Conceitos em Programação Web

Feito por: Guilherme Henrique Almeida da Silva

Contexto Histórico

A história do desenvolvimento web dinâmico começa com a Web Estática no início dos anos 1990. Nesse período, os sites eram compostos apenas por documentos HTML servidos diretamente pelo servidor ao navegador do usuário, sem qualquer forma de personalização ou interação dinâmica. As páginas eram fixas, o conteúdo era o mesmo para todos os usuários e qualquer atualização exigia a alteração manual do HTML. A interação com o usuário era extremamente limitada.

Com o crescimento da internet e a necessidade de tornar os sites mais interativos, surgiu o CGI (Common Gateway Interface). Essa técnica, amplamente utilizada a partir de meados dos anos 1990, permitia que servidores web executassem scripts externos (normalmente em linguagens como C ou Perl) em resposta a requisições de usuários, como o envio de formulários. O CGI possibilitou, pela primeira vez, a geração de conteúdo dinâmico. No entanto, cada requisição iniciava um novo processo no sistema operacional, o que compromete a escalabilidade e o desempenho, especialmente sob alta carga.

Para resolver essas limitações, surgiram os primeiros mecanismos de **server-side scripting** no final dos anos 1990. Diferente do CGI tradicional, os scripts do lado do servidor (como PHP, ASP e JSP) são interpretados diretamente pelo servidor sem a necessidade de criar novos processos para cada requisição. Isso trouxe ganhos significativos de desempenho e simplificou o desenvolvimento de aplicações web dinâmicas, pois os códigos podiam ser embutidos diretamente dentro do HTML. Além disso, permitiam acesso a bancos de dados, sessões de usuários, e lógica condicional para personalizar a resposta enviada ao navegador.

Durante os anos 2000, esses scripts server-side evoluíram com o surgimento dos frameworks backend, como Laravel (PHP), Django (Python), Ruby on Rails (Ruby) e ASP.NET MVC (C#). Esses frameworks trouxeram organização ao código por meio de padrões como o MVC (Model-View-Controller), abstrações de banco de dados por meio de ORMs (Object-Relational Mappers), e ferramentas que aceleram o desenvolvimento de aplicações robustas e escaláveis.

Paralelamente, no lado do cliente, o JavaScript ganhou destaque com a introdução da técnica AJAX (Asynchronous JavaScript and XML). AJAX permitiu que páginas web se comunicassem com o servidor de forma assíncrona, sem precisar recarregar a página inteira. Isso marcou o início de uma nova era na

experiência do usuário, com páginas mais rápidas e interativas. Começou-se então a delegar parte do processamento para o navegador, dividindo responsabilidades entre o cliente e o servidor.

A partir da década de 2010, surgiu o conceito de SPAs (Single Page Applications), apoiado por frameworks JavaScript como Angular, React e Vue.js. Nessa abordagem, a aplicação web carrega uma única página HTML inicial e, a partir dela, manipula dinamicamente o conteúdo com JavaScript, fazendo requisições a APIs no backend. Isso tornou possível criar interfaces altamente responsivas, semelhantes a aplicações desktop, com separação clara entre frontend e backend. A comunicação passou a ser feita majoritariamente por APIs REST ou GraphQL, utilizando o formato JSON.

Hoje, a arquitetura web moderna é baseada em aplicações full stack que combinam tecnologias de frontend e backend de maneira desacoplada. No backend, linguagens como JavaScript (Node.js), Python (FastAPI), Go e outras são utilizadas para construir APIs robustas. No frontend, frameworks como Next.js, Nuxt e SvelteKit permitem renderização híbrida (SSR, SSG), que oferece melhor desempenho e SEO. Além disso, com a popularização da computação em nuvem e serviços como AWS, Vercel e Netlify, o deployment e a escalabilidade de aplicações web se tornaram mais acessíveis e automatizados.

A transição do CGI para o scripting do lado do servidor e, posteriormente, para arquiteturas modernas baseadas em APIs e SPAs, reflete a necessidade crescente por desempenho, escalabilidade, manutenibilidade e experiência do usuário cada vez mais rica e dinâmica.

O que é o HTML?

HTML (HyperText Markup Language) é a linguagem de marcação padrão usada para criar e estruturar páginas da web. Ele não é uma linguagem de programação (como JavaScript ou Python), mas sim uma linguagem que organiza o conteúdo — como textos, imagens, vídeos e links — dizendo ao navegador como exibi-los na tela.

Principais características do HTML:

- **Marca a estrutura da página:** usa **tags** (etiquetas) como `<h1>`, `<p>`, `<a>`, `<img>`, etc., para identificar cabeçalhos, parágrafos, links, imagens, listas e muito mais.

- **Funciona com CSS e JavaScript:** o HTML cuida da estrutura, o **CSS** da aparência (cores, espaçamento, estilos) e o **JavaScript** da interatividade (animações, cliques, validações, etc.).

- **Hipertexto:** os links (hiperlinks) conectam páginas entre si, permitindo navegação fluida na web.

- **Base de todo site:** qualquer página web, mesmo as mais complexas, têm **HTML como base**.

O que é uma CGI?

CGI significa Common Gateway Interface. É uma interface padrão que permite que servidores web executem scripts ou programas externos e retornem os resultados para o navegador do usuário. Em outras palavras, CGI é o que permitiu às páginas web deixarem de ser puramente estáticas e começarem a responder dinamicamente às ações dos usuários, ou seja, deu início ao dinamismo das páginas web.

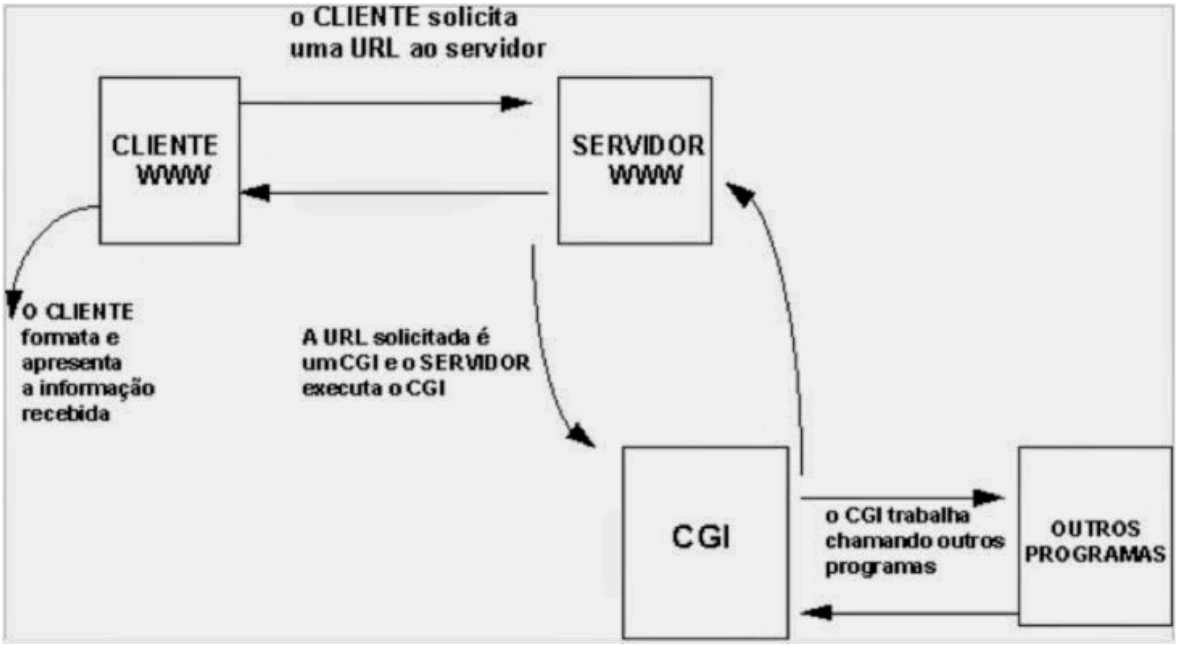
Como funciona uma CGI?

- O usuário envia uma requisição a uma página (por exemplo, ao enviar um formulário HTML).

- O servidor web identifica que essa requisição deve ser processada por um **script CGI** (normalmente arquivos .cgi, .pl, .py, .php, etc.).

- O servidor executa esse script como se fosse um programa no sistema operacional (geralmente no Linux/Unix, como processo filho).

- O script processa os dados recebidos (GET ou POST), realiza alguma lógica (consultar banco, gerar HTML dinâmico etc.) e envia **um conteúdo HTML como resposta** ao navegador.



Diferença entre CGI e tecnologias modernas (como Node, PHP ou frameworks):

Tecnologia	Geração de Resposta	Performance	Facilidade
CGI Puro	Executa novo processo a cada requisição	Baixa (cria processo novo)	Média
FastCGI	Reutiliza processos	Alta	Melhor
PHP/Node/ASP	Embutido no servidor ou via módulo	Alta	Alta
Frameworks (Django, Express)	Orientado a rotas, MVC, etc.	Muito alta	Alta

Exemplo de CGI:

```
#include <stdio.h>

int main(void) {
    // Cabeçalho HTTP necessário para resposta CGI
```

```
    printf("Content-type: text/html\n\n");

    // Conteúdo HTML
    printf("<html>\n");
    printf("<head><title>CGI em C</title></head>\n");
    printf("<body>\n");
    printf("<h1>Olá, Mundo!</h1>\n");
    printf("<p>Este é um script CGI escrito em C.</p>\n");
    printf("</body>\n");
    printf("</html>\n");

    return 0;
}
```

Observação: Esse script é executado pelo servidor, e o navegador só vê o HTML de saída.

Onde ficam os scripts CGI?

Geralmente em um diretório reservado, como: `/cgi-bin/`

Exemplo de URL: `http://meusite.com/cgi-bin/formulario.cgi`

Lembre-se: http é a porta 80 e o https é a porta 8080

Como testar o código de exemplo?

- **Salve o código como:** `ola.cgi`.
- **Compile com:** `gcc ola.cgi -o ola.cgi`
- **Dê permissão de execução:** `chmod +x ola.cgi`
- **Mova para o diretório** `cgi-bin` **do servidor**: `sudo mv ola.cgi /usr/lib/cgi-bin/`
- **Acesse no navegador:** `http://localhost/cgi-bin/ola.cgi`

O que é o CSS?

CSS (Cascading Style Sheets — Folhas de Estilo em Cascata) é uma **linguagem de estilo** usada para **descrever a aparência** de documentos escritos em HTML ou XML. Ele **separa o conteúdo da apresentação visual**, permitindo que os desenvolvedores controlem a formatação, layout e estilo de elementos da página web. O **CSS não é uma linguagem de marcação.**

Tipos de CSS

Inline (dentro da tag) **Exemplo:**

```
<p style="color: red;">Texto em vermelho</p>
```

Interno (dentro do `<head>`) **Exemplo:**

```
<style>
  p { color: green; }
</style>
```

Externo (arquivo `.css` separado) **Exemplo:**

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

Observação: O link deve estar dentro da tag <head>

Como o CSS funciona?

- O navegador carrega o **HTML**.
- Ele carrega os arquivos **CSS vinculados**.
- Cria a **árvore DOM** (Document Object Model).
- Aplica os estilos definidos no CSS à DOM.
- Renderiza a página com o estilo final.

Estrutura de uma regra (Sintaxe) do CSS ?

Uma regra CSS é composta por **seletor**, **propriedade** e **valor**:

```
h1 {
  color: red;
  font-size: 24px;
}
```

Seletor: h1 aplica o estilo a todos os títulos h1.
Propriedades: color, font-size.
Valores: red, 24px.

O que é o Scripting do Lado do Servidor (Server-Side Scripting)?

O **Scripting do Lado do Servidor** é uma técnica usada no desenvolvimento web em que os códigos são executados no **servidor**, antes de a página ser enviada ao navegador do usuário. Essa abordagem permite que as páginas web sejam **dinâmicas**, ou seja, geradas sob demanda com base em dados armazenados em bancos de dados, sessões de usuário, parâmetros de requisição, entre outros.

Quando o usuário acessa uma página web, o navegador envia uma requisição HTTP ao servidor. Se essa página utilizar scripts de servidor (escritos em linguagens como **PHP, Node.js, Python, Ruby**, entre outras), o código é **interpretado no servidor**. O resultado da execução — geralmente um documento HTML — é enviado ao navegador. O cliente **não tem acesso ao código do servidor**, apenas ao HTML final renderizado.

Essa técnica **substitui o modelo tradicional de CGI** (Common Gateway Interface), que gerava um novo processo no servidor a cada requisição. No Scripting do Lado do Servidor, o código é executado em um ambiente **persistente**, com melhor performance e escalabilidade. Assim, além de economizar recursos do sistema, oferece mais velocidade, segurança e facilidade de manutenção.

Em resumo: o navegador **não executa** o script do lado do servidor — ele apenas recebe o resultado final (HTML, CSS, JS) já processado e pronto para renderização.

Linguagens comuns para scripting no servidor (Tecnologias usadas)?

- ➢ **PHP** (Usando xampp)
- ➢ **Python** (com frameworks como Django, Flask)
- ➢ **Node.js** (JavaScript no servidor)
- ➢ **Ruby on Rails**
- ➢ **ASP.NET** (C#)
- ➢ **Java** (JSP, Spring)

O que é o Scripting do Lado do Cliente (Client-Side Scripting)?

É uma **técnica de desenvolvimento web** onde o código é executado diretamente **no navegador do usuário (cliente)**, **não no servidor**. Esse tipo de scripting é fundamental para **criar páginas interativas, responsivas e dinâmicas**, sem a

necessidade de recarregar a página toda a cada ação do usuário.O **servidor envia os arquivos** HTML, CSS e JavaScript para o navegador.O **JavaScript** é interpretado **pelo navegador (cliente)** e executado localmente.Isso permite manipular conteúdo, responder a eventos do usuário, e até buscar dados via APIs **— sem recarregar a página inteira**.

Tecnologias Usadas?

- **JavaScript** → principal linguagem de client-side scripting.
- **HTML + CSS** → para estrutura e estilo, que o script manipula dinamicamente.
- **APIs do navegador** → como **DOM, Fetch, Web Storage, Geolocation, Canvas, etc.**
- **Frameworks modernos**: **React, Vue, Angular, Svelte** — que evoluíram o client-side scripting para outro nível **(Single Page Applications, por exemplo).**

Por que supera o HTML estático?

- HTML estático é **fixo**: o conteúdo é sempre o mesmo, a menos que o servidor o altere.
- Client-side scripting **permite interações em tempo real**, sem depender do servidor.
- Reduz a carga no servidor: muitas ações podem ser resolvidas diretamente no cliente.
- Melhora a experiência do usuário (UX): **mais fluidez, menos recarregamento, mais dinamismo**

Diferença entre Server-Side e Client-Side Scripting?

Característica	Client-Side Scripting	Server-Side Scripting
Onde é executado	Navegador do usuário	Servidor web
Linguagens comuns	JavaScript	PHP, Python, Node.js, Ruby, etc.
Acesso a dados locais	Pode acessar DOM e localStorage	Pode acessar banco de dados e sessões
Visibilidade do código	Total (o código é entregue ao cliente)	Nenhuma (executa antes de chegar ao cliente)

Exemplo de uso	Validação de formulário, animações	Geração de HTML com base em banco de dados

O que SSR (Server-Side Rendering)?

Server-Side Rendering é uma técnica onde o conteúdo HTML completo da página é gerado no servidor a cada requisição do usuário. Ou seja, quando um cliente acessa uma rota, o servidor processa os dados, executa o código necessário (por exemplo, consultas a banco de dados), monta o HTML com o conteúdo dinâmico já renderizado e envia esse HTML completo para o navegador.
Essa técnica melhora o tempo de carregamento inicial e é muito eficiente para SEO, pois os mecanismos de busca recebem o conteúdo já pronto, facilitando a indexação.

Exemplo de uso: aplicações web com conteúdo dinâmico frequente, que precisa estar sempre atualizado quando o usuário acessa a página (ex: e-commerce, sites de notícias, dashboards).

O que SSG(Static Site Generation)?

Static Site Generation é a geração do HTML das páginas no momento da **construção (build)** do projeto, e não no momento da requisição. Isso significa que o conteúdo da página é pré-gerado e salvo como arquivos HTML estáticos, que são servidos rapidamente a qualquer visitante.
Por ser estático, o tempo de resposta é muito rápido e o site pode ser hospedado em CDNs, resultando em alta performance com baixo custo. No entanto, ele é mais adequado para conteúdo que **não muda com frequência**.

Exemplo de uso: blogs, portfólios, documentações, landing pages — onde o conteúdo é o mesmo para todos os visitantes.

O que SEO (Search Engine Optimization)?

SEO é um conjunto de técnicas e práticas aplicadas no desenvolvimento e conteúdo de sites com o objetivo de **otimizar a indexação por mecanismos de busca** (como Google, Bing) e melhorar o posicionamento nas páginas de resultados (SERPs). Isso inclui uso correto de **tags semânticas HTML**, **URLs amigáveis**, **velocidade de carregamento**, **estrutura de navegação**, **conteúdo de qualidade**, **uso adequado de palavras-chave** e **acessibilidade**.

Resumo da Evolução da Programação Web + Conceitos?

Etapa	Linguagens	Característica principal
Web Estática(WWW)	HTML	Conteúdo fixo
CGI	C,PERL,SHELL	Execução externa por processo
Server-Side Scripting	PHP,ASP,JSP	Scripts integrados ao servidor (sem novos processos)
Frameworks Backend	Laravel, Django, Rails	Estruturação e produtividade
Client-Side Scripting	JavaScript + AJAX	Interatividade no navegador
Frontend Moderno	React, Vue, Angular	Aplicações SPA, renderização dinâmica
Web Full Stack/API	Node.js, APIs REST	Separação frontend/backend, dados via API