APRESENTAÇÃO
DE APOIO

ARQUITETURA CLIENT-SIDE

Andre Luiz Mendes Pereira – Aula 02

PUCRS online uol edtech

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 hipermídia, programação de sistemas para Web, frameworks multiplataforma para dispositivos móveis. Atua como revisor técnico da Sagah.

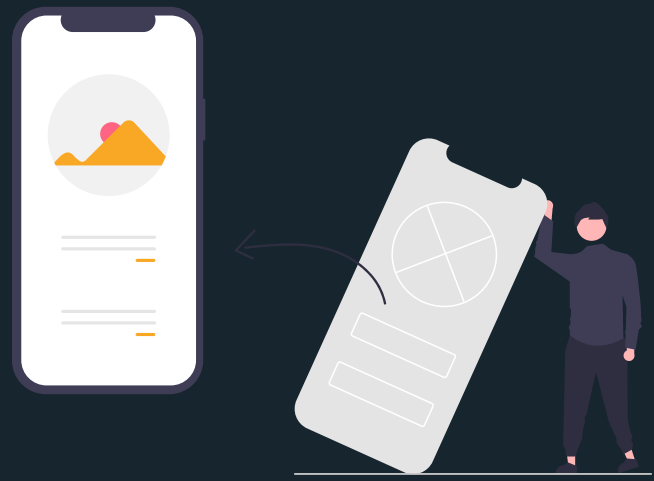
Ementa da disciplina

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.

Introdução

Observando os diversos fatores que falamos sobre arquitetura client-side até agora, deve ter ficado mais claro o quanto projetos de desenvolvimento de aplicações web estão ganhando complexidade.

Mas porque estamos mais complexos ?
Quais os fatores da mudança ?
Quais as consequências?

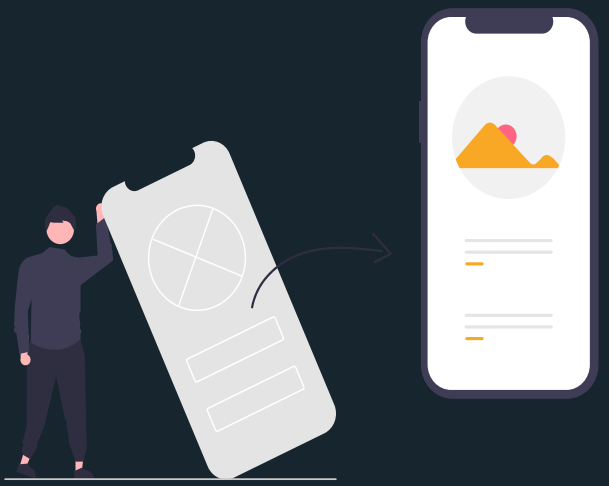


Serviços

Introdução

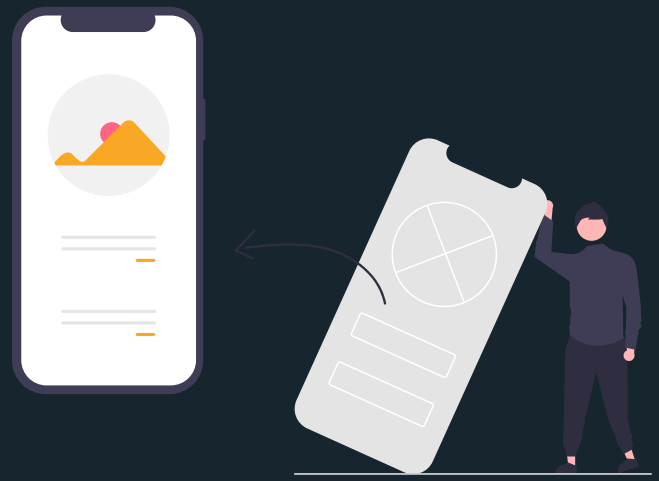
Vamos destacar algumas motivações.

Responder ao mercado exigente
Time-to-market
Necessidade de ampliar oferta de serviços
Aumento número de usuários
Empresas tradicionais buscando evolução
Empresas introduzindo equipes ágeis e multidisciplinares



Introdução

E é neste contexto que gostaria de que refletíssemos sobre a essência da arquitetura client-side

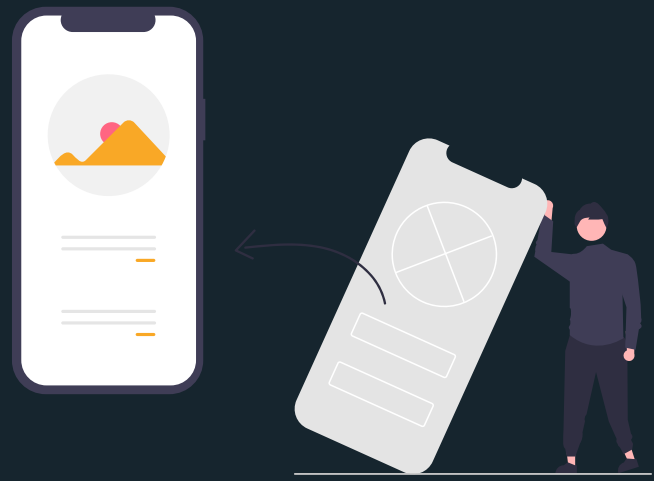


Serviços

Introdução

Grande parte do contexto da arquitetura client-side lida com serviços ou micro serviços.

Estes serviços ou micro serviços resultam em prover informações que serão visualizadas pelo usuário ou gerarão outras interações.



Aprofundar na Arquitetura Client-side

Micro serviços

Micro Serviços

Mas o que seriam micro serviços ?

Os micro-serviços web são uma abordagem arquitetural para o desenvolvimento de software, na qual uma aplicação é dividida em componentes independentes, isolado.

Estrategicamente, os micro-serviço permitem você buscar escala, maior produtividade do time, manutenibilidade, melhores testes, agilidade na implantação dentre outros benefícios.

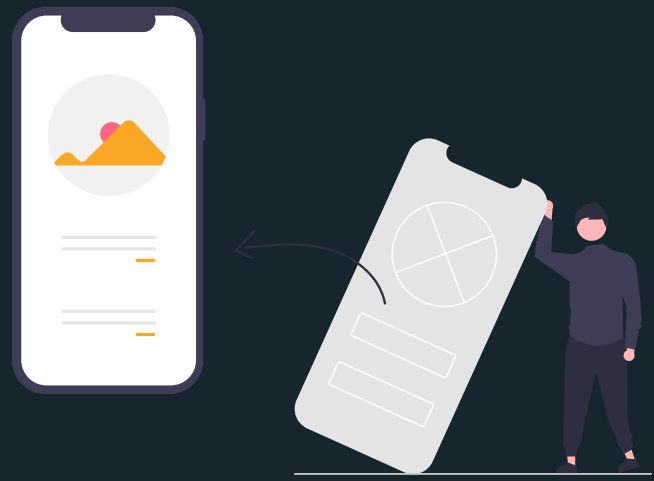
Micro Serviços

Introdução

Mas antes de avançarmos... é importante termos uma visão de correlação histórica com monolitos.

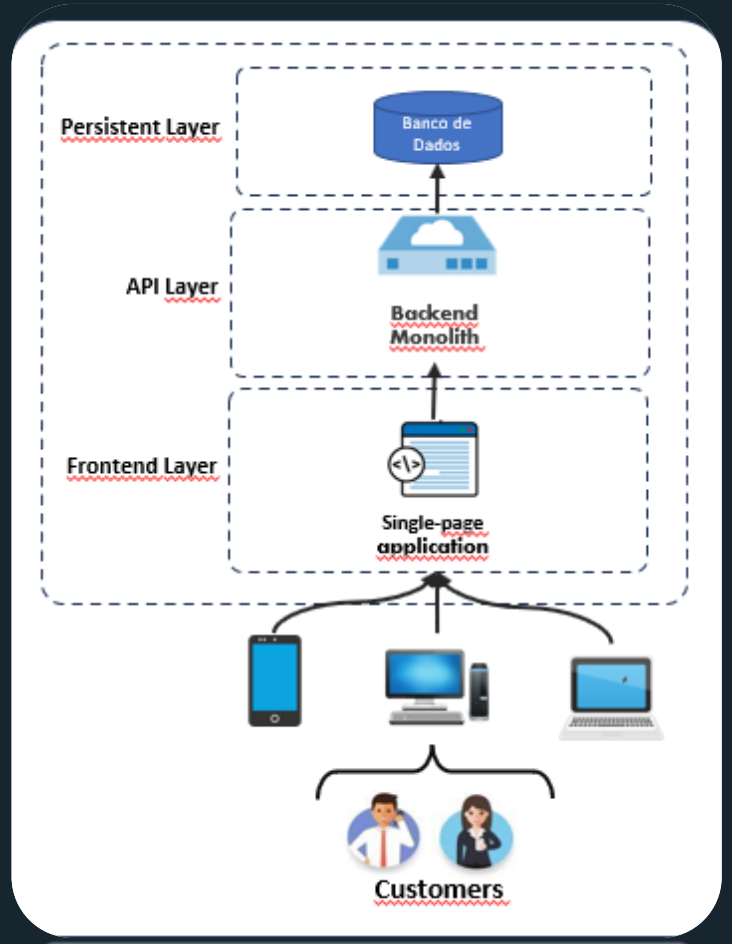
Uma evolução natural inclusive.

Quando adotar um ou outro ?
Quais motivações de usar micro serviços?
Quais estratégias de construir ou migrar serviços em micro serviços ?



Micro Serviços

Consequências - Mercado

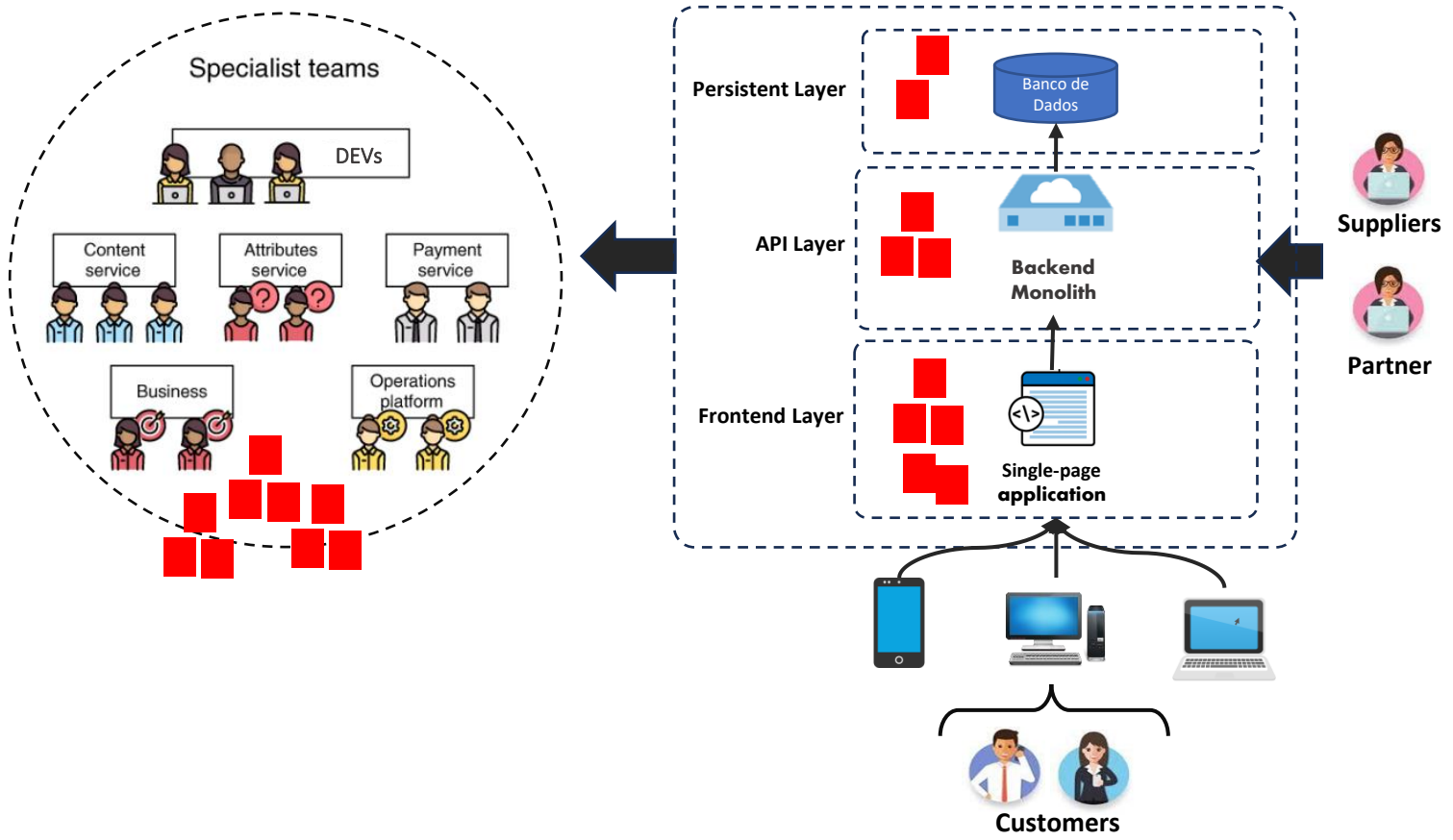


Cenário muito comum nas empresas, startups e principalmente tradicionais que refatoraram seu produto para Serviços Web. Percebe-se que já existe um conceito de camadas no projeto separando APIs do negócio **mas na mesma solução**.

Porque SPA e Monolito ?

Micro Serviços

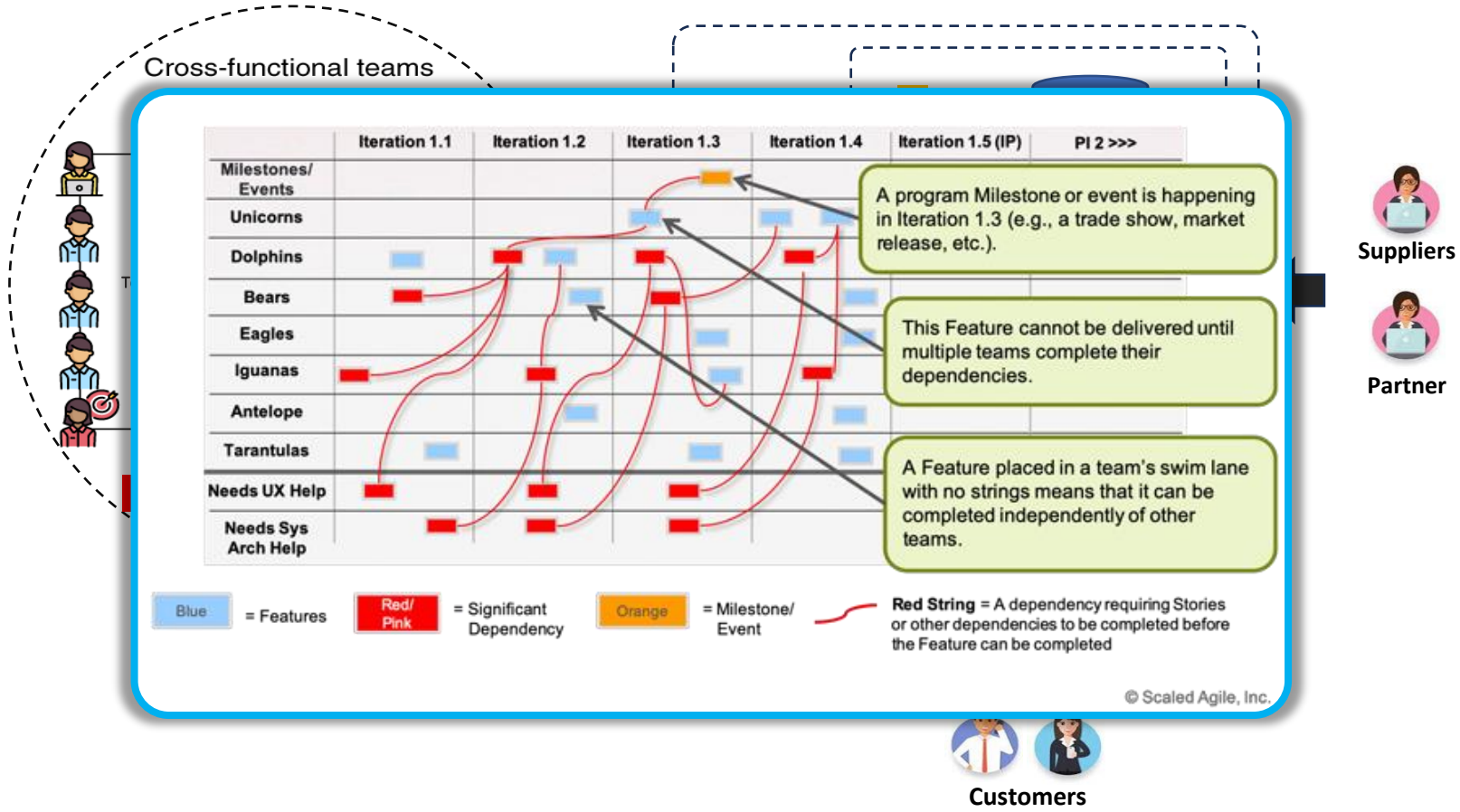
Empresas Organizadas por Skills



Necessidade de evolução

Vamos refletir sobre um estudo de caso....

Empresas Organizadas por Multifuncionalidade



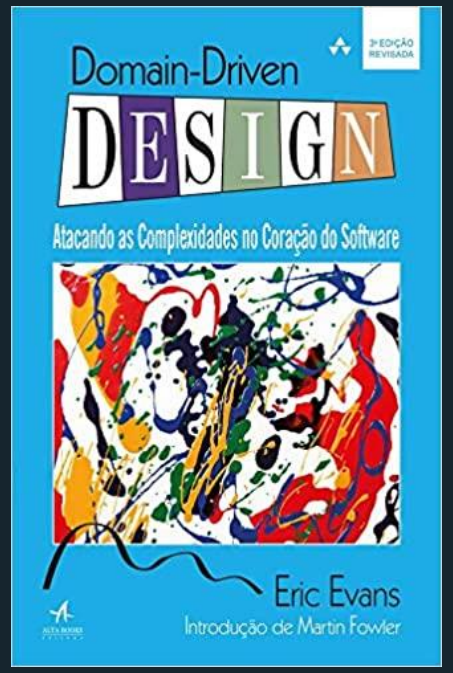
Micro Serviços

Estratégia para multifuncionalidade

Ao definirmos equipes multifuncionais a primeira iniciativa que precisamos pensar ao falarmos desta estratégia é que se tenha um mapeamento claro do modelo do negócio, seus domínios, subdomínios e suas fronteiras.

Algumas técnicas como DDD auxiliam neste processo.

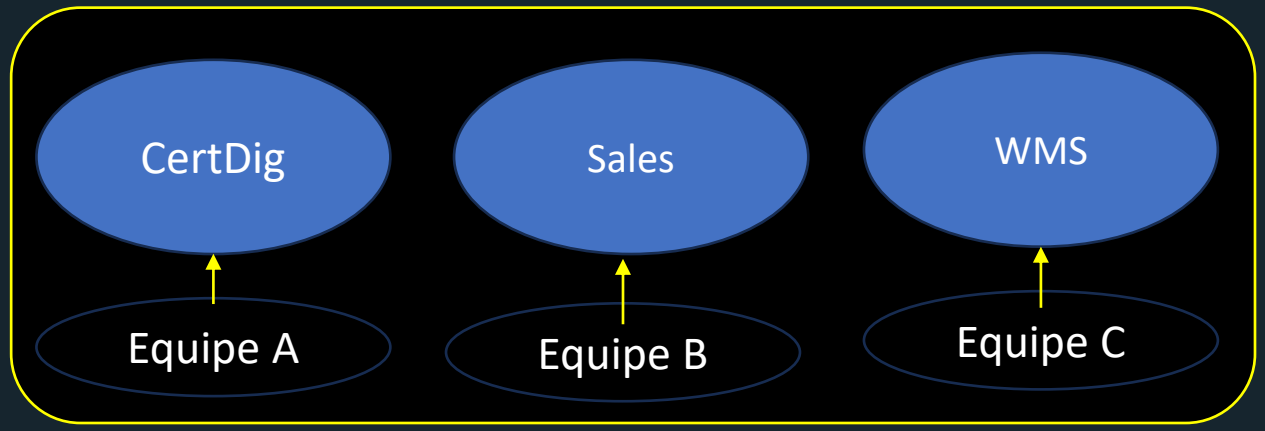
Esta técnica auxiliam empresa a encontrar as fronteiras dos seus micro serviços e conduzir tais apis de negócio direcionados pelos domínios



Estratégia Micro serviços

Falando resumidamente do DDD podemos mencionar que:

1. Domínio seria o problema a ser resolvido pelo software
2. Domínios são divididos em sub-domínios que tipicamente refletem a estrutura das organizações.
3. Então refletindo sobre isto, dividir o time para ser responsável por completo do sub-domínio pode ser uma boa prática.



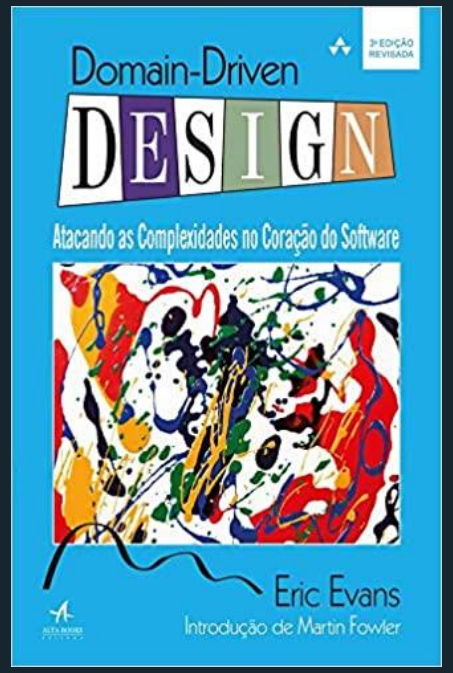
Micro Serviços

Estratégia Micro frontends

Esta técnica não só nos permite definir fronteiras mas estabelecermos estratégias de migração dos monolitos para micro serviços.

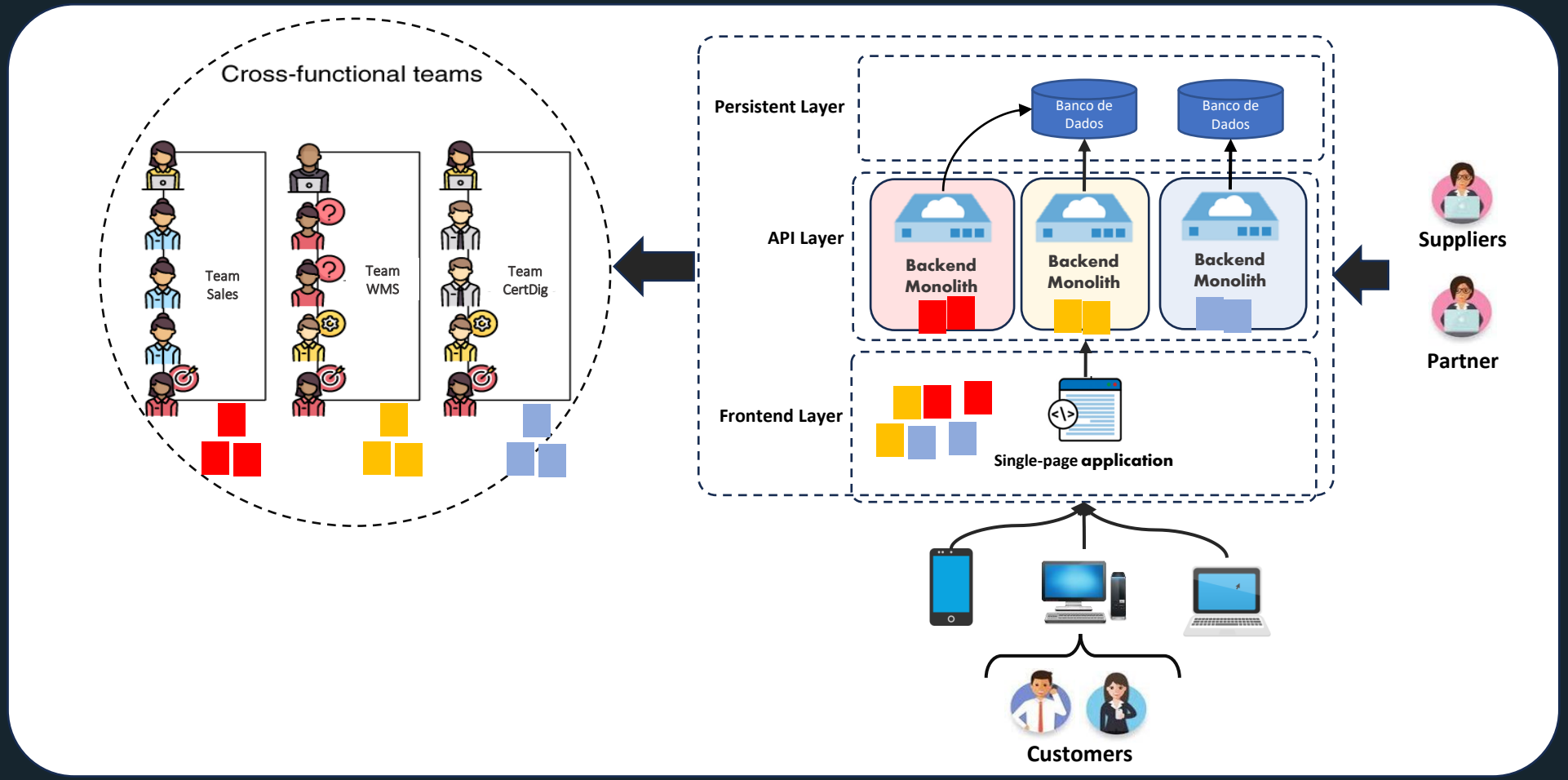
Seja transformando-os em monolitos modulares, prática adotada pela Shopify. Seja construindo micro serviços, prática adotada pela Netflix.

Tudo dependerá da estratégia para atender contexto .



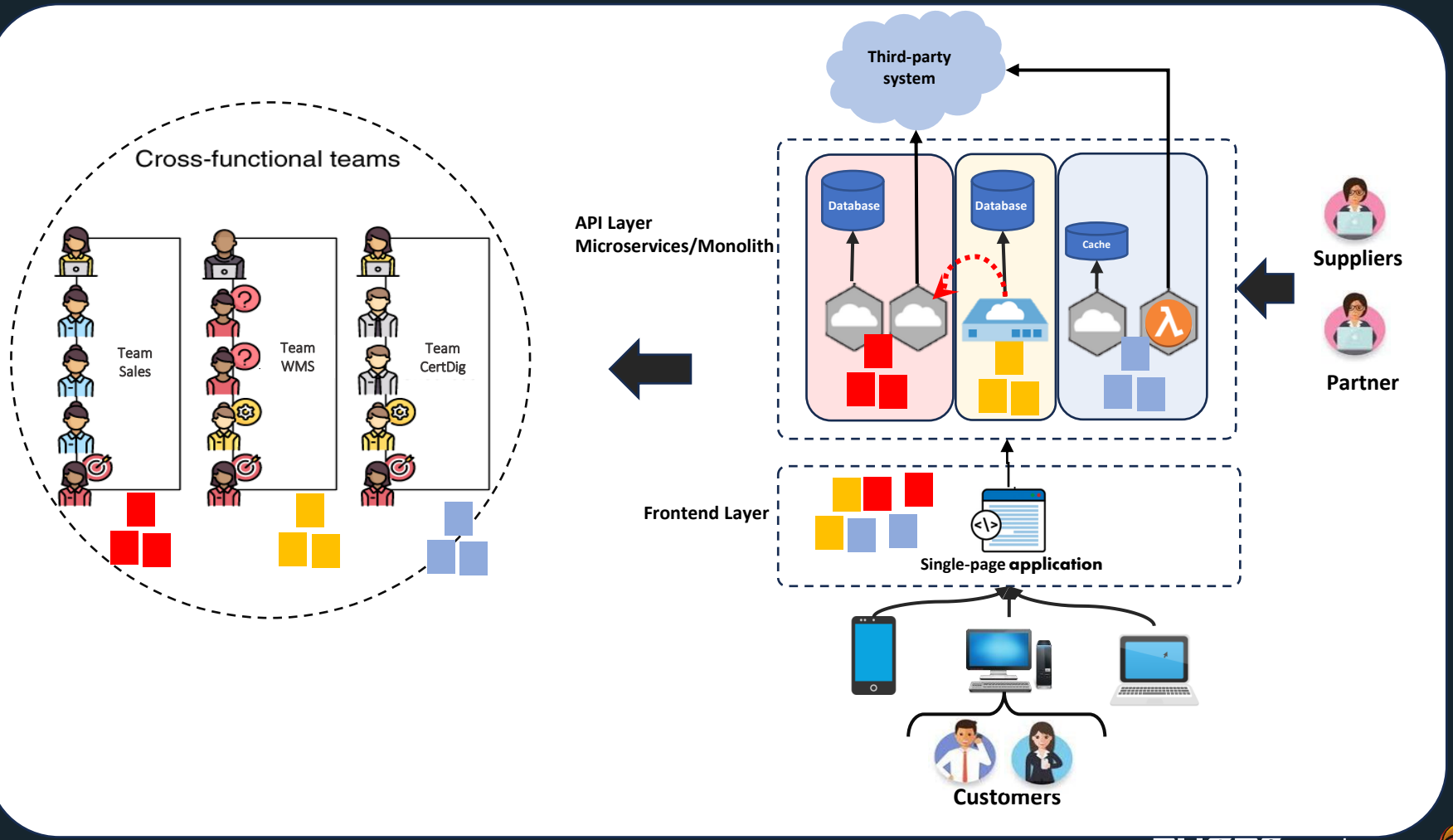
Entende-se por contexto, toda a estrutura da empresa, cultura, habilidades dos desenvolvedores, cronograma, orçamento, etc.

Multifuncionalidade – Monolito Modular



Micro Serviços

Micro serviços e Monolito



Micro Serviços

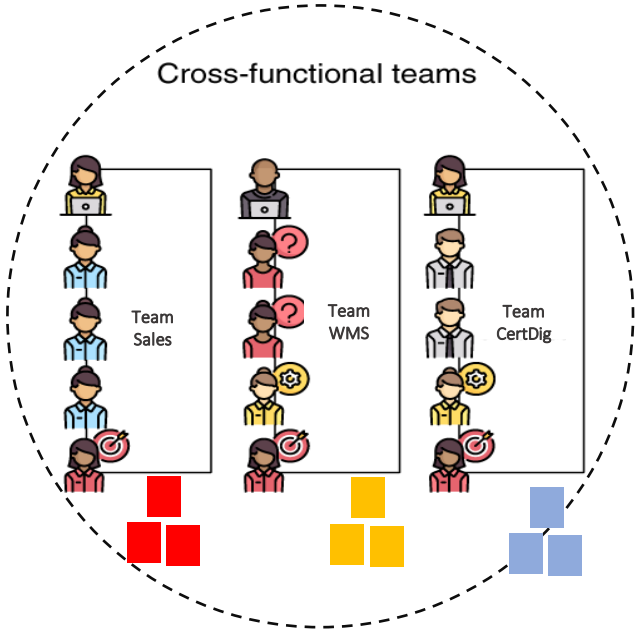
Gateway – Evolução do modelo

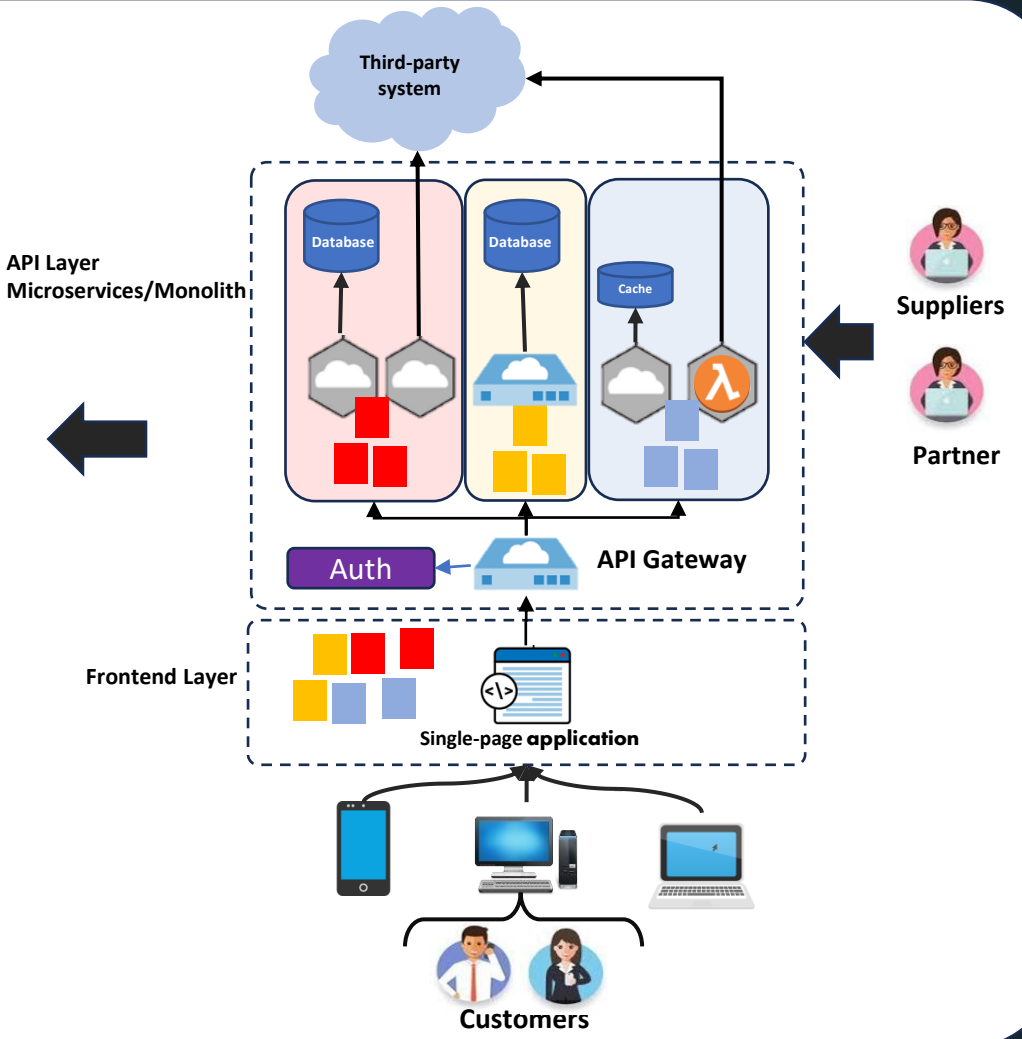
Com este novo modelo, torna-se importante alguns personagens.

O gateway é uma camada de API, servindo de ponte entre o FrontEnd e os serviços.

Esta camada servirá de gerenciamento de acessos, inclusive autenticação centralizada, aos serviços o micro serviços.

Bem como, de orquestração com serviços externos ou preocupações transversais as responsabilidades do serviços.



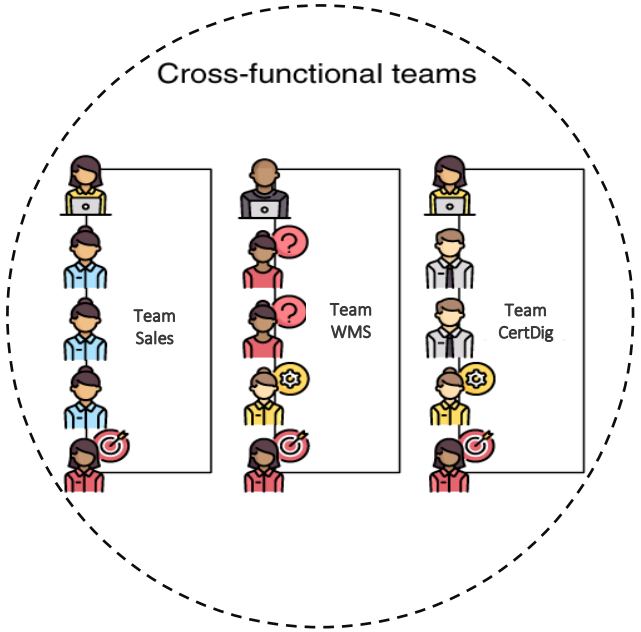


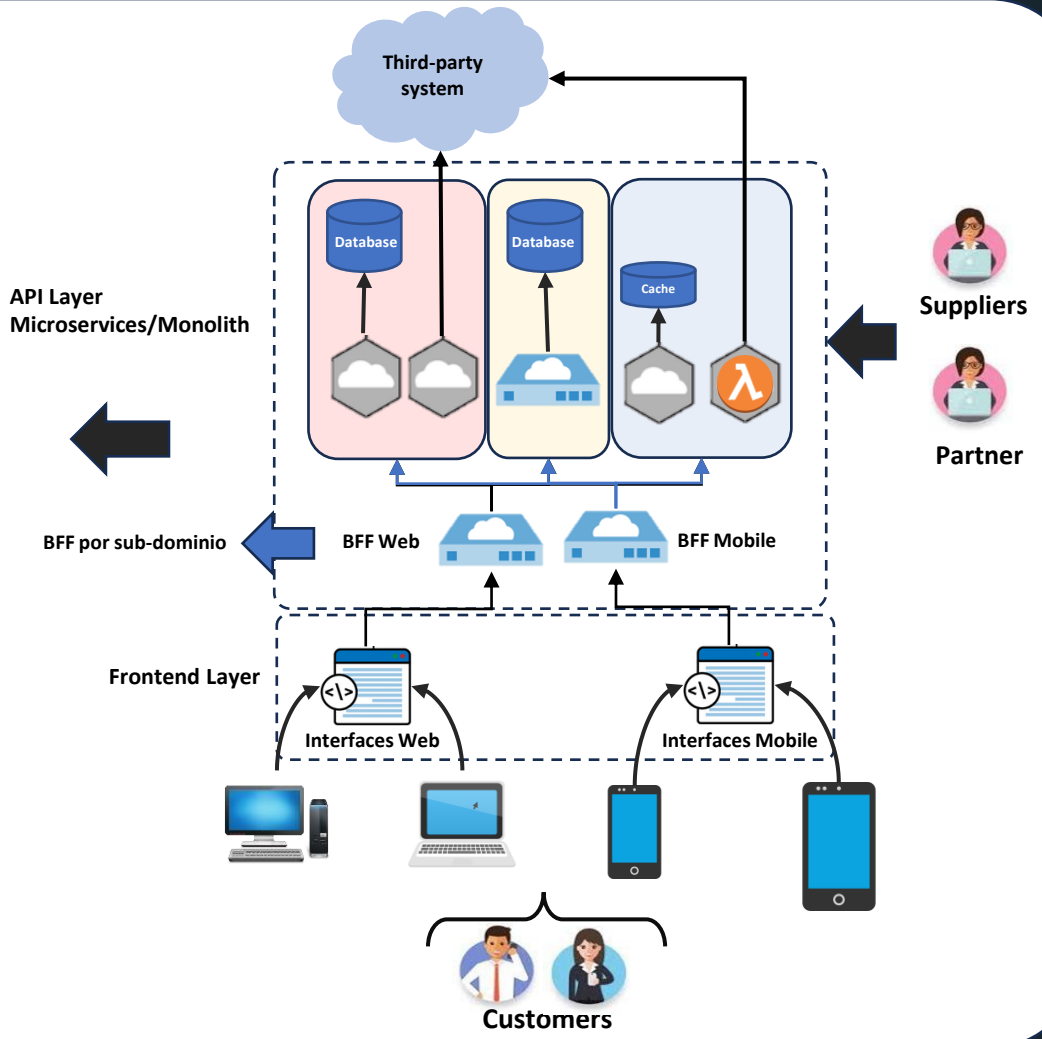
Micro Serviços

BFF – Backend for Frontend – Evolução do modelo

Outro personagem importante para evoluirmos ainda mais seria o BFF.

A principal distinção entre um BFF e um gateway de agregação central é que um BFF tem um propósito único por natureza – é desenvolvido para uma interface de usuário específica. Esse padrão tem se provado ser muito bem-sucedido pois melhora a experiencia e otimiza a troca de dados.



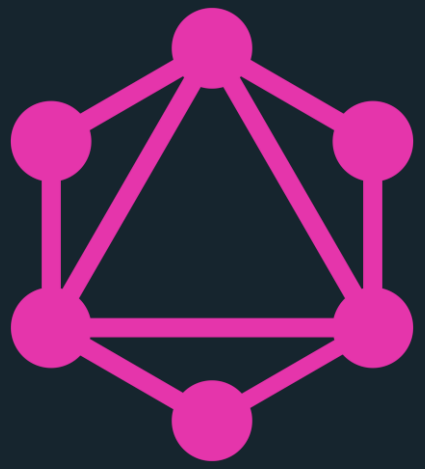


Micro Serviços

GraphQL – O aprimoramento

O GraphQL se apresenta como uma solução mais sofisticada que o BFF para atender diversos dispositivos e interfaces.

GraphQL é uma linguagem de consulta que permite que os clientes executem queries como parâmetro da chamada do serviço para acessar ou modificar dados específicos e sob demanda. Assim como o SQL, o GraphQL permite que essas queries sejam alteradas dinamicamente, possibilitando que o cliente defina exatamente as informações que quer obter.



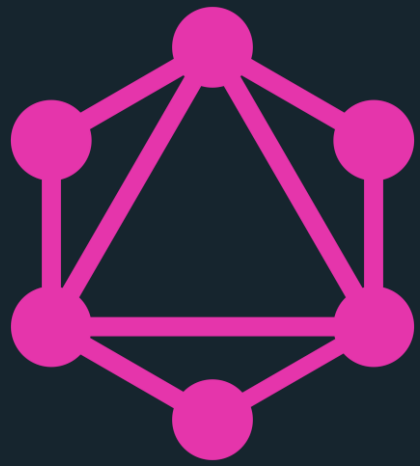
Com o GraphQL, podemos gerar uma requisição pedindo somente os campos necessários, hoje usando APIs REST o dado deveria estar contido no retorno exigindo chamadas extras para obter dados complementares.

Micro Serviços

GraphQL

Especialmente para desenvolvedores de front-end, o GraphQL representa uma ótima maneira de recuperar os dados necessários para renderizar uma exibição, desacoplar a complexidade de uma camada de API, racionalizar a resposta da API em um gráfico e permitir que qualquer cliente reduza o número de idas e vindas ao servidor para compor a IU.

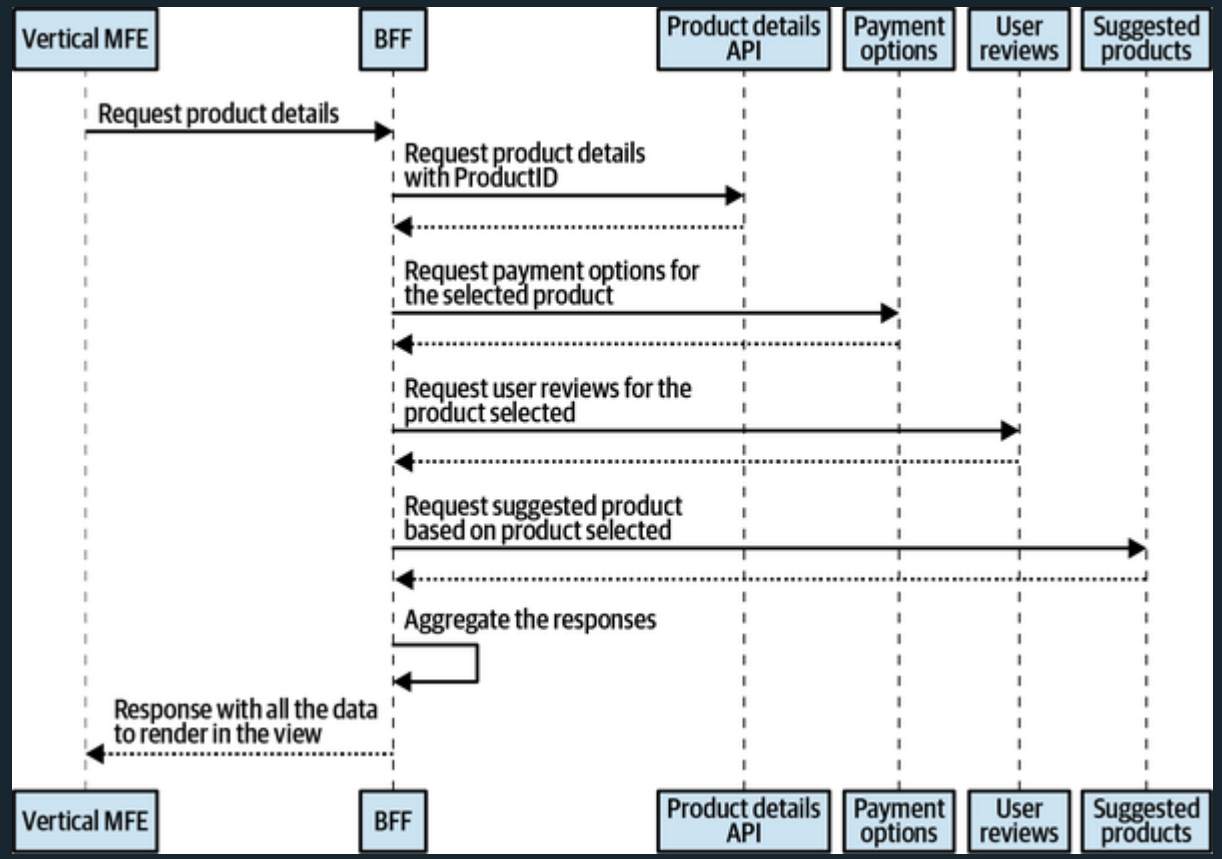
O GraphQL pode ser uma boa opção quando utilizamos estilos **de microfrontends**. Teríamos um único ponto de entrada em que possamos integrar variando a necessidade do retorno, sob medida para o fragmento de tela que desejamos..



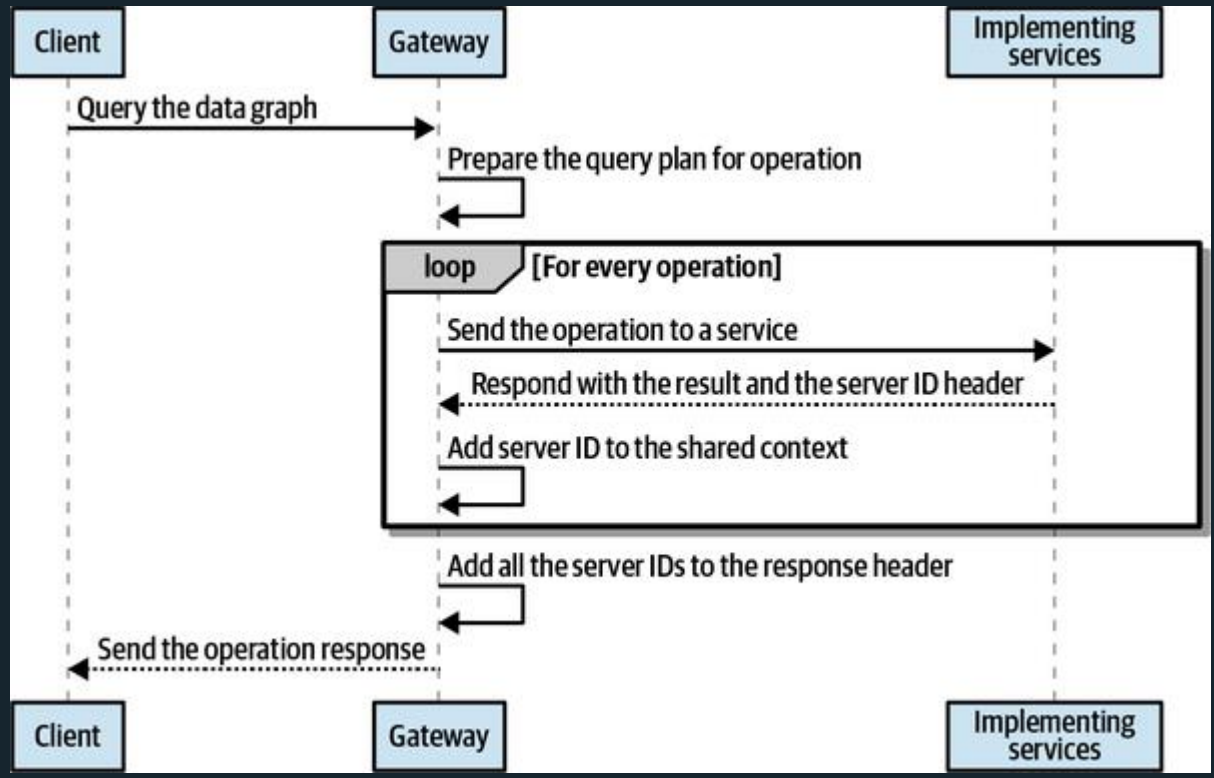
Micro Serviços

BFF *versus* GraphQL – Diagrama de Sequência

BFF – Necessitaríamos *N* BFFs

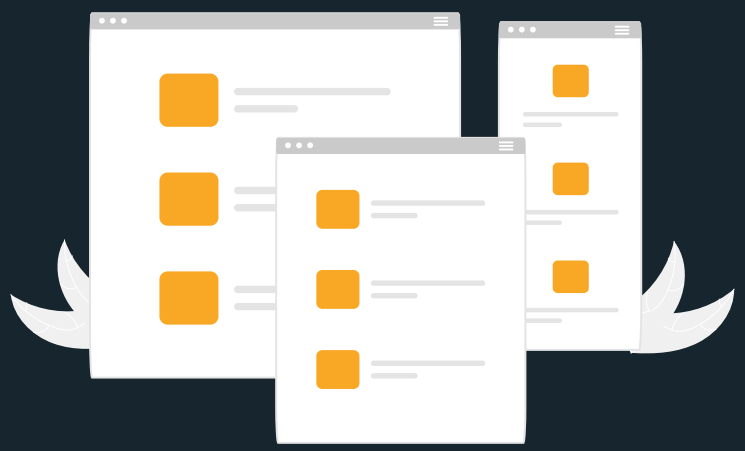


GraphQl – usaria *um* Gateway



Micro Serviços

Micro serviços - Vantagens



1. Heterogeneidade de tecnologia
2. Equipes independentes e focadas
3. Robustez
4. Escalabilidade
5. Facilidade de implantação
6. Alinhamento organizacional
7. Composição

Micro Serviços

Reflexões ...

Não menospreze o uso de monolitos.

Começarmos projetos com monolitos pode ser uma boa opção. Pode ser uma excelente opção para validar uma hipótese ou para Startups em novos projetos.

São mais fáceis e pensar, manter e implantar. Pensar na concepção de um produto, nas diversas camadas, APIs, escalabilidade, suas plataformas, etc.

Micro Serviços

Reflexões ...

Fato este que , empresas com Shopify vem obtendo sucesso na modularização e até mesmo a Amazon (noticiou recentemente) que está revendo suas estruturas de Micro serviços para Monolito.

Importante ter estes conceitos de estilos arquiteturais, bem como, introdução de ferramentas como BFF ou GraphiQL, pois a arquitetura client-side vem ganhando complexidade e exigindo maior estruturação dos projetos

Aprofundar na Arquitetura Client-side

Micro frontends

Micro frontends

Introdução

Falamos até agora de estratégias de backend amplamente adotadas no mercado.

Mas como adaptá-las ao frontend?

O que deu errado ?

Micro frontends

O que são Micro Frontends?

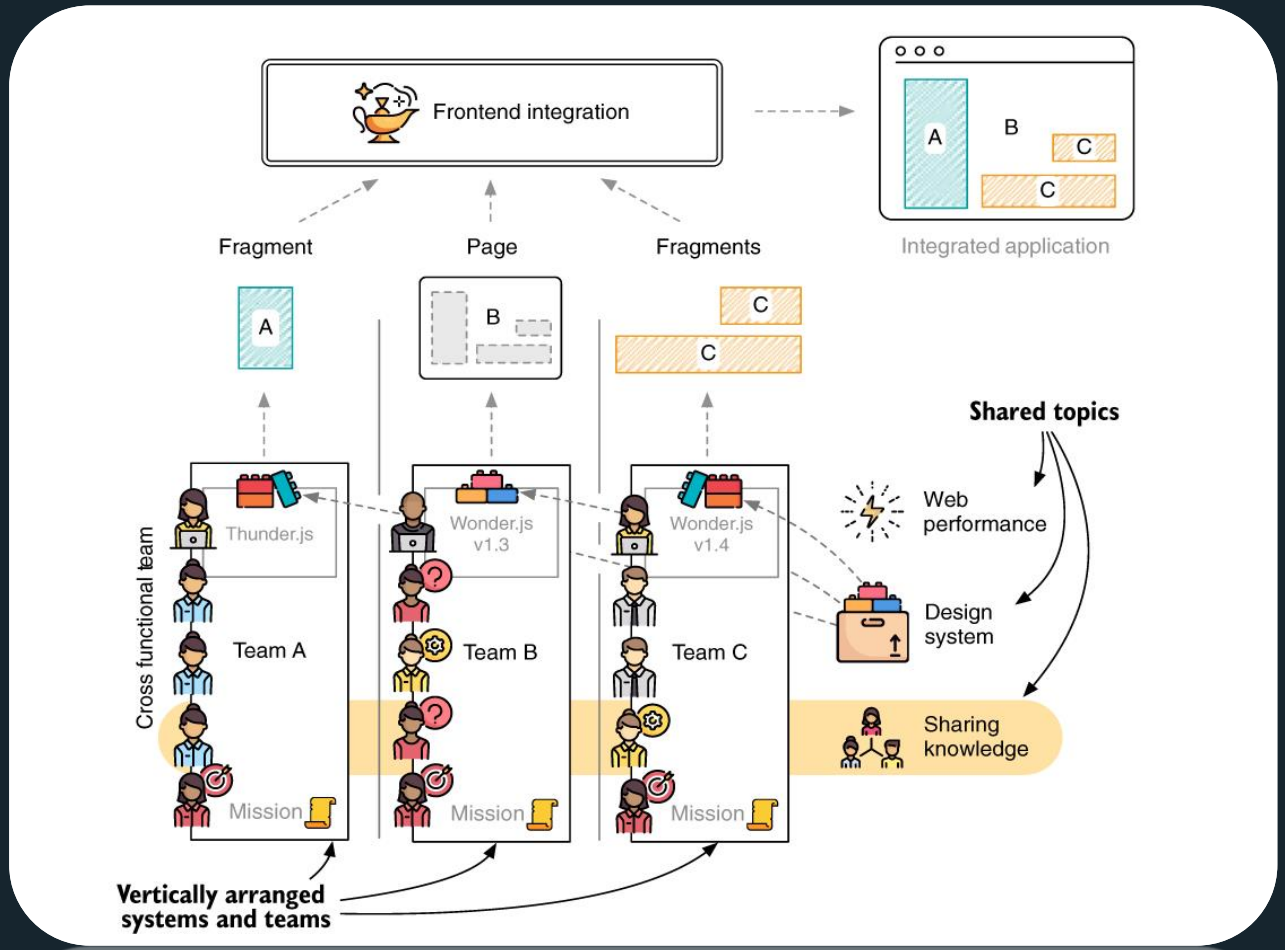
Os micro frontends é um estilo arquitetural que trata de dividir interfaces grandes em pedações menores e mais gerenciáveis com fronteiras entre eles.

O termo Micro frontends estende o conceito de micro serviços ao frontend. Um Micro frontend é para o frontend como o micro serviços é para o backend. Sua aplicação é agnóstica a tecnologia ou framework, se trata de uma estratégia.

Fonte: https://martinfowler.com/articles/micro-frontends.html

Micro frontends

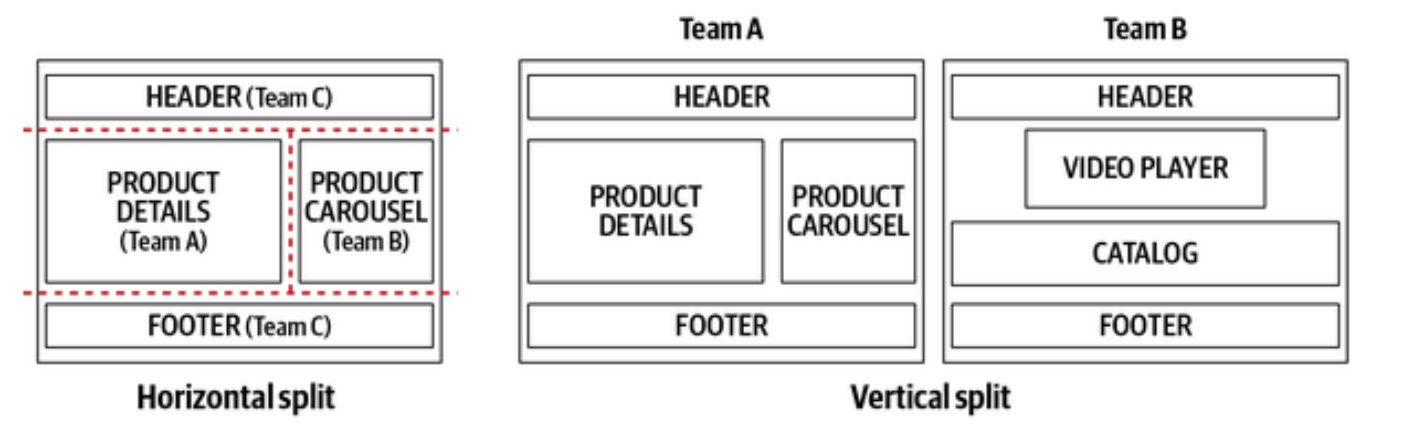
Os desafios de escolher a melhor abordagem



Fonte:
Geers, Michael. Micro Frontends in Action Manning Publications

Micro frontends

Luca Mezzalira arquiteto da DANZ, apresenta uma abordagem estratégica para que a organização se organize com base na escolha técnica da composição dos Micro Frontends. Em seu livro Building Micro-Frontends ele aborda o tema da decisão horizontal e vertical.

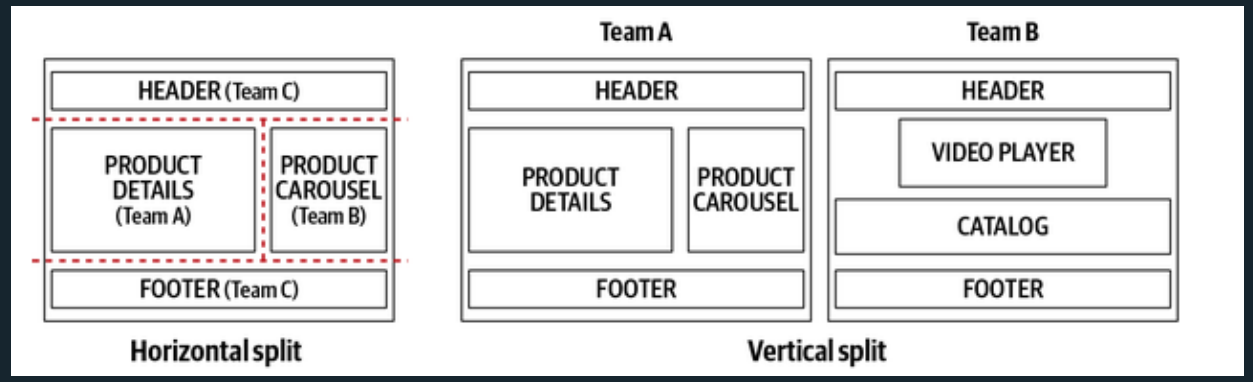


Fonte: https://learning.oreilly.com/library/view/building-micro-frontends/9781492082989/ch03.html

Micro frontends

1. **Horizontal** – a interface é composta por vários fragmentos (iframes ou web componentes) na mesma visualização. Várias equipes responsáveis por partes da exibição.

2. **Vertical** – cada equipe é responsável por um sub-domínio e negócios, como a catálogo, autenticação , etc.

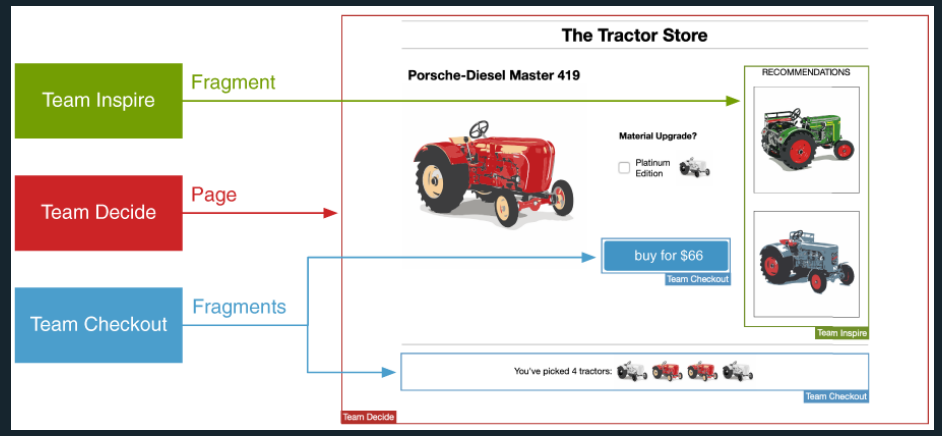


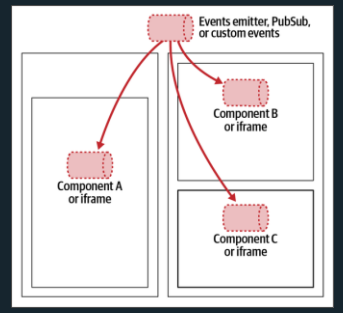
Fonte: https://learning.oreilly.com/library/view/building-micro-frontends/9781492082989/ch03.html

Micro frontends

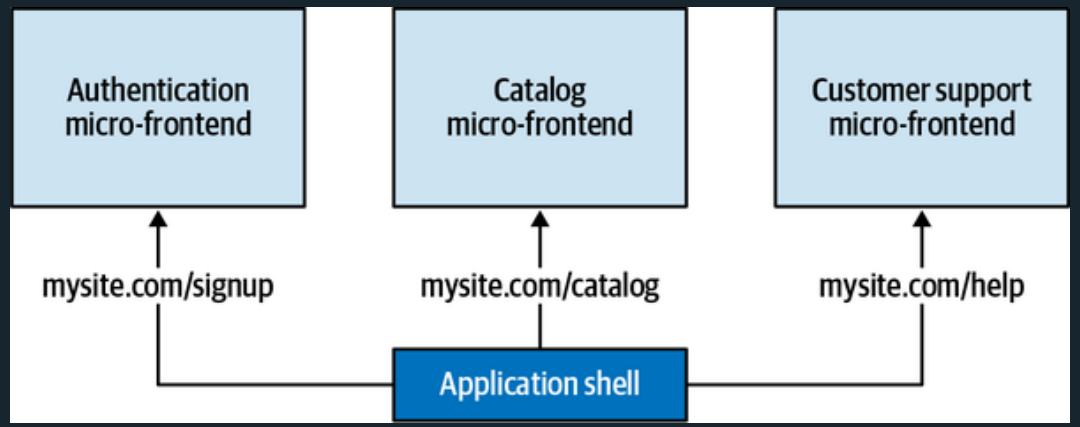
Micro frontends - **Abordagem Horizontal x Vertical**

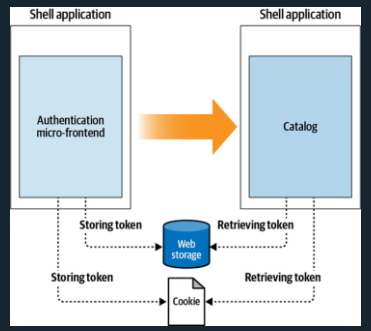
Horizontal





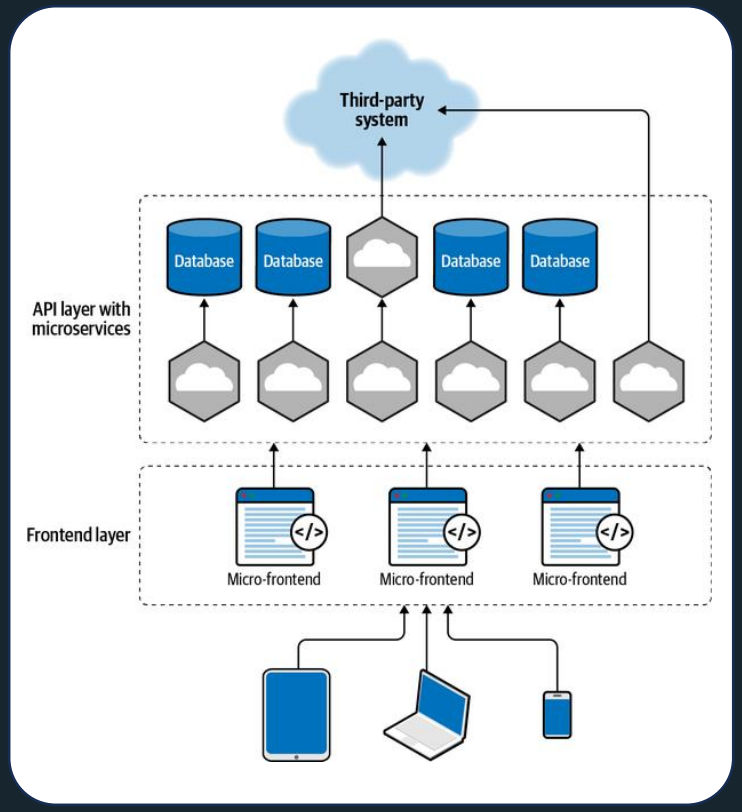
Vertical





Micro frontends

Micro frontends



Fonte:
Mezzalira, Luca . Building Micro-Frontends, O´Reilly Media

Porque adotar Micro frontends:

1. Otimizar o tempo de entrega de um recurso ao cliente
2. Times autônomos e interdisciplinares (Crossfunctions)
3. Organizações mais escaláveis
4. Bases de código menores, mais coesas e sustentáveis
5. A capacidade de atualizar, atualizar ou até mesmo reescrever partes do frontend de maneira mais incremental do que era possível anteriormente
6. Estratégia de testes A/B ou Blue-Green

Não é por acaso que essas vantagens principais são algumas das mesmas que os micros serviços podem oferecer.

Alguns desafios são:
estrutura, fronteiras, estado, comunicação e integração

Micro Frontends

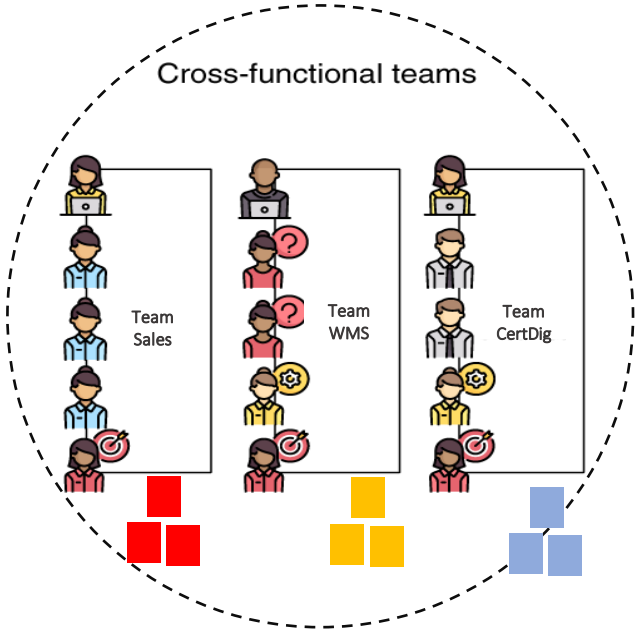
Agora que conceituado vamos fazer um “gancho” ao nosso escopo de micro serviços.

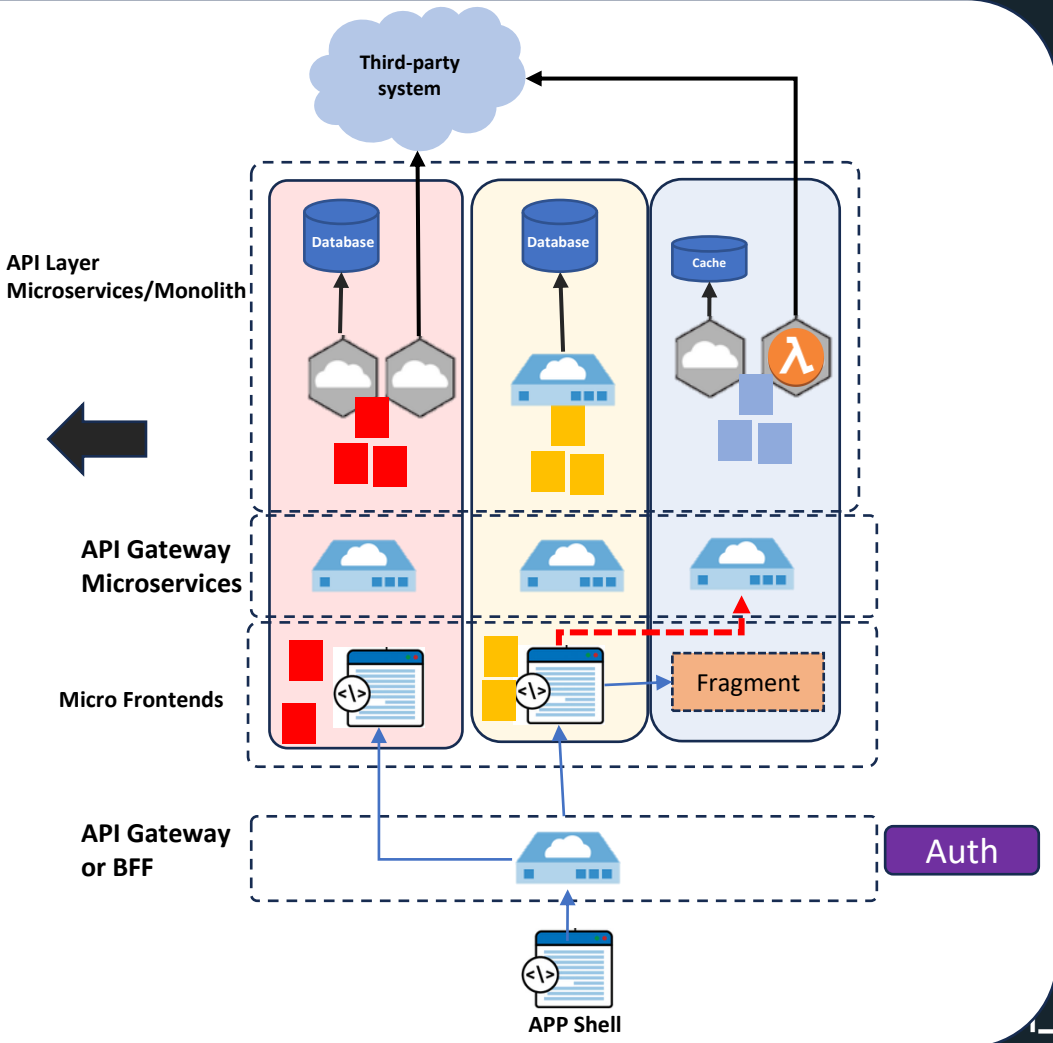
Micro frontends

Micro Frontends – Estrutura

Neste caso compilamos uma estrutura horizontal onde as equipes tomam conta do seus ativos de sub-domínio não apenas dos seus micro serviços mas também dos seus micro frontends..

A equipe de assinatura digital oferece serviço mas também permite inserir um Webcomponents para assinatura com um botão apenas.

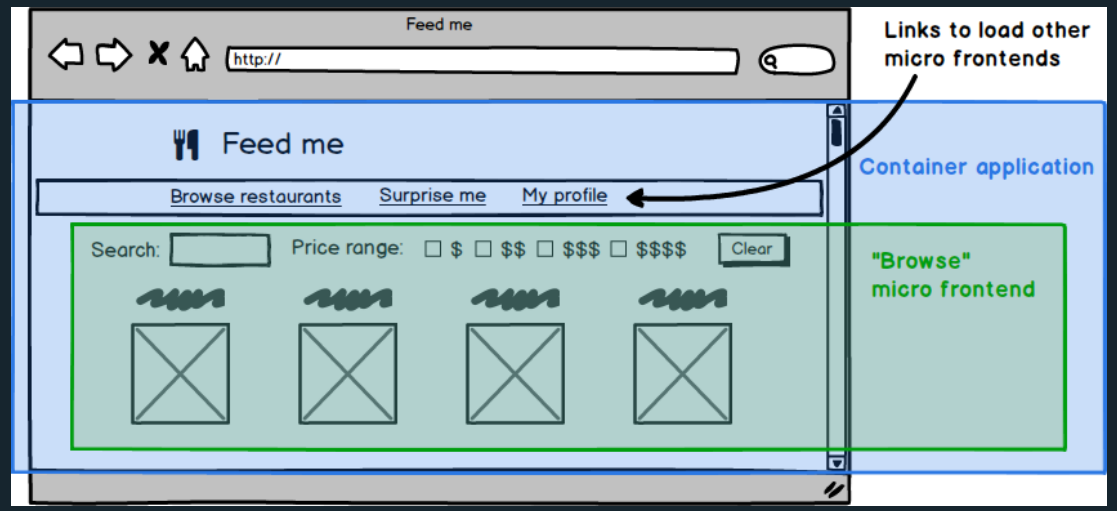




Micro frontends

Micro Frontends

Vamos ver na prática um exemplo de micro frontends



Neste exemplo, observaremos uma abordagem por Javascript chamando fragmentos de telas

Para o AppShell seria o container

- https://demo.microfrontends.com /
- https://github.com/micro-frontends-demo

Micro frontends

Reflexões...

Ultimamente, estamos vendo cada vez mais atenção sendo dada à arquitetura geral e às estruturas organizacionais necessárias para o desenvolvimento web moderno e complexo.

Em particular, estamos vendo padrões surgindo para decompor monólitos de front-end em pedaços menores e mais simples, que podem ser desenvolvidos, testados e implantados independentemente, enquanto ainda aparecem para os clientes como um único produto coeso.

Micro frontends

Reflexões...

Os conceitos e ideias aqui descritos não são novos. Os micro frontends, por exemplo, são bastante populares no setor de comércio eletrônico.

A Amazon, Netflix e Spotify são exemplos interessantes.

Serviço de streaming de esportes DAZN também reconstruiu seu front-end monolítico como uma arquitetura de micro-frontends modular, trabalhando com o micro serviços por sub-domínio.

O caminho mais rápido e ao meu ver mais consistente seria uma abordagem horizontal utilizando o mapeamento de domínio e sub-domínio com ajuda do DDD.

Aprofundar na Arquitetura Client-side

Requisitos Arquiteturais

Requisitos Arquiteturais

Introdução

Um requisito arquitetural é aquele que afeta a arquitetura de um software.

São princípios e diretrizes que nos ajudam a criar interfaces de forma sólida, escaláveis e de fácil manutenção.

Requisitos Arquiteturais

Introdução

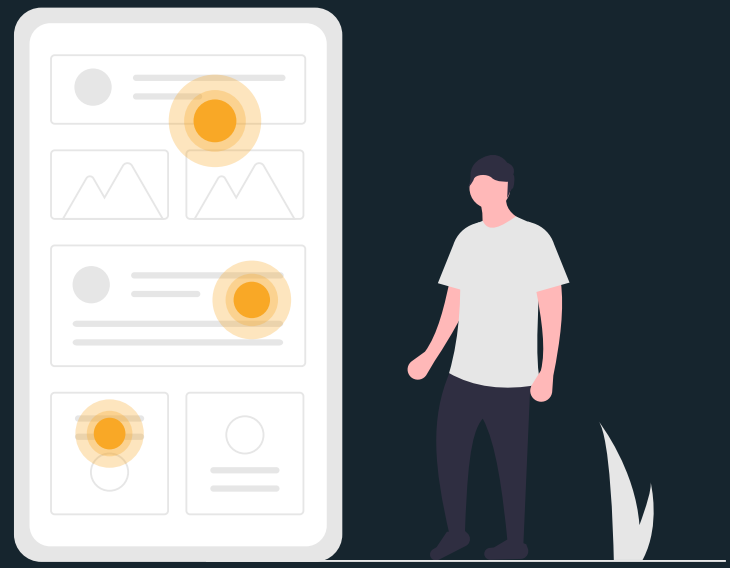
Muitas vezes tais requisitos **são negligenciados** durante o estudo arquitetural e construção do produto ampliando os riscos da aplicação.

Requisitos Arquiteturais

Requisitos importantes negligenciados

Vamos tentar explorar os principais requisitos ao criarmos produtos web:

1. Divisão de responsabilidades
2. Responsividade
3. Componentização
4. Gerenciamento de estado
5. Desempenho
6. Acessibilidade



Requisitos Arquiteturais

Requisito - Separation of Concerns (Separando responsabilidades)

Objetivo: Lida com a separação clara das diferentes responsabilidades dentro do sistema (apresentação, lógica de negócio e interações com serviços (internos e externos).

Descrição da negligência: Falta de separação clara das responsabilidades entre apresentação, lógica de negócios e interações externas.

Riscos: Código confuso, dificuldade de manutenção.

Soluções: Adotar princípios de design como SOLID combinamos com padrões de arquitetura, como MVP, MVC ou FLUX, e boas práticas de organização do código.

Requisitos Arquiteturais

Requisito - Separation of Concerns (Separando responsabilidades)

Clean Arquitecture – Arquitetura Limpa é um tipo de abordagem de divisão do projeto em camadas independentes.

A arquitetura limpa se apoia no SOLID para organizar estruturas de dados em classes e como tais classes devem ser interconectadas. No frontend você pode estender o conceito de classes para componentes também.

O objetivo dos princípios do SOLID é a criação de estruturas de software limpa que tolere mudanças, sejam fáceis de entender e sejam a base de componentes que possam ser usados em muitos sistemas de software.

O uso da palavra "classe" não implica que esses princípios sejam aplicáveis apenas a softwares orientados a objetos. Uma classe é apenas um agrupamento acoplado de funções e dados.

Fonte: Martin, Robert C.. Arquitetura Limpa (Robert C. Martin) (pp. 101-102). Alta Books. Edição do Kindle.

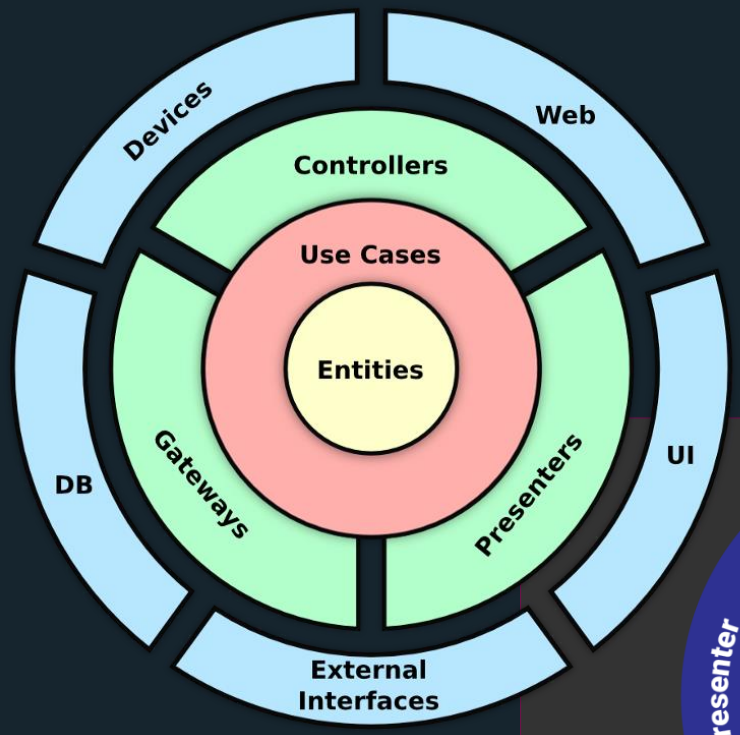
Requisitos Arquiteturais

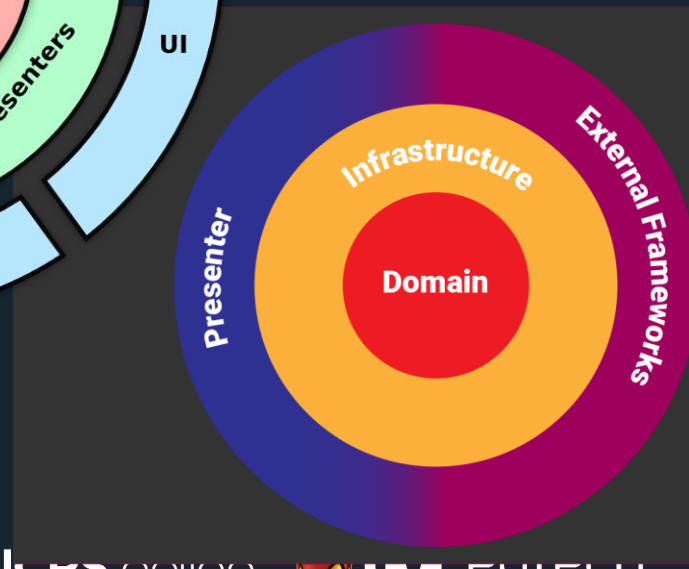
Requisito - Separation of Concerns (Separando responsabilidades)

Algumas iniciativas, buscam adaptar os princípios da arquitetura Limpa (clean architecture) a um tipo de framework e sua linguagem.

Um exemplo é o Clean Dart que visa uma proposta de arquitetura limpa para o Dart/Flutter

https://github.com/Flutterando/Clean-Dart
https://github.com/falsy/react-with-clean-architecture





Requisitos Arquiteturais

Requisito - Componentização

Objetivo: Lida com a busca de melhorar para distribuição, sustentação, isolamento, compartilhamento e reuso do código.

Descrição da negligência: Falta de componentização e reutilização de código.

Riscos: Duplicação de código, governança, baixa produtividade de desenvolvimento, baixa qualidade nos testes.

Soluções: Adotar um sistema de componentes reutilizáveis, seguir um dos princípios de design DRY (Don't Repeat Yourself).

Requisitos Arquiteturais

Requisito - Componentização

CSS Modernos – Estratégia de Encapsulamento

Com desenvolvimento cada vez mais complexo, com diversos particularidades, algumas estratégias ajudam na produtividade do desenvolvimento. Facilitando a distribuição, sustentação, compartilhamento e reuso do código fonte com práticas modernas.

Requisitos Arquiteturais

Requisito - Componentização

CSS-Modules – Isolando folhas de estilos.

CSS Modules é uma técnica que permite o escopo local de estilos em um aplicativo da web, evitando o conflito destes entre diferentes componentes. Ele é amplamente utilizado em frameworks como React e Angular para modularizar o CSS e fornecer uma abordagem mais organizada para o desenvolvimento.

Contexto de uso:

1. Muitas equipes compartilhando projeto, estilos ou fragmentos.
2. Quando necessário desacoplamento de interfaces seja com microfrontends ou equipes modulares.
3. Não gerarmos conflitos entre projetos que utilizem mesmo nome de classe entre eles
4. Não quisermos que o estilo “vaze” além do seu escopo de projeto

Sendo assim, mesmo tendo nomes de classes comuns em diferentes partes do projeto mantidos por diversas equipes não precisará se preocupar com conflitos de estilo. Cada componente terá seu próprio identificador único.

Requisitos Arquiteturais

Requisito - Componentização

Como funciona : Ao termos o código "buildado" ferramentas de bundlers como webpack irá gerar nomes de classes do seu CSS de forma única para cada componente ajuda na localização do nosso escopo.

Identifica no React que utilizará CSS Modules
react-scripts@2.0.0

Button.module.css

```
.error {
  background-color: red;
}
```

Button.jsx

```
import React, { Component } from 'react';
import styles from './Button.module.css';
import './another-stylesheet.css';

class Button extends Component {
  render() {
    // reference as a js object
    return <button className={styles.error}>Error Button</button>;
  }
}
```

Definindo css local

Referência ao CSS global por exemplo

Referência ao estilo local

Resultado observado na página web

```
<button class="Button_error_ax7yz">Error Button</button>
```

Button.css

```
.error{
  background-color: red;
}
```

Button.ts

```
import { Component } from '@angular/core';

@Component({
selector: 'app-button',
templateUrl: './button.component.html',
styleUrls: ['./button.css']
})
export class ButtonComponent {
// codigo typescript ...
}
```

Decorando o typescript com referências

Para informar css local utilizar opção "styleUrls

Usando botão estilizado no html local

```
<button class="error">Error Button</button>
```

Requisitos Arquiteturais

Requisito - Componentização

Styles Compomentes – Criando componentes de estilos

Styled Compomentes ou Componentes estilizados, como CSS Modules, tem objetivo de modularizar e encapsular os estilos. Mas de uma forma diferente criando componentes.

Style componentes é uma biblioteca em React permitindo escrever CSS diretos nos arquivos JS (CSS-in-JS). Para esta abordagem fazemos uso da biblioteca 'styled-components' ,

Aderir aos estilos componentizados depende do contexto do projeto e do time. O importante neste caso é que os desenvolvedores possam ter independências para escrever seus estilos sem que atrapalhem uns aos outros.

Requisitos Arquiteturais

Requisito - Componentização

React
Definindo componente StyledCard usando JavaScript

```
import React from 'react';
import styled from 'styled-components';

const StyledCard = styled.div`
  border-radius: 10px;
  box-shadow: 0px 2px 5px rgba(0, 0, 0, 0.2);
  padding: 10px;
  background-color: ${props => props.isActive ? 'lightblue' : 'initial'};
`;

const Card = ({ isActive }) => {
  return
    <StyledCard isActive={isActive}>
      {/* Conteúdo do cartão */}
    </StyledCard>;
};

export default Card;
```

Usando biblioteca

Passando propos.

Uso e condição e arrow function

Usando objeto

Conseguimos utilizar abordagem de styled componentes através do conceito de viewencapsulation.

Entretanto, como o framework usa Typescript e possui muitos recursos de estilos internos. Cabe como melhor prática usar recursos do próprio Angular como ngClass. para passar propriedades para estilos no html..

```
@Component({
  selector: 'app-card',
  templateUrl: './card.component.html',
  styleUrls: ['./card.component.css']
})
export class CardComponent {
  @Input() isActive: boolean;
}
```

```
<div class="card" [ngClass]="{'active': isActive}">
  <!-- Conteúdo do cartão -->
</div>
```

Exemplo:
https://github.com/bradtraversy/huddle_styled_components/tree/main/src/components

Requisitos Arquiteturais

Requisito - Componentização

WEB Compoments – Criando componentes Web Isolados

Web Components são uma especificação do W3C (World Wide Web Consortium) que permite criar componentes reutilizáveis e encapsulados em novas tags HTML.

Componentes personalizados ou widgets, funcionarão em navegadores modernos e podem ser usados com qualquer biblioteca ou estendido facilmente com novos elementos com estilo encapsulado e comportamento personalizado.

Requisitos Arquiteturais

Requisito - Componentização

Os componentes web se apresentam como uma grande oportunidade para isolamento de elementos de frontends. Todas os principais frameworks de interface do usuário, como React, Angular e Vue, são capazes de gerar componentes da Web.

Para utilizar Web Components em Angular ou React, geralmente não é necessário instalar nada adicional. Tanto Angular quanto React têm suporte nativo para a criação e uso de Web Components. No entanto, é importante considerar o suporte aos Web Components em navegadores mais antigos. Nesses casos, é recomendável o uso de polyfills.

Polyfills: Um polyfill é uma biblioteca que deve ser instalada no framework que ao importar no projeto serve de middleware que implementa recursos em navegadores que não os suportam nativamente. Eles permitem que você utilize recursos mais recentes em navegadores mais antigos, preenchendo as lacunas de compatibilidade.

Requisitos Arquiteturais

WebComponents

React

```
import Child from './Child';
import { createRoot, Root } from 'react-dom/client';

class ChildWebComponent extends HTMLElement {
    static get observedAttributes() {
        return ['title']
    }
    mountPoint!: HTMLSpanElement;
    root!: Root;

render() {
        const title = this.getAttribute('title') || 'child';
        if(!this.root) this.root = createRoot(this.mountPoint);
        this.root.render(<Child title={title} />);
    }

connectedCallback() {
        this.mountPoint = document.createElement('span');
        this.attachShadow({ mode: 'open'
}).appendChild(this.mountPoint);

        this.render();
    }

attributeChangedCallback() {
        this.render();
    }
}

export default ChildWebComponent;

customElements.define('react-child', ChildWebComponent);
```

Angular

projects/ng-root/src/index.html

```
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/x-icon" href="favicon.ico">
<script src="http://localhost:8081/child.js"></script>
```

projects/ng-root/src/app/app.component.html

```
<ng-child *ngIf="show" [title]="title"></ng-child>
<react-child *ngIf="show" [title]="title"></react-child>
```

Requisitos Arquiteturais

Gerenciamento de Estado

Objetivo: Lida com a complexidade do gerenciamento de estado. À medida que a aplicação cresce, a falta de um sistema de gerenciamento de estado adequado pode gerar diversos problemas.

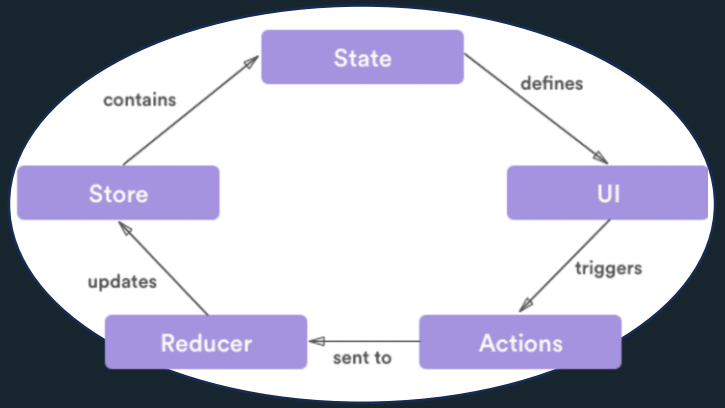
Descrição da negligência: Ignorar a complexidade do gerenciamento de estado em aplicações frontend pode resultar em problemas de integridade visual, informações imprecisas, código confuso e de difícil depuração.

Riscos: Informações sendo apresentadas incorretamente, código confuso, difícil de depurar.

Soluções: Utilizar bibliotecas ou frameworks especializados, como Redux ou Vuex, e seguir as melhores práticas recomendadas.

Requisitos Arquiteturais

Gerenciamento de Estado



Um estado representa uma parte de um componente no JavaScript. Sempre que há interação do usuário com sua aplicação existirá um novo estado para aplicação, e se necessário uma nova interface do usuário deve ser renderizada para refletir as modificações.

O gerenciamento de estado refere-se à prática de gerenciar tais os estados da aplicação e seus componentes. Para realizar o gerenciamento de estado.

Você pode pensar em estado como qualquer informação em sua IU que muda ao longo do tempo, geralmente acionada pela interação do usuário.

Requisitos Arquiteturais

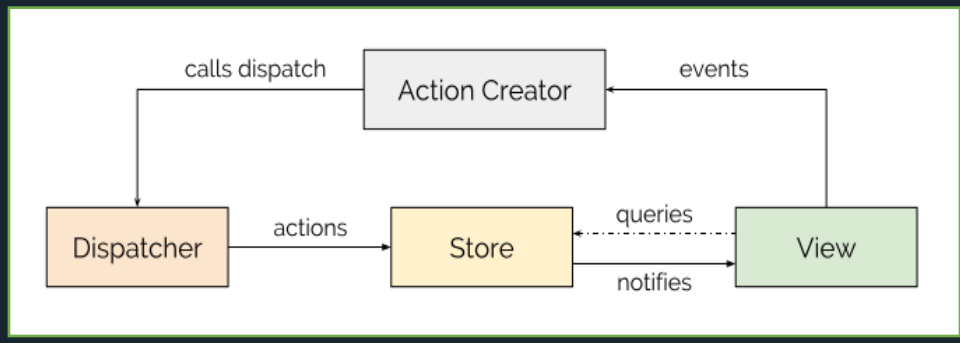
Gerenciamento de Estado

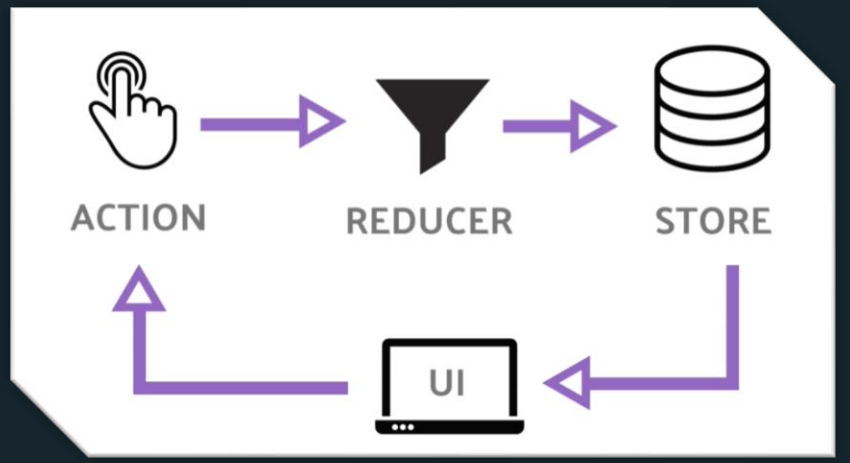
Redux- Gerenciando estados

Redux ajuda a escrever aplicativos com comportamento estável, executados em diferentes ambientes (cliente,seridor ou nativo).

Redux utiliza o **padrão arquitetural Flux**, mantendo a lógica e estado encapsulado na sua aplicção, facilitando a manutenção dos dados e testes.

Redux possui uma extensão no DevTools permitindo debugar e analisar o estado da aplicação.





Para usar Redux em React, deve-se fazer uso da biblioteca React-Redux.

Requisitos Arquiteturais

Performance

Objetivo: Lida com otimização de desempenho da aplicação

Descrição da negligência: Não avaliar padrões e abordagens de renderização e desenvolvimento ideal para o contexto.

Riscos: Tempos de carregamento longos, experiência do usuário insatisfatória.

Soluções: Realizar testes de desempenho, aplicar técnicas de otimização, como minificação e compressão de recursos (como imagens). Otimizar uso de CDNs e caches, etc

Requisitos Arquiteturais

Performance

Alguns itens relevantes a serem considerados na sua análise de performance no frontend:

Tamanho da página (Page Size): O tamanho da página se refere ao tamanho total dos recursos (HTML, CSS, JavaScript, imagens, etc.) que são baixados para exibir uma página.

Time to First Byte (TTFB): é o tempo desde o início da navegação até o primeiro byte de dados retornado ao cliente. É a nossa primeira etapa na lista de verificação de desempenho da Web

Requisitos Arquiteturais

Performance

Alguns itens relevantes a serem considerados:

Tempo de carregamento da página (Page Load Time): tempo de carregamento da página se refere ao tempo necessário para que uma página seja totalmente carregada no navegador.

Métricas Core Web Vitals: conjunto de métricas, recentes (2020), recomendadas pelo Google para medir a qualidade da experiência do usuário em sites. Estas métricas estudam a First Contentful Paint (FCP) e o Largest Contentful Paint (LCP) que fornecem insights sobre quando o conteúdo começa a ser exibido na tela e quando o maior elemento visível é renderizado, respectivamente.

https://httparchive.org/reports/state-of-the-web

Requisitos Arquiteturais

Performance

Algumas ferramentas que auxiliam no estudo do desempenho

- Chrome DevTools = O Chrome DevTools fornece análises detalhadas sobre tudo o que acontece enquanto sua página é carregada ou executada. Consulte Introdução à análise do desempenho do tempo de execução para se familiarizar com a IU do painel Desempenho.
  - https://developer.chrome.com/docs/devtools/

- WebPageTest = O WebPageTest é uma ferramenta de desempenho da web que usa navegadores reais para acessar páginas e coletar métricas de tempo.

  - https://developer.chrome.com/docs/lighthouse/overview/

Requisitos Arquiteturais

Performance

Algumas ferramentas que auxiliam no estudo do desempenho

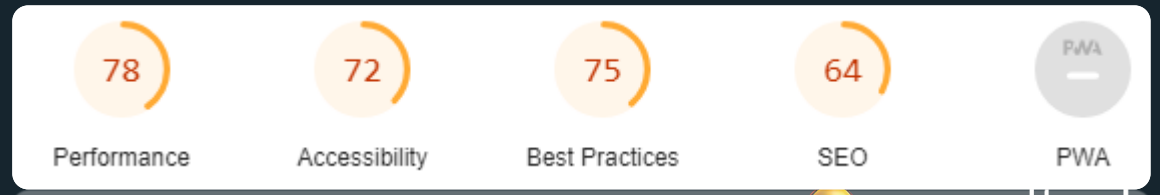
- Lighthouse =O Lighthouse é uma ferramenta de auditoria automatizada de código aberto para melhorar o desempenho, a qualidade e a correção de seus aplicativos da web.
  - https://developer.chrome.com/docs/lighthouse/overview/

Requisitos Arquiteturais

Performance

Dicas para melhorar a performance do site

- Implantação e otimização de redes de entrega de conteúdo CDN
- Implementar técnicas como pré-busca (pre-fetching), pré-renderização (pre-rendering).
- Utilizar-se de protocolos mais avançados HTTP/2 e HTTP/3, que oferecem melhorias no carregamento de recursos e redução da latência
- Otimizar, dimensionar e utilizar de formatos mais otimizados de imagens
- Fazer uso de minificação de arquivos (Javascript e css)

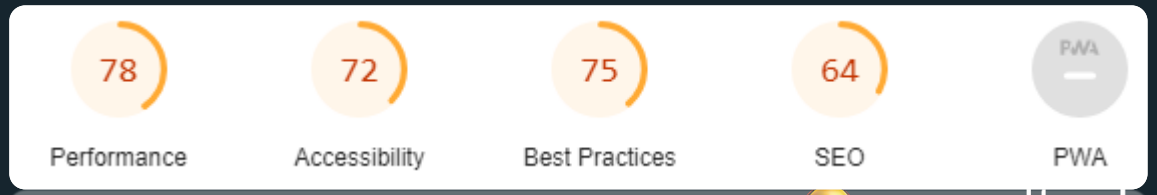


Requisitos Arquiteturais

Performance

Dicas para melhorar a performance do site

- Retirar códigos de bibliotecas inseridas desnecessariamente (imports) e evitar excesso de dependências.
- Fazer sempre que possível uso de carregamento assíncrono de recursos.
- Otimizar estrutura sistema utilizando-se de CSS-in-JS para otimizar carregamento sob-demanda (não baixar css “cheio”)
- Fazer uso apropriado de cache para os recursos estáticos, como imagens, CSS e JavaScript, Configurar servidores para utilizarem recursos.



Requisitos Arquiteturais

Acessibilidade

Objetivo: A acessibilidade trata de tornar o site ou aplicativo utilizável por pessoas com deficiências, incluindo deficiência visual, auditiva, motora ou cognitiva. Isso envolve fornecer alternativas para conteúdo não textual, cores contrastantes, navegação via teclado, etc.

Descrição da negligência: frequentemente negligenciada, deixando de atender às necessidades de uma parcela significativa de usuários. Isso pode levar a uma experiência excludente para pessoas com deficiências.

Riscos: exclusão social, prejudicar usabilidade, prejudicar SEO

Soluções: É necessário adotar práticas de desenvolvimento inclusivas, seguindo as diretrizes de acessibilidade, como as estabelecidas pelas WCAG (Web Content Accessibility Guidelines). Realizar testes com tecnologias assistivas e garantir a compatibilidade com leitores de tela são passos importantes para melhorar a acessibilidade.

Requisitos Arquiteturais

Acessibilidade

Requisitos Arquiteturais

Acessibilidade

Algumas informações relevantes:

- 27% da população on-line global está usando a pesquisa por voz no celular .
- 85% dos vídeos do Facebook são assistidos sem som .
- Quando você faz uma pergunta a assistentes de voz como Siri, Alexa e Cortana, eles normalmente leem a resposta em uma página da Web usando a tecnologia de leitor de tela que existe desde que os computadores pessoais existem .
- Atualmente acessibilidade expande fronteiras da comunidade de deficientes ela permite auxiliar qualquer o usuário, por exemplo quando estiver no silencioso.
- Em análise feita pelo httparchive.org observamos que apenas 12% dos sites em 2022 atendiam requisitos de acessibilidade. (https://almanac.httparchive.org/en/2022/accessibility#introduction)

Requisitos Arquiteturais

Acessibilidade

Algumas informações relevantes:

WCAG (Web Content Accessibility Guidelines). definem a forma de como tornar o conteúdo da Web mais acessível para pessoas com deficiência

https://www.w3.org/WAI/fundamentals/accessibility-intro/

ARIA: definem as formas de tornar o conteúdo e as aplicações da Rede Mundial - *Web* - (especialmente aqueles desenvolvidos com Ajax e JavaScript) mais acessíveis às pessoas com deficiência.

https://developer.mozilla.org/pt-BR/docs/Web/Accessibility/ARIA

Requisitos Arquiteturais

Acessibilidade

Dicas de acessibilidade

- Utilize o Lighthouse para criticar e apontar sugestões.
- Adote padrões contidos e sugeridos pela WCAG
  - Um guia de acessibilidade por ser visto aqui: https://www.smashingmagazine.com/2021/03/complete-guide-accessible-front-end-components/
- Utilize de frameworks como material.io pois seis componentes já possuem algum tipo de tratamento para acessibilidade.
  - https://m3.material.io/foundations/accessible-design/overview
- Utilize o relatório da HttpArchive.org para acompanhar tendências e sugestões.
  - **https://almanac.httparchive.org/en/2022/accessibility**

Requisitos Arquiteturais

Responsividade - RWD

Objetivo: A responsividade refere-se à capacidade de um site ou aplicativo se adaptar a diferentes tamanhos de tela e dispositivos, proporcionando uma experiência de usuário consistente.

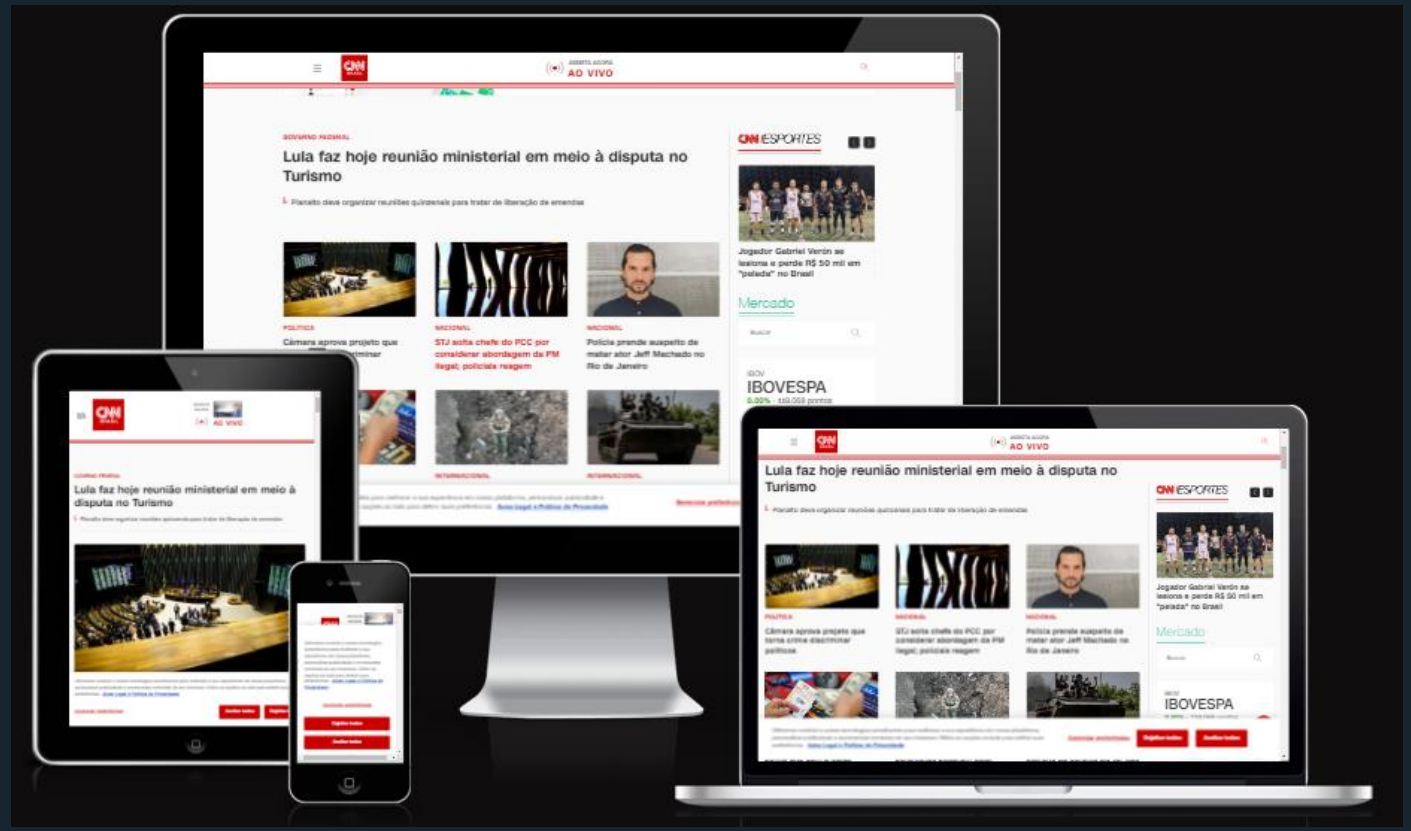
Descrição da negligência: Muitas vezes, os desenvolvedores não dedicam tempo suficiente para garantir que seus projetos sejam responsivos. Como resultado, a interface pode ficar distorcida ou não utilizável em dispositivos móveis ou em telas de tamanhos diferentes.

Riscos: quebra interface. experiência usuário, informações inconsistentes

Soluções: A solução envolve a adoção de design responsivo, usando técnicas como layouts flexíveis, media queries e componentes adaptáveis. É importante testar o projeto em uma variedade de dispositivos e tamanhos de tela para garantir uma experiência consistente.

Requisitos Arquiteturais

Responsividade



Vamos avaliar
https://www.cnnbrasil.com.br/
https://ui.dev/amiresponsive

Requisitos Arquiteturais

Responsividade - RWD

- **Recursos básicos essenciais:**

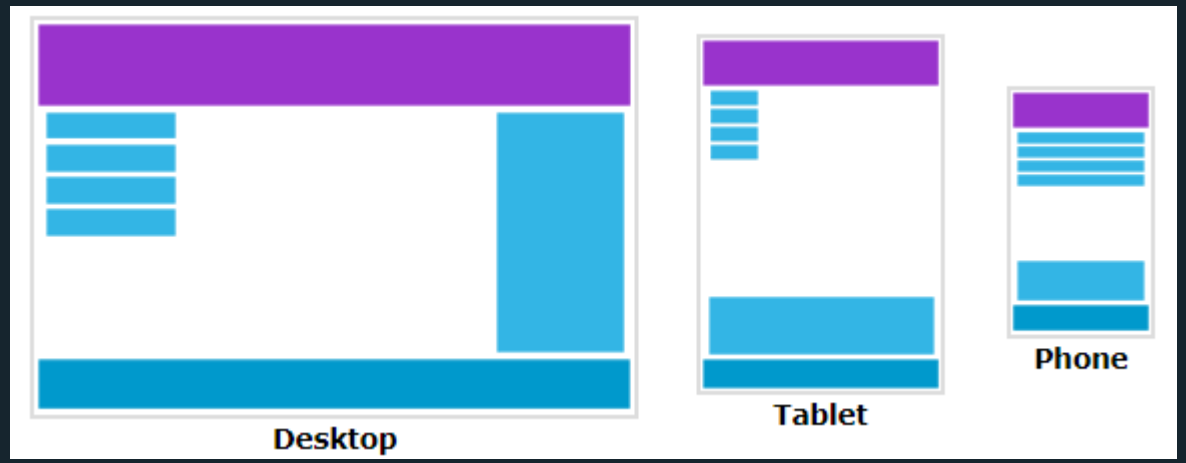
  - **Media queries = as media queries são filtros que podem ser plicados aso estilos CSS com base em alguma característica do dispositivo onde a pagina apresentada**
  - **Ajuste do Viewport =**A viewport é a área visível do usuário de uma página da web.
  - **Layout flexíveis (flexbox e grid)**
  - **Alguns frameworks :** Bootstrap, Foundation**,** W3.CSS

Requisitos Arquiteturais

Responsividade - RWD

Media queries

- Ela usa a referência “@media” regra para incluir um bloco de propriedades CSS somente se uma determinada condição for verdadeira.
- Importante uso do medias queries seriam definir pontos de interrupções das interfaces
- Outro fato relevante seria definir dimensões variáveis aos elementos de acordo com tamanho de telas diferentes aos tipos de tela aos estilos (Fontes, imagens, vídeos, etc)
- Vamos ver na prática




```css
/* Extra small devices (phones, 600px and down) */
@media only screen and (max-width: 600px) {
  .example {background: red;
  font-size: 20px;
  }
}

/* Small devices (portrait tablets and large phones, 600px and up) */
@media only screen and (min-width: 600px) {
  .example {background: green;
  font-size: 40px}
}

/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {
  .example {background: blue;
  font-size: 80px}
}

/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {
  .example {background: orange;
  font-size: 120px}
}

/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {
  .example {background: pink;
  font-size: 160px}
}
```

```css
img {
  width: 100%;
  height: auto;
}
```

```css
video {
  width: 100%;
  height: auto;
}
```

Requisitos Arquiteturais

Responsividade - RWD

Viewport

- Isso fornece ao navegador instruções sobre como controlar as dimensões e o dimensionamento da página.
- Afeta conteúdo e acessibilidade (inclusive zoom)

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Requisitos Arquiteturais

Responsividade - RWD

Dicas:

1. **Comece seu projeto com mentalidade de layout responsivo**
2. **Caso precise converter um projeto, não comece querendo converter tudo. Escolha uma funcionalidade ou um design específico**
    - **Comece com grande impacto pelo menor esforço. Primeiro com estilos globais. Isto permitirá um avanço considerável sem muito esforço na mudança do layout em si: tipografia, cores, planos de fundo, defina percentis de tamanhos máximo de imagens. Estas ações ajudam a gerar impacto mas ainda pode deixar layout comprometidos mas já habilitados**
3. **Evite tamanhos fixos de largura e altura nas propriedades do css. E já comece com isto!**
4. **Faça para mobile-first**

Requisitos Arquiteturais

Responsividade - RWD

Dicas:

1. **Defina media queries para melhorar e adaptar o layout para diversas plataformas.**
2. **Normalmente os layouts começam a quebrar com 40em de “with”, dependendo do contexto. Considere como uma dica para ajuda-lo.**
3. **Evite muitos pontos de interrupções min e max diferentes no seu projeto. Caso o designer tenha ajustes tente manter um padrão. Apenas mude o ponto de interrupção se for estritamente necessário. Talvez 2 ou 3 pontos seja mais que suficiente.**
4. **Não construa seus layouts olhando para dispositivos ou telas específicas. Pelos inúmeros dispositivos no mercado inclusive versões diferentes no mesmo dispositivo isto pode ser contra-produtivo.**
    - **Tenha sempre alinhado como design resoluções mínimas e máximas e trabalhe na responsividade nestes momentos.**

Requisitos Arquiteturais

Responsividade - RWD

Dicas:

- Responsive Web Design
  - https://web.dev/responsive-web-design-basics/
- Introdução ao Design Responsivo
  - https://www.w3schools.com/css/css_rwd_intro.asp
- **Guia para web design responsivo em 2023**
  - **https://webflow.com/blog/responsive-web-design**
- **Porque seu site precisa ser otimizado para dispositivos móveis.**
  - **https://rockcontent.com/blog/mobile-friendly/**
- Framework W3school
  - https://www.w3schools.com/html/html_responsive.asp

Aprofundar na Arquitetura Client-side

Tendências

Tendências

Low Code

Low code é uma abordagem de desenvolvimento de software que permite criar aplicativos com o mínimo de codificação manual.

•Objetivos do Low Code é acelerar o desenvolvimento de aplicativos, reduzir a dependência de desenvolvedores especializados e promover a colaboração entre desenvolvedores e usuários de negócios.

•Low Code oferece maior produtividade, redução de custos, maior agilidade na entrega de projetos e maior capacidade de adaptação às mudanças de requisitos.

Tendências

Low Code

Exemplos de ferramentas populares:

OutSystems: Uma plataforma líder de desenvolvimento low code que oferece recursos avançados para criação de aplicativos web e móveis.

- https://www.outsystems.com/schedule-demo

Microsoft Power Apps: Uma ferramenta da Microsoft que permite criar aplicativos empresariais de forma rápida e simples, usando uma interface intuitiva.
https://powerapps.microsoft.com/pt-br

AWS Honey code : permite criar aplicativos personalizados sem a necessidade de codificação manual intensiva. Ele usa uma abordagem visual e baseada em planilhas para criar interfaces de usuário, lógica de negócios e automações.
https://www.honeycode.aws

Tendências

Low Code

Reflexão

As ferramentas no code / low code, são boas ferramentas mas como “ferramentas” existem limitações. Elas propõe muitas possibilidades e até permitem acoplar códigos externos, ou acessar serviços web, permitindo estender e ampliar seu uso.

Mas em qual momento faz sentido aderi-las ?

https://www.youtube.com/watch?v=jhoa6QMQAe4

Tendências

Inteligência Artificial aplicado a arquitetura Client-side

Na melhor em introduzirmos o tópico de tendências através de um meme que vem percorrendo as redes sociais após o Google I/O de 2023...

Então não podemos deixar de comentar sobre IA concordam ?

Tendências

Inteligência Artificial aplicado a arquitetura Client-side

Chatbots

Chatbots são programas de computador que utilizam inteligência artificial utilizando-se de linguagem natural para interagir e conversar com seres humanos de forma automatizada. Eles são projetados para simular uma conversa natural por meio de mensagens de texto, voz ou outros meios de comunicação.

Os chatbos vem sendo adotados por empresas para interfaciar o atendimento ao cliente, fornecer suporte técnico, realizar vendas, etc. **Ao falarmos de chatbot precisamos ter em mente o escopo das intenções.**

Eles são orientados a intenções e eventos.

Brokers ajudam na construção e são bem úteis como Twillio e TakeBlip que abstraem as APIs dos aplicativos conversacionais e interligam com sistemas de IA para linguagem natural.

Tendências

Inteligência Artificial aplicado a arquitetura Client-side

Os LLMs (large language models), são modelos de aprendizado de máquina (*machine learning*) que utilizam algoritmos de aprendizado para processar e entender a linguagem dos seres humanos. Eles são treinados com imensas quantidades de dados para aprender padrões de linguagem de modo a conseguirem desempenhar algumas funções.

São compostos por múltiplas camadas de redes neurais, que trabalham em conjunto para analisar textos e prever o que vem em seguida.

Uma aplicação muito comum na arquitetura client-side são na construção de chatbots e no desenvolvimento com ChatGpt, CoPilot, CodeWhisperer e Duet.

Tendências

Inteligência Artificial aplicado a arquitetura Client-side

Ferramentas de LLM como Chatgpt, Co-pilot, CodeWhisperer ou Duet são assistentes voltados ao desenvolvimento, elas vem nos ajudando ou apoiando na construção de código fontes.

- Tais ferramentas podem nos ajudar a economizar tempo, aumentar a produtividade e reduzir erros, aproveitando as sugestões de código fornecidas pela ferramenta. Ele pode sugerir soluções para problemas comuns, auxiliar na implementação de padrões de design, fazer críticas, fornecer código úteis e muito mais. Através de Snippets.

Tendências

IA no desenvolvimento

ChatGpt = criado pela OpenIA, é um chat LLM de linguagem natural que usa rede neural para responder questionamentos diversos inclusive de desenvolvimento. Não é voltada pra desenvolvimento

GitHub Copilot X= é um assistente de programação da Github utilizando ChatGpt da OpenAI, desenvolvedora do chatgpt "plugado" no VSCode fornece sugestões de códigos com

CodeWhisperer = criado pela AWS, sua funcionalidade é semelhante ao co-pilot oferecendo códigos. O diferencial desta ferramenta está ajudar na implementação de chamadas de APIs em serviços populares da AWS, bem como, validar se seu código atende o AWS Well-Architecte d atendendo as práticas recomentadas pela AWS.

Tendências

IA no desenvolvimento

As ferramentas IA introduziram uma nova era, mas traz com elas novos desafios.

Não obstantes, debates sobre a legalidade do uso de códigos proprietários já começaram.

IA no desenvolvimento

Conclusão

Ainda temos muito para evoluir , mas é certo que este caminho não tem volta, existem muitos benefícios e como são tecnologias disruptivas precisamos agora precisamos tratar exceções e ajustar as leis para poder limita a liberdade de uso de algumas funções.

Importante salientar que tais ferramentas são treinadas constantemente atualizando seu conhecimento.

E podem ser tendenciosas apresentando conteúdos inseguros. maliciosos e com aplicação sensível no seu produto. Sempre que for inserir um código entenda seu contexto e objetivos e alinhe ao seu negócio.

Aprofundar na Arquitetura Client-side

Conclusão

Conclusão

Depois de tudo que aprendemos aqui observamos claramente uma coisa....

Conclusão

SOFTWARE MUDAM E ISSO SIGNIFICA QUE PRECISAMOS CONSTRUIR UMA CULTURA RESILIENTE PARA SOLUÇÃO.

e como preparar nosso produto para
este cenário ?

Conclusão

DECISÕES ARQUITETURIAS
ALINHADAS AO CONTEXTO DO DESAFIO

Conclusão

Algumas considerações:

Crie uma cultura resiliente
Foco na entrega de valor para o cliente
Insira um mindset evolutivo e progressivo na empresa
Conscientização e pertencimento
Tenha um MVA e preserve a todo momento.
Pense sempre nas pessoas do time que trabalharão no seu produto
Não falamos de testes e Devops pois entendo que terão acesso por outra disciplina, mas vale focar nestes temas pois fazem parte do sucesso do produto.

Conclusão

Não existe bala de prata, tudo deve ser estudado e ajustado ao contexto.

Mantenha sempre a mente aberta é mais importante!

PUCRS online uol edtech