Curso de Desenvolvimento Front-End

Carga Horária Total: 120 horas

Módulo 1: Introdução ao Desenvolvimento Web (10 horas)

1.1 O que é desenvolvimento web?

- **História e evolução da web:**
  - Breve histórico da internet e do desenvolvimento web.
  - Como a web evoluiu de documentos estáticos para aplicações dinâmicas e interativas.
- **Diferença entre front-end e back-end:**
  - Definição e responsabilidades do front-end e back-end.
  - Como essas áreas se complementam no desenvolvimento de uma aplicação web.
- **Tecnologias e ferramentas do desenvolvedor web:**
  - HTML, CSS, JavaScript: A tríade do front-end.
  - Introdução a ferramentas de desenvolvimento como editores de código, navegadores e emuladores.

1.2 Configuração do Ambiente de Desenvolvimento

- **Instalação do VS Code ou outro editor de texto:**
  - Passo a passo para instalar o VS Code.
  - Configuração básica (temas, extensões, etc.).
- **Introdução ao Git e GitHub:**
  - Conceito de controle de versão.
  - Criação de um repositório Git e primeiros comandos.
- **Configuração de ambiente local com Live Server:**
  - Configuração de um servidor local para visualização de projetos em tempo real.
  - Uso da extensão Live Server no VS Code.

Módulo 2: HTML5 (20 horas)

2.1 Estrutura Básica do HTML

- **Tags e elementos:**
  - Estrutura de uma tag HTML (`<tagname>content</tagname>`).
  - Elementos de bloco vs. elementos inline.
  - Hierarquia de elementos (pai e filho).
- **Semântica HTML:**
  - Uso de tags semânticas (`<header>`, `<nav>`, `<section>`, `<article>`, `<footer>`).
  - Importância da semântica para SEO e acessibilidade.
- **Estrutura básica de uma página HTML:**

    - O elemento `<doctype>` e sua importância.
    - Estrutura mínima de uma página: `<html>`, `<head>`, `<body>`.
    - Meta tags e seus atributos (viewport, charset, etc.).

2.2 Formulários e Inputs

- **Criação de formulários:**
    - Tags principais: `<form>`, `<input>`, `<label>`, `<textarea>`, `<select>`.
    - Atributos importantes: `action`, `method`, `name`, `value`, `placeholder`.
- **Tipos de inputs:**
    - Tipos comuns: `text`, `email`, `password`, `radio`, `checkbox`, `file`, `submit`.
    - Validação de formulários com HTML5 (ex: `required`, `pattern`, `minlength`, `maxlength`).
- **Validação de formulários:**
    - Validação de dados do lado do cliente.
    - Atributos de validação nativos do HTML5.

2.3 Multimídia

- **Inserção de imagens, vídeos e áudios:**
    - Tags: `<img>`, `<video>`, `<audio>`.
    - Atributos importantes: `src`, `alt`, `controls`, `autoplay`, `loop`.
- **Tags de vídeo e áudio:**
    - Exemplos de uso de `<video>` e `<audio>`.
    - Integração com diferentes formatos de mídia (MP4, OGG, etc.).
- **Atributos importantes (alt, title, etc.):**
    - Importância do atributo `alt` para acessibilidade e SEO.
    - Uso do atributo `title` para descrições adicionais.

2.4 Atividades Práticas

- **Criação de páginas web estáticas com HTML5:**
    - Criação de uma página de perfil pessoal.
    - Criação de um formulário de contato simples.
- **Projeto prático: Página pessoal ou de portfólio:**
    - Estruturação de um portfólio com seções de biografia, projetos e contatos.
    - Uso de tags semânticas para melhor organização.

Módulo 3: CSS3 (25 horas)

3.1 Conceitos Básicos de CSS

- **Seletores e propriedades:**
    - Seletores básicos (elemento, classe, id).
    - Propriedades CSS mais comuns (color, font-size, margin, padding).
- **Box Model:**
    - Conceito de Box Model (content, padding, border, margin).

    - Diferença entre box-sizing: content-box e border-box.
- **Tipos de posicionamento (static, relative, absolute, fixed):**
    - Definição e exemplos de uso de cada tipo de posicionamento.
    - Aplicações práticas (menus fixos, elementos sobrepostos).

3.2 Estilização Avançada

- **Flexbox:**
    - Conceito e utilidade do Flexbox para layouts.
    - Propriedades principais (justify-content, align-items, flex-direction, flex-wrap).
- **Grid Layout:**
    - Introdução ao CSS Grid e comparação com Flexbox.
    - Propriedades do Grid (grid-template-rows, grid-template-columns, gap).
- **Animações e Transições:**
    - Criação de transições suaves com `transition`.
    - Animações com `@keyframes` e `animation`.

3.3 Responsividade

- **Media Queries:**
    - Introdução ao conceito de media queries.
    - Criação de layouts responsivos com diferentes breakpoints.
- **Design responsivo (mobile-first, desktop-first):**
    - Abordagens mobile-first vs. desktop-first.
    - Como ajustar tipografia e espaçamento para diferentes dispositivos.
- **Unidades relativas e absolutas:**
    - Diferença entre px, em, rem, %.
    - Uso de unidades relativas para layouts responsivos.

3.4 Atividades Práticas

- **Estilização de layouts complexos usando Flexbox e Grid:**
    - Criação de um layout de página com múltiplas colunas e cabeçalho fixo.
    - Ajuste de layouts para dispositivos móveis.
- **Projeto prático: Site responsivo para múltiplos dispositivos:**
    - Desenvolvimento de um site completo que se adapta a diferentes tamanhos de tela.
    - Teste e depuração de problemas de responsividade.

Módulo 4: JavaScript (30 horas)

4.1 Introdução ao JavaScript

- **Sintaxe e tipos de dados:**
    - Declaração de variáveis com `var`, `let`, `const`.
    - Tipos de dados primitivos (string, number, boolean, null, undefined).
- **Variáveis, operadores e expressões:**
    - Operadores aritméticos, lógicos e de comparação.
    - Operadores de atribuição e incremento/decremento.

- **Condicionais e loops:**
    - Estruturas de controle de fluxo (`if`, `else if`, `else`, `switch`).
    - Estruturas de repetição (`for`, `while`, `do while`).

4.2 Funções e Escopos

- **Declaração de funções:**
    - Criação de funções com `function`.
    - Funções com parâmetros e retorno de valores.
- **Funções anônimas e arrow functions:**
    - Conceito de funções anônimas.
    - Uso de arrow functions (`const func = () => {}`).
- **Escopos e closures:**
    - Escopo global vs. escopo local.
    - Conceito de closures e exemplos práticos.

4.3 Manipulação do DOM

- **Seleção de elementos:**
    - Métodos de seleção: `getElementById`, `getElementsByClassName`, `querySelector`, `querySelectorAll`.
- **Eventos e interatividade:**
    - Criação e manipulação de eventos (`click`, `mouseover`, `keydown`).
    - Adição e remoção de ouvintes de eventos (`addEventListener`, `removeEventListener`).
- **Manipulação de atributos e estilos:**
    - Alteração de atributos com `setAttribute` e `getAttribute`.
    - Estilização dinâmica com `style`.

4.4 Programação Assíncrona

- **Callbacks:**
    - Introdução ao conceito de callbacks.
    - Uso de callbacks para operações assíncronas.
- **Promises e async/await:**
    - Conceito de Promises (`new Promise`, `then`, `catch`).
    - Simplificação de código assíncrono com `async/await`.
- **Fetch API e manipulação de dados:**
    - Realização de requisições HTTP com `fetch`.
    - Manipulação de respostas em JSON e tratamento de erros.

4.5 Atividades Práticas

- **Criação de scripts interativos para páginas web:**
    - Desenvolvimento de uma calculadora simples.
    - Criação de uma galeria de imagens interativa.
- **Projeto prático: Aplicação web simples com manipulação de DOM:**

- Desenvolvimento de uma aplicação To-Do List com funcionalidades de adicionar, remover e editar tarefas.

Módulo 5: Bootstrap (20 horas)

5.1 Introdução ao Bootstrap

- **O que é Bootstrap?**
  - Histórico do Bootstrap e sua importância no desenvolvimento web.
  - Como o Bootstrap simplifica a criação de layouts responsivos.
- **Instalação e configuração:**
  - Inclusão do Bootstrap via CDN e instalação local.
  - Estrutura básica de uma página Bootstrap.
- **Estrutura básica do Bootstrap:**
  - Introdução ao sistema de grid do Bootstrap.
  - Uso de containers, rows e columns para criação de layouts.

5.2 Componentes do Bootstrap

- **Grid System:**
  - Conceito de grid responsivo.
  - Criação de layouts complexos usando o grid system.
- **Buttons, Navbars, Cards, etc.:**
  - Uso de componentes como botões (`.btn`), navbars (`.navbar`), e cards (`.card`).
  - Customização e estilo de componentes.
- **Modals, Alerts e outros componentes interativos:**
  - Criação de modais, alerts e tooltips.
  - Controle de comportamento com JavaScript (abertura e fechamento de modais, por exemplo).

5.3 Customização e Temas

- **Personalização do Bootstrap com variáveis:**
  - Alteração de variáveis Sass para customização global.
  - Exemplo: Mudança de cores, fontes, tamanhos padrão.
- **Uso de temas Bootstrap:**
  - Instalação e uso de temas pré-fabricados.
  - Ajuste e personalização de temas existentes.
- **Criação de temas personalizados:**
  - Criação de um tema do zero usando Sass.
  - Como criar um estilo coeso e personalizado para um projeto Bootstrap.

5.4 Atividades Práticas

- **Desenvolvimento de um site utilizando Bootstrap:**
  - Criação de uma landing page moderna e responsiva.
  - Integração de componentes Bootstrap para melhorar a usabilidade.
- **Projeto prático: Landing page ou blog com Bootstrap:**

- Desenvolvimento de um blog ou landing page com foco em design e usabilidade.
- Customização do Bootstrap para alinhar com a identidade visual do projeto.

Módulo 6: Frameworks JavaScript (15 horas)

6.1 Introdução a Frameworks e Bibliotecas

- **O que são frameworks e bibliotecas?**
  - Definição e diferenças entre frameworks e bibliotecas.
  - Por que usar frameworks no desenvolvimento front-end.
- **Diferenças entre frameworks e bibliotecas:**
  - Exemplo de bibliotecas: jQuery.
  - Exemplo de frameworks: React.js, Vue.js.

6.2 React.js

- **Conceitos básicos de React:**
  - O que é React e como ele funciona.
  - Componentes e a abordagem de desenvolvimento baseada em componentes.
- **Componentes e JSX:**
  - Criação de componentes com JSX.
  - Composição de componentes.
- **Estado e Props:**
  - Definição de estado e props.
  - Passagem de dados entre componentes através de props.

6.3 Vue.js

- **Conceitos básicos de Vue:**
  - O que é Vue e suas principais características.
  - Comparação entre Vue e React.
- **Diretivas e bindings:**
  - Uso de diretivas como `v-if`, `v-for`, `v-bind`, `v-model`.
  - Interpolação de dados e manipulação de eventos.
- **Componentes e propriedades reativas:**
  - Criação de componentes Vue.
  - Reatividade no Vue e como ela simplifica a manipulação do DOM.

6.4 Atividades Práticas

- **Criação de uma aplicação web simples com React ou Vue:**
  - Desenvolvimento de um projeto simples como um contador ou lista de tarefas.
  - Exploração das principais funcionalidades do framework escolhido.
- **Projeto prático: To-Do List ou Calculadora com React/Vue:**
  - Desenvolvimento de uma aplicação completa com funcionalidades interativas.
  - Implementação de estado e manipulação de eventos.

Módulo 7: Controle de Versão e Deploy (10 horas)

7.1 Git e Controle de Versão

- **Comandos básicos do Git:**
  - Iniciar um repositório (`git init`), adicionar arquivos (`git add`), commitar (`git commit`).
  - Visualização de histórico (`git log`), status (`git status`).
- **Branches, merges e pull requests:**
  - Criação e uso de branches (`git branch`, `git checkout`).
  - Fusão de branches (`git merge`) e resolução de conflitos.
  - Criação de pull requests no GitHub.
- **Gerenciamento de projetos com GitHub:**
  - Hospedagem de repositórios no GitHub.
  - Colaboração em projetos com issues e pull requests.
  - Uso do GitHub Pages para hospedar sites estáticos.

7.2 Deploy de Aplicações Web

- **Ferramentas de deploy (Netlify, Vercel, GitHub Pages):**
  - Introdução a diferentes ferramentas de deploy.
  - Vantagens e desvantagens de cada ferramenta.
- **Hospedagem de sites estáticos:**
  - Publicação de sites no Netlify, Vercel, ou GitHub Pages.
  - Configuração de SSL, redirecionamentos e configurações básicas.
- **Configuração de DNS e domínios:**
  - Como registrar e configurar um domínio personalizado.
  - Configuração de DNS para apontar para o serviço de hospedagem.

7.3 Atividades Práticas

- **Publicação de um projeto final:**
  - Deploy de um projeto web desenvolvido durante o curso.
  - Configuração de um domínio personalizado e SSL.
- **Deploy de um site completo utilizando ferramentas de deploy:**
  - Publicação de um site usando Netlify ou Vercel.
  - Configuração de redirecionamentos e outras opções de deploy.

Módulo 8: Projeto Final (10 horas)

8.1 Desenvolvimento do Projeto Final

- **Definição do projeto:**
  - Escolha de um projeto realista para ser desenvolvido durante o módulo.
  - Definição de requisitos e objetivos do projeto.
- **Planejamento e design:**
  - Criação de wireframes e mockups para o projeto.
  - Planejamento do desenvolvimento em etapas (backlog, sprints, etc.).

- **Implementação:**
    - Desenvolvimento do projeto com base nos conhecimentos adquiridos.
    - Iterações, testes e melhorias contínuas.

8.2 Apresentação e Feedback

- **Apresentação do projeto para a turma:**
    - Demonstração do projeto finalizado.
    - Explicação das escolhas de design e implementação.
- **Feedback e melhoria contínua:**
    - Recebimento de feedback dos colegas e instrutores.
    - Implementação de melhorias baseadas no feedback.

8.3 Conclusão do Curso

- **Revisão dos principais conceitos:**
    - Recapitulação dos principais temas abordados durante o curso.
    - Discussão sobre desafios e aprendizados.
- **Dicas para continuar aprendendo e se aperfeiçoar:**
    - Sugestões de recursos adicionais (livros, cursos, blogs, comunidades).
    - Como acompanhar as tendências e inovações no desenvolvimento front-end.
- **Preparação para o mercado de trabalho:**
    - Como montar um portfólio.
    - Dicas para entrevistas e processos seletivos.
    - Participação em comunidades e eventos da área.

Módulo 1: Introdução ao Desenvolvimento Web (10 horas)

Objetivo do Módulo:

Este módulo visa proporcionar uma compreensão completa do desenvolvimento web, abordando sua história, evolução, e as diferenças fundamentais entre front-end e back-end. Além disso, os alunos irão configurar seu ambiente de desenvolvimento e aprender a usar as principais ferramentas que serão essenciais ao longo do curso.

1.1 O que é Desenvolvimento Web? (2 horas)

1.1.1 História e Evolução da Web

- **Contextualização:**
    - A web transformou radicalmente a forma como acessamos e compartilhamos informações. Compreender sua evolução é crucial para entender o cenário atual do desenvolvimento web.
- **Conteúdo:**
    - **Web 1.0:**
        - **Definição:** A Web 1.0 refere-se à primeira fase da internet, onde os sites eram estáticos e não interativos. O conteúdo era criado e atualizado por webmasters e os usuários tinham um papel passivo.
        - **Exemplos:**
            - Páginas pessoais simples, como o “Geocities”.
            - Sites de empresas, onde o conteúdo era apenas informativo, como catálogos de produtos.
            - **Exemplo Prático:** Criar uma página HTML estática com texto e imagens, sem qualquer interatividade.
    - **Web 2.0:**
        - **Definição:** A Web 2.0 trouxe a interatividade e a colaboração em massa. Plataformas como blogs, redes sociais e wikis permitiram que os usuários criassem e compartilhassem conteúdo.
        - **Exemplos:**
            - Blogs como WordPress, onde os usuários podem postar artigos.
            - Redes sociais como Facebook e Twitter, onde os usuários criam e compartilham conteúdo.
            - **Exemplo Prático:** Discutir como um site de rede social permite interações complexas, como postar, comentar e compartilhar conteúdos.
    - **Web 3.0:**
        - **Definição:** A Web 3.0 é a fase atual, que foca na personalização, descentralização e inteligência artificial. Ela promete uma internet mais inteligente, onde os dados são conectados de forma mais significativa.

        - **Exemplos:**
            - Aplicações descentralizadas (dApps) que rodam em blockchain.
            - Assistentes virtuais como Siri e Alexa, que entendem e respondem a comandos em linguagem natural.
            - **Exemplo Prático:** Explorar como a blockchain está sendo utilizada para criar contratos inteligentes (smart contracts) e discutir as implicações disso para o futuro da web.
- **Atividade:**
    - **Discussão em Grupo:** Os alunos devem debater sobre como a web mudou suas vidas e como ela pode evoluir nos próximos anos. Devem também compartilhar exemplos pessoais de como utilizaram diferentes fases da web.

1.1.2 Diferença entre Front-End e Back-End

- **Contextualização:**
    - A arquitetura de uma aplicação web é dividida em duas partes principais: o front-end e o back-end. Entender a diferença entre essas duas camadas é essencial para qualquer desenvolvedor.
- **Conteúdo:**
    - **Front-End:**
        - **Definição:** O front-end é a parte do desenvolvimento web que lida com tudo que o usuário vê e interage diretamente. Ele se concentra na criação de interfaces de usuário (UI) e experiências de usuário (UX).
        - **Tecnologias Principais:** HTML (estrutura), CSS (estilo), JavaScript (interatividade).
        - **Responsabilidades:**
            - Criar layouts responsivos que funcionem em diversos dispositivos.
            - Garantir que o site seja acessível e tenha uma boa experiência de usuário.
            - **Exemplo Prático:** Criar um formulário de contato simples que captura o nome, e-mail e mensagem do usuário, estilizando-o para parecer atraente e fácil de usar.
    - **Back-End:**
        - **Definição:** O back-end é a parte do desenvolvimento web que lida com a lógica de negócios, banco de dados, e interações do servidor. Ele processa os dados que são enviados do front-end e os armazena ou manipula conforme necessário.
        - **Tecnologias Principais:** Linguagens como Python, Ruby, PHP, Node.js, e frameworks como Django, Laravel, Express.
        - **Responsabilidades:**
            - Gerenciar autenticação de usuários e sessões.
            - Conectar-se a um banco de dados para armazenar ou recuperar dados.

        - **Exemplo Prático:** Criar uma API simples que recebe dados de um formulário (como o exemplo de front-end) e os armazena em um banco de dados.
- **Atividade:**
    - **Exercício de Mapeamento:** Os alunos devem criar um fluxograma mostrando como os dados se movem de uma página de formulário no front-end para o servidor no back-end e vice-versa. Isso ajuda a visualizar o fluxo de trabalho entre as duas camadas.

1.1.3 Tecnologias e Ferramentas do Desenvolvedor Web

- **Contextualização:**
    - Desenvolver para a web exige o domínio de várias ferramentas e tecnologias. Este tópico visa introduzir as principais ferramentas que serão usadas ao longo do curso.
- **Conteúdo:**
    - **HTML, CSS, JavaScript:**
        - **HTML:**
            - **Definição:** HTML é a linguagem de marcação que define a estrutura básica de uma página web.
            - **Exemplo Prático:** Criar uma página HTML básica com um título, parágrafos e uma lista.
        - **CSS:**
            - **Definição:** CSS é a linguagem de estilo usada para descrever a apresentação de um documento HTML.
            - **Exemplo Prático:** Estilizar a página HTML criada anteriormente, adicionando cores, espaçamento, e fontes.
        - **JavaScript:**
            - **Definição:** JavaScript é uma linguagem de programação que permite a criação de conteúdo dinâmico e interativo em uma página web.
            - **Exemplo Prático:** Adicionar uma função JavaScript que exibe uma mensagem de boas-vindas ao carregar a página.
    - **Ferramentas de Desenvolvimento:**
        - **Editores de Código:**
            - **VS Code:** Um dos editores de código mais populares, com suporte para extensões que facilitam o desenvolvimento.
            - **Exemplo Prático:** Configurar o VS Code com extensões essenciais como Prettier (para formatação de código) e Live Server (para visualização ao vivo).
        - **Navegadores:**
            - **Google Chrome e DevTools:** O Chrome é amplamente utilizado por desenvolvedores devido às suas poderosas ferramentas de desenvolvedor embutidas.
            - **Exemplo Prático:** Utilizar o DevTools para inspecionar elementos de uma página e alterar o CSS em tempo real.

        - **Git e Controle de Versão:**
            - **Definição:** Git é um sistema de controle de versão distribuído que permite rastrear mudanças no código e colaborar com outros desenvolvedores.
            - **Exemplo Prático:** Inicializar um repositório Git e fazer os primeiros commits de um projeto web simples.
- **Atividade:**
    - **Exploração Prática:** Os alunos irão explorar o DevTools do Google Chrome para inspecionar a estrutura de uma página web real, experimentar modificações de CSS ao vivo e visualizar o impacto dessas mudanças.

1.2 Configuração do Ambiente de Desenvolvimento (2 horas)

1.2.1 Escolha do Editor de Código

- **Contextualização:**
    - Um editor de código eficiente é fundamental para a produtividade do desenvolvedor. Este tópico explora as opções disponíveis e como configurá-las.
- **Conteúdo:**
    - **Instalação e Configuração:**
        - **VS Code:** Passo a passo para baixar e instalar o Visual Studio Code.
        - **Configurações Essenciais:** Configuração do tema, atalhos e extensões básicas.
        - **Exemplo Prático:** Configurar o VS Code para trabalhar com HTML, CSS, e JavaScript, instalando extensões como Emmet (para facilitar a escrita de código HTML) e Live Server (para visualização em tempo real).

1.2.2 Instalação de Navegadores e Ferramentas

- **Contextualização:**
    - Testar o código em diferentes navegadores garante que o site funcione corretamente para todos os usuários. Ferramentas como DevTools ajudam na depuração e otimização.
- **Conteúdo:**
    - **Navegadores Essenciais:**
        - **Instalação de Google Chrome e Firefox:** Instruções para instalar e configurar navegadores.
        - **DevTools do Chrome:** Exploração de funcionalidades como inspecionar elementos, console JavaScript, e análise de desempenho.
        - **Exemplo Prático:** Realizar um teste de responsividade utilizando as ferramentas de emulação de dispositivos móveis no DevTools do Chrome.

1.2.3 Controle de Versão com Git

- **Contextualização:**

        - Git é uma ferramenta essencial para qualquer desenvolvedor moderno. Ele permite que você rastreie as alterações no código e colabore com outros desenvolvedores.
- **Conteúdo:**
    - **Instalação do Git:**
        - **Guia de Instalação:** Passos para instalar Git no Windows, macOS e Linux.
        - **Configuração Inicial:** Configuração de nome de usuário, e-mail, e editor de texto preferido.
    - **Comandos Básicos:**
        - **git init:** Inicializar um repositório Git.
        - **git add e git commit:** Adicionar mudanças ao repositório e salvar um snapshot dessas mudanças.
        - **git status:** Verificar o estado do repositório.
        - **Exemplo Prático:** Criar um repositório Git local, fazer alterações em um projeto HTML/CSS simples, e realizar commits dessas mudanças.
- **Atividade:**
    - **Exercício Prático:** Criar um projeto simples, versioná-lo com Git, e simular um fluxo básico de trabalho de um desenvolvedor (adicionar mudanças, commitar, e verificar o histórico de commits).

Este conteúdo abrangente para o Módulo 1 dá aos alunos uma base sólida no desenvolvimento web, equipando-os com o conhecimento e as ferramentas necessárias para avançar para os módulos seguintes, onde conceitos mais complexos serão explorados.

Módulo 2: Fundamentos de HTML e CSS (20 horas)

Objetivo do Módulo:

Este módulo tem como objetivo ensinar os fundamentos de HTML e CSS, as linguagens que formam a base de qualquer site ou aplicação web. Os alunos aprenderão como estruturar documentos HTML, estilizar páginas usando CSS, e criar layouts responsivos que funcionam em diversos dispositivos.

2.1 Fundamentos de HTML (8 horas)

2.1.1 Estrutura Básica de um Documento HTML

- **Contextualização:**
    - HTML é a base de qualquer página web. Conhecer sua estrutura e seus elementos essenciais é o primeiro passo para criar um site.
- **Conteúdo:**
    - **Elementos HTML:**

        - **Definição:** HTML é composto de elementos que estruturam e organizam o conteúdo de uma página web.
        - **Estrutura Básica:** `<!DOCTYPE html>`, `<html>`, `<head>`, `<title>`, `<body>`.
        - **Exemplo Prático:** Criar um documento HTML simples com título, cabeçalho, parágrafos, e uma lista de itens.
    - **Tags Comuns:**
        - **Tags de Texto:** `<h1> - <h6>`, `<p>`, `<strong>`, `<em>`.
        - **Tags de Imagem e Link:** `<img>`, `<a>`.
        - **Exemplo Prático:** Adicionar uma imagem e links a um documento HTML.

2.1.2 Trabalhando com Formulários em HTML

- **Contextualização:**
    - Formulários são essenciais para capturar dados dos usuários. Entender como criá-los e organizá-los é crucial.
- **Conteúdo:**
    - **Elementos de Formulário:**
        - **Tags de Formulário:** `<form>`, `<input>`, `<label>`, `<textarea>`, `<button>`, `<select>`.
        - **Tipos de Inputs:** Texto, e-mail, senha, radio, checkbox, file, etc.
        - **Exemplo Prático:** Criar um formulário de contato com campos para nome, e-mail, mensagem e um botão de envio.
    - **Atributos Importantes:**
        - **Action e Method:** Definir para onde os dados serão enviados e o método de envio (`GET`, `POST`).
        - **Placeholder, Required:** Melhorando a usabilidade dos formulários.
        - **Exemplo Prático:** Criar um formulário de inscrição em newsletter com validação básica de campos obrigatórios.

2.1.3 Listas, Tabelas e Imagens

- **Contextualização:**
    - Listas e tabelas são usadas para organizar informações de forma clara e estruturada. Imagens enriquecem o conteúdo visual.
- **Conteúdo:**
    - **Listas:**
        - **Tipos de Listas:** Ordenadas (`<ol>`), não ordenadas (`<ul>`), listas de descrição (`<dl>`).
        - **Exemplo Prático:** Criar uma lista de tarefas usando `<ul>` e `<li>`.
    - **Tabelas:**
        - **Elementos de Tabela:** `<table>`, `<tr>`, `<td>`, `<th>`, `<thead>`, `<tbody>`.
        - **Exemplo Prático:** Criar uma tabela para exibir uma lista de produtos com nomes, preços e descrições.
    - **Imagens:**

        - **Tag `<img>`:** Atributos `src`, `alt`, `width`, `height`.
        - **Exemplo Prático:** Inserir e ajustar uma imagem em uma página HTML.

2.1.4 Semântica e Acessibilidade em HTML

- **Contextualização:**
    - A semântica HTML ajuda a criar documentos mais organizados e acessíveis, melhorando a experiência do usuário e o SEO.
- **Conteúdo:**
    - **Elementos Semânticos:**
        - **Tags Semânticas:** `<header>`, `<footer>`, `<article>`, `<section>`, `<aside>`, `<nav>`.
        - **Exemplo Prático:** Estruturar uma página HTML utilizando tags semânticas para criar uma estrutura lógica e clara.
    - **Acessibilidade:**
        - **Boas Práticas:** Uso de `alt` para imagens, `label` associado a inputs, navegação via teclado.
        - **Exemplo Prático:** Melhorar a acessibilidade de um formulário existente, adicionando descrições e rótulos.

2.2 Fundamentos de CSS (12 horas)

2.2.1 Introdução ao CSS

- **Contextualização:**
    - CSS é usado para estilizar documentos HTML. Ele permite controlar cores, layouts, tipografia e muito mais.
- **Conteúdo:**
    - **Sintaxe CSS:**
        - **Seletores:** Tag, classe (`.`), ID (#), pseudo-classes (`:hover`).
        - **Propriedades e Valores:** Cor, fonte, espaçamento, borda.
        - **Exemplo Prático:** Criar uma folha de estilo CSS para alterar a cor de fundo e a cor do texto de uma página HTML.
    - **Aplicação de CSS:**
        - **Inline, Interno e Externo:** Estilos aplicados diretamente no elemento, dentro da tag `<style>`, ou em arquivos `.css`.
        - **Exemplo Prático:** Criar um arquivo CSS externo e linká-lo a um documento HTML.

2.2.2 Cores, Tipografia e Unidades de Medida

- **Contextualização:**
    - O design visual de um site é influenciado por cores, fontes e o uso de espaços. Entender como manipular esses elementos é essencial para criar interfaces atraentes.
- **Conteúdo:**

- **Cores:**
    - **Códigos de Cores:** Hexadecimal, RGB, HSL.
    - **Propriedades de Cor:** `color`, `background-color`, `border-color`.
    - **Exemplo Prático:** Alterar as cores de texto, fundo e bordas de uma página para criar um design coeso.
- **Tipografia:**
    - **Fontes:** Uso de fontes da web (Google Fonts), propriedades `font-family`, `font-size`, `font-weight`, `line-height`.
    - **Exemplo Prático:** Aplicar uma fonte personalizada a um documento e ajustar o espaçamento entre linhas e letras.
- **Unidades de Medida:**
    - **Px, Em, Rem:** Diferenças e quando usar cada uma.
    - **Exemplo Prático:** Comparar o uso de `px` e `em` para dimensionar fontes e margens em um layout.

2.2.3 Layouts e Posicionamento com CSS

- **Contextualização:**
    - Criar layouts flexíveis e responsivos é uma habilidade fundamental no desenvolvimento web moderno. CSS fornece várias técnicas para controle de layout e posicionamento de elementos.
- **Conteúdo:**
    - **Display:**
        - **Block vs Inline:** Diferenças entre elementos `block` e `inline`.
        - **Display Properties:** `display: block;`, `display: inline;`, `display: inline-block;`.
        - **Exemplo Prático:** Transformar um conjunto de elementos inline em blocos e vice-versa.
    - **Modelos de Layout:**
        - **Box Model:** `margin`, `border`, `padding`, `width`, `height`.
        - **Flexbox:** Conceitos de `flex container` e `flex items`, propriedades como `justify-content`, `align-items`.
        - **Grid Layout:** Definição de `grid container`, `grid items`, criação de colunas e linhas.
        - **Exemplo Prático:** Criar um layout de página com Flexbox que alinha itens no centro da tela, e outro com Grid para dividir a página em colunas e linhas.
    - **Posicionamento:**
        - **Positioning:** `static`, `relative`, `absolute`, `fixed`, `sticky`.
        - **Z-index:** Controle da ordem de sobreposição dos elementos.
        - **Exemplo Prático:** Posicionar uma barra de navegação no topo da página, fixa mesmo quando o usuário rola a página.

2.2.4 Design Responsivo e Media Queries

- **Contextualização:**

        - O design responsivo garante que um site seja acessível e utilizável em qualquer dispositivo, seja um desktop, tablet ou smartphone.
- **Conteúdo:**
    - **Responsive Design:**
        - **Viewport:** Uso da tag `<meta viewport>` para controlar a escala em dispositivos móveis.
        - **Fluid Grids:** Criação de layouts fluidos usando unidades relativas.
        - **Exemplo Prático:** Criar um layout fluido que se ajusta automaticamente ao redimensionar a janela do navegador.
    - **Media Queries:**
        - **Sintaxe:** `@media` para aplicar estilos condicionais baseados em largura de tela, orientação, etc.
        - **Breakpoints Comuns:** Definição de pontos de interrupção para diferentes tamanhos de tela.
        - **Exemplo Prático:** Criar um layout com media queries que altere a organização dos elementos em tamanhos de tela específicos, como reorganizar uma barra lateral para aparecer abaixo do conteúdo principal em telas pequenas.

Módulo 3: Introdução ao JavaScript (30 horas)

Objetivo do Módulo:

Este módulo visa introduzir os alunos ao JavaScript, uma linguagem fundamental para adicionar interatividade a páginas web. Os tópicos cobrem desde os conceitos básicos até a manipulação do DOM e operações assíncronas. Cada seção inclui exemplos de código para reforçar o aprendizado.

3.1 Fundamentos de JavaScript (10 horas)

3.1.1 Introdução ao JavaScript e Configuração do Ambiente

- **Contextualização:**
    - JavaScript é uma linguagem de programação essencial para criar interatividade em páginas web. Dominar essa linguagem é crucial para quem deseja se tornar um desenvolvedor front-end.
- **Conteúdo:**
    - **História e Importância do JavaScript:**
        - JavaScript surgiu em 1995 e rapidamente se tornou a linguagem padrão para a web, permitindo que desenvolvedores criem interações dinâmicas nas páginas.
    - **Integração com HTML:**

- A linguagem pode ser incorporada diretamente em documentos HTML usando a tag `<script>`.
- **Exemplo Prático:**

```
html
Copiar código
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Introdução ao JavaScript</title>
</head>
<body>
    <h1>Bem-vindo ao JavaScript</h1>
    <script>
        alert('Olá, Mundo!');
    </script>
</body>
</html>
```

Neste exemplo, um alerta com a mensagem "Olá, Mundo!" é exibido assim que a página é carregada.

- **Configurando o Ambiente de Desenvolvimento:**
  - Utilize editores como VS Code e configure o navegador para facilitar o processo de desenvolvimento e depuração.

3.1.2 Variáveis, Tipos de Dados e Operadores

- **Contextualização:**
  - Variáveis permitem armazenar dados que podem ser manipulados ao longo do código. Conhecer os tipos de dados e operadores é essencial para qualquer operação em JavaScript.
- **Conteúdo:**
  - **Declaração de Variáveis:**
    - Em JavaScript, você pode declarar variáveis usando `let`, `const`, ou `var`.
    - **Exemplo Prático:**

```
javascript
Copiar código
let nome = 'Maria';
const idade = 28;
var cidade = 'São Paulo';

console.log(nome); // Maria
console.log(idade); // 28
console.log(cidade); // São Paulo
```

  - **Tipos de Dados:**
    - JavaScript suporta vários tipos de dados, como números, strings e booleanos.
    - **Exemplo Prático:**

```
javascript
Copiar código
```

```
let preco = 19.99; // Número
let produto = 'Camiseta'; // String
let disponivel = true; // Boolean

console.log(typeof preco); // number
console.log(typeof produto); // string
console.log(typeof disponivel); // boolean
```

- **Operadores:**
  - JavaScript inclui operadores aritméticos, de comparação e lógicos.
  - **Exemplo Prático:**

```
javascript
Copiar código
let a = 10;
let b = 20;

console.log(a + b); // 30 (Soma)
console.log(a > b); // false (Comparação)
console.log(a < b && a > 5); // true (Lógico)
```

3.1.3 Controle de Fluxo: Condicionais e Loops

- **Contextualização:**
  - O controle de fluxo permite que você crie lógicas condicionais e repita blocos de código com base em determinadas condições.
- **Conteúdo:**
  - **Condicionais:**
    - Utilize `if`, `else` e `else if` para executar blocos de código com base em condições específicas.
    - **Exemplo Prático:**

```
javascript
Copiar código
let nota = 85;

if (nota >= 90) {
    console.log('Excelente');
} else if (nota >= 70) {
    console.log('Aprovado');
} else {
    console.log('Reprovado');
}
```

      Neste exemplo, se a nota for maior ou igual a 90, exibe "Excelente"; se for maior ou igual a 70, exibe "Aprovado"; caso contrário, exibe "Reprovado".
  - **Loops:**
    - Os loops `for`, `while` e `do...while` permitem repetir blocos de código.
    - **Exemplo Prático:**

```
javascript
Copiar código
for (let i = 0; i < 5; i++) {
    console.log('Número:', i);
}
```

```
let contador = 0;
while (contador < 5) {
    console.log('Contador:', contador);
    contador++;
}
```

Este exemplo utiliza um loop `for` para exibir os números de 0 a 4 e um loop `while` para fazer o mesmo.

3.1.4 Funções e Escopos

- **Contextualização:**
  - Funções são blocos de código que podem ser reutilizados. O escopo define onde as variáveis e funções estão disponíveis dentro do código.
- **Conteúdo:**
  - **Definição de Funções:**
    - Funções podem ser definidas de maneira declarativa ou através de expressões.
    - **Exemplo Prático:**

```
javascript
Copiar código
function saudacao(nome) {
    return 'Olá, ' + nome + '!';
}

console.log(saudacao('João')); // Olá, João!
```

Aqui, a função `saudacao` recebe um nome como parâmetro e retorna uma mensagem de saudação.

  - **Escopos:**
    - O escopo local é limitado a funções ou blocos, enquanto o escopo global está disponível em todo o script.
    - **Exemplo Prático:**

```
javascript
Copiar código
let mensagem = 'Olá, Mundo!'; // Escopo global

function exibirMensagem() {
    let saudacao = 'Bem-vindo!'; // Escopo local
    console.log(mensagem);
    console.log(saudacao);
}

exibirMensagem();
console.log(saudacao); // Erro: saudacao não está definida
fora da função
```

A variável `saudacao` está disponível apenas dentro da função `exibirMensagem`.

3.2 Manipulação do DOM com JavaScript (10 horas)

3.2.1 O Que é o DOM?

- **Contextualização:**
    - O DOM (Document Object Model) é uma interface que permite manipular elementos HTML e suas propriedades usando JavaScript.
- **Conteúdo:**
    - **Entendendo o DOM:**
        - O DOM representa a estrutura de um documento HTML como uma árvore de objetos que podem ser manipulados.
    - **Exemplo Prático:**

      ```
      javascript
      Copiar código
      // Acessando o conteúdo de um elemento pelo ID
      let titulo = document.getElementById('titulo');
      console.log(titulo.innerText);
      ```

      Supondo que exista um elemento `<h1 id="titulo">Bem-vindo ao Site</h1>`, o código acima acessa e exibe o texto "Bem-vindo ao Site".

3.2.2 Seleção e Manipulação de Elementos

- **Contextualização:**
    - A seleção e manipulação de elementos no DOM é essencial para criar páginas interativas e dinâmicas.
- **Conteúdo:**
    - **Seletores DOM:**
        - Métodos como `getElementById`, `getElementsByClassName`, e `querySelector` permitem selecionar elementos específicos.
        - **Exemplo Prático:**

          ```
          javascript
          Copiar código
          let paragrafos = document.querySelectorAll('p');
          paragrafos.forEach(paragrafo => {
              paragrafo.style.color = 'blue';
          });
          ```

          Neste exemplo, todos os elementos `<p>` na página terão a cor do texto alterada para azul.
    - **Manipulação de Conteúdo:**
        - É possível alterar o conteúdo de texto ou HTML de um elemento com `innerText` e `innerHTML`.
        - **Exemplo Prático:**

          ```
          javascript
          Copiar código
          let mensagem = document.getElementById('mensagem');
          mensagem.innerText = 'Este é o novo texto!';
          ```

Aqui, o texto do elemento com ID `mensagem` é alterado.

- **Manipulação de Atributos e Estilos:**
  - A manipulação de atributos pode ser feita usando `setAttribute`, enquanto estilos são alterados diretamente através do objeto `style`.
  - **Exemplo Prático:**

```
javascript
Copiar código
let imagem = document.querySelector('img');
imagem.setAttribute('alt', 'Descrição da imagem');
imagem.style.border = '2px solid red';
```

Neste exemplo, um atributo `alt` é adicionado a uma imagem, e a borda da imagem é alterada para uma linha vermelha de 2px.

3.2.3 Manipulação de Eventos

- **Contextualização:**
  - Eventos são ações como cliques ou teclas pressionadas, e o JavaScript pode responder a essas ações, criando interatividade.
- **Conteúdo:**
  - **Tipos de Eventos:**
    - Eventos comuns incluem `click`, `mouseover`, e `keydown`.
    - **Exemplo Prático:**

```
javascript
Copiar código
let botao = document.getElementById('meuBotao');
botao.addEventListener('click', function() {
    alert('Botão clicado!');
});
```

Este exemplo exibe um alerta quando o botão com ID `meuBotao` é clicado.

- **Adicionando e Removendo Eventos:**
  - Use `addEventListener` para adicionar e `removeEventListener` para remover eventos.
  - **Exemplo Prático:**

```
javascript
Copiar código
function mudarTexto() {
    this.innerText = 'Clicado!';
}

let botao = document.getElementById('meuBotao');
botao.addEventListener('click', mudarTexto);
```

Aqui, o texto do botão muda para "Clicado!" após o clique.

- **Eventos Padrão:**
  - Eventos padrão podem ser evitados usando `preventDefault`.
  - **Exemplo Prático:**

```
javascript
Copiar código
let link = document.getElementById('meuLink');
link.addEventListener('click', function(event) {
    event.preventDefault();
    alert('Link desativado!');
});
```

Este exemplo previne o comportamento padrão do link e exibe um alerta.

3.2.4 Manipulação de Elementos Dinâmicos

- **Contextualização:**
  - O JavaScript permite criar, remover e modificar elementos HTML em tempo real, permitindo uma experiência de usuário dinâmica.
- **Conteúdo:**
  - **Criando e Inserindo Elementos:**
    - Novos elementos podem ser criados com `createElement` e inseridos com `appendChild`.
    - **Exemplo Prático:**

      ```
      javascript
      Copiar código
      let novoParagrafo = document.createElement('p');
      novoParagrafo.innerText = 'Este é um novo parágrafo!';
      document.body.appendChild(novoParagrafo);
      ```

      Este código cria e adiciona um novo parágrafo ao final do corpo do documento.
  - **Removendo Elementos:**
    - Elementos podem ser removidos usando `removeChild` ou `remove`.
    - **Exemplo Prático:**

      ```
      javascript
      Copiar código
      let paragrafo =
      document.getElementById('paragrafoParaRemover');
      paragrafo.remove();
      ```

      Aqui, o parágrafo com o ID `paragrafoParaRemover` é removido do documento.
  - **Clonando Elementos:**
    - Elementos podem ser clonados usando `cloneNode`.
    - **Exemplo Prático:**

      ```
      javascript
      Copiar código
      let formulario = document.getElementById('meuFormulario');
      let formularioClone = formulario.cloneNode(true);
      document.body.appendChild(formularioClone);
      ```

      Este exemplo clona um formulário e o insere no documento.

3.3 Introdução ao JavaScript Assíncrono (10 horas)

3.3.1 Conceitos de JavaScript Assíncrono

- **Contextualização:**
  - O JavaScript assíncrono permite a execução de operações sem bloquear o fluxo do código, essencial para criar interfaces responsivas.
- **Conteúdo:**
  - **Assincronicidade:**
    - Operações assíncronas permitem que o código continue sendo executado enquanto uma tarefa é concluída em segundo plano.
    - **Callback Functions:**

      ```
      javascript
      Copiar código
      function saudar(callback) {
          setTimeout(() => {
              callback('Olá, Mundo!');
          }, 2000);
      }

      saudar(function(mensagem) {
          console.log(mensagem);
      });
      ```

      Neste exemplo, a função `saudar` exibe a mensagem "Olá, Mundo!" após um atraso de 2 segundos.
  - **Promises:**
    - `Promise` é um objeto que representa a eventual conclusão (ou falha) de uma operação assíncrona.
    - **Exemplo Prático:**

      ```
      javascript
      Copiar código
      let promessa = new Promise((resolve, reject) => {
          let sucesso = true;
          if (sucesso) {
              resolve('Operação bem-sucedida!');
          } else {
              reject('Falha na operação.');
          }
      });

      promessa.then((mensagem) => {
          console.log(mensagem);
      }).catch((erro) => {
          console.error(erro);
      });
      ```

      Este exemplo cria uma `Promise` que é resolvida ou rejeitada com base em uma condição, e o resultado é tratado com `then` e `catch`.

3.3.2 Async/Await

- **Contextualização:**

    - Async/Await é uma sintaxe mais simples e limpa para trabalhar com operações assíncronas em JavaScript.
- **Conteúdo:**
    - **Async/Await:**
        - Funções assíncronas retornam uma `Promise` e podem utilizar `await` para esperar a conclusão de uma operação.
        - **Exemplo Prático:**

```
javascript
Copiar código
async function buscarDados() {
    try {
        let resposta = await
fetch('https://jsonplaceholder.typicode.com/posts');
        let dados = await resposta.json();
        console.log(dados);
    } catch (erro) {
        console.error('Erro:', erro);
    }
}

buscarDados();
```

Neste exemplo, a função `buscarDados` utiliza `fetch` para buscar dados de uma API e os exibe no console. Se houver um erro, ele é capturado e exibido.

Módulo 4: Manipulação Avançada do DOM e Eventos (30 horas)

Objetivo do Módulo:

Este módulo aprofunda a manipulação do DOM e o gerenciamento de eventos em JavaScript. Os tópicos incluem técnicas avançadas de manipulação do DOM, gerenciamento de eventos mais complexo e manipulação de formulários. Exemplos de código são fornecidos para ilustrar os conceitos.

4.1 Manipulação Avançada do DOM (15 horas)

4.1.1 Trabalhando com Classes e IDs

- **Contextualização:**
    - Manipular classes e IDs é essencial para alterar dinamicamente o estilo e o comportamento dos elementos na página.
- **Conteúdo:**
    - **Manipulando Classes:**
        - Use métodos como `classList.add`, `classList.remove` e `classList.toggle` para alterar classes de elementos.
        - **Exemplo Prático:**

```
javascript
Copiar código
let elemento = document.getElementById('minhaDiv');
elemento.classList.add('novaClasse'); // Adiciona a classe
'novaClasse'
elemento.classList.remove('outraClasse'); // Remove a classe
'outraClasse'
elemento.classList.toggle('classeAlternativa'); // Adiciona ou
remove a classe 'classeAlternativa'
```

- **Manipulando IDs:**
  - IDs são únicos na página e podem ser usados para selecionar e manipular elementos específicos.
  - **Exemplo Prático:**

```
javascript
Copiar código
let elemento = document.querySelector('#minhaDiv');
elemento.id = 'novoID'; // Altera o ID do elemento para
'novoID'
```

4.1.2 Criando e Manipulando Elementos Dinâmicos

- **Contextualização:**
  - A criação e manipulação de elementos dinâmicos permite adicionar conteúdo à página em tempo real, respondendo às ações do usuário.
- **Conteúdo:**
  - **Criando Elementos:**
    - Utilize `createElement` para criar novos elementos e `appendChild` para adicioná-los ao DOM.
    - **Exemplo Prático:**

```
javascript
Copiar código
let novoElemento = document.createElement('div');
novoElemento.textContent = 'Este é um novo elemento!';
document.body.appendChild(novoElemento);
```

  - **Modificando Elementos:**
    - Altere o conteúdo e atributos dos elementos criados dinamicamente.
    - **Exemplo Prático:**

```
javascript
Copiar código
novoElemento.setAttribute('class', 'classeCriada');
novoElemento.style.backgroundColor = 'yellow';
```

4.1.3 Manipulando Estruturas de Tabelas

- **Contextualização:**
  - Manipular tabelas é uma habilidade importante para exibir e atualizar dados estruturados de forma dinâmica.
- **Conteúdo:**
  - **Criando Linhas e Células:**

- Adicione linhas e células a tabelas existentes usando `insertRow` e `insertCell`.
- **Exemplo Prático:**

```
javascript
Copiar código
let tabela = document.getElementById('minhaTabela');
let novaLinha = tabela.insertRow();
let novaCelula = novaLinha.insertCell();
novaCelula.textContent = 'Novo conteúdo';
```

- **Manipulando Dados da Tabela:**
  - Atualize e remova linhas e células conforme necessário.
  - **Exemplo Prático:**

```
javascript
Copiar código
let linha = tabela.deleteRow(1); // Remove a segunda linha da
tabela
```

4.1.4 Utilizando Templates HTML

- **Contextualização:**
  - Templates HTML são úteis para criar estruturas de conteúdo que podem ser reutilizadas e manipuladas dinamicamente.
- **Conteúdo:**
  - **Definindo e Usando Templates:**
    - Defina templates com a tag `<template>` e use `content` para acessar e clonar o conteúdo.
    - **Exemplo Prático:**

```
html
Copiar código
<template id="meuTemplate">
    <div class="item">
        <h2>Título</h2>
        <p>Descrição</p>
    </div>
</template>
<script>
    let template =
document.getElementById('meuTemplate').content;
    document.body.appendChild(template.cloneNode(true));
</script>
```

4.2 Gerenciamento Avançado de Eventos (10 horas)

4.2.1 Eventos de Teclado e Mouse

- **Contextualização:**
  - A compreensão dos eventos de teclado e mouse permite criar interfaces interativas e responsivas.
- **Conteúdo:**

- **Eventos de Teclado:**
  - Use eventos como `keydown`, `keyup`, e `keypress` para capturar entradas do teclado.
  - **Exemplo Prático:**

```
javascript
Copiar código
document.addEventListener('keydown', function(event) {
    console.log('Tecla pressionada:', event.key);
});
```

- **Eventos de Mouse:**
  - Manipule eventos como `click`, `mouseover`, e `mouseout` para interações com o mouse.
  - **Exemplo Prático:**

```
javascript
Copiar código
document.getElementById('minhaDiv').addEventListener('mouseove
r', function() {
    this.style.backgroundColor = 'lightblue';
});
```

4.2.2 Eventos Personalizados

- **Contextualização:**
  - Eventos personalizados permitem que você defina e dispare seus próprios eventos, facilitando a comunicação entre diferentes partes do código.
- **Conteúdo:**
  - **Criando e Disparando Eventos Personalizados:**
    - Use `CustomEvent` para criar e despachar eventos personalizados.
    - **Exemplo Prático:**

```
javascript
Copiar código
let evento = new CustomEvent('minhaAcao', { detail:
{ mensagem: 'Evento personalizado!' } });
document.dispatchEvent(evento);

document.addEventListener('minhaAcao', function(e) {
    console.log(e.detail.mensagem);
});
```

4.2.3 Delegação de Eventos

- **Contextualização:**
  - A delegação de eventos permite que você adicione um único ouvinte de eventos para múltiplos elementos, melhorando o desempenho e a organização do código.
- **Conteúdo:**
  - **Implementando Delegação de Eventos:**
    - Adicione um ouvinte de eventos a um pai para capturar eventos de seus filhos.
    - **Exemplo Prático:**

```
javascript
Copiar código
document.getElementById('container').addEventListener('click',
function(event) {
    if (event.target && event.target.matches('button')) {
        alert('Botão clicado!');
    }
});
```

4.2.4 Manipulação de Formulários e Validação

- **Contextualização:**
    - Manipular e validar formulários é essencial para capturar e garantir a integridade dos dados inseridos pelos usuários.
- **Conteúdo:**
    - **Manipulando Formulários:**
        - Acesse e manipule valores de formulários com `form.elements` e `form.submit()`.
        - **Exemplo Prático:**

```
javascript
Copiar código
let formulario = document.getElementById('meuFormulario');
formulario.addEventListener('submit', function(event) {
    event.preventDefault();
    let nome = formulario.elements['nome'].value;
    console.log('Nome:', nome);
});
```

    - **Validação de Formulários:**
        - Utilize propriedades como `validity` e métodos como `checkValidity` para validar dados de entrada.
        - **Exemplo Prático:**

```
javascript
Copiar código
let input = document.getElementById('meuInput');
if (!input.checkValidity()) {
    console.log('O valor inserido é inválido.');
}
```

4.3 Criação de Interfaces Dinâmicas e Responsivas (5 horas)

4.3.1 Criando Interfaces Dinâmicas

- **Contextualização:**
    - Interfaces dinâmicas respondem às interações do usuário e podem atualizar o conteúdo e a aparência da página em tempo real.
- **Conteúdo:**
    - **Atualizando o Conteúdo Dinamicamente:**
        - Modifique o conteúdo da página em resposta a ações do usuário.

- **Exemplo Prático:**

```
javascript
Copiar código
document.getElementById('btnAtualizar').addEventListener('clic
k', function() {
    document.getElementById('conteudo').innerHTML = 'Conteúdo
atualizado!';
});
```

4.3.2 Design Responsivo com JavaScript

- **Contextualização:**
  - JavaScript pode ser usado para ajustar a interface com base em diferentes tamanhos de tela e dispositivos.
- **Conteúdo:**
  - **Ajustando Layout Dinamicamente:**
    - Alterar estilos e elementos com base nas dimensões da tela.
    - **Exemplo Prático:**

```
javascript
Copiar código
window.addEventListener('resize', function() {
    if (window.innerWidth < 600) {
        document.body.style.backgroundColor = 'lightgray';
    } else {
        document.body.style.backgroundColor = 'white';
    }
});
```

Módulo 5: Introdução ao Frameworks Front-End (30 horas)

Objetivo do Módulo:

Este módulo apresenta frameworks populares para desenvolvimento front-end, focando em React, Angular e Vue.js. Os tópicos incluem instalação, configuração básica e criação de aplicações simples com cada framework.

5.1 Introdução ao React (10 horas)

5.1.1 Configuração do Ambiente

- **Contextualização:**
  - O React é uma biblioteca JavaScript para construir interfaces de usuário. Configurar o ambiente é o primeiro passo para começar a desenvolver com React.
- **Conteúdo:**
  - **Instalação com Create React App:**
    - Utilize o Create React App para iniciar um novo projeto React.
    - **Exemplo Prático:**

```
bash
Copiar código
npx create-react-app meuApp
cd meuApp
npm start
```

- **Estrutura do Projeto:**
  - Familiarize-se com a estrutura básica do projeto gerado.

5.1.2 Componentes React

- **Contextualização:**
  - Componentes são os blocos fundamentais da aplicação React. Eles permitem criar interfaces complexas a partir de componentes menores.
- **Conteúdo:**
  - **Criando Componentes Funcionais:**
    - Defina e utilize componentes funcionais.
    - **Exemplo Prático:**

```
javascript
Copiar código
function MeuComponente() {
    return <h1>Olá, React!</h1>;
}

export default MeuComponente;
```

  - **Propriedades e Estado:**
    - Passe dados para componentes com `props` e mantenha o estado interno com `useState`.
    - **Exemplo Prático:**

```
javascript
Copiar código
import React, { useState } from 'react';

function Contador() {
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>Você clicou {count} vezes</p>
            <button onClick={() => setCount(count + 1)}>Clique
aqui</button>
        </div>
    );
}

export default Contador;
```

5.1.3 Manipulação de Eventos no React

- **Contextualização:**
  - Manipular eventos no React é feito de forma semelhante ao JavaScript tradicional, mas integrado com a sintaxe JSX.
- **Conteúdo:**

- **Eventos e Manipuladores:**
  - Adicione manipuladores de eventos a elementos JSX.
  - **Exemplo Prático:**

```
javascript
Copiar código
function MeuBotao() {
    const handleClick = () => {
        alert('Botão clicado!');
    };

    return <button onClick={handleClick}>Clique aqui</button>;
}
```

5.2 Introdução ao Angular (10 horas)

5.2.1 Configuração do Ambiente

- **Contextualização:**
  - Angular é um framework para construção de aplicações web robustas. Configurar o ambiente é essencial para começar a trabalhar com Angular.
- **Conteúdo:**
  - **Instalação com Angular CLI:**
    - Utilize Angular CLI para criar um novo projeto Angular.
    - **Exemplo Prático:**

```
bash
Copiar código
npm install -g @angular/cli
ng new meuApp
cd meuApp
ng serve
```

  - **Estrutura do Projeto:**
    - Conheça a estrutura básica do projeto gerado pelo Angular CLI.

5.2.2 Componentes Angular

- **Contextualização:**
  - Componentes em Angular são a base para construir a interface de usuário, semelhante aos componentes em React.
- **Conteúdo:**
  - **Criando Componentes:**
    - Defina e utilize componentes com `@Component`.
    - **Exemplo Prático:**

```
typescript
Copiar código
import { Component } from '@angular/core';

@Component({
  selector: 'app-meu-componente',
  template: `<h1>Olá, Angular!</h1>`,
```

```
})
export class MeuComponenteComponent { }
```

- **Propriedades e Binding:**
  - Passe dados e interaja com o DOM usando binding.
  - **Exemplo Prático:**

```
typescript
Copiar código
import { Component } from '@angular/core';

@Component({
  selector: 'app-contador',
  template: `
    <p>Você clicou {{ count }} vezes</p>
    <button (click)="incrementar()">Clique aqui</button>
  `,
})
export class ContadorComponent {
  count = 0;

  incrementar() {
    this.count++;
  }
}
```

5.2.3 Manipulação de Eventos no Angular

- **Contextualização:**
  - A manipulação de eventos em Angular é feita através de binding e diretivas específicas.
- **Conteúdo:**
  - **Eventos e Binding:**
    - Adicione manipuladores de eventos usando a sintaxe de binding.
    - **Exemplo Prático:**

```
typescript
Copiar código
import { Component } from '@angular/core';

@Component({
  selector: 'app-botao',
  template: `<button (click)="mostrarAlerta()">Clique
aqui</button>`,
})
export class BotaoComponent {
  mostrarAlerta() {
    alert('Botão clicado!');
  }
}
```

5.3 Introdução ao Vue.js (10 horas)

5.3.1 Configuração do Ambiente

- **Contextualização:**

    - Vue.js é um framework progressivo para construir interfaces de usuário. Configurar o ambiente é o primeiro passo para usar Vue.js.
- **Conteúdo:**
    - **Instalação com Vue CLI:**
        - Utilize Vue CLI para iniciar um novo projeto Vue.js.
        - **Exemplo Prático:**

```
bash
Copiar código
npm install -g @vue/cli
vue create meuApp
cd meuApp
npm run serve
```

    - **Estrutura do Projeto:**
        - Familiarize-se com a estrutura do projeto criado pelo Vue CLI.

5.3.2 Componentes Vue.js

- **Contextualização:**
    - Componentes em Vue.js permitem construir a interface do usuário de forma modular e reativa.
- **Conteúdo:**
    - **Criando Componentes:**
        - Defina e use componentes Vue.js com a sintaxe de arquivos `.vue`.
        - **Exemplo Prático:**

```
vue
Copiar código
<template>
  <h1>Olá, Vue.js!</h1>
</template>

<script>
export default {
  name: 'MeuComponente'
}
</script>
```

    - **Propriedades e Estado:**
        - Passe dados para componentes e gerencie o estado interno com `data`.
        - **Exemplo Prático:**

```
vue
Copiar código
<template>
  <div>
    <p>Você clicou {{ count }} vezes</p>
    <button @click="incrementar">Clique aqui</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
```

```
      count: 0
    }
  },
  methods: {
    incrementar() {
      this.count++;
    }
  }
}
</script>
```

5.3.3 Manipulação de Eventos no Vue.js

- **Contextualização:**
  - Vue.js simplifica a manipulação de eventos através da diretiva `v-on`.
- **Conteúdo:**
  - **Eventos e Manipuladores:**
    - Adicione manipuladores de eventos utilizando a diretiva `v-on`.
    - **Exemplo Prático:**

```
vue
Copiar código
<template>
  <button @click="mostrarAlerta">Clique aqui</button>
</template>

<script>
export default {
  methods: {
    mostrarAlerta() {
      alert('Botão clicado!');
    }
  }
}
</script>
```

Módulo 6: Bootstrap e Design Responsivo (30 horas)

Objetivo do Módulo:

Este módulo explora o Bootstrap, um framework front-end popular para o desenvolvimento de design responsivo e estilização rápida. Os tópicos incluem a configuração do Bootstrap, criação de layouts responsivos e personalização de componentes.

6.1 Introdução ao Bootstrap (10 horas)

6.1.1 Configuração do Bootstrap

- **Contextualização:**
  - O Bootstrap facilita a criação de sites responsivos e esteticamente agradáveis com uma variedade de componentes prontos para uso.
- **Conteúdo:**

- **Instalação via CDN:**
  - Inclua o Bootstrap diretamente no seu projeto usando um CDN.
  - **Exemplo Prático:**

```
html
Copiar código
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Meu Projeto Bootstrap</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/b
ootstrap.min.css">
</head>
<body>
    <h1>Olá, Bootstrap!</h1>
    <script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/um
d/popper.min.js"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/boo
tstrap.min.js"></script>
</body>
</html>
```

- **Instalação via NPM:**
  - Instale o Bootstrap usando o NPM para projetos mais complexos.
  - **Exemplo Prático:**

```
bash
Copiar código
npm install bootstrap
```

No arquivo `src/index.js`:

```
javascript
Copiar código
import 'bootstrap/dist/css/bootstrap.min.css';
```

6.1.2 Estrutura Básica do Bootstrap

- **Contextualização:**
  - Entender a estrutura e os principais componentes do Bootstrap é fundamental para aplicar o framework de forma eficaz.
- **Conteúdo:**
  - **Sistema de Grid:**
    - Utilize o sistema de grid para criar layouts responsivos.
    - **Exemplo Prático:**

```
html
Copiar código
<div class="container">
```

```
    <div class="row">
        <div class="col-md-4">Coluna 1</div>
        <div class="col-md-4">Coluna 2</div>
        <div class="col-md-4">Coluna 3</div>
    </div>
</div>
```

- **Componentes Básicos:**
    - Explore os componentes básicos como botões e alertas.
    - **Exemplo Prático:**

```
html
Copiar código
<button type="button" class="btn btn-primary">Botão
Primário</button>
<div class="alert alert-warning" role="alert">
    Alerta de Aviso!
</div>
```

6.2 Layouts Responsivos com Bootstrap (10 horas)

6.2.1 Sistema de Grid e Layouts

- **Contextualização:**
    - O sistema de grid do Bootstrap permite criar layouts flexíveis que se ajustam a diferentes tamanhos de tela.
- **Conteúdo:**
    - **Sistema de Grid Flexível:**
        - Trabalhe com diferentes tamanhos de colunas e offsets para criar layouts responsivos.
        - **Exemplo Prático:**

```
html
Copiar código
<div class="container">
    <div class="row">
        <div class="col-sm-6 col-md-4">Coluna 1</div>
        <div class="col-sm-6 col-md-4">Coluna 2</div>
        <div class="col-sm-6 col-md-4">Coluna 3</div>
    </div>
</div>
```

6.2.2 Utilizando Classes de Utilitários

- **Contextualização:**
    - Classes de utilitários do Bootstrap ajudam a ajustar rapidamente estilos e espaçamentos.
- **Conteúdo:**
    - **Espaçamento e Alinhamento:**
        - Use classes como `mt-3`, `mb-2`, `text-center` para ajustar o espaçamento e alinhamento.

- **Exemplo Prático:**

```
html
Copiar código
<div class="container mt-5">
    <div class="row">
        <div class="col text-center mb-4">
            <h2>Texto Centralizado</h2>
        </div>
    </div>
</div>
```

- **Cores e Tipografia:**
  - Aplique classes de cores e tipografia para estilizar o texto e os fundos.
  - **Exemplo Prático:**

```
html
Copiar código
<div class="bg-primary text-white p-3">
    <h1>Texto em Fundo Azul</h1>
</div>
```

6.2.3 Trabalhando com Navegação e Barreiras

- **Contextualização:**
  - O Bootstrap fornece componentes para criar navegação e barras de navegação responsivas.
- **Conteúdo:**
  - **Barra de Navegação:**
    - Crie barras de navegação usando o componente `navbar`.
    - **Exemplo Prático:**

```
html
Copiar código
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">Meu Site</a>
    <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarNav" aria-
controls="navbarNav" aria-expanded="false" aria-label="Toggle
navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
            <li class="nav-item active">
                <a class="nav-link" href="#">Início <span
class="sr-only">(atual)</span></a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Sobre</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Contato</a>
            </li>
        </ul>
    </div>
</nav>
```

6.3 Personalização e Temas com Bootstrap (10 horas)

6.3.1 Customizando o Bootstrap

- **Contextualização:**
    - Personalizar o Bootstrap permite ajustar o framework para atender às necessidades específicas do projeto.
- **Conteúdo:**
    - **Modificando Variáveis:**
        - Use Sass para modificar variáveis do Bootstrap e personalizar a aparência.
        - **Exemplo Prático:**

```
scss
Copiar código
@import "~bootstrap/scss/bootstrap";

$primary: #ff5733; // Altere a cor primária
@include bootstrap;
```

6.3.2 Criando Temas Customizados

- **Contextualização:**
    - A criação de temas customizados permite manter a consistência visual em aplicações complexas.
- **Conteúdo:**
    - **Criando um Tema:**
        - Defina estilos e componentes personalizados para criar um tema único.
        - **Exemplo Prático:**

```
scss
Copiar código
@import "~bootstrap/scss/bootstrap";

.custom-theme {
    background-color: #f0f0f0;
    .navbar {
        background-color: #333;
        .nav-link {
            color: #fff;
        }
    }
}
```

Módulo 7: Ferramentas de Desenvolvimento Front-End (30 horas)

Objetivo do Módulo:

Este módulo aborda ferramentas essenciais para o desenvolvimento front-end, incluindo controle de versão, build tools e gerenciamento de pacotes. Os tópicos incluem o uso de Git, Webpack, e ferramentas de linting e formatação.

7.1 Controle de Versão com Git (10 horas)

7.1.1 Introdução ao Git e GitHub

- **Contextualização:**
    - Git é um sistema de controle de versão distribuído que ajuda a gerenciar o histórico de alterações no código. GitHub é uma plataforma para hospedagem e colaboração de projetos Git.
- **Conteúdo:**
    - **Instalação e Configuração:**
        - Instale o Git e configure informações básicas.
        - **Exemplo Prático:**

          ```
          bash
          Copiar código
          git config --global user.name "Seu Nome"
          git config --global user.email "seuemail@example.com"
          ```

    - **Comandos Básicos:**
        - Comandos como `git init`, `git add`, `git commit`, e `git push` são fundamentais para o controle de versão.
        - **Exemplo Prático:**

          ```
          bash
          Copiar código
          git init
          git add .
          git commit -m "Primeiro commit"
          git remote add origin
          https://github.com/usuario/repositorio.git
          git push -u origin master
          ```

7.1.2 Branching e Merge

- **Contextualização:**
    - Branching e merge são essenciais para trabalhar em múltiplas funcionalidades simultaneamente e integrar alterações.
- **Conteúdo:**
    - **Criando Branches:**
        - Crie e gerencie branches para trabalhar em funcionalidades separadas.
        - **Exemplo Prático:**

          ```
          bash
          Copiar código
          ```

```
git checkout -b nova-funcionalidade
```

- **Merge de Branches:**
  - Integre alterações de diferentes branches.
  - **Exemplo Prático:**

```
bash
Copiar código
git checkout master
git merge nova-funcionalidade
```

7.2 Build Tools e Automatização (10 horas)

7.2.1 Webpack

- **Contextualização:**
  - Webpack é uma ferramenta de build que permite empacotar módulos JavaScript e outros recursos.
- **Conteúdo:**
  - **Configuração Básica:**
    - Configure o Webpack para um projeto simples.
    - **Exemplo Prático: Arquivo `webpack.config.js`:**

```
javascript
Copiar código
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  }
};
```

  - **Uso de Loaders e Plugins:**
    - Adicione loaders e plugins para processar arquivos e otimizar o build.
    - **Exemplo Prático: Configuração para Babel:**

```
bash
Copiar código
npm install --save-dev babel-loader @babel/core @babel/preset-
env
```

Atualização no `webpack.config.js`:

```
javascript
Copiar código
module.exports = {
  // ...configuração existente
  module: {
    rules: [
      {
```

```
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: ['@babel/preset-env']
          }
        }
      }
    ]
  }
};
```

7.2.2 Ferramentas de Linting e Formatação

- **Contextualização:**
  - Linting e formatação ajudam a manter a qualidade e consistência do código.
- **Conteúdo:**
  - **ESLint:**
    - Configure o ESLint para garantir a qualidade do código JavaScript.
    - **Exemplo Prático:**

      ```
      bash
      Copiar código
      npm install eslint --save-dev
      npx eslint --init
      ```

  - **Prettier:**
    - Use Prettier para formatar automaticamente o código.
    - **Exemplo Prático:**

      ```
      bash
      Copiar código
      npm install --save-dev prettier
      echo {}> .prettierrc.json
      ```

7.3 Gerenciamento de Pacotes com NPM e Yarn (10 horas)

7.3.1 Introdução ao NPM e Yarn

- **Contextualização:**
  - NPM e Yarn são ferramentas para gerenciar pacotes JavaScript e suas dependências.
- **Conteúdo:**
  - **NPM:**
    - Comandos básicos para gerenciar pacotes e scripts.
    - **Exemplo Prático:**

      ```
      bash
      Copiar código
      npm init
      npm install lodash
      npm run start
      ```

- **Yarn:**
    - Comandos básicos e diferenças em relação ao NPM.
    - **Exemplo Prático:**

```
bash
Copiar código
yarn init
yarn add lodash
yarn start
```

Módulo 8: Performance e Otimização Front-End (20 horas)

Objetivo do Módulo:

Este módulo cobre técnicas e melhores práticas para otimização de desempenho em aplicações front-end. Inclui tópicos sobre otimização de recursos, lazy loading e boas práticas para melhorar o tempo de carregamento.

8.1 Otimização de Imagens e Recursos (10 horas)

8.1.1 Compressão de Imagens

- **Contextualização:**
    - Imagens não otimizadas podem afetar negativamente o desempenho da aplicação.
- **Conteúdo:**
    - **Ferramentas de Compressão:**
        - Utilize ferramentas como ImageOptim e TinyPNG para reduzir o tamanho das imagens.
        - **Exemplo Prático:**
            - **TinyPNG:** tinyjpg.com

8.1.2 Minificação e Compressão de Arquivos

- **Contextualização:**
    - Minificar e comprimir arquivos CSS e JavaScript reduz o tempo de carregamento.
- **Conteúdo:**
    - **Usando Webpack:**
        - Configure plugins para minificar arquivos.
        - **Exemplo Prático: Arquivo `webpack.config.js`:**

```
javascript
Copiar código
const TerserPlugin = require('terser-webpack-plugin');

module.exports = {
  // ...configuração existente
  optimization: {
    minimize: true,
```

```
    minimizer: [new TerserPlugin()],
  },
};
```

8.2 Técnicas de Lazy Loading (5 horas)

8.2.1 Lazy Loading de Imagens e Componentes

- **Contextualização:**
  - Lazy loading carrega recursos apenas quando são necessários, melhorando a performance.
- **Conteúdo:**
  - **Imagens com Lazy Loading:**
    - Use a API de `IntersectionObserver` para carregar imagens somente quando entram na viewport.
    - **Exemplo Prático:**

```
html
Copiar código
<img data-src="imagem.jpg" class="lazyload" alt="Imagem">
<script>
  document.addEventListener("DOMContentLoaded", function() {
    const lazyImages =
[].slice.call(document.querySelectorAll("img.lazyload"));

    if ("IntersectionObserver" in window) {
      let lazyImageObserver = new
IntersectionObserver(function(entries, observer) {
        entries.forEach(function(entry) {
          if (entry.isIntersecting) {
            let lazyImage = entry.target;
            lazyImage.src = lazyImage.dataset.src;
            lazyImage.classList.remove("lazyload");
            lazyImageObserver.unobserve(lazyImage);
          }
        });
      });

      lazyImages.forEach(function(lazyImage) {
        lazyImageObserver.observe(lazyImage);
      });
    } else {
      // Fallback para navegadores que não suportam
IntersectionObserver
      lazyImages.forEach(function(lazyImage) {
        lazyImage.src = lazyImage.dataset.src;
        lazyImage.classList.remove("lazyload");
      });
    }
  });
</script>
```

8.2.2 Lazy Loading de Componentes em Frameworks

- **Contextualização:**

    - Em frameworks como React e Vue.js, o lazy loading de componentes pode melhorar a performance.
- **Conteúdo:**
    - **React:**
        - Utilize `React.lazy` e `Suspense` para carregar componentes sob demanda.
        - **Exemplo Prático:**

```
javascript
Copiar código
import React, { Suspense, lazy } from 'react';

const MeuComponente = lazy(() => import('./MeuComponente'));

function App() {
  return (
    <div>
      <Suspense fallback={<div>Carregando...</div>}>
        <MeuComponente />
      </Suspense>
    </div>
  );
}
```

    - **Vue.js:**
        - Use a função `defineAsyncComponent` para carregamento assíncrono.
        - **Exemplo Prático:**

```
javascript
Copiar código
import { defineAsyncComponent } from 'vue';

export default {
  components: {
    MeuComponente: defineAsyncComponent(() =>
      import('./MeuComponente.vue')
    )
  }
};
```

8.3 Boas Práticas de Performance (5 horas)

8.3.1 Melhoria do Tempo de Carregamento

- **Contextualização:**
    - Adotar boas práticas de performance pode melhorar significativamente o tempo de carregamento da aplicação.
- **Conteúdo:**
    - **Minificação de CSS e JS:**
        - Utilize ferramentas para minificar e otimizar arquivos CSS e JS.
    - **Caching de Recursos:**

- Configure o cache para recursos estáticos para reduzir o tempo de carregamento em visitas subsequentes.
- **Exemplo Prático: Configuração de Cache:**

```
html
Copiar código
<meta http-equiv="Cache-Control" content="max-age=31536000">
```