PERGUNTAS TEÓRICAS TESTE TÉCNICO - Zerum Tecnologia

1. Questões de JavaScript

A diferença entre `var`, `let` e `const`.

- **`var`:**
  - Declara uma variável com escopo **global** ou **de função**.
  - Pode ser redeclarada e reatribuída.
  - Não respeita o escopo de bloco.
  - Exemplo:

```
var x = 10;
if (true) {
  var x = 20; // Redefine a mesma variável
  console.log(x); // 20
}
console.log(x); // 20
```

- **`let`:**
  - Declara uma variável com escopo de **bloco**.
  - Pode ser reatribuída, mas não redeclarada no mesmo escopo.
  - Exemplo:

```
let y = 10;
if (true) {
  let y = 20; // Nova variável no escopo do bloco
  console.log(y); // 20
}
console.log(y); // 10
```

- **`const`:**
  - Declara uma constante com escopo de **bloco**.
  - Não pode ser reatribuída, mas objetos e arrays podem ser alterados.
  - Exemplo:

```
const z = 10;
// z = 20; // Erro: Não é possível reatribuir
const obj = { a: 1 };
obj.a = 2; // Permitido
console.log(obj.a); // 2
```

O que é `this` em JavaScript, e como seu valor é determinado?

- **`this`** refere-se ao **contexto** de execução.
- Seu valor é determinado pelo modo como a função é chamada:

  - **Função normal:** `this` refere-se ao objeto global (`window` no navegador, `global` no Node.js).
  - Exemplo:

```
function normalFunction() {
  console.log(this); // window ou global
}
```

  - **Método de objeto:** `this` refere-se ao objeto que contém o método.
  - Exemplo:

```
const obj = {
  value: 42,
  showThis() {
console.log(this.value); // 42
  },
};
obj.showThis();
```

- **Função de seta:** Não tem seu próprio `this`; herda do contexto onde foi definida.
- Exemplo:

```
const obj = {
  value: 42,
  arrow: () => console.log(this), // Herda de onde
foi criada (objeto global)
};
obj.arrow();
```

Como funciona o modelo de eventos em JavaScript? Explique o conceito de "event bubbling" e "event delegation".

- **Modelo de eventos:**
  - Eventos são acionados em resposta a interações do usuário (ex.: cliques).
  - JavaScript usa um sistema baseado em captura, bubbling e propagação.

- **Event bubbling:**
  - O evento é disparado no elemento alvo e sobe na hierarquia do DOM

```
document.getElementById('child').addEventListener('c
lick', () => {
  console.log('Child clicked!');
});
document.getElementById('parent').addEventListener('
click', () => {
  console.log('Parent clicked!');
});
// Output: "Child clicked!", depois "Parent
clicked!"
```

Event delegation:

- Usa o bubbling para gerenciar eventos em elementos filhos por meio de um pai comum.
- Exemplo:

```
document.getElementById('parent').addEventListener('
click', (event) => {
  if (event.target.tagName === 'BUTTON') {
console.log('Button clicked!');
  }
});
```

O que são Promises em JavaScript? Dê um exemplo básico de uso.

- Promises são objetos que representam a eventual **finalização** (ou falha) de uma operação assíncrona.
- **Estados:** pendente, resolvida, rejeitada.
- Exemplo:

```
const promise = new Promise((resolve, reject) => {
  const success = true;
  if (success) resolve('Operação bem-sucedida');
  else reject('Erro na operação');
});

promise
  .then((result) => console.log(result))
  .catch((error) => console.error(error));
```

O que é a função `fetch` e como ela funciona?

- **`fetch`** é usada para realizar solicitações HTTP assíncronas.
- Retorna uma **Promise**.
- Exemplo:

```
fetch('https://api.example.com/data')
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error('Erro:', error));
```

2. Questões de Testes Unitários

O que são testes unitários e qual sua importância no desenvolvimento?

- Testes unitários verificam funcionalidades isoladas de um sistema (ex.: funções ou métodos).
- **Importância:**
  - Detectar erros precocemente.
  - Garantir que mudanças no código não quebrem funcionalidades existentes.
  - Facilitar refatorações.

Explique a diferença entre testes unitários, testes de integração e testes de ponta a ponta.

- **Unitários:** Testam partes isoladas (ex.: uma função).
- **Integração:** Testam interações entre módulos.
- **Ponta a ponta:** Simulam o fluxo completo do usuário (UI, backend, etc.).

Como você faria para testar um componente que depende de uma API externa?

- Usar **mocks** para simular a API e testar o comportamento do componente:
- Exemplo:

```
jest.mock('./api', () => ({
  fetchData: jest.fn(() => Promise.resolve({ data: 'mock
data' })),
}));
```

3. Questões de Front-End Geral

Qual a diferença entre flexbox e grid no CSS? Quando usar cada um?

- **Flexbox:** Alinha elementos em uma única direção (linha ou coluna).
  - Ideal para layouts **unidimensionais**.

- **Grid:** Cria layouts bidimensionais (linhas e colunas).
    - Ideal para **layouts completos**.

Explique o conceito de "responsividade" no design web e como garantir isso.

- **Responsividade:** Capacidade de um site se adaptar a diferentes tamanhos de tela.
- **Como garantir:**
    - Uso de media queries (`@media`).
    - Layouts fluidos (ex.: `%`, `vh`, `vw`).
    - Uso de unidades responsivas como `rem` e `em`.

O que é o DOM e qual sua importância no desenvolvimento web?

- **DOM (Document Object Model):**
    - Representação hierárquica de um documento HTML.
- **Importância:**
    - Permite manipulação dinâmica de conteúdo e estrutura via JavaScript

```
document.getElementById('demo').innerText = 'Texto
alterado!';
```