Gabarito dos exercícios

A seguir encontra-se uma sugestão de solução para os exercícios propostos.

Exercício 1

Crie uma função que recebe três parâmetros retorna uma `Promise` .

1. Caso algum dos parâmetros recebidos não seja um número, rejeite a Promise com o motivo `"Informe apenas números"` .
2. Caso todos os parâmetros sejam numéricos, some os dois primeiros e multiplique o resultado pelo terceiro (`(a + b) * c`).
3. Caso o resultado seja menor que 50, rejeite a Promise com o motivo `"Valor muito baixo"`
4. Caso o resultado seja maior que 50, resolva a Promise com o valor obtido.

Resolução

1. Criar uma pasta para o projeto e, nela um arquivo `index.js` contendo a base da função:

Copiar

```
function doMath(a, b, c) {
  return new Promise((resolve, reject) => {});
}
```

2. Adicionar validação para garantir que todos os valores são numéricos:

Copiar

```
// function doMath(a, b, c) {
  // return new Promise((resolve, reject) => {
    /* Caso o tipo de algum parâmetro não seja `number`, rejeitamos
a Promise */
    if (typeof a !== 'number' || typeof b !== 'number' || typeof c
!== 'number')
      reject('Informe apenas números');
  //});
//}
```

3. Validar se o resultado é maior que 50 e resolver ou rejeitar a Promise

Copiar

```
// function doMath(a, b, c) {
//   return new Promise((resolve, reject) => {
//     /* Caso o tipo de algum parâmetro não seja `number`, rejeitamos a Promise */
//     if (typeof a !== 'number' || typeof b !== 'number' || typeof c !== 'number')
//       reject('Informe apenas números');

      const result = (a + b) * c;

      if (result < 50) {
        return reject('Valor muito baixo');
      }

      resolve(result);
//   });
// }
```

4. Chamar a função nas condições de entrada e verificar sua saída

Copiar

```
// function doMath(a, b, c) {
//   return new Promise((resolve, reject) => {
//     /* Caso o tipo de algum parâmetro não seja `number`, rejeitamos a Promise */
//     if (typeof a !== 'number' || typeof b !== 'number' || typeof c !== 'number')
//       reject('Informe apenas números');

//     const result = (a + b) * c;

//     if (result < 50) {
//       return reject('Valor muito baixo');
//     }

//     resolve(result);
//   });
// }

doMath(10, 10, 10)
```

```
  .then(resolve => console.log(resolve))
  .catch(error => console.log(error))

doMath(1, 1, 'a')
  .then(resolve => console.log(resolve))
  .catch(error => console.log(error))

doMath(1, 1, 1)
  .then(resolve => console.log(resolve))
  .catch(error => console.log(error))
```

Exercício 2

Escreva código para consumir a função construída no exercício anterior

1. Gere um número aleatório de 1 a 100 para cada parâmetro que a função recebe. Para gerar um número aleatório, utilize o seguinte trecho de código: `Math.floor(Math.random() * 100 + 1)` .
2. Chame a função do exercício anterior, passando os três números aleatórios como parâmetros.
3. Utilize `then` e `catch` para manipular a Promise retornada pela função:
    1. Caso a Promise seja rejeitada, escreva na tela o motivo da rejeição.
    2. Caso a Promise seja resolvida, escreva na tela o resultado do cálculo.

Resolução

1. No mesmo arquivo, criar as funções para gerar números aleatórios e chamar `doMath` .

Copiar

```
// function doMath(a, b, c) {
// ...
// }

function getRandomNumber() {
  return Math.floor(Math.random() * 100 + 1);
}

function callDoMath() {
  /* Criamos um novo array de 3 posições
```

```
   * e utilizamos o `map` para gerar um número aleatório
   * para cada posição do Array
   */
  const randomNumbers = Array.from({ length: 3
}).map(getRandomNumber);
}
```

2. Realizar chamada e lidar com o resultado

Copiar

```
// function doMath(a, b, c) {
// ...
// }

// function getRandomNumber() {
//   return Math.floor(Math.random() * 100 + 1);
// }

// function callDoMath() {
//   /* Criamos um novo array de 3 posições e utilizamos o `map`
para gerar um número aleatório para cada posição do Array */
//   const randomNumbers = Array.from({ length: 3
}).map(getRandomNumber);

    doMath(...randomNumbers)
      .then(result => console.log(result))
      .catch(err => console.error(err.message))
// }
```

Exercício 3

Reescreva o código do exercício anterior para que utilize `async/await` .

- Lembre-se: a palavra chave `await` só pode ser utilizada dentro de funções `async` .

Resolução

1. Transformar `callDoMath` em uma `async function`

Copiar

```
async function callDoMath() {
```

```
//   /* Criamos um novo array de 3 posições e utilizamos o `map`
para gerar um número aleatório para cada posição do Array */
//   const randomNumbers = Array.from({ length: 3
}).map(getRandomNumber);

//   doMath(...randomNumbers)
//     .then((result) => console.log(result))
//     .catch((err) => console.error(err.message));
// }
```

2. Substituir o tratamento de sucesso (`then`) pela palavra chave `await`

Copiar

```
// async function callDoMath() {
//   /* Criamos um novo array de 3 posições e utilizamos o `map`
para gerar um número aleatório para cada posição do Array */
//   const randomNumbers = Array.from({ length: 3
}).map(getRandomNumber);

    const result = await doMath(...randomNumbers)
//     .catch((err) => console.error(err.message));

//   console.log(result);
// }
```

3. Substituir o tratamento de erro (`catch`) pela estrutura `try ... catch`

Copiar

```
// async function callDoMath() {
//   /* Criamos um novo array de 3 posições e utilizamos o `map`
para gerar um número aleatório para cada posição do Array */
//   const randomNumbers = Array.from({ length: 3
}).map(getRandomNumber);

    try {
//     const result = await doMath(...randomNumbers);
//     console.log(result);
    } catch (err) {
      console.error(err);
    }
// }
```

Exercício 4

Realize o download deste arquivo e salve-o como `simpsons.json` . Utilize o arquivo baixado para realizar os requisitos abaixo.

- Você pode utilizar `then` e `catch` , `async/await` ou uma mistura dos dois para escrever seu código. Procure não utilizar callbacks.

1. Crie uma função que leia todos os dados do arquivo e imprima cada personagem no formato `id - Nome` . Por exemplo: `1 - Homer Simpson` .
2. Crie uma função que receba o `id` de uma personagem como parâmetro e retorne uma `Promise` que é resolvida com os dados da personagem que possui o `id` informado. Caso não haja uma personagem com o `id` informado, rejeite a Promise com o motivo "id não encontrado".
3. Crie uma função que altere o arquivo `simpsons.json` retirando os personagens com `id` 10 e 6.
4. Crie uma função que leia o arquivo `simpsons.json` e crie um novo arquivo, chamado `simpsonFamily.json` , contendo as personagens com `id` de 1 a 4.
5. Crie uma função que adicione ao arquivo `simpsonFamily.json` o personagem `Nelson Muntz` .
6. Crie uma função que substitua o personagem `Nelson Muntz` pela personagem `Maggie Simpson` no arquivo `simpsonFamily.json` .

Resolução

1. Crie uma função que leia todos os dados do arquivo e imprima cada personagem no formato <code>id - Nome</code>. Por exemplo: <code>1 - Homer Simpson</code>.
2. Importar o módulo `fs/promises` e realizar a leitura do arquivo

Copiar

```
const fs = require('fs').promises;

fs.readFile('./simpsons.json', 'utf-8')
  .then((fileContent) => {});
```

2. Converter o conteúdo do arquivo de JSON para um Array utilizando `JSON.parse`

Copiar

```
// const fs = require('fs').promises;

// fs.readFile('./simpsons.json', 'utf-8')
//   .then((fileContent) => {
```

```
      return JSON.parse(fileContent);
//   });
```

3. Mapear cada objeto do Array para uma string no formato correto

Copiar

```
// const fs = require('fs').promises;

// fs.readFile('./simpsons.json', 'utf-8')
//   .then((fileContent) => {
//     return JSON.parse(fileContent);
//   })
    .then((simpsons) => {
      return simpsons.map(({ id, name }) => `${id} - ${name}`);
    });
```

4. Exibir as strings na tela

Copiar

```
// const fs = require('fs').promises;

// fs.readFile('./simpsons.json', 'utf-8')
//   .then((fileContent) => {
//     return JSON.parse(fileContent);
//   })
//   .then((simpsons) => {
//     return simpsons.map(({ id, name }) => `${id} - ${name}`);
//   })
    .then((strings) => {
      strings.forEach((string) => console.log(string));
    });
```

2. Crie uma função que receba o `id` de uma personagem como parâmetro e retorne uma `Promise` que é resolvida com os dados da personagem que possui o `id` informado. Caso não haja uma personagem com o `id` informado, rejeite a Promise com o motivo "id não encontrado".
3. Importar o módulo `fs/promises` , criar a função e realizar a leitura do arquivo e a conversão do JSON em objeto

Copiar

```
const fs = require('fs').promises;

async function getSimpsonById(id) {
```

```
  const simpsons = await fs
    .readFile('./simpsons.json', 'utf-8')
    .then((fileContent) => JSON.parse(fileContent));
}
```

2. Realizar a busca pelo Simpson desejado e, caso não encontrar, disparar um erro

Copiar

```
// const fs = require('fs').promises;

// async function getSimpsonById(id) {
//   const simpsons = await fs
//     .readFile('./simpsons.json', 'utf-8')
//     .then((fileContent) => JSON.parse(fileContent));

    const chosenSimpson = simpsons.find((simpson) => simpson.id === id);

    if (!chosenSimpson) {
      /* A palavra-chave `throw` dispara um erro que deve ser tratado por quem chamou nossa função.
       * Em funções `async`, utilizar `throw` faz com que a Promise seja rejeitada,
       * tendo como motivo o que passarmos para o `throw`.
       * Ou seja, a linha abaixo rejeita a Promise da nossa função com o motivo 'id não encontrado'
       */
      throw new Error('id não encontrado');
    }
// }
```

3. Caso a personagem exista, resolver a Promise com suas informações

Copiar

```
// const fs = require('fs').promises;

// async function getSimpsonById(id) {
//   const simpsons = await fs
//     .readFile('./simpsons.json', 'utf-8')
//     .then((fileContent) => JSON.parse(fileContent));
```

```
//   const chosenSimpson = simpsons.find((simpson) => simpson.id === id);

//   if (!chosenSimpson) {
//     /* A palavra-chave `throw` dispara um erro que deve ser tratado por quem chamou nossa função.
//      * Em funções `async`, utilizar `throw` faz com que a Promise seja rejeitada,
//      * tendo como motivo o que passarmos para o `throw`.
//      * Ou seja, a linha abaixo rejeita a Promise da nossa função com o motivo 'id não encontrado'
//      */
//     throw new Error('id não encontrado');
//   }

    /* Da mesma forma que `throw` aciona o fluxo de erro e rejeita a Promise,
     * `return` aciona o fluxo de sucesso e resolve a Promise.
     * Sendo assim, a linha abaixo é equivalente a chamar `resolve(chosenSimpson)`
     * dentro do executor de uma Promise.
     */
    return chosenSimpson;
// }
```

3. Crie uma função que altere o arquivo `simpsons.json` retirando os personagens com `id` 10 e 6.
4. Importar o módulo `fs/promises` , criar a função e realizar a leitura do arquivo e o parsing do JSON

Copiar

```
const fs = require('fs').promises;

async function filterSimpsons() {
  const simpsons = await fs
    .readFile('./simpsons.json', 'utf-8')
    .then((fileContent) => JSON.parse(fileContent));
}
```

2. Filtrar o array para remover as personagens que devem ser removidas

Copiar

```
// const fs = require('fs').promises;
```

```
// async function filterSimpsons() {
//   const simpsons = await fs
//     .readFile('./simpsons.json', 'utf-8')
//     .then((fileContent) => JSON.parse(fileContent));

    const newArray = simpsons.filter(simpson => simpson.id !== '10'
&& simpson.id !== '6');
// }
```

3. Escrever no arquivo o novo array filtrado

Copiar

```
// const fs = require('fs').promises;

// async function filterSimpsons() {
//   const simpsons = await fs
//     .readFile('./simpsons.json', 'utf-8')
//     .then((fileContent) => JSON.parse(fileContent));

//   const newArray = simpsons.filter(simpson => simpson.id !== '10'
&& simpson.id !== '6');

    await fs.writeFile('./simpsons.json',
JSON.stringify(newArray));
// }
```

4. Crie uma função que leia o arquivo `simpsons.json` e crie um novo arquivo, chamado `simpsonFamily.json` , contendo as personagens com `id` de 1 a 4.
5. Importar o módulo `fs/promises` , criar a função e realizar a leitura do arquivo e o parsing do JSON

Copiar

```
const fs = require('fs').promises;

async function createSimpsonsFamily() {
  const simpsons = await fs
    .readFile('./simpsons.json', 'utf-8')
    .then((fileContent) => JSON.parse(fileContent));
}
```

2. Criar um novo array apenas com os membros da família

Copiar

```
// const fs = require('fs').promises;

// async function createSimpsonsFamily() {
//   const simpsons = await fs
//     .readFile('./simpsons.json', 'utf-8')
//     .then((fileContent) => JSON.parse(fileContent));

    const simpsonsFamily = simpsons.filter(simpson => [1, 2, 3,
4].includes(simpson.id));
// }
```

3. Escrever o novo arquivo no disco

Copiar

```
// const fs = require('fs').promises;

// async function createSimpsonsFamily() {
//   const simpsons = await fs
//     .readFile('./simpsons.json', 'utf-8')
//     .then((fileContent) => JSON.parse(fileContent));

//   const simpsonsFamily = simpsons.filter(simpson => [1, 2, 3,
4].includes(simpson.id));

    await fs.writeFile('./simpsonsFamily.json',
JSON.stringify(simpsonsFamily));
// }
```

5. Crie uma função que adicione ao arquivo `simpsonFamily.json` o personagem `Nelson Muntz` .
6. Importar o módulo `fs/promises` , criar a função e realizar a leitura do arquivo e o parsing do JSON

Copiar

```
const fs = require('fs').promises;

async function addNelsonToFamily() {
  const simpsonsFamily = await fs
    .readFile('./simpsonsFamily.json', 'utf-8')
    .then((fileContent) => JSON.parse(fileContent));
}
```

2. Adicionar uma nova pessoa ao array de `simpsonsFamily`

Copiar

```
// const fs = require('fs').promises;

// async function addNelsonToFamily() {
//   const simpsonsFamily = await fs
//     .readFile('./simpsonsFamily.json', 'utf-8')
//     .then((fileContent) => JSON.parse(fileContent));

    simpsonsFamily.push({ "id": "8","name": "Nelson Muntz" });
// }
```

3. Escrever o novo conteúdo do arquivo

Copiar

```
// const fs = require('fs').promises;

// async function addNelsonToFamily() {
//   const simpsonsFamily = await fs
//     .readFile('./simpsonsFamily.json', 'utf-8')
//     .then((fileContent) => JSON.parse(fileContent));

//   simpsonsFamily.push({ "id": "8","name": "Nelson Muntz" });

    await fs.writeFile('./simpsonsFamily.json', simpsonsFamily);
// }
```

6. Crie uma função que substitua o personagem `Nelson Muntz` pela personagem `Maggie Simpson` no arquivo `simpsonFamily.json` .

Copiar

```
// Importamos o módulo de promises do fs
const fs = require('fs').promises;

function replaceNelson () {
  // Realizamos a leitura do arquivo
  return fs.readFile('./simpsonsFamily.json', 'utf-8')
    // Interpretamos o conteúdo como JSON
    .then((fileContent) => JSON.parse(fileContent))
    // Filtramos o array para remover o personagem Nelson
    .then((simpsons) => simpsons.filter((simpson) => simpson.id !==
'8'))
    // Criamos um novo Array contendo a personagem Maggie
    .then((simpsonsWithoutNelson) => simpsonsWithoutNelson
      .concat([{ id: '8', name: 'Maggie Simpson' }]))
```

```
    // Escrevemos o novo array no arquivo
    .then((simpsonsWithMaggie) =>
      fs.writeFile('./simpsonsFamily.json',
JSON.stringify(simpsonsWithMaggie)));
}
```

Exercício 5

Crie uma função que lê e escreve vários arquivos ao mesmo tempo.

1. Utilize o `Promise.all` para manipular vários arquivos ao mesmo tempo.
2. Dado o seguinte array de strings: `['Finalmente', 'estou', 'usando', 'Promise.all', '!!!']` Faça com que sua função crie um arquivo contendo cada string, sendo o nome de cada arquivo igual a `file<index + 1>.txt` . Por exemplo, para a string "Finalmente", o nome do arquivo é `file1.txt` .
3. Programe sua função para que ela faça a leitura de todos os arquivos criados no item anterior, armazene essa informação e escreva em um arquivo chamado `fileAll.txt` . O conteúdo do arquivo `fileAll.txt` deverá ser `Finalmente estou usando Promise.all !!!` .

Resolução

1. Importar o módulo fs e criar a função com o Array de strings

Copiar

```
const fs = require('fs').promises;

async function arrayToFile() {
  const strings = ['Finalmente', 'estou', 'usando', 'Promise.all',
'!!!']
}
```

2. Utilizar a função `map` para criar um Array de Promises, cada um criando um arquivo

Copiar

```
// const fs = require('fs').promises;

// async function arrayToFile() {
```

```
//   const strings = ['Finalmente', 'estou', 'usando',
'Promise.all', '!!!']

    const createFilePromises = strings.map((string, index) =>
      fs.writeFile(`./file${index + 1}.txt`, string)
    );
// }
```

3. Utilizar `Promise.all` para aguardar a escrita de todos os arquivos

Copiar

```
// const fs = require('fs').promises;

// async function arrayToFile() {
//   const strings = ['Finalmente', 'estou', 'usando',
'Promise.all', '!!!']

//   const createFilePromises = strings.map((string, index) =>
//     fs.writeFile(`./file${index + 1}.txt`, string)
//   );

    await Promise.all(createFilePromises);
// }
```

4. Realizar a leitura dos arquivos criados

Copiar

```
// const fs = require('fs').promises;

// async function arrayToFile() {
//   const strings = ['Finalmente', 'estou', 'usando',
'Promise.all', '!!!'];

//   const createFilePromises = strings.map((string, index) =>
//     fs.writeFile(`./file${index + 1}.txt`, string)
//   );

//   await Promise.all(createFilePromises);

    const fileNames = [
      'file1.txt',
      'file2.txt',
      'file3.txt',
      'file4.txt',
```

```
      'file5.txt',
    ];

    const fileContents = await Promise.all(
      fileNames.map((fileName) => fs.readFile(fileName, 'utf-8'))
    );
// }
```

5. Concatenar o conteúdo dos arquivos e criar o arquivo novo

Copiar

```
// const fs = require('fs').promises;

// async function arrayToFile() {
//   const strings = ['Finalmente', 'estou', 'usando',
'Promise.all', '!!!'];

//   const createFilePromises = strings.map((string, index) =>
//     fs.writeFile(`./file${index + 1}.txt`, string)
//   );

//   await Promise.all(createFilePromises);

//   const fileNames = [
//     'file1.txt',
//     'file2.txt',
//     'file3.txt',
//     'file4.txt',
//     'file5.txt',
//   ];

//   const fileContents = await Promise.all(
//     fileNames.map((fileName) => fs.readFile(fileName, 'utf-8'))
//   );

    const newFileContent = fileContents.join(' ');

    await fs.writeFile('./fileAll.txt', newFileContent);
// }
```

Bônus

Exercício 1

Crie um script que mostre na tela o conteúdo de um arquivo escolhido pelo usuário:

1. Pergunte à pessoa usuária qual arquivo ela deseja ler.
2. Leia o arquivo indicado.
3. Caso o arquivo não exista, exiba na tela "Arquivo inexistente" e encerre a execução do script.
4. Caso o arquivo exista, escreva seu conteúdo na tela.

Resolução

1. Importamos os módulos que vamos utilizar: `fs/promises` e `readline`

Copiar
```
const fs = require('fs').promises;
const readline = require('readline');
```

2. Para facilitar a solicitação de input, criamos uma função que utiliza o `readline.question` , mas retorna uma Promise

Copiar
```
const fs = require('fs').promises;
const readline = require('readline');

function question(message) {
  // Realizamos o uso do readline conforme mostrado na documentação.
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
  });

  return new Promise((resolve) => {
    // No entanto, ao abrirmos a pergunta,
    // fazemos isso dentro de uma Promise.
    rl.question(message, (answer) => {
      rl.close();

      // Dessa forma, quando obtivermos a resposta,
      // podemos resolver nossa Promise com essa resposta.
      // Assim, quem chama nossa função não precisa
      // se preocupar com callbacks, e pode obter a resposta
      // através da Promise que retornamos.
```

```
      resolve(answer);
    });
  });
}
```

3. Criamos a função que será responsável pelo fluxo todo. Vamos chamá-la de `start` :

Copiar

```
// const fs = require('fs').promises;
// const readline = require('readline');

// function question(message) {
//   // Realizamos o uso do readline conforme mostrado na documentação.
//   const rl = readline.createInterface({
//     input: process.stdin,
//     output: process.stdout
//   });

//   return new Promise((resolve) => {
//     // No entanto, ao abrirmos a pergunta,
//     // fazemos isso dentro de uma Promise.
//     rl.question(message, (answer) => {
//       rl.close();

//       // Dessa forma, quando obtivermos a resposta,
//       // podemos resolver nossa Promise com essa resposta.
//       // Assim, quem chama nossa função não precisa
//       // se preocupar com callbacks, e pode obter a resposta
//       // através da Promise que retornamos.
//       resolve(answer);
//     });
//   });
// }

async function start() {
  // Como nossa função `question` retorna uma Promise,
  // podemos utilzar `await` para obter a resposta.
  const fileName = await question('Digite o caminho do arquivo que deseja ler: ');

  try {
```

```
    // Tentamos realizar a leitura do arquivo
    const fileContent = await readFile(fileName, 'utf-8');
    // E exibir seu resultado na tela
    console.log(fileContent);
  } catch (err) {
    // Caso um erro aconteça, exibimos a mensagem de erro na tela.
    console.log('Arquivo inexistente');
  }
}

start();
```

Exercício 2

Crie um script que substitua uma palavra por outra em um arquivo escolhido pela pessoa usuária:

1. Pergunte à pessoa usuária qual arquivo deseja utilizar.
2. Leia o arquivo.
3. Caso o arquivo não exista, exiba um erro na tela e encerre a execução do script.
4. Caso o arquivo exista, solicite a palavra a ser substituída.
5. Solicite a nova palavra, que substituirá a palavra anterior.
6. Imprima na tela o conteúdo do arquivo com as palavras já substituídas.
7. Pergunte o nome do arquivo de destino.
8. Salve o novo arquivo no caminho de destino.

Dica: Utilize a classe RegExp do JS para substituir todas as ocorrências da palavra com `replace(new RegExp(palavra, 'g'), novaPalavra)` .

Resolução

1. Como no exercício anterior, começamos importando os módulos necessários e criando a função `question` .

Copiar

```
const fs = require('fs').promises;
const readline = require('readline');

function question(message) {
  // Realizamos o uso do readline conforme mostrado na documentação.
```

```
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
  });

  return new Promise((resolve) => {
    // No entanto, ao abrirmos a pergunta,
    // fazemos isso dentro de uma Promise.
    rl.question(message, (answer) => {
      rl.close();

      // Dessa forma, quando obtivermos a resposta,
      // podemos resolver nossa Promise com essa resposta.
      // Assim, quem chama nossa função não precisa
      // se preocupar com callbacks, e pode obter a resposta
      // através da Promise que retornamos.
      resolve(answer);
    });
  });
}
```

2. Criamos a função `start` , responsável pelo fluxo, e perguntamos o nome do arquivo a ser lido

Copiar

```
const fs = require('fs').promises;
const readline = require('readline');

function question(message) {
  // Realizamos o uso do readline conforme mostrado na documentação.
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
  });

  return new Promise((resolve) => {
    // No entanto, ao abrirmos a pergunta,
    // fazemos isso dentro de uma Promise.
    rl.question(message, (answer) => {
      rl.close();

      // Dessa forma, quando obtivermos a resposta,
      // podemos resolver nossa Promise com essa resposta.
```

```
      // Assim, quem chama nossa função não precisa
      // se preocupar com callbacks, e pode obter a resposta
      // através da Promise que retornamos.
      resolve(answer);
    });
  });
}

async function start() {
  const fileName = await question('Arquivo a ser lido: ');
}

start();
```

3. Realizamos a leitura do arquivo, utilizando um `catch` para tratar erros.

Copiar

```
// const fs = require('fs').promises;
// const readline = require('readline');

// function question(message) {
//   // Realizamos o uso do readline conforme mostrado na
documentação.
//   const rl = readline.createInterface({
//     input: process.stdin,
//     output: process.stdout
//   });

//   return new Promise((resolve) => {
//     // No entanto, ao abrirmos a pergunta,
//     // fazemos isso dentro de uma Promise.
//     rl.question(message, (answer) => {
//       rl.close();

//       // Dessa forma, quando obtivermos a resposta,
//       // podemos resolver nossa Promise com essa resposta.
//       // Assim, quem chama nossa função não precisa
//       // se preocupar com callbacks, e pode obter a resposta
//       // através da Promise que retornamos.
//       resolve(answer);
//     });
//   });
```

```
// }

// async function start() {
//   const fileName = await question('Arquivo a ser lido: ');

    const originalContent = await fs.readFile(fileName, 'utf-8')
      // Caso aconteça um erro ao ler o arquivo
      .catch(err => {
        // Mostramos o erro na tela
        console.error(`Erro ao ler o arquivo: ${err}`);
        // E retornamos `false`.
        // O valor retornado aqui do catch é o valor que será
armazenado
        // na variável `originalContent`.
        return false;
      })

    // Se `originalContent` estiver vazia ou contiver um valor
falso,
    // quer dizer que ocorreu um erro na leitura do arquivo e não
devemos prosseguir.
    // Utilizamos o `return` para encerrar a execução
    if (!originalContent) return;
// }

// start();
```

4. Perguntamos quais palavras deverão ser substituídas, realizamos a substituição e exibimos o resultado na tela

Copiar

```
// const fs = require('fs').promises;
// const readline = require('readline');

// function question(message) {
//   // Realizamos o uso do readline conforme mostrado na
documentação.
//   const rl = readline.createInterface({
//     input: process.stdin,
//     output: process.stdout
//   });

//   return new Promise((resolve) => {
```

```
//     // No entanto, ao abrirmos a pergunta,
//     // fazemos isso dentro de uma Promise.
//     rl.question(message, (answer) => {
//       rl.close();

//       // Dessa forma, quando obtivermos a resposta,
//       // podemos resolver nossa Promise com essa resposta.
//       // Assim, quem chama nossa função não precisa
//       // se preocupar com callbacks, e pode obter a resposta
//       // através da Promise que retornamos.
//       resolve(answer);
//     });
//   });
// }

// async function start() {
//   const fileName = await question('Arquivo a ser lido: ');

//   const originalContent = await fs.readFile(fileName, 'utf-8')
//     // Caso aconteça um erro ao ler o arquivo
//     .catch(err => {
//       // Mostramos o erro na tela
//       console.error(`Erro ao ler o arquivo: ${err}`);
//       // E retornamos `false`.
//       // O valor retornado aqui do catch é o valor que será
armazenado
//       // na variável `originalContent`.
//       return false;
//     })

//   // Se `originalContent` estiver vazia ou contiver um valor
falso,
//   // quer dizer que ocorreu um erro na leitura do arquivo e não
devemos prosseguir.
//   // Utilizamos o `return` para encerrar a execução
//   if (!originalContent) return;

    const oldWord = await question('Qual palavra deseja substituir?
');
    const newWord = await question('E qual palavra deve ficar em
seu lugar? ');
```

```
    const newContent = originalContent.replace(new RegExp(oldWord,
'g'), newWord);

    console.log('Resultado da substituição: ');
    console.log(newContent);
// }

// start();
```

5. Por último, perguntamos o nome do arquivo onde salvar o resultado e escrevemos no disco.

Copiar

```
// const fs = require('fs').promises;
// const readline = require('readline');

// function question(message) {
//   // Realizamos o uso do readline conforme mostrado na
documentação.
//   const rl = readline.createInterface({
//     input: process.stdin,
//     output: process.stdout
//   });

//   return new Promise((resolve) => {
//     // No entanto, ao abrirmos a pergunta,
//     // fazemos isso dentro de uma Promise.
//     rl.question(message, (answer) => {
//       rl.close();

//       // Dessa forma, quando obtivermos a resposta,
//       // podemos resolver nossa Promise com essa resposta.
//       // Assim, quem chama nossa função não precisa
//       // se preocupar com callbacks, e pode obter a resposta
//       // através da Promise que retornamos.
//       resolve(answer);
//     });
//   });
// }

// async function start() {
//   const fileName = await question('Arquivo a ser lido: ');
```

```
//   const originalContent = await fs.readFile(fileName, 'utf-8')
//     // Caso aconteça um erro ao ler o arquivo
//     .catch(err => {
//       // Mostramos o erro na tela
//       console.error(`Erro ao ler o arquivo: ${err}`);
//       // E retornamos `false`.
//       // O valor retornado aqui do catch é o valor que será
armazenado
//       // na variável `originalContent`.
//       return false;
//     })

//   // Se `originalContent` estiver vazia ou contiver um valor
falso,
//   // quer dizer que ocorreu um erro na leitura do arquivo e não
devemos prosseguir.
//   // Utilizamos o `return` para encerrar a execução
//   if (!originalContent) return;

//   const oldWord = await question('Qual palavra deseja substituir?
');
//   const newWord = await question('E qual palavra deve ficar em
seu lugar? ');

//   const newContent = originalContent.replace(new RegExp(oldWord,
'g'), newWord);

//   console.log('Resultado da substituição: ');
//   console.log(newContent);

    const destinationPath = await question('Onde deseja salvar o
resultado? ');
    await fs.writeFile(destinationPath, newContent);
// }

// start();
```