

PROGRAMAÇÃO FRONT-END 150 HORAS

Capacidades Técnicas

1. Utilizar semântica de linguagem de marcação conforme normas:

Consiste em estruturar o código HTML de forma semântica, garantindo acessibilidade, SEO otimizado e melhor manutenção do código. Isso envolve o uso adequado de elementos como `<header>`, `<article>`, `<section>`, `<nav>`, `<aside>`, entre outros, conforme as diretrizes do W3C e boas práticas de desenvolvimento web.

Aula: Semântica na Linguagem de Marcação (HTML)

Slide 1 - Capa

- Título: **Semântica na Linguagem de Marcação (HTML)**
- Imagem ilustrativa
- Nome do instrutor

♦ Slide 2 - Objetivo da Aula

- Explicar a importância da semântica no HTML
- Demonstrar boas práticas de marcação
- Mostrar exemplos de código correto e incorreto

♦ Slide 3 - O que é Semântica?

- Definição de semântica
- Exemplo prático com linguagem humana (frase ambígua vs. frase clara)

♦ Slide 4 - Semântica no HTML

- O HTML semântico define significado aos elementos
- Exemplo: `<header>`, `<article>`, `<section>`

♦ Slide 5 - Por que usar HTML Semântico?

- Melhor acessibilidade

- Melhor indexação por motores de busca (SEO)
- Código mais organizado e legível

♦ Slide 6 - Exemplo de Código Semântico

- Código correto usando `<header>`, `<nav>`, `<article>`, `<footer>`
- Código semântico vs. código não semântico (`<div>` para tudo)

♦ Slide 7 - Elementos Semânticos Comuns

- Tabela com elementos e suas funções:
 - `<header>` - Cabeçalho
 - `<nav>` - Navegação
 - `<main>` - Conteúdo principal
 - `<section>` - Seção de conteúdo
 - `<article>` - Artigo independente

♦ Slide 8 - Elementos Não Semânticos

- `<div>` e `` são úteis, mas sem significado próprio
- Quando usá-los corretamente

♦ Slide 9 - Como a Semântica Afeta o SEO?

- Motores de busca analisam a estrutura da página
- Melhoria no ranqueamento ao usar HTML semântico corretamente

♦ Slide 10 - Como a Semântica Afeta a Acessibilidade?

- Leitores de tela identificam melhor os elementos
- Exemplo prático com `<button>` vs. `<div>` com evento de clique

♦ Slide 11 - Exemplo de Página Semântica

- Estrutura de uma página web bem organizada

♦ Slide 12 - Boas Práticas de HTML Semântico

- Usar tags apropriadas
- Evitar `<div>` e `` desnecessários
- Manter a hierarquia lógica

♦ Slide 13 - Caso Prático: Antes e Depois

- Página com `<div>` para tudo
- Página reformulada com HTML semântico

♦ Slide 14 - Erros Comuns e Como Corrigi-los

- Usar `<section>` sem título adequado
- Usar `<article>` para conteúdos não independentes
- Trocar `<div class="menu">` por `<nav>`

♦ Slide 15 - Exercício 1: Identifique o Erro

- Exibir um código errado e pedir para os alunos corrigirem

♦ Slide 16 - Exercício 2: Reescrevendo Código

- Página simples escrita sem semântica
- Alunos devem transformar para HTML semântico

♦ Slide 17 - Ferramentas Úteis

- Validador do W3C
- Lighthouse (Google Chrome)

♦ Slide 18 - Resumo da Aula

- Revisão dos principais conceitos

♦ Slide 19 - Discussão e Dúvidas

- Momento para perguntas

♦ Slide 20 - Próximos Passos

- Praticar refatoração de código
- Aplicar boas práticas em projetos reais

2. Elaborar formulários de página web:

Inclui a criação de formulários interativos usando HTML e aprimorados com CSS e JavaScript. Envolve a utilização de elementos como `<input>`, `<select>`, `<textarea>`, `<button>`, além da implementação de validação (HTML5, JavaScript) e integração com back-end para envio de dados.

Aula 1: Formulários em Páginas Web

1. Capa
2. Objetivo da aula
3. O que são formulários no HTML?
4. Elementos básicos de formulários (`<input>`, `<label>`, `<select>`, `<textarea>`)
5. Tipos de `<input>` e suas aplicações
6. Atributos essenciais (`required`, `placeholder`, `pattern`, `maxlength`)
7. Validação de formulários (HTML5 vs. JavaScript)
8. Organização de formulários com `<fieldset>` e `<legend>`
9. Melhorando a experiência com formulários acessíveis
10. Uso de máscaras em campos (exemplo: CPF, telefone)
11. Envio de formulários (métodos `GET` e `POST`)
12. Integrando formulários com APIs
13. Estilização de formulários com CSS
14. Utilização de bibliotecas para formulários dinâmicos (Formik, React Hook Form)
15. Segurança em formulários (CSRF, XSS)
16. Caso prático: Criando um formulário completo
17. Erros comuns e como evitá-los
18. Exercício prático: Criando um formulário validado
19. Resumo da aula
20. Dúvidas e discussão

3. Utilizar ferramentas gráficas para interface web e mobile (4 horas):

Refere-se ao uso de softwares e ferramentas para prototipação e design de interfaces, como Figma, Adobe XD, Sketch e Photoshop. Isso inclui a criação de wireframes, mockups e design responsivo para garantir uma boa experiência do usuário.

Aula 2: Ferramentas Gráficas para Interface Web e Mobile

1. Capa
2. Objetivo da aula
3. O que são ferramentas gráficas para UI/UX?
4. Principais ferramentas (Figma, Adobe XD, Sketch, Photoshop)
5. Diferença entre Wireframe, Mockup e Protótipo
6. Criando um wireframe básico
7. Trabalhando com grid e layout responsivo
8. Design System: o que é e como usar
9. Uso de tipografia e cores no design
10. Exportação de assets para web e mobile
11. Criando componentes reutilizáveis no Figma
12. Plugins úteis para UI Design
13. Testes de usabilidade com protótipos
14. Exportando layouts para código HTML/CSS
15. Diferenças entre UI para web e mobile
16. Adaptação de interfaces para diferentes resoluções
17. Uso de animações e microinterações no design
18. Caso prático: Criando um design no Figma
19. Exercício prático: Criar um layout básico
20. Resumo da aula e dúvidas

4. Adequar a interface web para diferentes dispositivos de acesso:

Trata-se da implementação de técnicas de design responsivo, como Media Queries, uso de unidades relativas (**em**, **rem**, **%**, **vh**, **vw**), Flexbox, CSS Grid e frameworks

como Bootstrap e Tailwind CSS. O objetivo é garantir que a interface funcione corretamente em desktops, tablets e smartphones.

Aula 3: Interfaces Web Responsivas

1. Capa
2. Objetivo da aula
3. O que é design responsivo?
4. Importância da responsividade na web
5. Unidades de medida flexíveis (`em`, `rem`, `%`, `vh`, `vw`)
6. Media Queries: adaptando o layout
7. Mobile-first vs. Desktop-first
8. Uso de CSS Grid e Flexbox
9. Frameworks responsivos (Bootstrap, Tailwind)
10. Imagens e vídeos responsivos (`srcset`, `picture`)
11. Testando responsividade no DevTools
12. Ajustando tipografia para diferentes telas
13. Performance em layouts responsivos
14. Princípios de acessibilidade no design responsivo
15. Diferenças entre layouts responsivos e adaptativos
16. Grid system e colunas flexíveis
17. Exemplo prático: Construindo um site responsivo
18. Erros comuns e soluções
19. Exercício prático: Criar uma landing page responsiva
20. Resumo da aula e dúvidas

5. Desenvolver interfaces web interativas com linguagem de programação:

Envolve o uso de JavaScript (ou TypeScript) para criar interatividade em páginas web. Isso pode incluir manipulação do DOM, eventos, animações, Single Page Applications (SPAs) com React, Vue.js ou Angular, e integração com APIs para carregamento dinâmico de conteúdo.

Aula 5: Desenvolvimento de Interfaces Interativas

1. Capa
2. Objetivo da aula
3. O que são interfaces interativas?
4. Introdução ao JavaScript para interatividade
5. Manipulação do DOM
6. Eventos de usuário (`click`, `hover`, `keydown`)
7. Criando animações com CSS e JavaScript
8. Efeitos de transição e transformação
9. Uso de bibliotecas como GSAP e Framer Motion
10. Melhorando a experiência do usuário com microinterações
11. Formulários dinâmicos com JavaScript
12. Modais e pop-ups interativos
13. Notificações e feedbacks visuais
14. Implementação de carrosséis e sliders
15. Uso de componentes dinâmicos com React
16. Criando Single Page Applications (SPAs)
17. Testando interatividade com DevTools
18. Exemplo prático: Criando uma interface interativa
19. Exercício prático: Criar uma página com interações
20. Resumo da aula e dúvidas

6. Aplicar técnicas de estilização de páginas web:

Abrange o uso de CSS puro ou pré-processadores como SASS e LESS para estilização avançada. Também inclui a aplicação de animações com CSS (`@keyframes`, `transition`, `transform`), uso de variáveis CSS, design system, temas personalizados e frameworks de estilização como Tailwind CSS e MUI.

Aula 6: Desenvolvimento com Frameworks

1. Capa
2. Objetivo da aula
3. O que são frameworks de front-end?
4. Diferenças entre React, Vue e Angular

5. Por que usar frameworks?
6. Estrutura básica de um projeto React
7. Componentização e reutilização de código
8. Gerenciamento de estado (Context API, Redux)
9. React Router para navegação dinâmica
10. Consumo de APIs em React
11. Hooks básicos (`useState`, `useEffect`)
12. Estilização em frameworks (Styled Components, Tailwind)
13. Otimização de performance
14. Testando componentes em frameworks
15. Integração com bibliotecas externas
16. Desenvolvimento mobile com React Native
17. Melhorando acessibilidade em frameworks
18. Exemplo prático: Criando um componente reutilizável
19. Exercício prático: Criar um mini-projeto com React
20. Resumo da aula e dúvidas

7. Desenvolver interfaces web utilizando frameworks:

Diz respeito ao uso de frameworks modernos como React, Angular e Vue.js para a construção de interfaces eficientes e modulares. Isso inclui conceitos como componentização, state management (Redux, Context API, Pinia) e otimização de performance (Lazy Loading, Server-side Rendering).

Aula 7: APIs e Experiência do Usuário (UX/UI)

1. Capa
2. Objetivo da aula
3. O que são APIs e por que utilizá-las?
4. Diferença entre REST e GraphQL
5. Como consumir APIs no front-end
6. Métodos HTTP e suas funções
7. Trabalhando com `fetch()` e Axios
8. Tratamento de erros em requisições
9. Exemplo prático: Consumo de uma API de filmes



10. Melhorando a performance de chamadas API
11. Cache e otimização com SWR e React Query
12. O que é UX e como impacta o usuário?
13. Diferença entre UX e UI
14. Princípios de design centrado no usuário
15. Heurísticas de Nielsen para UX
16. Testes de usabilidade e acessibilidade
17. Melhores práticas para formulários acessíveis
18. Exemplo prático: Melhorando a usabilidade de um site
19. Exercício prático: Criar uma API fictícia e consumir no front-end
20. Resumo da aula e dúvidas

8. Desenvolver interfaces web consumindo API:

Consiste na integração de aplicações web com APIs RESTful ou GraphQL para buscar, enviar e atualizar dados. Isso envolve o uso de `fetch()`, Axios, Apollo Client, WebSockets e autenticação via OAuth, JWT ou cookies para garantir segurança na comunicação.




Aula: Desenvolvimento de Interfaces Web Consumindo APIs

Slide 1 – Capa

-  **Título:** Desenvolvimento de Interfaces Web Consumindo APIs
 -  **Subtítulo:** RESTful, GraphQL, WebSockets e Autenticação
-

Introdução às APIs

Slide 2 – Objetivo da Aula

-  Entender o que são APIs e como consumi-las
-  Utilizar `fetch()`, Axios e Apollo Client
-  Integrar RESTful e GraphQL

- 📌 Implementar autenticação via JWT, OAuth e cookies
- 📌 Trabalhar com WebSockets para comunicação em tempo real

Slide 3 – O que é uma API?

- 📌 API significa **Application Programming Interface**
- 📌 Permite a comunicação entre diferentes sistemas
- 📌 Tipos: **RESTful, GraphQL e WebSockets**

Slide 4 – API RESTful

- 📌 Baseada no protocolo **HTTP**
- 📌 Usa métodos: **GET, POST, PUT, DELETE**
- 📌 Exemplo de requisição:

json

CopiarEditar

```
GET https://api.exemplo.com/usuarios
```

Slide 5 – API GraphQL

- 📌 Alternativa ao REST
- 📌 Permite buscar apenas os dados necessários
- 📌 Exemplo de requisição GraphQL:

graphql

CopiarEditar

```
query {  
  
  user(id: 1) {  
  
    name
```

```
    email

  }

}
```

Slide 6 – WebSockets

- 📌 Comunicação **bidirecional** em tempo real
 - 📌 Útil para chats, notificações e atualizações dinâmicas
 - 📌 Exemplo: Socket.io no Node.js
-

📌 Consumo de APIs no Frontend

Slide 7 – Métodos de Requisição HTTP

- 📌 **GET** – Buscar dados
- 📌 **POST** – Criar um novo recurso
- 📌 **PUT/PATCH** – Atualizar um recurso
- 📌 **DELETE** – Excluir um recurso

Slide 8 – Fazendo Requisições com **fetch()**

- 📌 **fetch()** é a API nativa do JavaScript
- 📌 Exemplo de requisição:

```
js
```

CopiarEditar

```
fetch('https://api.exemplo.com/dados')

  .then(response => response.json())
```

```
.then(data => console.log(data))

.catch(error => console.error('Erro:', error));
```

Slide 9 – Fazendo Requisições com Axios

📌 Axios simplifica requisições HTTP

📌 Exemplo de uso:

js

CopiarEditar

```
import axios from 'axios';

axios.get('https://api.exemplo.com/dados')

  .then(response => console.log(response.data))

  .catch(error => console.error('Erro:', error));
```

Slide 10 – Consumindo GraphQL com Apollo Client

📌 Exemplo de consulta GraphQL em React:

js

CopiarEditar

```
import { gql, useQuery } from '@apollo/client';
```

```
const GET_USERS = gql`
```

```
  query {
```

```
    users {
```

```
      id
```

```
      name
```

```
    }
```

```
  }
```

```
`;
```

```
const Users = () => {
```

```
  const { data, loading, error } = useQuery(GET_USERS);
```

```
  if (loading) return <p>Carregando...</p>;
```

```
  if (error) return <p>Erro ao buscar dados!</p>;
```


```
    return <ul>{data.users.map(user => <li  
key={user.id}>{user.name}</li>)}</ul>;
```

```
};
```


Autenticação e Segurança

Slide 11 – Autenticação com JWT

 JSON Web Token (JWT) é amplamente usado para autenticação segura

 Passos:

1. Usuário faz login
2. API gera um **token JWT**
3. O frontend armazena o token (localStorage, cookies)
4. O token é enviado em todas as requisições

 Exemplo de envio de JWT:

js


CopiarEditar

```
axios.get('https://api.exemplo.com/usuario', {  
  
  headers: { Authorization: `Bearer ${token}` }  
  
});
```

Slide 12 – Autenticação com OAuth

 OAuth permite login com Google, Facebook, GitHub

 Usuário é redirecionado para autenticação

 API retorna um **token de acesso**



Slide 13 – Uso de Cookies para Autenticação

 Cookies armazenam tokens no navegador


 Melhor para proteger contra ataques **XSS**

Trabalhando com WebSockets

Slide 14 – Introdução ao WebSocket

-  Conexão persistente entre cliente e servidor
-  Exemplo de uso: chats, jogos online, notificações

Slide 15 – Exemplo de WebSocket com JavaScript

-  Criando uma conexão WebSocket:

js

CopiarEditar

```
const socket = new WebSocket('wss://servidor.com');
```

```
socket.onmessage = (event) => {  
  
    console.log('Mensagem recebida:', event.data);  
  
};
```

```
socket.send('Olá, servidor!');
```

Prática e Testes

Slide 16 – Testando APIs com Postman

- 📌 O Postman permite enviar requisições e visualizar respostas
- 📌 Passos básicos para testar uma API

Slide 17 – Erros Comuns e Como Corrigir

- 📌 **CORS bloqueado** – Solução: configurar no backend
 - 📌 **Erro 401 (Não autorizado)** – Verificar autenticação
 - 📌 **Erro 404 (Não encontrado)** – Endpoint incorreto
-

📌 Casos Práticos

Slide 18 – Criando um CRUD com API

- 📌 **Create** – Criar usuário
 - 📌 **Read** – Buscar usuários
 - 📌 **Update** – Atualizar usuário
 - 📌 **Delete** – Excluir usuário
- 📌 Exemplo de criação com Axios:

js

CopiarEditar

```
axios.post('https://api.exemplo.com/usuarios', {  
  
  name: "João",  
  
  email: "joao@email.com"  
  
});
```

Slide 19 – Exercício Prático

- 📌 Criar uma aplicação React que consome uma API pública
 - 📌 Exibir dados na tela e implementar um formulário para envio de informações
-

📌 Conclusão e Próximos Passos

Slide 20 – Resumo da Aula

- 📌 O que aprendemos:
 - ✅ O que são APIs e como consumi-las
 - ✅ Diferenças entre RESTful, GraphQL e WebSockets
 - ✅ Como usar `fetch()`, Axios e Apollo Client
 - ✅ Autenticação segura com JWT e OAuth
 - ✅ Comunicação em tempo real com WebSockets

📌 Próximos Passos:

- 🚀 Praticar o consumo de APIs com React
- 🚀 Implementar autenticação JWT em projetos reais
- 🚀 Criar um app interativo usando WebSockets

9. Diferenciar os aspectos de aplicabilidade entre as experiências do usuário (UX) e a interface do usuário (UI):

Significa compreender a diferença entre UX (experiência do usuário) e UI (interface do usuário). UX foca na usabilidade, acessibilidade, jornada do usuário e testes A/B, enquanto UI envolve o design visual, tipografia, paleta de cores e layout. Saber diferenciar e aplicar esses conceitos é essencial para criar interfaces intuitivas e eficientes.