2. ARQUITETURA DE APLICAÇÕES WEB

ARQUITETURA DE APLICAÇÕES WEB

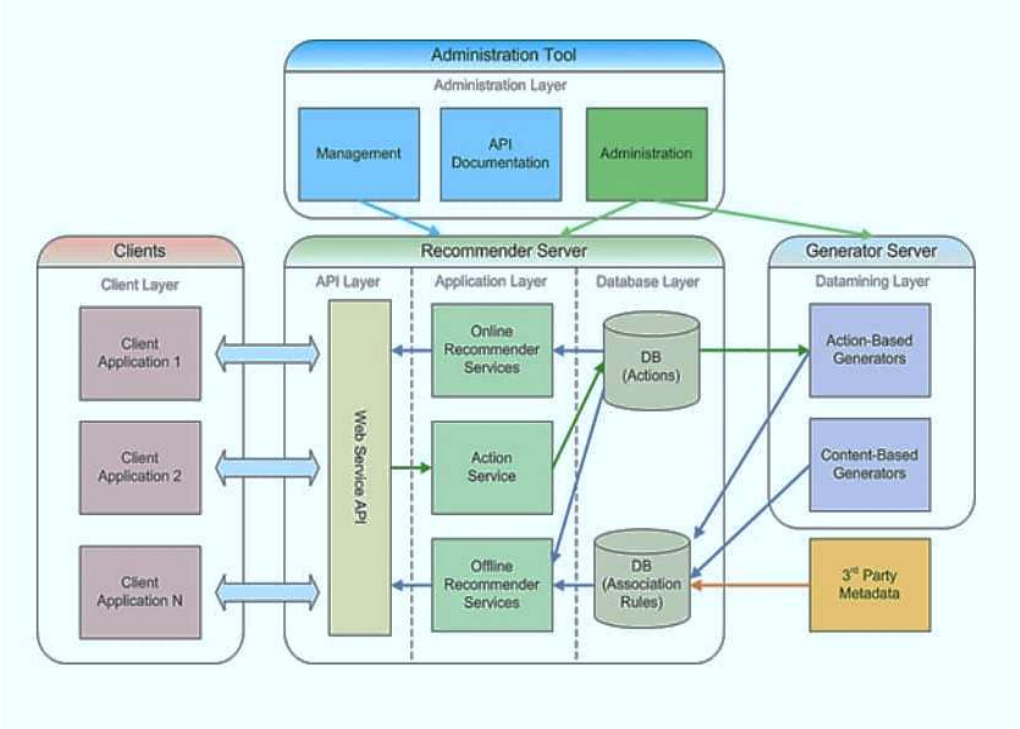
O que é arquitetura de aplicações web?

Em palavras simples, é um esboço de como vários componentes do seu aplicativo web interagem entre si.

Pode ser tão simples quanto definir o relacionamento entre o cliente e o servidor. Também pode ser tão complexo quanto definir as inter-relações entre um enxame de servidores, backend de contêineres, balanceadores de carga, gateways API e frontend de página única voltados para o usuário. Dito isto, raramente se trata de escolher a linguagem de programação na qual você vai escrever seu código.

ARQUITETURA DE APLICAÇÕES WEB

Como você projeta seu aplicativo web tem um papel fundamental tanto em sua usabilidade quanto em sua otimização de custos. Veja como é um exemplo de arquitetura de aplicações web no papel:



ARQUITETURA DE APLICAÇÕES WEB

Por que a arquitetura de aplicações web é importante?

A arquitetura da aplicações web é, sem dúvida, uma das partes mais importantes do seu aplicativo web. Caso você optar por desenvolver seu aplicativo web com uma arquitetura específica em mente, você certamente receberá muitos benefícios quando se trata de manter e crescer seu aplicativo. Entretanto, a escolha da arquitetura certa, amplifica ainda mais esses benefícios.

ARQUITETURA DE APLICAÇÕES WEB

Fácil adaptação às necessidades da empresa

- A aplicação é vital para o negócio e deve ser flexível para acompanhar as mudanças do mercado, adaptando-se às necessidades em constante evolução da empresa.
- Uma boa arquitetura web antecipa necessidades futuras, permitindo adaptações eficientes conforme o negócio evolui.
- Por exemplo, ao construir um eCommerce escalável, optar por uma arquitetura de microsserviços em vez de uma monolítica proporciona maior flexibilidade para atender a uma ampla gama de serviços e clientes no futuro.

Desenvolvimento organizado

O desenvolvimento de aplicações sem uma arquitetura em mente, você arrisca perder tempo e dinheiro reorganizando seus componentes e estabelecendo novas regras para ajudar a facilitar a colaboração entre os membros da sua equipe – tempo e dinheiro que de outra forma poderiam ter sido gastos em outro lugar.

ARQUITETURA DE APLICAÇÕES WEB

Melhor gerenciamento da base de código

- Gerenciar um aplicativo envolve organizar arquivos, modularizar e configurar pipelines para garantir um desenvolvimento eficiente.
- Uma boa arquitetura facilita mudanças, separa componentes problemáticos e promove independência entre funcionalidades, simplificando o processo.
- Ela também acelera o desenvolvimento e prepara o aplicativo para futuras migrações tecnológicas sem grandes reestruturações.

ARQUITETURA DE APLICAÇÕES WEB

Segurança aprimorada

- A arquitetura de aplicações considera a segurança desde o planejamento, permitindo implementar medidas eficazes antes do lançamento.
- Por exemplo, uma arquitetura de microsserviços em um app de streaming separa componentes como autenticação e conteúdo, garantindo flexibilidade e minimizando impactos em caso de falhas.
- Já em uma arquitetura monolítica, falhas em autenticação poderiam comprometer todo o sistema, criando riscos indesejáveis, como a liberação gratuita de conteúdo pago.

TIPOS DE ARQUITETURA DE APLICAÇÕES WEB

ARQUITETURA DE PÁGINA ÚNICA - SPA

A arquitetura de aplicativos de página única (Single-Page Application ou SPA) é tão simples quanto seu nome: o aplicativo inteiro é baseada em uma única página. Uma vez que o usuário baixa o seu aplicativo, ele não precisa navegar para qualquer outra página da internet. O aplicativo é tornado dinâmico o suficiente para buscar e renderizar telas que atendam aos requisitos dos usuários enquanto eles navegam pelo próprio aplicativo.

ARQUITETURA DE PÁGINA ÚNICA - SPA

Vantagens da arquitetura SPA

- Você pode construir aplicativos web altamente interativas.
- Os SPAs são fáceis de escalar.
- Otimizar o SPA para desempenho não requer muito esforço.

ARQUITETURA DE PÁGINA ÚNICA - SPA

Contras da arquitetura SPA

- SPA limitam a flexibilidade com hiperlinks e SEO.
- A renderização inicial é normalmente lenta.
- A navegação através do aplicativo pode ser pouco intuitiva.

ARQUITETURA PROGRESSIVA DE APLICATIVOS WEB - PWA

A arquitetura Progressive Web Application (PWA) é construída com base na Arquitetura de Página Única, fornecendo capacidades offline para o aplicativo web. Tecnologias como Capacitor e Ionic são usadas para construir PWAs que podem fornecer aos usuários uma experiência uniforme entre plataformas.

Semelhantes ao SPA, os PWAs são suaves e sem problemas. Com a capacidade adicional de serem instalados em dispositivos do usuário (por workers de serviço), seus usuários obtêm uma experiência mais uniforme com seu aplicativo.

ARQUITETURA PROGRESSIVA DE APLICATIVOS WEB - PWA

Prós da arquitetura PWA

- Os aplicativos funcionam de forma muito suave e oferecem compatibilidade entre plataformas.
- A escalabilidade é simples.
- Acesso off-line e APIs nativas do dispositivo, como workers em segundo plano e notificações push, são acessíveis aos desenvolvedores.

ARQUITETURA PROGRESSIVA DE APLICATIVOS WEB - PWA

Contras da arquitetura PWA

- Há um suporte limitado para gerenciamento de links e SEO.
- Mover atualizações para PWA offline é mais complexo do que com aplicativos nativos.
- Há um suporte limitado para PWA em navegadores de internet e sistemas operacionais.

ARQUITETURA RENDERIZADA DO LADO DO SERVIDOR – SSR

Na renderização do lado do servidor (SSR), às páginas da internet frontend são renderizadas em um servidor backend após serem solicitadas pelo usuário. Isto ajuda a reduzir a carga no dispositivo cliente ao receber um HTML estático, CSS, e JS webpage.

Os aplicativos SSR são muito populares entre os blogs e sites de eCommerce. Isto porque eles tornam o gerenciamento de links e SEO bastante simples. Além disso, a primeira renderização para aplicativos SSR é bastante rápida, já que o cliente não precisa processar nenhum código JS para renderizar às telas.

ARQUITETURA RENDERIZADA DO LADO DO SERVIDOR - SSR

Prós da Arquitetura SSR

- Estes aplicativos são ótimos para sites pesados em SEO.
- O carregamento da primeira página é quase instantânea na maioria dos casos.
- Você pode combiná-lo com um serviço de cache para melhorar ainda mais o desempenho do seu aplicativo.

ARQUITETURA RENDERIZADA DO LADO DO SERVIDOR – SSR

Contras da arquitetura SSR

- Não é recomendado para páginas de internet complexas ou pesadas, uma vez que o servidor pode levar tempo para gerar totalmente a página, resultando em um atraso na primeira renderização.
- Ele é recomendado principalmente para aplicativos que não focam muito na interface do usuário e estão apenas procurando por uma maior escalabilidade ou segurança.

ARQUITETURA MODELO CLIENTE-SERVIDOR

O modelo Cliente-Servidor é uma arquitetura de comunicação em redes onde um cliente faz requisições e um servidor responde a essas requisições. Ele é amplamente utilizado na internet e em aplicações distribuídas.

- **Cliente:** Dispositivo ou programa que solicita serviços ou dados ao servidor (exemplo: navegador web, aplicativo mobile).
- **Servidor:** Máquina ou software que processa as requisições dos clientes e fornece respostas (exemplo: um servidor web, um banco de dados).
- **Comunicação:** Geralmente ocorre via protocolos como HTTP, FTP ou WebSocket.

ARQUITETURA MODELO CLIENTE-SERVIDOR

Prós da arquitetura Modelo Cliente-Servidor

- Separação clara entre cliente e servidor, facilitando a manutenção.
- Possibilita escalabilidade, já que vários clientes podem acessar o mesmo servidor.
- Permite centralizar dados e processamento, garantindo mais segurança.

ARQUITETURA MODELO CLIENTE-SERVIDOR

Contras da arquitetura Modelo Cliente-Servidor

- Pode gerar sobrecarga no servidor caso muitos clientes acessem simultaneamente.
- A comunicação pode ser lenta se houver problemas de rede.
- O cliente depende do servidor para funcionar, tornando-o vulnerável a falhas no servidor.

ARQUITETURA MVC (MODEL-VIEW-CONTROLLER)

A arquitetura MVC é um padrão de design utilizado para organizar o código de aplicações, separando a lógica em três camadas: Model, View e Controller.

Model (Modelo)

- Responsável pelos dados e regras de negócio.
- Interage com o banco de dados.
- Exemplo: Uma classe Usuario que lida com a persistência de dados.

ARQUITETURA MVC (MODEL-VIEW-CONTROLLER)

View (Visão)

- Responsável pela interface com o usuário.
- Exibe as informações e captura as interações.
- Exemplo: Arquivos HTML com JavaScript e CSS.

Controller (Controlador)

- Gerencia as requisições do usuário.
- Atualiza o modelo e escolhe qual visão será exibida.
- Exemplo: Um arquivo UsuarioController.php que recebe dados do formulário e chama o Model para salvar no banco.

ARQUITETURA MVC (MODEL-VIEW-CONTROLLER)

Prós da Arquitetura Mvc (Model-view-controller)

- Separação de responsabilidades melhora a organização do código.
- Facilita a manutenção e a escalabilidade do sistema.
- Permite que diferentes equipes trabalhem simultaneamente em cada camada.
- Reutilização de código, já que a camada de Model pode ser usada por diferentes Views.

ARQUITETURA MVC (MODEL-VIEW-CONTROLLER)

Contras da Arquitetura Mvc (Model-view-controller)

- Pode adicionar complexidade desnecessária para projetos pequenos.
- A comunicação entre camadas pode gerar um leve impacto na performance.
- Pode ser difícil de aprender e implementar corretamente para iniciantes.

ARQUITETURA REST (REPRESENTATIONAL STATE TRANSFER)

REST é um estilo de arquitetura para construção de APIs baseadas em HTTP. Ele define um conjunto de princípios que tornam a comunicação entre sistemas escalável e eficiente.

Principais Características do REST:

- **Baseado em Recursos:** Cada endpoint representa um recurso (exemplo: /usuarios, /produtos).
- **Uso de Métodos HTTP:** GET → Buscar dados, POST → Criar novos dados, PUT → Atualizar dados, DELETE → Remover dados.
- **Formato de Dados:** Normalmente JSON, mas pode ser XML.
- **Stateless:** O servidor não mantém estado entre requisições.

ARQUITETURA REST (REPRESENTATIONAL STATE TRANSFER)

Prós da Arquitetura Rest

- Simples e fácil de integrar com diferentes tecnologias e plataformas.
- Utiliza HTTP, que já é amplamente suportado e conhecido.
- Stateless (sem estado), o que melhora a escalabilidade.
- APIs RESTful podem ser consumidas por qualquer tipo de cliente (web, mobile, IoT).

ARQUITETURA REST (REPRESENTATIONAL STATE TRANSFER)

Contras da Arquitetura Rest

- A ausência de estado pode exigir mais requisições para obter informações contextuais.
- Nem todas as APIs seguem o padrão REST corretamente, gerando inconsistências.
- Pode ser ineficiente para operações em tempo real, onde WebSockets são mais recomendados.

ARQUITETURA REST (REPRESENTATIONAL STATE TRANSFER)

Comparação MVC vs REST

MVC	REST
Padrão de design de software	Estilo de arquitetura para APIs
Organiza código em camadas (Model, View, Controller)	Define princípios para comunicação entre sistemas
Muito usado em aplicações web (exemplo: Laravel, Django)	Muito usado para criar APIs (exemplo: Node.js com Express, Flask)

EXERCICIO 2

EXERCÍCIO 2: Verifique sua Média

Agora faremos um exercício de lógica em **JavaScript** + **HTML**, onde o usuário insere o nome do aluno, as notas N1 e N2, e o sistema calcula a média para verificar se ele está aprovado ou reprovado.

Funcionalidades:

✓O aluno insere o **nome**, **nota 1 (N1)** e **nota 2 (N2)**.
✓O sistema **calcula a média** e verifica se é **maior ou igual a 7**.
✓Exibe uma **mensagem personalizada** para cada caso.

EXERCÍCIO 2: Verifique sua Média

```
<h2>Verifique sua Média</h2>
  <label for="nome">Nome do Aluno:</label>
  <input type="text" id="nome" />
  <br /><br />
  <label for="nota1">Nota 1:</label>
  <input type="number" id="nota1" min="0" max="10" />
  <br /><br />
  <label for="nota2">Nota 2:</label>
  <input type="number" id="nota2" min="0" max="10" />
  <br /><br />
  <button onclick="calcularMedia()">Calcular Média</button>
  <h3 id="resultado"></h3>
```

EXERCÍCIO 2: Verifique sua Média

```
<script>
    function calcularMedia() {
      let nome = document.getElementById("nome").value;
      let nota1 = parseFloat(document.getElementById("nota1").value);
      let nota2 = parseFloat(document.getElementById("nota2").value);
      let resultado = document.getElementById("resultado");

      // Validação
      if (nome === "" || isNaN(nota1) || isNaN(nota2)) {
        resultado.innerText =
          "Por favor, preencha todos os campos corretamente!";
        return;
      }
```

EXERCÍCIO 2: Verifique sua Média

```
// Cálculo da média
    let media = (nota1 + nota2) / 2;
       // Verificação de aprovação
     if (media >= 7) {
      resultado.innerText = `Parabéns, Você foi APROVADO com média ` + media;
         resultado.style.color = "green";
      } else {
         resultado.innerText = `Estude mais um pouco, Sua média foi ` + media;
         resultado.style.color = "red";
      }
     }
   </script>
```

TRABALHO EM GRUPO DE UM SISTEMA WEB

TEMAS:

1. Plataformas de Ensino Online
2. Plataformas de Streaming para Cursos de Tecnologia
3. Plataformas de Freelancers
4. Marketplaces de Produtos Digitais
5. Plataformas de Assinatura ou Comunidade
6. Lojas Virtuais ou E-commerce
7. Aplicativos de Delivery ou Serviços

TRABALHO EM GRUPO DE UM SISTEMA WEB

GRUPO: Máximo 8 pessoas;
TRABALHO DIGITADO: Proposta De Projeto Software
DEFESA: De 15 a 20 min.

DIGITADO E DEFESA: 25/03/25
DESENVOLVIDO: 14/05/25