DESENVOLVIMENTO DE APLICAÇÕES WEB COM ORIENTAÇÃO A OBJETOS

LABORATÓRIO DE COMPUTAÇÃO CIENTÍFICA

Curso Superior de Tecnologia em Sistemas para Internet

Campus Nordeste UnU - Posse

Universidade Estadual de Goiás

RONALDO FERREIRA DA SILVA
APARECIDO ALVES DA SILVA JÚNIOR

DESENVOLVIMENTO DE APLICAÇÕES WEB COM ORIENTAÇÃO A OBJETOS

Esta apostila foi elaborada para servir como material didático do curso de extensão sobre Desenvolvimento de Aplicações Web com Orientação a Objetos, desenvolvido pela Universidade Estadual de Goiás (UEG) – Unidade Universitária de Posse, voltado para estudantes do Ensino Médio. Ela aborda, de forma gradual e ilustrada (com exemplos práticos), os principais conceitos da Programação Orientada a Objetos aplicada ao desenvolvimento de aplicações Web. O conteúdo está distribuído em módulos, acompanhando a ementa e cronograma do curso. Cada módulo incluirá explicações detalhadas, exemplos práticos, imagens e exercícios.

Posse-GO
Junho/2025

“Eu acredito que, às vezes, são as pessoas que ninguém espera nada que fazem as coisas que ninguém consegue imaginar. ”

Alan Turing *– matemático e cientista, considerado o pai da computação.*

FICHA CATALOGRÁFICA

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da UEG
com os dados fornecidos pelo(a) autor(a).

FD451
d

Ferreira da Silva, Ronaldo
Desenvolvimento de Aplicações Web com Orientação a Objetos / Ronaldo Ferreira da Silva; orientador Ronaldo Ferreira da Silva; co-orientador Aparecido Alves da Silva Júnior. -- Posse, 2025.
166 p.

Graduação - Sistemas Para Internet -- Unidade de Posse, Universidade Estadual de Goiás, 2025.

1. Programação Web. 2. Orientação a Objetos. 3. Curso de Extensão. I. Ferreira da Silva, Ronaldo, orient. II. Alves da Silva Júnior, Aparecido, co-orient. III. Título.

A UNIVERSIDADE ESTADUAL DE GOIÁS (UEG)

A Universidade Estadual de Goiás (UEG) é uma universidade pública multicampi do Estado de Goiás, criada pela Lei Estadual 13.456, de 16 de abril de 1999.

Nos termos do seu Estatuto, aprovado pelo Decreto Estadual nº 9.593, de 17 de janeiro de 2020 e do Regimento Geral aprovado por seu Conselho Universitário, a UEG é uma instituição de ensino, pesquisa e extensão com finalidade científica e tecnológica, de natureza cultural e educacional, com caráter público, gratuito e laico. Trata-se de uma autarquia do poder executivo do Estado de Goiás, com autonomia didático-científica, administrativa e de gestão financeira e patrimonial, nos termos do Artigo 207 da Constituição da República Federativa do Brasil, do Artigo 161 da Constituição do Estado de Goiás e da Lei Estadual nº 18.971, de 23 de julho de 2015. Rege-se por seu Estatuto, seu Regimento Geral e por suas normas complementares.

Universidade
Estadual de Goiás

EXTENSÃO UNIVERSITÁRIA

"A Extensão Universitária, sob o princípio constitucional da indissociabilidade entre ensino, pesquisa e extensão, é um processo interdisciplinar, educativo, cultural, científico e político que promove a interação transformadora entre Universidade e outros setores da sociedade."

Este material foi desenvolvido no escopo de um curso de extensão vinculado ao Curso Superior de Tecnologia em Sistemas para Internet denominado **Desenvolvimento de Aplicações Web com Orientação a Objetos**, sob a coordenação do professor Ronaldo Ferreira da Silva e colaboração do técnico Aparecido Alves da Silva Júnior.

CURSO DE EXTENSÃO

O curso de extensão **Desenvolvimento de Aplicações Web com Orientação a Objetos** tem como objetivo introduzir estudantes do ensino médio ao universo da programação e da construção de aplicações Web modernas, com foco em conceitos de Orientação a Objetos. Ao longo de 60 horas, os participantes terão contato com fundamentos da Web, linguagens de marcação e estilo (HTML e CSS), lógica de programação, linguagem de programação *client-side* (JavaScript) e desenvolvimento de aplicações dinâmicas com uma linguagem *back-end* orientada a objetos utilizando PHP. O curso combina teoria e prática, culminando em um projeto final que reforça a aprendizagem por meio da aplicação dos conhecimentos adquiridos. Ao final, os alunos estarão aptos a compreender e desenvolver aplicações *Web* simples, com estruturas bem organizadas, promovendo o pensamento lógico, a criatividade e o interesse por carreiras na área de tecnologia.

ORGANIZAÇÃO DESTA APOSTILA

Esta apostila apresenta de forma incremental e gradual conceitos básicos sobre desenvolvimento de aplicações Web com Orietnação Objetos.

Módulo 1: Introdução à Web

1.1 O que é a Web?

- **Conteúdo:** Explicação do conceito de World Wide Web, sua história breve e como difere da internet.
- **Ilustrações:** Diagrama simples mostrando a interconexão de dispositivos, talvez uma linha do tempo básica da web.

1.2 Funcionamento básico da Web (cliente e servidor)

- **Conteúdo:** Detalhamento do modelo cliente-servidor. Como o navegador (cliente) solicita informações e o servidor as entrega.
- **Ilustrações:** Diagramas de fluxo mostrando a comunicação entre cliente e servidor.
- **Exemplo prático (Conceitual):** Um cenário de usuário acessando um site de notícias.

1.3 Protocolo HTTP e URLs

- **Conteúdo:** Explicação de HTTP (*Hypertext Transfer Protocol*) como a linguagem da web. Estrutura de uma URL (*Uniform Resource Locator*) e seus componentes (protocolo, domínio, caminho, etc.).
- **Ilustrações:** Quebra de uma URL em seus componentes visuais.
- **Exemplo de código (Conceitual/Pseudo):** Uma requisição HTTP GET simples para ilustrar o conceito.
- **Exercícios:** Identificação de componentes em diferentes URLs.

1.4 Navegadores e servidores

- **Conteúdo:** Papel dos navegadores (Chrome, Firefox, Edge) como intérpretes de código e clientes web. Papel dos servidores web (Apache, Nginx) no hospedamento e entrega de conteúdo.
- **Ilustrações:** Ícones de navegadores e servidores.
- **Comentários:** Foco na interação do usuário com o navegador.

1.5 Introdução ao HTML e estrutura de uma página

- **Conteúdo:** O que é HTML (Hypertext Markup Language). Estrutura básica de um documento HTML, elementos e tags comuns.
- **Ilustrações:** Renderização da página HTML no navegador.
- **Exercícios:** Criar uma página HTML simples com diferentes elementos.

Módulo 2: Fundamentos da Programação

2.1 Lógica de programação: conceitos iniciais

- **Conteúdo:** Algoritmos, pseudocódigo, fluxogramas. Sequência, decisão e repetição.
- **Ilustrações:** Exemplos de fluxogramas simples.
- **Exemplo prático (Pseudocódigo):** Um algoritmo para calcular a média de duas notas.
- **Exercícios:** Criar pseudocódigos para problemas do dia a dia.

2.2 Variáveis, operadores e expressões

- **Conteúdo:** Tipos de dados (números, texto/strings, booleanos). Declaração e atribuição de variáveis. Operadores aritméticos, relacionais e lógicos. Precedência de operadores.
- **Exercícios:** Escrever expressões que resultem em valores específicos.

2.3 Estruturas condicionais (if, else, switch)

- **Conteúdo:** Tomada de decisões com if, else if, else. Múltiplas escolhas com switch.
- **Fluxogramas:** Ilustrando as estruturas condicionais.
- **Exercícios:** Implementar condições para diferentes cenários.

2.4 Laços de repetição (for, while)

- **Conteúdo:** Repetição de código com for (para um número conhecido de iterações) e while (para repetições baseadas em uma condição).
- **Fluxogramas:** Ilustrando os laços de repetição.
- **Exercícios:** Usar laços para exibir sequências de números ou repetir ações.

2.5 Funções e escopo de variáveis

- **Conteúdo:** O que são funções e por que usá-las (reusabilidade, organização). Declaração e chamada de funções. Parâmetros e retorno de valores. Escopo de variáveis (local e global).
- **Ilustrações:** Diagramas mostrando o fluxo de execução de uma função e o alcance das variáveis.

- **Exercícios:** Criar funções para tarefas específicas

Módulo 3: Introdução ao CSS (*Cascading Style Sheets* - Folhas de Estilo em Cascata)

- **Conteúdo:** O que é CSS e por que usá-lo. Seletores (tag, id, class). Propriedades de estilo (cor, fonte, tamanho, espaçamento). Box model. Inclusão de CSS (inline, interno, externo).
- **Exemplos de código (HTML/CSS):** Página HTML com estilos aplicados.
- **Ilustrações:** Box model visual, exemplos de elementos estilizados.

Módulo 4: Introdução ao Bootstrap (Framework CSS)

- **Conteúdo:** Conceitos do framework baseado em CSS para criação do front-end da aplicação. Integração do Bootstrap ao projeto prático
- **Exemplos de código:** Aplicação das classes para estilização e utilização dos componentes prontos.

Módulo 5: Fundamentos de JavaScript para Web

- **Conteúdo:** O DOM (*Document Object Model*). Manipulação de elementos HTML com JavaScript. Eventos (click, submit, keyup). Validação de formulários simples. Introdução a AJAX (requisições assíncronas).
- **Exemplos de código (HTML/CSS/JavaScript):** Formulário simples com validação, mudança de conteúdo na página via JS.

Módulo 6: Programação Orientada a Objetos (POO)

- **Conteúdo:** Conceitos de POO: Classes, Objetos, Atributos, Métodos. Pilares da POO: Encapsulamento, Herança, Polimorfismo, Abstração.
- **Exemplos de código (PHP):** Definição de classes e criação de objetos. Exemplo de herança simples.

Módulo 7: Introdução ao PHP e programação no lado do servidor

- **Conteúdo:** O que é PHP. Sintaxe básica. Variáveis, tipos de dados. Operadores. Estruturas de controle de fluxo. Manipulação de formulários (GET/POST). Inclusão de arquivos.
- **Exemplos de código (PHP):** Páginas PHP simples que processam dados de formulários HTML.

Módulo 8: Introdução ao CodeIgniter (Framework PHP)

- **Conteúdo:** O que é um framework. Visão geral do CodeIgniter (MVC - Model-View-Controller). Instalação e configuração básica. Roteamento. Controladores, Views.
- **Exemplos de código (PHP com CodeIgniter):** Uma aplicação "Olá, Mundo" em CodeIgniter.

Módulo 9: Banco de Dados e SQL

- **Conteúdo:** O que é um banco de dados relacional. Modelagem de dados (tabelas, colunas, chaves primárias/estrangeiras). Introdução ao SQL (SELECT, INSERT, UPDATE, DELETE).
- **Exemplos de código (SQL):** Criação de tabelas, inserção de dados, consultas básicas.

Módulo 10: Integração do PHP e CodeIgniter com Banco de Dados

- **Conteúdo:** Conexão com banco de dados em PHP. Utilizando a camada de banco de dados do CodeIgniter. Operações CRUD (Create, Read, Update, Delete) com CodeIgniter e MySQL/SQLite.
- **Exemplos de código (PHP com CodeIgniter e Banco de Dados):** Uma pequena aplicação de listagem/cadastro de itens.

Módulo 11: Segurança básica na Web

- **Conteúdo:** SQL Injection, XSS (Cross-Site Scripting). Medidas de prevenção. Senhas seguras (hashing).
- **Comentários:** Foco em boas práticas.

Módulo 12: Controle de Versão com Git

- **Conteúdo:** O que é controle de versão. Conceitos básicos do Git (repositório, commit, branch, merge). Comandos básicos (git init, add, commit, push, pull).
- **Exemplos práticos:** Cenários de uso do Git para desenvolvimento individual e em equipe (conceitual).

Módulo 13: Projeto prático: Sistema de Controle de Ocorrências

- **Conteúdo:** Aplicação de todos os conceitos aprendidos em um projeto web completo (ex: um blog simples, um sistema de tarefas).
- **Exemplos de código (HTML, CSS, JS, PHP/CodeIgniter, SQL):** Desenvolvimento iterativo do projeto.
- **Comentários:** Orientação passo a passo para construir o projeto.

SUMÁRIO

Módulo 1: Introdução à Web e HTML

Neste primeiro módulo, vamos explorar os conceitos fundamentais que sustentam a *World Wide Web* e da linguagem de marcação HTML (*HyperText Markup Language*). Compreender como a web funciona é o primeiro passo essencial para se tornar um desenvolvedor de aplicações web.

1.1 O que é a Web?

A ***World Wide Web*** **(www)**, geralmente chamada apenas de "Web", é um sistema de documentos interconectados e outros recursos, acessíveis via Internet. Ela foi criada por Tim Berners-Lee em 1989, com o objetivo de facilitar o compartilhamento de informações entre pesquisadores.

Pense na Internet como uma vasta rede de estradas e na Web como os diversos tipos de edifícios e cidades construídos sobre essas estradas – blogs, lojas online, redes sociais, sistemas bancários, etc. A Web é o que torna a Internet útil para a maioria das pessoas, permitindo que naveguemos por sites, assistamos a vídeos, e nos comuniquemos globalmente.

- **Internet:** A infraestrutura física da rede (cabos, roteadores, servidores).
- **Web:** Um dos serviços que rodam sobre a Internet, composto por páginas e recursos interligados.

Ilustração 1.1: Diferença entre Internet e Web.

Fonte: www.dreamstime.com

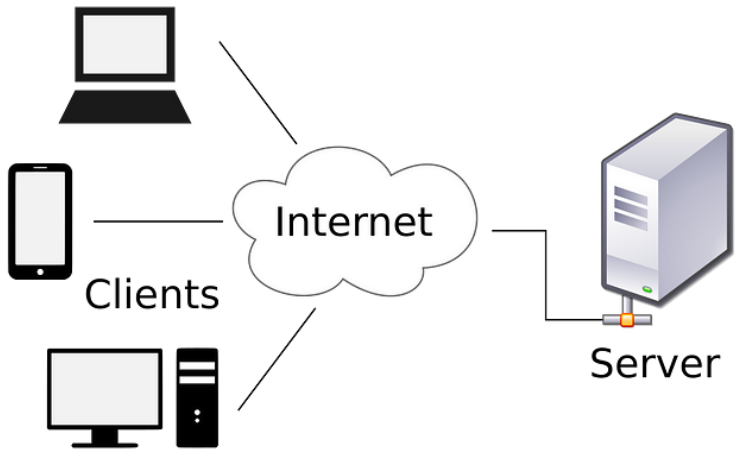
1.2 Funcionamento básico da Web (cliente e servidor)

A Web funciona principalmente com base em um modelo chamado **Cliente-Servidor**.

- **Cliente:** É o programa que você usa para acessar a Web, como o seu navegador (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari). Quando digita um endereço ou clica em um link, seu navegador atua como o cliente, fazendo uma **requisição** (pedido) por uma página ou recurso.
- **Servidor:** É um computador potente que armazena as páginas web, imagens, vídeos e outros arquivos. Quando um servidor recebe uma requisição de um cliente, ele processa essa requisição e envia uma **resposta** (o arquivo solicitado) de volta para o cliente.

Esse processo acontece em milissegundos, e é por isso que parece que as páginas aparecem instantaneamente na sua tela.

Ilustração 1.2: Modelo Cliente-Servidor.



Fonte: https://medium.com/

1.3 Protocolo HTTP e URLs

Para que cliente e servidor possam "conversar", eles precisam seguir um conjunto de regras, um **protocolo**. Na Web, o principal protocolo é o **HTTP** (*Hypertext Transfer Protocol*).

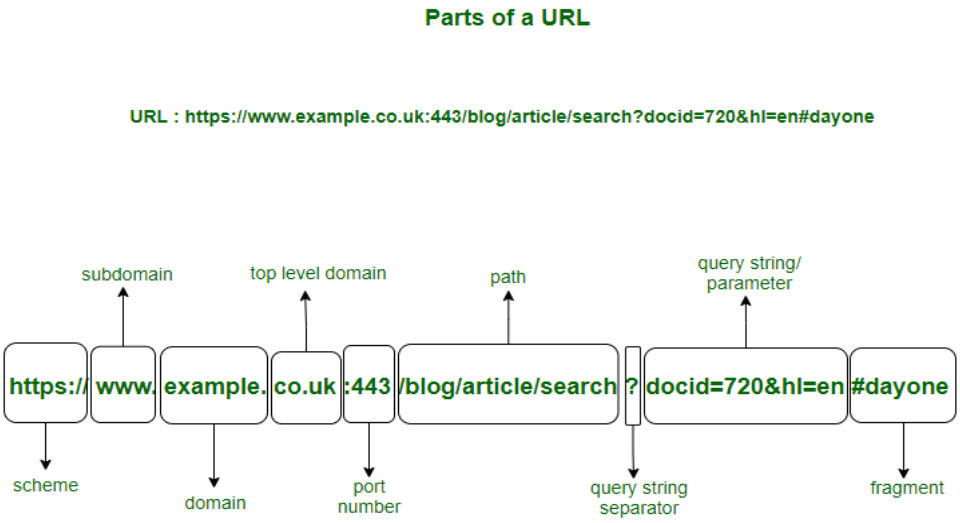
O HTTP define como as mensagens são formatadas e transmitidas, e como os servidores web e navegadores devem responder aos comandos. Quando você vê

http:// ou https:// no início de um endereço, isso indica que o HTTP (ou sua versão segura, HTTPS) está sendo usado.

Uma **URL** (*Uniform Resource Locator*) é o endereço único de cada recurso na Web. É como o "CEP" de uma página web ou imagem. Uma URL é composta por várias partes: https://www.exemplo.com.br/caminho/pagina.html?parametro=valor#secao

- https://: **protocolo** - Indica como a comunicação deve ser feita.
- www.exemplo.com.br: **domínio** - O nome do site, que é mais fácil de lembrar que o endereço IP do servidor.
- /caminho/pagina.html: **caminho** - A localização do recurso específico no servidor.
- ?parametro=valor: **parâmetros de consulta (*query string*)** - Informações extras enviadas ao servidor (opcional).
- #secao: **fragmento/âncora** - Uma parte específica dentro da página (opcional, processada pelo navegador, não enviada ao servidor).

Ilustração 1.3: Estrutura de uma URL.



Fonte: www.geeksforgeeks.org

1.4 Navegadores e Servidores

1.4.1 Navegadores (clientes web)

Os navegadores são programas essenciais que permitem aos usuários acessar e interagir com as informações na *World Wide Web (www)*. Eles são responsáveis por:

- **Fazer requisições HTTP:** Pedir páginas e recursos aos servidores.
- **Interpretar o código:** Entender HTML (estrutura), CSS (estilo) e JavaScript (interatividade) para exibir a página corretamente.
- **Renderizar a página:** Desenhar os elementos na tela, aplicando os estilos e executando os scripts.
- **Gerenciar sessão:** Guardar informações como cookies para manter você logado em sites.

Ilustração 1.4: Exemplos de Navegadores Web.

Fonte: www.alamy.com

1.4.2 Servidores Web

Servidores web são softwares (e também os computadores onde esse software roda) que "servem" conteúdo da web. Eles estão sempre conectados à internet e esperando por requisições de clientes. Alguns dos mais comuns incluem:

- **Apache HTTP Server:** Muito popular e robusto.
- **Nginx:** Conhecido por sua alta performance e escalabilidade.
- **Microsoft IIS (*Internet Information Services*):** Servidor web da Microsoft.

Quando você acessa um site, o navegador envia uma requisição para o servidor onde o site está hospedado. O servidor então localiza a página ou recurso solicitado e o envia de volta ao navegador.

Ilustração 1.5: Um Servidor Web.

Fonte: https://analysistecnologia.wordpress.com

1.5 Introdução ao HTML e Estrutura de uma página

HTML (*HyperText Markup Language*) é a linguagem padrão para criar páginas web. Ela é usada para estruturar o conteúdo de uma página, definindo seus elementos como títulos, parágrafos, imagens, links, tabelas, e muito mais. HTML não é uma linguagem de programação, mas uma **linguagem de marcação**.

Um documento HTML é composto por uma série de **elementos**, que são representados por **tags**. A maioria das tags vêm em pares: uma tag de abertura (<tag>) e uma tag de fechamento (</tag>). O conteúdo fica entre elas.

Estrutura básica de um documento HTML:

Todo documento HTML segue uma estrutura fundamental:

Exemplo de Código (HTML):

HTML

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Minha Primeira Página Web</title>
</head>
<body>
  <h1>Bem-vindo à minha página!</h1>
  <p>Este é um parágrafo de exemplo.</p>
```

```
  <a href="https://www.ueg.br/posse">Visite a UEG Posse </a>
  <img src="https://via.placeholder.com/150" alt="Placeholder de imagem">
</body>
</html>
```

1. **<!DOCTYPE html>**: Declaração que define o tipo de documento, informando ao navegador que se trata de um documento HTML5.
2. **<html lang="pt-BR">**: O elemento raiz de toda a página HTML. O atributo lang="pt-BR" indica que o idioma principal do conteúdo é Português (Brasil), o que ajuda na acessibilidade e em motores de busca.
3. **<head>**: Contém metadados sobre a página, como o título que aparece na aba do navegador, a codificação de caracteres (<meta charset="UTF-8">), e links para folhas de estilo CSS ou scripts JavaScript externos. O conteúdo dentro de <head> não é visível diretamente na página.
    - **<meta charset="UTF-8">**: Define a codificação de caracteres para que o navegador exiba corretamente caracteres especiais (como acentos e "ç").
    - **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Essencial para tornar sua página responsiva, garantindo que ela se ajuste bem a diferentes tamanhos de tela (celulares, tablets, desktops).
    - **<title>**: O texto que aparece na barra de título do navegador ou na aba da página.
4. **<body>**: Contém todo o conteúdo visível da página, como textos, imagens, vídeos, botões, formulários, etc.

Veja um exemplo prático da estrutura básica e alguns elementos comuns:

```
<!DOCTYPE html>
<!--
  Este é um exemplo básico de uma página HTML.
  Ele demonstra a estrutura fundamental e alguns elementos comuns.
-->
<html lang="pt-BR">
<head>
  <!-- Define a codificação de caracteres para suporte a acentuação e caracteres especiais -->
  <meta charset="UTF-8">
```

```
  <!-- Configura a viewport para garantir que a página seja responsiva em diferentes dispositivos -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Define o título da página, que aparece na aba do navegador -->
  <title>Minha Primeira Página Web Interativa</title>
  <!-- Inclui o Tailwind CSS para estilização rápida e responsiva -->
  <script src="https://cdn.tailwindcss.com"></script>
  <style>
    /* Define a fonte Inter como padrão para todo o corpo do documento */
    body {
      font-family: "Inter", sans-serif;
    }
  </style>
</head>
<body class="bg-gradient-to-r from-blue-500 to-purple-600 min-h-screen flex items-center justify-center p-4">
  <!-- Container principal com estilo de cartão, centralizado na tela -->
  <div class="bg-white p-8 rounded-2xl shadow-xl max-w-lg w-full text-center">
    <!-- Título principal da página -->
    <h1 class="text-4xl font-bold text-gray-800 mb-4">Bem-vindo à Web!</h1>
    <!-- Parágrafo de introdução -->
    <p class="text-gray-600 mb-6">
      Esta é a sua primeira página web! O HTML estrutura o conteúdo, e o CSS (com Tailwind) o deixa bonito.
      A interação virá com JavaScript!
    </p>

    <!-- Seção de demonstração de elementos -->
    <div class="space-y-4 mb-6">
      <!-- Exemplo de um link -->
      <a href="https://www.ueg.br/posse" target="_blank" class="inline-block bg-blue-600 hover:bg-blue-700 text-white font-semibold py-2 px-6 rounded-lg transition duration-300 ease-in-out transform hover:scale-105 shadow-md">
        Visitar a UEG Posse
      </a>

      <!-- Exemplo de uma imagem com fallback (placeholder) -->
      <img src="https://placehold.co/300x150/e0e0e0/555555?text=Sua+Imagem"
        alt="Placeholder de Imagem"
```

```
          class="mx-auto rounded-lg shadow-sm w-full max-w-[300px]">

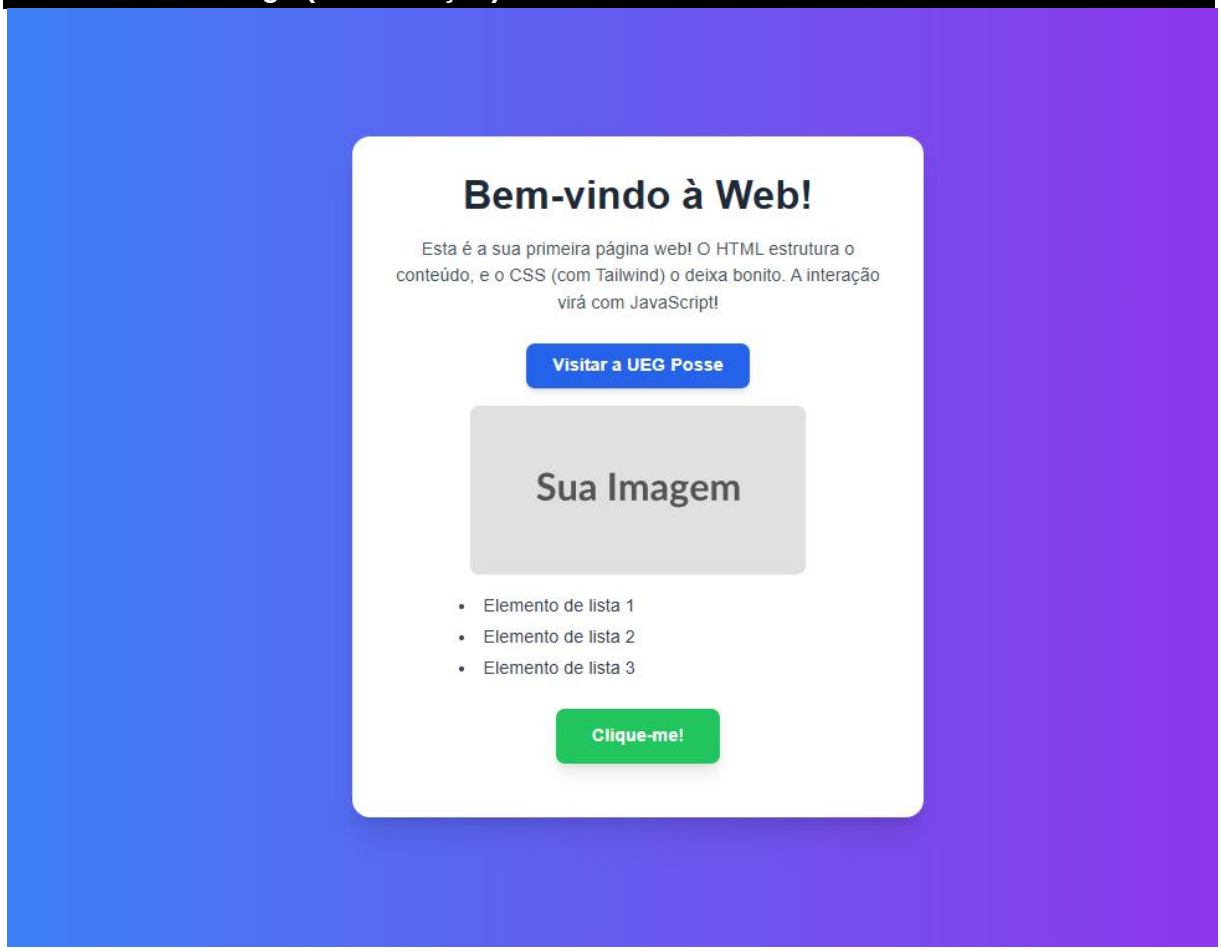
        <!-- Exemplo de uma lista não ordenada -->
        <ul class="list-disc list-inside text-left mx-auto max-w-xs text-gray-700">
          <li class="mb-1">Elemento de lista 1</li>
          <li class="mb-1">Elemento de lista 2</li>
          <li>Elemento de lista 3</li>
        </ul>
      </div>

      <!-- Botão interativo que será usado para demonstrações futuras com JavaScript -->
      <button id="meuBotao" class="bg-green-500 hover:bg-green-600 text-white font-bold py-3
px-8 rounded-lg transition duration-300 ease-in-out transform hover:scale-105 shadow-lg">
        Clique-me!
      </button>

      <!-- Parágrafo para exibir a mensagem do clique do botão -->
      <p id="mensagem" class="mt-4 text-gray-700 text-lg font-medium"></p>
    </div>

    <!-- Script JavaScript para adicionar interatividade -->
    <script>
      // Adiciona um "listener" de evento ao botão com o ID "meuBotao"
      // Quando o botão é clicado, a função dentro é executada.
      document.getElementById('meuBotao').addEventListener('click', function() {
        // Seleciona o parágrafo com o ID "mensagem"
        const mensagemParagrafo = document.getElementById('mensagem');
        // Altera o texto do parágrafo para exibir uma mensagem
        mensagemParagrafo.textContent = "Você clicou no botão! Parabéns por sua primeira
interação!";
        // Adiciona uma classe Tailwind para estilizar a mensagem
        mensagemParagrafo.classList.add('text-green-700', 'font-semibold');
      });
    </script>
</body>
</html>
```

Resultado do código (Visualização)



Módulo 2: Fundamentos da Programação

Depois de entender como a Web funciona e como estruturar e estilizar páginas, é hora de mergulhar no coração da criação de software: a **programação**. Este módulo introduzirá os conceitos fundamentais que são a base para qualquer linguagem de programação, incluindo JavaScript, PHP ou qualquer outra que você venha a aprender.

2.1 Lógica de programação: Conceitos iniciais

A **lógica de programação** é a forma como você organiza pensamentos e instruções para resolver um problema. É a base para construir qualquer programa de computador. Antes de escrevermos uma única linha de código, precisamos saber *o que* queremos que o computador faça e *como* ele deve fazer.

2.1.1 Algoritmos

Um **algoritmo** é uma sequência finita de instruções claras, bem definidas e não ambíguas, que são executadas para resolver um problema ou realizar uma tarefa. Pense numa receita de bolo: é um algoritmo para fazer um bolo!

Características de um bom algoritmo:

- **Finitude:** Deve ter um número limitado de passos.
- **Definição:** Cada passo deve ser claro e preciso.
- **Eficiência:** Deve resolver o problema de forma otimizada.
- **Generalidade:** Deve ser aplicável a diferentes situações do mesmo tipo de problema.

Exemplo de Algoritmo (Culinário):

1. Pegar ovos, farinha, açúcar, leite.
2. Misturar ovos e açúcar.
3. Adicionar farinha e leite, misturando.
4. Despejar a massa numa forma.
5. Levar ao forno por 30 minutos.
6. Retirar do forno.
7. Deixar arrefecer.
8. Servir.

Ilustração 2.1.1: Um Algoritmo para Tarefas Diárias.

Fonte: vinanhatrang.com

2.1.2 Pseudocódigo

Pseudocódigo é uma forma de descrever um algoritmo usando uma linguagem informal, que mistura termos da linguagem natural com algumas convenções de programação. Ele não é executado por um computador, mas serve para planejar o código antes de escrevê-lo numa linguagem real.

Vantagens:

- Fácil de entender (parecido com português/inglês).
- Não exige sintaxe perfeita, foca na lógica.
- Ajuda a estruturar o pensamento.

Exemplo de Pseudocódigo: Calcular Média de Notas

```
ALGORITMO CalcularMediaNotas
 VAR
  nota1, nota2, media : REAL

 INICIO
  ESCREVER "Digite a primeira nota: "
  LER nota1

  ESCREVER "Digite a segunda nota: "
  LER nota2
```

```
  media <- (nota1 + nota2) / 2

  SE media >= 6.0 ENTÃO
    ESCREVER "Aluno Aprovado! Média: " + media
  SENÃO
    ESCREVER "Aluno Reprovado. Média: " + media
  FIM_SE
FIM
```

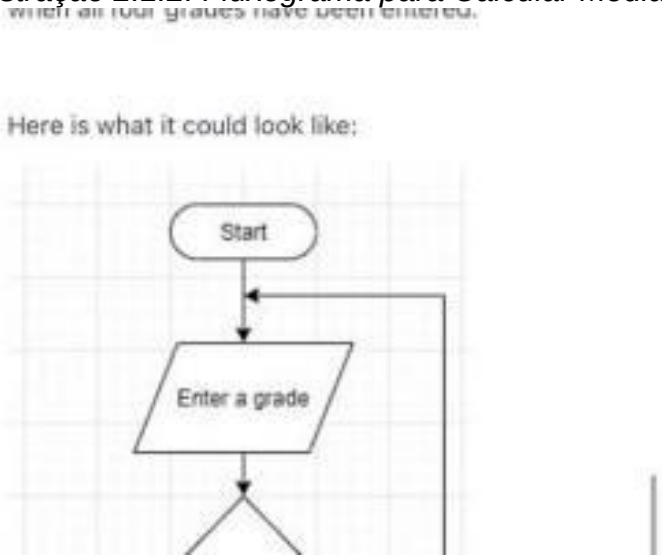
2.1.3 Fluxogramas

Fluxogramas são representações gráficas de algoritmos. Eles usam símbolos padronizados para representar as diferentes etapas e a direção do fluxo lógico.

Símbolos comuns:

- **Terminal (início/fim):** Oval.
- **Processamento:** Retângulo (operações, cálculos).
- **Entrada/saída (*input/output*):** Paralelogramo (ler dados, exibir resultados).
- **Decisão:** Losango (pontos onde o fluxo pode seguir diferentes caminhos, geralmente com "Sim/Não" ou "Verdadeiro/Falso").
- **Linhas de fluxo:** Setas que conectam os símbolos, indicando a ordem de execução.

Ilustração 2.1.2: Fluxograma para Calcular Média.



Fonte: www.chegg.com

2.2 Variáveis, Operadores e Expressões

2.2.1 Variáveis

Uma **variável** é um espaço na memória do computador usado para armazenar dados temporariamente. É como uma "caixa" rotulada onde você pode guardar diferentes tipos de informações (números, textos, etc.) e cujo conteúdo pode mudar ao longo da execução do programa.

Regras para nomes de variáveis (gerais):

- Devem começar com uma letra, $ (em PHP) ou _ (em outras linguagens).
- Podem conter letras, números e _.
- Não podem conter espaços.
- São geralmente *case-sensitive* (diferenciam maiúsculas de minúsculas).
- Devem ter nomes descritivos (ex: idadeAluno é melhor que x).

Declaração e atribuição (em JavaScript): Em JavaScript, usamos **let**, **const** ou **var** para declarar variáveis.

- **let:** Usado para variáveis que podem ter seu valor reatribuído.
- **const:** Usado para variáveis cujo valor não deve mudar após a atribuição inicial (constantes).

- **var:** Uma forma mais antiga de declarar variáveis, com um escopo menos previsível (menos recomendado em código moderno).

JavaScript

```
// Exemplo de declaração e atribuição de variáveis em JavaScript
let nome = "Alice"; // Declarando uma variável 'nome' e atribuindo o texto "Alice"
let idade = 25;    // Declarando 'idade' e atribuindo o número 25
const PI = 3.14159; // Declarando uma constante 'PI'

console.log(nome);  // Saída: Alice
console.log(idade); // Saída: 25

idade = 26; // O valor da variável 'idade' pode ser alterado
console.log(idade); // Saída: 26

// PI = 3.14; // Isso causaria um erro, pois PI é uma constante!
```

Tipos de dados

Os dados armazenados em variáveis têm diferentes **tipos**. Compreender os tipos de dados é fundamental, pois eles determinam que operações podem ser realizadas.

Números (Numbers): Representam valores numéricos, inteiros ou com casas decimais.

```
let quantidade = 10;   // Inteiro
let preco = 99.99;    // Com casas decimais (Float/Double)
```

Textos (Strings): Representam sequências de caracteres (palavras, frases). São delimitados por aspas simples ou duplas.

```
let saudacao = "Olá, mundo!";
let caractere = 'A';
```

Booleanos (Booleans): Representam valores lógicos: true (verdadeiro) ou false (falso). Usados em decisões.

```
let estaLogado = true;
let temPermissao = false;
```

Arrays: Coleções ordenadas de itens. São como listas.

```
let frutas = ["Maçã", "Banana", "Uva"];
let numeros = [1, 2, 3, 4, 5];
```

Objetos: Coleções de pares chave-valor, representando entidades mais complexas (visto mais a fundo em POO).

```
let pessoa = {
  nome: "Maria",
  idade: 30,
  cidade: "São Paulo"
};
```

Operadores

Operadores são símbolos que realizam operações em um ou mais valores (operandos).

Operadores aritméticos: Realizam operações matemáticas.

- + (Adição): 5 + 3 resulta 8
- - (Subtração): 10 - 4 resulta 6
- * (Multiplicação): 2 * 7 resulta 14
- / (Divisão): 15 / 3 resulta 5
- % (Módulo - resto da divisão): 10 % 3 resulta 1 (o resto de 10 dividido por 3)
- ** (Exponenciação): 2 ** 3 resulta 8 (2 elevado à 3ª potência)

```
let resultadoSoma = 10 + 5;    // 15
let resultadoModulo = 17 % 5;  // 2
```

Operadores de atribuição: Atribuem um valor a uma variável.

- = (Atribuição simples): x = 10
- += (Adição e atribuição): x += 5 é o mesmo que x = x + 5
- -= (Subtração e atribuição): x -= 2 é o mesmo que x = x - 2
- E assim por diante para *=, /=, %=.

```
let pontuacao = 100;
pontuacao += 20; // pontuacao agora é 120
```

Operadores de comparação: Comparam dois valores e retornam um booleano (true ou false).

- == (Igual a - compara apenas o valor): '5' == 5 resulta true

- === (Estritamente igual a - compara valor e tipo): '5' === 5 resulta false
- != (Diferente de - valor): '5' != 6 resulta true
- !== (Estritamente diferente de - valor ou tipo): '5' !== 5 resulta true
- > (Maior que): 7 > 5 resulta true
- < (Menor que): 3 < 8 resulta true
- >= (Maior ou igual a): 10 >= 10 resulta true
- <= (Menor ou igual a): 4 <= 5 resulta true

```
let idadeMinima = 18;
let sualdade = 20;
let podeEntrar = sualdade >= idadeMinima; // true
```

Operadores lógicos: Combinam ou modificam condições booleanas.

- && (AND lógico): Retorna true se *ambas* as condições forem true.
    - true && true resulta true
    - true && false resulta false
- || (OR lógico): Retorna true se *pelo menos uma* das condições for true.
    - true || false resulta true
    - false || false resulta false
- ! (NOT lógico): Inverte o valor booleano.
    - !true resulta false
    - !false resulta true

```
let temCartao = true;
let saldoSuficiente = false;
let podeComprar = temCartao && saldoSuficiente; // false
```

Expressões

Uma **expressão** é uma combinação de valores, variáveis e operadores que, quando avaliada, produz um único valor.

```
// Exemplos de expressões
let a = 10;
let b = 5;

let expressao1 = a + b * 2;          // 10 + 10 = 20
let expressao2 = (a > b) && (a % b === 0); // (true) && (true) = true
```

```
let expressao3 = "Olá " + nome;       // "Olá Alice"
```

2.3 Estruturas condicionais (*if, else, switch*)

As **estruturas condicionais** permitem que o seu programa tome decisões e execute blocos de código diferentes com base em certas condições.

2.3.1 if, else if, else

A estrutura if é a mais básica para tomar decisões.

- **if (condição):** O código dentro do bloco será executado apenas se a condição for true.
- **else if** (outraCondição): Se a primeira if for false, esta condição será verificada. Você pode ter vários else if.
- **else**: Se nenhuma das condições if ou else if for true, o código dentro do else será executado.

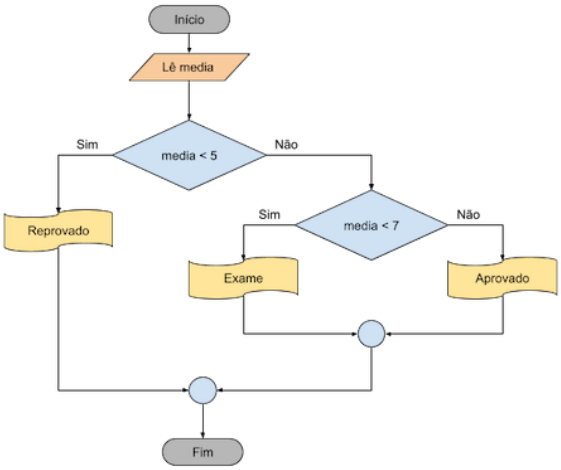
```
// Exemplo: Verificar a idade para votar
let idadeCandidato = 17;

if (idadeCandidato >= 18) {
  console.log("Você tem idade para votar.");
} else if (idadeCandidato >= 16) {
  console.log("Você pode votar como menor emancipado (em alguns países).");
} else {
  console.log("Você ainda não tem idade para votar.");
}

// Exemplo: Aninhar condições (evite aninhamento excessivo para clareza)
let temperatura = 28;
let chovendo = true;

if (temperatura > 25) {
  if (chovendo) {
    console.log("Dia quente e chuvoso. Fique em casa!");
  } else {
    console.log("Dia quente e ensolarado. Ótimo para um passeio!");
  }
} else {
  console.log("Temperatura amena.");
}
```

Ilustração 2.3.1: Fluxo de uma Estrutura If-Else.



Fonte: http://userdir.luzerna.ifc.edu.br

2.3.2 switch

A estrutura *switch* é útil quando você tem uma variável que pode ter múltiplos valores e deseja executar um código diferente para cada um desses valores.

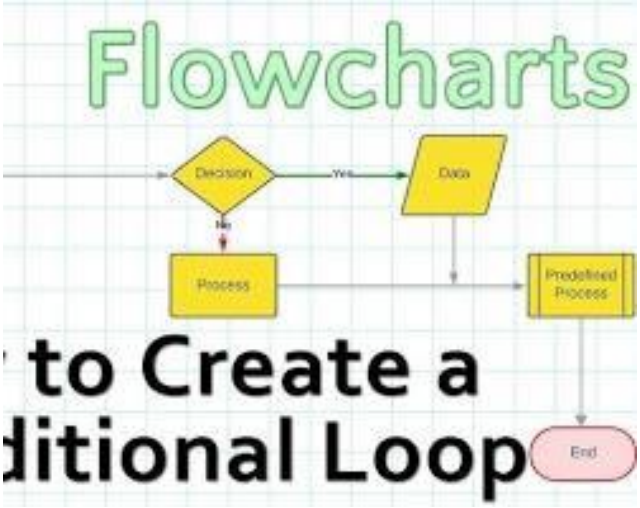
- **switch** (expressao): A expressao é avaliada.
- **case** valor: Se o valor da expressao corresponder a este valor, o código sob este case é executado.
- **break**: Essencial! Ele faz com que o programa "saia" do switch após encontrar um case correspondente. Sem ele, o código continuaria a executar os próximos case (comportamento chamado "fall-through").
- **default**: (Opcional) O código sob default é executado se a expressao não corresponder a nenhum case.

```
// Exemplo: Mensagem baseada no dia da semana
let diaDaSemana = 4; // 1 = Domingo, 2 = Segunda, ..., 7 = Sábado

switch (diaDaSemana) {
  case 1:
    console.log("Domingo: Dia de descanso!");
    break;
  case 2:
    console.log("Segunda-feira: Início da semana de trabalho/estudos.");
    break;
  case 3:
  case 4: // Múltiplos cases para o mesmo bloco de código
```

```
    console.log("Terça/Quarta-feira: Meio da semana.");
    break;
  case 5:
    console.log("Quinta-feira: Quase lá!");
    break;
  case 6:
    console.log("Sexta-feira: Fim de semana à vista!");
    break;
  case 7:
    console.log("Sábado: Dia de lazer!");
    break;
  default:
    console.log("Número de dia inválido.");
}
```

Ilustração 2.3.2: Fluxo de uma Estrutura Switch.



Fonte: www.youtube.com

2.4 Laços de repetição (for, while)

Laços de repetição (ou *loops*) são estruturas que permitem executar um bloco de código repetidamente, enquanto uma condição for verdadeira ou por um número específico de vezes.

2.4.1 for

O laço for é ideal quando você sabe (ou pode determinar) quantas vezes quer que o código se repita.

Sintaxe: for (inicialização; condição; incremento/decremento)

- Inicialização: Executada uma única vez no início do laço (ex: let i = 0).

- Condição: Verificada antes de cada iteração. Se true, o laço continua; se false, o laço termina.
- Incremento/decremento: Executado no final de cada iteração (ex: i++).

```
// Exemplo: Contar de 1 a 5
console.log("Contagem com for:");
for (let i = 1; i <= 5; i++) {
  console.log("Número: " + i);
}
// Saída:
// Número: 1
// Número: 2
// Número: 3
// Número: 4
// Número: 5

// Exemplo: Iterar sobre um array
let frutas = ["Maçã", "Banana", "Laranja"];
console.log("\nMinhas frutas (com for):");
for (let i = 0; i < frutas.length; i++) {
  console.log(`- ${frutas[i]}`);
}
// Saída:
// - Maçã
// - Banana
// - Laranja
```

2.4.2 while

O laço while é usado quando você quer que o código se repita *enquanto uma condição for verdadeira*, e o número de repetições não é conhecido de antemão.
Sintaxe: while (condição)

```
// Exemplo: Repetir enquanto um contador for menor que 3
console.log("\nContagem com while:");
let contador = 0;
while (contador < 3) {
  console.log("Passo: " + contador);
  contador++; // Lembre-se de mudar a condição para evitar um loop infinito!
}
// Saída:
// Passo: 0
// Passo: 1
// Passo: 2
```

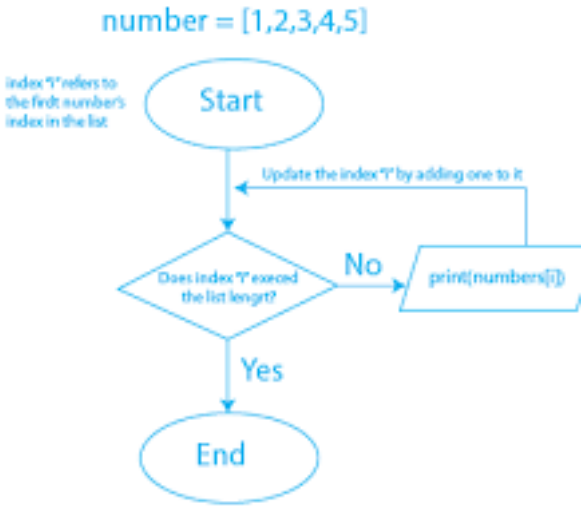
Cuidado: Se a condição do while nunca se tornar false, você terá um **loop infinito**, que pode travar o seu programa ou navegador!

2.4.3 do-while (Variação do While)

O laço do-while é semelhante ao while, mas a sua condição é verificada *após* a execução do bloco de código. Isso significa que o bloco de código será executado pelo menos uma vez, mesmo que a condição seja inicialmente falsa.

```
// Exemplo: do-while
let numero = 0;
do {
  console.log("O número é: " + numero);
  numero++;
} while (numero < -5); // Condição é falsa, mas o bloco executa uma vez.
// Saída:
// O número é: 0
```

Ilustração 2.4.1: Fluxo de um Laço de Repetição (While).



Fonte: www.prepbytes.com

2.5 Funções e escopo de variáveis

2.5.1 Funções

Uma **função** é um bloco de código organizado e reutilizável que executa uma tarefa específica. Elas são essenciais para modularizar o seu código, tornando-o mais legível, fácil de manter e de depurar.

Benefícios do uso de funções:

- **Reusabilidade:** Escreva o código uma vez, use-o em vários lugares.
- **Organização:** Divide um problema grande em partes menores e gerenciáveis.

- **Legibilidade:** O código fica mais fácil de entender.
-

Sintaxe de Função (em JavaScript):

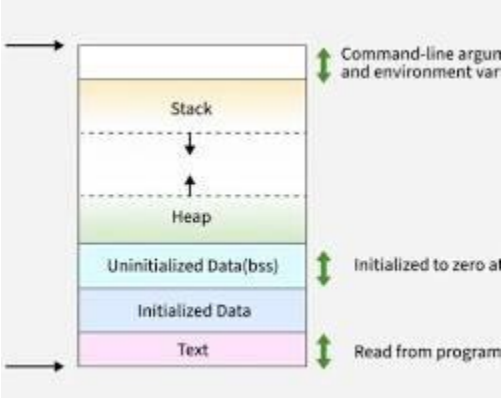
```
// Função sem parâmetros e sem retorno
function exibirMensagemBoasVindas(){
  console.log("Bem-vindo ao mundo da programação!");
}
exibirMensagemBoasVindas(); // Chamando a função

// Função com parâmetros
function saudacao(nome){
  console.log(`Olá, ${nome}!`);
}
saudacao("Carolina"); // Saída: Olá, Carolina!
saudacao("Fernando"); // Saída: Olá, Fernando!

// Função com parâmetros e retorno de valor
function somarNumeros(num1, num2){
  let resultado = num1 + num2;
  return resultado; // A função "devolve" um valor
}
let total = somarNumeros(10, 7);
console.log("A soma é: " + total); // Saída: A soma é: 17

// Função com parâmetro opcional e valor padrão
function cumprimentar(nome = "Usuário"){ // Se 'nome' não for fornecido, usa "Usuário"
  console.log(`Olá, ${nome}! Tenha um bom dia.`);
}
cumprimentar("Lúcia"); // Saída: Olá, Lúcia! Tenha um bom dia.
cumprimentar();      // Saída: Olá, Usuário! Tenha um bom dia.
```

Ilustração 2.5.1: Funções como Blocos Reutilizáveis de Código.



Fonte: www.geeksforgeeks.org

2.5.2 Escopo de variáveis

O **escopo** de uma variável define a parte do código onde essa variável pode ser acedida. Entender o escopo é crucial para evitar erros e garantir que suas variáveis se comportem como o esperado.

- **Escopo global:** Variáveis declaradas fora de qualquer função ou bloco. Podem ser acedidas de qualquer lugar no programa.
- **Escopo local (de Função):** Variáveis declaradas dentro de uma função. Só podem ser acedidas de dentro dessa função. Quando a função termina, as variáveis locais são destruídas.
- **Escopo de bloco (let e const):** Em JavaScript moderno, let e const têm Escopo de bloco, o que significa que são acessíveis apenas dentro do bloco ({}) onde foram declaradas (ex: dentro de um if, for, while).

JavaScript

```
// Exemplo de Escopo de Variáveis em JavaScript

let variavelGlobal = "Sou uma variável global"; // Escopo global

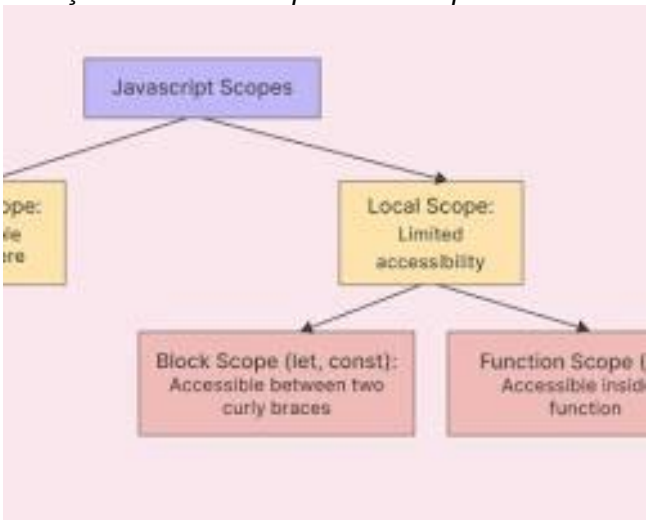
function exemploEscopo() {
  let variavelLocal = "Sou uma variável local"; // Escopo local (da função)
  console.log(variavelGlobal); // Posso aceder à global
  console.log(variavelLocal);  // Posso aceder à local
}

exemploEscopo(); // Chama a função

console.log(variavelGlobal); // Posso aceder à global fora da função
// console.log(variavelLocal); // ERRO! variavelLocal não está definida aqui, pois está fora do
seu escopo.

// Exemplo de Escopo de bloco com let e const
if (true) {
  let mensagem = "Variável de Escopo de bloco";
  const ID = 123;
  console.log(mensagem); // Acessível aqui
  console.log(ID);      // Acessível aqui
}
// console.log(mensagem); // ERRO! 'mensagem' não está definida fora do bloco if
// console.log(ID);      // ERRO! 'ID' não está definida fora do bloco if
```

Ilustração 2.5.2: Hierarquia de Escopo de Variáveis.



Fonte: unstop.com

Exercícios:

1. **Algoritmo e pseudocódigo:** Escreva um algoritmo e, em seguida, um pseudocódigo para determinar se um número digitado pelo usuário é par ou ímpar.
2. **Variáveis e operadores:**
    - Declare duas variáveis, base e altura, e atribua a elas valores numéricos.
    - Calcule a área de um triângulo usando a fórmula (base * altura) / 2.
    - Exiba o resultado no console.
    - Verifique se a área calculada é maior que 10 e exiba true ou false no console.
3. **Estruturas condicionais:**
    - Crie uma variável pontuacao com um valor entre 0 e 100.
    - Use if, else if, else para exibir uma mensagem:
        - "Excelente!" se a pontuação for >= 90.
        - "Muito Bom!" se a pontuação for entre 70 e 89.
        - "Satisfatório." se a pontuação for entre 50 e 69.
        - "Precisa melhorar." se a pontuação for < 50.
    - Use um switch para simular uma escolha de menu (1, 2, 3, 4), onde cada número corresponde a uma opção diferente (ex: "Ver Perfil", "Editar Dados", "Sair"). Exiba a opção escolhida.

4. **Laços de repetição:**
    - Use um laço for para exibir todos os números pares de 2 a 10.
    - Use um laço while para pedir ao usuário (simulado com prompt() ou uma variável que você muda manualmente para testar) que digite um número até que ele digite '0'. Conte quantos números foram digitados antes de '0'.
5. **Funções e escopo:**
    - Crie uma função chamada calcularIMC que aceite peso (kg) e altura (metros) como parâmetros. A função deve calcular o IMC (peso / (altura * altura)) e retornar o valor.
    - Chame a função calcularIMC com seus próprios dados e exiba o resultado.
    - Crie uma função mostrarEscopo que declare uma variável mensagemLocal dentro dela. Tente aceder a mensagemLocal de fora da função e observe o erro. Crie uma variavelGlobal e acesse-a dentro e fora da função para demonstrar o escopo.

Módulo 3: Introdução ao CSS (*Cascading Style Sheets* - Folhas de Estilo em Cascata)

Depois de aprender a estruturar minimamente o conteúdo das suas páginas com HTML e conceitos básicos de lógica de programação, o próximo passo é torná-las visualmente atraentes. É aqui que o **CSS** entra em ação! O CSS é como o "estilista" da sua página web, responsável por cores, fontes, layouts e muito mais.

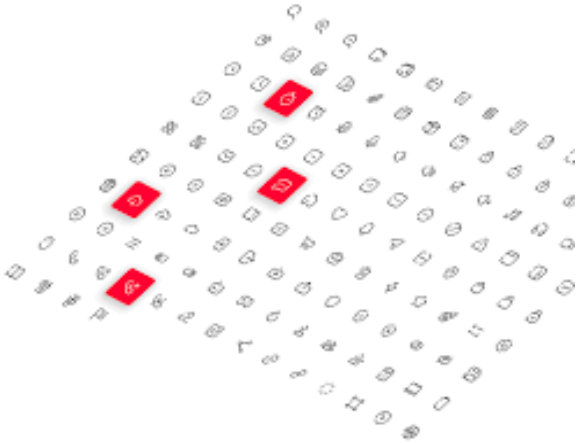
3.1 O que é CSS e por que usá-lo?

CSS significa ***Cascading Style Sheets*** (Folhas de Estilo em Cascata). É uma linguagem de folhas de estilo usada para descrever a apresentação de um documento escrito em HTML. Ou seja, ele define como os elementos HTML devem ser exibidos no ecrã, em papel ou noutras mídias.

Por que usar CSS?

- **Separação de "preocupações":** O CSS permite que você separe a **estrutura** (HTML) do **estilo** (CSS) da sua página. Isso torna o código mais organizado, fácil de ler e de manter. Imagina se cada parágrafo tivesse de ter o seu estilo diretamente no HTML? Seria um caos!
- **Reutilização:** Você pode definir estilos uma vez e aplicá-los a várias páginas HTML, garantindo consistência visual em todo o seu website.
- **Controle de layout:** O CSS oferece ferramentas poderosas para posicionar elementos, criar layouts responsivos (que se adaptam a diferentes tamanhos de ecrã) e muito mais.
- **Melhoria da experiência do usuário:** Uma página bem estilizada é mais agradável e fácil de usar.

Ilustração 3.1: HTML para estrutura, CSS para estilo.



Fonte: www.hi-interactive.com

3.2 Seletores (Tag, ID, Class)

Para aplicar estilos a elementos HTML, você precisa "selecioná-los". O CSS oferece diferentes tipos de seletores:

1. Seletor de Tag (Tipo): Aplica estilo a todas as instâncias de uma tag HTML específica.

```
p { /* Seleciona todos os parágrafos */
  color: blue;
}
```

2. Seletor de ID: Aplica estilo a um único elemento com um id específico. Um id deve ser único por página.

```
<h1 id="titulo-principal">Meu Título</h1>
```

```
```css
#titulo-principal { /* Seleciona o elemento com id="titulo-principal" */
  font-size: 36px;
}
```

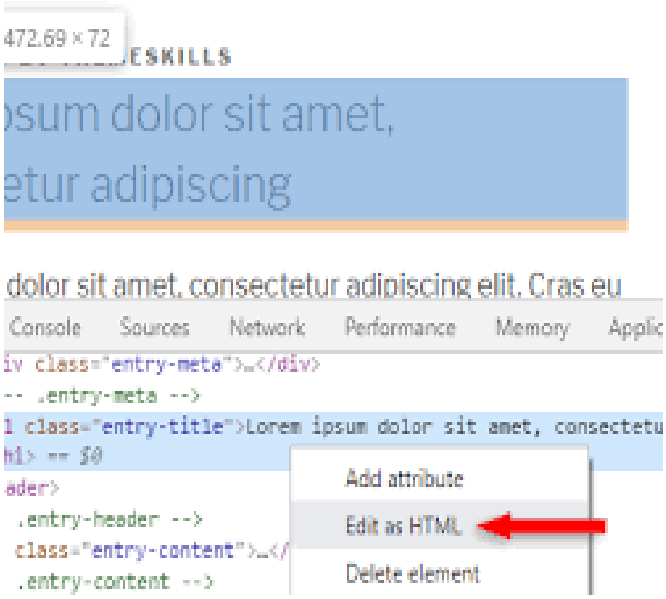
3. Seletor de Classe: Aplica estilo a todos os elementos que possuem uma class específica. Você pode aplicar a mesma classe a vários elementos e um elemento pode ter várias classes.

```
<p class="destaque">Este texto é importante.</p>
<span class="destaque">Este também!</span>
```

```
```css
.destaque { /* Seleciona todos os elementos com class="destaque" */
  font-weight: bold;
  color: red;
```

```
}
```

Ilustração 3.2: Tipos de Seletores CSS.



Fonte: themeskills.com

3.3 Propriedades de estilo

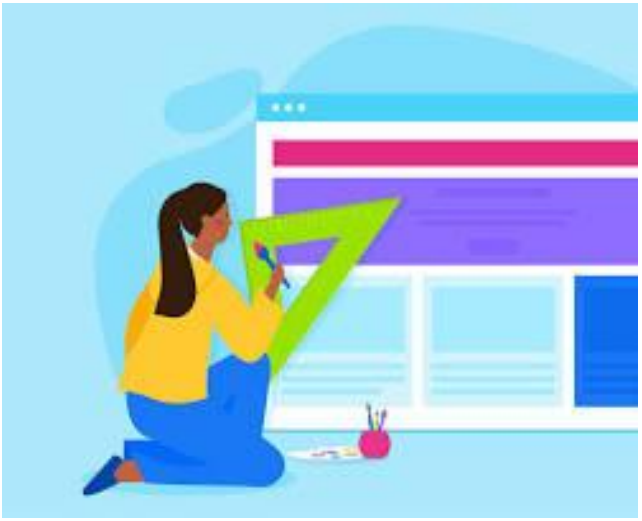
Depois de selecionar um elemento, você usa **propriedades** CSS para definir o seu estilo. Cada propriedade tem um nome (ex: color) e um valor (ex: blue). Algumas propriedades comuns incluem:

- color: Define a cor do texto.
- font-family: Define a fonte do texto (ex: Arial, Verdana).
- font-size: Define o tamanho do texto (ex: 16px, 1.2em).
- background-color: Define a cor de fundo de um elemento.
- width, height: Define a largura e a altura de um elemento.
- margin, padding: Define o espaçamento externo e interno de um elemento (ver Box Model abaixo).
- border: Define a borda de um elemento.
- text-align: Alinhamento do texto (ex: left, center, right).

```
/* Exemplo de declaração de propriedades */
body {
  font-family: Arial, sans-serif;
  background-color: #f0f0f0; /* Cor de fundo cinzenta clara */
}
```

```
h2 {
  color: #333; /* Cor do texto cinzento escuro */
  text-align: center;
  margin-top: 20px; /* Margem superior */
}
```

Ilustração 3.3: Exemplos de Propriedades CSS.



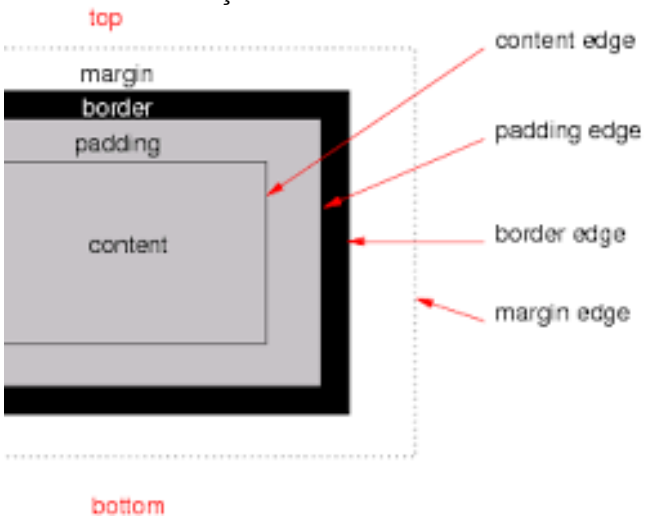
Fonte: www.jotform.com

3.4 Box Model (Modelo de Caixa)

Todo elemento HTML no CSS é considerado uma caixa retangular. O **Box Model** descreve como estas caixas são estruturadas e como o espaço é ocupado numa página. Ele é composto por quatro partes:

1. **Conteúdo (content):** Onde o conteúdo real (texto, imagem) do elemento é exibido.
2. **Padding (preenchimento):** Espaço entre o conteúdo e a borda. É um espaçamento **interno**.
3. **Border (borda):** Uma linha que envolve o padding e o conteúdo.
4. **Margin (margem):** Espaço fora da borda, usado para criar espaço entre elementos. É um espaçamento **externo**.

Ilustração 3.4: O Box Model.



Fonte: www.w3.org

Entender o Box Model é crucial para controlar o layout e o espaçamento dos seus elementos.

3.5 Inclusão de CSS (Inline, Interno, Externo)

Existem três formas principais de incluir estilos CSS numa página HTML:

1. **Estilo inline:** Aplicado diretamente na tag HTML, usando o atributo style. É útil para estilos rápidos e específicos, mas não é recomendado para grandes quantidades de código.

```
<p style="color: purple; font-size: 18px;">Este parágrafo tem um estilo inline.</p>
```

2. **Estilo interno (Embedded):** Definido dentro da tag <style> no <head> do documento HTML. É bom para estilos específicos de uma única página.

```
<head>
  <style>
    h1{
      text-decoration: underline;
    }
  </style>
</head>
```

3. **Estilo externo:** A forma mais recomendada e profissional. O CSS é escrito num arquivo separado (.css) e ligado ao documento HTML usando a tag <link> no <head>.

arquivo style.css:

```
body{
```

```
  background-color: lightblue;
}
```

arquivo index.html:

```
<head>
  <link rel="stylesheet" href="style.css">
</head>
```

4. Esta abordagem mantém o HTML limpo e permite que o mesmo arquivo CSS seja usado por múltiplas páginas.

Ilustração 3.5: Métodos de Inclusão de CSS.

Fonte: https://www.bitdegree.org/

Exemplo de página com estilos CSS

```
<!DOCTYPE html>
<!--
  Este arquivo HTML demonstra a aplicação de estilos CSS.
  Utiliza uma combinação de estilo interno (para regras base) e Tailwind CSS
  para uma estilização eficiente e responsiva.
-->
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Minha Página com Estilos CSS</title>
  <!-- Inclui o Tailwind CSS via CDN para acesso a classes utilitárias -->
  <script src="https://cdn.tailwindcss.com"></script>
  <style>
    /* Define a fonte "Inter" como padrão para todo o documento, para melhor legibilidade */
    body {
      font-family: "Inter", sans-serif;
      /* Um pequeno ajuste para garantir que a cor de fundo padrão não seja completamente substituída
         pelo Tailwind, se houver conflitos, e para adicionar um fallback. */
      background-color: #f8f8f8; /* Cor de fundo muito clara para o corpo */
    }

    /* Estilo interno: Seletor de Tag para o h1 */
    h1 {
      /* Text-shadow para um efeito subtil de profundidade */
      text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.2);
    }

    /* Estilo interno: Seletor de Classe personalizado */
    .card-personalizado {
      border: 2px solid #a78bfa; /* Borda roxa */
      box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0, 0, 0.05); /* Sombra mais pronunciada */
    }

    /* Estilo interno: Demonstração do Box Model em um elemento específico */
    .box-model-demo {
      background-color: #d1fae5; /* Verde claro para o conteúdo */
      padding: 20px; /* Preenchimento interno */
      border: 5px solid #10b981; /* Borda sólida verde */
      margin: 30px auto; /* Margem externa (auto para centrar horizontalmente) */
      max-width: 400px; /* Largura máxima para a demonstração */
      box-sizing: border-box; /* Garante que padding e border sejam incluídos na largura/altura */
      text-align: center;
      border-radius: 12px; /* Cantos arredondados */
    }
```

```
  </style>
</head>
<body class="bg-gray-100 min-h-screen flex flex-col items-center py-8">
  <!-- Contenedor principal para centralizar o conteúdo e aplicar um estilo geral -->
  <div class="max-w-4xl w-full mx-auto p-6 bg-white rounded-xl shadow-lg">

    <!-- Título principal da página com estilo de tag e classes Tailwind -->
    <h1 class="text-5xl font-extrabold text-center text-purple-700 mb-8 mt-4">
      A Magia do CSS!
    </h1>

    <!-- Seção sobre Estilos Inline -->
    <section class="mb-10 p-6 bg-blue-50 rounded-lg card-personalizado">
      <h2 class="text-3xl font-semibold text-blue-800 mb-4">3.5.1 Estilo Inline</h2>
      <p class="text-gray-700 leading-relaxed mb-4">
        Este parágrafo possui um <strong style="color: darkred; font-weight: bold;">estilo inline</strong>,
        aplicado diretamente na tag HTML. É útil para ajustes rápidos, mas não para grandes quantidades de estilo.
      </p>
      <p style="background-color: #fffacd; padding: 10px; border-radius: 8px; border: 1px dashed #daa520; color: #8b4513;">
        <span class="font-medium">Exemplo:</span> A cor de fundo e a borda desta caixa foram definidas com estilo inline.
      </p>
    </section>

    <!-- Seção sobre Seletores de ID e Classe e Estilo Interno -->
    <section class="mb-10 p-6 bg-green-50 rounded-lg card-personalizado">
      <h2 class="text-3xl font-semibold text-green-800 mb-4">3.5.2 Estilo Interno e Seletores</h2>
      <p class="text-gray-700 leading-relaxed mb-4">
        O estilo interno é definido na secção <code class="bg-gray-200 px-1 rounded">&lt;style&gt;</code> do <code class="bg-gray-200 px-1 rounded">&lt;head&gt;</code>.
        Aqui usamos seletores para aplicar estilos a elementos específicos ou grupos.
      </p>

      <h3 id="subtitulo-destaque" class="text-2xl font-bold text-indigo-700 mb-3">
```

```
        Subtítulo com ID (único na página)
      </h3>
      <p class="paragrafo-com-classe text-gray-700 mb-3">
        Este é um parágrafo que usa uma <span class="font-bold text-red-600">classe CSS</span> para estilização.
      </p>
      <p class="paragrafo-com-classe text-gray-700">
        Podemos ter múltiplos elementos com a <span class="font-bold text-red-600">mesma classe</span>.
      </p>
    </section>

    <!-- Seção sobre Box Model -->
    <section class="mb-10 p-6 bg-yellow-50 rounded-lg card-personalizado">
      <h2 class="text-3xl font-semibold text-yellow-800 mb-4">3.4 O Box Model (Modelo de Caixa)</h2>
      <p class="text-gray-700 leading-relaxed mb-4">
        Todo elemento HTML é uma caixa. O Box Model define as suas partes: Conteúdo, Padding, Borda e Margem.
      </p>
      <div class="box-model-demo">
        <p class="text-gray-800 font-semibold">Conteúdo da Caixa</p>
        <p class="text-sm text-gray-600 mt-2">
          (Este é o seu conteúdo. O verde claro é o `background-color` do conteúdo,
          os 20px são o `padding`, a linha grossa é a `border`, e o espaço à volta é a `margin`).
        </p>
      </div>
      <p class="text-gray-700 text-center mt-4">
        Observe como o padding e a borda afetam o tamanho total da caixa.
      </p>
    </section>

    <!-- Seção de Exercícios (apenas como um lembrete visual) -->
    <section class="p-6 bg-purple-50 rounded-lg card-personalizado">
      <h2 class="text-3xl font-semibold text-purple-800 mb-4">Próximo Passo: Exercícios!</h2>
      <p class="text-gray-700 leading-relaxed">
        Agora é a sua vez de praticar! Consulte o material da apostila para os exercícios propostos e comece a estilizar as suas próprias páginas.
```

```
      </p>
    </section>

  </div>
</body>
</html>
```

Resultado do código (visualização)

Exercícios:

1. Crie uma nova página HTML (exercicio_css.html).
2. Adicione um título (<h1>), um parágrafo (<p>) e uma lista não ordenada (<ul> com <li>).
3. Utilize CSS Externo (crie um arquivo estilos.css):
    - Defina a cor de fundo do body para um azul claro.
    - Alinhe o título <h1> ao centro e defina a sua cor para um cinzento escuro.
    - Defina a cor do texto do parágrafo para um verde e o tamanho da fonte para 18px.
    - Dê às li da lista uma cor de texto diferente (ex: laranja).
4. Experimente adicionar um border, padding e margin a um dos elementos para observar o Box Model em ação.

Módulo 4: Introdução ao Bootstrap (Framework CSS)

Nos módulos anteriores, exploramos o HTML para estruturar o conteúdo e o CSS para estilizá-lo. Embora o CSS puro nos dê controle total, pode ser demorado criar todos os estilos do zero, especialmente para interfaces responsivas. É aqui que entram os Frameworks CSS, e o Bootstrap é um dos mais utilizados.

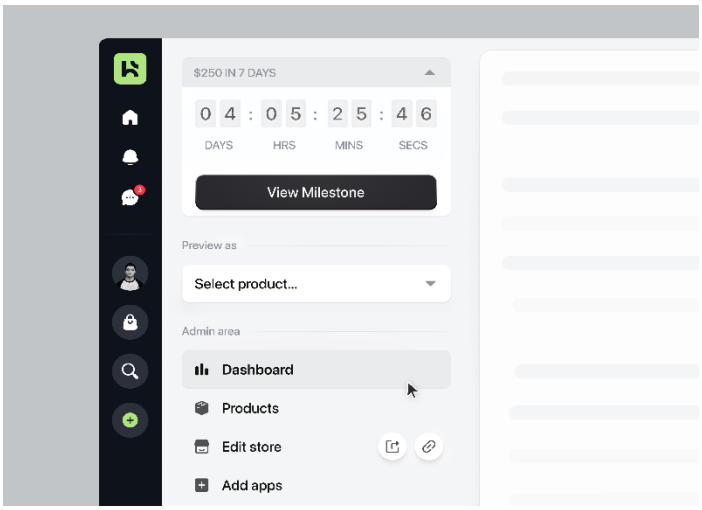
4.1 O que é Bootstrap e por que usá-lo?

Bootstrap é um framework de código aberto para desenvolvimento front-end que facilita a criação de websites responsivos e com design moderno. Ele inclui modelos de design baseados em HTML e CSS para tipografia, formulários, botões, navegação e outros componentes de interface, bem como extensões opcionais de JavaScript.

Por que usar Bootstrap?

- **Design responsivo por padrão**: Construído com uma abordagem "mobile-first", o Bootstrap garante que o seu website se adapte automaticamente a diferentes tamanhos de ecrã (desktops, tablets, telemóveis) com pouca ou nenhuma configuração adicional.
- **Componentes prontos**: Oferece uma vasta biblioteca de componentes reutilizáveis e pré-estilizados (botões, navbars, cards, modais, carrosséis, etc.), acelerando significativamente o desenvolvimento.
- **Consistência visual**: Ajuda a manter um design consistente em todo o website, pois todos os componentes seguem as diretrizes visuais do framework.
- **Fácil de aprender e usar**: Baseia-se em classes CSS que podem ser adicionadas diretamente ao seu HTML, tornando-o acessível mesmo para iniciantes.
- **Vasta comunidade e documentação**: Uma das maiores comunidades de usuários e uma documentação completa e fácil de consultar.

Ilustração 4.1: Componentes e responsividade do Bootstrap.



Fonte: https://layers-uploads-prod.s3.eu-west-2.amazonaws.com

4.2 Inclusão do Bootstrap

A forma mais comum e rápida de começar a usar o Bootstrap é através de uma CDN (*Content Delivery Network*). Isso significa que os arquivos CSS e JavaScript do Bootstrap são carregados de um servidor externo, sem a necessidade de os descarregar e hospedar no seu próprio projeto.

Para incluir o Bootstrap 5 (a versão mais recente e recomendada) no seu arquivo HTML, adicione as seguintes linhas dentro da sua tag <head>:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Minha Página com Bootstrap</title>

  <!-- Link para o CSS do Bootstrap -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
</head>
<body>
  <!-- O seu conteúdo HTML aqui -->

  <!-- Scripts do Bootstrap (colocados no final do <body> para melhor performance) -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
```

```
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdsIK1eN7N6jIeHz"
crossorigin="anonymous"></script>
</body>
</html>
```

- O link para o arquivo .css deve estar no <head>.
- O script .js deve ser colocado antes do fechamento da tag </body> para garantir que o HTML já foi carregado antes da execução do JavaScript do Bootstrap.

O bootstrap.bundle.min.js inclui o Popper.js, necessário para alguns componentes.

Ilustração 4.2: Inclusão do Bootstrap via CDN.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.7/dist/css/bootstrap.min.css" rel='
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.7/dist/js/bootstrap.bundle.min.js'
  </body>
</html>
```

Fonte: https://getbootstrap.com/

4.3 Sistema de *grid* responsivo

Um dos recursos mais poderosos do Bootstrap é o seu sistema de *grid* responsivo. Ele permite criar layouts complexos e alinhados que se ajustam automaticamente a diferentes tamanhos de ecrã.

O sistema de grid é baseado em 12 colunas e utiliza uma série de classes para organizar o conteúdo:

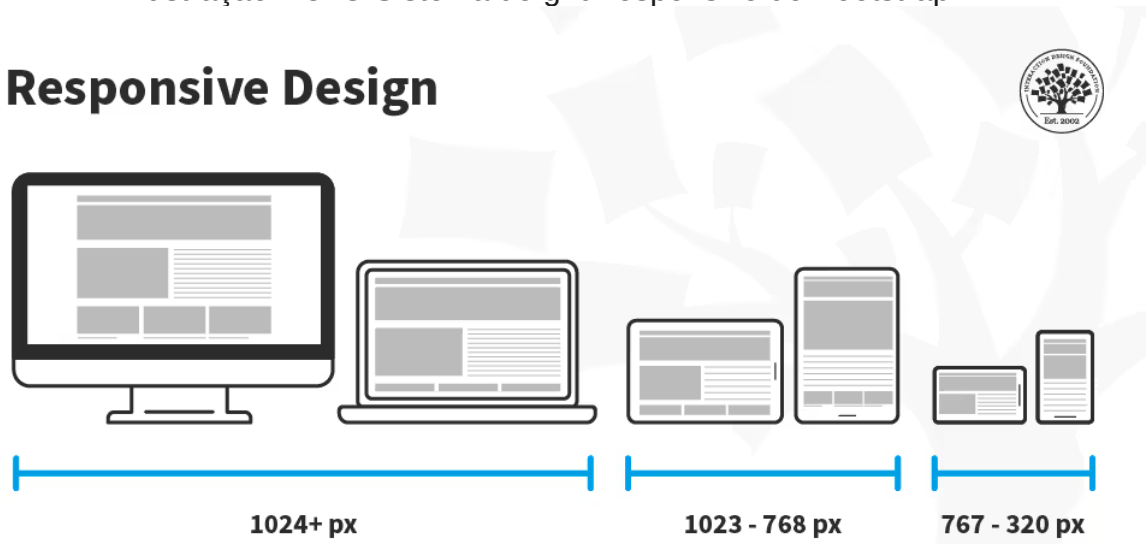
- .container ou .container-fluid: Define a largura máxima do seu conteúdo .container tem uma largura fixa em diferentes breakpoints, enquanto container-fluid ocupa 100% da largura do pai.
- .row: Define uma linha dentro do container. As colunas devem ser colocadas dentro de uma row.
- .col (e variações como .col-sm, .col-md, .col-lg, .col-xl): Definem as colunas.

- col: Faz com que as colunas tenham larguras iguais.
- col-sm-X, col-md-X, etc.: Definem a largura da coluna em diferentes *breakpoints* (tamanhos de ecrã):
- sm: small (pequeno, para telemóveis)
- md: medium (médio, para tablets)
- lg: large (grande, para desktops)
- xl: extra large (extra grande)
- xxl: extra extra large

Exemplo de grid:

```
<div class="container">
  <div class="row">
    <!-- Esta coluna ocupará 6 das 12 colunas em ecrãs médios ou maiores,
       e 12 colunas (largura total) em ecrãs pequenos. -->
    <div class="col-12 col-md-6 bg-light border p-3">
      Conteúdo da Coluna 1
    </div>
    <!-- Esta coluna também ocupará 6 das 12 colunas em ecrãs médios ou maiores. -->
    <div class="col-12 col-md-6 bg-light border p-3">
      Conteúdo da Coluna 2
    </div>
  </div>
  <div class="row mt-3">
    <!-- Colunas com larguras diferentes em ecrãs grandes -->
    <div class="col-lg-4 bg-light border p-3">
      Col 1 (4/12)
    </div>
    <div class="col-lg-8 bg-light border p-3">
      Col 2 (8/12)
    </div>
  </div>
</div>
```

Ilustração 4.3: O Sistema de grid responsivo do Bootstrap.



Fonte: https://public-images.interaction-design.org

4.4 Componentes comuns do Bootstrap

O Bootstrap oferece uma vasta gama de componentes pré-estilizados que podem ser facilmente integrados nas suas páginas. Aqui estão alguns exemplos:

4.4.1 Botões

Fácil de criar botões com diferentes estilos e tamanhos.

```
<button type="button" class="btn btn-primary">Botão Primário</button>
<button type="button" class="btn btn-success">Botão Sucesso</button>
<button type="button" class="btn btn-danger btn-lg">Botão Grande Perigo</button>
```

4.4.2 Formulários

Estilize os seus formulários para uma aparência moderna e consistente.

```
<form>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">Endereço de Email</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">
    <div id="emailHelp" class="form-text">Nunca partilharemos o seu email com ninguém.</div>
  </div>
  <div class="mb-3">
    <label for="exampleInputPassword1" class="form-label">Senha</label>
    <input type="password" class="form-control" id="exampleInputPassword1">
  </div>
```

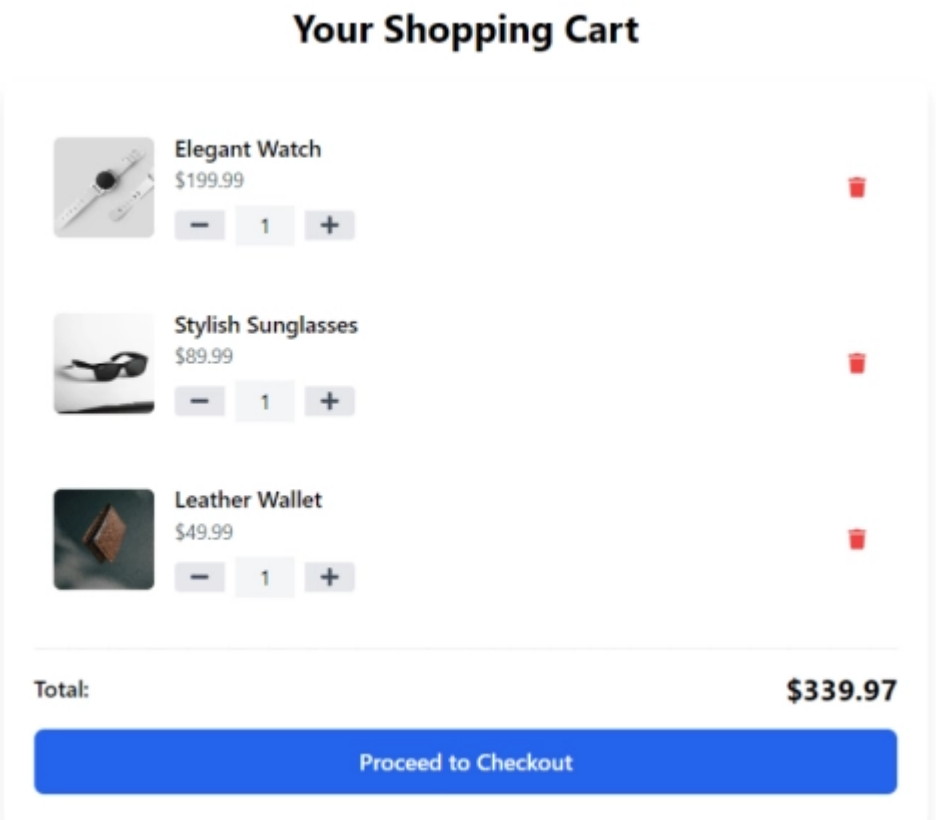
```
  <button type="submit" class="btn btn-primary">Submeter</button>
</form>
```

4.4.3 Barra de Navegação (Navbar)

Crie navbars responsivas e profissionais com facilidade.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Minha Marca</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
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">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Funcionalidades</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Preços</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

Ilustração 4.4: Exemplos de Componentes Bootstrap.



Fonte: https://purecode.ai

4.5 Integração do Bootstrap com o Projeto de Ocorrências

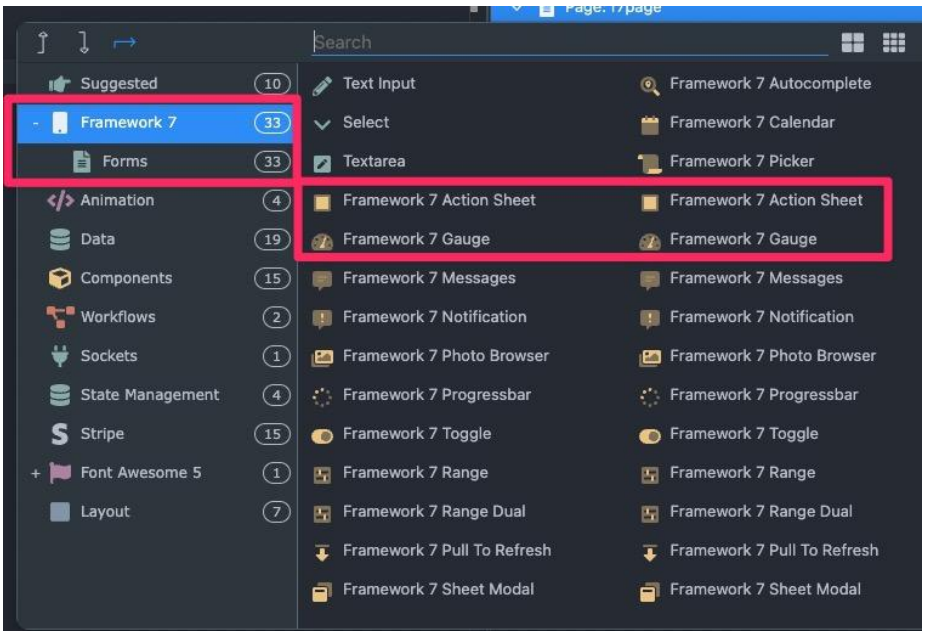
No nosso projeto de Sistema de Ocorrências Disciplinares, o Bootstrap será fundamental para construir a interface do usuário de forma rápida e eficiente.
Como o Bootstrap seria utilizado:

- **Layout**: O sistema de grid seria usado para organizar o layout das páginas, como o dashboard, listagens e formulários, garantindo que sejam responsivos.
    - Ex: O dashboard pode ter um layout de 2 colunas em desktop (col-md-6 col-lg-4 para a sidebar e col-md-6 col-lg-8 para o conteúdo principal) e empilhar em telemóveis.
- **Componentes da UI**:
    - Navbar: Para o cabeçalho do sistema, com links para Home, Gestão de Usuários, Ocorrências, Relatórios e Logout.

- Formulários: Todos os formulários (Login, Cadastro de Usuários, Cadastro de Ocorrência) seriam estilizados com as classes de formulário do Bootstrap (.form-control, .form-label, .mb-3).
- Tabelas: As listagens de usuários e ocorrências seriam tabelas estilizadas com .table, .table-striped, etc.
- Botões: Todos os botões de ação (Submeter, Editar, Excluir) utilizariam as classes .btn do Bootstrap.
- Alertas: Mensagens de sucesso ou erro (ex: após um cadastro) poderiam usar as classes de alerta (.alert-success, .alert-danger).
- Modais: Para confirmações de exclusão ou formulários em pop-up.

- **Utilitários**: Classes de utilidade para espaçamento (.m-, .p-), alinhamento de texto (.text-center), display (.d-flex) e responsividade (.d-none .d-md-block).
- Ao invés de escrever CSS personalizado para cada um desses elementos, usaríamos as classes pré-definidas do Bootstrap, o que economizaria muito tempo e garantiria uma aparência profissional.

Ilustração 4.5: Aplicação do Bootstrap no Sistema de Ocorrências.



Fonte: https://community.wappler.io

Exercícios:

1. Crie um novo arquivo HTML (exercicio_bootstrap.html) e inclua o Bootstrap 5 via CDN.
2. Dentro do body, crie um div com a classe container.
3. Dentro do container, crie uma linha (.row) com três colunas de largura igual (.col). Adicione um texto simples dentro de cada coluna e veja como elas se comportam ao redimensionar a janela do navegador.
4. Substitua as colunas do exercício 3 por uma estrutura que mostre:
    - Uma coluna de largura total em telemóveis (col-12).
    - Duas colunas de largura igual em tablets (col-md-6).
    - Três colunas de largura igual em desktops (col-lg-4).
    - Adicione um fundo (bg-light) e borda (border) a cada coluna para melhor visualização.
5. Adicione um formulário simples de contacto com campos para nome, email e mensagem, utilizando as classes de formulário do Bootstrap para estilização. Inclua um botão de submissão.

Módulo 5: Fundamentos de JavaScript para Web

Até agora, aprendemos a estruturar uma página com HTML e a estilizá-la com CSS. Mas e se quisermos que a página responda às ações do usuário, como cliques em botões, preenchimento de formulários ou animações dinâmicas? É para isso que serve o **JavaScript!** O JavaScript é a linguagem de programação que torna as páginas web interativas e dinâmicas.

5.1 O que é JavaScript?

JavaScript (JS) é uma linguagem de programação leve, interpretada ou compilada *just-in-time (*no Node.js*)*, com funções de primeira classe. É mais conhecida como a linguagem de script para páginas Web, mas também é usada em muitos ambientes fora do navegador, como Node.js (para servidores), aplicações desktop, e até mesmo hardware.

O que o JavaScript faz na web?

- **Interatividade:** Responde a eventos (cliques, passagem do rato, submissão de formulários).
- **Manipulação do conteúdo:** Altera, adiciona ou remove elementos HTML e seus estilos CSS dinamicamente.
- **Validação de dados:** Verifica se os dados inseridos em formulários estão corretos antes de serem enviados ao servidor.
- **Comunicação com servidores:** Permite carregar dados sem recarregar a página (AJAX).
- **Animações e efeitos visuais:** Cria galerias de imagens, sliders, e outros efeitos.

Ilustração 4.1: Uma página web dinâmica com JavaScript.

Fonte: crocoblock.com

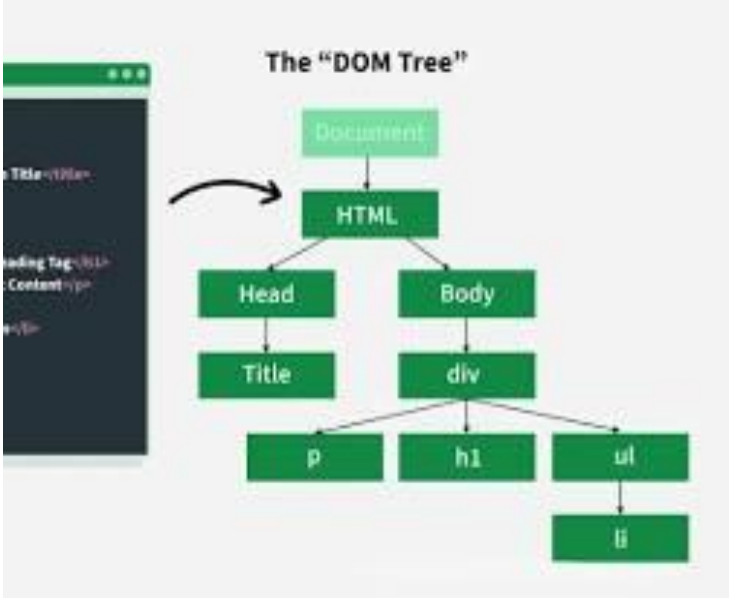
5.2 O DOM (*Document Object Model*)

Para que o JavaScript possa interagir com o HTML e o CSS, ele precisa de uma forma de "entender" a estrutura da página. Essa forma é o **DOM (*Document Object Model*)**.

O DOM é uma representação em árvore da sua página HTML. Cada elemento HTML (como um <p>, , <img>) é um **nó** na árvore do DOM. O JavaScript pode aceder a esses nós, modificá-los, adicionar novos, ou removê-los.

Pense no DOM como um mapa detalhado da sua página, que o JavaScript pode usar para encontrar e manipular qualquer parte dela.

Ilustração 5.2: Representação simplificada do DOM como uma árvore.



Fonte: www.geeksforgeeks.org

5.3 Manipulação de elementos HTML com JavaScript

A principal forma de interagir com o HTML usando JavaScript é por meio da manipulação do DOM. Existem vários métodos para selecionar elementos:

- document.getElementById('idDoElemento'): Seleciona um único elemento pelo seu atributo id.
- document.getElementsByClassName('nomeDaClasse'): Seleciona uma coleção de elementos pela sua class.
- document.getElementsByTagName('nomeDaTag'): Seleciona uma coleção de elementos pela sua tag.
- document.querySelector('seletorCSS'): Seleciona o *primeiro* elemento que corresponde a um seletor CSS (muito útil!).
- document.querySelectorAll('seletorCSS'): Seleciona *todos* os elementos que correspondem a um seletor CSS.

Depois de selecionar um elemento, pode modificar as suas propriedades, conteúdo ou estilo:

```
// Exemplo de manipulação do DOM
// 1. Selecionar um elemento pelo ID
const titulo = document.getElementById('tituloPrincipal');
console.log(titulo.textContent); // Exibe o texto atual do título

// 2. Modificar o conteúdo de texto
titulo.textContent = 'Olá, JavaScript!';
titulo.style.color = '#2563eb'; // Mudar a cor usando estilo inline via JS
```

```
// 3. Selecionar vários elementos por classe
const paragrafos = document.querySelectorAll('.paragrafo-estilo');

// Iterar sobre os parágrafos e adicionar uma classe CSS
paragrafos.forEach(paragrafo => {
  paragrafo.classList.add('texto-destaque'); // Adiciona uma classe CSS
});

// 4. Criar um novo elemento e adicioná-lo à página
const novoParagrafo = document.createElement('p');
novoParagrafo.textContent = 'Este parágrafo foi adicionado pelo JavaScript!';
novoParagrafo.classList.add('text-gray-600', 'mt-4', 'italic');

const container = document.getElementById('containerPrincipal');
container.appendChild(novoParagrafo); // Adiciona o novo parágrafo ao final do container
```

5.4 Eventos

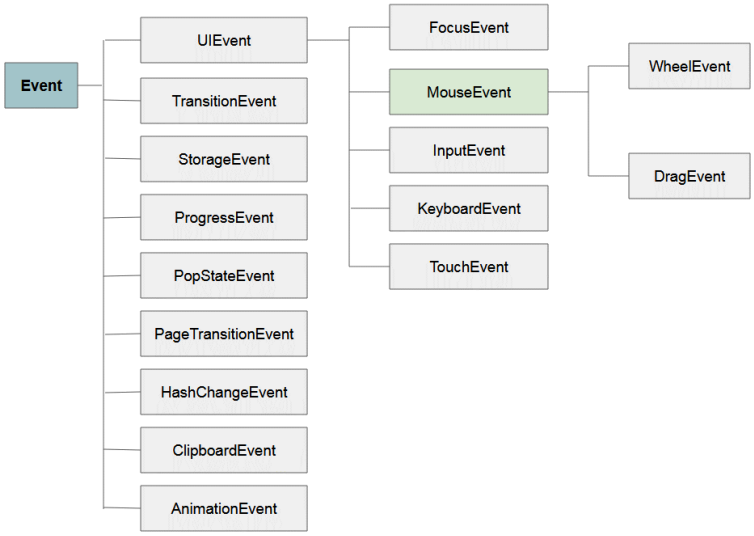
Os **eventos** são ações que acontecem na página Web, como cliques do usuário, carregamento da página, movimento do rato, pressionar de teclas, etc. O JavaScript pode "escutar" esses eventos e executar código em resposta a eles.

O método mais comum para lidar com eventos é addEventListener():

```
// Exemplo de manipulação de eventos
// Seleciona o botão
const meuBotao = document.getElementById('meuBotao');
const mensagemArea = document.getElementById('mensagemArea');

// Adiciona um "escutador de eventos" (event listener) para o clique no botão
meuBotao.addEventListener('click', function() {
  mensagemArea.textContent = 'O botão foi clicado!';
  mensagemArea.classList.remove('hidden'); // Torna a mensagem visível
  mensagemArea.classList.add('text-green-600', 'font-bold');
});

// Exemplo de evento de entrada de texto
const campoTexto = document.getElementById('campoTexto');
const contadorCaracteres = document.getElementById('contadorCaracteres');

campoTexto.addEventListener('input', function() {
  const numCaracteres = campoTexto.value.length;
  contadorCaracteres.textContent = `Caracteres: ${numCaracteres}`;
```

```
});
```

Ilustração 4.3: Um evento de clique a ativar uma função JavaScript.



Fonte: https://o7planning.org

5.5 Inclusão de JavaScript

Assim como o CSS, o JavaScript pode ser incluído de três formas:

1. **Inline:** Diretamente no atributo de uma tag HTML (pouco recomendado para a maioria dos casos).

```
<button onclick="alert('Olá do JS inline!');">Clique-me</button>
```

2. **Interno (Embedded):** Dentro da tag <script> no HTML, geralmente antes do fechamento da tag </body> para garantir que o HTML já foi carregado.

```
<body>
  <!-- Conteúdo HTML -->
  <script>
    // Código JavaScript aqui
    console.log('Script interno carregado!');
  </script>
</body>
```

3. **Externo:** A forma mais recomendada. O JavaScript é escrito num arquivo .js separado e ligado ao HTML com a tag <script src="caminho/do/seu/script.js"></script>. Colocar o <script> externo antes de </body> é uma boa prática.

```
<!-- index.html -->
```

```
<body>
  <!-- Conteúdo HTML -->
  <script src="meu_script.js"></script>
</body>
```

5.6 Bibliotecas JavaScript úteis (Ex: jQuery)

Escrever todo o código JavaScript "do zero" pode ser repetitivo e complexo para tarefas comuns. É por isso que existem as **bibliotecas JavaScript**. Uma biblioteca é um conjunto de código JavaScript pré-escrito que oferece funções prontas para facilitar o desenvolvimento.

5.6.1 jQuery

jQuery é uma das bibliotecas JavaScript mais populares e antigas, conhecida por simplificar a manipulação do DOM, gestão de eventos e animações. Embora frameworks mais modernos como React ou Vue.js sejam amplamente usados hoje, o jQuery ainda é encontrado em muitos projetos e é excelente para entender os conceitos de abstração que as bibliotecas oferecem.

Por que usar jQuery?

- **Sintaxe simplificada:** Torna a seleção e manipulação do DOM muito mais fácil e concisa.
- **Compatibilidade entre navegadores:** Resolve muitas inconsistências entre navegadores, que eram um problema maior no passado.
- **Manipulação de eventos:** Simplifica a adição e remoção de "*event listeners*".
- **AJAX:** Facilita as requisições assíncronas ao servidor.

Como incluir jQuery:

Você pode incluir o jQuery no seu projeto usando um CDN (*Content Delivery Network*), que é a forma mais fácil:

```
<head>
  <!-- Outras tags head -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
```

```
</head>
```

Sempre inclua o jQuery *antes* de qualquer script JavaScript que o utilize.

Exemplo de jQuery vs. JavaScript puro:

Ação	JavaScript puro	jQuery
Selecionar um elemento	document.getElementById('meuId')	$('#meuId')
Selecionar por classe	document.querySelectorAll('.minhaClasse')	$('.minhaClasse')
Clicar num botão	elemento.addEventListener('click', function(){})	$('#meuBotao').click(function(){})
Mudar texto	elemento.textContent = 'Novo Texto'	$('#meuId').text('Novo Texto')
Mudar CSS	elemento.style.color = 'red'	$('#meuId').css('color', 'red')
Esconder/ Mostrar	elemento.style.display = 'none'	$('#meuId').hide() / $('#meuId').show()

5.6.2 Outras bibliotecas úteis:

- **Axios / Fetch API (não é uma biblioteca, é uma API nativa):** Para fazer requisições HTTP (AJAX) de forma mais moderna e robusta que o jQuery.
- **Lodash / Underscore.js:** Bibliotecas de utilidades que fornecem funções para manipulação de arrays, objetos, strings, etc.
- **Moment.js / Luxon:** Para trabalhar com datas e horas.
- **Chart.js / D3.js:** Para criar gráficos e visualizações de dados interativas.

Exemplo de código JavaScript

```
<!DOCTYPE html>
<!--
  Este arquivo HTML demonstra os fundamentos de JavaScript e como
  a biblioteca jQuery pode ser utilizada para simplificar a manipulação do DOM e eventos.
-->
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Interatividade com JavaScript e jQuery</title>
```

```
  <!-- Inclui o Tailwind CSS via CDN para uma estilização rápida e responsiva -->
  <script src="https://cdn.tailwindcss.com"></script>
  <!-- Inclui o jQuery via CDN. É importante carregar o jQuery antes de qualquer script
       que o utilize, para que a função $() esteja disponível. -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
  <style>
    /* Define a fonte "Inter" como padrão para todo o documento */
    body {
      font-family: "Inter", sans-serif;
    }
  </style>
</head>
<body class="bg-gray-50 min-h-screen flex items-center justify-center p-4">
  <!-- Contenedor principal com estilo de cartão, centralizado na tela -->
  <div id="mainContainer" class="bg-white p-8 rounded-2xl shadow-xl max-w-2xl w-full text-
center">
    <h1 class="text-4xl font-bold text-gray-800 mb-6">JavaScript e jQuery na Prática</h1>

    <!-- Seção de demonstração de manipulação do DOM e eventos com JavaScript puro -->
    <section class="mb-8 p-6 bg-blue-50 rounded-lg shadow-sm">
      <h2 class="text-3xl font-semibold text-blue-700 mb-4">JavaScript Puro (DOM e
Eventos)</h2>
      <p id="textoDinamico" class="text-lg text-gray-700 mb-4">
        Este é o texto original. Observe como ele muda ao interagir.
      </p>
      <button id="mudarTextoBtn" class="bg-blue-600 hover:bg-blue-700 text-white font-
semibold py-2 px-6 rounded-lg transition duration-300 transform hover:scale-105 shadow-
md">
        Mudar Texto (JS Puro)
      </button>
      <p id="mensagemEstado" class="mt-4 text-green-600 font-medium hidden">
        Texto alterado com sucesso!
      </p>
    </section>

    <!-- Seção de demonstração com jQuery -->
    <section class="mb-8 p-6 bg-purple-50 rounded-lg shadow-sm">
      <h2 class="text-3xl font-semibold text-purple-700 mb-4">Ação com jQuery</h2>
      <p id="paragrafoJquery" class="text-lg text-gray-700 mb-4">
```

```
        Este é um parágrafo que será afetado pelo jQuery.
      </p>
      <button id="alternarVisibilidadeBtn" class="bg-purple-600 hover:bg-purple-700 text-white font-semibold py-2 px-6 rounded-lg transition duration-300 transform hover:scale-105 shadow-md mr-4">
        Alternar Visibilidade (jQuery)
      </button>
      <button id="animarCorBtn" class="bg-yellow-500 hover:bg-yellow-600 text-gray-800 font-semibold py-2 px-6 rounded-lg transition duration-300 transform hover:scale-105 shadow-md">
        Animar Cor (jQuery)
      </button>
    </section>

    <!-- Seção de demonstração de evento de entrada de texto -->
    <section class="p-6 bg-green-50 rounded-lg shadow-sm">
      <h2 class="text-3xl font-semibold text-green-700 mb-4">Entrada de Texto em Tempo Real</h2>
      <input type="text" id="meuInput" placeholder="Digite algo aqui..."
        class="w-full p-3 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-green-500 mb-4 text-gray-800">
      <p id="exibirInput" class="text-lg text-gray-700 font-semibold">
        O que você digita aparecerá aqui.
      </p>
    </section>

  </div>

  <!-- Script JavaScript principal -->
  <script>
    // === JavaScript Puro ===

    // Garante que o DOM esteja completamente carregado antes de executar o script
    document.addEventListener('DOMContentLoaded', function(){
      const textoDinamico = document.getElementById('textoDinamico');
      const mudarTextoBtn = document.getElementById('mudarTextoBtn');
      const mensagemEstado = document.getElementById('mensagemEstado');

      // Adiciona um listener de evento para o clique no botão
```

```
        mudarTextoBtn.addEventListener('click', function() {
            textoDinamico.textContent = 'O texto foi alterado pelo JavaScript puro!';
            textoDinamico.classList.add('text-blue-900', 'font-bold'); // Adiciona classes Tailwind
            mensagemEstado.classList.remove('hidden'); // Mostra a mensagem de estado
        });

        const meuInput = document.getElementById('meuInput');
        const exibirInput = document.getElementById('exibirInput');

        // Evento 'input' dispara a cada vez que o valor do input muda
        meuInput.addEventListener('input', function() {
            exibirInput.textContent = meuInput.value;
        });
    });

    // === JavaScript com jQuery ===

    // $(document).ready() é a forma jQuery de garantir que o DOM está carregado
    $(document).ready(function() {
        const $paragrafoJquery = $('#paragrafoJquery'); // Armazena o elemento jQuery numa
variável
        const $alternarVisibilidadeBtn = $('#alternarVisibilidadeBtn');
        const $animarCorBtn = $('#animarCorBtn');

        // Alternar visibilidade de um elemento com jQuery
        $alternarVisibilidadeBtn.on('click', function() {
            $paragrafoJquery.toggle(500); // .toggle() mostra ou esconde com animação de 500ms
        });

        // Animar a cor de texto de um elemento com jQuery
        let isBlue = true;
        $animarCorBtn.on('click', function() {
            if (isBlue) {
                $paragrafoJquery.css('color', 'red');
            } else {
                $paragrafoJquery.css('color', '#4a007f'); // Volta para a cor roxa (padrão)
            }
            isBlue = !isBlue; // Alterna o estado
        });
```

```
        // Exemplo de como usar jQuery para mudar o background do body após o carregamento
        $('body').css('background-color', '#e0f2f7'); // Mudar para um azul claro

    });
  </script>
</body>
</html>
```

Resultado do código (visualização)

Exercícios:

1. Crie uma nova página HTML (exercicio_js.html).
2. Adicione um parágrafo com um id="textoDinamico" e um botão com um id="mudarTextoBtn".

3. Inclua o jQuery via CDN no <head>.
4. Usando JavaScript (com jQuery e/ou puro):
    - Quando o botão for clicado, altere o texto do parágrafo para "O texto foi mudado pelo JavaScript!".
    - Altere a cor de fundo do body para um azul claro quando a página for totalmente carregada (window.onload ou $(document).ready()).
    - Adicione um campo de entrada de texto (<input type="text" id="meuInput">) e um parágrafo vazio ( ). À medida que o usuário digita no campo, o texto deve aparecer em tempo real no parágrafo abaixo.

Módulo 6: Programação Orientada a Objetos (POO)

Até agora, aprendemos a construir páginas web estáticas e a adicionar interatividade do lado do cliente com JavaScript. No entanto, para criar aplicações web complexas e escaláveis, precisamos de uma forma mais organizada e eficiente de escrever código. É aqui que entra a **Programação Orientada a Objetos (POO)**. A POO é um paradigma de programação que organiza o código em "objetos" – que são estruturas que combinam dados e as funções que operam sobre esses dados.

6.1 O que é Programação Orientada a Objetos?

A Programação Orientada a Objetos (POO), ou *Object-Oriented Programming* (OOP) em inglês, é uma abordagem de desenvolvimento de software que modela o mundo real em termos de objetos. Em vez de pensar em programas como uma sequência de instruções, pensamos neles como coleções de objetos que interagem entre si.

Principais vantagens da POO:

- **Modularidade:** O código é dividido em módulos independentes (objetos), tornando-o mais fácil de gerenciar.
- **Reusabilidade:** Objetos podem ser reutilizados em diferentes partes de um projeto ou em projetos futuros.
- **Manutenibilidade:** Códigos organizados são mais fáceis de entender, depurar e manter.
- **Flexibilidade:** Permite criar sistemas mais adaptáveis a mudanças.

Ilustração 6.1: A POO como blocos de construção de software.

Fonte: sam.se

6.2 Conceitos fundamentais da POO: Classes e Objetos

Para entender a POO, precisamos começar pelos seus blocos de construção essenciais: **Classes** e **Objetos**.

6.2.1 Classes

Uma **Classe** é como um "plano" ou "molde" para criar objetos. Ela define as características (propriedades ou atributos) e os comportamentos (funções ou métodos) que os objetos desse tipo terão. A classe não é um objeto em si, mas uma descrição de como um objeto deve ser.

Pense numa classe como o projeto de uma casa: o projeto descreve quantos quartos, banheiros, o tipo de telhado, mas não é a casa em si.

PHP

```
<?php
// Definição de uma Classe em PHP
class Carro {
    // Propriedades (Atributos) do Carro
    public $marca;
    public $modelo;
    public $cor;
    public $velocidadeAtual;

    // Construtor: um método especial que é chamado automaticamente
    // quando um novo objeto é criado a partir desta classe.
    public function __construct($marca, $modelo, $cor) {
```

```
    $this->marca = $marca;
    $this->modelo = $modelo;
    $this->cor = $cor;
    $this->velocidadeAtual = 0; // Inicia com velocidade zero
  }

  // Métodos (Comportamentos) do Carro
  public function acelerar($aumento){
    $this->velocidadeAtual += $aumento;
    echo "O " . $this->modelo . " acelerou para " . $this->velocidadeAtual . " km/h.<br>";
  }

  public function frear($diminuicao){
    $this->velocidadeAtual -= $diminuicao;
    if ($this->velocidadeAtual < 0){
      $this->velocidadeAtual = 0;
    }
    echo "O " . $this->modelo . " freou para " . $this->velocidadeAtual . " km/h.<br>";
  }

  public function exibirInfo(){
    echo "Marca: " . $this->marca . ", Modelo: " . $this->modelo . ", Cor: " . $this->cor . ",
Velocidade: " . $this->velocidadeAtual . " km/h.<br>";
  }
}
?>
```

6.2.2 Objetos

Um **Objeto** é uma instância concreta de uma classe. É a "materialização" do molde. Você pode criar múltiplos objetos a partir da mesma classe, e cada objeto terá os seus próprios valores para as propriedades, mas compartilhará os mesmos métodos.

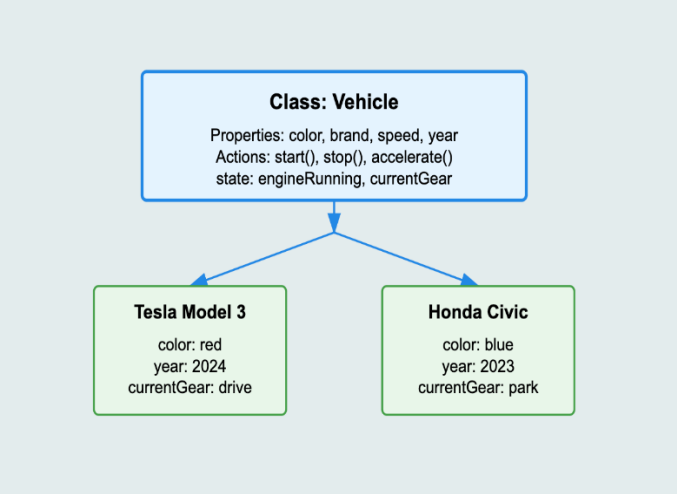
Continuando com a analogia da casa: o objeto é a casa real que foi construída a partir do projeto. Você pode construir várias casas (objetos) a partir do mesmo projeto (classe).

PHP

```
<?php
// Criação de Objetos (Instâncias da Classe Carro)

// Criando o primeiro objeto: um carro da marca Toyota, modelo Corolla, cor Prata
$meuCarro = new Carro("Toyota", "Corolla", "Prata");

// Criando o segundo objeto: um carro da marca Ford, modelo Focus, cor Azul
$carroDoAmigo = new Carro("Ford", "Focus", "Azul");
```

```
// Acessando propriedades dos objetos
echo "Meu carro é um " . $meuCarro->marca . " " . $meuCarro->modelo . ".<br>";
echo "O carro do meu amigo é um " . $carroDoAmigo->marca . " " . $carroDoAmigo->modelo .
".<br>";

// Chamando métodos dos objetos
$meuCarro->acelerar(50); // O Corolla acelerou para 50 km/h.
$carroDoAmigo->acelerar(80); // O Focus acelerou para 80 km/h.

$meuCarro->exibirInfo();    // Marca: Toyota, Modelo: Corolla, Cor: Prata, Velocidade: 50 km/h.
$carroDoAmigo->exibirInfo(); // Marca: Ford, Modelo: Focus, Cor: Azul, Velocidade: 80 km/h.

$meuCarro->frear(20);       // O Corolla freou para 30 km/h.
$meuCarro->exibirInfo();    // Marca: Toyota, Modelo: Corolla, Cor: Prata, Velocidade: 30 km/h.
?>
```

Ilustração 6.2: Classe como molde, Objetos como instâncias.



Fonte: www.udacity.com

6.3 Atributos (propriedades) e Métodos (comportamentos)

Dentro de uma classe, definimos:

- **Atributos (ou Propriedades):** São as variáveis que representam as características ou estados de um objeto. No exemplo do Carro, marca, modelo, cor e velocidadeAtual são atributos. Eles definem "o que" o objeto é ou "como" ele está.
- **Métodos (ou Comportamentos):** São as funções que representam as ações que um objeto pode realizar ou as operações que podem ser feitas sobre ele. No exemplo do Carro, acelerar(), frear() e exibirInfo() são métodos. Eles definem "o que" o objeto pode fazer.

Visibilidade (encapsulamento básico)

Em POO, é comum definir a **visibilidade** dos atributos e métodos para controlar como eles podem ser acedidos de fora da classe. Isso faz parte do princípio do **Encapsulamento**.

- **public**: O atributo ou método pode ser acedido de qualquer lugar (dentro ou fora da classe). Usamos public nos exemplos acima.
- **private**: O atributo ou método só pode ser acedido de dentro da própria classe.
- **protected**: O atributo ou método pode ser acedido de dentro da própria classe e de classes que herdam dela (ver Herança).

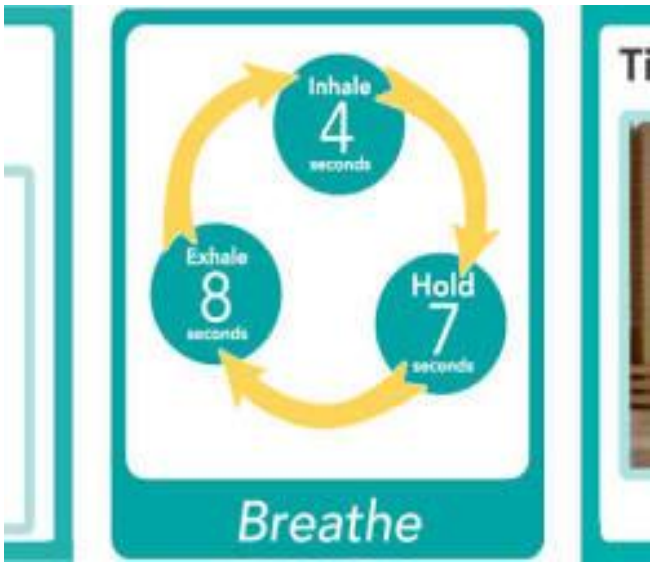
É uma boa prática tornar os atributos private ou protected e fornecer métodos public (chamados "*getters*" e "*setters*") para aceder e modificar esses atributos. Isso permite um controle maior sobre como os dados são manipulados.

PHP

```php
<?php
class Pessoa {
  private $nome; // Atributo privado
  private $idade; // Atributo privado

  public function __construct($nome, $idade) {
    $this->nome = $nome;
    $this->setIdade($idade); // Usando o setter para validar a idade
  }

  // Método público para obter o nome (Getter)
  public function getNome() {
    return $this->nome;
  }

  // Método público para definir a idade (Setter) com validação
  public function setIdade($novaIdade) {
    if ($novaIdade >= 0 && $novaIdade <= 150) {
      $this->idade = $novaIdade;
    } else {
      echo "Idade inválida!<br>";
    }
  }

  // Método público para obter a idade (Getter)
  public function getIdade() {
    return $this->idade;
  }

  // Método público para um comportamento
  public function apresentar() {
```

```
        echo "Olá, sou " . $this->nome . " e tenho " . $this->idade . " anos.<br>";
    }
}

$p1 = new Pessoa("Ana", 25);
$p1->apresentar(); // Olá, sou Ana e tenho 25 anos.

// Tentando aceder diretamente a um atributo privado (resultará em erro/warning se não
houver getter/setter)
// echo $p1->nome; // Isso geraria um erro se 'nome' fosse private e não houvesse getter.
echo $p1->getNome() . "<br>"; // Acessando via getter

$p1->setIdade(180); // Idade inválida!
$p1->apresentar(); // A idade continua a ser 25, pois a alteração inválida foi ignorada.
?>
```

Ilustração 6.3: Atributos e Métodos de um Objeto.



Fonte: autisminternetmodules.org

6.4 Pilares da POO: Encapsulamento, Herança, Polimorfismo e Abstração

Estes são os quatro princípios fundamentais que guiam o design de sistemas orientados a objetos:

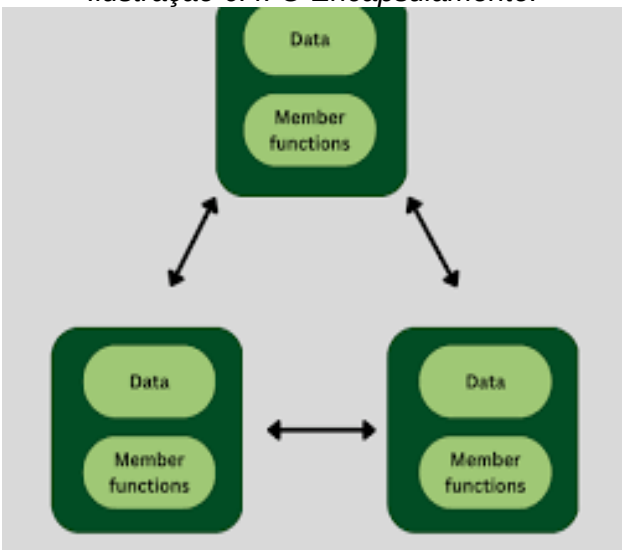
6.4.1 Encapsulamento

O **Encapsulamento** é o princípio de agrupar os dados (atributos) e os métodos que operam sobre esses dados numa única unidade (a classe) e de esconder os detalhes internos da implementação de um objeto, expondo apenas o que é necessário. Isso é feito usando modificadores de visibilidade (public, private, protected).

Benefícios:

- **Controle de acesso:** Impede que dados internos sejam modificados de forma inadequada.
- **Proteção de dados:** Garante a integridade dos dados do objeto.
- **Flexibilidade:** Permite mudar a implementação interna de uma classe sem afetar o código que a utiliza.

Ilustração 6.4: O Encapsulamento.



Fonte: logicmojo.com

6.4.2 Herança

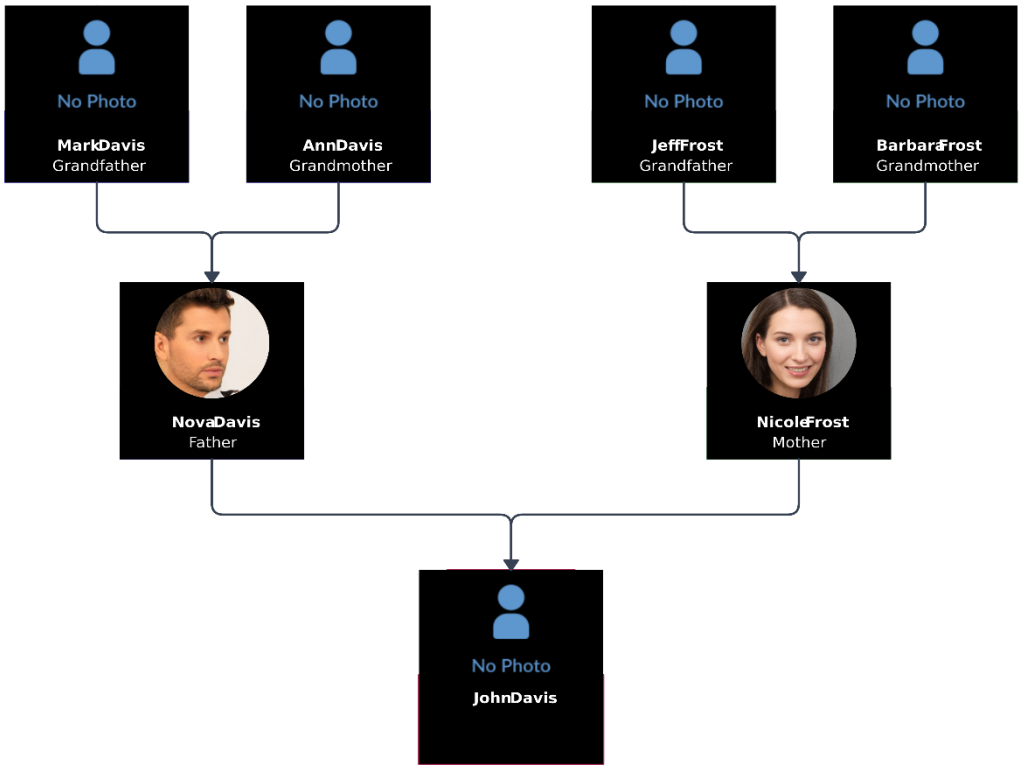
A **Herança** permite que uma classe (chamada **classe filha** ou subclasse) adquira as propriedades e métodos de outra classe (chamada **classe pai** ou superclasse). Isso promove a reusabilidade de código e estabelece uma relação "é um tipo de" entre classes (ex: um Cachorro *é um tipo de* Animal).

PHP

```php
<?php
// Classe Pai (Superclasse)
class Animal {
    public $nome;

    public function __construct($nome) {
        $this->nome = $nome;
    }

    public function fazerBarulho() {
        echo $this->nome . " faz um barulho.<br>";
    }
}

// Classe Filha (Subclasse) que herda de Animal
class Cachorro extends Animal {
    public function fazerBarulho() {
        echo $this->nome . " late: Au Au!<br>";
    }
    public function buscarBola() {
        echo $this->nome . " está buscando a bola.<br>";
    }
}

// Classe Filha (Subclasse) que herda de Animal
class Gato extends Animal {
    public function fazerBarulho() {
        echo $this->nome . " mia: Miau!<br>";
    }
    public function arranharMoveis() {
        echo $this->nome . " está arranhando os móveis.<br>";
    }
}

$meuCachorro = new Cachorro("Rex");
$meuGato = new Gato("Félix");

$meuCachorro->fazerBarulho(); // Rex late: Au Au!
$meuCachorro->buscarBola();   // Rex está buscando a bola.

$meuGato->fazerBarulho();     // Félix mia: Miau!
```

```
$meuGato->arranharMoveis();  // Félix está arranhando os móveis.
?>
```

Ilustração 6.5: A Herança em POO.



Fonte: creately.com

6.4.3 Polimorfismo

Polimorfismo significa "muitas formas". Em POO, refere-se à capacidade de objetos de diferentes classes responderem à mesma mensagem (chamada de método) de maneiras diferentes, desde que compartilhem uma interface comum (como herdar da mesma classe pai ou implementar a mesma interface).

No exemplo acima, tanto Cachorro quanto Gato têm o método fazerBarulho(), mas cada um o implementa de forma diferente.

PHP

```
<?php
// Usando o exemplo anterior
```

```
function descreverAnimal(Animal $animal) {
   echo "O animal " . $animal->nome . " faz: ";
   $animal->fazerBarulho(); // Polimorfismo em ação!
}

$animalGenerico = new Animal("Bicho");
descreverAnimal($animalGenerico); // O animal Bicho faz: Bicho faz um barulho.

$meuCachorro = new Cachorro("Rex");
descreverAnimal($meuCachorro);    // O animal Rex faz: Rex late: Au Au!

$meuGato = new Gato("Félix");
descreverAnimal($meuGato);       // O animal Félix faz: Félix mia: Miau!
?>
```

Ilustração 6.6: O Polimorfismo.

Fonte: unfocussed.com

6.4.4 Abstração

A **Abstração** é o processo de simplificar a realidade complexa, focando apenas nos detalhes essenciais e ocultando informações irrelevantes. Em POO, isso é alcançado através de classes abstratas e interfaces.

- **Classes abstratas:** Classes que não podem ser instanciadas diretamente e podem conter métodos abstratos (métodos sem implementação), que devem ser implementados pelas classes filhas.

- **Interfaces:** Definem um conjunto de métodos que uma classe deve implementar, sem fornecer nenhuma implementação. Uma classe pode implementar várias interfaces.

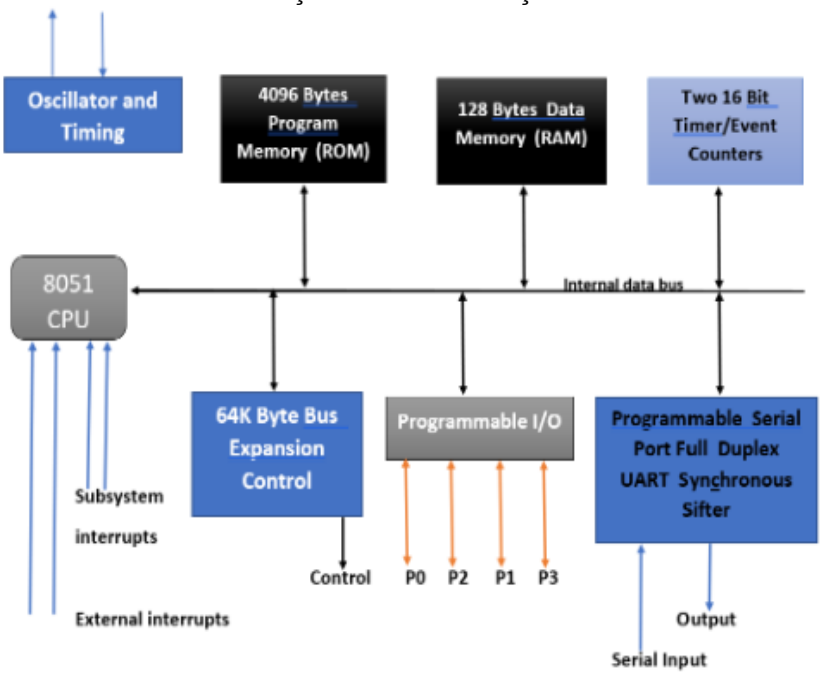
Benefícios:

- **Simplificação:** Reduz a complexidade ao expor apenas o necessário.
- **Foco:** Ajuda a focar nos aspetos importantes de um sistema.
- **Contrato:** Interfaces criam um "contrato" que as classes devem seguir.

PHP

```php
<?php
// Exemplo de Classe Abstrata
abstract class Veiculo {
  public $cor;

  public function __construct($cor) {
    $this->cor = $cor;
  }

  // Método abstrato: deve ser implementado pelas classes filhas
  abstract public function mover();

  public function getCor() {
    return $this->cor;
  }
}

class Moto extends Veiculo {
  public function mover() {
    echo "A moto está a andar sobre duas rodas.<br>";
  }
}

class CarroPasseio extends Veiculo {
  public function mover() {
    echo "O carro está a rolar sobre quatro rodas.<br>";
  }
}

$minhaMoto = new Moto("Vermelha");
echo "Cor da moto: " . $minhaMoto->getCor() . "<br>";
$minhaMoto->mover(); // A moto está a andar sobre duas rodas.

$meuCarroPasseio = new CarroPasseio("Preto");
echo "Cor do carro: " . $meuCarroPasseio->getCor() . "<br>";
$meuCarroPasseio->mover(); // O carro está a rolar sobre quatro rodas.
?>
```

Ilustração 6.7: A Abstração.



Fonte: www.geeksforgeeks.org

Exercícios:

1. Crie uma classe Livro com atributos titulo, autor e anoPublicacao. Adicione um construtor e um método exibirDetalhes() que imprima as informações do livro. Crie dois objetos Livro e chame o método exibirDetalhes() para cada um.
2. Modifique a classe Livro para que os atributos titulo e autor sejam private. Crie métodos public getTitulo() e setTitulo() (e similarmente para autor) para aceder e modificar esses atributos.
3. Crie uma classe pai Funcionario com atributos nome e salario e um método calcularSalarioAnual().
4. Crie uma classe filha Gerente que herde de Funcionario e adicione um atributo departamento. Sobrescreva o método calcularSalarioAnual() na classe Gerente para incluir um bónus de 10% no salário. Crie um objeto Gerente e teste.

Módulo 7: Introdução ao PHP e Programação no lado do servidor (*server-side*)

Até agora, focamos no "front-end" (lado do cliente) com HTML, CSS e JavaScript. No entanto, para criar aplicações web dinâmicas que interagem com bases de dados, gerem sessões de usuário, ou processam lógica de negócio complexa, precisamos de uma linguagem de "back-end" (lado do servidor). É aqui que o **PHP** entra em cena!

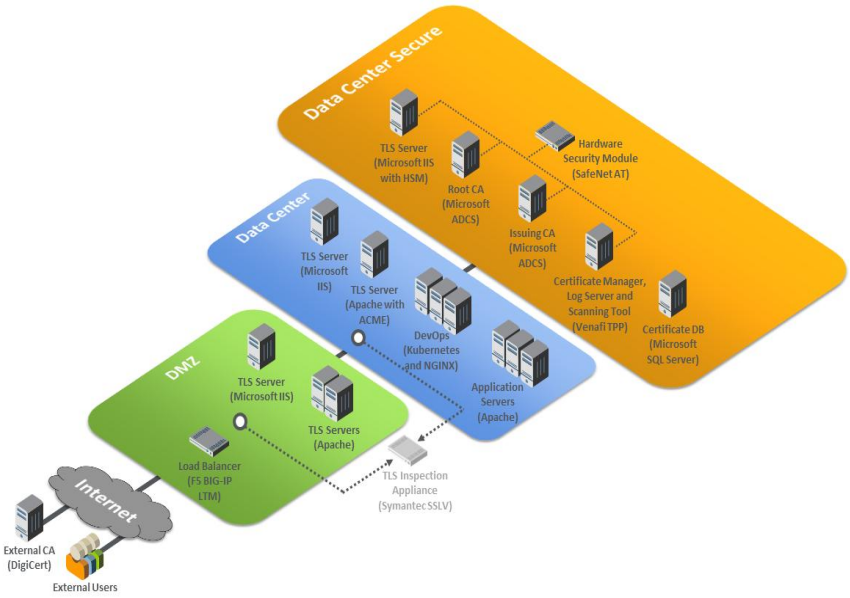
7.1 O que é PHP?

PHP (um acrônimo recursivo para "PHP: *Hypertext Preprocessor*") é uma linguagem de *script* de propósito geral, de código aberto, amplamente utilizada, especialmente adequada para o desenvolvimento web. É executada no lado do servidor, o que significa que o código PHP é processado num servidor web e o resultado (geralmente HTML) é enviado para o navegador do cliente.

Por que usar PHP para desenvolvimento web?

- **Fácil de aprender:** A sua sintaxe é relativamente simples e tem uma curva de aprendizagem suave, especialmente para quem já tem alguma base em lógica de programação.
- **Integração com HTML:** Pode ser incorporado diretamente em arquivos HTML, o que facilita a geração de conteúdo dinâmico.
- **Vasta comunidade e documentação:** Existem muitos recursos, tutoriais e uma grande comunidade para ajudar a resolver problemas.
- **Compatibilidade com bases de dados:** Suporta uma vasta gama de bases de dados, como MySQL, PostgreSQL, SQLite, entre outras.
- **Ecossistema rico:** Existem muitos *frameworks* (como o CodeIgniter, que veremos a seguir) e sistemas de gestão de conteúdo (CMS) como o WordPress, que são construídos em PHP.

Ilustração 7.1: O papel do PHP no lado do servidor.



Fonte: www.nccoe.nist.gov

7.2 Sintaxe Básica do PHP

Os arquivos PHP geralmente terminam com a extensão .php. O código PHP é executado dentro de marcadores especiais:

```
<?php
// O código PHP começa aqui
// ... seu código PHP ...
?>
// O código PHP termina aqui (opcional no final do arquivo)
```

Tudo o que estiver fora destes marcadores é tratado como texto puro (geralmente HTML).

7.2.1 Comentários

Comentários são linhas de código que são ignoradas pelo interpretador PHP. São essenciais para documentar o seu código e torná-lo mais compreensível.

```
<?php
// Este é um comentário de uma linha (estilo C++)
```

```
# Este também é um comentário de uma linha (estilo shell script)

/*
Este é um comentário de múltiplas linhas.
Ele pode cobrir várias linhas de código.
*/

echo "Olá, mundo!<br>"; // Você pode comentar no final da linha também
?>
```

7.2.2 Variáveis

Variáveis em PHP são usadas para armazenar informações. Elas começam com o símbolo do dólar ($), seguido pelo nome da variável. O PHP é uma linguagem de tipagem dinâmica, o que significa que não precisa de declarar o tipo da variável (ela assume o tipo do valor que lhe é atribuído).

```
<?php
$nome = "João";      // Variável de string
$idade = 30;         // Variável de número inteiro
$altura = 1.75;      // Variável de número de ponto flutuante
$estaAtivo = true;   // Variável booleana
$preco = 29.99;      // Variável de número de ponto flutuante

echo "Nome: " . $nome . "<br>";
echo "Idade: " . $idade . " anos<br>";
echo "Altura: " . $altura . " metros<br>";

// Concatenação de strings em PHP usa o operador de ponto (.)
echo "O " . $nome . " tem " . $idade . " anos.<br>";
?>
```

7.2.3 Tipos de Dados

PHP suporta vários tipos de dados:

- **String:** Sequência de caracteres (ex: "Olá mundo!").
- **Integer:** Números inteiros (ex: 10, -5).
- **Float (ou Double):** Números com casas decimais (ex: 3.14, 0.5).
- **Boolean:** Valores verdadeiros (true) ou falsos (false).

- **Array:** Uma coleção de valores (veremos mais detalhes à frente).
- **Object:** Instâncias de classes (visto no Módulo 5).
- **NULL:** Um valor especial que significa "sem valor".
- **Resource:** Uma variável especial que contém uma referência a um recurso externo (ex: uma ligação a uma base de dados).

```
<?php
$texto = "Apostila PHP";      // String
$numeroInteiro = 123;         // Integer
$numeroDecimal = 10.5;        // Float
$booleano = false;            // Boolean
$listaCores = array("vermelho", "verde", "azul"); // Array (introdução)

echo gettype($texto) . "<br>";       // string
echo gettype($numeroInteiro) . "<br>";  // integer
echo gettype($numeroDecimal) . "<br>";  // double
echo gettype($booleano) . "<br>";      // boolean
echo gettype($listaCores) . "<br>";    // array
?>
```

7.3 Operadores e Expressões

O PHP utiliza operadores muito semelhantes aos de outras linguagens de programação.

- **Operadores Aritméticos:** + (adição), - (subtração), * (multiplicação), / (divisão), % (módulo - resto da divisão), ** (exponenciação).
- **Operadores de Atribuição:** = (atribui), += (adiciona e atribui), -= (subtrai e atribui), etc.
- **Operadores de Comparação:** == (igual a valor), === (igual a valor e tipo), != (diferente de valor), !== (diferente de valor ou tipo), < (menor que), > (maior que), <= (menor ou igual), >= (maior ou igual).
- **Operadores Lógicos:** && (AND), || (OR), ! (NOT).

```
<?php
$x = 10;
$y = 5;

// Aritméticos
echo "Soma: " . ($x + $y) . "<br>";      // 15
echo "Multiplicação: " . ($x * $y) . "<br>"; // 50

// Atribuição
```

```
$z = 10;
$z += 5; // $z agora é 15
echo "Z após atribuição: " . $z . "<br>";

// Comparação
if ($x > $y) {
   echo "X é maior que Y<br>";
}

// Lógicos
$idade = 20;
$temHabilitacao = true;
if ($idade >= 18 && $temHabilitacao) {
   echo "Pode dirigir<br>";
}
?>
```

7.4 Estruturas condicionais (if, else, switch) e laços de repetição (for, while, foreach)

Assim como em JavaScript, o PHP oferece estruturas de controle de fluxo para tomar decisões e repetir blocos de código.

7.4.1 Estruturas condicionais

if, **else if**, **else**: Para executar código com base em condições.

```
<?php
$nota = 75;

if ($nota >= 60) {
   echo "Aprovado!<br>";
} else if ($nota >= 40) {
   echo "Recuperação.<br>";
} else {
   echo "Reprovado.<br>";
}
?>
```

switch: Para múltiplas escolhas com base no valor de uma variável.

```
<?php
$diaSemana = "Terça";
```

```
switch ($diaSemana) {
   case "Segunda":
      echo "Dia de recomeçar!<br>";
      break;
   case "Sexta":
      echo "Quase fim de semana!<br>";
      break;
   default:
      echo "Um dia normal...<br>";
}
?>
```

7.4.2 Laços de repetição

for: Quando se sabe o número de iterações.

```
<?php
echo "Contagem com for:<br>";
for ($i = 0; $i < 5; $i++) {
   echo "Número: " . $i . "<br>";
}
?>
```

while: Enquanto uma condição for verdadeira.

```
<?php
echo "Contagem com while:<br>";
$contador = 0;
while ($contador < 3) {
   echo "Passo: " . $contador . "<br>";
   $contador++;
}
?>
```

foreach: Especialmente útil para iterar sobre arrays.

```
<?php
$frutas = array("Maçã", "Banana", "Laranja");
echo "Minhas frutas:<br>";
foreach ($frutas as $fruta) {
   echo " - " . $fruta . "<br>";
}

$pessoa = array("nome" => "Carlos", "idade" => 28, "cidade" => "Lisboa");
echo "Informações da pessoa:<br>";
foreach ($pessoa as $chave => $valor) {
   echo $chave . ": " . $valor . "<br>";
```

```
}
?>
```

7.5 Funções em PHP

Funções são blocos de código reutilizáveis que realizam uma tarefa específica. Elas ajudam a organizar o seu código e a evitar repetições.

```
<?php
// Função sem parâmetros e sem retorno
function saudar(){
   echo "Olá! Bem-vindo ao PHP!<br>";
}
saudar(); // Chamada da função

// Função com parâmetros e retorno
function somar($num1, $num2){
   $resultado = $num1 + $num2;
   return $resultado;
}
$somaTotal = somar(15, 20);
echo "A soma é: " . $somaTotal . "<br>"; // A soma é: 35

// Função com parâmetro opcional e valor padrão
function cumprimentar($nome = "Visitante"){
   echo "Olá, " . $nome . "!<br>";
}
cumprimentar("Maria");   // Olá, Maria!
cumprimentar();        // Olá, Visitante!
?>
```

7.6 Manipulação de formulários (GET e POST)

Uma das tarefas mais comuns em desenvolvimento web backend é processar dados enviados por formulários HTML. O PHP tem superglobais ($_GET e $_POST) que facilitam muito este processo.

- **$_GET**: Usado para coletar dados enviados via método GET. Os dados são visíveis na URL (ideal para pesquisa, links).
- **$_POST**: Usado para coletar dados enviados via método POST. Os dados não são visíveis na URL (ideal para informações sensíveis, como senhas, ou grandes volumes de dados).

Exemplo Básico (HTML):

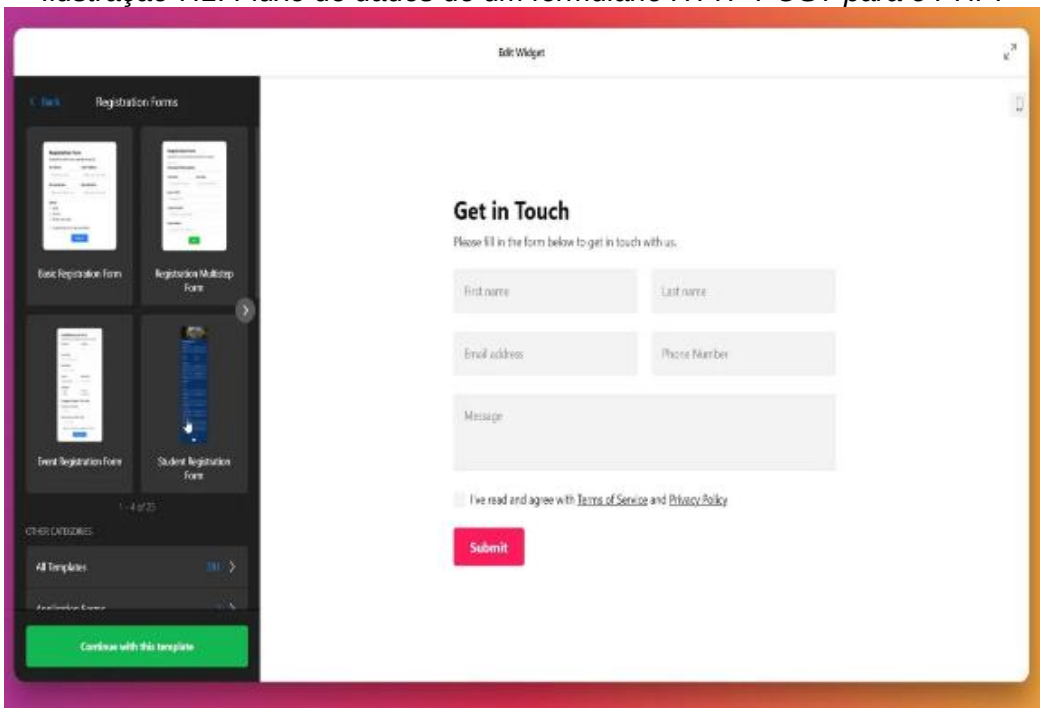
```
<!-- formulario.html -->
<form action="processar.php" method="POST">
  Nome: <input type="text" name="nome"><br>
  Email: <input type="email" name="email"><br>
  <button type="submit">Enviar</button>
</form>
```

Exemplo de Processamento (PHP - processar.php):

```
<?php
// Verifica se o formulário foi submetido via POST
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // Coleta os dados do formulário
  $nome = $_POST['nome'];
  $email = $_POST['email'];

  // Pode adicionar validação aqui
  if (empty($nome) || empty($email)) {
    echo "Por favor, preencha todos os campos.";
  } else {
    echo "Olá, " . htmlspecialchars($nome) . "!<br>";
    echo "O seu email é: " . htmlspecialchars($email) . "<br>";
  }
} else {
  echo "O formulário não foi submetido via POST.";
}
?>
```

Importante: Sempre use htmlspecialchars() ao exibir dados de usuários para prevenir ataques de Cross-Site Scripting (XSS).

Ilustração 7.1: Fluxo de dados de um formulário HTTP POST para o PHP.



Fonte: elfsight.com

7.7 Inclusão de arquivos

Para organizar o seu código PHP, pode dividi-lo em múltiplos arquivos e incluí-los onde for necessário. Isso é útil para cabeçalhos, rodapés, funções comuns ou classes.

- include 'nome_do_arquivo.php';: Inclui o arquivo. Se o arquivo não for encontrado, gera um WARNING mas o script continua a ser executado.
- require 'nome_do_arquivo.php';: Inclui o arquivo. Se o arquivo não for encontrado, gera um FATAL ERROR e o script é interrompido.
- include_once 'nome_do_arquivo.php';: Similar ao include, mas garante que o arquivo será incluído apenas uma vez, mesmo que seja chamado múltiplas vezes.
- require_once 'nome_do_arquivo.php';: Similar ao require, mas garante que o arquivo será incluído apenas uma vez.

É uma boa prática usar require_once para arquivos essenciais (como classes ou configurações) e include_once para partes não críticas.

```
<?php
// arquivo: cabecalho.php
echo "<header><h1>Meu Website PHP</h1></header>";
```

```
echo "<nav><a href='#'>Home</a> | <a href='#'>Sobre</a></nav>";

// arquivo: rodape.php
echo "<footer><p>&copy; 2024 Minha Empresa. Todos os direitos reservados.</p></footer>";

// arquivo: index.php
?>
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <title>Página Principal</title>
</head>
<body>
  <?php include 'cabecalho.php'; ?>

  <main>
    <h2>Bem-vindo à página principal!</h2>
    <p>Conteúdo dinâmico da página.</p>
  </main>

  <?php require 'rodape.php'; ?>
</body>
</html>
```

Exemplo completo de script PHP

```php
<?php
// Este arquivo demonstra os conceitos básicos de PHP.
// Para executá-lo, precisa de um servidor web com PHP instalado (ex: XAMPP, WAMP, MAMP)
// ou pode usar o servidor web integrado do PHP: php -S localhost:8000
// E aceda a http://localhost:8000/php_basico.php no seu navegador.

// Módulo 6.2: Sintaxe Básica, Comentários e Variáveis

// Comentário de uma linha
# Outro tipo de comentário de uma linha

/*
Este é um comentário
de múltiplas linhas
*/

echo "<h1>Módulo 6: Introdução ao PHP</h1>";
echo "<h2>6.2 Sintaxe Básica, Comentários e Variáveis</h2>";

// Declaração de variáveis
$nomeAluno = "Ana Silva"; // String
$idadeAluno = 17;       // Integer
$mediaNotas = 8.5;      // Float
$aprovado = true;       // Boolean
```

```php
echo "<p>Nome do Aluno: " . $nomeAluno . "</p>";
echo "<p>Idade: " . $idadeAluno . " anos</p>";
echo "<p>Média das Notas: " . $mediaNotas . "</p>";
echo "<p>Aprovado? " . ($aprovado ? "Sim" : "Não") . "</p>"; // Operador ternário para booleanos

echo "<hr>";

// Módulo 6.3: Operadores e Expressões
echo "<h2>6.3 Operadores e Expressões</h2>";

$num1 = 20;
$num2 = 7;

echo "<p>Soma ($num1 + $num2): " . ($num1 + $num2) . "</p>";
echo "<p>Subtração ($num1 - $num2): " . ($num1 - $num2) . "</p>";
echo "<p>Multiplicação ($num1 * $num2): " . ($num1 * $num2) . "</p>";
echo "<p>Divisão ($num1 / $num2): " . ($num1 / $num2) . "</p>";
echo "<p>Módulo ($num1 % $num2): " . ($num1 % $num2) . "</p>";

$resultadoComparacao = ($num1 > $num2);
echo "<p>$num1 é maior que $num2? " . ($resultadoComparacao ? "Verdadeiro" : "Falso") . "</p>";

$idadeParaVotar = 18;
$temTituloEleitor = true;

// Exemplo de operador lógico AND
if ($idadeAluno >= $idadeParaVotar && $aprovado) {
    echo "<p>O aluno pode votar e está aprovado.</p>";
} else {
    echo "<p>O aluno não cumpre todos os requisitos para votar e/ou não está aprovado.</p>";
}

echo "<hr>";

// Módulo 6.4: Estruturas Condicionais e Laços de Repetição
echo "<h2>6.4 Estruturas Condicionais e Laços de Repetição</h2>";

// Estrutura IF-ELSE IF-ELSE
$temperatura = 25;
if ($temperatura > 30) {
    echo "<p>Está muito quente!</p>";
} elseif ($temperatura > 20) {
    echo "<p>Temperatura agradável.</p>";
} else {
    echo "<p>Está frio.</p>";
}

// Estrutura SWITCH
$corFavorita = "azul";
switch ($corFavorita) {
    case "vermelho":
        echo "<p>A sua cor favorita é vermelho.</p>";
```

```
    break;
  case "azul":
    echo "<p>A sua cor favorita é azul.</p>";
    break;
  default:
    echo "<p>Não conheço a sua cor favorita.</p>";
}

// Laço FOR
echo "<h3>Laço FOR:</h3>";
for ($i = 1; $i <= 3; $i++) {
  echo "<p>Iteração número: " . $i . "</p>";
}

// Laço WHILE
echo "<h3>Laço WHILE:</h3>";
$j = 0;
while ($j < 2) {
  echo "<p>Contador WHILE: " . $j . "</p>";
  $j++;
}

// Laço FOREACH para arrays indexados
echo "<h3>Laço FOREACH (Arrays Indexados):</h3>";
$listaCompras = array("Pão", "Leite", "Ovos");
foreach ($listaCompras as $item) {
  echo "<p>- " . $item . "</p>";
}

// Laço FOREACH para arrays associativos
echo "<h3>Laço FOREACH (Arrays Associativos):</h3>";
$infoProduto = array(
  "nome" => "Laptop",
  "preco" => 1200,
  "estoque" => 50
);
foreach ($infoProduto as $chave => $valor) {
  echo "<p>" . ucfirst($chave) . ": " . $valor . "</p>"; // ucfirst() para deixar a primeira letra
maiúscula
}

echo "<hr>";

// Módulo 6.5: Funções em PHP
echo "<h2>6.5 Funções em PHP</h2>";

// Função simples
function saudacaoPersonalizada($nome) {
  return "Olá, " . htmlspecialchars($nome) . "! Seja bem-vindo(a).";
}

echo "<p>" . saudacaoPersonalizada("Mariana") . "</p>";
```

```php
echo "<p>" . saudacaoPersonalizada("Pedro") . "</p>";

// Função com valor de retorno e cálculo
function calcularAreaRetangulo($largura, $altura) {
    return $largura * $altura;
}

$area = calcularAreaRetangulo(10, 5);
echo "<p>A área do retângulo é: " . $area . " m²</p>";

echo "<hr>";

// Módulo 6.6: Manipulação de Formulários (GET e POST)
echo "<h2>6.6 Manipulação de Formulários</h2>";

// Este exemplo assume que este arquivo é 'php_basico.php'
// e que o formulário submete para este mesmo arquivo (action="")

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Processa dados POST
    $usuario = isset($_POST['usuario']) ? htmlspecialchars($_POST['usuario']) : "N/A";
    $senha = isset($_POST['senha']) ? "********" : "N/A"; // Não exiba senhas reais!

    echo "<h3>Dados Recebidos (POST):</h3>";
    echo "<p>Nome de Usuário: " . $usuario . "</p>";
    echo "<p>Senha: " . $senha . "</p>";
} elseif ($_SERVER["REQUEST_METHOD"] == "GET") {
    // Processa dados GET
    $parametroGet = isset($_GET['param']) ? htmlspecialchars($_GET['param']) : "Nenhum parâmetro GET.";
    echo "<h3>Dados Recebidos (GET):</h3>";
    echo "<p>Parâmetro 'param' na URL: " . $parametroGet . "</p>";
}

// Formulário de exemplo para POST
echo "<h3>Formulário POST:</h3>";
echo '<form method="POST" action="php_basico.php">
        <label for="usuario">Nome de Usuário:</label><br>
        <input type="text" id="usuario" name="usuario" class="p-2 border rounded-md"><br><br>
        <label for="senha">Senha:</label><br>
        <input type="password" id="senha" name="senha" class="p-2 border rounded-md"><br><br>
        <button type="submit" class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded-md">Enviar (POST)</button>
    </form>';

// Exemplo de link com parâmetro GET
echo "<h3>Exemplo de Link GET:</h3>";
echo '<a href="php_basico.php?param=exemplo_get" class="text-blue-600 hover:underline">Testar GET com link</a>';

echo "<hr>";
```

```php
// Módulo 6.7: Inclusão de Arquivos
echo "<h2>6.7 Inclusão de Arquivos</h2>";

// Criar arquivos temporários para demonstração
file_put_contents('cabecalho_temp.php', '<div class="bg-gray-200 p-4 text-center rounded-md mb-4"><h2>Cabeçalho Simples</h2></div>');
file_put_contents('rodape_temp.php', '<div class="bg-gray-200 p-4 text-center rounded-md mt-4"><p>&copy; ' . date('Y') . ' Apostila PHP</p></div>');
file_put_contents('funcoes_temp.php', '<?php function saudacaoTemp() { echo "<p class=\'text-indigo-600\'>Esta saudação vem de um arquivo incluído!</p>"; } ?>');

// Inclui o cabeçalho
include_once 'cabecalho_temp.php';

// Inclui funções
include_once 'funcoes_temp.php';
saudacaoTemp();

echo "<p>Este é o conteúdo principal da página.</p>";

// Inclui o rodapé
include_once 'rodape_temp.php';

// Limpa os arquivos temporários após a execução (opcional, para ambiente de demo)
unlink('cabecalho_temp.php');
unlink('rodape_temp.php');
unlink('funcoes_temp.php');

?>
```

CSS

```css
<!-- Estilos Tailwind CSS para o corpo do documento e elementos PHP -->
<style>
  body {
    font-family: "Inter", sans-serif;
    line-height: 1.6;
    color: #333;
    background-color: #f4f7f6;
    padding: 20px;
    max-width: 800px;
    margin: 20px auto;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
  }
  h1, h2, h3 {
    color: #2c3e50;
    margin-bottom: 15px;
  }
  p {
    margin-bottom: 10px;
  }
  hr {
```

```
    border: 0;
    height: 1px;
    background: #eee;
    margin: 20px 0;
  }
  code {
    background-color: #e6e6e6;
    padding: 2px 4px;
    border-radius: 4px;
    font-family: monospace;
  }
  .p-2 { padding: 0.5rem; }
  .border { border-width: 1px; }
  .rounded-md { border-radius: 0.375rem; }
  .bg-blue-500 { background-color: #3b82f6; }
  .hover\:bg-blue-700:hover { background-color: #1d4ed8; }
  .text-white { color: #fff; }
  .font-bold { font-weight: 700; }
  .py-2 { padding-top: 0.5rem; padding-bottom: 0.5rem; }
  .px-4 { padding-left: 1rem; padding-right: 1rem; }
  .rounded-md { border-radius: 0.375rem; }
  .hidden { display: none; } /* Para demonstração do hidden na mensagem de estado */
  .bg-gray-200 { background-color: #e2e8f0; }
  .text-center { text-align: center; }
  .mb-4 { margin-bottom: 1rem; }
  .mt-4 { margin-top: 1rem; }
  .text-indigo-600 { color: #4f46e5; }
</style>
```

Exercícios:

1. Crie um arquivo info.php que exiba o seu nome, idade e cidade usando variáveis PHP.
2. Crie um formulário HTML (calculadora.html) com dois campos numéricos e um botão de submissão.
3. Crie um arquivo PHP (processar_calculadora.php) que receba os dois números do formulário, realize as quatro operações aritméticas básicas (adição, subtração, multiplicação, divisão) e exiba os resultados.
4. Modifique o processar_calculadora.php para usar uma função para cada operação aritmética (ex: function adicionar($n1, $n2)).
5. Crie um arquivo menu.php com um menu de navegação HTML. Em seguida, crie um index.php e um sobre.php e inclua o menu.php em ambos, usando include_once.

Módulo 8: Introdução ao CodeIgniter (Framework PHP)

Nos módulos anteriores, aprendemos os fundamentos do PHP. No entanto, desenvolver aplicações web complexas usando apenas PHP "puro" pode tornar o código desorganizado, difícil de manter e escalar. É aí que os **Frameworks PHP** se tornam indispensáveis.

8.1 O que é um Framework e por que usá-lo?

Um ***framework*** (arcabouço, em português) é um conjunto de ferramentas, bibliotecas e um esqueleto de código pré-definido que fornece uma base para o desenvolvimento de software. Ele padroniza a forma como as aplicações são construídas, promovendo boas práticas e acelerando o processo de desenvolvimento.

Por que usar um framework PHP?

- **Organização de código:** Impõe uma estrutura organizada (como MVC) que torna o código mais limpo e fácil de entender, especialmente em equipas.
- **Aceleração do desenvolvimento:** Muitas tarefas comuns (como ligação a bases de dados, validação de formulários, segurança) já vêm prontas, poupando tempo.
- **Segurança:** Frameworks incorporam funcionalidades de segurança que ajudam a proteger a sua aplicação contra vulnerabilidades comuns (ex: SQL Injection, XSS).
- **Manutenibilidade:** Códigos padronizados são mais fáceis de manter e depurar ao longo do tempo.
- **Comunidade:** A maioria dos frameworks tem uma comunidade ativa, o que significa mais recursos, suporte e atualizações.

Ilustração 8.1: Um Framework como alicerce para a sua aplicação.

Fonte: www.bigrentz.com

8.2 Visão Geral do CodeIgniter (MVC)

CodeIgniter é um popular *framework* de desenvolvimento rápido de aplicações web em PHP. É conhecido pela sua simplicidade, performance e documentação clara, sendo uma ótima escolha para iniciantes em frameworks.

O CodeIgniter segue o padrão de arquitetura **MVC (*Model-View-Controller*)**.

O Padrão MVC (*Model-View-Controller*)

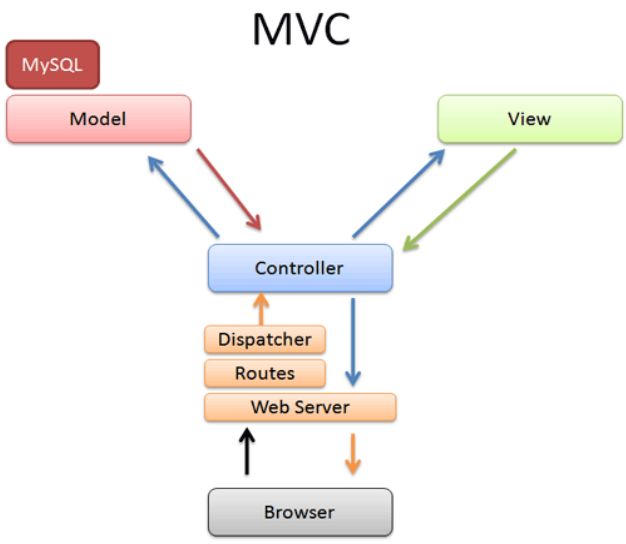
O MVC é um padrão de design de software que divide uma aplicação em três componentes principais, cada um com uma responsabilidade específica:

1. **Model (Modelo):**
    - Representa os dados e a lógica de negócio da aplicação.
    - Interage com a base de dados (buscar, guardar, atualizar, apagar dados).
    - Contém as regras de validação e a lógica para manipular os dados.
    - O Modelo *não* se comunica diretamente com o usuário; ele se comunica com o Controlador.

- **Exemplo:** Um modelo Usuario conteria métodos para guardar, obter e validar dados de usuários na base de dados.

2. **View (Visão):**
   - Responsável pela interface do usuário.
   - Exibe os dados ao usuário, geralmente em formato HTML.
   - Recebe os dados do Controlador e formata-os para apresentação.
   - A visão *não* contém lógica de negócio nem interage com a base de dados.
   - **Exemplo:** Um arquivo HTML/PHP que exibe uma lista de produtos ou um formulário de login.
3. **Controller (Controlador):**
   - Atua como o intermediário entre o modelo e a visão.
   - Recebe as requisições do usuário (via URL), processa-as, interage com o Modelo para obter/manipular dados e, em seguida, seleciona a visão apropriada para exibir o resultado.
   - É o cérebro da aplicação, coordenando o fluxo de dados e a interação entre as outras duas partes.
   - **Exemplo:** Um controlador Produtos que, ao receber uma requisição para /produtos/lista, pede ao Modelo_Produto todos os produtos e, em seguida, carrega a View_ListaProdutos para exibi-los.

Ilustração 8.2: O fluxo de dados no padrão MVC.



Fonte: stackoverflow.com

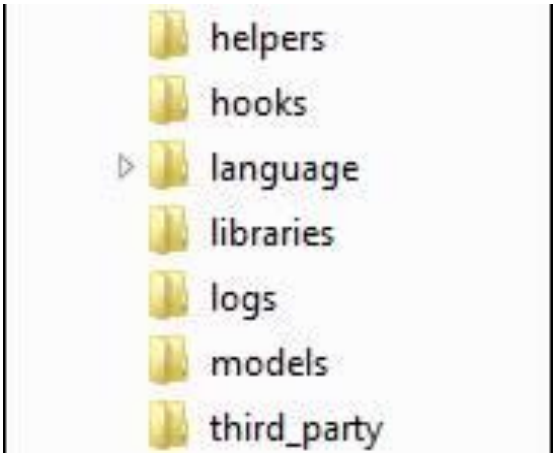
A vantagem do MVC é a **separação de responsabilidades**: cada componente tem uma única responsabilidade, o que torna o código mais modular, fácil de testar e de manter.

8.3 Instalação e configuração básica (Conceitual)

A instalação do CodeIgniter geralmente envolve os seguintes passos (estes são conceptuais para a apostila, pois a execução direta aqui é complexa):

1. **Download:** Descarregue a versão mais recente do CodeIgniter do site oficial.
2. **Extração:** Extraia os arquivos para a pasta raiz do seu servidor web (ex: htdocs no XAMPP/WAMP, ou a pasta do seu projeto local).
3. **Estrutura de pastas:** O CodeIgniter vem com uma estrutura de pastas bem definida (ex: app, public, system, writable).
    - app/Controllers: Onde os seus controladores PHP ficam.
    - app/Models: Onde os seus modelos PHP ficam.
    - app/Views: Onde os seus arquivos de vista (HTML com PHP) ficam.
    - public: A pasta acessível publicamente (onde o index.php principal reside).
4. **Configuração do ambiente:** Editar arquivos de configuração para definir as configurações da base de dados, URL base da aplicação, etc. (ex: app/Config/Database.php, app/Config/App.php).
5. **Servidor Web:** Certificar-se de que o seu servidor web (Apache ou Nginx) está configurado para apontar para a pasta public do CodeIgniter.

Ilustração 8.3: Estrutura de pastas de um projeto CodeIgniter.



Fonte: www.geeksforgeeks.org

8.4 Roteamento (*Routing*)

O **roteamento** é como o CodeIgniter mapeia as URLs digitadas pelo usuário para os controladores e métodos corretos na sua aplicação. É o sistema que decide "para onde" uma requisição deve ir.

No CodeIgniter, as rotas são definidas num arquivo de configuração (ex: app/Config/Routes.php). Você pode definir rotas simples, rotas com parâmetros, e até rotas que redirecionam para URLs diferentes.

Criando rotas no Codeigniter

```
<?php // Este é um exemplo simplificado de como as rotas podem ser definidas em CodeIgniter.
// Arquivo: app/Config/Routes.php (conceitual para a apostila)

use CodeIgniter\Router\RouteCollection; // Usado para tipagem (conceitual)

/**
 * @var RouteCollection $routes
 */

// Rota padrão: Quando o usuário acede à URL base (ex: http://localhost:8080)
// Ele será direcionado para o método 'index' do controlador 'Home'.
$routes->get('/', 'Home::index');

// Uma rota simples que mapeia a URL /sobre para o método 'sobre' do controlador 'Paginas'.
$routes->get('/sobre', 'Paginas::sobre');

// Uma rota com um parâmetro: /usuario/123 será mapeado para o método 'ver' do controlador Usuario,
// passando '123' como argumento.
$routes->get('/usuario/(:num)', Usuario::ver/$1');
```

```
// Uma rota para submissão de formulário (método POST)
$routes->post('/contacto/enviar', 'Contacto::enviar');

// Você pode agrupar rotas (útil para áreas de administração)
$routes->group('admin', function($routes){
   $routes->get('/', 'Admin\Dashboard::index');
   $routes->get('produtos', 'Admin\Produtos::listar');
});
```

Ilustração 8.4: O Roteamento na aplicação web.

Fonte: www.vecteezy.com

8.5 Controladores (*Controllers*)

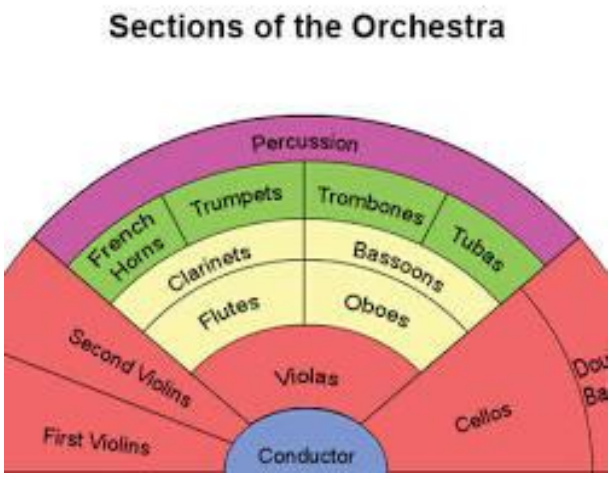
Os **controladores** são o coração do CodeIgniter no contexto do MVC. Eles são arquivos PHP que herdam de uma classe base do CodeIgniter e contêm métodos que são executados quando uma URL específica é acedida.

Responsabilidades de um Controlador:

- Receber a requisição HTTP.
- Interagir com o Modelo (se necessário) para obter ou processar dados.
- Passar os dados para a Vista.
- Carregar a Vista e enviá-la de volta ao navegador.

```php
<?php // Exemplo conceitual de um Controlador no CodeIgniter
// Arquivo: app/Controllers/Home.php (conceitual para a apostila)

namespace App\Controllers; // Define o namespace para o controlador

use App\Models\ProdutoModel; // Importa o modelo (veremos mais à frente)

class Home extends BaseController // Controladores herdam de BaseController ou Controller
{
  // Método padrão que é executado quando a rota '/' é acedida
  public function index()
  {
    // Carrega uma "vista" (o arquivo HTML/PHP que exibirá o conteúdo)
    // O CodeIgniter sabe procurar em app/Views/
    return view('welcome_message'); // Exemplo de vista padrão do CodeIgniter
  }

  // Outro método que pode ser acedido via URL (ex: /home/exemplo)
  public function exemplo()
  {
    $dados = [
      'titulo' => 'Página de Exemplo',
      'subtitulo' => 'Demonstração de um método no controlador.'
    ];

    // Carrega uma vista e passa os dados para ela
    return view('exemplo_view', $dados);
  }

  // Exemplo de interação com um Modelo (será mais detalhado no Módulo 9)
  public function produtos()
  {
    $produtoModel = new ProdutoModel(); // Instancia o modelo
    $listaProdutos = $produtoModel->getTodosProdutos(); // Obtém dados do modelo

    $dados = [
      'titulo' => 'Lista de Produtos',
      'produtos' => $listaProdutos
    ];

    return view('lista_produtos_view', $dados);
  }
}
```

Ilustração 8.5: O Controlador a orquestrar a aplicação.



Fonte: www.open.edu

8.6 Visão (*Views*)

As **visões** são a camada de apresentação da sua aplicação CodeIgniter. Elas contêm o HTML, CSS e JavaScript que será exibido ao usuário. As vistas também podem conter pequenas porções de código PHP para exibir dados que lhes são passados pelo Controlador.

Características das visões:

- São arquivos PHP, mas o seu objetivo principal é gerar HTML.
- Recebem dados do Controlador.
- *Não* devem conter lógica de negócio complexa ou interações diretas com a base de dados.

```
<?php // Exemplo conceitual de uma Vista no CodeIgniter
// Arquivo: app/Views/exemplo_view.php (conceitual para a apostila)
?>

<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title><?= esc($titulo ?? 'Título Padrão') ?></title> <!-- Exibe o título passado pelo controlador -
->
  <script src="https://cdn.tailwindcss.com"></script>
```

```
  <style>
    body { font-family: "Inter", sans-serif; }
  </style>
</head>
<body class="bg-gray-100 flex items-center justify-center min-h-screen">
  <div class="bg-white p-8 rounded-lg shadow-lg text-center max-w-md w-full">
    <h1 class="text-4xl font-bold text-indigo-700 mb-4">
      <?= esc($titulo ?? 'Bem-vindo') ?>
    </h1>
    <h2 class="text-xl text-gray-600 mb-6">
      <?= esc($subtitulo ?? 'Conteúdo gerado dinamicamente.') ?>
    </h2>

    <p class="text-gray-700 mb-4">
      Este é um exemplo de uma **Vista** no CodeIgniter. Ela recebe dados do
      **Controlador** e os exibe.
    </p>

    <?php if (isset($produtos) && !empty($produtos)): // Verifica se há produtos para exibir ?>
      <h3 class="text-2xl font-semibold text-green-700 mt-8 mb-4">Nossos Produtos:</h3>
      <ul class="list-disc list-inside text-left mx-auto max-w-xs text-gray-800">
        <?php foreach ($produtos as $produto): // Loop para exibir cada produto ?>
          <li><?= esc($produto->nome) ?> - R$ <?= esc(number_format($produto->preco, 2, ',',
'.')) ?></li>
        <?php endforeach; ?>
      </ul>
    <?php else: ?>
      <p class="text-gray-500 italic mt-8">Nenhum produto disponível no momento.</p>
    <?php endif; ?>

    <a href="/" class="mt-8 inline-block bg-blue-500 hover:bg-blue-600 text-white font-bold
py-2 px-6 rounded-lg transition duration-300 transform hover:scale-105">
      Voltar à Home
    </a>
  </div>
</body>
</html>
```

Ilustração 8.6: A Vista a exibir a interface do usuário.

Fonte: www.justinmind.com

8.7 Exemplo prático simplificado (conceitual)

Para ter uma ideia de como tudo se encaixa, vamos imaginar um fluxo simples no CodeIgniter:

1. **Requisição:** O usuário acede à URL http://seusite.com/artigos/lista.
2. **Roteador:** O arquivo app/Config/Routes.php vê esta URL e a mapeia para o método lista() do controlador Artigos.
3. **Controlador (Artigos.php):**
    - O método lista() é chamado.
    - Ele chama o ArtigoModel (um Modelo) para obter todos os artigos da base de dados.
    - Recebe os artigos do Modelo.
    - Carrega a vista lista_artigos_view.php e passa os dados dos artigos para ela.
4. **Visão (lista_artigos_view.php):**
    - Recebe os dados dos artigos.
    - Gera o HTML para exibir uma lista formatada de artigos.

5. **Resposta:** O HTML gerado pela vista é enviado de volta ao navegador do usuário.

Este é o ciclo básico do MVC em ação, proporcionando uma estrutura clara e organizada para a sua aplicação.

Exercícios:

1. Explique com as suas palavras a diferença entre um Modelo, uma Vista e um Controlador no padrão MVC.
2. Por que é vantajoso usar um framework como o CodeIgniter em vez de desenvolver tudo em PHP puro? Mencione pelo menos três razões.
3. Pesquise na documentação oficial do CodeIgniter como seria a estrutura de um método simples num Modelo para "obter todos os usuários" (não precisa de escrever código, apenas descrever a ideia).
4. Considere um formulário de contacto. Descreva o fluxo de uma submissão deste formulário através do MVC no CodeIgniter, indicando qual componente faria cada parte do trabalho.

Módulo 9: Banco de Dados e SQL

Para que as nossas aplicações Web sejam realmente dinâmicas e armazenem informações de forma persistente (ou seja, que não se percam quando fechamos o navegador ou o servidor), precisamos de utilizar **Bases de Dados**. Este módulo irá introduzi-lo aos conceitos de bases de dados e à linguagem SQL, que é usada para interagir com elas.

9.1 O que é um Banco de Dados Relacional?

Um **Banco de Dados** (ou Base de Dados) é uma coleção organizada de informações (dados) que são armazenadas e acedidas eletronicamente a partir de um sistema de computador.

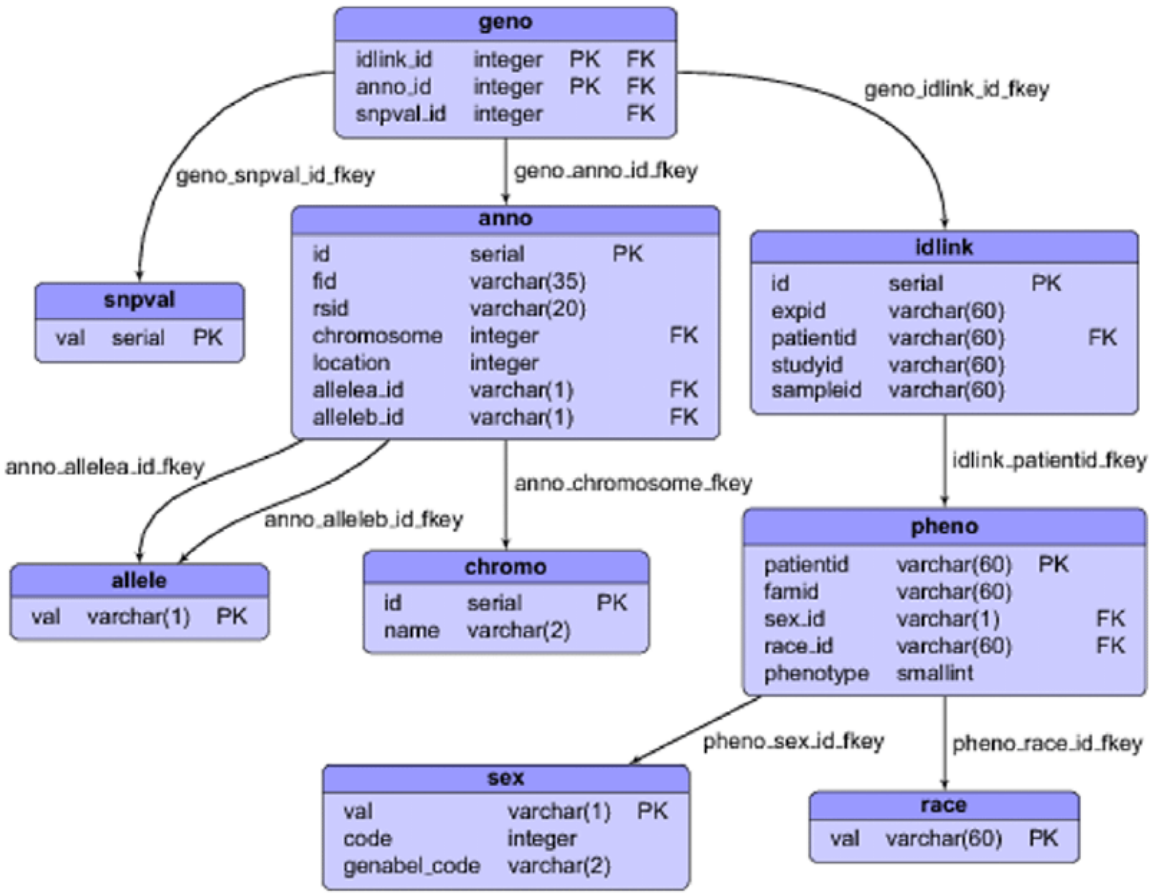
Existem vários tipos de bases de dados, mas o mais comum e amplamente utilizado no desenvolvimento web é o **Banco de Dados Relacional**.

Banco de Dados Relacional

Num banco de dados relacional, os dados são organizados em **tabelas** (também chamadas de relações), que consistem em linhas e colunas.

- **Tabelas:** Pense numa tabela como uma folha de cálculo. Cada tabela representa uma "entidade" ou um tipo de informação (ex: Usuários, Produtos, Pedidos).
- **Colunas (campos/atributos):** Cada coluna na tabela representa uma característica ou atributo dos dados (ex: nome, email, preco).
- **Linhas (registros/tuplas):** Cada linha na tabela representa um único item ou registro de dados para essa entidade (ex: um usuário específico, um produto específico).

Ilustração 9.1: Estrutura básica de uma tabela em banco de dados.

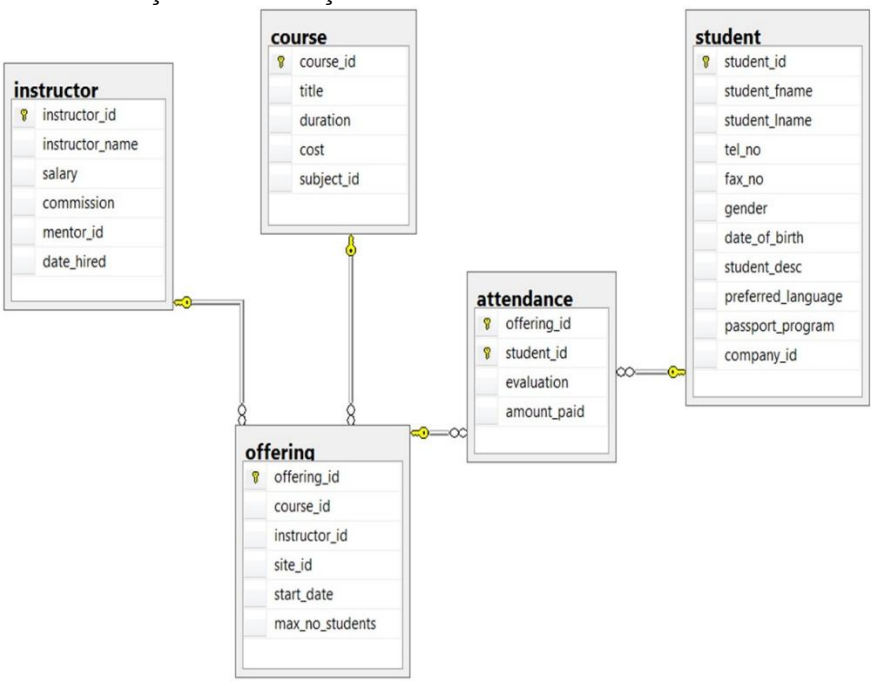


Fonte: www.researchgate.net

Relações

A força dos bancos de dados relacionais está na capacidade de estabelecer **relações** entre tabelas. Por exemplo, uma tabela Pedidos pode estar relacionada a uma tabela Usuários através de um ID de usuário, indicando qual usuário fez qual pedido.

Ilustração 9.2: Relação entre tabelas no banco de dados.



Fonte: opentextbc.ca

Existem vários sistemas de gestão de banco de dados (DBMS - *Database Management Systems*) relacionais populares:

- **MySQL:** Muito popular para aplicações web, frequentemente usado com PHP.
- **PostgreSQL:** Poderoso e robusto, com muitas funcionalidades avançadas.
- **SQLite:** Um banco de dados leve, que armazena os dados num único arquivo, ideal para aplicações pequenas ou testes.
- **SQL Server (Microsoft):** Usado em ambientes corporativos, especialmente com tecnologias Microsoft.
- **Oracle Database:** Um dos maiores e mais poderosos bancos de dados comerciais.

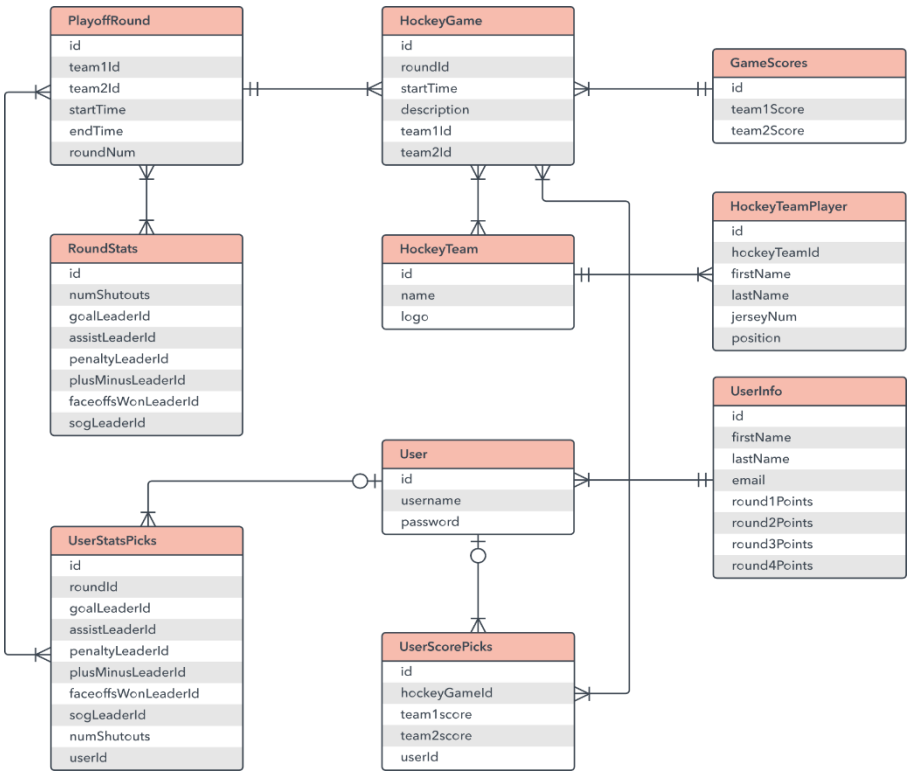
9.2 Modelagem de Dados

Antes de criar as tabelas no banco de dados, é crucial realizar a **modelagem de dados**. Este processo envolve projetar a estrutura do banco de dados para que ele armazene os dados de forma eficiente e sem redundâncias.

Conceitos chave na modelagem:

- **Entidade:** Representa um objeto ou conceito do mundo real sobre o qual queremos armazenar informações (ex: Pessoa, Produto, Departamento). Uma entidade geralmente se torna uma tabela.
- **Atributo:** Uma característica ou propriedade de uma entidade (ex: para Pessoa, atributos seriam nome, idade, endereco). Um atributo geralmente se torna uma coluna.
- **Chave primária (*Primary Key* - PK):** Uma ou mais colunas que identificam unicamente cada linha numa tabela. É como um "RG" para cada registro. Não pode haver valores duplicados e não pode ser nulo.
    - **Exemplo:** A coluna id_usuario na tabela usuarios.
- **Chave estrangeira (*Foreign Key* - FK):** Uma coluna (ou conjunto de colunas) numa tabela que se refere à Chave Primária de outra tabela. Ela estabelece a relação entre as tabelas.
    - **Exemplo:** A coluna id_usuario na tabela pedidos, que referencia o id_usuario da tabela usuarios.

Ilustração 9.3: Exemplo de Modelagem de Dados (ERD simplificado).



Fonte: www.lucidchart.com

9.3 Introdução ao SQL (*Structured Query Language*)

SQL (Structured Query Language) é a linguagem padrão usada para comunicar e manipular bancos de dados relacionais. É a "língua" que usamos para "falar" com o MySQL, PostgreSQL, etc.

Com o SQL, podemos:

- **Criar** bases de dados e tabelas.
- **Inserir** novos dados.
- **Consultar** (ler) dados existentes.
- **Atualizar** dados.
- **Apagar** dados.

Existem diferentes categorias de comandos SQL:

- **DDL (Data Definition Language):** Para definir a estrutura do banco de dados (ex: CREATE, ALTER, DROP).
- **DML (Data Manipulation Language):** Para manipular os dados dentro das tabelas (ex: SELECT, INSERT, UPDATE, DELETE).
- **DCL (Data Control Language):** Para controlar o acesso e permissões (ex: GRANT, REVOKE).
- **TCL (Transaction Control Language):** Para gerenciar transações (ex: COMMIT, ROLLBACK).

Vamos focar nos comandos DDL e DML mais comuns.

9.4 Comandos SQL essenciais (DML e DDL)

Os comandos essenciais SQL se dividem em DML (*Data Manipulation Language* – Linguagem de Manipulação de Dados) e DDL (*Data Definition Language* – Linguagem de Definição de Dados)

9.4.1 Criar uma Tabela (CREATE TABLE)

Este comando é usado para definir a estrutura de uma nova tabela no banco de dados.

SQL

```
-- Exemplo de CREATE TABLE para uma tabela de 'usuarios'
CREATE TABLE usuarios (
```

```
  id_usuario INT PRIMARY KEY AUTO_INCREMENT, -- Chave primária auto-incrementável
  nome VARCHAR(100) NOT NULL,            -- Nome do usuário (máximo 100 caracteres, não pode ser nulo)
  email VARCHAR(100) UNIQUE NOT NULL,    -- Email do usuário (único e não nulo)
  senha VARCHAR(255) NOT NULL,           -- Senha (geralmente armazenada como hash)
  data_cadastro DATETIME DEFAULT CURRENT_TIMESTAMP -- Data e hora de cadastro (padrão é a hora atual)
);

-- Exemplo de CREATE TABLE para uma tabela de 'produtos'
CREATE TABLE produtos (
  id_produto INT PRIMARY KEY AUTO_INCREMENT,
  nome_produto VARCHAR(200) NOT NULL,
  descricao TEXT,                        -- Campo de texto longo
  preco DECIMAL(10, 2) NOT NULL,         -- Preço (10 dígitos no total, 2 após a vírgula)
  estoque INT DEFAULT 0
);

-- Exemplo de CREATE TABLE com chave estrangeira para 'pedidos'
CREATE TABLE pedidos (
  id_pedido INT PRIMARY KEY AUTO_INCREMENT,
  id_usuario INT NOT NULL,               -- Chave estrangeira para a tabela 'usuarios'
  data_pedido DATETIME DEFAULT CURRENT_TIMESTAMP,
  valor_total DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario) -- Define a chave estrangeira
);
```

9.4.2 Inserir dados (INSERT INTO)

Para adicionar novas linhas (registros) a uma tabela:

SQL

```
-- Inserir um novo usuário
INSERT INTO usuarios (nome, email, senha)
VALUES ('Ana Silva', 'ana.silva@email.com', 'hash_da_senha_ana');

-- Inserir outro usuário, especificando todas as colunas
INSERT INTO usuarios (id_usuario, nome, email, senha, data_cadastro)
VALUES (NULL, 'Bruno Costa', 'bruno.costa@email.com', 'hash_da_senha_bruno', NOW()); -- NOW() para data/hora atual

-- Inserir um produto
INSERT INTO produtos (nome_produto, descricao, preco, estoque)
VALUES ('Teclado Mecânico', 'Teclado gamer com switches azuis.', 350.00, 100);
```

```
-- Inserir um pedido para o usuário com id_usuario = 1
INSERT INTO pedidos (id_usuario, valor_total)
VALUES (1, 350.00);
```

9.4.3 Consultar dados (SELECT)

O comando SELECT é o mais usado e poderoso, permitindo-lhe buscar dados das tabelas.

SQL

```
-- Selecionar todas as colunas de todos os usuários
SELECT * FROM usuarios;

-- Selecionar apenas o nome e email dos usuários
SELECT nome, email FROM usuarios;

-- Selecionar produtos com preço superior a 100
SELECT nome_produto, preco FROM produtos WHERE preco > 100.00;

-- Selecionar usuários cujo nome começa com 'A'
SELECT nome, email FROM usuarios WHERE nome LIKE 'A%';

-- Selecionar e ordenar usuários por nome (ascendente)
SELECT * FROM usuarios ORDER BY nome ASC;

-- Selecionar os 5 produtos mais caros
SELECT nome_produto, preco FROM produtos ORDER BY preco DESC LIMIT 5;

-- Contar o número total de usuários
SELECT COUNT(*) AS total_usuarios FROM usuarios;

-- Calcular o preço médio dos produtos
SELECT AVG(preco) AS preco_medio FROM produtos;
```

9.4.4 Atualizar dados (UPDATE)

Para modificar dados existentes numa ou mais linhas de uma tabela:

SQL

```
-- Atualizar o email de um usuário específico (ID 1)
UPDATE usuarios
SET email = 'ana.silva_novo@email.com'
WHERE id_usuario = 1;

-- Atualizar o estoque de um produto
```

```
UPDATE produtos
SET estoque = estoque - 1 -- Diminui o estoque em 1
WHERE id_produto = 10;
```

Cuidado: Sempre use a cláusula WHERE no UPDATE para especificar quais linhas devem ser atualizadas. Sem WHERE, todas as linhas da tabela seriam atualizadas!

9.4.5 Apagar dados (DELETE FROM)

Para remover linhas de uma tabela:

SQL

```
-- Apagar o usuário com ID 2
DELETE FROM usuarios
WHERE id_usuario = 2;

-- Apagar todos os produtos com estoque zero
DELETE FROM produtos
WHERE estoque = 0;
```

Cuidado: Assim como no UPDATE, a cláusula WHERE é crucial no DELETE. Sem ela, todas as linhas da tabela seriam apagadas!

Exercícios:

1. Desenhe um pequeno modelo de dados (entidades, atributos, chaves primárias e estrangeiras) para um sistema de biblioteca que armazene informações sobre Livros, Autores e Empréstimos.
2. Escreva o comando SQL CREATE TABLE para cada uma das tabelas que você desenhou no exercício 1. Certifique-se de incluir chaves primárias e estrangeiras.
3. Escreva comandos SQL INSERT INTO para adicionar três livros e dois autores nas suas respetivas tabelas.
4. Escreva um comando SQL SELECT para:
    - Selecionar todos os livros.
    - Selecionar o título e o ano de publicação dos livros publicados após 2000.
    - Contar quantos livros existem na tabela.

5. Escreva um comando SQL UPDATE para mudar o ano de publicação de um livro específico.
6. Escreva um comando SQL DELETE para apagar um livro que tenha um título específico.

Módulo 10: Integração do PHP e CodeIgniter com Banco de Dados

Uma aplicação web dinâmica só se torna verdadeiramente útil quando consegue armazenar e recuperar dados de forma persistente. É aqui que a integração do PHP (especialmente com um framework como o CodeIgniter) com um banco de dados se torna fundamental. Neste módulo, vamos explorar como conectar a sua aplicação CodeIgniter a um banco de dados e realizar operações essenciais.

10.1 Configuração do Banco de Dados no CodeIgniter

Antes de interagir com o banco de dados, o CodeIgniter precisa saber como se conectar a ele. As informações de conexão são definidas num arquivo de configuração.

No CodeIgniter, a configuração do banco de dados é feita no arquivo app/Config/Database.php. Você precisará preencher os detalhes de conexão, como tipo de banco de dados, hostname, nome de usuário, senha e nome da base de dados.

Exemplo de configuração (app/Config/Database.php - Conceitual):

```
<?php
// app/Config/Database.php (Exemplo simplificado para a apostila)

namespace Config;

use CodeIgniter\Database\Config; // Usado para tipagem (conceitual)

/**
 * Database Configuration
 */
class Database extends Config
{
    public $default = [
        'DSN'      => '',
        'hostname' => 'localhost', // Endereço do seu servidor de banco de dados
        'username' => 'root',      // Nome de usuário do banco de dados (ex: 'root' para XAMPP/WAMP)
        'password' => '',          // Senha do banco de dados (ex: vazio para 'root' no XAMPP/WAMP)
        'database' => 'minha_aplicacao_db', // Nome da base de dados que você criou
        'DBDriver' => 'MySQLi',    // Driver do banco de dados (ex: MySQLi para MySQL)
        'DBPrefix' => '',          // Prefixo para nomes de tabelas (opcional)
```

```
    'pConnect' => false,
    'DBDebug'  => (ENVIRONMENT !== 'production'), // Debug ativado em desenvolvimento
    'charset'  => 'utf8',
    'DBCollat' => 'utf8_general_ci',
    'swapPre'  => '',
    'encrypt'  => false,
    'compress' => false,
    'strictOn' => false,
    'failover' => [ ],
    'port'     => 3306, // Porta padrão do MySQL
  ];
}
```

Importante: Nunca use credenciais de banco de dados em produção sem segurança adequada. Use variáveis de ambiente ou ferramentas de gestão de segredos. Para ambientes de desenvolvimento local, root sem senha é comum.

Ilustração 10.1: Configuração da conexão ao banco de dados.

```php
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
// ...
?>
```

Fonte: https://www.researchgate.net

10.2 Modelos (*Models*) no CodeIgniter

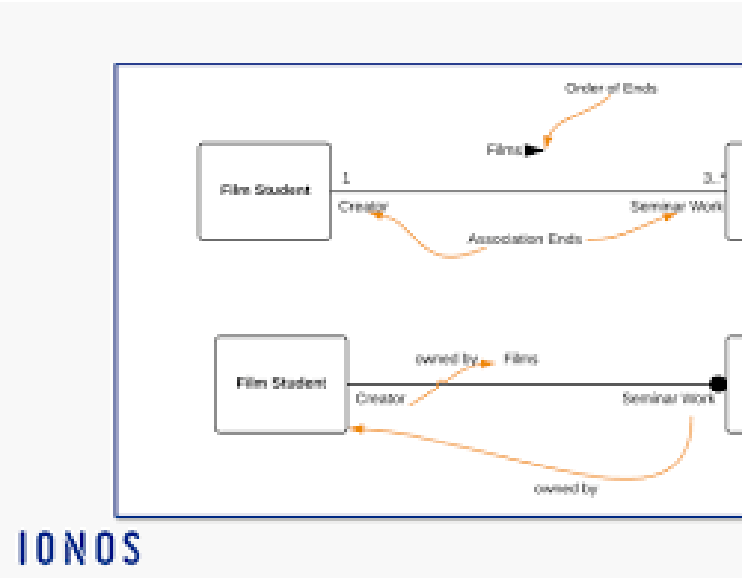
No padrão MVC, a camada **Model** é a responsável por toda a interação com o banco de dados e a lógica de negócio relacionada aos dados. No CodeIgniter, você cria **Classes Modelo** (ou simplesmente "Modelos") que herdam da classe CodeIgniter\Model.

Esses modelos fornecem uma forma abstrata e conveniente de interagir com as tabelas do seu banco de dados, sem precisar escrever SQL puro a todo momento (embora você ainda possa fazer isso quando necessário).

Responsabilidades de um Modelo:

- Definir a tabela a que o modelo está associado.
- Definir a chave primária da tabela.
- Lidar com operações CRUD (Criar, Ler, Atualizar, Apagar) no banco de dados.
- Implementar a lógica de negócio relacionada aos dados (ex: validar dados antes de salvar).

Ilustração 10.2: O papel do Modelo na interação com o banco de dados.



Fonte: www.ionos.com

10.3 Operações CRUD com CodeIgniter Models

Vamos ver como as operações CRUD (Create, Read, Update, Delete) são realizadas usando os Modelos do CodeIgniter.

10.3.1 Criar (Create) / Inserir Dados (insert())

Para adicionar novos registos a uma tabela:

PHP

```
<?php // Exemplo conceitual de um Modelo em CodeIgniter para Usuários
// Arquivo: app/Models/UsuarioModel.php

namespace App\Models;

use CodeIgniter\Model;

class UsuarioModel extends Model
```

```
{
    protected $table = 'usuarios'; // Nome da tabela no banco de dados
    protected $primaryKey = 'id_usuario'; // Chave primária da tabela

    protected $useAutoIncrement = true; // A chave primária é auto-incrementável

    // Campos que podem ser preenchidos em operações de insert/update
    protected $allowedFields = ['nome', 'email', 'senha', 'data_cadastro'];

    // Método para inserir um novo usuário
    public function criarUsuario($dados)
    {
        // $dados seria um array associativo como ['nome' => 'Novo Usuário', 'email' =>
'novo@email.com', 'senha' => 'hash_da_senha']
        return $this->insert($dados); // Retorna o ID do novo registro ou false em caso de falha
    }
}

// Como um Controlador chamaria:
/*
namespace App\Controllers;

use App\Models\UsuarioModel; // Importa o Modelo

class Usuarios extends BaseController
{
    public function cadastrar()
    {
        $usuarioModel = new UsuarioModel();

        $dadosNovoUsuario = [
            'nome' => 'Fernando Lima',
            'email' => 'fernando.lima@exemplo.com',
            'senha' => password_hash('minhasenha123', PASSWORD_DEFAULT), // Sempre armazene
senhas com hash!
            'data_cadastro' => date('Y-m-d H:i:s')
        ];

        if ($usuarioModel->criarUsuario($dadosNovoUsuario)) {
            echo "Usuário cadastrado com sucesso!";
        } else {
            echo "Falha ao cadastrar usuário.";
        }
    }
}
*/
```

10.3.2 Ler (Read) / consultar dados (find(), findAll(), where())

Para buscar dados de uma ou mais linhas:

PHP

```php
<?php
// No seu UsuarioModel.php (continuação)

class UsuarioModel extends Model
{
  // ... propriedades já definidas ...

  // Método para obter um usuário pelo ID
  public function getUsuario($id)
  {
    return $this->find($id); // Retorna um array com os dados do usuário ou null
  }

  // Método para obter todos os usuários
  public function getTodosUsuarios()
  {
    return $this->findAll(); // Retorna um array de arrays (ou objetos) com todos os usuários
  }

  // Método para obter usuários por email (exemplo de condição WHERE)
  public function getUsuarioPorEmail($email)
  {
    // where('coluna', 'valor')
    return $this->where('email', $email)->first(); // first() retorna o primeiro resultado que
corresponde
  }

  // Exemplo de consulta mais complexa com construtor de query
  public function getUsuariosAtivos()
  {
    return $this->where('status', 'ativo')
          ->orderBy('nome', 'ASC')
          ->findAll();
  }
}

// Como um Controlador chamaria:
/*
namespace App\Controllers;

use App\Models\UsuarioModel;

class Usuarios extends BaseController
{
  public function listar()
  {
    $usuarioModel = new UsuarioModel();
    $usuarios = $usuarioModel->getTodosUsuarios();

    // Passa os dados para uma Vista para serem exibidos
```

```
    // return view('lista_usuarios_view', ['usuarios' => $usuarios]);
  }

  public function ver($id)
  {
    $usuarioModel = new UsuarioModel();
    $usuario = $usuarioModel->getUsuario($id);

    if ($usuario) {
      // return view('detalhes_usuario_view', ['usuario' => $usuario]);
    } else {
      // return view('usuario_nao_encontrado');
    }
  }
}
*/
```

10.3.3 Atualizar (Update) dados (update())

Para modificar registos existentes:

PHP

```
<?php
// No seu UsuarioModel.php (continuação)

class UsuarioModel extends Model
{
  // ... propriedades e métodos já definidos ...

  // Método para atualizar informações de um usuário
  public function atualizarUsuario($id, $dados)
  {
    // $dados seria um array associativo como ['nome' => 'Ana Nova', 'email' =>
'ana.nova@email.com']
    return $this->update($id, $dados); // Retorna true em sucesso, false em falha
  }
}

// Como um Controlador chamaria:
/*
namespace App\Controllers;

use App\Models\UsuarioModel;

class Usuarios extends BaseController
{
  public function editar($id)
  {
    $usuarioModel = new UsuarioModel();

    // Dados recebidos de um formulário de edição
```

```
    $dadosAtualizacao = $this->request->getPost(); // Obtém dados do POST

    if ($usuarioModel->atualizarUsuario($id, $dadosAtualizacao)) {
      echo "Usuário atualizado com sucesso!";
    } else {
      echo "Falha ao atualizar usuário.";
    }
  }
}
*/
```

10.3.4 Apagar (Delete) dados (delete())

Para remover registos de uma tabela:

PHP

```
<?php
// No seu UsuarioModel.php (continuação)

class UsuarioModel extends Model
{
  // ... propriedades e métodos já definidos ...

  // Método para apagar um usuário pelo ID
  public function apagarUsuario($id)
  {
    return $this->delete($id); // Retorna true em sucesso, false em falha
  }
}

// Como um Controlador chamaria:
/*
namespace App\Controllers;

use App\Models\UsuarioModel;

class Usuarios extends BaseController
{
  public function remover($id)
  {
    $usuarioModel = new UsuarioModel();

    if ($usuarioModel->apagarUsuario($id)) {
      echo "Usuário apagado com sucesso!";
    } else {
      echo "Falha ao apagar usuário.";
    }
  }
}
*/
```

10.4 Boas práticas e considerações

- **Validação de dados:** Sempre valide os dados de entrada (input) do usuário no seu Modelo ou Controlador antes de os inserir ou atualizar no banco de dados para evitar erros e vulnerabilidades de segurança. O CodeIgniter possui ferramentas de validação integradas.
- **Segurança de senhas:** Nunca armazene senhas em texto puro. Use funções de hash seguras (ex: password_hash() no PHP) antes de guardar e password_verify() para verificar.
- **Tratamento de erros:** Implemente um tratamento de erros adequado para lidar com falhas na conexão ao banco de dados ou em operações SQL.
- ***Queries* personalizadas:** Embora os modelos do CodeIgniter sejam ótimos, para queries muito complexas ou específicas, você pode usar o Query Builder do CodeIgniter ou escrever SQL puro diretamente nos seus modelos.

Este módulo forneceu uma visão geral de como o CodeIgniter Model interage com o banco de dados para realizar operações CRUD. A prática é essencial para dominar estes conceitos.

Exercícios:

1. Imagine que você está a criar um sistema de gestão de tarefas. Modele as tabelas necessárias (tarefas, usuários) e identifique as chaves primárias e estrangeiras.
2. Escreva o código conceitual de um TarefaModel no CodeIgniter que permita:
    - Criar uma nova tarefa.
    - Obter todas as tarefas de um usuário específico.
    - Atualizar o status de uma tarefa para "concluída".
    - Apagar uma tarefa.
3. Descreva como um Controlador_Tarefas chamaria os métodos do TarefaModel para exibir uma lista de tarefas, criar uma nova, e apagar uma existente.

Módulo 11: Segurança básica na Web

Construir aplicações Web funcionais é apenas metade do trabalho; a outra metade, e igualmente importante, é garantir que elas sejam **seguras**. A segurança na web é um campo vasto, mas neste módulo, vamos cobrir as vulnerabilidades mais comuns e as melhores práticas para proteger as suas aplicações.

11.1 Por que a segurança na Web é importante?

A segurança na web refere-se às medidas e proteções tomadas para salvaguardar os dados e a integridade das aplicações web contra ataques maliciosos. Num mundo cada vez mais digital, as violações de segurança podem ter consequências graves:

- **Perda de dados sensíveis:** Informações pessoais de usuários (nomes, e-mails, senhas, dados bancários).
- **Danos à reputação:** Uma empresa ou organização que sofre um ataque pode perder a confiança dos seus clientes.
- **Perdas financeiras:** Roubo de dados financeiros, fraude, ou interrupção de serviços que causam prejuízo.
- **Ataques de *defacement*:** Alteração visual de um website para fins maliciosos.
- **Conformidade legal:** Muitas regulamentações (como o LGPD – Lei Geral de Proteção de Dados Pessoais) exigem que as empresas protejam os dados dos usuaários.

Ilustração 11.1: A segurança como um escudo para a sua aplicação web.

Fonte: www.shutterstock.com

11.2 Ataques comuns e prevenção

Vamos explorar algumas das vulnerabilidades mais frequentes e como as suas aplicações podem ser protegidas.

10.2.1 Injeção de SQL (*SQL Injection*)

A **Injeção de SQL** é um ataque onde um atacante insere (injeta) código SQL malicioso através de campos de entrada de usuário, como formulários ou parâmetros de URL. Se a aplicação não "limpar" ou "tratar" essa entrada corretamente antes de usá-la numa consulta SQL, o atacante pode executar comandos arbitrários no banco de dados.

Como funciona (exemplo simples):

Imagine um campo de login onde você pede o nome de usuário. Se um atacante digitar: ' OR '1'='1

Se o código PHP construir a query assim (VULNERÁVEL):

```
<?php
// CÓDIGO VULNERÁVEL A INJEÇÃO DE SQL
$username = $_POST['username'];
$password = $_POST['password']; // Senha também seria vulnerável

$sql = "SELECT * FROM usuarios WHERE username = '$username' AND password = '$password'";
// Se $username for "' OR '1'='1", a query se torna:
```

```
// SELECT * FROM usuarios WHERE username = '' OR '1'='1' AND password = '...'
// A condição '1'='1' é sempre verdadeira, ignorando a senha e logando o atacante.
?>
```

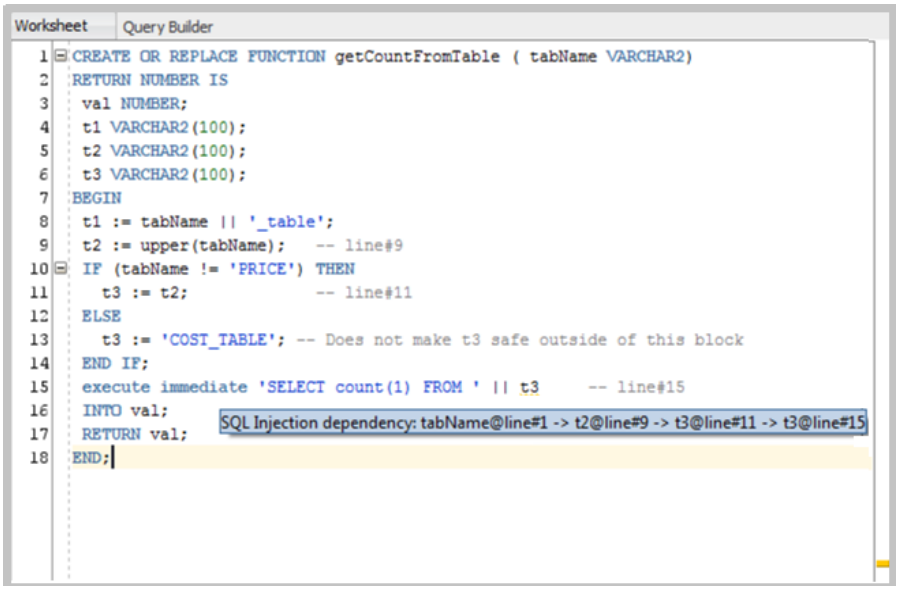
Prevenção:

A melhor forma de prevenir SQL *Injection* é usar ***Prepared Statements*** **(Declarações Preparadas)** com **Parâmetros Vinculados**. Quase todos os drivers de banco de dados e frameworks (como o CodeIgniter) suportam isso.

- As *prepared statements* separam a definição da consulta dos valores dos dados. O banco de dados "prepara" a estrutura da consulta e, em seguida, os valores são "vinculados" a essa consulta. Isso impede que a entrada do usuário seja interpretada como parte do código SQL.

```
<?php
// CÓDIGO SEGURO: USANDO PREPARED STATEMENTS (Conceitual para PHP puro com PDO)
// No CodeIgniter, os Models já fazem isso por você na maioria das vezes.

$stmt = $pdo->prepare("SELECT * FROM usuarios WHERE username = :username AND
password = :password");
$stmt->bindParam(':username', $username);
$stmt->bindParam(':password', $password);
$stmt->execute();
$user = $stmt->fetch();
?>
```

Ilustração 11.2: Prevenção de Injeção de SQL com Prepared Statements.



Fonte: docs.oracle.com

11.2.2 *Cross-Site Scripting* (XSS)

O ***Cross-Site Scripting*** **(XSS)** ocorre quando um atacante injeta scripts maliciosos (geralmente JavaScript) em páginas web visualizadas por outros usuários. O navegador da vítima executa esse script, que pode roubar cookies, credenciais de sessão, ou defacement do site.

Como funciona (exemplo simples):

Imagine um campo de comentário onde o usuário pode postar texto. Se um atacante postar: Olá a todos! <script>alert('Você foi atacado! ' + document.cookie);</script>

Se a aplicação exibir esse comentário diretamente no HTML (VULNERÁVEL):

```
<!-- CÓDIGO VULNERÁVEL A XSS -->
<p>
  Olá a todos! <script>alert('Você foi atacado! ' + document.cookie);</script>
</p>
```

Quando outro usuário visita a página, o script é executado no seu navegador, exibindo um alerta com os seus cookies.

Prevenção:

Sempre **escape** ou **sanitize** (limpe) a entrada do usuário antes de exibi-la na página HTML. Isso significa converter caracteres especiais (como <, >, ', ", &) em suas entidades HTML correspondentes (<, >, ', ", &), para que o navegador os interprete como texto e não como código.

- No PHP, use htmlspecialchars() ou htmlentities().
- No CodeIgniter, a função esc() nas Views faz isso automaticamente.

```
<?php
// CÓDIGO SEGURO: ESCAPANDO A SAÍDA PARA PREVENIR XSS
$comentario = $_POST['comentario']; // Assume que vem de um formulário
echo "<p>" . htmlspecialchars($comentario, ENT_QUOTES, 'UTF-8') . "</p>";

// Em Views do CodeIgniter:
// <?= esc($variavel_com_dados_do_usuario) ?>
?>
```

Ilustração 11.3: Prevenção de XSS com escaping de saída.

Fonte: tryhackme.com

11.2.3 Senhas seguras

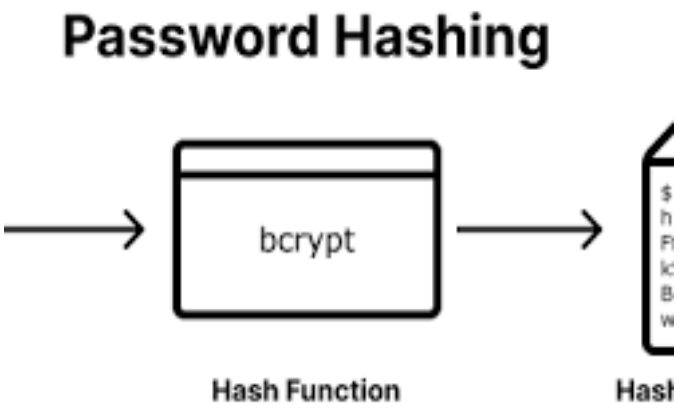
O armazenamento e a gestão de senhas são cruciais para a segurança dos usuários.

Boas práticas:

- **Nunca armazene senhas em texto puro.** Se o seu banco de dados for comprometido, todas as senhas dos usuários estarão expostas.
- **Use funções de hash seguras:** Funções de hash (como password_hash() do PHP) transformam a senha numa string de caracteres de comprimento fixo (o hash) que é unidirecional (não pode ser revertida para a senha original).
    - Sempre use um **salt** (um dado aleatório adicionado à senha antes do hashing) para prevenir ataques de "rainbow table". O password_hash() faz isso automaticamente.
- **Verificação:** Para verificar uma senha, você recalcula o hash da senha fornecida pelo usuário e compara-o com o hash armazenado. Use password_verify() no PHP.

```
<?php
// Armazenar uma senha (durante o registo)
$senhaPura = "minhaSenhaSecreta123";
$senhaHash = password_hash($senhaPura, PASSWORD_DEFAULT); // PASSWORD_DEFAULT
usa o algoritmo mais forte disponível (atualmente bcrypt)
echo "Hash da senha: " . $senhaHash . "<br>";

// Verificar uma senha (durante o login)
$senhaDigitada = "minhaSenhaSecreta123";
if (password_verify($senhaDigitada, $senhaHash)) {
   echo "Senha correta! Login bem-sucedido.<br>";
} else {
   echo "Senha incorreta.<br>";
}

// Tentativa com senha errada
$senhaErrada = "senhaErrada";
if (password_verify($senhaErrada, $senhaHash)) {
   echo "Login bem-sucedido.<br>";
} else {
   echo "Senha incorreta (tentativa errada).<br>";
}
?>
```

Ilustração 11.4: Armazenamento seguro de senhas com hashing.



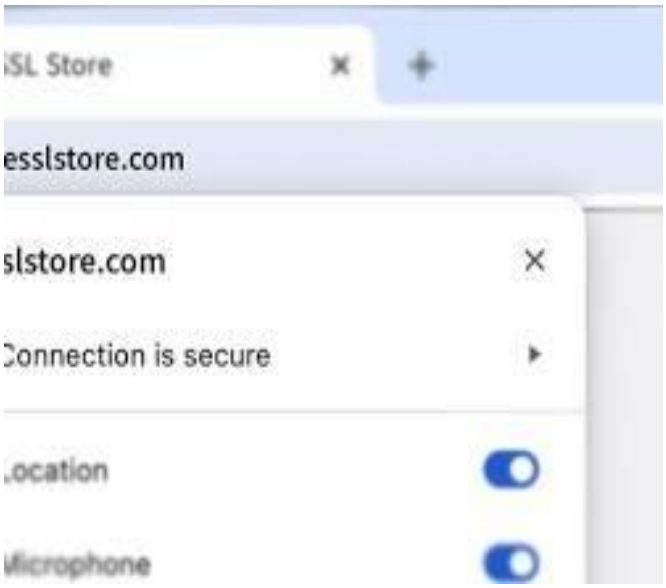
Fonte: www.authgear.com

11.3 Outras considerações de segurança

- **Validação de entrada:** Além de escapar a saída, sempre valide os dados de entrada no lado do servidor. Verifique tipos de dados, comprimentos, formatos (ex: e-mail válido), e se os valores estão dentro dos limites esperados. Isso ajuda a prevenir muitos tipos de ataques.
- **Autenticação e autorização:**
    - **Autenticação:** Verificar a identidade do usuário (login).
    - **Autorização:** Determinar o que um usuário autenticado tem permissão para fazer (ex: um administrador pode apagar usuários, um usuário normal não).
- **Gestão de sessões:** Use sessões seguras. Regenerar IDs de sessão após o login para evitar ataques de fixação de sessão. Armazene dados mínimos na sessão.
- **HTTPS (SSL/TLS):** Sempre use HTTPS para encriptar a comunicação entre o navegador do usuário e o servidor. Isso protege os dados em trânsito contra intercetação.
- **Controle de acesso:** Implemente controlo de acesso adequado para recursos e funcionalidades, garantindo que apenas usuários autorizados possam acedê-los.

- **Atualização de software:** Mantenha o seu sistema operativo, servidor web (Apache/Nginx), PHP, CodeIgniter e todas as bibliotecas atualizadas para as versões mais recentes, pois as atualizações geralmente incluem correções de segurança.
- **Relatórios de erros:** Em ambiente de produção, desative a exibição de erros detalhados. Erros detalhados podem vazar informações sensíveis sobre a sua aplicação. Registre os erros em arquivos de log.

Ilustração 11.5: O uso de HTTPS para comunicação segura.



Fonte: www.thesslstore.com

Exercícios:

1. Um colega de turma está a criar um formulário de login e planeia armazenar as senhas dos usuários diretamente no banco de dados. Explique-lhe por que esta é uma má ideia e qual a melhor alternativa.
2. Você está a criar um livro de visitas onde os usuários podem deixar comentários. Que tipo de ataque pode ocorrer se você exibir os comentários diretamente no HTML sem nenhum tratamento? Como você pode prevenir isso em PHP?

3. Pesquise sobre "CSRF (*Cross-Site Request Forgery*)" e, em poucas palavras, explique o que é e por que é uma ameaça. (Não precisa de código de prevenção, apenas o conceito).
4. Por que é importante manter o PHP e o CodeIgniter sempre atualizados para as versões mais recentes?

Módulo 12: Controle de versão com Git

Ao desenvolver uma aplicação Web, você estará constantemente a fazer alterações no código: adicionar novas funcionalidades, corrigir erros, refatorar partes do sistema. Sem uma forma de gerir essas alterações, o processo pode tornar-se caótico, especialmente ao trabalhar em equipa. É aqui que os **Sistemas de Controle de Versão (SCV)** entram em jogo, e o **Git** é o mais popular deles.

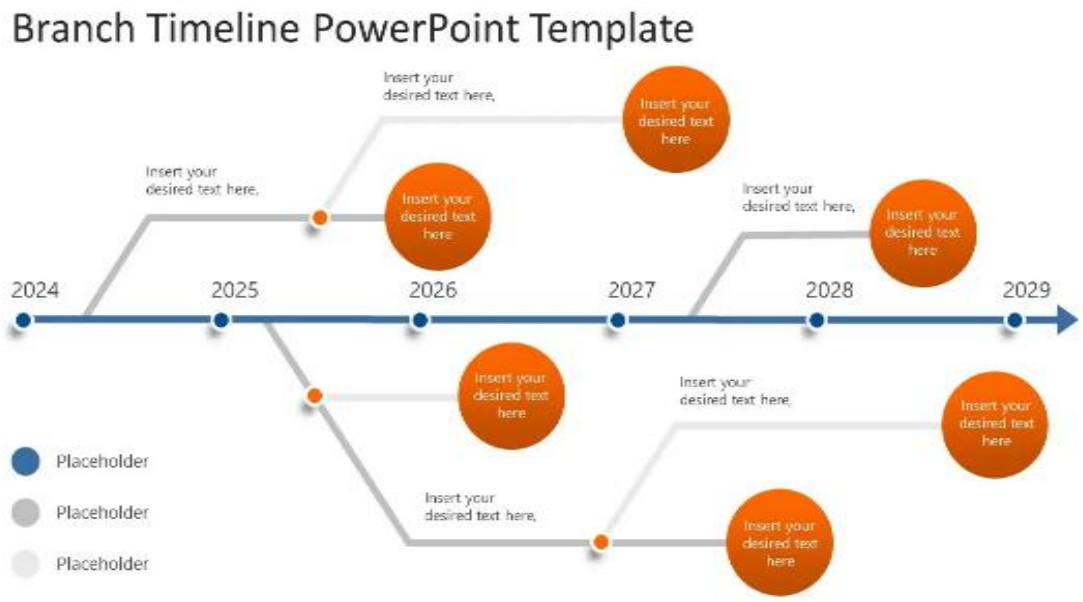
12.1 O que é Controle de versão e por que usá-lo?

Controle de versão é um sistema que regista as alterações feitas num arquivo ou conjunto de arquivos ao longo do tempo, para que você possa recuperar versões específicas mais tarde. Pense nele como uma "máquina do tempo" para o seu código.

Por que usar um Sistema de Controle de Versão (VCS)?

- **Histórico completo:** Cada alteração é registada, permitindo que você veja quem fez o quê, quando e porquê.
- **Colaboração eficaz:** Múltiplas pessoas podem trabalhar no mesmo projeto simultaneamente sem sobrescrever o trabalho umas das outras.
- **Recuperação de erros:** Se algo der errado (um bug introduzido, código acidentalmente apagado), você pode facilmente reverter para uma versão anterior funcional.
- **Experimentação segura:** Você pode criar "ramificações" (*branches*) do seu código para experimentar novas funcionalidades sem afetar a versão principal.
- **Organização:** Mantém o seu projeto organizado e rastreável.

Ilustração 12.1: O Controle de Versão como uma Linha do Tempo do Projeto.



Fonte: slidemodel.com

12.2 Conceitos básicos do Git

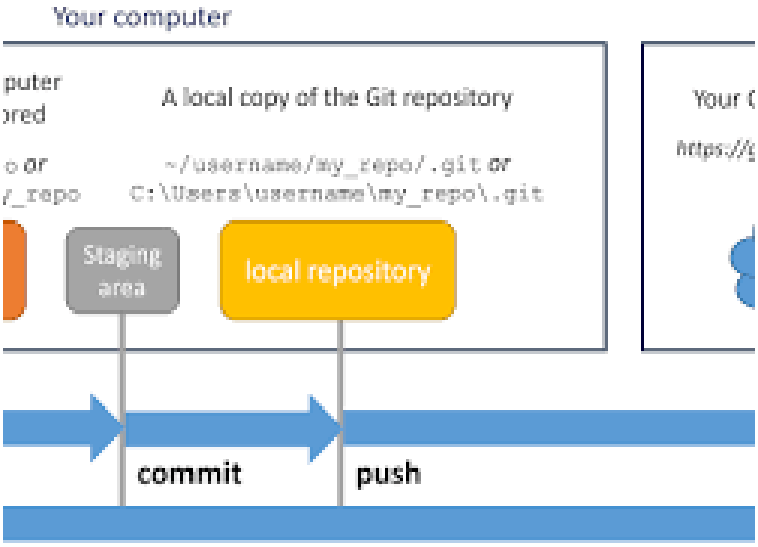
O Git é um **Sistema de Controle de Versão Distribuído (DVCS)**. Isso significa que cada desenvolvedor tem uma cópia completa do histórico do repositório no seu computador local, o que permite trabalhar offline e torna o sistema mais robusto.

12.2.1 Repositório (Repository)

Um **Repositório Git** (ou "repo") é onde o Git armazena todas as suas informações de controle de versão para um projeto específico. Ele contém todos os arquivos do projeto e todo o histórico de alterações.

- **Repositório Local:** A cópia do repositório no seu computador.
- **Repositório Remoto:** A cópia do repositório hospedada num servidor (ex: GitHub, GitLab, Bitbucket), usada para colaboração e backup.

Ilustração 12.2: Repositório Local vs. Remoto.



Fonte: hellorob.org

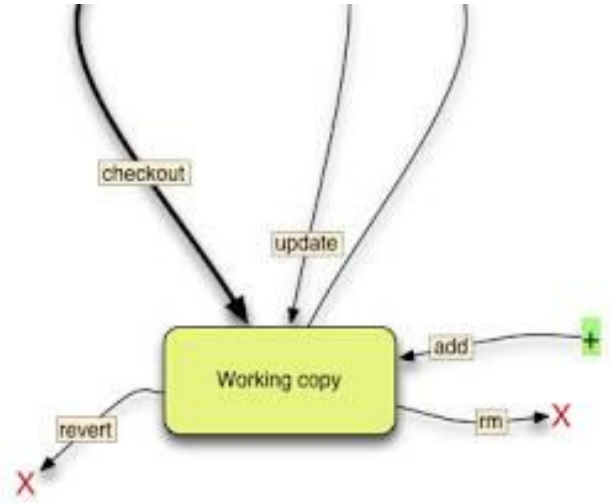
12.2.2 Commit

Um **commit** é o registo de um conjunto de alterações no seu repositório. Cada commit é como um "ponto de save" no seu projeto. Ele inclui:

- As alterações nos arquivos.
- Uma mensagem de commit que descreve o que foi alterado e porquê.
- O autor da alteração.
- Um identificador único (hash SHA-1).

Comits devem ser pequenos, focados e com mensagens claras.

Ilustração 12.3: Um Commit como um "Ponto de Save".



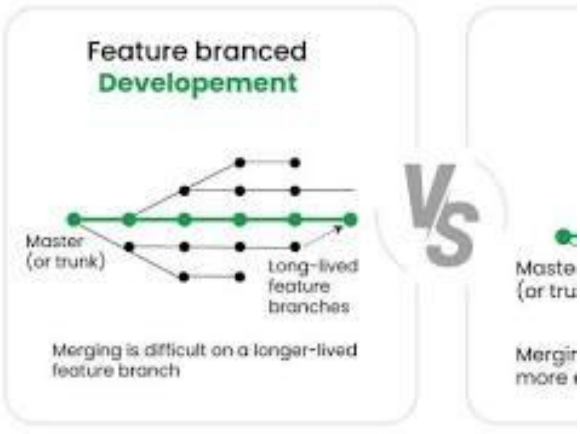
Fonte: stevebennett.me

12.2.3 Branch (Ramificação)

Um **Branch** (ramo/ramificação) é uma linha independente de desenvolvimento. Quando você cria um branch, está a criar uma cópia do seu código num determinado ponto no tempo, onde pode fazer alterações sem afetar a linha principal (geralmente chamada de main ou master).

- É ideal para desenvolver novas funcionalidades, corrigir bugs, ou experimentar ideias.
- Você pode ter múltiplos branches no mesmo repositório.

Ilustração 12.4: Branches para Desenvolvimento Paralelo.

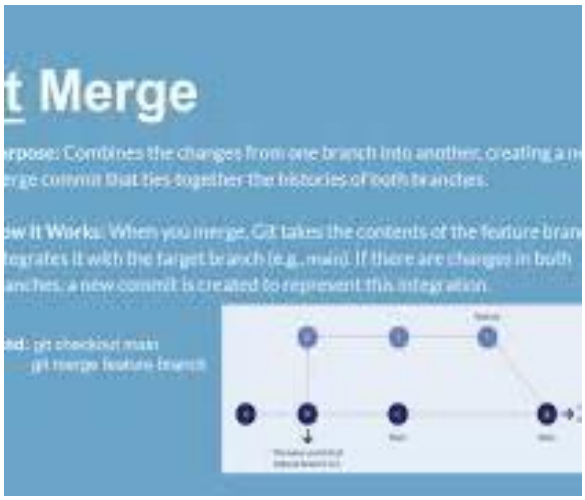


Fonte: www.geeksforgeeks.org

12.2.4 Merge (União)

Merge é o processo de integrar as alterações de um branch em outro. Uma vez que você terminou de desenvolver uma funcionalidade num branch, você pode uni-la (merge) de volta ao branch principal.

Ilustração 12.5: O Processo de Merge.



Fonte: www.thoughtwin.com

12.3 Comandos básicos do Git

Para usar o Git, você precisará de uma linha de comando (terminal).

12.3.1 Configuração inicial

Antes de começar, configure o seu nome e email. Eles serão usados nos seus commits.

```
git config --global user.name "Seu Nome"
git config --global user.email "seu.email@exemplo.com"
```

12.3.2 Iniciar um repositório

Para iniciar um novo repositório Git numa pasta existente:

```
cd minha_pasta_do_projeto
git init
```

12.3.3 Adicionar arquivos (git add)

Antes de fazer um commit, você precisa dizer ao Git quais arquivos (ou alterações dentro dos arquivos) você quer incluir no próximo commit. Isso é feito adicionando-os à "staging area" (área de preparação).

```
git add nome_do_arquivo.html    # Adiciona um arquivo específico
git add pasta/              # Adiciona todos os arquivos numa pasta
git add .                   # Adiciona todas as alterações na pasta atual
```

12.3.4 Fazer um commit (git commit)

Depois de adicionar as alterações à staging area, você pode fazer o commit.

```
git commit -m "Mensagem do meu commit: Descreve o que foi feito."
# Exemplo: git commit -m "Adiciona a página inicial e estilos básicos"
```

12.3.5 Verificar o estado (git status)

Este comando mostra o estado atual do seu repositório: quais arquivos foram alterados, quais estão na staging area, etc.

```
git status
```

12.3.6 Ver o histórico (git log)

Mostra o histórico de commits do repositório.

```
git log
```

12.3.7 Criar e mudar de branch (git branch, git checkout)

```
git branch nome_do_branch          # Cria um novo branch
git branch                         # Lista todos os branches
git checkout nome_do_branch        # Muda para um branch existente
git checkout -b novo_branch_e_muda    # Cria e muda para um novo branch (atalho)
```

12.3.8 Unir branches (git merge)

Para unir um branch ao seu branch atual (certifique-se de estar no branch que vai receber as alterações, ex: main).

```
git checkout main               # Vai para o branch principal
git merge nome_do_branch_para_unir   # Une as alterações do outro branch
```

12.3.9 Lidar com repositórios remotos (git remote, git push, git pull)

Para colaborar e fazer backup, você interage com repositórios remotos (como no GitHub).

```
git remote add origin https://github.com/seu-usuario/seu-repo.git  # Adiciona um remoto (geralmente 'origin')
git push -u origin main                          # Envia o branch 'main' para o remoto pela primeira vez
git push                                  # Envia as alterações para o remoto
git pull                                  # Puxa as alterações do remoto para o seu local
```

12.4 Fluxo de trabalho básico com Git

Um fluxo de trabalho comum para um projeto seria:

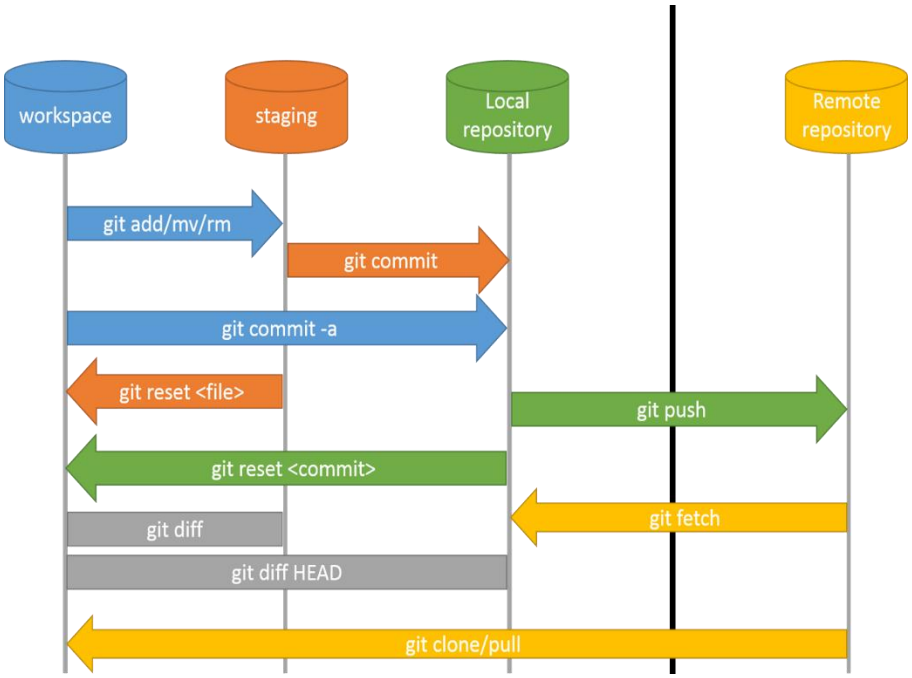
1. **Clonar (ou criar):** git clone <URL_do_repo_remoto> ou git init.
2. **Trabalhar:** Fazer alterações nos arquivos do projeto.
3. **Verificar Estado:** git status para ver o que foi alterado.
4. **Adicionar:** git add . (para todas as alterações).
5. **Commit:** git commit -m "Mensagem significativa".
6. **Puxar (Pull):** git pull origin main (antes de enviar, para ter certeza de que você tem as últimas alterações de outras pessoas).
7. **Enviar (Push):** git push origin main.

Para novas funcionalidades ou bugs, o ideal é usar branches:

1. git checkout main (ir para o branch principal).
2. git pull origin main (garantir que está atualizado).
3. git checkout -b feature/minha-nova-funcionalidade (criar e ir para um novo branch).
4. Desenvolver, adicionar e commitar no novo branch.
5. git push origin feature/minha-nova-funcionalidade (enviar o seu branch para o remoto).

6. Quando a funcionalidade estiver pronta, voltar para main e unir: git checkout main git merge feature/minha-nova-funcionalidade git push origin main

Ilustração 12.6: Fluxo de Trabalho Comum com Git.



Fonte: blog.isquaredsoftware.com

Exercícios:

1. Crie uma nova pasta no seu computador e inicie um repositório Git nela (git init).
2. Crie um arquivo de texto simples chamado meu_primeiro_arquivo.txt dentro dessa pasta e escreva algo nele.
3. Adicione este arquivo ao Git e faça o seu primeiro commit com uma mensagem como "Primeiro commit: adiciona arquivo de teste".
4. Modifique o meu_primeiro_arquivo.txt e adicione mais uma linha.
5. Verifique o estado do repositório (git status) e observe as mudanças.
6. Crie um novo branch chamado feature/nova-linha.
7. Mude para o branch feature/nova-linha.
8. Adicione e comite as alterações do meu_primeiro_arquivo.txt no branch feature/nova-linha com a mensagem "Adiciona nova linha no branch feature".
9. Volte para o branch main (ou master). Observe que a nova linha não aparece.

10. Una o branch feature/nova-linha no branch main (git merge).
11. Verifique o meu_primeiro_arquivo.txt novamente no branch main para confirmar que a nova linha foi integrada.

Módulo 12: Projeto Prático - Sistema de Ocorrências Disciplinares

Chegamos ao último módulo da nossa apostila, onde você terá a oportunidade de consolidar todo o conhecimento adquirido num projeto prático. Este projeto simulará um sistema de gestão de ocorrências disciplinares para uma escola, aplicando HTML, CSS, JavaScript, PHP e CodeIgniter, além de conceitos de banco de dados e controle de versão.

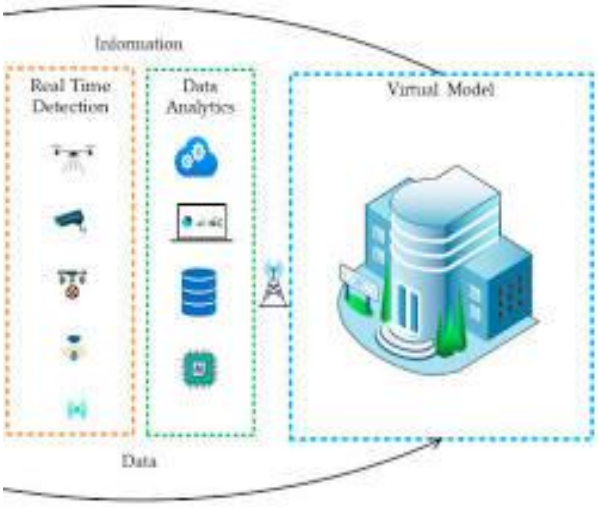
12.1 Introdução ao Projeto: Sistema de Ocorrências Disciplinares

O objetivo deste projeto é construir uma aplicação web que permita a uma escola registar, consultar e gerir ocorrências disciplinares de alunos. O sistema deve ser acessível a diferentes tipos de usuários (administradores, professores, alunos), cada um com níveis de permissão específicos.

Funcionalidades principais:

- **Autenticação de Usuários:** Login e Logout.
- **Gestão de Usuários:** Cadastro e visualização de usuários com diferentes níveis de acesso (Admin, Professor, Aluno).
- **Gestão de Ocorrências:** Cadastro, visualização, edição e exclusão de ocorrências disciplinares.
- **Anexos:** Possibilidade de anexar arquivos (imagens, documentos) a uma ocorrência.
- **Relatórios:** Geração de relatórios básicos sobre as ocorrências.

Ilustração 12.1: Sistema de Ocorrências Disciplinares da Escola.



Fonte: www.mdpi.com

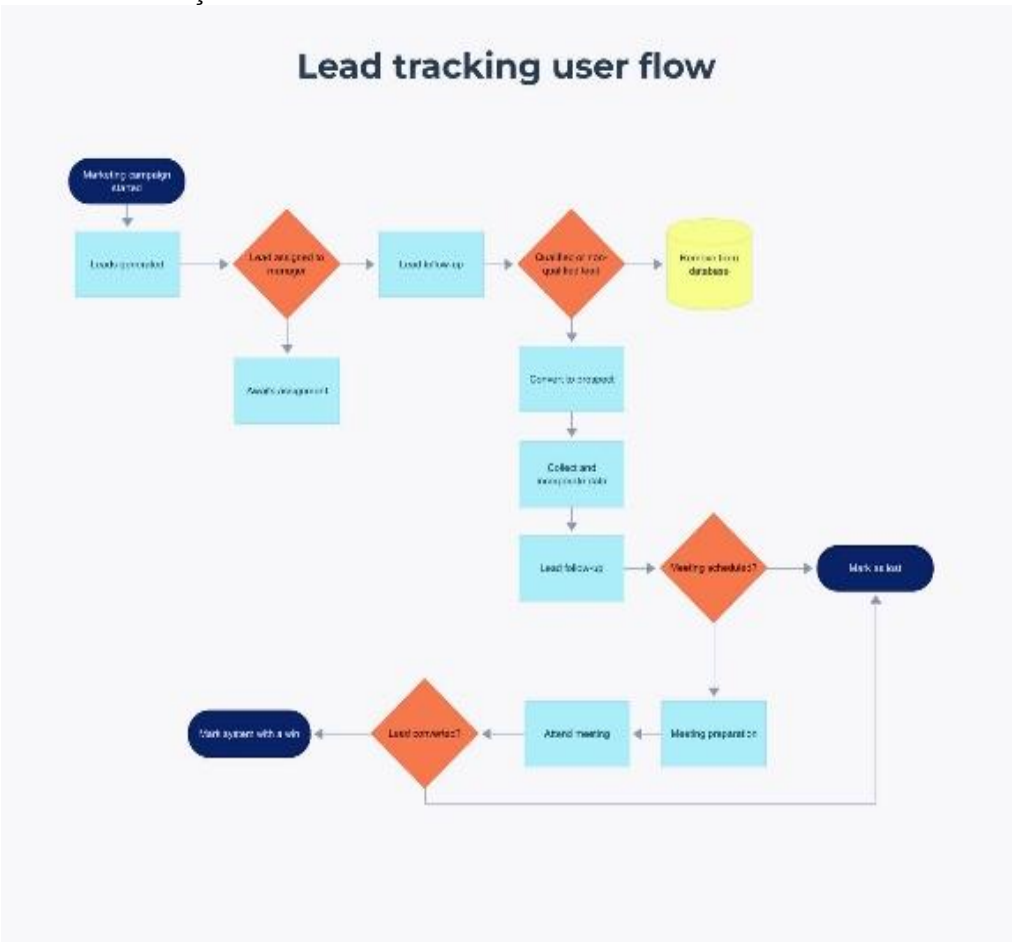
12.2 Requisitos funcionais

Vamos detalhar as funcionalidades que o sistema deve ter:

1. **Módulo de Autenticação:**
    - **Login:** Formulário para usarios inserirem email/nome e senha entre outros atributos.
    - **Logout:** Funcionalidade para encerrar a sessão.
    - **Controle de Acesso:** Redirecionamento de usuários não autenticados para a página de login.
2. **Módulo de Gestão de Usuários:**
    - **Cadastro de Usuários (apenas Admin):** Formulário para um administrador registar novos Usuários (alunos, professores, outros administradores), definindo seu nome, email, senha e nível de acesso.
    - **Listagem de Usuários (apenas Admin):** Tabela listando todos os usuários registrados.
    - **Edição/Exclusão de usua´rio (apenas Admin):** Funcionalidades para modificar ou remover registos de usuários.
3. **Módulo de Gestão de Ocorrências:**
    - **Cadastro de Ocorrência (Admin/Professor):** Formulário para registar uma nova ocorrência, incluindo:
        - Aluno envolvido (seleção de uma lista de alunos existentes).

        - Professor responsável (seleção automática ou manual).
        - Data e Hora da ocorrência.
        - Tipo de ocorrência (ex: Atraso, Comportamento Inadequado, Ausência).
        - Descrição detalhada.
        - **Anexos:** Campo para upload de um ou mais arquivos (ex: imagens, PDFs).
    - **Listagem de Ocorrências:**
        - **Admin/Professor:** Ver todas as ocorrências.
        - **Aluno:** Ver apenas as ocorrências em que está envolvido.
    - **Visualização Detalhada da Ocorrência:** Página para ver todos os detalhes de uma ocorrência, incluindo anexos.
    - **Edição/Exclusão de Ocorrência (Admin/Professor):** Funcionalidades para modificar ou remover registos de ocorrências.
4. **Módulo de Relatórios (apenas Admin/Professor):**
    - **Relatório por Aluno:** Lista de todas as ocorrências de um aluno específico.
    - **Relatório por Período:** Lista de ocorrências dentro de um intervalo de datas.
    - **Relatório por Tipo de Ocorrência:** Agrupamento e contagem de ocorrências por tipo.

Ilustração 12.2: Exemlo de Fluxo de Usuários e Permissões.



Fonte: slickplan.com

12.3 Modelagem do Banco de Dados

Para armazenar os dados do sistema, propomos a seguinte estrutura de tabelas relacionais:

12.3.1 Tabela usuarios

Armazena informações de todos os usuários do sistema, incluindo o nível de acesso.

Coluna	Tipo de Dado	Restrições	Descrição
id_usuario	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único do usuário.
nome	VARCHAR(255)	NOT NULL	Nome completo do usuário.

email	VARCHAR(255)	NOT NULL, UNIQUE	Email do usuário (usado para login).
senha	VARCHAR(255)	NOT NULL	Senha hash do usuário.
nivel_acesso	ENUM('admin', 'professor', 'aluno')	NOT NULL, DEFAULT 'aluno'	Define o papel do usuário no sistema.
data_cadastro	DATETIME	DEFAULT CURRENT_TIMESTAMP	Data e hora do registo do usuário.

12.3.2 Tabela ocorrencias

Armazena os detalhes de cada ocorrência disciplinar.

Coluna	Tipo de Dado	Restrições	Descrição
id_ocorrencia	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único da ocorrência.
id_aluno	INT	NOT NULL, FOREIGN KEY	ID do aluno envolvido (referencia usuarios.id_usuario onde nivel_acesso='aluno').
id_professor	INT	NOT NULL, FOREIGN KEY	ID do professor que registou a ocorrência (referencia usuarios.id_usuario onde nivel_acesso='professor').
data_hora	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Data e hora em que a ocorrência aconteceu.
tipo_ocorrencia	VARCHAR(100)	NOT NULL	Ex: 'Atraso', 'Comportamento', 'Ausência'.
descricao	TEXT	NOT NULL	Descrição detalhada da ocorrência.
status	ENUM('aberta', 'resolvida')	NOT NULL, DEFAULT 'aberta'	Status da ocorrência.

12.3.3 Tabela anexos

Armazena informações sobre os arquivos anexados a cada ocorrência.

Coluna	Tipo de Dado	Restrições	Descrição
id_anexo	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único do anexo.
id_ocorrencia	INT	NOT NULL, FOREIGN KEY	ID da ocorrência a que o anexo pertence.
nome_arquivo	VARCHAR(255)	NOT NULL	Nome original do arquivo.
caminho_arquivo	VARCHAR(255)	NOT NULL, UNIQUE	Caminho completo do arquivo no servidor.
tipo_mime	VARCHAR(100)	NOT NULL	Tipo MIME do arquivo (ex: 'image/jpeg', 'application/pdf').
tamanho_bytes	INT	NOT NULL	Tamanho do arquivo em bytes.
data_upload	DATETIME	DEFAULT CURRENT_TIMESTAMP	Data e hora do upload do anexo.

[Image of an Entity-Relationship Diagram (ERD) showing the three tables (usuarios, ocorrencias, anexos) with their columns and arrows indicating one-to-many relationships (e.g., one user can have many occurrences, one occurrence can have many attachments).]
Ilustração 12.3: Diagrama de Entidade-Relacionamento do Banco de Dados.

12.4 Estrutura da Aplicação com CodeIgniter (MVC)

Vamos organizar o projeto usando o padrão MVC do CodeIgniter:

12.4.1 Controladores (app/Controllers/)

- **Auth.php:** Lida com o login e logout de usuários.
  - Métodos: login(), processaLogin(), logout().
- **Admin.php:** Controlador para funcionalidades exclusivas do administrador (gestão de usuários).
  - Métodos: index(), listarUsuarios(), cadastrarUsuario(), editarUsuario(), excluirUsuario().

- **Ocorrencias.php:** Lida com as operações CRUD de ocorrências.
    - Métodos: index() (listar), cadastrar(), editar(), visualizar(), excluir().
- **Relatorios.php:** Responsável por gerar os diferentes tipos de relatórios.
    - Métodos: index(), porAluno(), porPeriodo(), porTipo().
- **Dashboard.php:** Página inicial após o login, com informações sumarizadas baseadas no nível de acesso.
    - Métodos: index().

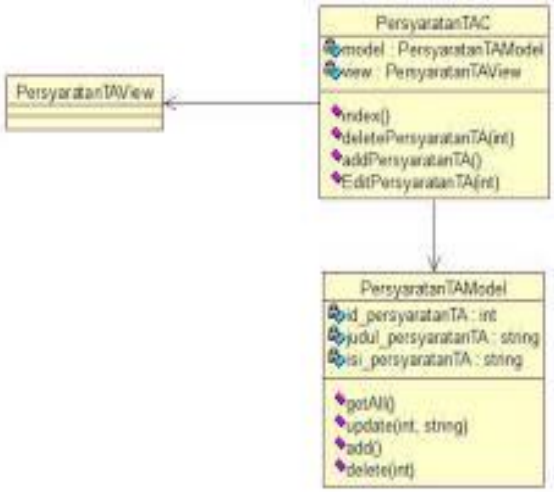
12.4.2 Modelos (app/Models/)

- **UsuarioModel.php:** Interage com a tabela usuarios.
    - Métodos: getUsuarioPorEmail(), criarUsuario(), getUsuariosPorNivel(), atualizarUsuario(), apagarUsuario(), getAlunos(), getProfessores().
- **OcorrenciaModel.php:** Interage com a tabela ocorrencias.
    - Métodos: criarOcorrencia(), getOcorrencia(), getTodasOcorrencias(), getOcorrenciasPorAluno(), getOcorrenciasPorPeriodo(), getOcorrenciasPorTipo(), atualizarOcorrencia(), apagarOcorrencia().
- **AnexoModel.php:** Interage com a tabela anexos.
    - Métodos: salvarAnexo(), getAnexosPorOcorrencia(), apagarAnexo().

12.4.3 Visões (app/Views/)

- **auth/login.php:** Formulário de login.
- **admin/usuarios/listar.php:** Tabela de usuários.
- **admin/usuarios/cadastro.php:** Formulário de cadastro/edição de usuário.
- **ocorrencias/cadastro.php:** Formulário de cadastro/edição de ocorrência.
- **ocorrencias/listar.php:** Tabela de ocorrências.
- **ocorrencias/visualizar.php:** Detalhes de uma ocorrência.
- **relatorios/index.php:** Menu de relatórios.
- **relatorios/por_aluno.php:** Relatório de ocorrências por aluno.
- **dashboard.php:** Página inicial personalizada.
- **templates/header.php, templates/footer.php, templates/sidebar.php:** Para layout e componentes reutilizáveis.

Ilustração 12.4: Componentes MVC no Sistema de Ocorrências.



Fonte: stackoverflow.com

12.5 Implementação chave

Vamos focar em alguns aspetos cruciais para a implementação:

12.5.1 Controle de acesso e níveis de usuário

A autenticação será feita verificando as credenciais no banco de dados e armazenando o id_usuario e nivel_acesso na sessão. A autorização (verificar permissões) pode ser feita usando **Filtros (Filters)** do CodeIgniter ou diretamente nos Controladores.

Exemplo Conceitual (app/Controllers/Auth.php - método processaLogin):

```
<?php
// ... (código do controlador Auth)

public function processaLogin()
{
    $email = $this->request->getPost('email');
    $senhaPura = $this->request->getPost('senha');

    $usuarioModel = new \App\Models\UsuarioModel();
    $usuario = $usuarioModel->getUsuarioPorEmail($email);

    if ($usuario && password_verify($senhaPura, $usuario['senha'])) {
        // Credenciais válidas. Armazena dados na sessão.
        $session = session();
        $session->set([
            'id_usuario' => $usuario['id_usuario'],
```

```
            'nome_usuario' => $usuario['nome'],
            'nivel_acesso' => $usuario['nivel_acesso'],
            'logado' => TRUE
        ]);

        // Redireciona com base no nível de acesso
        if ($usuario['nivel_acesso'] === 'admin') {
            return redirect()->to('/admin/dashboard');
        } else {
            return redirect()->to('/dashboard');
        }
    } else {
        // Falha no login
        session()->setFlashdata('erro', 'Email ou senha inválidos.');
        return redirect()->back()->withInput();
    }
}
?>
```

Exemplo conceitual (app/Config/Filters.php ou dentro de um Controller):

Para restringir acesso a certas rotas (ex: /admin/* só para 'admin'):

```
<?php
// app/Config/Filters.php (Exemplo de um filtro para verificar admin)
// (Você criaria um filtro como App\Filters\AdminFilter.php)

namespace App\Filters;

use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;

class AdminFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null)
    {
        $session = session();
        if (!$session->get('logado') || $session->get('nivel_acesso') !== 'admin') {
            session()->setFlashdata('erro', 'Acesso negado. Apenas administradores.');
            return redirect()->to('/login');
        }
    }

    public function after(RequestInterface $request, ResponseInterface $response, $arguments
= null)
    {
        // Nada para fazer depois da requisição
    }
}
```

E no app/Config/Filters.php você registaria este filtro para as rotas admin.

12.5.2 Cadastro de Ocorrências com anexos

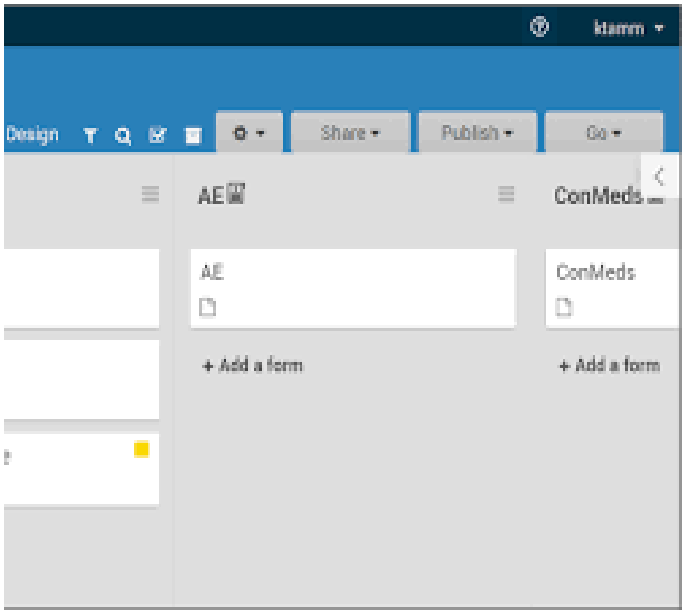
O formulário de cadastro de ocorrências deve ter um campo de upload de arquivos. O CodeIgniter facilita o tratamento de uploads.

Exemplo Conceitual (app/Controllers/Ocorrencias.php - método cadastrar):

```
<?php
// ... (código do controlador Ocorrencias)

public function cadastrar()
{
    if ($this->request->getMethod() === 'post') {
        $ocorrenciaModel = new \App\Models\OcorrenciaModel();
        $anexoModel = new \App\Models\AnexoModel();

        // Validação dos dados do formulário (IMPORTANTE!)
        $rules = [
            'id_aluno' => 'required|integer',
            'data_ocorrencia' => 'required|valid_date',
            'tipo_ocorrencia' => 'required',
            'descricao' => 'required',
            'anexos.*' => 'max_size[anexos,2048]|ext_in[anexos,jpg,jpeg,png,pdf]' // Até 2MB, apenas
jpg, png, pdf
        ];

        if (!$this->validate($rules)) {
            session()->setFlashdata('errors', $this->validator->getErrors());
            return redirect()->back()->withInput();
        }

        // Dados da ocorrência
        $dadosOcorrencia = [
            'id_aluno' => $this->request->getPost('id_aluno'),
            'id_professor' => session()->get('id_usuario'), // Professor logado
            'data_hora' => $this->request->getPost('data_ocorrencia') . ' ' . $this->request-
>getPost('hora_ocorrencia'),
            'tipo_ocorrencia' => $this->request->getPost('tipo_ocorrencia'),
            'descricao' => $this->request->getPost('descricao')
        ];

        $idOcorrencia = $ocorrenciaModel->insert($dadosOcorrencia); // Insere a ocorrência

        if ($idOcorrencia) {
            // Lidar com o upload de arquivos
            $files = $this->request->getFiles();
            if ($files && $files['anexos']) {
                foreach ($files['anexos'] as $file) {
```

```
            if ($file->isValid() && !$file->hasMoved()) {
                $newName = $file->getRandomName(); // Gera um nome único para o arquivo
                $filePath = WRITEPATH . 'uploads/' . $newName; // Caminho para guardar

                $file->move(WRITEPATH . 'uploads', $newName); // Move o arquivo para a pasta de
uploads

                // Guarda as informações do anexo no banco de dados
                $anexoModel->insert([
                    'id_ocorrencia' => $idOcorrencia,
                    'nome_arquivo' => $file->getClientName(),
                    'caminho_arquivo' => $newName, // Guarda apenas o nome gerado, o caminho
base é fixo
                    'tipo_mime' => $file->getClientMimeType(),
                    'tamanho_bytes' => $file->getSize(),
                ]);
            }
          }
        }
        session()->setFlashdata('sucesso', 'Ocorrência cadastrada com sucesso!');
        return redirect()->to('/ocorrencias');
      } else {
        session()->setFlashdata('erro', 'Falha ao cadastrar ocorrência.');
        return redirect()->back()->withInput();
      }
    }
    // Lógica para exibir o formulário (GET request)
    $usuarioModel = new \App\Models\UsuarioModel();
    $alunos = $usuarioModel->where('nivel_acesso', 'aluno')->findAll();
    return view('ocorrencias/cadastro', ['alunos' => $alunos]);
}
```

Ilustração 12.5: Fluxo de Cadastro de Ocorrência com Anexos.



Fonte: docs.openclinica.com

12.5.3 Relatórios

Os relatórios envolverão consultas ao banco de dados, possivelmente com JOINs entre tabelas para obter informações combinadas. A exibição será feita nas Vistas.

Exemplo Conceitual (app/Models/OcorrenciaModel.php - método getOcorrenciasPorAluno):

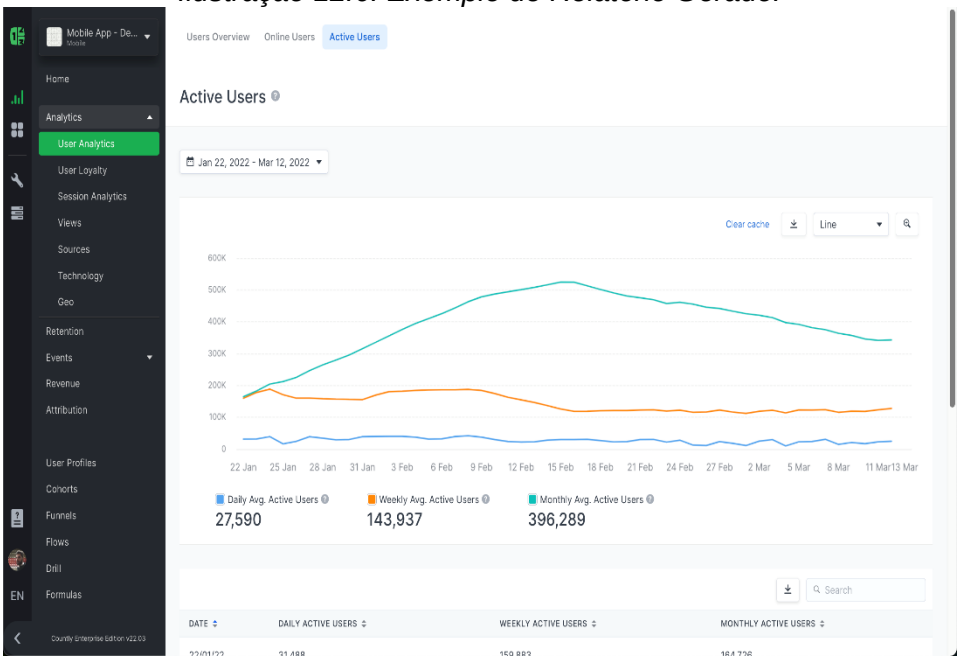
```
<?php
// ... (código do modelo OcorrenciaModel)

public function getOcorrenciasPorAluno($idAluno)
{
   return $this->select('ocorrencias.*, usuarios.nome as nome_aluno, professores.nome as nome_professor')
            ->join('usuarios', 'usuarios.id_usuario = ocorrencias.id_aluno') // Join com a tabela de alunos
            ->join('usuarios as professores', 'professores.id_usuario = ocorrencias.id_professor') // Join com a tabela de professores
            ->where('ocorrencias.id_aluno', $idAluno)
            ->orderBy('data_hora', 'DESC')
            ->findAll();
}
```

```
// Exemplo de uma query para relatório por período (sem join, para simplificar)
public function getOcorrenciasPorPeriodo($dataInicio, $dataFim)
{
   return $this->where('data_hora >=', $dataInicio)
            ->where('data_hora <=', $dataFim)
            ->orderBy('data_hora', 'ASC')
            ->findAll();
}
```

Exemplo conceitual (app/Views/relatorios/por_aluno.php):

```
<!-- ... (código HTML/CSS para layout) ... -->
<h1>Relatório de Ocorrências do Aluno: <?= esc($aluno['nome']) ?></h1>
<table class="table-auto w-full border-collapse border border-gray-400">
  <thead>
    <tr class="bg-gray-200">
      <th class="border px-4 py-2">Data/Hora</th>
      <th class="border px-4 py-2">Tipo</th>
      <th class="border px-4 py-2">Descrição</th>
      <th class="border px-4 py-2">Professor</th>
    </tr>
  </thead>
  <tbody>
    <?php if (!empty($ocorrencias)): ?>
      <?php foreach ($ocorrencias as $ocorrencia): ?>
        <tr>
          <td class="border px-4 py-2"><?= esc($ocorrencia['data_hora']) ?></td>
          <td class="border px-4 py-2"><?= esc($ocorrencia['tipo_ocorrencia']) ?></td>
          <td class="border px-4 py-2"><?= esc($ocorrencia['descricao']) ?></td>
          <td class="border px-4 py-2"><?= esc($ocorrencia['nome_professor']) ?></td>
        </tr>
      <?php endforeach; ?>
    <?php else: ?>
      <tr><td colspan="4" class="border px-4 py-2 text-center">Nenhuma ocorrência
encontrada para este aluno.</td></tr>
    <?php endif; ?>
  </tbody>
</table>
<!-- ... (código HTML/CSS para layout) ... -->
```

Ilustração 12.6: Exemplo de Relatório Gerado.



Fonte: support.countly.com

12.6 Próximos passos e desafios

A implementação deste projeto será um excelente exercício para aplicar tudo o que aprendeu. Comece por:

1. **Configurar o ambiente:** Instale um servidor web (Apache/Nginx), PHP e MySQL. Instale o CodeIgniter.
2. **Criar o Banco de Dados:** Execute os comandos CREATE TABLE para criar as tabelas usuarios, ocorrencias e anexos.
3. **Configurar o CodeIgniter:** Ajuste o app/Config/Database.php e app/Config/App.php (para URL base, etc.).
4. **Desenvolver o módulo de Autenticação:** Crie o Auth Controller, UsuarioModel e as Vistas de login/dashboard. Implemente o sistema de sessão e controle de acesso.
5. **Desenvolver a Gestão de Usuarios:** Crie as funcionalidades de CRUD para Admin.
6. **Desenvolver a Gestão de Ocorrências:** Implemente os formulários, a lógica do Controller e do Model para criar, listar, visualizar, editar e excluir ocorrências.

7. **Implementar upload de anexos:** Adicione a funcionalidade de upload e a gestão de arquivos e registos na tabela anexos.
8. **Desenvolver os Relatórios:** Crie as consultas no OcorrenciaModel e as Vistas para exibir os relatórios.
9. **Estilização:** Use Tailwind CSS para deixar a interface bonita e responsiva.
10. **Controle de versão:** Use Git para gerir o seu código, fazendo commits frequentes e utilizando branches para cada nova funcionalidade.

Desafios avançados (para quando se sentir confortável):

- Implementar um sistema de notificação (ex: enviar email ao aluno quando uma ocorrência é registada).
- Adicionar um campo de pesquisa para ocorrências e usuárioes.
- Criar um dashboard mais interativo com gráficos (usando Chart.js ou D3.js no frontend).
- Implementar a recuperação de senha.
- Tratamento de erros mais robusto e logging.

12.7 Referências para aprofundamento

Para complementar a sua aprendizagem e aprofundar os conceitos abordados nesta apostila, recomendamos os seguintes recursos:

Geral e fundamentos de desenvolvimento Web:

- **MDN Web Docs (Mozilla Developer Network):**
    - Um dos recursos mais completos e atualizados para HTML, CSS, JavaScript e conceitos web em geral.
    - https://developer.mozilla.org/pt-BR/ (Versão em Português do Brasil)
- **W3Schools:**
    - Tutoriais simples e exemplos interativos para HTML, CSS, JavaScript, SQL e PHP. Excelente para revisão rápida e prática.
    - https://www.w3schools.com/ (Conteúdo em inglês, mas muito intuitivo)

PHP:

- **Documentação oficial do PHP:**
    - A fonte mais autorizada e completa para a linguagem PHP. Essencial para detalhes específicos de funções e recursos.
    - https://www.php.net/manual/pt_BR/ (Versão em Português do Brasil)

CodeIgniter:

- **Documentação oficial do CodeIgniter:**
    - Indispensável para aprender e usar o framework. Contém guias de usuário, tutoriais e referências de API.
    - https://codeigniter.com/user_guide/ (Para a versão 4, em inglês. Há traduções para versões anteriores por terceiros, mas o oficial é o mais recomendado)

Banco de Dados e SQL:

- **SQL Tutorial (W3Schools):**
    - Um guia prático para aprender e praticar comandos SQL básicos e avançados.
    - https://www.w3schools.com/sql/
- **MySQL *Documentation*:**
    - Documentação detalhada para o sistema de gerenciamento de banco de dados MySQL.
    - https://dev.mysql.com/doc/

Git e Controle de Versão:

- **Pro Git Book (livro gratuito):**
    - Um livro abrangente e gratuito sobre Git, cobrindo desde os fundamentos até tópicos avançados.

    - https://git-scm.com/book/pt-br/v2 (Versão em Português do Brasil)
- **GitHub Docs:**
    - Recursos e guias sobre como usar o GitHub (uma plataforma popular para hospedar repositórios Git) e seus recursos.
    - https://docs.github.com/pt/ (Versão em Português)
    -

Segurança na Web:

- **OWASP Top 10:**
    - Uma lista das 10 vulnerabilidades de segurança mais críticas em aplicações web, mantida pela Open Web Application Security Project (OWASP). Essencial para qualquer desenvolvedor.
    - https://owasp.org/www-project-top-ten/ (Versão em inglês, mas os conceitos são universais e traduzíveis)

Este projeto irá exigir tempo e dedicação, mas o resultado será uma aplicação web funcional que demonstra um domínio significativo dos conceitos de desenvolvimento web.

Desejamos uma excelente aventura no desafiante e instigante universo da programação!

ÍNDICE REMISSIVO