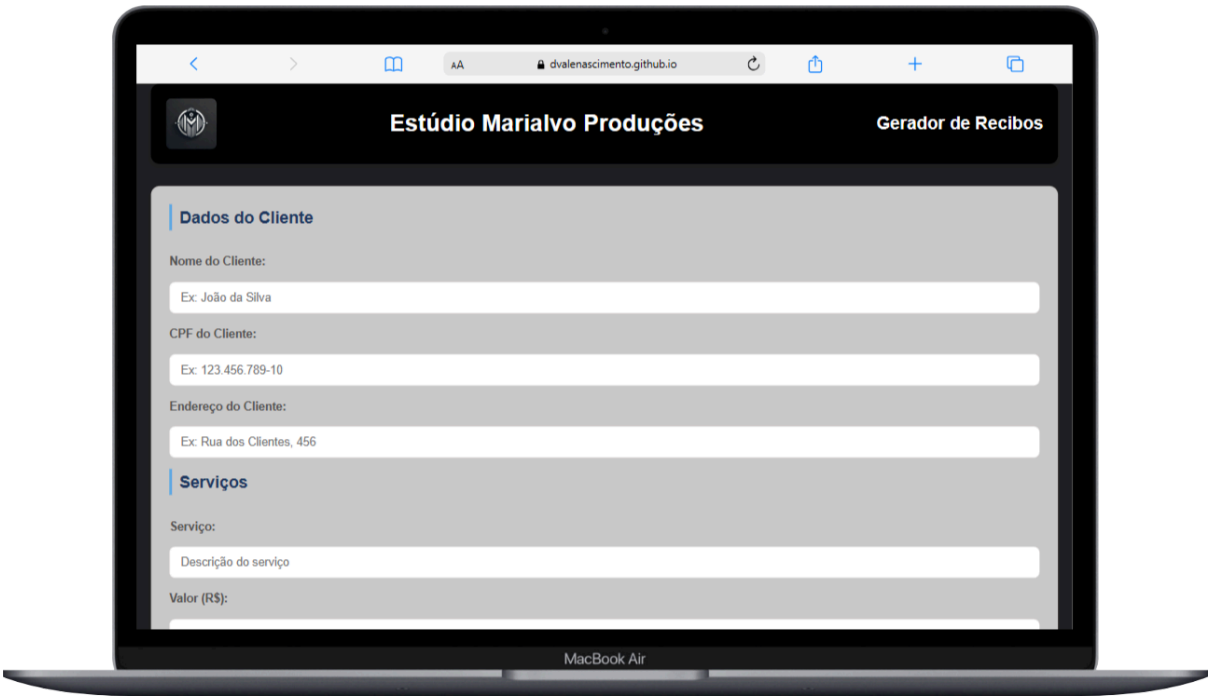
Análise de Código JavaScript

Projeto: Gerador de Recibos Estúdio M Produções



Sumário

Código do Projeto

```
const { jsPDF } = window.jspdf;

let servicos = [];

// Adicionar serviços à lista
document.getElementById("addServico").addEventListener("click", () => {
  const servico = document.getElementById("servico").value.trim();
  const valor = parseFloat(document.getElementById("valor").value);

  if (servico && !isNaN(valor)) {
    servicos.push({ descricao: servico, valor: valor.toFixed(2) });
    atualizarListaServicos();
    document.getElementById("servico").value = "";
    document.getElementById("valor").value = "";
  } else {
    alert("Preencha os campos do serviço corretamente!");
  }
});

// Atualizar a lista de serviços
function atualizarListaServicos() {
  const lista = document.getElementById("servicos");
  lista.innerHTML = ""; // Limpar a lista existente
  servicos.forEach((item) => {
    const li = document.createElement("li");
    li.textContent = `${item.descricao} - R$ ${item.valor}`;
    lista.appendChild(li);
  });
}

// Gerar PDF
document.getElementById("gerarRecibo").addEventListener("click", () => {
  const nomeCliente = document.getElementById("nomeCliente").value.trim();
  const cpfCliente = document.getElementById("cpfCliente").value.trim();
  const enderecoCliente = document.getElementById("enderecoCliente").value.trim();
  const formaPagamento = document.getElementById("formaPagamento").value;

  if (!nomeCliente || !cpfCliente || !enderecoCliente || servicos.length === 0 ||
!formaPagamento) {
    alert("Preencha todos os campos obrigatórios e adicione pelo menos um serviço.");
    return;
  }
```

```
  const doc = new jsPDF();
  const total = servicos.reduce((sum, item) => sum + parseFloat(item.valor), 0).toFixed(2);

  // Caminho do logo
  const logoPath = "assets/logo.png";
  const img = new Image();
  img.src = logoPath;

  img.onload = () => {
    doc.addImage(img, "PNG", 10, 10, 50, 20); // Adicionar logo ao PDF
    gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total);
  };

  img.onerror = () => {
    console.warn("Logo não encontrada, gerando recibo sem logo.");
    gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total);
  };
});

// Gerar o PDF completo
function gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento,
total) {
  doc.setFontSize(18);
  doc.text("RECIBO", 105, 40, { align: "center" });

  doc.setFontSize(12);
  doc.text("Emitente: Estúdio Marialvo Produções", 20, 60);
  doc.text("CNPJ: 12.345.678/0001-99", 20, 70);
  doc.text("Endereço: Rua Garcia Redondo, 23, Compensa, Manaus/AM", 20, 80);

  doc.text(`Recebido de: ${nomeCliente}`, 20, 100);
  doc.text(`CPF: ${cpfCliente}`, 20, 110);
  doc.text(`Endereço: ${enderecoCliente}`, 20, 120);

  let y = 140;
  servicos.forEach((item) => {
    doc.text(`${item.descricao} - R$ ${item.valor}`, 20, y);
    y += 10;
  });

  doc.text(`Valor Total: R$ ${total}`, 20, y + 10);
  doc.text(`Forma de Pagamento: ${formaPagamento}`, 20, y + 20);
  doc.text(`Data: ${new Date().toLocaleDateString()}`, 20, y + 40);
  doc.text("Assinatura do Emitente: _____", 20, y + 60);
  doc.text("Assinatura do Cliente: _____", 20, y + 70);

  doc.save("recibo.pdf");
}
```

Apresentação

Este código implementa um sistema de geração de recibos personalizados em PDF utilizando a biblioteca jsPDF. Ele possui funcionalidades para adicionar serviços, listar os serviços adicionados e gerar um recibo em PDF com informações detalhadas. Vamos analisar cada parte do código em detalhes:

---

1. Importação da Biblioteca jsPDF

```
const { jsPDF } = window.jspdf;
```

- A biblioteca jsPDF é importada da variável `window.jspdf`.
- `jsPDF` é usado para criar e manipular arquivos PDF.

---

2. Declaração da Lista de Serviços

```
let servicos = [];
```

- Um array chamado `servicos` armazena os objetos contendo as descrições e valores dos serviços adicionados pelo usuário.

---

3. Adicionar Serviços à Lista

```
document.getElementById("addServico").addEventListener("click", () => {
  const servico = document.getElementById("servico").value.trim();
  const valor = parseFloat(document.getElementById("valor").value);

  if (servico && !isNaN(valor)) {
    servicos.push({ descricao: servico, valor: valor.toFixed(2) });
    atualizarListaServicos();
    document.getElementById("servico").value = "";
    document.getElementById("valor").value = "";
  } else {
    alert("Preencha os campos do serviço corretamente!");
  }
});
```

- **Evento de clique no botão "Adicionar Serviço"**:
  - Captura o valor do campo `servico` e o transforma em texto com `trim()` para remover espaços desnecessários.
  - Captura o valor do campo `valor` e o converte para um número decimal (`parseFloat`).
  - **Validação**: Verifica se o nome do serviço não está vazio e se o valor é numérico.
  - Se válido, adiciona o serviço ao array `servicos` como um objeto `{ descricao, valor }`, onde o valor é fixado em duas casas decimais (`toFixed(2)`).
  - Chama a função `atualizarListaServicos()` para exibir a lista atualizada.
  - Limpa os campos de entrada (`servico` e `valor`).

---

4. Atualizar a Lista de Serviços

```
function atualizarListaServicos() {
  const lista = document.getElementById("servicos");
  lista.innerHTML = ""; // Limpa a lista atual
  servicos.forEach((item) => {
    const li = document.createElement("li");
    li.textContent = `${item.descricao} - R$ ${item.valor}`;
    lista.appendChild(li);
  });
}
```

- **Função `atualizarListaServicos`**:
  - Obtém o elemento da lista (`<ul>` ou `<ol>`) onde os serviços serão exibidos.
  - Limpa a lista (`innerHTML = ""`).
  - Para cada item no array `servicos`, cria um elemento `<li>` e define seu conteúdo como `descricao` e `valor` do serviço.
  - Adiciona o elemento `<li>` à lista exibida.

---

5. Gerar o PDF

```
document.getElementById("gerarRecibo").addEventListener("click", () => {
  const nomeCliente = document.getElementById("nomeCliente").value.trim();
  const cpfCliente = document.getElementById("cpfCliente").value.trim();
  const enderecoCliente = document.getElementById("enderecoCliente").value.trim();
  const formaPagamento = document.getElementById("formaPagamento").value;
```

```
  if (!nomeCliente || !cpfCliente || !enderecoCliente || servicos.length === 0 ||
!formaPagamento) {
    alert("Preencha todos os campos obrigatórios e adicione pelo menos um serviço.");
    return;
  }

  const doc = new jsPDF();
  const total = servicos.reduce((sum, item) => sum + parseFloat(item.valor), 0).toFixed(2);

  const logoPath = "assets/logo.png";
  const img = new Image();
  img.src = logoPath;

  img.onload = () => {
    doc.addImage(img, "PNG", 10, 10, 50, 20);
    gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total);
  };

  img.onerror = () => {
    console.warn("Logo não encontrada, gerando recibo sem logo.");
    gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total);
  };
});
```

- **Evento de clique no botão "Gerar Recibo"**:
  - Captura os valores dos campos obrigatórios: nome do cliente, CPF, endereço, forma de pagamento.
  - **Validação**: Exibe um alerta se os campos obrigatórios não forem preenchidos ou se nenhum serviço tiver sido adicionado.
  - Cria uma instância de `jsPDF` para gerar o arquivo PDF.
  - Calcula o total dos serviços com `reduce` (soma os valores de cada serviço no array `servicos`).
  - Carrega a imagem do logo (`assets/logo.png`) usando o objeto `Image()`:
    - **Se carregada com sucesso**: Adiciona a imagem ao PDF e chama `gerarReciboPDF` para preencher os dados.
    - **Se falhar**: Emite um aviso no console e gera o recibo sem o logo.

---

6. Função `gerarReciboPDF`

```
function gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento,
total) {
  doc.setFontSize(18);
  doc.text("RECIBO", 105, 40, { align: "center" });

  doc.setFontSize(12);
```

```
  doc.text("Emitente: Estúdio Marialvo Produções", 20, 60);
  doc.text("CNPJ: 12.345.678/0001-99", 20, 70);
  doc.text("Endereço: Rua Garcia Redondo, 23, Compensa, Manaus/AM", 20, 80);

  doc.text(`Recebido de: ${nomeCliente}`, 20, 100);
  doc.text(`CPF: ${cpfCliente}`, 20, 110);
  doc.text(`Endereço: ${enderecoCliente}`, 20, 120);

  let y = 140;
  servicos.forEach((item) => {
    doc.text(`${item.descricao} - R$ ${item.valor}`, 20, y);
    y += 10;
  });

  doc.text(`Valor Total: R$ ${total}`, 20, y + 10);
  doc.text(`Forma de Pagamento: ${formaPagamento}`, 20, y + 20);
  doc.text(`Data: ${new Date().toLocaleDateString()}`, 20, y + 40);
  doc.text("Assinatura do Emitente: _____", 20, y + 60);
  doc.text("Assinatura do Cliente: _____", 20, y + 70);

  doc.save("recibo.pdf");
}
```

- Esta função preenche o conteúdo do PDF com os dados do cliente, os serviços adicionados e o total.
- **Destaques**:
  - Define tamanhos de fonte e alinhamento do texto.
  - Lista os serviços em ordem, ajustando a posição vertical (`y`) dinamicamente.
  - Adiciona assinaturas e salva o arquivo com o nome `recibo.pdf`.

---

7. Fluxo Geral

1. O usuário adiciona serviços usando o botão "Adicionar Serviço".
2. A lista de serviços é exibida na página.
3. Após preencher os campos obrigatórios, o usuário clica no botão "Gerar Recibo".
4. O sistema valida os dados, gera o PDF e permite o download.

Possíveis Melhorias

1. **Validações mais robustas** para CPF, valores e outros campos.
2. Adicionar suporte a diferentes idiomas e moedas.
3. Opção de envio automático do PDF por e-mail.

1. Importação da Biblioteca jsPDF

A linha de código:

const { jsPDF } = window.jspdf;

realiza a importação da biblioteca jsPDF, que é amplamente utilizada para criar e manipular documentos PDF diretamente no navegador. Vamos detalhar o que essa linha faz:

---

a. O que é `window.jspdf`?

- **`window`**: É o objeto global no navegador que contém todas as variáveis, funções e objetos definidos globalmente.
- **`jspdf`**: Ao incluir a biblioteca jsPDF no projeto (por exemplo, com uma tag `<script>` no HTML ou instalando via um gerenciador de pacotes como npm), ela adiciona a propriedade `jspdf` ao objeto `window`.
    - Isso significa que a biblioteca fica acessível globalmente como `window.jspdf`.

---

b. O que é `{ jsPDF }`?

- A biblioteca jsPDF expõe várias funcionalidades. A principal delas é a classe ou função `jsPDF`, usada para criar documentos PDF.
- A sintaxe `{ jsPDF }` utiliza **desestruturação de objetos**:
    - O objeto `window.jspdf` contém a propriedade `jsPDF`.
    - A desestruturação extrai a propriedade `jsPDF` e a armazena na constante `jsPDF`.
    - Agora, você pode usá-la diretamente no código sem a necessidade de escrever `window.jspdf.jsPDF`.

---

c. Por que usar `jsPDF`?

A função ou classe `jsPDF` é a base para criar e manipular PDFs. Após importá-la, você pode:

Criar uma instância de um documento PDF vazio:

```
const doc = new jsPDF();
```

1. Isso cria um documento PDF em branco.

Adicionar conteúdo ao PDF: Após criar uma instância, você pode adicionar texto, imagens, formas, tabelas, etc. Exemplo:

```
doc.text("Hello, world!", 10, 10); // Adiciona texto no ponto (10, 10)
```

2.

Salvar o PDF no computador do usuário: Depois de construir o documento PDF, ele pode ser salvo:

```
doc.save("meu_documento.pdf");
```

3.

---

d. Fluxo Geral da Importação

1. A biblioteca jsPDF é carregada no navegador (normalmente com `<script src="jspdf.min.js"></script>` ou via bundlers como Webpack).
2. O jsPDF se torna disponível globalmente como `window.jspdf`.
3. A linha `const { jsPDF } = window.jspdf;` faz a desestruturação e permite que `jsPDF` seja usado diretamente no código.

---

e. Vantagens da Biblioteca jsPDF

1. **Independência do Servidor**:
    - Toda a geração do PDF ocorre no navegador, sem a necessidade de enviar os dados para o servidor.
    - Útil para projetos onde os dados não podem sair do lado do cliente.
2. **Criação Direta no Navegador**:
    - Permite gerar PDFs instantaneamente sem plugins externos (como o Adobe Acrobat).
    - Exemplo: criar notas fiscais, recibos, relatórios, etc.
3. **Recursos Avançados**:
    - Adicionar texto, imagens, gráficos, tabelas e até mesmo código QR aos PDFs.
    - Controlar layout, tamanhos de página, margens e muito mais.
4. **Compatibilidade**:
    - Funciona com todos os navegadores modernos.

Resumo

A linha:

const { jsPDF } = window.jspdf;

1. Acessa a biblioteca jsPDF através do objeto global `window.jspdf`.
2. Desestrutura a função `jsPDF` e a torna acessível diretamente no código.
3. Permite criar instâncias para gerar e manipular documentos PDF no navegador.

Exemplo completo de uso:

```
const { jsPDF } = window.jspdf;

const doc = new jsPDF(); // Cria um documento PDF em branco
doc.text("Meu primeiro PDF com jsPDF!", 10, 10); // Adiciona texto
doc.save("exemplo.pdf"); // Salva o arquivo como "exemplo.pdf"
```

Assim, a biblioteca oferece uma maneira simples e eficiente de criar PDFs diretamente no navegador.

2. Declaração da Lista de Serviços

```
let servicos = [];
```

A linha acima cria uma **variável chamada `servicos`**, que será utilizada para armazenar os serviços que o usuário adiciona ao sistema. Esta variável é um **array vazio inicialmente**, pois no início do programa nenhum serviço foi inserido.

Explicação detalhada:

1. **O que é um array?**
   - Um **array** é uma estrutura de dados em JavaScript que armazena uma lista de valores. Esses valores podem ser de qualquer tipo, como números, strings, objetos ou até outros arrays.
   - Neste caso, o array `servicos` será usado para armazenar **objetos** que representam cada serviço.

---

2. **Por que usar um array?**
    - O array permite que o sistema armazene vários serviços de forma dinâmica, ou seja, os serviços podem ser adicionados ou removidos conforme necessário.
    - Cada serviço é armazenado como um **objeto**, que contém as seguintes propriedades:
        - `descricao`: Uma string que descreve o serviço (exemplo: "Manutenção de Computadores").
        - `valor`: Um número que representa o preço do serviço (exemplo: 150.00).

---

3. **Estado inicial do array:**
    - Quando o programa é carregado, o array `servicos` é vazio (`[]`), já que ainda não existem serviços cadastrados pelo usuário.
    - À medida que o usuário adiciona serviços, o array será preenchido dinamicamente com objetos.

---

4. **Exemplo de uso do array:**
    - Suponha que o usuário adicione dois serviços:
        - Serviço 1: Descrição = "Formatação de Computador", Valor = 100.00
        - Serviço 2: Descrição = "Troca de Tela", Valor = 200.00

O array `servicos` terá o seguinte conteúdo:

```
[
 { descricao: "Formatação de Computador", valor: "100.00" },
 { descricao: "Troca de Tela", valor: "200.00" }
]
```

    -

---

5. **Como o array é preenchido?**

O array é atualizado pela lógica contida no evento **"Adicionar Serviço"**, que foi configurado com o seguinte código:

```
document.getElementById("addServico").addEventListener("click", () => {
 const servico = document.getElementById("servico").value.trim();
 const valor = parseFloat(document.getElementById("valor").value);

 if (servico && !isNaN(valor)) {
  servicos.push({ descricao: servico, valor: valor.toFixed(2) });
 }
});
```

-
    - **`servicos.push()`**: Adiciona um novo objeto ao array `servicos`.
    - O objeto contém as informações fornecidas pelo usuário nos campos de descrição e valor.

---

6. **Função do array no sistema:**
    - O array `servicos` é fundamental para:
        - **Armazenar os serviços adicionados** pelo usuário.
        - **Atualizar a interface da lista de serviços** exibida na página.
        - **Calcular o total de valores** quando o recibo for gerado.
        - **Preencher os dados do recibo** com as informações detalhadas de cada serviço.

---

7. **Resumo:**
    - `servicos` é um array que começa vazio.
    - Ele é usado para armazenar os serviços adicionados pelo usuário.
    - Cada item do array é um objeto com propriedades `descricao` e `valor`.
    - Este array permite que o sistema gerencie dinamicamente a lista de serviços e utilize essas informações na geração do recibo.

3. Adicionar Serviços à Lista

A funcionalidade "Adicionar Serviços à Lista" permite que o usuário insira serviços e seus respectivos valores em campos de entrada e os adicione a uma lista dinâmica. Vamos analisar o código detalhadamente:

O Código

```
document.getElementById("addServico").addEventListener("click", () => {
  const servico = document.getElementById("servico").value.trim();
  const valor = parseFloat(document.getElementById("valor").value);

  if (servico && !isNaN(valor)) {
    servicos.push({ descricao: servico, valor: valor.toFixed(2) });
    atualizarListaServicos();
    document.getElementById("servico").value = "";
    document.getElementById("valor").value = "";
  } else {
```

```
    alert("Preencha os campos do serviço corretamente!");
  }
});
```

Detalhamento Passo a Passo

1. Captura do Evento de Clique

```
document.getElementById("addServico").addEventListener("click", () => { ... });
```

- **O que acontece?**
  - Adiciona um ouvinte de eventos ao botão identificado pelo ID `addServico`.
  - Esse ouvinte monitora o evento de clique no botão.
  - Quando o botão é clicado, a função dentro do `addEventListener` é executada.

---

2. Captura e Processamento dos Campos de Entrada

```
const servico = document.getElementById("servico").value.trim();
const valor = parseFloat(document.getElementById("valor").value);
```

- **Captura do Serviço (`servico`)**
  - O valor do campo de texto com o ID `servico` é capturado usando `document.getElementById("servico").value`.
  - O método `.trim()` é usado para remover espaços em branco no início e no final da string, garantindo que entradas como `" Limpeza "` sejam tratadas corretamente como `"Limpeza"`.
- **Captura do Valor (`valor`)**
  - O valor do campo de entrada numérico com o ID `valor` é capturado e convertido para um número decimal usando `parseFloat`.
  - Isso garante que o valor seja um número válido para cálculos posteriores.

---

3. Validação dos Campos

```
if (servico && !isNaN(valor)) {
  // Lógica para adicionar o serviço à lista
} else {
  alert("Preencha os campos do serviço corretamente!");
}
```

- **Condições da Validação:**
    - `servico`: Verifica se o campo de serviço não está vazio. Strings vazias (`""`) são avaliadas como `false` em um contexto booleano.
    - `!isNaN(valor)`: Garante que o valor seja numérico (ou seja, `NaN` é negado). Isso impede entradas inválidas como `"abc"` no campo de valor.
- **Se a validação falhar:**
    - O código exibe um alerta informando o usuário que os campos precisam ser preenchidos corretamente.

---

4. Adição do Serviço à Lista
servicos.push({ descricao: servico, valor: valor.toFixed(2) });

- **O que é feito?**
    - O serviço válido é adicionado ao array `servicos` como um objeto.
    - O objeto contém duas propriedades:
        - `descricao`: Nome do serviço capturado no campo.
        - `valor`: O valor numérico capturado, formatado para **duas casas decimais** com `.toFixed(2)`. Isso garante que o valor seja exibido corretamente como, por exemplo, `12.50` em vez de `12.5`.
- **Por que usar o array `servicos`?**
    - O array `servicos` funciona como uma estrutura de dados que armazena todos os serviços adicionados. Ele pode ser usado para gerar uma lista dinâmica ou realizar cálculos, como somar os valores.

---

5. Atualização da Lista
atualizarListaServicos();

- Após adicionar o serviço ao array, a função `atualizarListaServicos()` é chamada para exibir a lista atualizada de serviços na página.
- A lógica de `atualizarListaServicos()` é separada em outra função para manter o código modular.

---

6. Limpeza dos Campos de Entrada
document.getElementById("servico").value = "";

```
document.getElementById("valor").value = "";
```

- Após adicionar o serviço à lista, os campos de entrada são limpos:
  - O campo `servico` recebe uma string vazia (`""`).
  - O campo `valor` também é resetado para uma string vazia (`""`).
- Isso melhora a experiência do usuário, pois deixa os campos prontos para receber novas informações sem que o usuário precise apagá-los manualmente.

---

7. Caso de Erro

```
else {
  alert("Preencha os campos do serviço corretamente!");
}
```

- Se a validação falhar (ou seja, o nome do serviço estiver vazio ou o valor não for numérico), um **alerta** é exibido para o usuário com a mensagem de erro.

Resumo do Fluxo

1. O usuário clica no botão "Adicionar Serviço".
2. O código captura os valores dos campos `servico` e `valor`.
3. Valida os valores:
   - O nome do serviço não pode estar vazio.
   - O valor deve ser numérico.
4. Se válidos:
   - O serviço é adicionado ao array `servicos` como um objeto `{ descricao, valor }`.
   - A lista de serviços exibida na página é atualizada.
   - Os campos de entrada são limpos.
5. Se inválidos:
   - Um alerta é exibido pedindo ao usuário para corrigir os campos.

Exemplo Prático

HTML

```
<input id="servico" type="text" placeholder="Nome do Serviço">
<input id="valor" type="number" placeholder="Valor (R$)">
<button id="addServico">Adicionar Serviço</button>
<ul id="servicos"></ul>
```

Saída

1. O usuário digita:
    - Serviço: `"Limpeza"`
    - Valor: `150.75`
2. Clica em "Adicionar Serviço".

O serviço aparece na lista:
Limpeza - R$ 150.75

3.
4. Os campos de entrada são limpos.

Benefícios

- **Interatividade**: Permite que o usuário adicione e visualize serviços dinamicamente.
- **Validação Simples**: Evita erros comuns como campos vazios ou valores inválidos.
- **Flexibilidade**: Os dados armazenados no array podem ser usados para cálculos ou geração de PDFs (como no restante do código).

4. Atualizar a Lista de Serviços

A função `atualizarListaServicos` é responsável por exibir dinamicamente a lista de serviços adicionados pelo usuário. Vamos detalhar cada passo do código:

O Código

```
function atualizarListaServicos() {
  const lista = document.getElementById("servicos");
  lista.innerHTML = ""; // Limpa a lista atual
  servicos.forEach((item) => {
    const li = document.createElement("li");
    li.textContent = `${item.descricao} - R$ ${item.valor}`;
    lista.appendChild(li);
  });
}
```

Detalhamento Passo a Passo

1. Seleciona o Elemento da Lista
const lista = document.getElementById("servicos");

- **O que acontece?**
  - O método `document.getElementById` é usado para capturar o elemento HTML que contém a lista de serviços.
  - Este elemento é identificado pelo ID `"servicos"`.
  - Geralmente, esse elemento é uma tag `<ul>` (lista não ordenada) ou `<ol>` (lista ordenada), onde cada serviço será exibido como um item (`<li>`).

---

2. Limpa a Lista Atual
lista.innerHTML = "";

- **Por que limpar a lista?**
  - Antes de atualizar a lista, ela é limpa para evitar que os itens antigos sejam exibidos junto com os novos.
  - A propriedade `innerHTML` define ou obtém o conteúdo HTML de um elemento.
    - Atribuir uma string vazia (`""`) remove todos os elementos filhos existentes no elemento da lista.

---

3. Itera pelo Array `servicos`
servicos.forEach((item) => { ... });

- **O que é `servicos`?**
  - `servicos` é um array que armazena todos os serviços adicionados pelo usuário. Cada elemento é um objeto com as propriedades:
    - `descricao`: O nome ou descrição do serviço.
    - `valor`: O valor do serviço (em formato numérico, com duas casas decimais).
- **Iteração com `forEach`**
  - O método `forEach` percorre cada item do array `servicos`.
  - Para cada elemento do array (chamado de `item` no exemplo), o código executa a lógica dentro da função de callback.

---

4. Cria um Novo Item da Lista

const li = document.createElement("li");

- **Criação do elemento `<li>`**
  - A função `document.createElement("li")` cria um novo elemento HTML `<li>` (item de lista).
  - Este elemento representará um serviço na lista exibida.

---

5. Define o Conteúdo do Item
li.textContent = `${item.descricao} - R$ ${item.valor}`;

- **Propriedade `textContent`**
  - Define o conteúdo de texto do elemento `<li>`.

O texto é formatado como:
`${item.descricao} - R$ ${item.valor}`

  -
    - `item.descricao`: Nome ou descrição do serviço (ex.: "Limpeza").
    - `item.valor`: Valor do serviço formatado com duas casas decimais (ex.: "150.00").

Por exemplo, se o serviço for:
{ descricao: "Limpeza", valor: "150.00" }
O texto será:
Limpeza - R$ 150.00

  -

---

6. Adiciona o Item à Lista
lista.appendChild(li);

- **`appendChild`**

  - Adiciona o elemento `<li>` recém-criado como filho do elemento de lista (capturado no início como `lista`).
  - Isso exibe o item na interface do usuário.

- **Fluxo**

- Para cada serviço no array `servicos`, um novo `<li>` é criado e adicionado à lista HTML.
- Quando o loop termina, todos os serviços do array são exibidos.

Resumo do Fluxo

1. A função `atualizarListaServicos` é chamada após adicionar um novo serviço ao array `servicos`.
2. Captura o elemento da lista (`<ul>` ou `<ol>`) onde os serviços serão exibidos.
3. Limpa o conteúdo existente na lista, evitando duplicações.
4. Para cada serviço no array `servicos`:
   - Cria um novo item de lista (`<li>`).
   - Define o texto do item como "descrição - R$ valor".
   - Adiciona o item à lista exibida na página.

Exemplo Prático

HTML

```
<ul id="servicos">
  <!-- Os itens da lista serão exibidos aqui -->
</ul>
```

Array `servicos`

```
const servicos = [
  { descricao: "Limpeza", valor: "150.00" },
  { descricao: "Reparos", valor: "200.50" },
];
```

Saída Esperada

Se a função `atualizarListaServicos` for chamada, o HTML resultante será:

```
<ul id="servicos">
  <li>Limpeza - R$ 150.00</li>
  <li>Reparos - R$ 200.50</li>
</ul>
```

Por que esta função é importante?

1. **Atualização Dinâmica**
    - Permite exibir os serviços em tempo real conforme o usuário os adiciona.
    - Não é necessário recarregar a página para ver as mudanças.
2. **Separação de Lógica**
    - A lógica de atualização da interface é encapsulada em uma função, tornando o código mais modular e reutilizável.
3. **Integração**
    - Trabalha diretamente com o array `servicos`, garantindo que a interface exiba sempre a lista mais recente de serviços.
4. **Experiência do Usuário**
    - Fornece feedback visual ao usuário, mostrando os serviços adicionados.

Vantagens

- Simplicidade: Fácil de entender e modificar.
- Escalabilidade: Pode ser adaptada para incluir botões de edição ou exclusão de serviços no futuro.
- Reutilização: Pode ser usada sempre que o array `servicos` for atualizado, garantindo sincronização entre os dados e a interface.

5. Gerar o PDF

Essa funcionalidade permite gerar um recibo em formato PDF com os dados fornecidos pelo usuário (nome, CPF, endereço, forma de pagamento, e serviços adicionados). Abaixo, vamos detalhar cada etapa do código:

O Código

```
document.getElementById("gerarRecibo").addEventListener("click", () => {
  const nomeCliente = document.getElementById("nomeCliente").value.trim();
  const cpfCliente = document.getElementById("cpfCliente").value.trim();
  const enderecoCliente = document.getElementById("enderecoCliente").value.trim();
  const formaPagamento = document.getElementById("formaPagamento").value;

  if (!nomeCliente || !cpfCliente || !enderecoCliente || servicos.length === 0 ||
!formaPagamento) {
    alert("Preencha todos os campos obrigatórios e adicione pelo menos um serviço.");
    return;
  }

  const doc = new jsPDF();
  const total = servicos.reduce((sum, item) => sum + parseFloat(item.valor), 0).toFixed(2);

  const logoPath = "assets/logo.png";
  const img = new Image();
  img.src = logoPath;

  img.onload = () => {
    doc.addImage(img, "PNG", 10, 10, 50, 20);
    gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total);
  };

  img.onerror = () => {
    console.warn("Logo não encontrada, gerando recibo sem logo.");
    gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total);
  };
});
```

Detalhamento Passo a Passo

1. Evento de Clique no Botão

```
document.getElementById("gerarRecibo").addEventListener("click", () => { ... });
```

- **O que acontece?**

- O método `addEventListener` adiciona um ouvinte de evento ao botão com o ID `"gerarRecibo"`.
- Quando o botão é clicado, a função especificada no segundo parâmetro é executada.

---

2. Captura dos Dados do Formulário

```
const nomeCliente = document.getElementById("nomeCliente").value.trim();
const cpfCliente = document.getElementById("cpfCliente").value.trim();
const enderecoCliente = document.getElementById("enderecoCliente").value.trim();
const formaPagamento = document.getElementById("formaPagamento").value;
```

- **Campos Capturados:**
  - **`nomeCliente`**: Nome do cliente.
  - **`cpfCliente`**: CPF do cliente.
  - **`enderecoCliente`**: Endereço do cliente.
  - **`formaPagamento`**: Método de pagamento escolhido.
- **Uso de `.trim()`**
  - Remove espaços em branco no início e no fim do texto para evitar problemas de validação.

---

3. Validação dos Dados

```
if (!nomeCliente || !cpfCliente || !enderecoCliente || servicos.length === 0 ||
!formaPagamento) {
  alert("Preencha todos os campos obrigatórios e adicione pelo menos um serviço.");
  return;
}
```

- **Condições de Validação:**
  - Verifica se:
    - O nome, CPF, endereço e forma de pagamento foram preenchidos.
    - Pelo menos um serviço foi adicionado à lista (`servicos.length === 0`).
- **Ação:**
  - Se algum campo estiver vazio ou nenhum serviço for adicionado, exibe um alerta ao usuário e interrompe a execução da função com `return`.

---

4. Criação do PDF
const doc = new jsPDF();

- **Instância de `jsPDF`:**
  - A classe `jsPDF` é usada para criar e manipular arquivos PDF.
  - `doc` é a instância do PDF que será gerado.

---

5. Cálculo do Total
const total = servicos.reduce((sum, item) => sum + parseFloat(item.valor), 0).toFixed(2);

- **O que acontece?**
  - O método `reduce` soma o valor de todos os serviços no array `servicos`.
  - `parseFloat(item.valor)` converte o valor do serviço (armazenado como string) para um número decimal.
  - `.toFixed(2)` formata o total com duas casas decimais.

Exemplo de cálculo:

```
 const servicos = [
  { descricao: "Limpeza", valor: "100.00" },
  { descricao: "Reparos", valor: "200.50" }
];

const total = servicos.reduce((sum, item) => sum + parseFloat(item.valor), 0).toFixed(2);
console.log(total); // "300.50"
```

-

---

6. Carregamento da Logo
const logoPath = "assets/logo.png";
const img = new Image();
img.src = logoPath;

- **Imagem da Logo:**
  - `logoPath` é o caminho relativo do arquivo de imagem da logo.
  - `img` é uma instância da classe `Image`, que permite carregar imagens no JavaScript.
  - `img.src` define o caminho da imagem que será carregada.

---

7. Manipulação do Carregamento da Imagem
Se a imagem carregar com sucesso:

```
 img.onload = () => {
  doc.addImage(img, "PNG", 10, 10, 50, 20);
  gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total);
};
```

-
    - `img.onload` é acionado quando a imagem é carregada.
    - O método `doc.addImage` insere a imagem no PDF.
        - `"PNG"`: Formato da imagem.
        - `10, 10`: Coordenadas X e Y (posição da imagem no PDF).
        - `50, 20`: Largura e altura da imagem.
    - Após adicionar a imagem, a função `gerarReciboPDF` é chamada para preencher os dados do recibo.

Se a imagem não for carregada:

```
 img.onerror = () => {
  console.warn("Logo não encontrada, gerando recibo sem logo.");
  gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total);
};
```

-
    - `img.onerror` é acionado quando ocorre um erro ao carregar a imagem.
    - Exibe uma mensagem de aviso no console.
    - A função `gerarReciboPDF` é chamada para gerar o recibo sem a logo.

Resumo da Função

1. **Evento de Clique:** O código é executado quando o botão "Gerar Recibo" é clicado.
2. **Captura e Validação:** Os dados do formulário são capturados e validados.
3. **Criação do PDF:** Uma nova instância de `jsPDF` é criada.
4. **Cálculo do Total:** O total dos serviços é calculado somando os valores no array `servicos`.
5. **Carregamento da Logo:**
    - Se a logo for carregada com sucesso, é adicionada ao PDF.
    - Se ocorrer um erro, o recibo é gerado sem a logo.
6. **Geração do Recibo:** A função `gerarReciboPDF` é chamada para preencher os dados e finalizar o recibo.

Por que isso é importante?

- **Automatização:**
    - Gera recibos automaticamente, eliminando a necessidade de preenchimento manual.
- **Profissionalismo:**
    - Inclui informações formatadas e (se possível) a logo da empresa, conferindo credibilidade ao documento.
- **Flexibilidade:**
    - Permite gerar recibos personalizados com os dados fornecidos pelo usuário.

6. Função `gerarReciboPDF`

A função `gerarReciboPDF` é responsável por preencher o conteúdo do recibo em PDF com os dados fornecidos (cliente, serviços e total) e salvá-lo como um arquivo chamado `recibo.pdf`. Abaixo está o detalhamento completo de como ela funciona:

O Código

```
function gerarReciboPDF(doc, nomeCliente, cpfCliente, enderecoCliente, formaPagamento, total) {
  doc.setFontSize(18);
  doc.text("RECIBO", 105, 40, { align: "center" });

  doc.setFontSize(12);
  doc.text("Emitente: Estúdio Marialvo Produções", 20, 60);
  doc.text("CNPJ: 12.345.678/0001-99", 20, 70);
  doc.text("Endereço: Rua Garcia Redondo, 23, Compensa, Manaus/AM", 20, 80);

  doc.text(`Recebido de: ${nomeCliente}`, 20, 100);
  doc.text(`CPF: ${cpfCliente}`, 20, 110);
  doc.text(`Endereço: ${enderecoCliente}`, 20, 120);

  let y = 140;
  servicos.forEach((item) => {
    doc.text(`${item.descricao} - R$ ${item.valor}`, 20, y);
    y += 10;
```

```
  });

  doc.text(`Valor Total: R$ ${total}`, 20, y + 10);
  doc.text(`Forma de Pagamento: ${formaPagamento}`, 20, y + 20);
  doc.text(`Data: ${new Date().toLocaleDateString()}`, 20, y + 40);
  doc.text("Assinatura do Emitente: _____", 20, y + 60);
  doc.text("Assinatura do Cliente: _____", 20, y + 70);

  doc.save("recibo.pdf");
}
```

Detalhamento Passo a Passo

1. Título do Recibo

```
doc.setFontSize(18);
doc.text("RECIBO", 105, 40, { align: "center" });
```

- **Tamanho da Fonte:** A fonte é configurada com tamanho 18 para destacar o título.
- **Texto:** A palavra "RECIBO" é adicionada ao PDF.
- **Posição:**
  - `x = 105`**:** Posição horizontal no centro do documento (ajustada para alinhamento central).
  - `y = 40`**:** Posição vertical.
  - `align: "center"`**:** Alinha o texto horizontalmente no centro.

---

2. Dados do Emitente

```
doc.setFontSize(12);
doc.text("Emitente: Estúdio Marialvo Produções", 20, 60);
doc.text("CNPJ: 12.345.678/0001-99", 20, 70);
doc.text("Endereço: Rua Garcia Redondo, 23, Compensa, Manaus/AM", 20, 80);
```

- **Tamanho da Fonte:** Configurado como 12 para o texto do corpo.
- **Informações Incluídas:**
  - Nome do emitente.
  - CNPJ fictício do emitente.
  - Endereço do emitente.
- **Posição:**
  - `x = 20`**:** Margem esquerda.
  - `y`**:** Começa em 60 e aumenta de 10 em 10 para as linhas subsequentes.

---

3. Dados do Cliente

```
doc.text(`Recebido de: ${nomeCliente}`, 20, 100);
doc.text(`CPF: ${cpfCliente}`, 20, 110);
doc.text(`Endereço: ${enderecoCliente}`, 20, 120);
```

- **Dados Capturados:**
  - Nome do cliente.
  - CPF do cliente.
  - Endereço do cliente.
- **Interpolação de Strings:**
  - Os valores das variáveis (`nomeCliente`, `cpfCliente`, `enderecoCliente`) são inseridos diretamente no texto usando **template literals** (`` `...` ``).
- **Posição:**
  - Começa na coordenada vertical `y = 100` e aumenta em incrementos de 10.

---

4. Lista de Serviços

```
let y = 140;
servicos.forEach((item) => {
  doc.text(`${item.descricao} - R$ ${item.valor}`, 20, y);
  y += 10;
});
```

- **Iteração com `forEach`:**
  - Para cada serviço no array `servicos`, adiciona uma linha ao PDF com o nome do serviço e seu valor.
  - Exemplo: `"Limpeza - R$ 100.00"`.
- **Posição Dinâmica:**
  - A variável `y` é usada para ajustar dinamicamente a posição vertical de cada serviço.
  - Inicializa em `140` e aumenta em incrementos de `10` para evitar sobreposição entre as linhas.

---

5. Valor Total e Forma de Pagamento

```
doc.text(`Valor Total: R$ ${total}`, 20, y + 10);
doc.text(`Forma de Pagamento: ${formaPagamento}`, 20, y + 20);
doc.text(`Data: ${new Date().toLocaleDateString()}`, 20, y + 40);
```

- **Informações Incluídas:**
  - Valor total dos serviços, formatado com duas casas decimais.

    - Forma de pagamento escolhida pelo cliente.
    - Data atual, obtida com `new Date().toLocaleDateString()`.
- **Posição Dinâmica:**
    - Começa logo abaixo da lista de serviços, ajustando o valor de `y` dinamicamente:
        - Total: `y + 10`.
        - Forma de pagamento: `y + 20`.
        - Data: `y + 40`.

---

6. Assinaturas

```
doc.text("Assinatura do Emitente: _____", 20, y + 60);
doc.text("Assinatura do Cliente: _____", 20, y + 70);
```

- **Linhas de Assinatura:**
    - Espaços para as assinaturas do emitente e do cliente.
- **Posição:**
    - Emitente: `y + 60`.
    - Cliente: `y + 70`.

---

7. Salvar o PDF

```
doc.save("recibo.pdf");
```

- **Método `save`:**
    - Salva o arquivo PDF gerado no dispositivo do usuário.
    - O arquivo é salvo com o nome `"recibo.pdf"`.

Resumo da Função

1. Define o título "RECIBO" centralizado no PDF.
2. Adiciona os dados do emitente (empresa).
3. Adiciona os dados do cliente (nome, CPF, endereço).
4. Lista os serviços prestados com seus valores.
5. Calcula e exibe o valor total e a forma de pagamento.
6. Adiciona a data e linhas para as assinaturas do emitente e do cliente.
7. Salva o recibo gerado como um arquivo PDF.

Por que é importante?

- **Automatização:**
    - Gera recibos de forma rápida e personalizada com os dados fornecidos.
- **Profissionalismo:**
    - Inclui detalhes completos, como informações do cliente e do emitente, além de assinaturas, conferindo legitimidade ao documento.
- **Facilidade de Uso:**
    - O cliente ou usuário final pode salvar e compartilhar o recibo gerado diretamente no formato PDF.