://randomuser.me/api')
            // Cabeçalho
            .then(resposta => {
                console.log(resposta);
                return resposta;
            })
            // Converte para JSON
            .then(resposta => resposta.json())
            .then(console.log)
    </script>
</body>

</html>
```

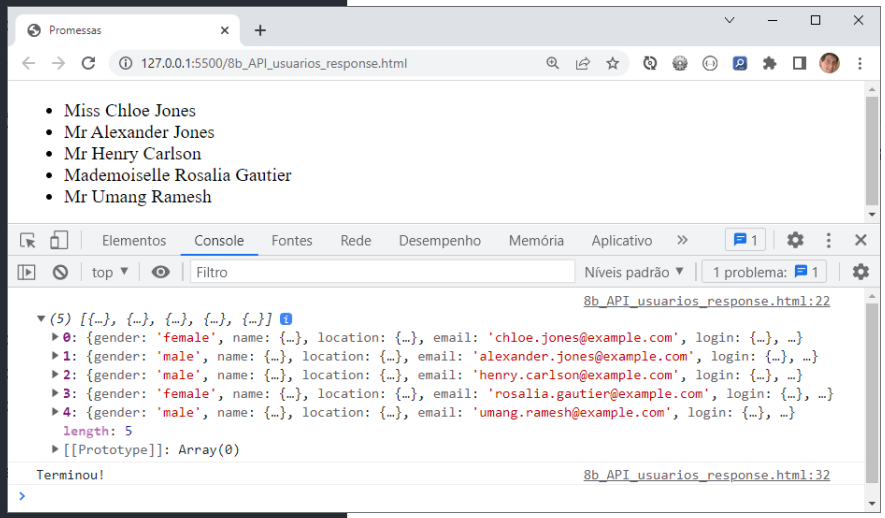


API Fetch

- Tratando os dados

aula25 - 8b_API_usuarios_response.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</title>
</head>

<body>
    <ul id="usuarios"></ul>

    <script>
        const ul = document.getElementById('usuarios');

        fetch('https://randomuser.me/api?results=5')
            .then(resposta => resposta.json())
            .then(resposta => {
                // throw new Error('teste');
                usuarios = resposta.results;
                console.log(usuarios);

                for (usuario of usuarios) {
                    let li = document.createElement('li');
                    li.innerHTML = `${usuario.name.title} ${usuario.name.first} ${usuario.name.last}`;
                    ul.appendChild(li);
                }

            })
            .catch(erro => console.log('Oops!' + erro))
            .finally(() => console.log('Terminou!'));
    </script>
</body>

</html>
```



ASYNC / AWAIT

- Os métodos `async` e `await` foram implementados no ES8 (ES2017) e constituem uma sintaxe que simplifica a programação assíncrona, facilitando o fluxo de escrita e leitura do código; assim é possível escrever código que funciona de forma assíncrona, porém lido e estruturado de forma síncrona
  - O async/await também trabalha com o código assíncrono baseado em promessas, porém esconde as promessas para que a leitura e a escrita seja mais fluídas
  - Definindo uma função como async, pode-se utilizar a palavra-chave await antes de qualquer expressão que retorne uma promessa; dessa forma, a execução da função externa (a função async) será pausada até que a promessa seja resolvida – ou seja, quando se usa await, o JavaScript vai aguardar até que a promessa finalize
  - A palavra-chave await recebe uma promessa e a transforma em um valor de retorno (ou lança uma exceção em caso de erro)
    - Se a promessa for finalizada com sucesso (*fulfilled*), o valor obtido é retornado
    - Se a promessa for rejeitada (*rejected*), é retornado o erro lançado pela exceção

ASYNC / AWAIT

- Leitura de arquivos: Versão com promessas (*promises*)

```
1  const fs = require('fs')
2
3  const leArquivo = arquivo => new Promise((resolve, reject) => {
4      fs.readFile(arquivo, (erro, conteudo) => {
5          if(erro) {
6              reject(erro)
7          } else {
8              resolve(conteudo)
9          }
10     })
11 })
12
13 // Encadeamento
14 const promessa = leArquivo('./in1.txt')
15 // console.log(promessa)
16     .then(conteudo => {
17         console.log(String(conteudo))
18         return leArquivo('./in2.txt')
19     })
20     .then(conteudo => {
21         console.log(String(conteudo))
22     })
23     .finally(() => {
24         console.log('Terminou!')
25     })
26
27 setTimeout(() => console.log(promessa), 1000)
```

UNOESC

CENTRO DE INOVAÇÃO VIDEIRA

GOVERNO DE SANTA CATARINA

ASYNC / AWAIT

- Leitura de arquivos: Versão com *async/await*

```
1  const fs = require('fs')
2
3  const leArquivo = arquivo => new Promise((resolve, reject) => {
4      fs.readFile(arquivo, (erro, conteudo) => {
5          if(erro) {
6              reject(erro)
7          } else {
8              resolve(conteudo)
9          }
10     })
11 })
12
13 init = async (valor) => {
14     try {
15         let conteudo1 = await leArquivo('./in1.txt')
16         let conteudo2 = await leArquivo('./in2.txt')
17         return String(conteudo1) + '\n' + String(conteudo2)
18     } catch(erro) {
19         console.log(erro)
20     }
21 };
22
23 init().then( conteudo => console.log(conteudo))
24 console.log('init', init())
```

Async / Await

- *Async/Await* podem ser usadas para escrever *Promises* de forma mais concisa

```
1  const random = () => {
2      return Promise.resolve(Math.trunc(Math.random()*100))
3  }
4
5  const sumRandomAsyncNums = () => {
6      let first, second, third
7
8      return random()
9          .then(v => {
10             first = v
11             return random()
12         })
13         .then(v => {
14             second = v
15             return random()
16         })
17         .then(v => {
18             third = v
19             return first + second + third
20         })
21         .then(v => {
22             console.log(`${first}+${second}+${third}=${v}`)
23         })
24 }
25
26 sumRandomAsyncNums()
```

```
1  const random = () => {
2      return Promise.resolve(Math.trunc(Math.random()*100))
3  }
4
5  const sumRandomAsyncNums = async() => {
6      const first = await random()
7      const second = await random()
8      const third = await random()
9
10     const v = first + second + third
11
12     console.log(`${first}+${second}+${third}=${v}`)
13 }
14
15 sumRandomAsyncNums()
```

Async / Await

- `navigator`: Informações sobre o navegador
  - Recurso de geolocalização

aula25 - 10_bom_navigator_geolocation.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>BOM - Navigator - Geolocalização</title>

    <script>
        let local;

        const requisitaPosicao = () => {
            const opcoes = {
                enableHighAccuracy: true,
                timeout: 5000,
                maximumAge: 0
            };

            return new Promise(function (resolve, reject) {
                navigator.geolocation.getCurrentPosition(
                    pos => {
                        resolve(local = `Latitude : ${pos.coords.latitude} x Longitude: ${pos.coords.longitude}`)
                    },
                    err => { reject(err) },
                    opcoes
                );
            });
        }

        async function infosNavegador() {
            await requisitaPosicao();

            console.log(local);

            infos = `appName: ${navigator.appName}  <br>`;
            infos += `appVersion : ${navigator.appVersion}<br>`;
            infos += `appCodeName : ${navigator.appCodeName}<br>`;
            infos += `product : ${navigator.product}<br>`;
            infos += `userAgent : ${navigator.userAgent}<br>`;
            infos += `platform : ${navigator.platform}<br>`;
            infos += `language : ${navigator.language}<br>`;
            infos += `online? ${navigator.onLine ? 'Sim' : 'Não'}<br>`;
            infos += `Java habilitado? ${navigator.javaEnabled() ? 'Sim' : 'Não'}<br>`;
            infos += `Conexão: ${navigator.connection}<br>`;
            infos += `Memória: ${navigator.deviceMemory} GB <br>`;
            infos += `Geolocalização: ${local}<br>`;

            document.getElementById('infos').innerHTML = infos;
        }
    </script>
</head>

<body style="text-align: center;">
    <button onclick="infosNavegador()">Informações Navegador</button>
    <div id="infos"></div>
</body>

</html>
```

UNOESC