



ARQUITETURA CLIENT-SIDE

Andre Luiz Mendes Pereira – Aula 01

Professores

ANDRE LUIZ MENDES PEREIRA

Professor Convidado

Fundador da Evolve, uma plataforma digital com propósito de mudar a clusterização do varejo, Andre lidera equipes de desenvolvimento ágeis multidisciplinares em multiplataformas, buscando a essência e resultados focados no business agility. Formado com honras ao mérito em Ciência da Computação, possui pós-graduação em Melhoria no Processos de Software, Arquitetura em Sistemas Distribuído e com MBA em Gestão Empresarial pela FGV. Agrega conhecimento nas diversas verticais na construção de softwares, dentre elas: arquitetura de soluções e de software, Devops e Cloud, contribuindo para que as empresas de TI encontrem as melhores soluções tecnológicas.

JÚLIO HENRIQUE ARAÚJO PEREIRA MACHADO

Professor PUCRS

Possui graduação em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (1997) e mestrado em Computação pela Universidade Federal do Rio Grande do Sul (2000). Atualmente é professor assistente na Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul. Tem experiência na área de Ciência da Computação, com ênfase em Linguagem Formais, Teoria da Computação e Linguagens de Programação, atuando principalmente nos seguintes temas: teoria dos autômatos, modelos de hipertexto, teoria das categorias, cursos hipermédia, programação de sistemas para Web, frameworks multiplataforma para dispositivos móveis. Atua como revisor técnico da Sagah.

Ementa da disciplina

Estudo de Arquitetura cliente-servidor para aplicações web SPAs (Single Page Applications). Estudo sobre frameworks cliente-side: React, Next.js, Redux, React Router, React Hook Form, Jest, Styled Components.

Professores

Desenvolvimento
Full-Stack



ANDRÉ LUIZ MENDES PEREIRA

Professor convidado

Fundador da Evolve, uma plataforma digital com propósito de mudar a clusterização do varejo, possui mais de 25 anos de experiência na área de desenvolvimento, atuando por 15 anos como CTO e COO. Lidera equipes de desenvolvimento ágeis multidisciplinares em multiplataformas, buscando a essência e resultados focados no Business Agility. Formado com honras ao mérito em Ciência da Computação, pós-graduado em Melhoria no Processos de Software, Arquitetura em Sistemas Distribuído e com MBA em Gestão Empresarial pela FGV. Agrega conhecimento nas diversas verticais na construção de softwares, dentre elas arquitetura de soluções e de software, Devops e Cloud. Contribuindo para que as empresas TI encontrem as melhores soluções tecnológicas.





Ementa da disciplina

Estudo de Arquitetura cliente-servidor para aplicações web SPAs (Single Page Applications). Estudo sobre frameworks client-side: React, Next.js, Redux, React Router, React Hook Form, Jest, Styled Components.



Arquitetura Client-Side

Por André Luiz Mendes Pereira - Aula 1



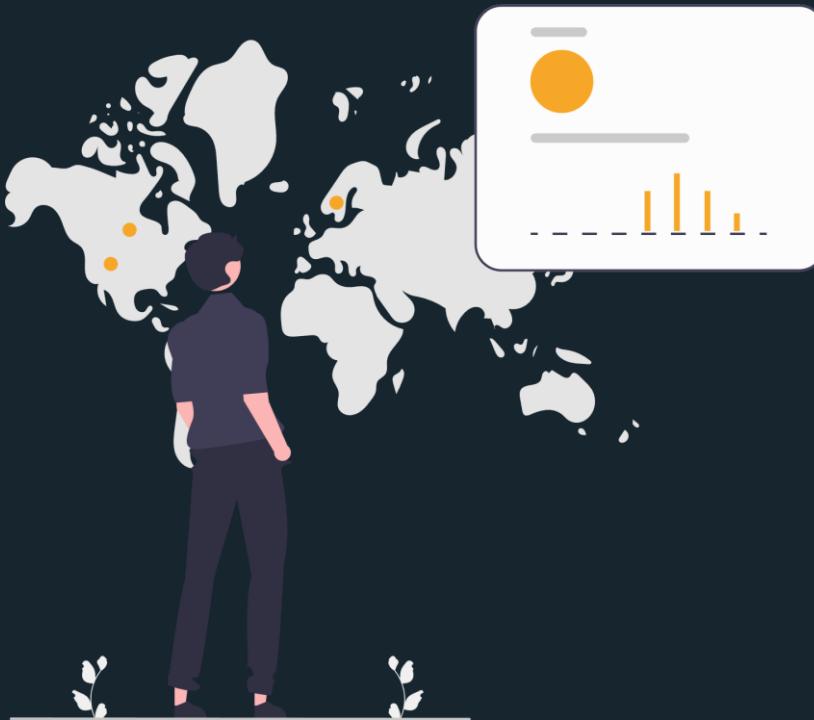
linkedin.com/in/andreluizmendespereira

Abertura

Desenvolvimento
Full-Stack



O contexto das nossas aulas será apresentar, conceitos e abordagens para gerar provocações que permitam estimular reflexões e aguçar o senso-crítico sobre abordagens client-side.



Entender a aplicação dos conceitos amplos da arquitetura client-side e não se apegar apenas em tecnologias específicas, que mudam a todo momento, permite ao profissional tomar melhores decisões em todos os ambientes e o diferenciar no mercado.

Tudo isto é essencial para o profissional full-stack!.

Conteúdo

Desenvolvimento
Full-Stack



1. Introdução
2. Fundamentos arquitetura client-side
3. Aprofundar na Arquitetura client-side
4. Tendências
5. Conclusão



Introdução



Arquiteturas modernas

Frontend

Páginas Web

Aplicativos móveis

Middlewares

Protocolos de rede

Redes

Backend

Serviços

Servidores

<https://marcomendes.online>

As arquiteturas modernas podem ser pensadas em 3 grandes partes

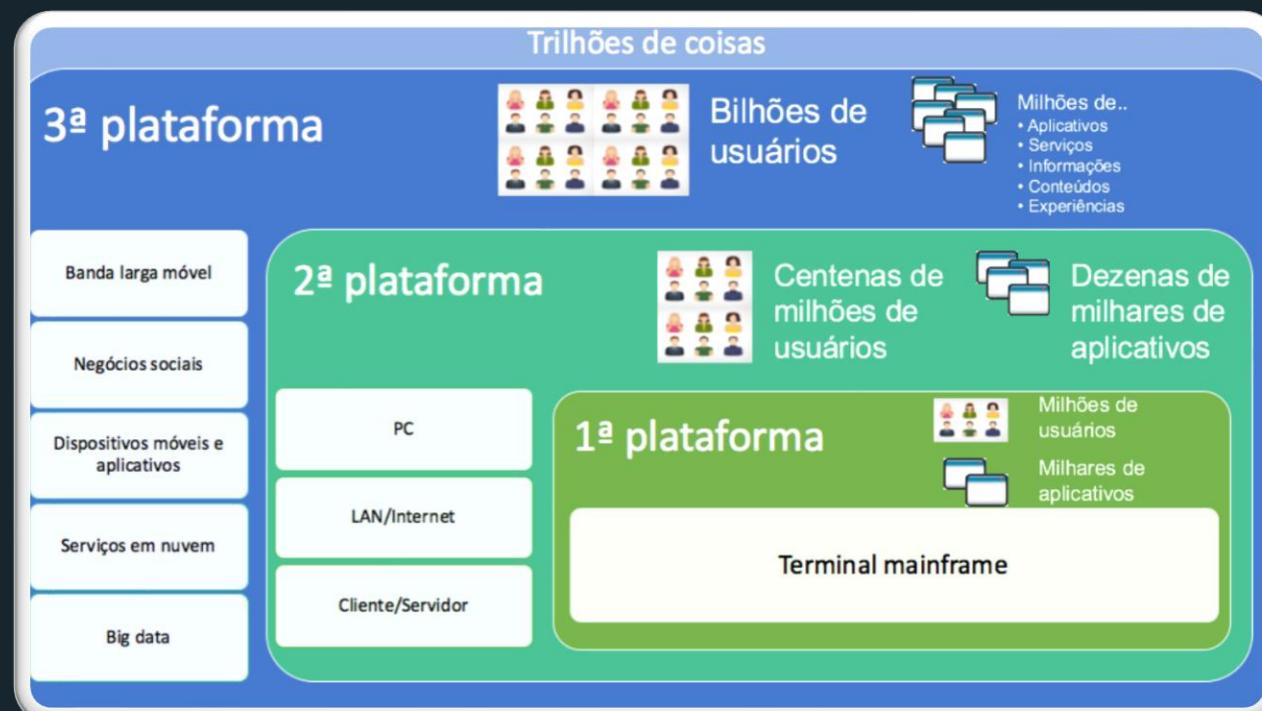


Nexus das Forças





A 3^a plataforma – A geração de uma nova plataforma arquitetural



Isto determina nova fase da arquitetura client-side, bem como, novos desafios e compromissos.



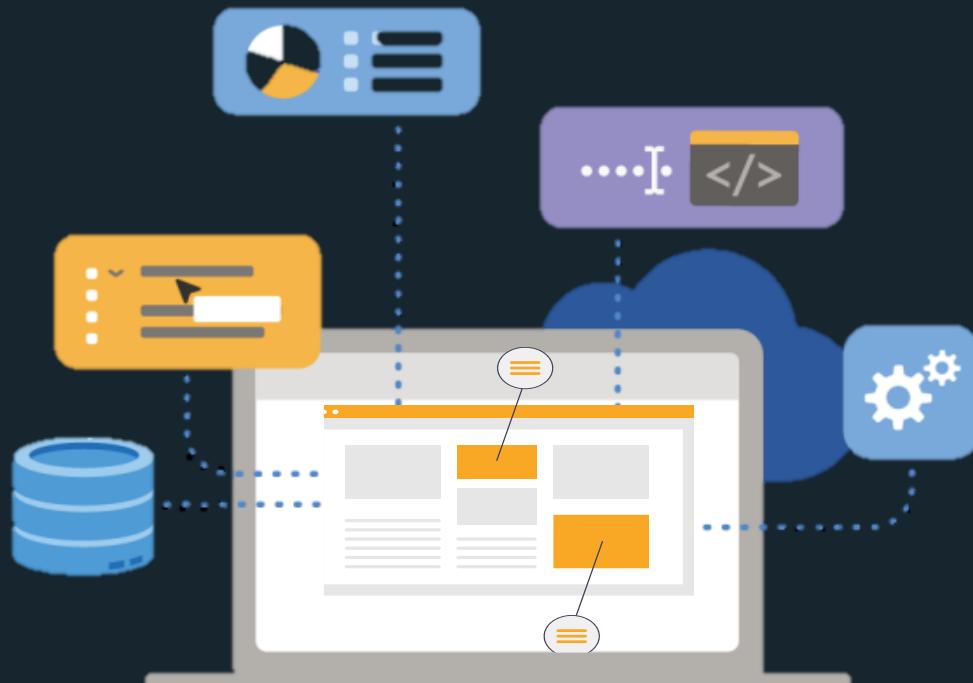
Sistemas distribuídos

Introdução aos Sistemas distribuídos

Sistemas de distribuídos



Um sistema distribuído é um conjunto de dispositivos e/ou serviços que independentes se apresentam a seus usuários, programas ou outros serviços como um sistema único e coerente utilizando-se de um sistema de conexão.

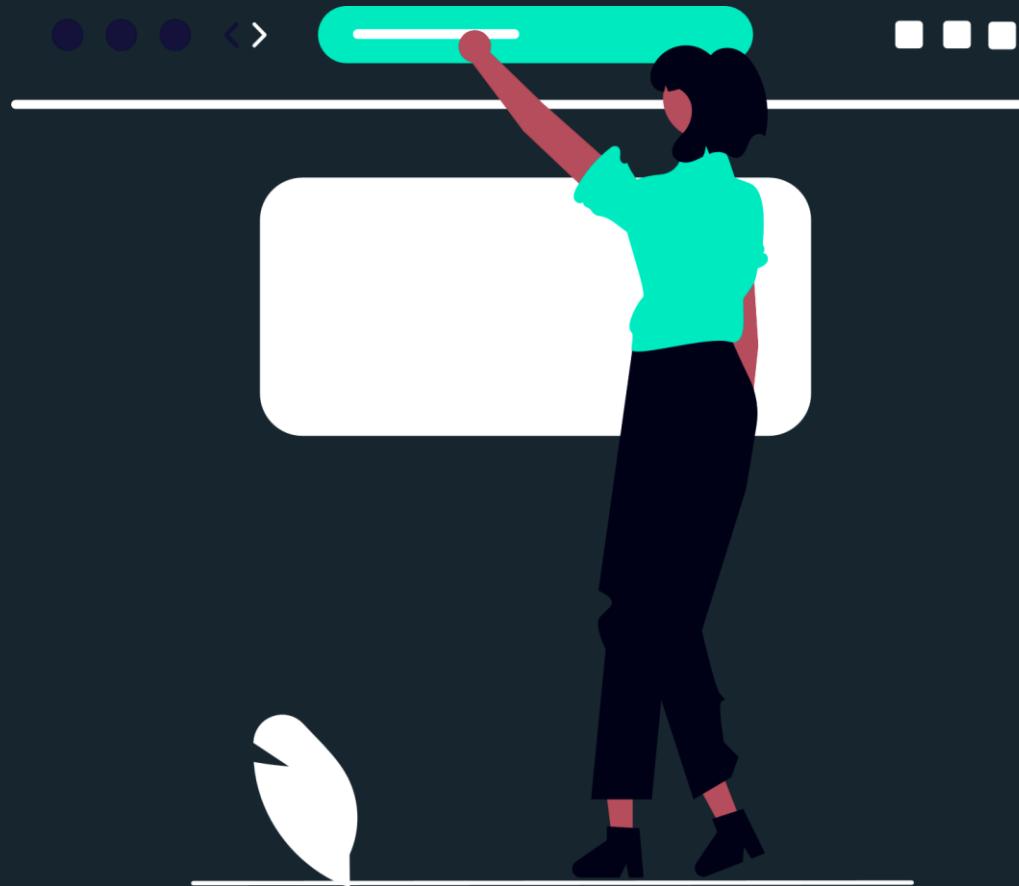


Sistemas de distribuídos



Uma “meta” importante no sistema distribuído é a transparência.

Esta transparência remete em ocultar do demandador (ex. usuário) a representação do dado, protocolos, autenticações entre serviços, localização, migração e relocação dos recursos, replicação de conteúdo, concorrências e falhas.



Sistemas de distribuídos



Reflexão ...

Então, para buscarmos minimizar os impactos em prol deste “ocultimos” para sistemas web, utilizar-se de boas estratégias como um bom design system, coerência no planejamento arquitetural (client-side e server-side) e o uso de ferramentas adequadas, é vital para o sucesso de projetos focados no front-end.





Arquitetura de Software

Conceitos de arquitetura de software



Conceito de Arquitetura de Software

A arquitetura de software é uma disciplina da engenharia de software que tem por objetivo suportar a tomada das decisões técnicas.

Alguns autores insistem em não conceituar pois ela depende de uma série de fatores. Inclusive, algumas referências tratam de forma incompleta.



Conceito de Arquitetura de Software

Meu conceito de arquitetura :

A arquitetura de software é uma estrutura complexa compreendendo decisões arquiteturais sobre elementos estruturais, software, características e modelagem que se relacionam em busca de gerar harmonia na construção de software levando em consideração principalmente seu ambiente e contexto.

Entende-se por contexto, toda a estrutura da empresa, cultura, habilidades dos desenvolvedores, cronograma etc.

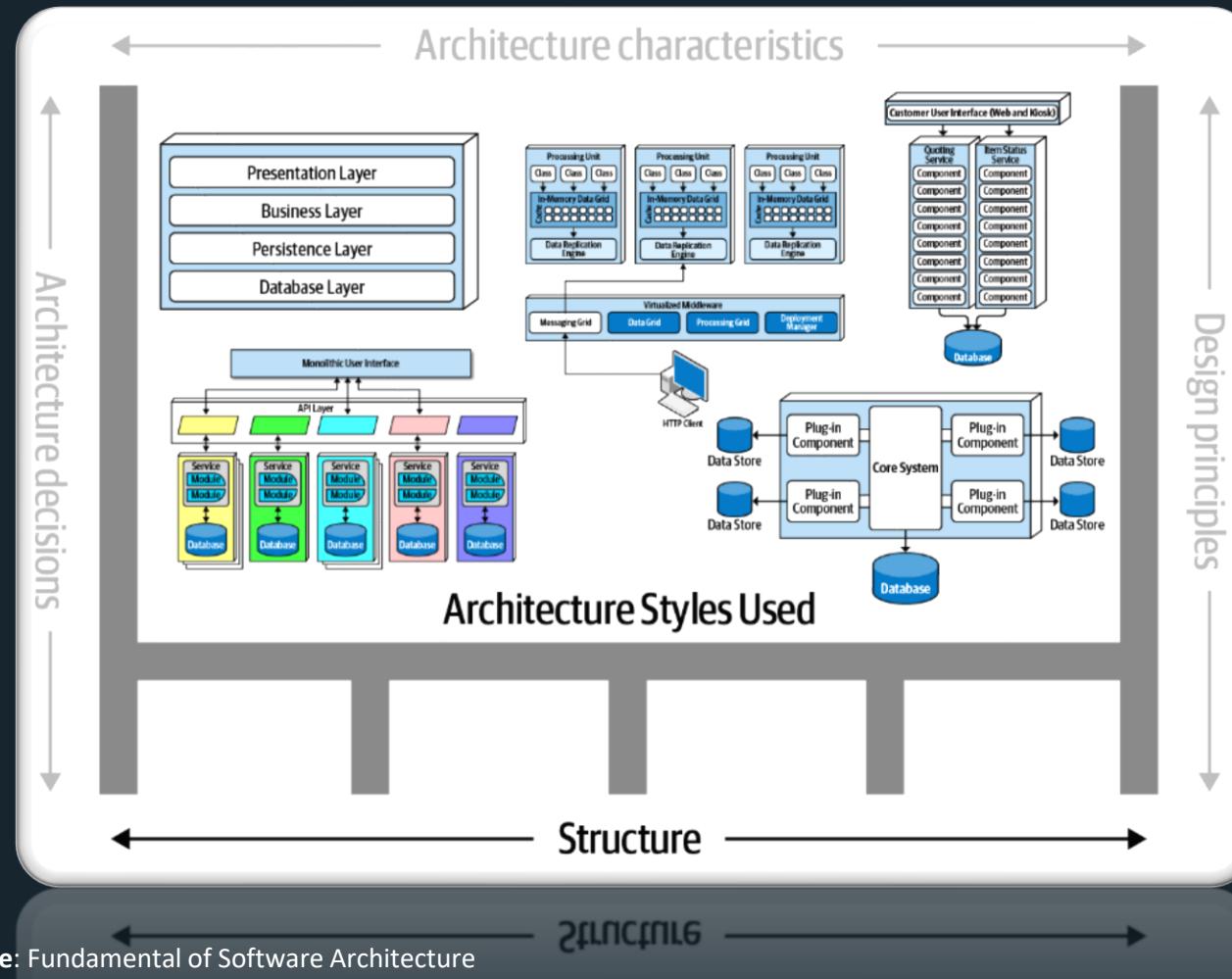


Arquitetura de software

Desenvolvimento
Full-Stack



Elementos da Arquitetura de Software



1. Princípios de Design
2. Características
3. Estrutura Arquitetural
4. Decisões Arquiteturais

Definição de arquitetura de software

Desenvolvimento
Full-Stack



Características ou Requisitos Arquiteturais

Operacional

Disponibilidade, desempenho, resiliência, segurança, robustez, escalabilidade.

Estrutural

Manutenibilidade, implantabilidade, configurabilidade, observabilidade, capacidade de atualização, performance, disponibilidade, testabilidade, interoperabilidade, escalabilidade, elasticidade, tolerância a falhas, etc

Transversal

Acessibilidade, armazenamento, autorização, privacidade, segurança, suportabilidade, usabilidade.

Decisões ou Padrões Arquiteturais

Padrões arquiteturais

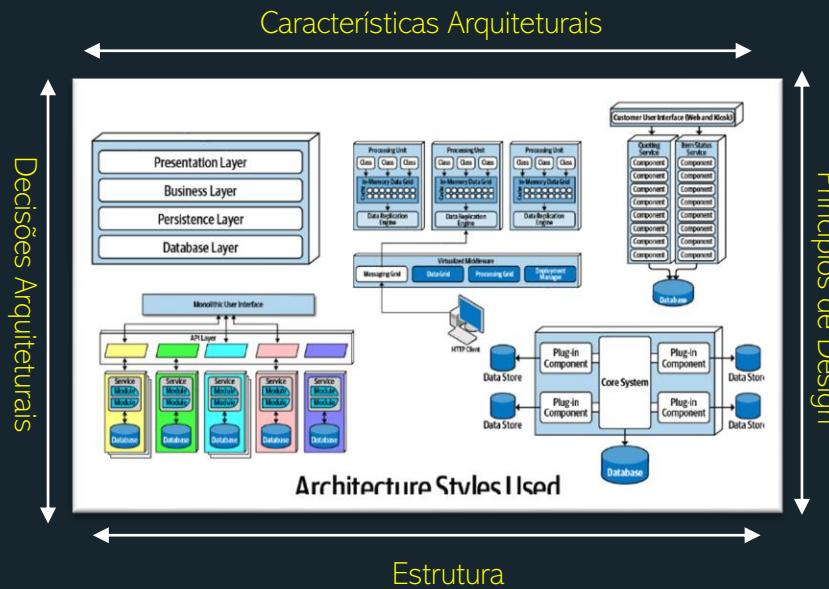
São estratégias de alto nível para ajudar a definir a responsabilidade dos componentes do sistema.

Exemplos de padrões arquiteturais.

CQRS

Onion Architecture

Hexagonal ou Adapters



Princípios de Design ou Modelagem

Abordagens de modelagem

São diretrizes para construção do sistema.

Exemplos de princípios de design

DDD (Domain-drive-design)

Estruturas ou Estilos Arquiteturais

Estilos arquiteturais

Estilo arquitetural é uma abordagem de como projetar e entregar uma aplicação. Podem trabalhar juntos em um mesmo projeto.

Exemplos de estilos arquiteturais.

Camadas

Microserviços

REST

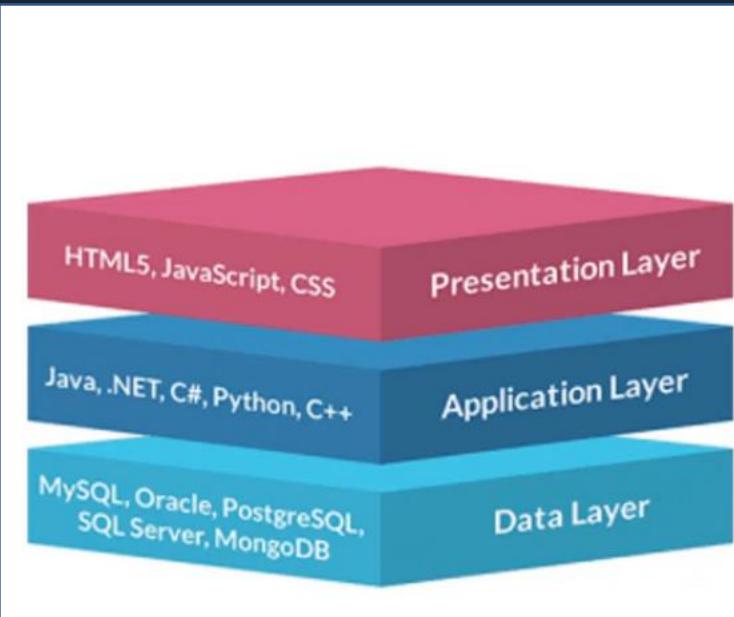
Monolítica

Arquitetura de software

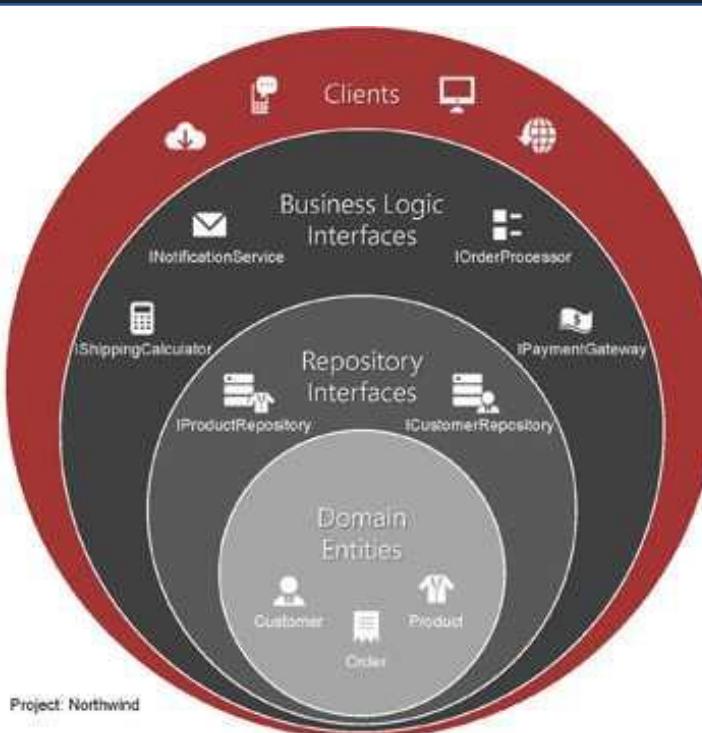
Desenvolvimento
Full-Stack



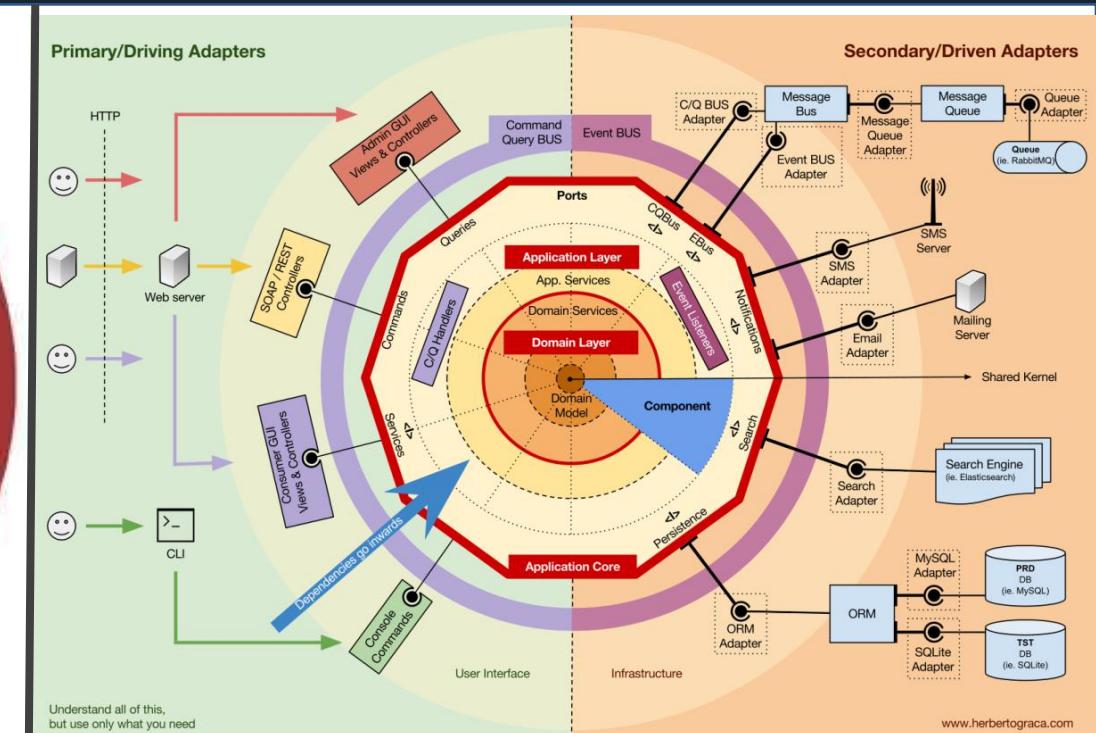
Enriquecendo – Exemplo de padrões



3-Tier Achitecture



Onion Architecture



Hexagonal Architecture



Reflexões ...

Qual arquitetura de software você deve implementar?

A resposta é “Depende”.

Não existe uma arquitetura única que funcione bem em todos os casos e em todos os momentos. Você tem que personalizar sua arquitetura para suas necessidades, aplicando os padrões que resolvem seus problemas e se adaptam melhor à sua situação.

Tenha em mente que o negócio evolui com o tempo.



Arquitetura de software

Desenvolvimento
Full-Stack



Reflexões ...

Agora, ao falarmos de Microserviços ou Microfrontends, sabemos que estamos tratando de um estilo arquitetural, e não de uma arquitetura. Ou seja, é uma parte das decisões arquiteturais. Depende de outros fatores como a estrutura de construção usando padrões arquiteturais adequados.

Técnicas como uso do DDD (Domain Driven Design) podem auxiliar tanto na modelagem da arquitetura quanto na definição de estilos e padrões, quanto nos requisitos arquiteturais. Como por exemplo, orquestrar micro frontends e micro serviços.





Fundamentos Arquitetura Client-side

Elementos

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



Dispositivos



Desktop e Notebooks



Celular e Tablets



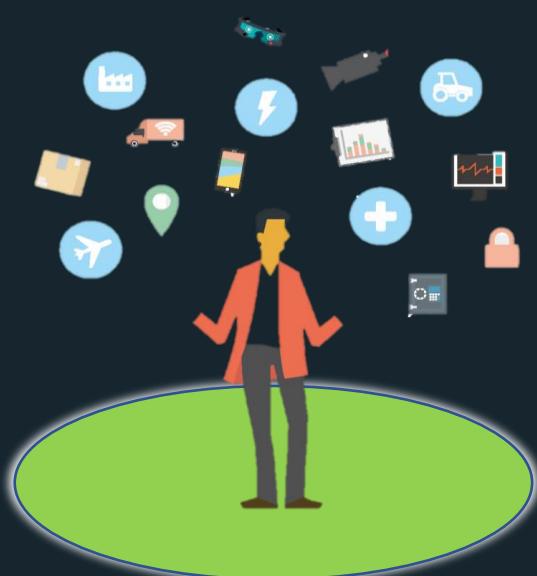
Assistentes Virtuais



SmartTv, TvBox
e Consoles Games



Vestíveis (Wearables)



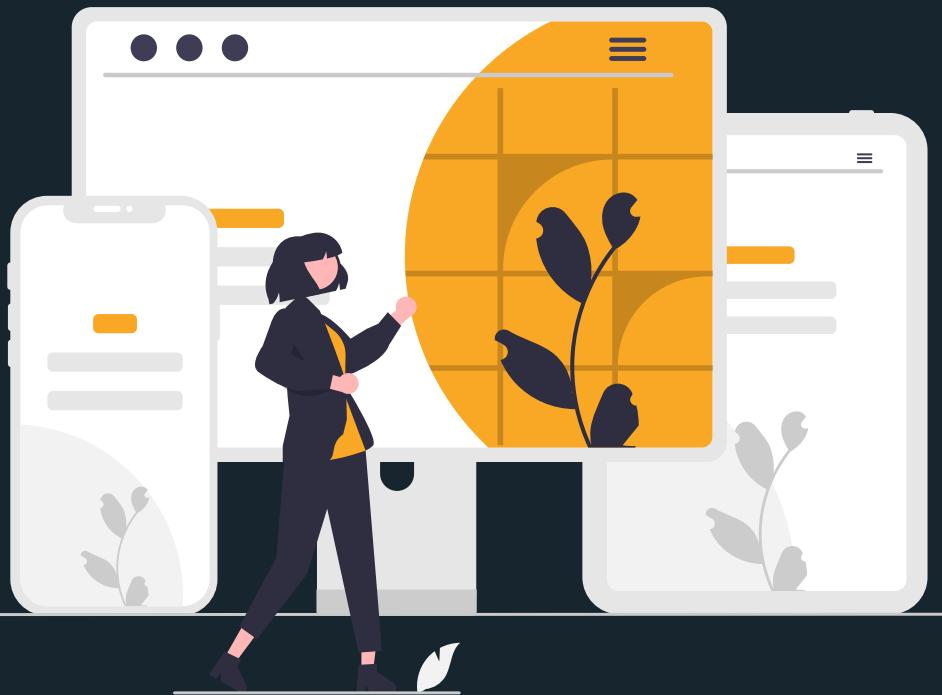
IoT - Internet das coisas

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



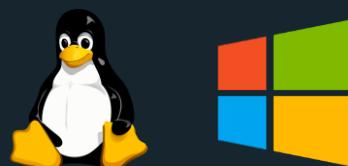
Plataformas



Browsers



Android e iOS



Linux e Windows



Mensageria



Design System

Design System envolve todo o contexto de estilos, padrões, componentes, ferramentas e bibliotecas que devem auxiliar equipes a projetar e construir um produto. Mantendo um padrão visual com base nas melhores práticas UX/UI auxiliando inclusive requisitos arquiteturais como usabilidade e acessibilidade.

Ao compartilhar artefatos de design, evitamos "reinventar a roda" permitindo a reutilização de código de design gerando maior produtividade.



Exemplos de sistema de Design
Material.io – Google
ADS – Atlassian Design System



Fundamentos Arquitetura Client-side

Resumo sobre Fundamentos Web

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



Introdução



A história e idealização da Internet, por Vint Cerf. Considerado um dos “pais da internet”



URI, URL e URN

URI – Uniform Resource Identifier:

Padrão para o endereçamento de recursos disponíveis na web que engloba conceitos de:

Tipos de URI

URL (Uniform Resource Locator) e URN (Uniform Resource Name)

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



URI, URL e URN

URL (Uniform Resource Locator): refere-se a um subconjunto de URLs que serve para referenciar um recurso e sua localização na rede, normalmente a internet

esquema://nome-do-servidor:porta/caminho/consulta?chave=valor&chave2=valor#fragmento

Autoridade - Obrigatório

Opcional

Estrutura de um URL

Esquema – identifica a forma de interação entre um cliente e servidor, como por exemplo: http, https, ftp, entre outros

Nome-do-servidor – nome ou número IP onde se encontra a aplicação servidor.

Porta – identifica a porta TCP/IP associada ao servidor. No caso padrão do HTTP, a porta é 80 e pode ser omitida

Caminho – indica o local exato onde o recurso se encontra

Consulta – dados não hierárquicos que detalham uma consulta normalmente sob chaves e valores.

Fragmento – identifica uma seção no recurso.



Introdução – URI, URL e URN

URN (Uniform Resource Name): é um padrão de URI que não indica qual protocolo deve ser utilizado para localizar e acessar um recurso. O objetivo do URN é permitir a separação entre a identificação (nome único) e localização (identificado). Permitindo acesso a uma identidade do dado mesmo com mudança da localização do servidor.

Cada instituição pode propor o formato da URN. Inclusive é importante observar que algumas empresas adotam conceitos de URN, como [Linkedin](#) em retornos json para gerar mapeamento a identidade de um campo.



Legenda:

NID : Namespace Identifier

NSS : Namespace Specific String

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



URI, URL e URN

URN - Exemplo Linkedin

Solicitando Informação

https

cópia de

```
GET https://api.linkedin.com/v2/shares/1234?projection=(id,owner)
```

JSON

cópia de

```
{
  "id": "1234",
  "owner": "urn:li:person:-f_Ut43FoQ"
}
```

Desserializando e buscando pela identidade recebida

https

cópia de

```
GET https://api.linkedin.com/v2/people/id=-f_Ut43FoQ?projection=(id,localizedFirstName,localizedLastName)
```

Fonte: <https://developer.linkedin.com>

<https://learn.microsoft.com/en-us/linkedin/shared/references/v2/urn-namespace?context=linkedin%2Fcontext>

<https://learn.microsoft.com/en-us/linkedin/shared/api-guide/concepts/urns>

Fundamentos Arquitetura Client-side

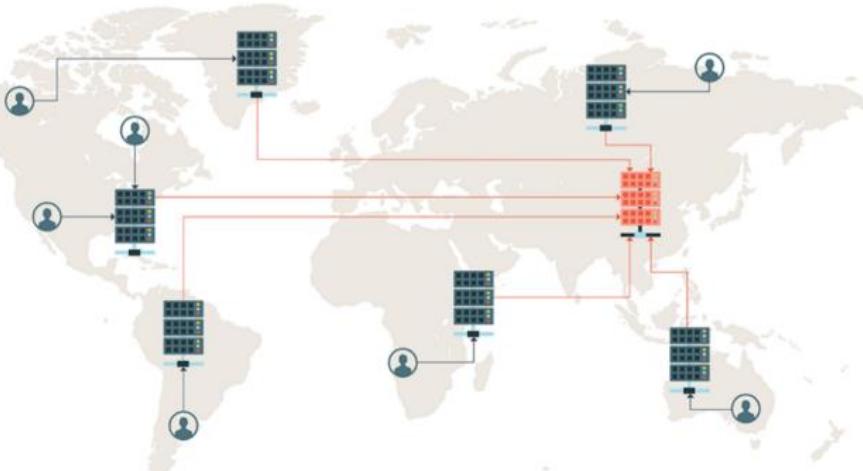
Desenvolvimento
Full-Stack



CDN (Content Delivery Network)

CONTENT DELIVERY NETWORK

USER CDN SERVER ORIGIN SERVER



<https://kickstartsolutions.in/benefits-of-content-delivery-network-cdn/>

CDN ou redes de distribuição de conteúdo são redes de servidores otimizados para entrega de conteúdo estáticos ou dinâmicos mais rapidamente ao usuário.

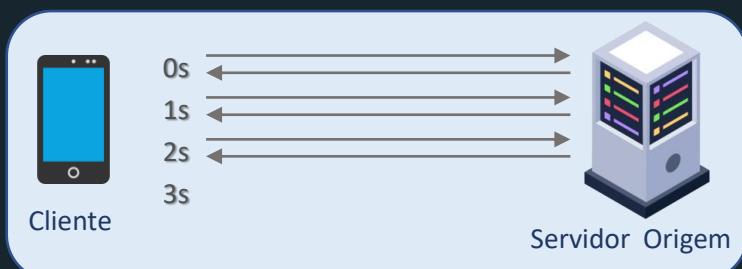
Armazenam comumente conteúdos cacheáveis (mas pode não ser), pois relativamente o conteúdo chegará mais rápido, ou seja, melhorará a latência da resposta.

Fundamentos Arquitetura Client-side

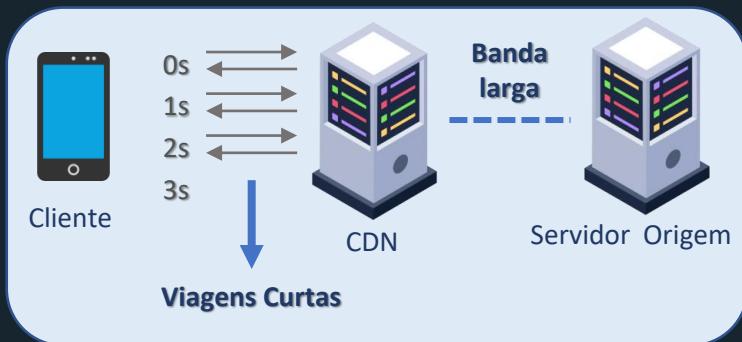
Desenvolvimento
Full-Stack



CDN (Content Delivery Network)



Quando um CDN é utilizado, uma nova conexão é estabelecida entre um cliente e um servidor CDN próximo. Este verificará conteúdo e enviará ao cliente. Caso o conteúdo não seja encontrado ou estiver expirado (time to live – TTL) solicita ao servidor nova versão.



Tipos de Conteúdos

Estáticos:
mudam raramente ou nunca
Conteúdo: imagens, vídeos, pdfs, bibliotecas js, css, html, etc.
TTL exemplo: 6 meses 1 ano

Dinâmicos:
Mudam com frequência,
Conteúdo como uma resposta de API, pagina home, um feed.
TTL exemplo: 5 segundos

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



Protocolos

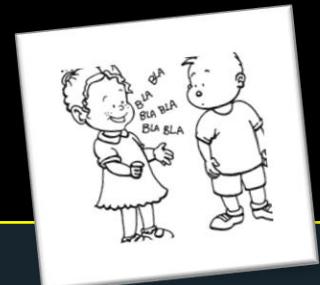
Protocolo é o conjunto de regras e padrões de sintaxe que permitem que dois sistemas computacionais conversem entre si.

O conjunto de protocolos pode ser visto em camadas (interface, rede, transporte e aplicação), onde uma confia na outra. Existem diversos tipos de protocolos, em diversas camadas, voltados para “conversar” assuntos específicos.



Camadas:

1. Aplicação: WWW, HTTP, SMTP, Telnet, FTP, SSH, NNTP, RDP, IRC, POP3, IMAP, SIP, DNS, PING;
2. Transporte: TCP, UDP, RTP, DCCP, SCTP;
3. Rede: IPv4, IPv6, IPsec, ICMP;
4. Interface: Ethernet, Modem, PPP, FDDI.



Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



HTTP (Protocolo de Transferência de Hipertexto)

```
GET /index.html HTTP/1.1
Host: www.exemplo.com
HTTP/1.1 200 OK
Date: Mon, 23 May 2016 22:38:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytesContent-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

Solicitação do cliente – Invocando método GET

É o protocolo fundamental para suportar a requisição a qualquer objeto Web, como uma página de um portal ou um vídeo no YouTube

Uma requisição HTTP começa com uma solicitação do cliente e uma resposta do servidor (arquitetura clientxservidor)..,

Ela possui diversos métodos como : GET, POST,PUT ou DELETE para consumir serviços WEB, falaremos mais adiante quando aprofundarmos em APIs

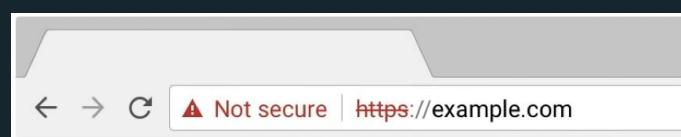
O protocolo HTTP é mantido
pelo World Wide
Consortium (W3C)

Fundamentos Arquitetura Client-side

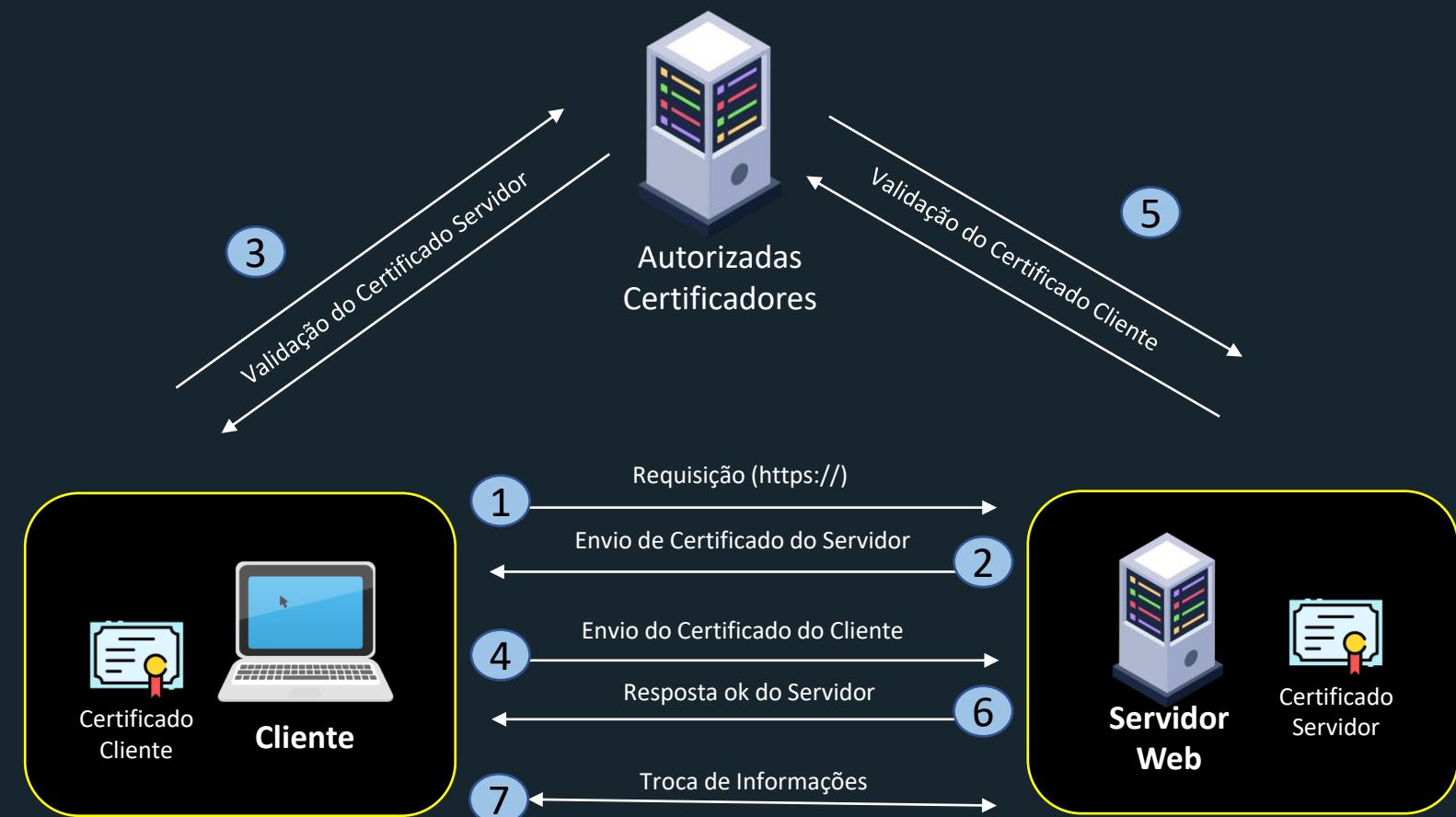
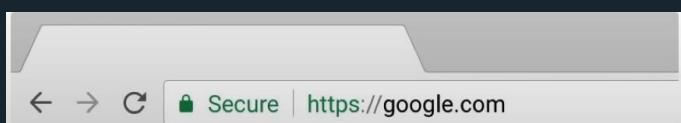
Desenvolvimento
Full-Stack



HTTPS (Protocolo de Transferência de Hipertexto Seguro)



VS





Fundamentos Arquitetura Client-side

Arquitetura Client-Side



Arquiteturais de Frontend

Diversos aspectos deverão ser levados em consideração em seu projeto frontend



Contexto
Cultura Organizacional
Mercado Alvo
Personas
Tecnologias disponíveis
Integrações externas

Sistema
Abordagens arquiteturais
Ambiente de desenvolvimento
Processo desenvolvimento

Componentes
Marcação HTML
Código CSS
JavaScript
Segurança e Infraestrutura



Tecnologias Web – A Santíssima Trindade

← CSS

Guias de Estilos | Frameworks CSS
Pré-processadores (Sass) | Layouts
Padrões CSS (CSS Modules)
Responsividade

Estilos visuais

Seu objetivo é promover a separação entre o formato e conteúdo, não misturando no html questões voltadas ao estilo visual do conteúdo.

A boa prática deste recurso ajuda otimizar a padronização estética da página, desenvolvimento e manutenibilidade e escalabilidade de componentes



→ JavaScript

Scripts | TypeScripts | Frameworks JS

Interação e manipulação DOM

Linguagem que permite manipulação do HTML, comunicação com servidores, persistência de dados (lokais) e interação com APIs internas do browser ou externas

Marcação HTML

HTML Semântica | SEO | Microdados
Acessibilidade WAI-ARIA

Estrutura da página

Linguagem de marcação interpretada pelos browsers.

Os hipertextos são documentos que utilizam hiperlinks para outros documentos relacionados, daí o nome Web (teia).
O HTTP é o protocolo destinado a trafegar o html,

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



JavaScript

JavaScript é uma linguagem de script orientada a objetos e plataforma cruzada usada para tornar as páginas da Web interativas (por exemplo, com animações complexas, botões clicáveis, menus pop-up etc.).

Há também versões mais avançadas do lado do servidor do JavaScript, que permitem adicionar mais funcionalidades a um site do que baixar arquivos (como colaboração em tempo real entre vários computadores).

Referência: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide>

JS

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



JavaScript

- No lado cliente existe inúmeras plataformas e frameworks que utilizam Javascript. Backbone, Knockout, Angular, React, Vue e Ember são exemplos de framework Javascript. No caso do Angular ele é um framework Javascript que utiliza como linguagem preferencial o Typescript.
- O lado do servidor o JavaScript fornece extensões que permitem uma aplicação comunique-se, por exemplo, com um banco de dados ou forneça serviços web. São linguagens Javascript do lado servidor Node.JS e Deno. O Deno inclusive foi desenvolvido pelo criador do Node, Ryan Dahl, em resposta para lacunas de segurança e escala do próprio NodeJS. Existe um vídeo muito interessante em que ele fala destas lacunas, deixarei abaixo o link como referência.

Video Ryan Dahl: <https://www.youtube.com/watch?v=M3BM9TB-8yA>

JS



TypeScript

TypeScript é uma linguagem de programação, criada pela Microsoft, que adiciona recursos de tipagem estática ao Javascript. O TypeScript é considerado uma linguagem superset do JavaScript, o que significa que qualquer código JavaScript válido também é código TypeScript válido.

O TypeScript verifica se há erros em um programa antes da execução tornando-o um *verificador de tipo estático*.



Fonte: <https://www.typescriptlang.org/docs/>

Fundamentos Arquitetura Client-side

Desenvolvimento
Full-Stack



TypeScript - Vantagens

Tipagem Estática

```
const user = {  
  firstName: "Angela",  
  lastName: "Davis",  
  role: "Professor",  
}  
  
console.log(user.name)  
  
Property 'name' does not exist on type '{ firstName: string;  
lastName: string; role: string; }'.
```

Autocompletar em IDEs

```
import express from "express"  
const app = express()  
  
app.get("/", function (req, res) {  
  res.send("Hello World")  
})  
  
app.listen(3001, function () {  
  console.log("App is running on port 3001")  
})
```

Códigos Tipados

```
function calculateArea(radius: number): number {  
  return Math.PI * radius * radius;  
}  
  
const radius: number = 5;  
const area: number = calculateArea(radius);  
console.log(`A área é: ${area}`);
```

Recursos de Orientação à objeto

```
interface User {  
  id: number  
  firstName: string  
  lastName: string  
  role: string  
}  
  
function updateUser(id: number, update: Partial<User>) {  
  const user = getUser(id)  
  const newUser = { ...user, ...update }  
  saveUser(id, newUser)  
}
```



React arquivos tsx

```
import * as React from "react";  
  
interface UserThumbnailProps {  
  img: string;  
  alt: string;  
  url: string;  
}  
  
export const UserThumbnail = (props: UserThumbnailProps) =>  
  <a href={props.url}>  
    <img src={props.img} alt={props.alt} />  
  </a>
```



JavaScript

```
function calculateArea(radius) {  
  return Math.PI * radius * radius;  
}  
  
const radius = 5;  
const area = calculateArea(radius);  
console.log(`A área é: ${area}`);
```



Tecnologias Web – HTML - Cabeçalho

```
<!DOCTYPE html>
<html>
<head> ... </head>
<body> ... </body>
</html>
```



Estrutura básica

Estrutura HTML

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <title>Página de Exemplo</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta http-equiv="expires" content="Mon, 22 jul2006 11:12:01 GMT" />
    <meta http-equiv="refresh" content="15;url=http://www.site.com" />
    <meta http-equiv="cache-control" content="no-cache" />
    <meta http-equiv="cache-control" content="no-cache" />
    <meta name="description" content="PUC RS Web Site">
    <meta name="author" content="Andre">
    <meta name="keyword" content="html, web, css">
    <link rel="stylesheet" href="style.css">
</head>
<body> ... </body>
</html>
```



MetaTags



Tecnologias Web – Protocolo OpenGraph

OpenGraph

Quando você tenta compartilhar um conteúdo no whatsapp ou redes sociais, costuma aparecer uma imagem acompanhada das informações referentes do site ao invés de apenas o título e a descrição.

Isto acontece pois o site utilizou o protocolo OpenGraph.

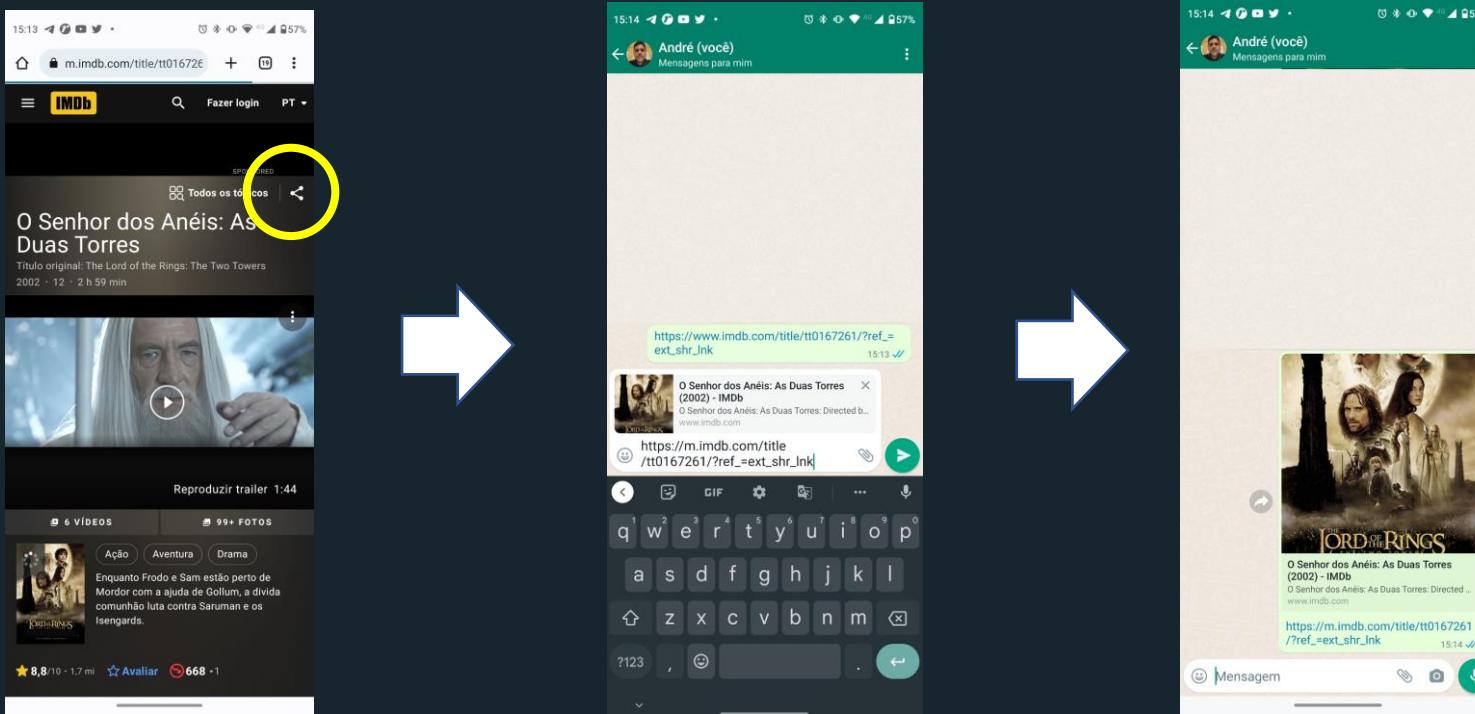
Arquitetura Client-Side

Desenvolvimento
Full-Stack



Tecnologias Web – HTML – Protocolo OpenGraph

Copia link do site e colar no whatsapp as informações serão renderizadas no servidor usando o protocolo OpenGraph determinado nas metatags.





Tecnologias Web – HTML - Cabeçalho

Estrutura HTML

```
<!DOCTYPE html>
<html lang="pt-br">
<head>

    <title>O Senhor dos Anéis: As Duas Torres (2002) - IMDb</title>
    <meta name="description" content="O Senhor dos Anéis: As Duas Torres: Dirigido por Peter Jackson.
Com Bruce Allpress, Sean Astin, John Bach, Sala Baker. Enquanto Frodo e Sam estão perto de Mordor com a ajuda de Gollum,
a divida comunhão luta contra Saruman e os Isengards." data-id="main">
    <meta name="author" content="IMDb" />
    <meta name="keywords" content="Reviews, Showtimes, DVDs, Photos, User Ratings, Synopsis, Trailers, Credits" />
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta http-equiv="expires" content="Mon, 22 jul2006 11:12:01 GMT" />
    <meta http-equiv="refresh" content="15;url=http://www.site.com" />
    <meta http-equiv="cache-control" content="no-cache" />
    <meta property="og:url" content="https://www.imdb.com/title/tt0167261/">
    <meta property="og:site_name" content="IMDb" />
    <meta property="og:title" content="The Lord of the Rings: The Two Towers (2002) - IMDb" />
    <meta property="og:type" content="video.movie" />
    <meta property="og:image" content="https://m.media-amazon.com/images/M/MV5BZGMxZTdjZmYtMmE2Ni00ZTdkLWI5NTgtNj
lmMjBiNzU2MmI5XkEyXkFqcGdeQXVyNjU0OTQ0OTY@._V1_FMjpg_UX1000_.jpg" />
    <meta property="og:image:height" content="1556.0165975103735" />
    <meta property="og:image:width" content="1000" />
    <meta name="description" content="O Senhor dos Anéis: As Duas Torres: Dirigido por Peter Jackson.
Com Bruce Allpress, Sean Astin, John Bach, Sala Baker. Enquanto Frodo e Sam estão perto de Mordor com a ajuda de Gollum,
a divida comunhão luta contra Saruman e os Isengards." data-id="main">
    <link rel="stylesheet" href="style.css" />

</head>
<body> ... </body>
</html>
```



Tecnologias Web – HTML – Protocolo OpenGraph

O protocolo Open Graph permite que qualquer página da Web se torne um objeto rico em um gráfico social. Para transformar suas páginas da web em objetos gráficos, você precisa adicionar metadados deste protocolo à sua página.

Básicos

`og:title` o título do seu objeto

`og:type` tipo do seu objeto, dependendo do tipo pode ser necessário outras propriedades

`og:url` a URL da imagem que representará o objeto

Opcionais

`og:image:secure_url-` um URL alternativo a ser usado se a página da Web exigir HTTPS

`og:image:type` Um tipo MIME para esta imagem.

`og:image:width` número de pixels de largura.

`og:image:height` número de pixels de altura.



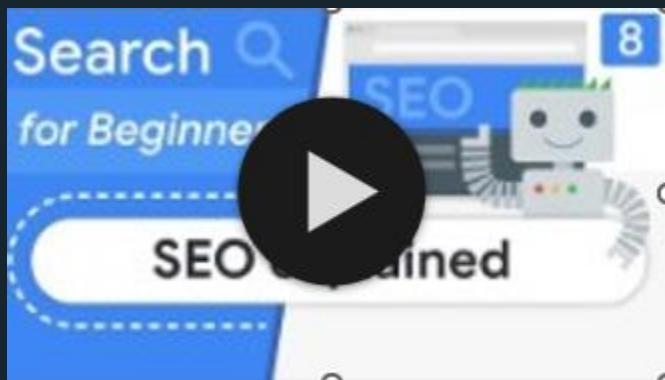
Tecnologias Web – SEO

Para seu site ou aplicação ser notada por mecanismos de busca, precisa-se cumprir alguns requisitos padrões. O conjunto de boas práticas e critérios são descritos em um processo chamado SEO.

Search Engine Optimization (SEO) nada mais é que um guia de otimização para mecanismos de pesquisa.

Ao consultar algum assunto os mecanismos de busca avaliam, inclusive darão relevância, para aqueles sites que de forma geral contenham:

1. Bom design (estrutura de títulos, conteúdos incluindo mídias)
2. Que contenham Keywords,
3. Bom mapeamento de conteúdo (metatags, boas práticas de formatação etc)
4. Boa estrutura de html e arquitetura, principalmente que utilizam boas práticas de otimização
5. Mapeadas as suas análises de tráfego (analytics),



Arquitetura Client-Side

Desenvolvimento
Full-Stack



Tecnologias Web – CSS – Boas Práticas

1. Prefira uso de nomes evite Ids
2. Defina e organize apropriadamente os componentes
3. Use namespaces de classes de forma consistente (ex myapp-Header-link)
4. Mantenha coerência entre mapeamento de nomes de arquivos e namespaces
5. Evite vazamento estilos fora dos componentes.
6. Evite vazamento de estilos dentro dos componentes.
7. Faça uso de pré-processadores (ex SASS)
8. Respeite limites dos componentes e aprenda padrões e os utilize (OOCSS, BEM, etc)

```
ui/
  layout/
    |--- Header.js          // component code
    |--- Header.scss         // component styles
    |--- Header.spec.js      // component-specific unit tests
    |--- Header.fixtures.json // any mock data the component tests might need
  utils/
    |--- Button.md           // usage documentation for the component
    |--- Button.js            // ...and so on, you get the idea
    |--- Button.scss
```

2

```
<div class="myapp-Header">...</div>
```

4

```
head|
```

```
Header.js myapp/ui/layout
```

```
Header.scss myapp/ui/layout
```

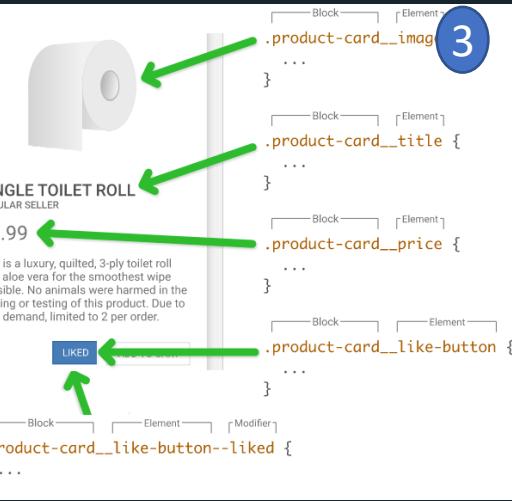
```
Header.spec.js myapp/ui/layout
```

```
Header.fixtures.json myapp/ui/layout
```

3

```
.myapp-Header {  
  background: black;  
  color: white;  
}  
  
&-link {  
  color: blue;  
}  
  
&-signup {  
  border: 1px solid gray;  
}
```

5



3

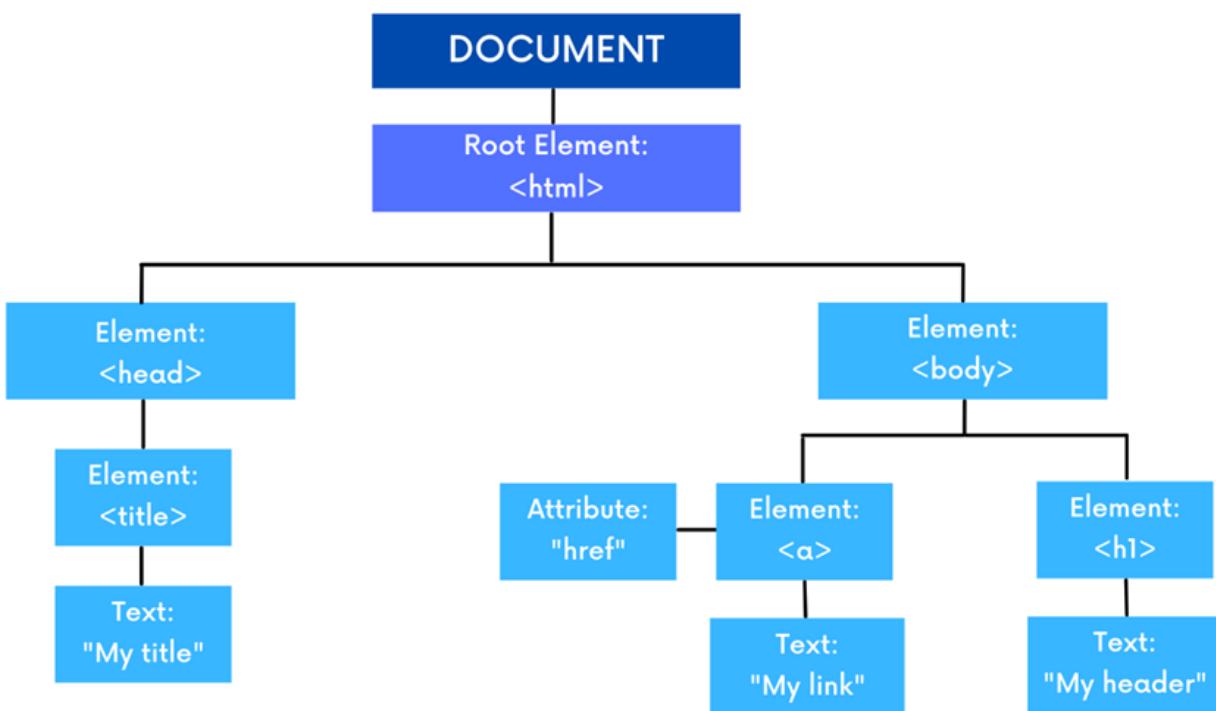
```
.myapp-Button {  
  @extend .myapp-utils-button;  
}
```

8

Arquitetura Client-Side



Tecnologias Web – DOM (Document Object Model)



Quando a página web é carregada, o navegador cria um DOM (Modelo de objeto do documento),
Ou seja, criar um objeto “document” para ser manipulado pelo Javascript.

Ele é construído como uma árvore de objetos.

Com o modelo de objeto, o JavaScript obtém todo o poder necessário para criar HTML dinâmico. Ele permite ao Javascript criar, alterar e excluir: elementos, atributos, estilos e principalmente eventos.



Arquitetura Client-Side

Serviços na Web

Serviços na Web

Desenvolvimento
Full-Stack



Introdução – Evolução

Necessidade de TI mais moderna e eficiente
Além da fronteira dos processos e departamentos
Precisava integrar sistemas diferentes (internos e com parceiros ou fornecedores)

Surge conceito de Serviços Digitais para:
Compartilhamento recursos
Interoperabilidade
Reutilização
Acoplamento
Otimização
(inclusive para componentes visuais)

Transformação digital exponenciada.
Surge necessidade de consumir serviços mais específicos até funcionalidades específicas.
Atender não só o SOA mas outras abordagens arquiteturais.



Empresas Tradicionais
Monolitos



SOA
Arquitetura Orientada a Serviços
Serviços



Transformação Digital
WebServices, Microserviços, Eventos

Evolução



Serviços

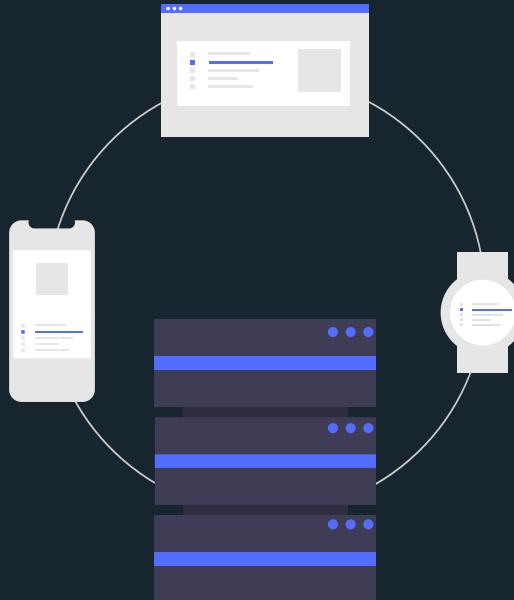
Mas o que seriam serviços ?

Serviços web : são recursos de software que disponibilizam funcionalidades através da web. Imagine um serviço web como uma plataforma que oferece um conjunto específico de funcionalidades que podem ser acessadas por outras aplicações. Essas funcionalidades são expostas através de uma interface de programação de aplicativos (API) que permite que outras aplicações se comuniquem e interajam com o serviço.





APIs – Interface de Programação de Aplicações



Webservices e APIs REST

Web Service é uma categoria específica de APIs que segue padrões e protocolos específicos para permitir a interoperabilidade entre sistemas distribuídos, enquanto APIs são interfaces mais amplas que podem ser usadas para diferentes propósitos e contextos de integração e interação de software (inclusive acesso a recursos dos dispositivos).



APIs Retornos

SOAP - XML

Fazendo uma requisição

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:prod="http://example.com/ProdutoService">
  <soap:Header/>
  <soap:Body>
    <prod:ConsultaProdutoRequest>
      | <prod:ProdutoID>12345</prod:ProdutoID>
    </prod:ConsultaProdutoRequest>
  </soap:Body>
</soap:Envelope>
```

Retorno - xml

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:prod="http://example.com/ProdutoService">
  <soap:Header/>
  <soap:Body>
    <prod:ConsultaProdutoResponse>
      | <prod:Produto>
      |   <prod:ProdutoID>12345</prod:ProdutoID>
      |   <prod:Nome>Camiseta</prod:Nome>
      |   <prod:Preco>29.99</prod:Preco>
      |   <prod:Descricao>Camiseta de algodão</prod:Descricao>
      | </prod:Produto>
    </prod:ConsultaProdutoResponse>
  </soap:Body>
</soap:Envelope>
```

XMLs possuem mecanismos de DTD incorporados que permitem estruturar seus elementos e atributos. Também possuem mecanismos para lidar com casos de exceção e erros, tornando-o mais adequado para ambientes em que a confiabilidade e a robustez são fundamentais

REST - JSON

Fazendo uma requisição

```
GET https://exemploclientside/produto?id=12345
```

Retorno - JSON

```
{
  "Produto": {
    "ProdutoID": "12345",
    "Nome": "Camiseta",
    "Preco": 29.99,
    "Descricao": "Camiseta de algodão"
  }
}
```

Arquivos JSON utilizam 30% menos de banda de passagem que arquivos XML. Isso o leva a ser o formato mais recomendável para a maior parte dos requisitos de troca de informações em aplicações Web



API REST – Consumir serviços usando métodos do HTTP

Métodos comumente utilizados :

1. **GET** = solicitar dados ao servidor
2. **POST** = enviar dados para processamento servidor. Muito usado para submeter formulários para que dados sejam inseridos no banco de dados
3. **PUT** = edita as informações de um determinado recurso Web
4. **DELETE** = remove determinado recurso Web

```
GET /index.html HTTP/1.1
Host: www.exemplo.com
HTTP/1.1 200 OK
Date: Mon, 23 May 2016 22:38:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytesContent-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

Solicitação do cliente – Invocando método GET

Os métodos HTTP mencionados acima trata de mudança de estado de recursos web. Mas fazendo uma analogia didática e simples mas superficial com CRUD teríamos

1. Create -> POST
2. Read -> GET
3. Update -> PUT
4. Delete -> DELETE



API REST – Boas práticas

1. Organize APIs ao longo de recursos (substantivos e não verbos -Ex. /filme/id e não /filme/consultar/id)
2. Padronize as suas APIs = adote uma convenção consistente. Uma dica seria usar substantivos no plural para URLs . Ex. /filmes/ator/id
3. Evite APIs anêmicas. = aquela que não traz o dado representativo. Lembre-se o objetivo do REST é mapear entidades de negócios e não de tabelas no banco de dados.
4. Crie APIs simples. Evite criar URLs de recursos mais complexos do que coleção/item/coleção
5. Use protocolo HTTPS/SSL.
6. Versione suas APIs
7. Estabeleça paginação para coleções com grandes volumes de dados
8. Use corretamente os códigos de retorno HTTP. Já vi *load balancer* da AWS derrubando tasks de serviços imaginando ser um ataque DDOS pelo uso incorreto de responses code.

Para mais informações :

<https://martinfowler.com/articles/richardsonMaturityModel.html>

<https://developer.marvel.com/docs>



Conclusão

Embora o SOA tenha enfrentado críticas na sua complexidade de implementação, com SOAP, como abordagem arquitetônica, foi extremamente relevante para as modelagens de arquitetura recentes e evolução tecnologia dos sistemas atuais.

Ele continua sendo uma abordagem relevante e amplamente adotada para adaptação e integração de sistemas tradicionais para web e no desenvolvimento de soluções distribuídas.

Uma evolução do SOA são a arquiteturas de micro serviços e destinada a eventos.





Reflexões ...

O conceito de serviços web para o profissional Frontend é essencial pois grande parte do trabalho junto ao client-side passará por algum tipo de consumo de serviços.

Seja por serviços de core do produto, seja para integrar com outras funcionalidades (como Google Maps, por exemplo), seja para acessar recursos dos dispositivos (utilizando de APIs Browsers).

Com a evolução e complexidade dos projetos, a arquitetura cliente-side muitas vezes precisará estar preparada para orquestrar esta diversidade de serviços para gerar informações ou afetar a experiência do usuário.





Aprofundar na Arquitetura Client-side

Contexto e Complexidade

Contexto e Complexidade

Desenvolvimento
Full-Stack

Jornada Usuário



Contexto e Complexidade

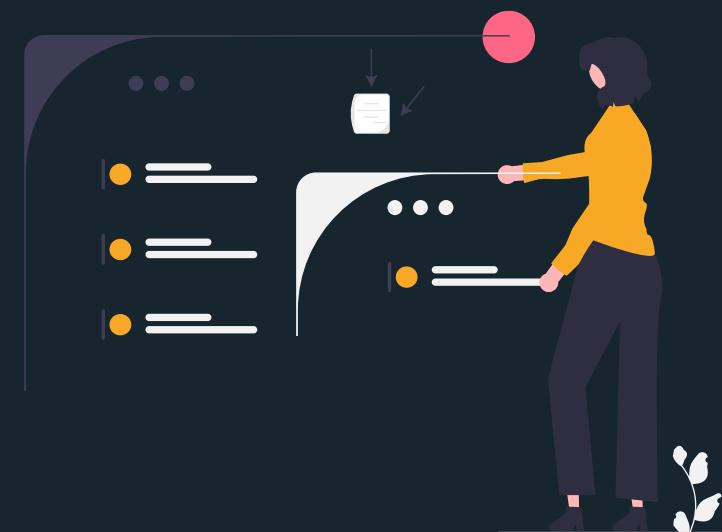
Desenvolvimento
Full-Stack



Nota importante

Definir o contexto e os tipos de abordagens arquiteturais levam a boa construção dos componentes ou escolha de bibliotecas, determinantes para o sucesso do projeto.

Saliento que nada é eterno. O contexto pode evoluir e as abordagens precisarão suportar adaptações



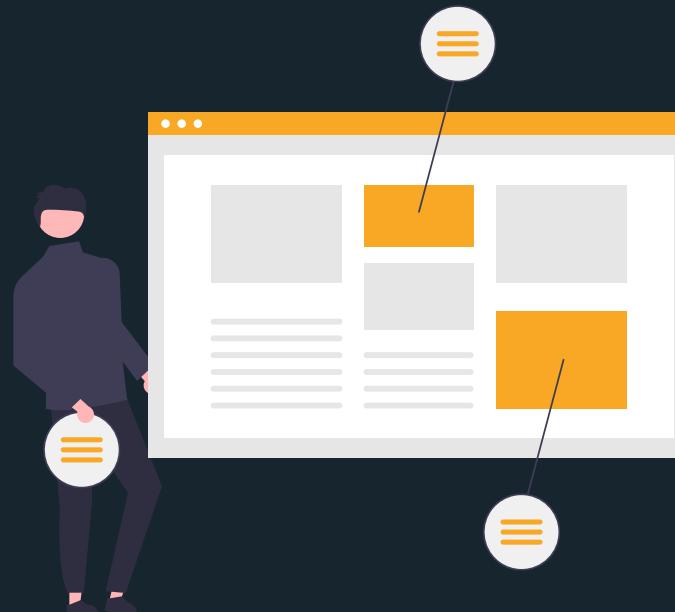
Contexto e Complexidade

Desenvolvimento
Full-Stack



Estratégia - Entendendo conceito de renderização

A renderização em tecnologias frontend refere-se ao processo de exibir e atualizar visualmente os elementos de uma interface de usuário em um dispositivo, utilizando um navegador da web ou em um aplicativo móvel. É a maneira como os dados são convertidos em elementos visuais que os usuários podem ver e interagir.

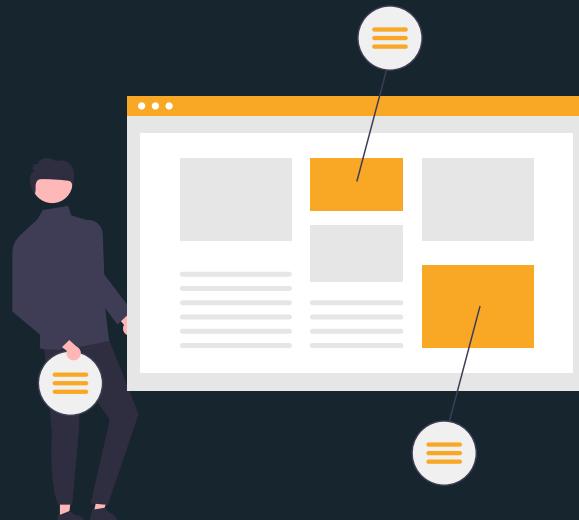




Estratégia - Tipos de Renderização no FrontEnd

Basicamente temos dois tipos de renderização na arquitetura client-side.

O CSR (Client Side Rendering) e SSR (Server Side Rendering)

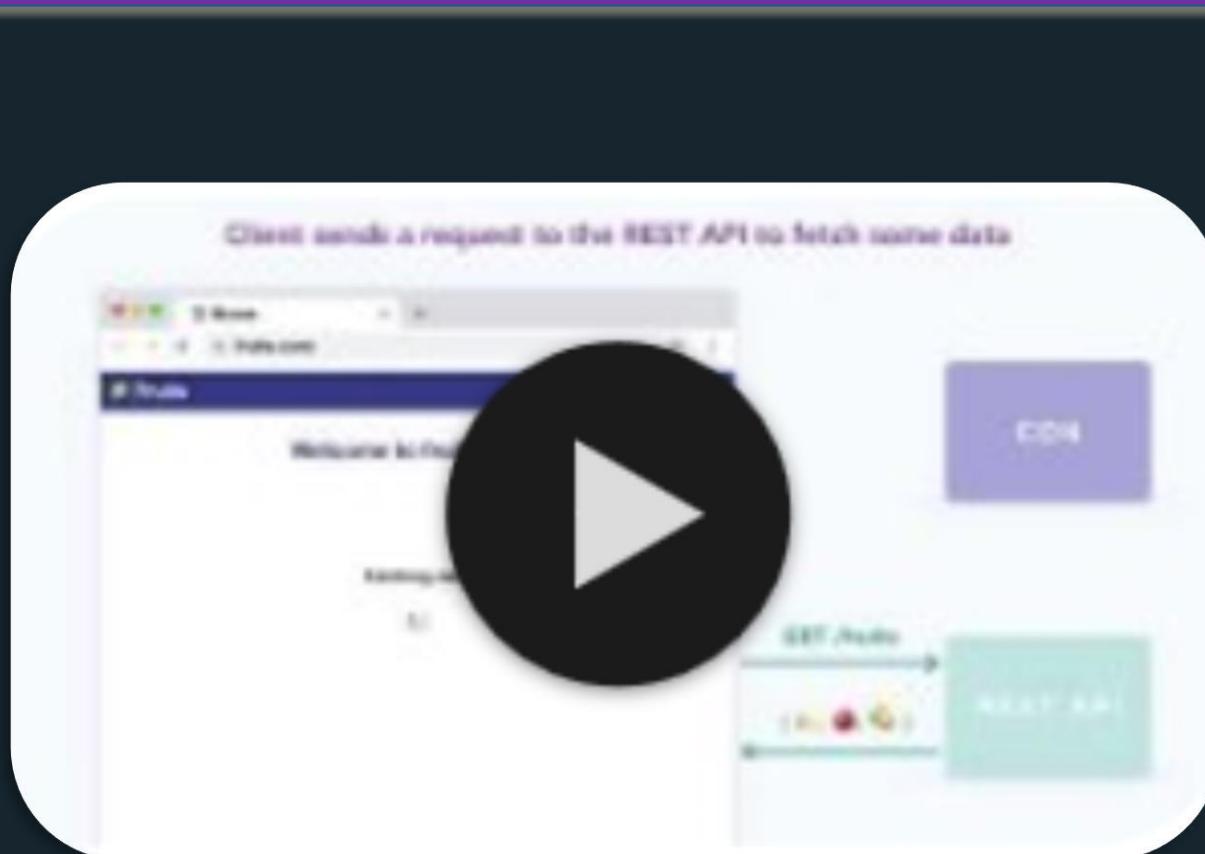
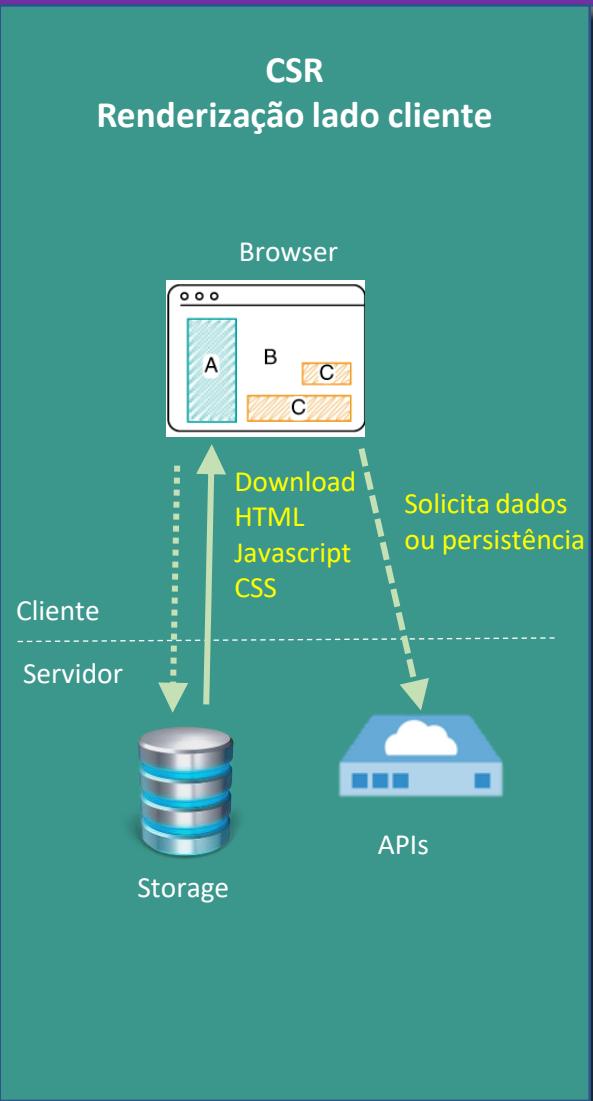


Contexto e Complexidade

Desenvolvimento
Full-Stack



CSR – Renderização no lado cliente

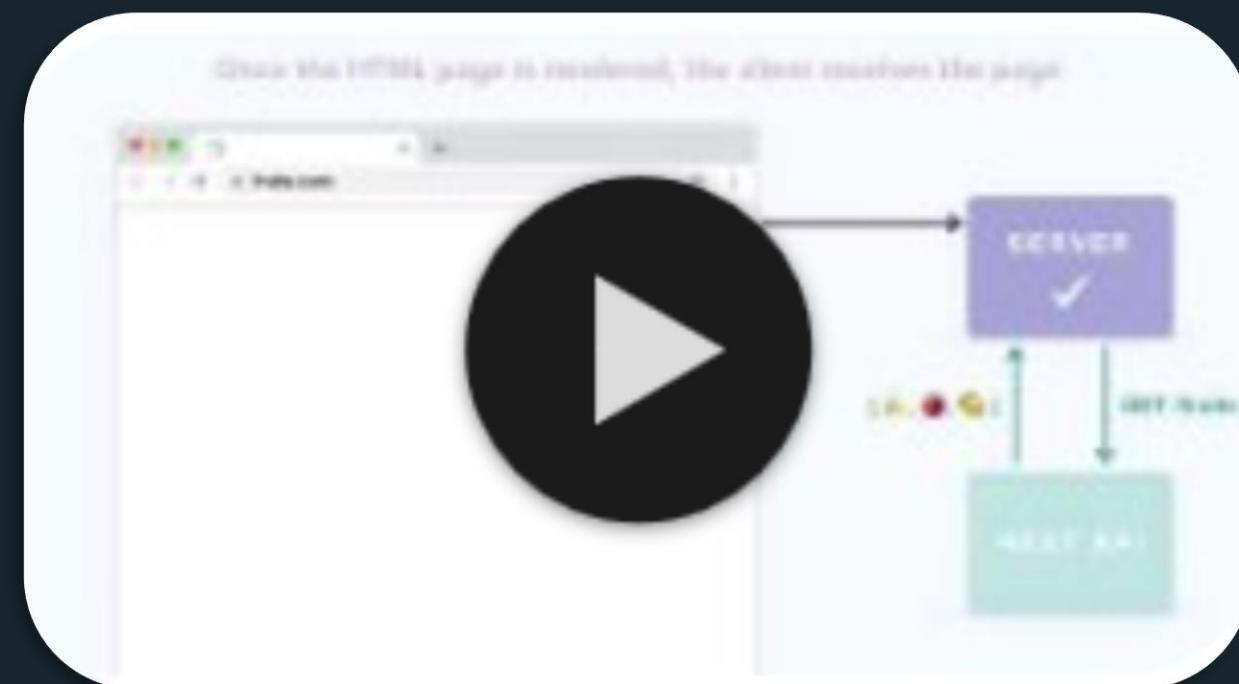
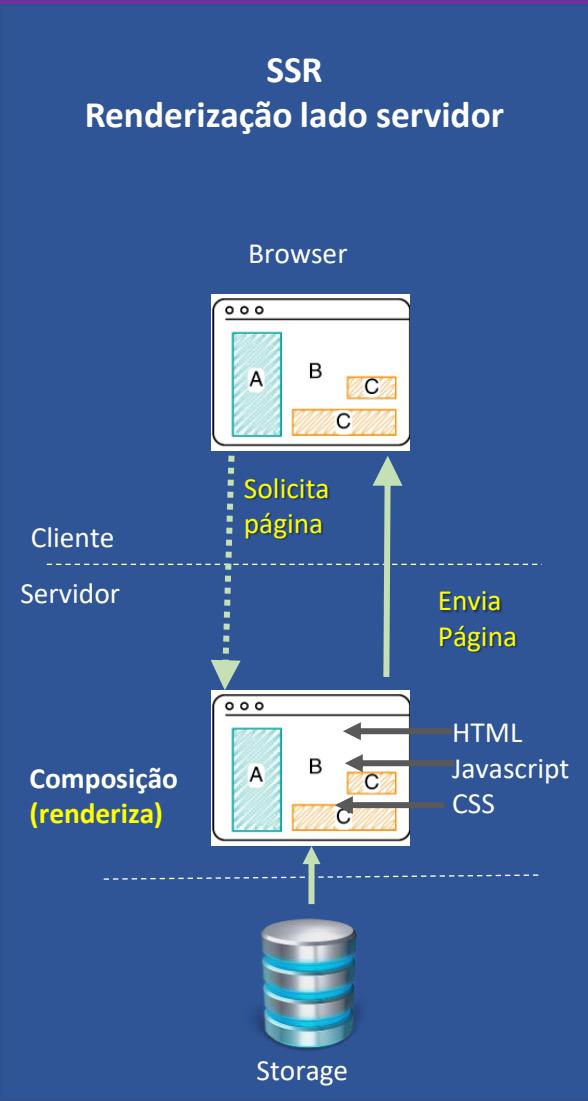


Contexto e Complexidade

Desenvolvimento
Full-Stack



SSR – Renderização no lado servidor

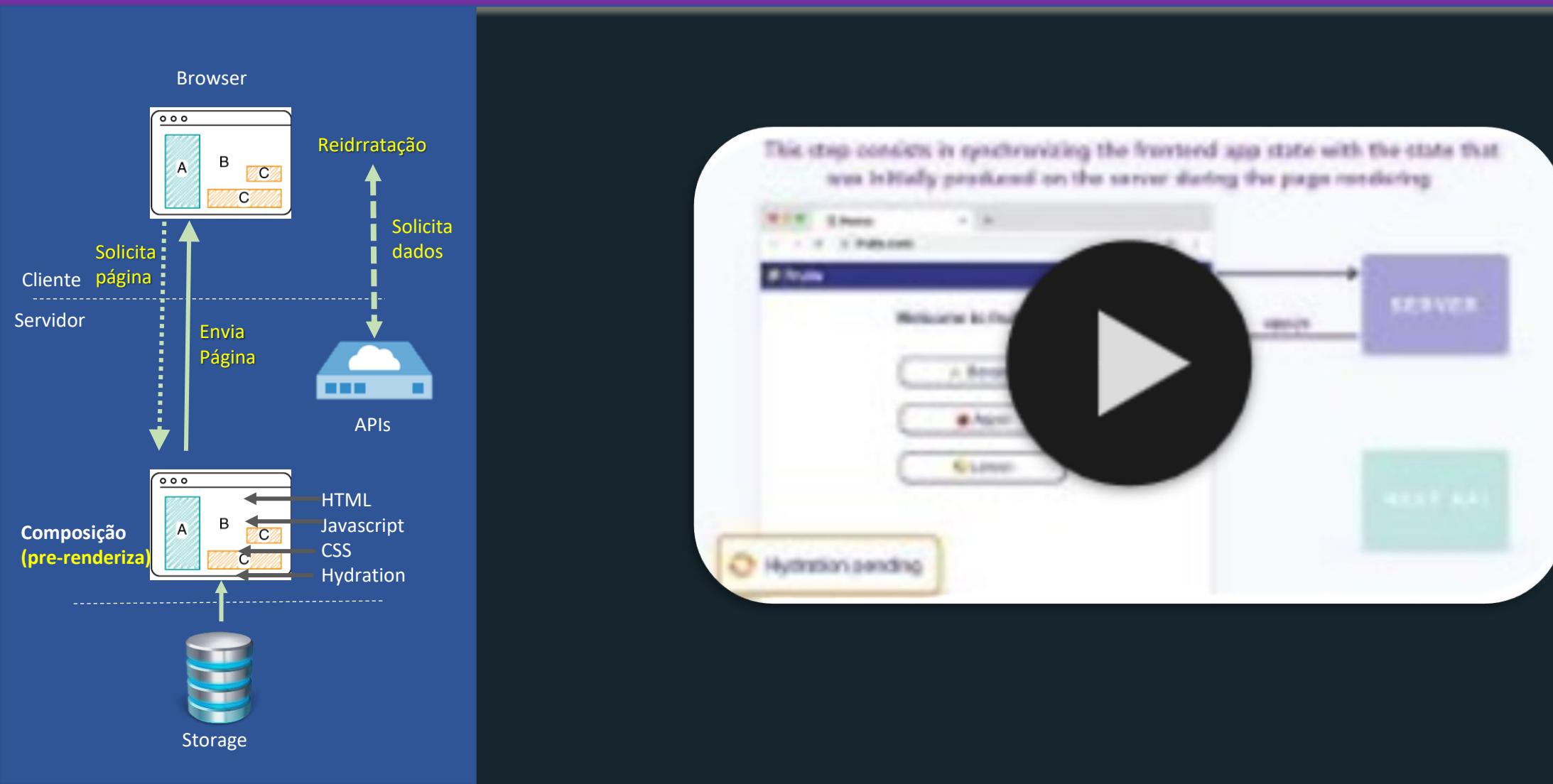


Contexto e Complexidade

Desenvolvimento
Full-Stack



Universal Rendering – Rehydration (reidratação)



Contexto e Complexidade

Desenvolvimento
Full-Stack



Exemplo Base e React

Exemplo React

```
<html>
  <head>
    <!-- cabeçalho, estilos, scripts, etc. -->
  </head>
  <body>
    <div id="app">
      <h1>Meu Blog</h1>
      <div id="posts">
        <!-- O conteúdo dos posts será reidratado aqui -->
      </div>
    </div>
    <script src="path/to/react.js"></script>
    <script src="path/to/react-dom.js"></script>
    <script src="path/to/bundle.js"></script>
  </body>
</html>
```

Onde será reidratado

```
// bundle.js
// Este é o código React para reidratar os dados pré-renderizados
// e renderizar a interface
// Aqui estão os dados pré-renderizados serializados em JSON
var postData = [
  {
    title: 'Título do Post 1',
    content: 'Conteúdo do Post 1'
  },
  {
    title: 'Título do Post 2',
    content: 'Conteúdo do Post 2'
  }
];

// Componente React para renderizar um post
function Post({ title, content }) {
  return (
    <div className="post">
      <h2>{title}</h2>
      <p>{content}</p>
    </div>
  );
}

// Componente React para renderizar a lista de posts
function PostList({ posts }) {
  return (
    <div className="posts">
      {posts.map((postData, index) => (
        <Post key={index} title={postData.title} content={postData.content} />
      ))}
    </div>
  );
}

// Encontra o elemento HTML onde a lista de posts será renderizada
var postsContainer = document.getElementById('posts');

// Renderiza a lista de posts no elemento HTML
ReactDOM.hydrate(<PostList posts={postData} />, postsContainer);
```

Mocando dados
"fetch"

Gerando json

Manipulando DOM

Reidratando "hydrate"
ao invés de "render"



Aprofundar na Arquitetura Client-side

Abordagens de desenvolvimento



SPA – Single Page Application

Os aplicativos de página única são muito adotados na atualidade, eles permitem que desenvolvedores criem páginas ricas, responsivas e interativas. Chegando próximo aos desenvolvimentos nativos. Toda a abordagem iniciou com uso de AJAX e códigos Javascripts complexos mas hoje fazemos uso de frameworks para simplificar este processo..

A adesão de frameworks como React, Angular, Vue foram essenciais para a aceleração do desenvolvimento e fornecem recursos de reutilização de código, otimização desempenho e estratégias de renderização tudo pra melhorar a experiência do usuário.

Toda SPA faz o trabalho de renderização no lado do cliente, mas nem toda aplicação que faz esse trabalho é uma SPA.



SPA – Renderização e a Lógica do Aplicativo

SPA são aplicações CSR (cliente side rendering),

Vale ressaltar que ao falarmos de CSR tratamos do sistema de renderização, os SPAs nos permitem que decidamos como vamos dividir a lógica do aplicativo entre servidor e cliente. E esta pode ser uma estratégia interessante.

Cliente magro/gordo ou Servidor magro/gordo ?



SPA – Vantagens e Desvantagens

Vantagens: altamente interativos, com atualizações frequentes e mais ágeis, tráfego mais leve, divide responsabilidade com servidor. Uso de bibliotecas de renderização universal vem ajudando na performance deste modelo.

Desvantagens: exige clientes mais potentes em processamento, problema mapeamento em ferramentas de busca (SEO), lento para carregar primeira página, ataques cross-site-scripting.

Outra questão relevante é que encontro atualmente no mercado soluções de código SPAs grandes, se tornando monolitos (dificultando implantação e distribuição).

Abordagens de Desenvolvimento

Desenvolvimento
Full-Stack



Mobile First

O conceito de mobile first começou a tracionar com a expansão dos dispositivos móveis e uma padronização dos sistemas operacionais embarcados nestes equipamentos. Sistemas operacionais como Android e iOS tornaram um padrão e surgiram diversas ferramentas capazes de desenvolver para estes ambientes.

No início, mesmo com a padronização, o desenvolvimento era complexo pois exigia linguagens de software específicas para códigos nativos. Uso de Kotlin e Swift foram os mais populares.

Atualmente existem kits de desenvolvimentos para frameworks que permitem construir aplicativos multiplataformas de forma muito mais fluida e simples. Estes kits geram códigos nativos a partir de um código único usando typescript ou dart por exemplo.



Mobile First – Vantagens e Desvantagens

Vantagens:

1. Instalação por lojas de aplicativos centralizado.
2. Maior performance no uso
3. Menor consumo de tráfego
4. Trabalhar offline
5. Enviar push notifications o que melhorar engajamento
6. Manter armazenamento local
7. Acesso aos recursos nativos dos dispositivos (incluindo NFC e acelerômetro)



Mobile First – Vantagens e Desvantagens

Desvantagens:

- 1. Instalação consome bom recurso do dispositivo.**
- 2. Resistência dos usuários para instalar um app. Foi comprovado que a grande maioria dos apps instalados são utilizando apenas uma vez.**
- 3. Não permite ser encontrado por mecanismos de busca**
- 4. Atualização de versão demanda atenção**

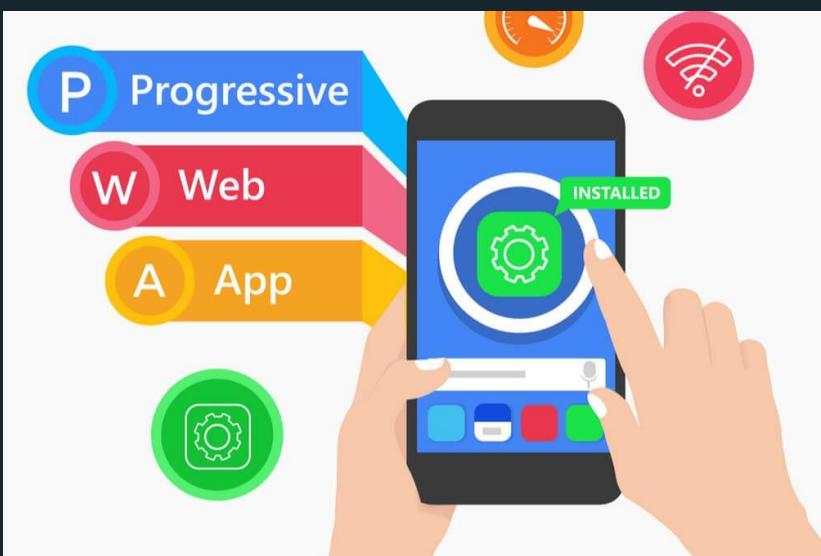


PWA – Progressive WebApp

O PWA (progressive web app) permitem criar aplicações web, normalmente SPAs, acessadas via browser que utilizam recursos nativos do dispositivo (como câmeras, biometria até mesmo envio de notificações) se comportando como um aplicativo nativo.

Seu nome “progressivo” se dá pois são aplicações que se adaptam de acordo com os recursos disponíveis do usuário. Aprimorados permitindo avançar uso de recursos (nativos do dispositivo por exemplo) se caso a plataforma permita, ou simplesmente se comportar com um site web em navegadores.

PWA unifica vantagens do desenvolvimento nativo com aplicações web responsivas.





PWA – Características

Existem algumas características para que possamos classificar ou até mesmo identificar uma aplicação PWA de acordo com o Google. São elas:

1. São detectáveis. PWA permitem mapeamento em sistemas de buscas
2. Aprimoramento progressivo.
3. Atualizações de app mais dinâmicas
4. São Apps engajáveis
5. São instaláveis (uso de Web App Manifest)
6. São construídos de forma responsiva. Se adaptando a qualquer dispositivo.
7. Permitem acessos offline
8. Permitem serem compartilhado pelas redes sociais sem maiores dificuldades



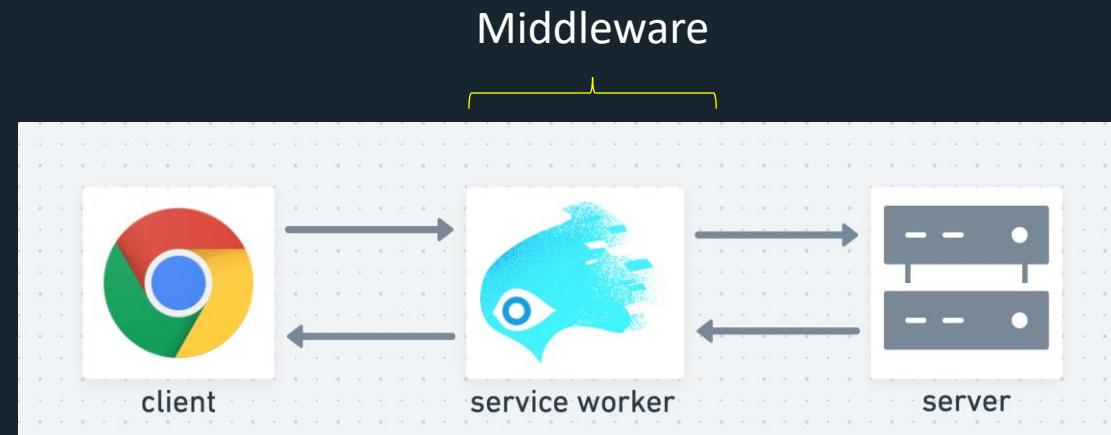
PWA – Service Worker

Podemos dizer que o service worker é o coração que impulsiona o PWA. Dentre outras coisas o service worker permitirá que seu aplicativo, por exemplo, trabalhe offline (utilizando recursos de cache avançado), interaja com o usuário mesmo que esteja fechado, como enviar notificações, usar recursos nativos como câmeras e biometria, e outras diversas APIs próprias para uso.

O service worker insere uma nova camada na aplicação cliente-servidor, ele se apresenta como uma ponte na comunicação, interceptando todas as chamadas. É como se fosse uma extensão do seu navegador servindo de camada intermediária após ser instalado.

Entenda mais:

<https://web.dev/service-worker-mindset/>

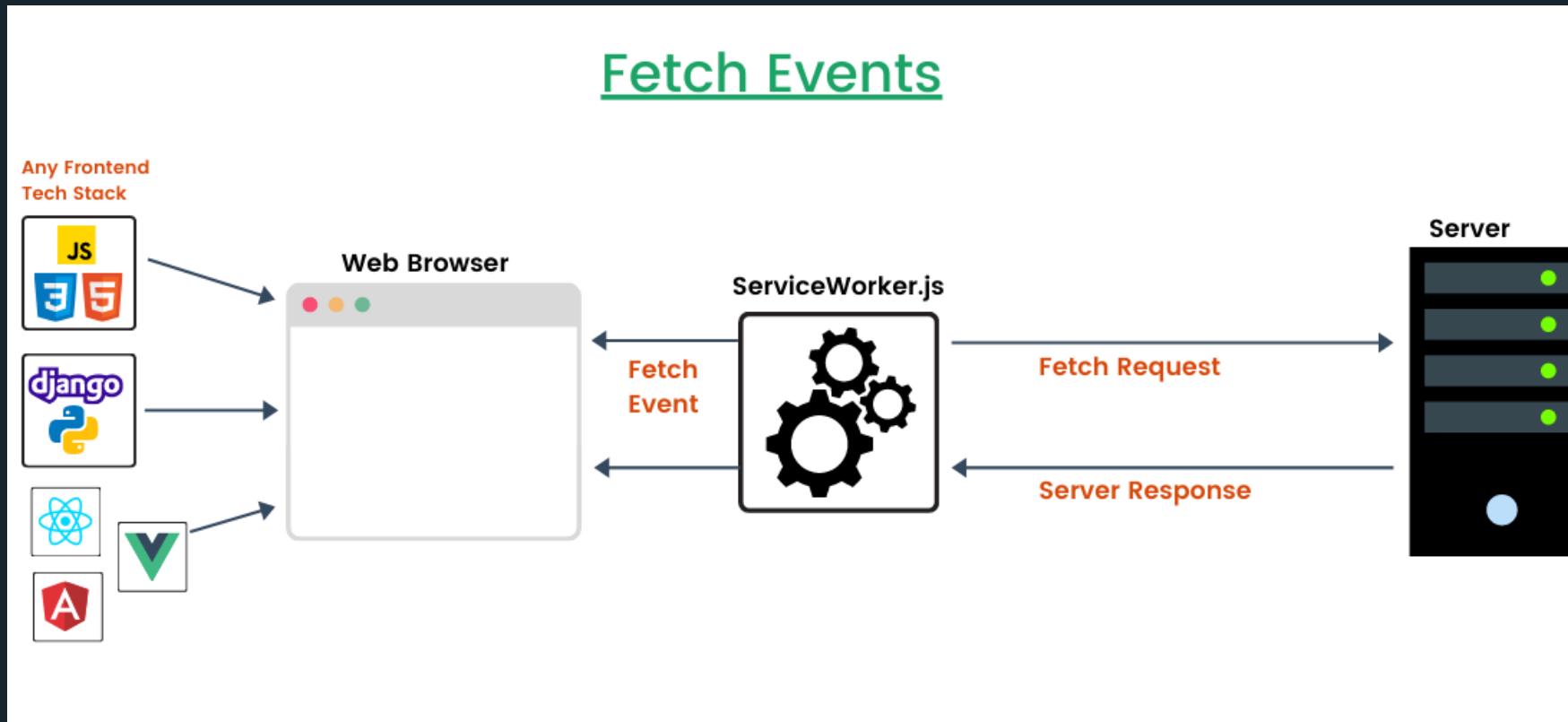


Abordagens de Desenvolvimento

Desenvolvimento
Full-Stack



PWA – Service Worker





PWA – Web Manifest

O manifesto web ou Web Manifest contém padrões mantidos pela W3C.

Trata de um arquivo JSON contendo informações para o navegador web de como deverá instalar a aplicação na home do dispositivo.

Neste arquivo é possível determinar por exemplo, o ícone, nome, descrição, cores background até ajustes de protocolo inserindo.

Um manifesto é vinculado a aplicação através de um link em seu html..

Abordagens de Desenvolvimento

Desenvolvimento
Full-Stack



PWA – Web Manifest

Manifesto

```
{  
  "lang": "en",  
  "dir": "ltr",  
  "name": "Super Racer 3000",  
  "short_name": "Racer3K",  
  "icons": [{  
    "src": "icon/lowres.webp",  
    "sizes": "64x64",  
    "type": "image/webp"  
  }, {  
    "src": "icon/lowres.png",  
    "sizes": "64x64"  
  }, {  
    "src": "icon/hd_hi",  
    "sizes": "128x128"  
  }],  
  "scope": "/",  
  "id": "superracer",  
  "start_url": "/start.html",  
  "display": "fullscreen",  
  "orientation": "landscape",  
  "theme_color": "aliceblue",  
  "background_color": "red"  
}
```

HTML

```
<!doctype>  
<html>  
  <title>Racer 3K</title>  
  
  <!-- Startup configuration -->  
  <link rel="manifest" href="manifest.webmanifest">  
  
  <!-- Fallback application metadata for Legacy browsers -->  
  <meta name="application-name" content="Racer3K">  
  <link rel="icon" sizes="16x16 32x32 48x48" href="lo_def.ico">  
  <link rel="icon" sizes="512x512" href="hi_def.png">
```



Vamos ver na prática!

Fonte: <https://w3c.github.io/manifest/#web-application-manifest>



PWA – Vantagens e Desvantagens

Vantagens:

- Sem sobra de dúvidas a maior vantagem dos PWAs é permitir desenvolver uma aplicação para diversas plataformas. Hoje sustentar o produto em cada plataforma exige muito investimento das Indústrias de software. Usando PWA estas empresas conseguem distribuir seu produto de forma mais otimizada com único deploy.
- Além de outros....

Desvantagens:

- Descoberta ainda é descentralizada dependendo de mecanismos de busca. Muitas indústrias de software disponibiliza seu aplicativo nas lojas de app.

Abordagens de Desenvolvimento

Desenvolvimento
Full-Stack



Reflexões ...

Técnicas de renderização e abordagens de desenvolvimento podem dizer muito sobre seu produto. Estes fatores podem alterar seu processo de desenvolvimento, sustentação e distribuição e podem refletir na experiência do usuário.

Existem diversas técnicas e oportunidades na arquitetura client-side que permitem estruturar um produto como híbrido. Estas técnicas estendem estilos arquiteturais encontrados no backend para o frontend, como o uso de Microfrontends.

E quando usar um ou outro ?

A resposta novamente é depende do contexto do projeto. Bom observar as tendências e mercado, como o site “Stateofjs” que apresenta pesquisas anuais e tendências.

https://2022.stateofjs.com/pt-PT/usage/#js_app_patterns





Aprofundar na Arquitetura Client-side Frameworks

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Frameworks

Os frameworks de desenvolvimento frontend são conjuntos de ferramentas, bibliotecas e padrões de código que facilitam a criação de interfaces de usuário interativas e responsivas em aplicativos web.

Tem como vantagem principal, ajudar em agilizar o desenvolvimento de software. Eles fornecem uma estrutura básica e consistente para o desenvolvimento, permitindo que os desenvolvedores se concentrem mais na lógica de negócios do aplicativo e menos em configurações e detalhes de baixo nível de execução.





Frameworks – Vantagens e Desvantagens

Observando o trabalho de construção de alguns projetos, sem e com frameworks, observamos junto as equipes de desenvolvimento alguns fatores de vantagens e desvantagens.

Vantagens:

Código, Agilidade, Compatibilidade, Responsividade e Comunidades

Desvantagens

Curva de aprendizado, restrições e sobrecarga

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Frameworks

Principais frameworks atualmente:

- 1.React - [209k stars](#)
- 2.Vue.js - [204k stars](#)
- 3.Next.js - [107k stars](#)
- 4.Angular - [88.5k stars](#)
- 5.Svelte - [68.3k stars](#)
- 6.Ember.js - [22.4k stars](#)

É importante ressaltar que o cenário de frameworks de desenvolvimento frontend está em constante evolução, com novas opções surgindo regularmente e outros frameworks se adaptando e melhorando suas funcionalidades.

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Frameworks – Senta que lá vem história....

A história do Angular....

Em 2009, a equipe do Google lança a primeira versão do Angular chamada de “AngularJS”, o framework era baseado em Javascript e seguida filosofia de programação declarativa. Teve boa adesão pois unificava em uma mesma plataforma a possibilidade de combinar injeção de dependência, abordagem declarativa em htmls e recursos de data-bind e componentização.

Em 2016 o Google lança o Angular 2 redesenhadado utilizando abordagem de componentes e a linguagem typeScript. Não foi simples, a transição, pois envolveu mudanças substanciais na arquitetura e nas APIs do framework. E códigos desenvolvidos em AngularJS não eram compatíveis com a nova estrutura. Isto gerou um desconforto no cenário de desenvolvimento , e por uma decisão estratégica a Google lança o Angular 4 (pulando a versão 3) deixando claro que a versão Angular 2 não era uma sequência compatível com AngularJS. Portanto, cabe ressaltar que não temos a versão 3 por este motivo.

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Frameworks – Senta que lá vem história....

Particularidades....

O Angular é um framework opinativo, ou seja, possui conceito de “baterias incluídas”. Um conjunto de recurso e padrões predefinidos embutidos no framework. Isto o faz ser mais verboso, ou seja, o mais simples componente em uma aplicação ainda carrega um bom código typescript.

Por ser opinativo, o Angular pode apresentar maior produtividade e menor curva de aprendizado inicialmente, por que teoricamente basta “ligar” e “usar”.

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Frameworks – **Senta que lá vem história....**

Vamos falar do React...

Em 2013, o Facebook lançou no mercado o React como uma biblioteca de código aberto. Com objetivo de resolver desafios de renderização eficiente de interfaces de usuário em tempo real.

A motivação do Facebook em lançar o React no mercado, **está relacionado** às suas necessidades internas de desenvolvimento de software e à busca por uma solução eficiente para lidar com a complexidade crescente dos aplicativos da empresa. A empresa precisava buscar melhores soluções de renderização, interatividade e performance para suas interfaces como feed por exemplo. E as abordagens atuais resultavam em dificuldade de sustentação e desempenho mais lento

Foi nesse contexto que a equipe do Facebook, começou a trabalhar em uma nova abordagem para a construção de interfaces de usuário, buscando uma solução que fosse rápida, escalável, modular e que permitisse uma renderização eficiente das alterações na interface.

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Frameworks – Senta que lá vem história....

Particularidades

O React e Vue são exemplos de framework não opinativo, ou seja, não impõe uma estrutura rígida e permite flexibilidade para desenvolvedores nas suas decisões sobre ferramentas e bibliotecas complementares.

Existem diversas formas de se implementar componentes e serviços utilizando-se de instalação e configuração de bibliotecas. Diferente do Angular que já possui na sua estrutura como nativo.

Por um lado traz mais flexibilidade, liberdade para o desenvolvedor fazer suas escolhas. Mas por outro insere maior complexidade e exige experiência sobre as dependências inseridas no projeto.

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Frameworks – **Senta que lá vem história....**

Vamos falar do Vue.js...

Uma estrutura acessível, de alto desempenho e versátil para a construção de interfaces de usuário da Web. Seja você um freelance, startup ou empresa maior. É assim que o Vue.js se apresenta no seu site oficial vuejs.org.

O Vue.js foi criado em 2014 por Evan You, com objetivo de oferecer um framework mais simples, flexível de baixa curva de aprendizagem para desenvolvimento web.

A motivação de Evan foi gerar Simplicidade, flexibilidade, reatividade e dinamismo

Essas características e a motivação por trás do Vue.js fizeram com que o framework ganhasse popularidade, sendo amplamente adotado em muitos projetos. Ele é utilizado por empresas de diferentes tamanhos e setores.



Frameworks – Senta que lá vem história....

Particularidades

O Vue.js permite adotar a linguagem typescript.

Principais recursos do Vue.js:

- **Renderização declarativa** : o Vue estende o HTML padrão com uma sintaxe de modelo que nos permite descrever declarativamente a saída HTML com base no estado do JavaScript.
- **Reatividade** : o Vue rastreia automaticamente as mudanças de estado do JavaScript e atualiza eficientemente o DOM quando as mudanças acontecem.



Next.js - Introdução

NEXT.js é um framework de desenvolvimento web baseado em React que visa trazer recursos para otimizar a arquitetura client-side e os aplicativos SPAs

Ele nos trás a possibilidade de adotar estratégias mais dinâmica entre cliente e servidor, oferecendo uma experiência de desenvolvimento eficiente e um desempenho aprimorado.

Principais recursos do NEXT.js:

1. Renderização otimizada dividindo componentes e operações entre CSR e SSR. Incluindo estratégia de páginas com pré-renderização com hidratação
2. Melhora a estratégia de SEO utilizando abordagem combinada de renderizações.
3. Roteamento avançado
4. Não existe configurações (incluso sistema de rotas, hot reloading, etc)
5. Ecossistema React, aproveita bibliotecas e componentes

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Next.js - Conclusão

- Combina desempenho aprimorado no SSR com a rica interatividade do CSR.
- Fornece uma maneira fácil de dividir a renderização do seu aplicativo.
- O sistema de pré-renderização por mais que seja otimizado leva a necessidade de Javascript adicional no cliente para tornar o HTML inicial interativo.

NEXT
.JS



Frameworks – Conclusão

Bom salientar que o conhecimento é muitas vezes é **transferível** e conceitos presentes em um framework podem ser aplicados em outro.

Importante dominar conceitos, definir fronteiras nos componentes e evitar muitas dependências **buscando a tecnologia** que traga mais conforto se encaixe melhor com os pré-requisitos e objetivos do seu projeto. É mais importante ter uma solução arquiteturada com **arquitetura limpa e escalável** que que adotar uma tecnologia.

Algumas dicas:

1. Alinhe arquitetura x frameworks
2. Evite múltiplas versões de frameworks
3. Em projetos de migração de legado adote o desenvolvimento progressivo



Frameworks – Conclusão

Se pudermos comparar os frameworks comentados, poderíamos concluir que:

Angular tem um vasto ecossistema, pode ser mais claro para que tem costume de desenvolvimento no backend, com desenvolvimento mais rígido. Mas exige pre-requisitos com curva aprendizado alto.

React e Vue são mais flexíveis, possuem arquivos menores por ser menos verboso, fácil de ser usado com outros frameworks proporcionando um desenvolvimento mais progressivo. Mas exige conhecimento adicional em várias ferramentas para auxiliar no desenvolvimento.



Frameworks – Mercado 2022 - StateofJS

- **Frameworks Frontend**
 - https://2022.stateofjs.com/pt-PT/libraries/front-end-frameworks/#front_end_frameworks_experience_linechart
- **Framework renderização**
 - https://2022.stateofjs.com/pt-PT/libraries/rendering-frameworks/#rendering_frameworks_experience_linechart
- **Recursos para aprender**
 - <https://2022.stateofjs.com/pt-PT/resources/>



Aprofundar na Arquitetura Client-side Frameworks Multiplataformas



Frameworks Mobile – Introdução

Desde o surgimento dos smartphones em 2008 , os aplicativos móveis fizeram parte de nosso cotidiano, fornecendo acesso a dados, serviços e entretenimentos.

Antes do advento dos frameworks de aplicativos móveis, os desenvolvedores enfrentavam a tarefa árdua, era necessário criar aplicativos nativos separados para cada plataforma seja Android ou iOS. Isso exigia aprender diferentes linguagens de programação, ambientes de desenvolvimento e bibliotecas específicas, o que resultava em um processo demorado e custoso. As linguagens muito populares foram Koltin (Android) e Swift (iOS).

Foi nesse cenário que os frameworks de aplicativos móveis surgiram como soluções inovadoras para simplificar e acelerar o desenvolvimento de aplicativos.

Eles prometem não só compartilhar código base entre plataformas mobile mas atualmente já atingem desktop e web com PWA.



Frameworks Mobile – Introdução

Esses frameworks fornecem uma camada de abstração sobre as complexidades das plataformas móveis, permitindo que os desenvolvedores escrevam um único conjunto de código-fonte que pode ser instalado entre diferentes plataformas.

Frameworks de aplicativos móveis no desenvolvimento eficiente é imensa. Eles oferecem uma série de benefícios.

1. Reutilização de código
2. Rapidez no desenvolvimento
3. Acesso a recursos nativos
4. Melhor performance da aplicação



Frameworks Mobile – Introdução

1. React Native:

- 1. Desenvolvido pelo Facebook.
- 2. Permite criar aplicativos nativos para iOS e Android usando JavaScript e a biblioteca React.
- 3. Compartilha código entre plataformas, resultando em maior eficiência no desenvolvimento.
- 4. A partir de maio/22 um novo sistema renderização JSI (Javascript Interface)



2. Flutter:

- 1. Desenvolvido pelo Google.
- 2. Permite criar interfaces de usuário nativas e de alto desempenho para iOS, Android, web e desktop usando a linguagem de programação Dart.
- 3. Utiliza a engine Skia para renderização, garantindo interfaces de usuário visualmente atraentes e responsivas.



3. Xamarin:

- 1. Desenvolvido pela Microsoft.
- 2. Permite desenvolver aplicativos nativos para iOS, Android e Windows usando C# e a plataforma .NET.
- 3. Compartilha grande parte do código, proporcionando maior produtividade e rapidez no desenvolvimento.



4. Ionic:

- 1. Baseado em HTML, CSS e JavaScript.
- 2. Permite criar aplicativos multiplataforma usando tecnologias da web.
- 3. Oferece uma ampla variedade de componentes e temas personalizáveis.

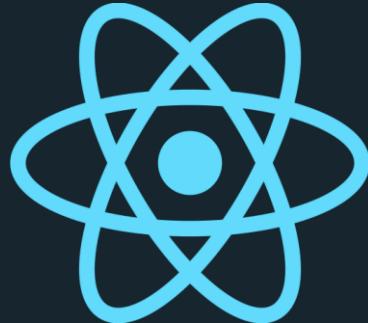


Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



React Native



React Native é um framework de desenvolvimento de aplicativos móveis criado pelo Facebook. Ele permite criar aplicativos nativos para iOS e Android usando JavaScript e a biblioteca React.

Com o React Native, os desenvolvedores podem criar interfaces de usuário utilizando componentes pré-fabricados semelhantes aos utilizados na construção de aplicativos tradicionais nativos. Esses componentes são renderizados de forma nativa, o que significa que o desempenho do aplicativo é semelhante ao de um aplicativo desenvolvido diretamente em Swift (para iOS) ou Java (para Android).

Para renderizar componentes ele faz uso de uma arquitetura denominada “React Native bridge architecture” que insere uma ponte de interação com dispositivo.

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



React Native

A principal vantagem do React Native é a capacidade de compartilhar código entre as plataformas iOS e Android. Com um único código-base, os desenvolvedores podem criar aplicativos para ambas as plataformas, o que economiza tempo e recursos. Além disso, o React Native oferece uma experiência de desenvolvimento mais rápida e iterativa, permitindo que as alterações sejam visualizadas instantaneamente sem a necessidade de compilação completa.

No entanto, é importante observar que nem todo o código é compartilhado entre as plataformas. Em alguns casos, pode ser necessário escrever código específico para cada plataforma, especialmente quando se trata de recursos ou APIs específicos de cada sistema operacional.

Site oficial:

<https://reactnative.dev>

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Flutter



Flutter é um framework de desenvolvimento de aplicativos móveis de código aberto criado pelo Google. Ele permite criar interfaces de usuário nativas e de alto desempenho para iOS, Android, Web e desktop usando uma única base de código escrita em Dart, uma linguagem de programação também desenvolvida pelo Google.

Ao contrário de abordagens tradicionais de desenvolvimento de aplicativos, onde a interface do usuário é renderizada usando componentes nativos, o Flutter utiliza sua própria engine de renderização chamada Skia para criar uma interface de usuário personalizada. Isso permite que os aplicativos Flutter tenham uma aparência consistente e fluida em todas as plataformas, semelhante a aplicativos nativos.

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Flutter

Se destaca no Flutter...

Um SDK (Kit de Desenvolvimento de Software): Uma coleção de ferramentas que irão ajudá-lo a desenvolver seus aplicativos.

Uma estrutura (biblioteca de interface do usuário baseada em widgets): oferece um amplo conjunto de widgets pré-fabricados e personalizáveis, permitindo que os desenvolvedores criem interfaces de usuário bonitas e responsivas com facilidade.

Repositório oficial de pacotes: extensa coleção de pacotes e plugins para soluções específicas e acessos recursos de dispositivos, como câmera, sensores, geolocalização e muito mais.

Site oficial:

<https://flutter.dev>

Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Flutter x React Native

Abordagens	Flutter	React Native
Linguagem programação	Dart	Javascript permitindo acesso rápido a quem desenvolve em React
Renderização	Biblioteca SKIA	Bridge (<i>Legacy Native Component</i>) <i>Fabric Native Components(*)</i>
Acesso recursos nativos	Plataform Channel	Bridge (<i>Native Modules</i>) <i>Turbo Native Modules(*)</i>
Desempenho/Performance	Adota uma camada para acesso a recursos nativo e engine SKIA que melhorar renderização.	Usa componente nativo que exigem mais configuração
Suporte para plataformas	Android, iOS, Web (PWA) e Desktop *com a mesma base de código	Android, iOS e Desktop * Pode ser necessário código adicional para adaptação
Comunidade	154k stars Possui plataforma com diversos pacotes	110k stars
Site Oficial	https://flutter.dev/	https://reactnative.dev
Estudo de Caso	Nubank https://flutter.dev/showcase/nubank	Shopify https://shopify.engineering/migrating-our-largest-mobile-app-to-react-native

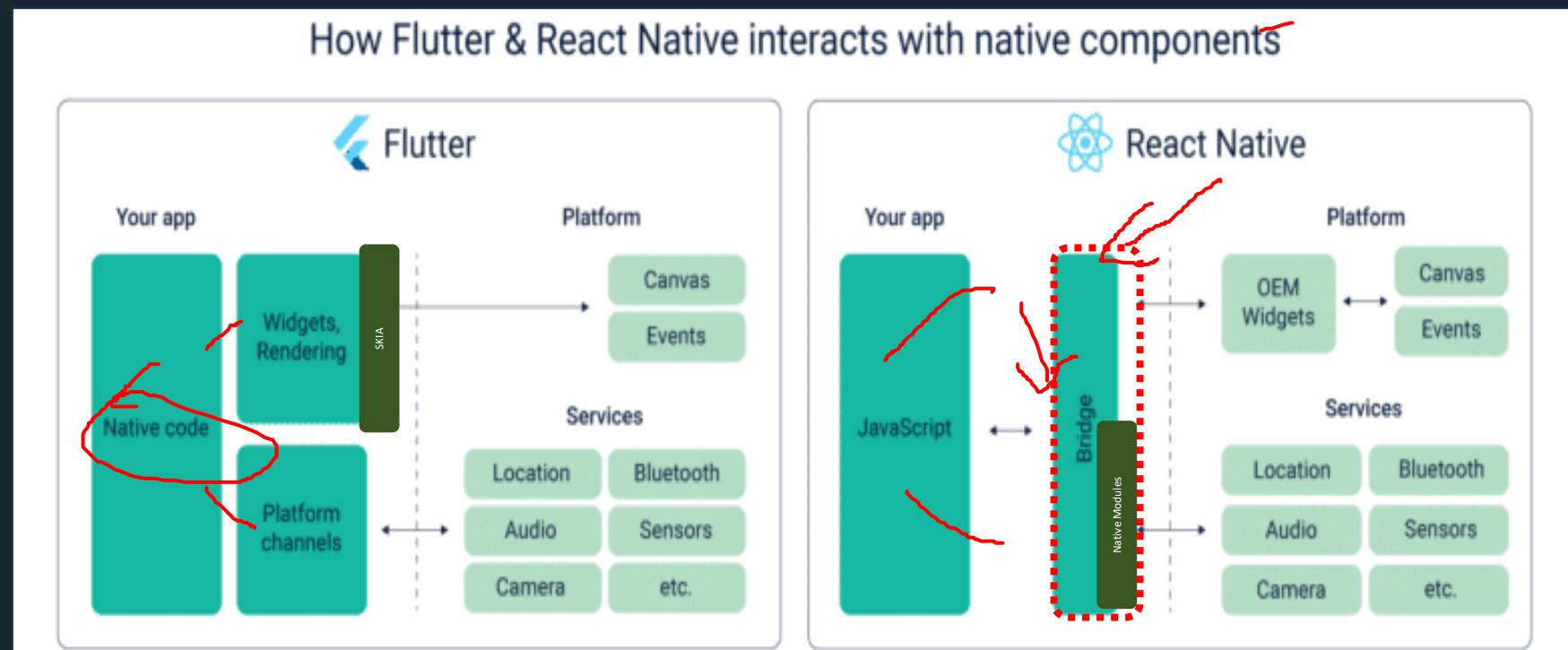
Aprofundar na Arquitetura Client-Side

Desenvolvimento
Full-Stack



Flutter x React Native

(*) Uma das maiores críticas ao framework do React Native era adotar a Bridge para que renderização e acesso a módulos nativos para cada plataforma para acessar recursos nativos. O Bridge era assíncrono e gerava problemas de compatibilidade. Desde maio/22 o React Native vem adotando uma nova arquitetura retirando o Bridge utilizando-se do JSI (Javascript Interface) uma abordagem fortemente tipada que promete acesso síncrono e maior compatibilidade pois o código todo é abstraído como uma unidade independente do sistema operacional.



PUCRS online  uol edtech