MBA USP ESALQ

Micro-front-end I

Vinicios Neves

MBA USP ESALQ

A responsabilidade pela idoneidade, originalidade
e licitude dos conteúdos didáticos apresentados é do professor.

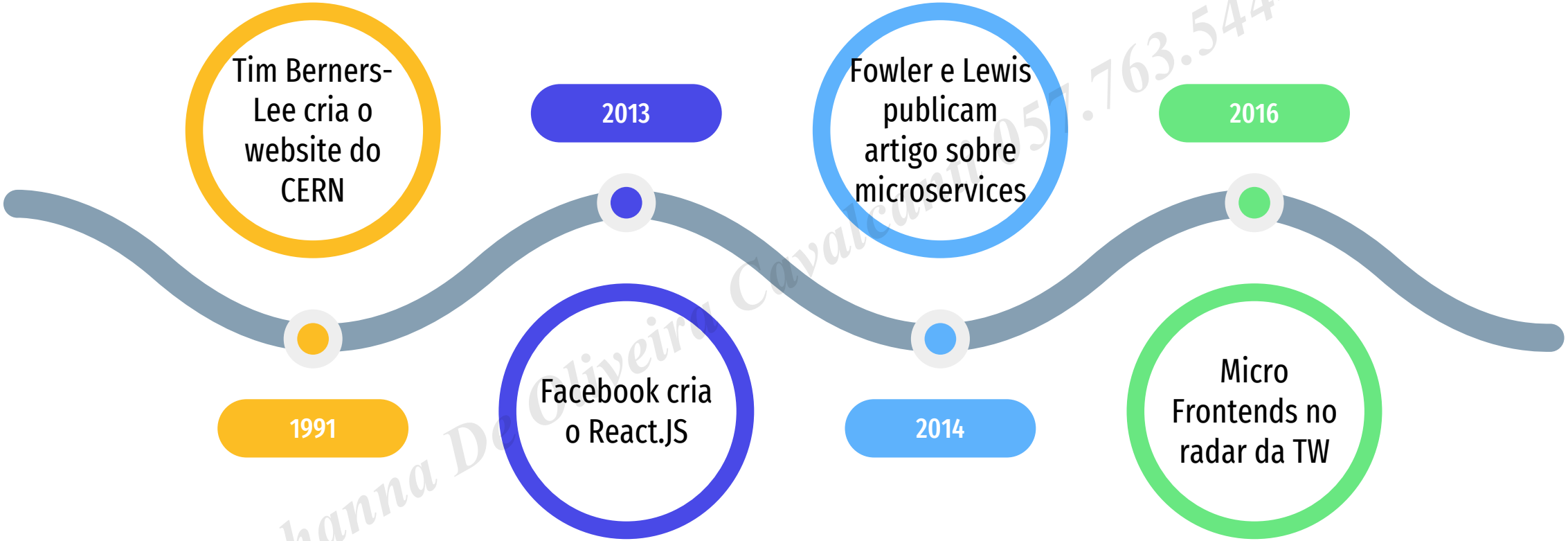
Proibida a reprodução, total ou parcial, sem autorização.

Lei nº 9610/98

Quem?


```
{
    "nome": "Vinicios Neves",
    "idade": 36,
    "naturalidade": "Nova Friburgo - RJ",
    "endereco": "Braga - Portugal",
    "profissao": "Tech Lead @ UsTwo"
}
```

Imagem retirada de "Movie Web"

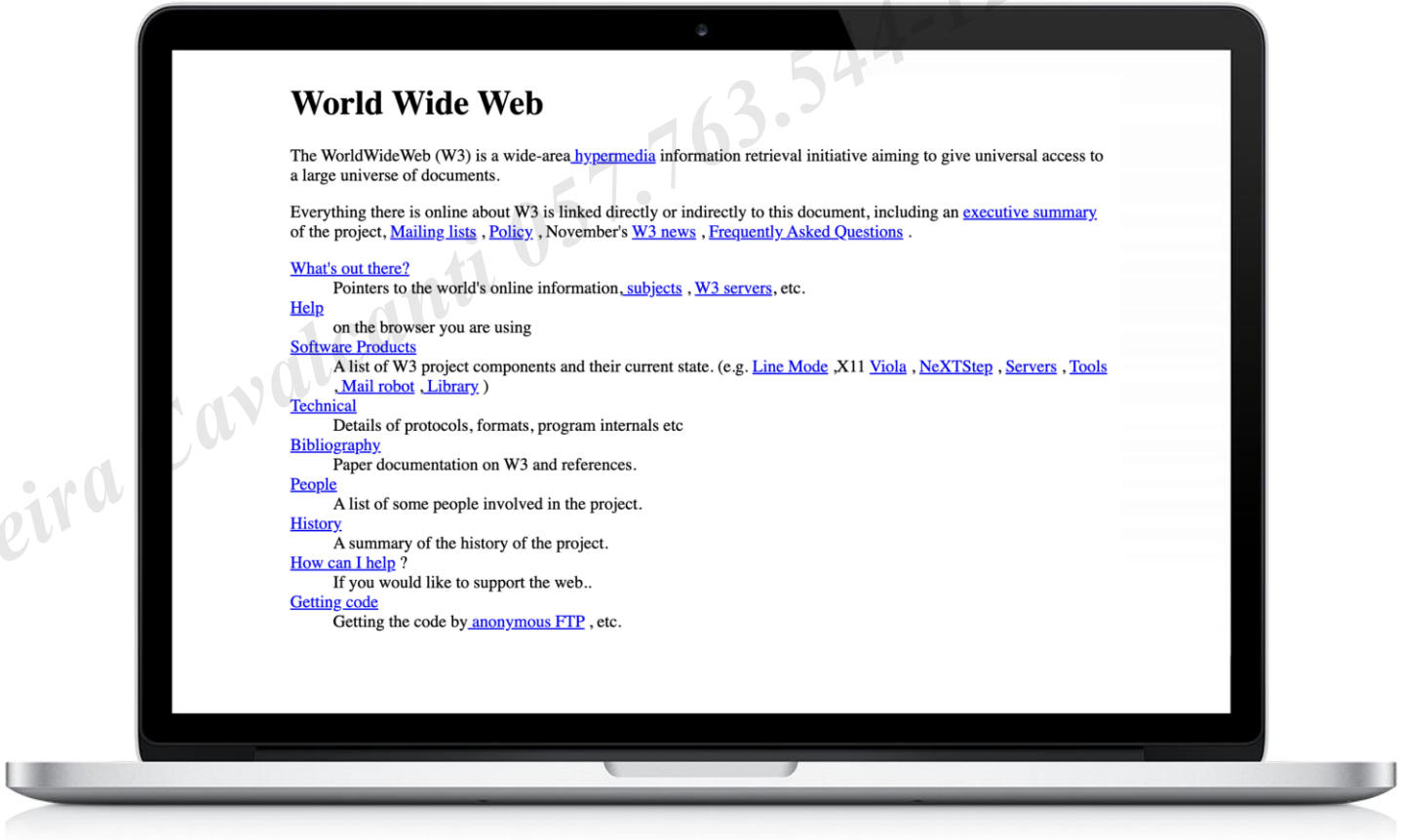


1991 e o nascimento da WWW

O primeiro website da história foi lançado em 6 de agosto de 1991 por Tim Berners-Lee, o criador da World Wide Web. Ele foi hospedado no CERN e tinha o seguinte endereço:

🔗 http://info.cern.ch (ainda acessível hoje!)

link pro site original



2013 e o nascimento do React como Open Source

- O React nasceu no Facebook para lidar com a complexidade do feed de notícias.
- A atualização do DOM era lenta, dificultando a manutenção do código.
- Jordan Walke criou o DOM virtual, otimizando as mudanças na interface.
- A solução teve tanto sucesso que o Facebook lançou o React como open-source em 2013.
- Desde então, o React se tornou um dos frameworks mais usados no frontend.

link pro site original

Frontend imperativo

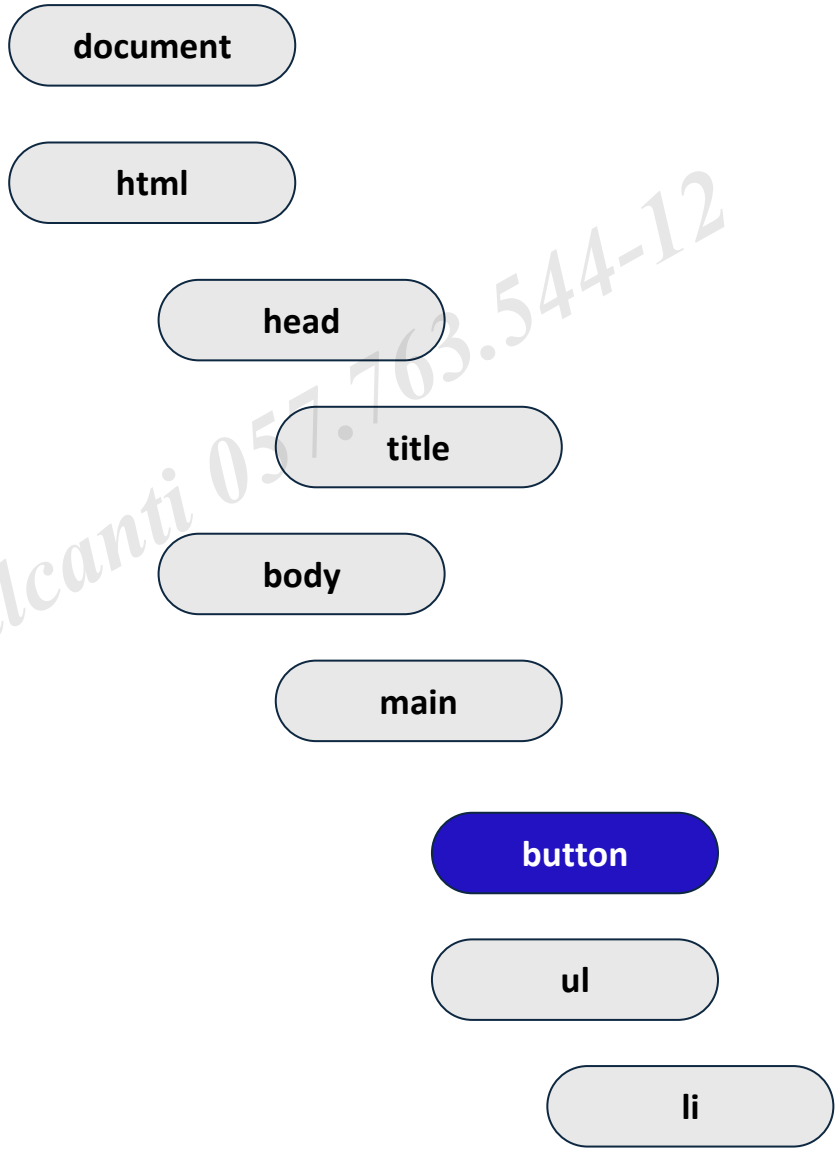
Diz ao navegador exatamente como mudar cada elemento na tela, passo a passo.

Cada interação exige manipular diretamente o DOM

1. Se um botão precisa mudar um texto ao ser clicado, o código imperativo busca o elemento, altera seu conteúdo e pode precisar atualizar estilos manualmente.
2. Com muitas interações, o código cresce descontroladamente, cheio de getElementById, addEventListener e manipulações diretas.

```
document.addEventListener("DOMContentLoaded", function () {
    const button = document.getElementById("btn");
    const textElement = document.getElementById("btn-text");

    button.addEventListener("click", function () {
      textElement.textContent = "Texto atualizado!";
      textElement.style.color = "red";
      textElement.style.fontWeight = "bold";
    });
  });

```

3. O navegador recalcula o layout e re-renderiza a página a cada mudança, tornando tudo mais lento.
4. O código vira uma bagunça, difícil de entender e manter, com várias partes alterando o DOM ao mesmo tempo.



Frontend declarativo

Você descreve o estado final da interface e o React cuida das mudanças no DOM

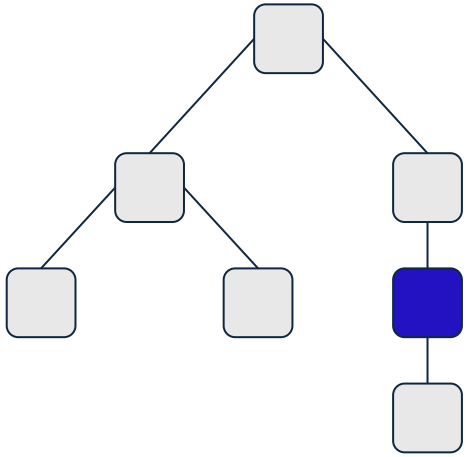
Cada interação exige manipular diretamente o DOM

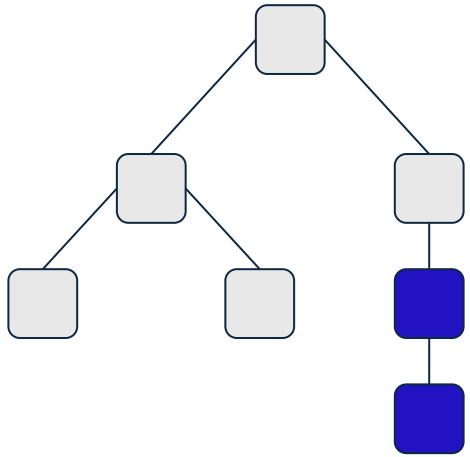
1. Em vez de dizer "como" modificar o DOM, você define "o que" quer exibir com base no estado.
2. Se um botão altera um texto, basta atualizar o estado que representa esse texto, e a interface se ajusta automaticamente.

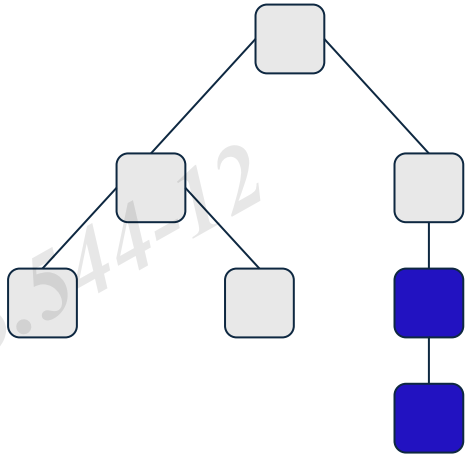
```jsx
import { useState } from "react"

function App() {
  const [text, setText] = useState("Texto inicial")

  return (
    <div>
      <p style={{ color: "black" }}>{text}</p>
      <button onClick={() => setText("Texto atualizado!")}>
        Alterar Texto
      </button>
    </div>
  )
}

export default App

```

3. O React usa um Virtual DOM para calcular apenas as mudanças necessárias, evitando atualizações desnecessárias no navegador.
4. O código fica mais previsível e fácil de manter, pois a interface reflete sempre o estado atual, sem manipulações diretas.

Virtual Dom

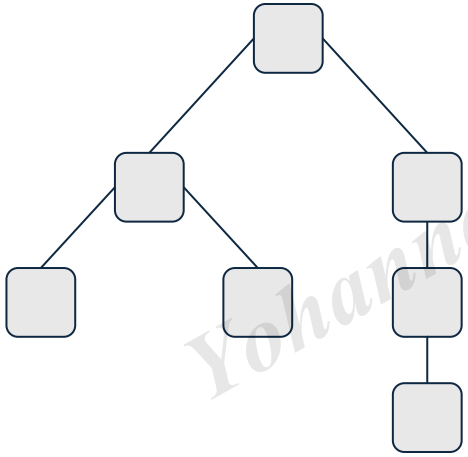


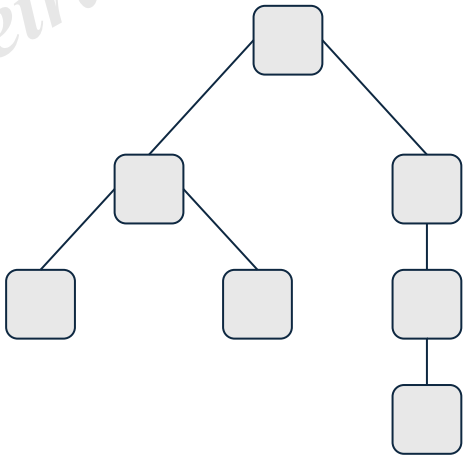


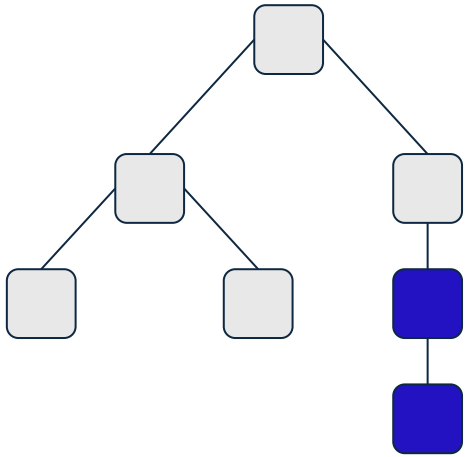


Mudança no estado → Calcula a diferença → Renderiza

Dom no browser



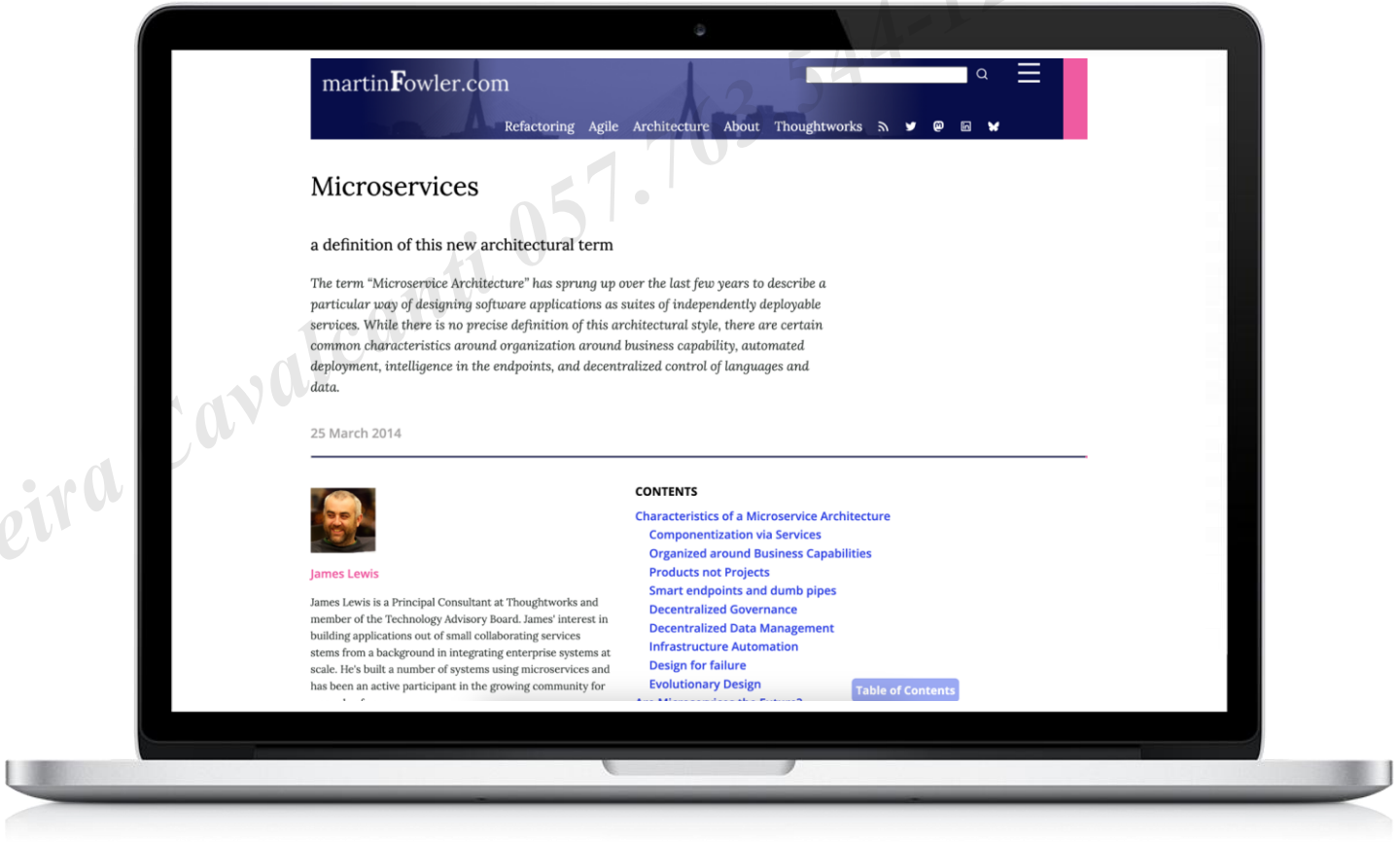




2014 e a popularização dos microservices

- Microservices resolvem problemas de escalabilidade e manutenção dos monolitos.
- Em monolitos, qualquer mudança exige um deploy completo.
- Com microservices, cada funcionalidade é independente.
- Isso facilita desenvolvimento, escalabilidade e implantação.
- Times ganham autonomia, resiliência e flexibilidade.

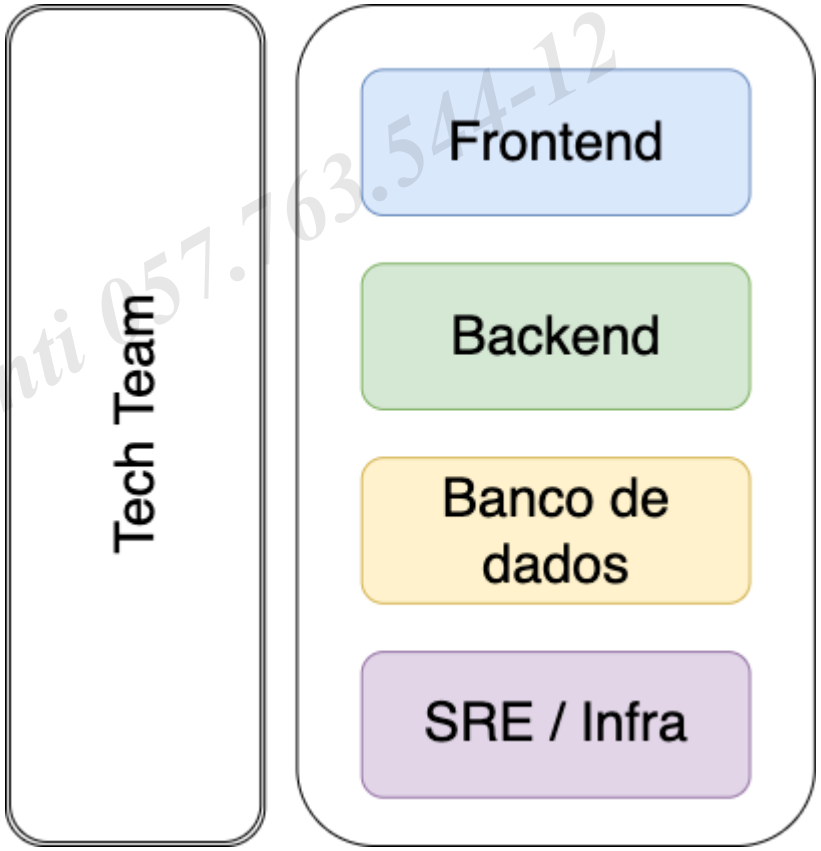
link pro artigo original



Um monolito para a todos governar

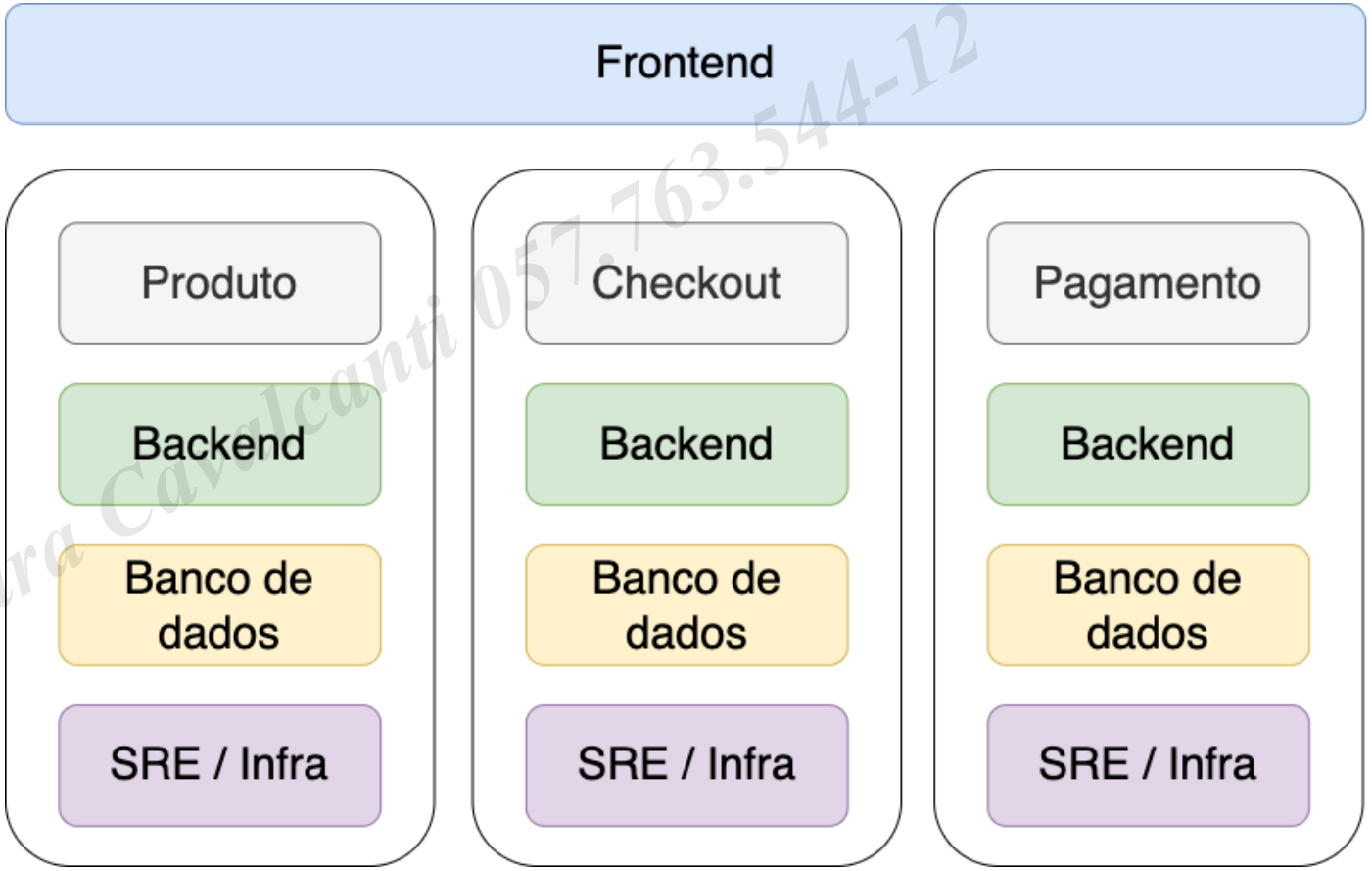
A palavra monólito vem do grego "**monolithos**" (**μόνολιθος**), que é formada por:

- "**monos**" (μόνος) → "único" ou "um só"
- "**lithos**" (λίθος) → "pedra"



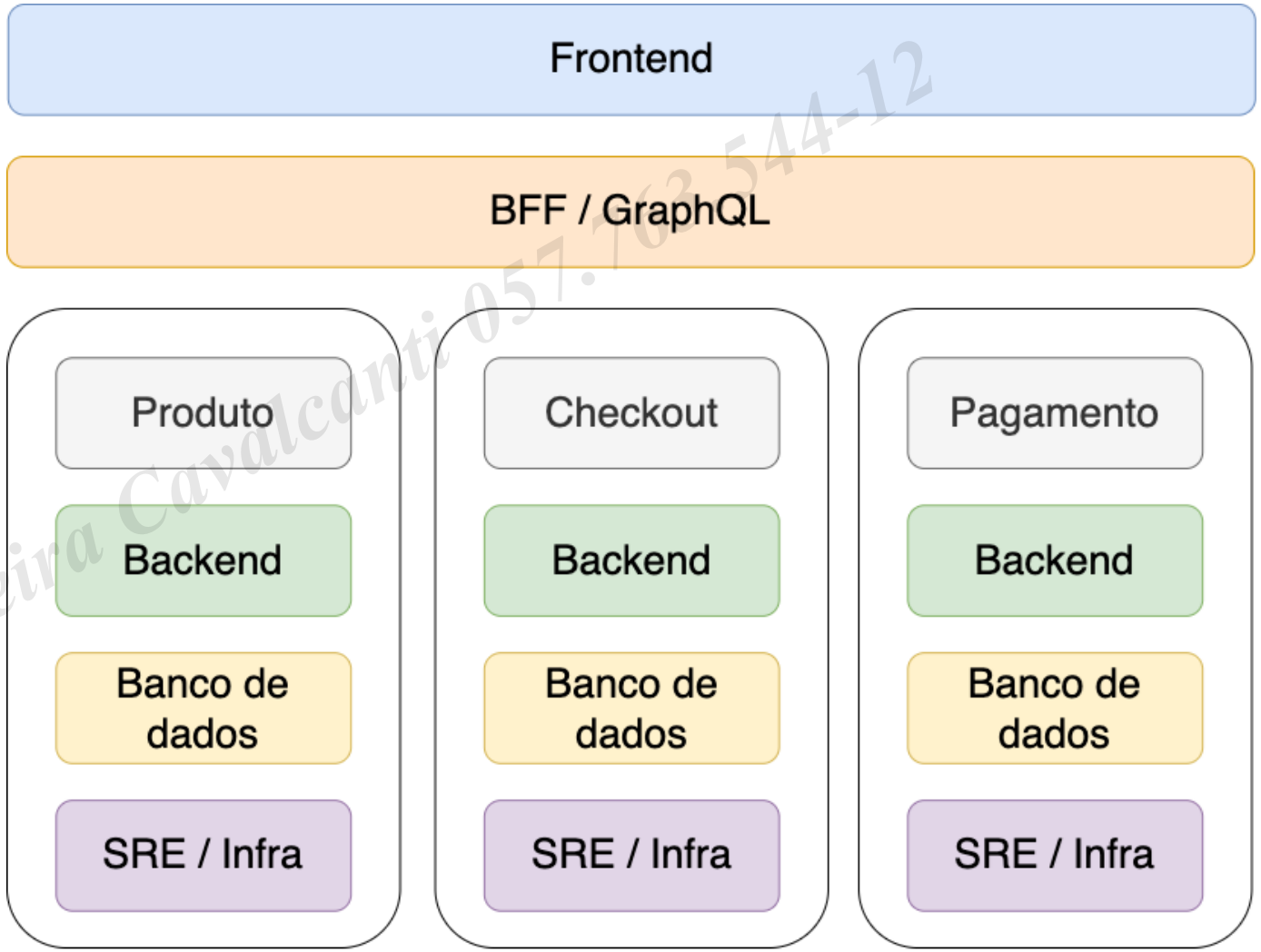
Microservices ao resgate

- Separar só o backend não resolve o monolito no frontend.
- Um frontend gigante ainda sofre com deploys complexos.
- O código continua acoplado e difícil de escalar.
- Times seguem dependentes e com pouca autonomia.



Uma camada de agregação

- BFF ou GraphQL simplificam o consumo de microservices.
- Agregam dados e reduzem a complexidade das requisições.
- Melhoram a performance e desacoplam parte da lógica do frontend.



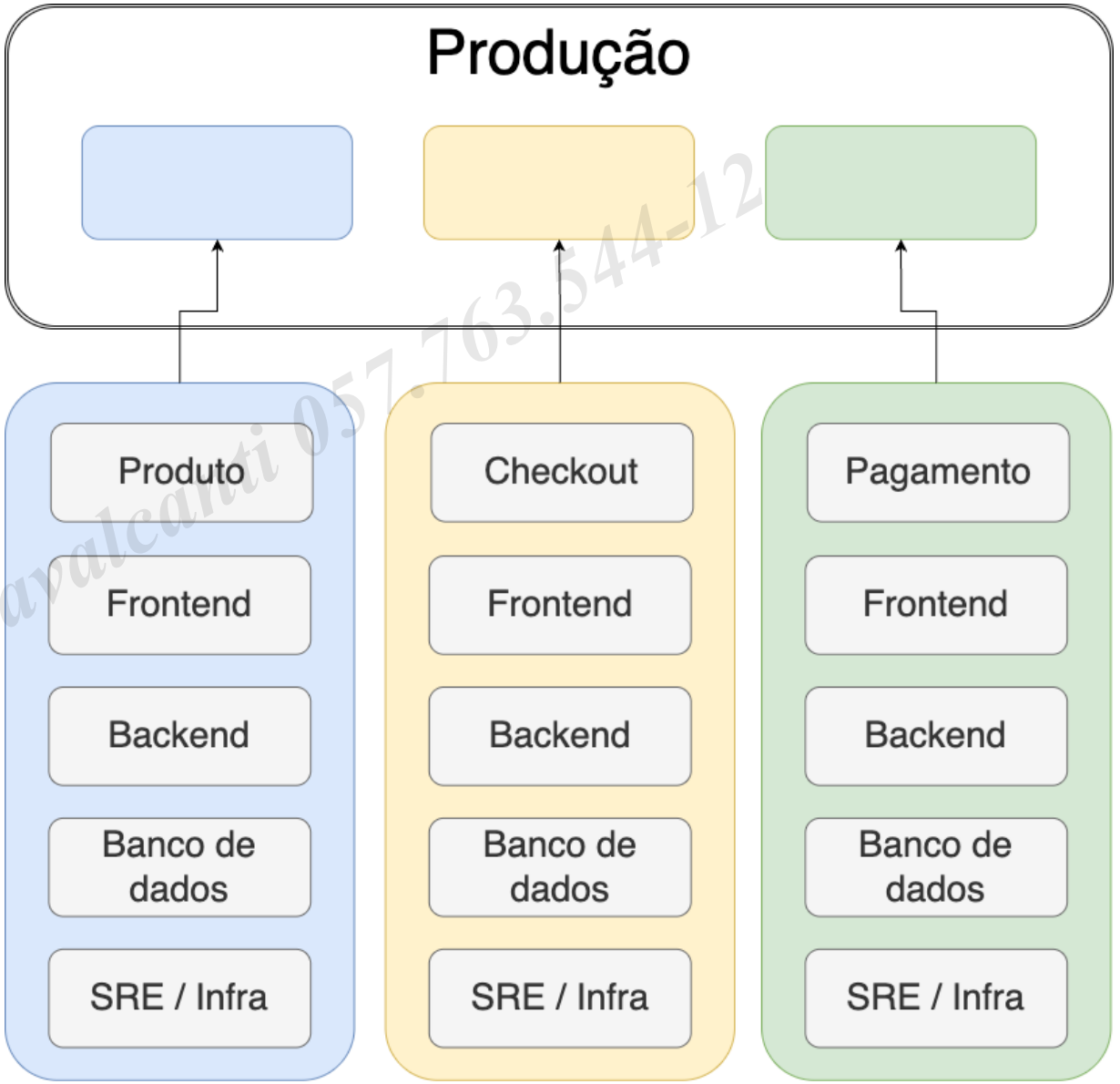
2016 e o boom dos micro frontends

- Micro Frontends dividem um site em funcionalidades independentes.
- Cada equipe foca em um domínio específico.
- Times são multifuncionais e desenvolvem tudo de ponta a ponta.
- Cuidam desde o banco de dados até o frontend.

link pro artigo original

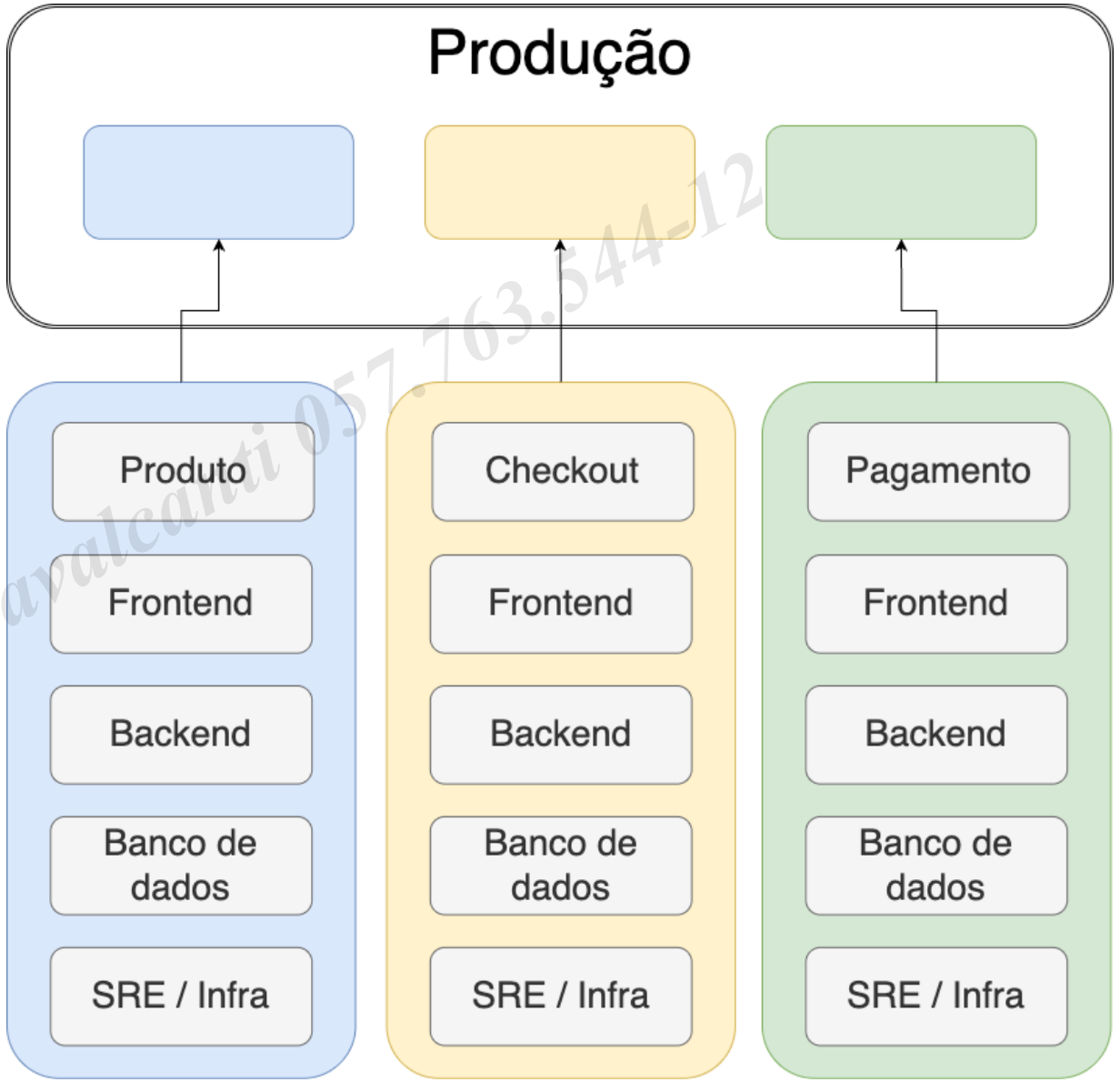
Declaração de independência

- Cada equipe cuida de todo o ciclo de vida da sua funcionalidade.
- Desenvolvem do banco de dados ao frontend de forma independente.
- Times evoluem suas partes sem depender de outros.
- Isso reduz gargalos e acelera entregas.



Organização

- A abordagem vai além da arquitetura técnica.
- Reflete como a empresa se organiza.
- A separação entre domínios deve fazer sentido para o negócio.
- Equipes precisam de autonomia real para inovar sem impactar o sistema todo.



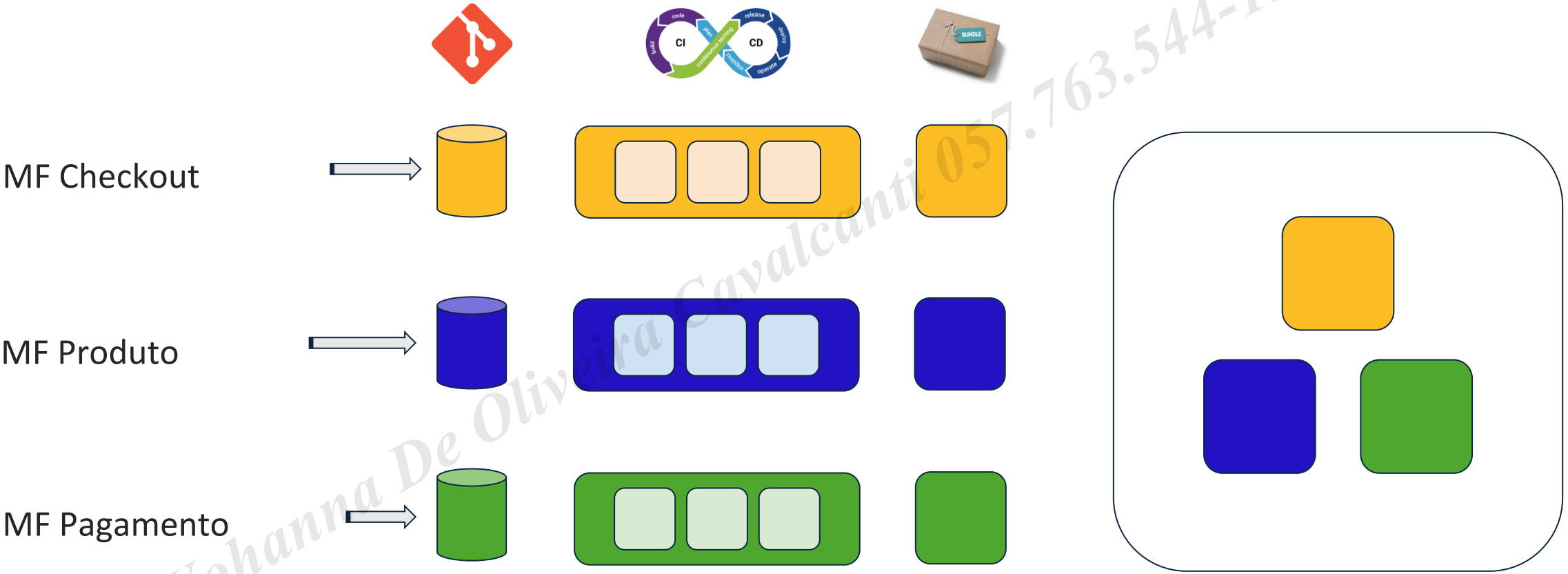


Imagem retirada de "Sticker Mania"

Cuidado com armadilhas

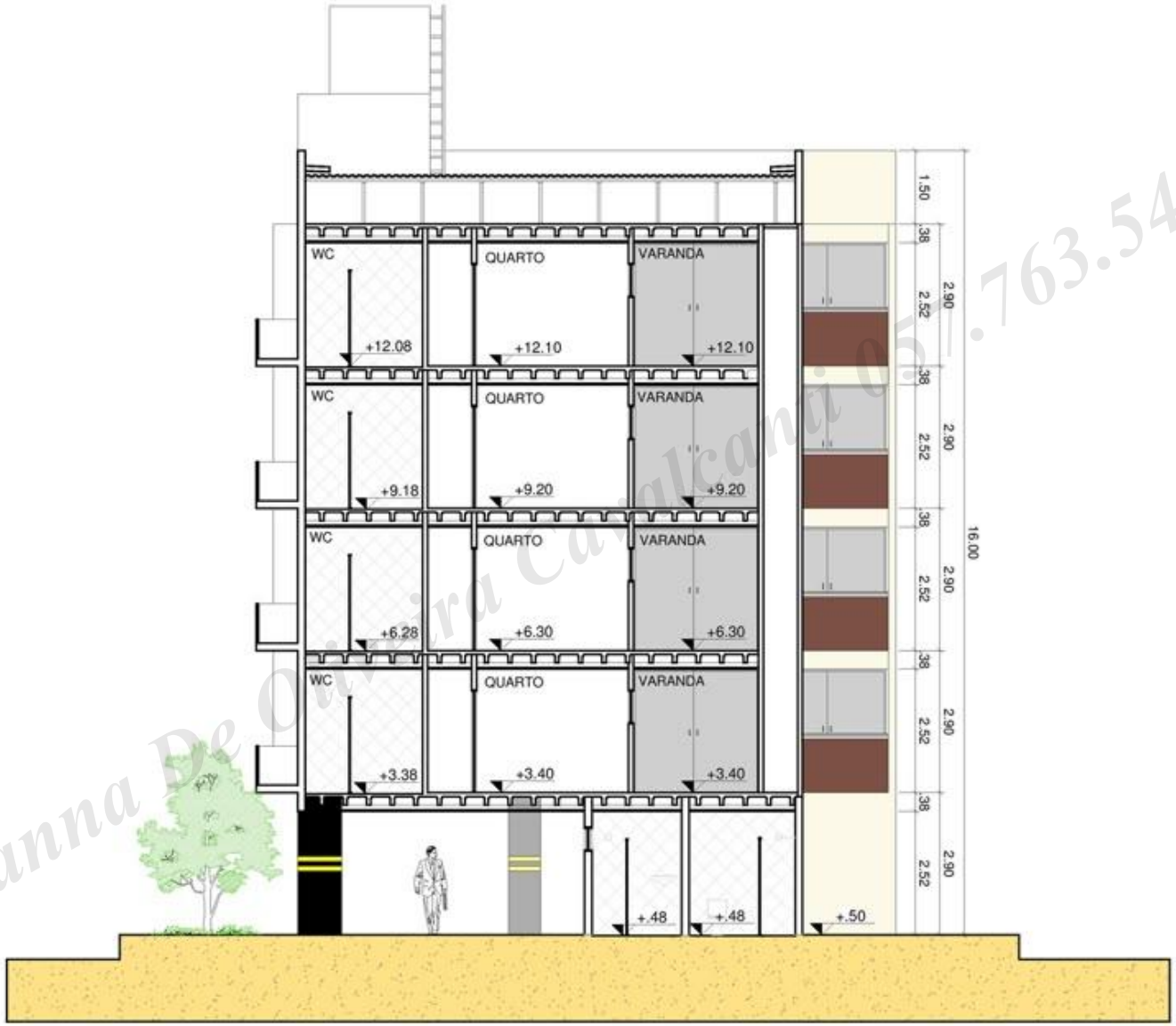
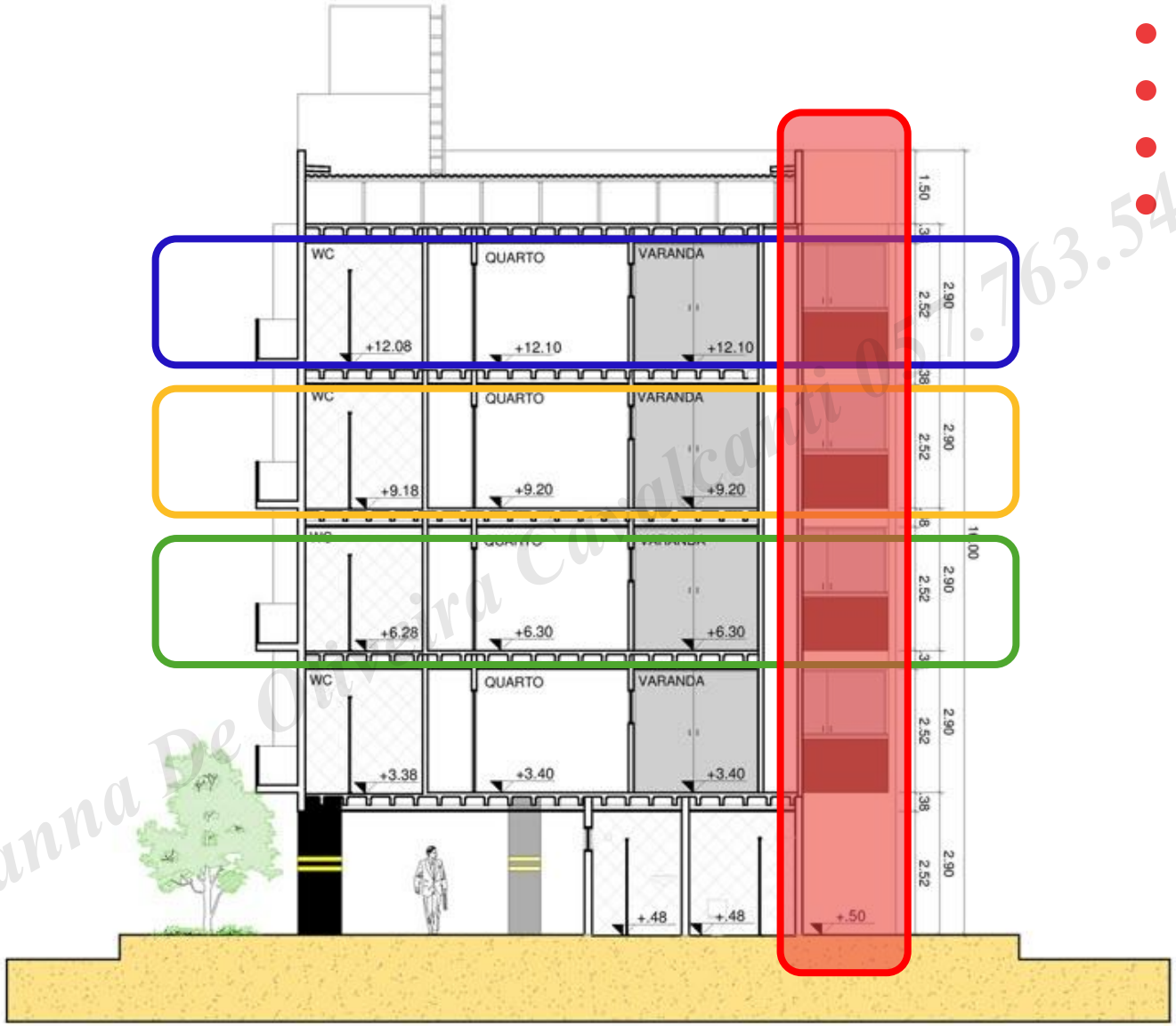


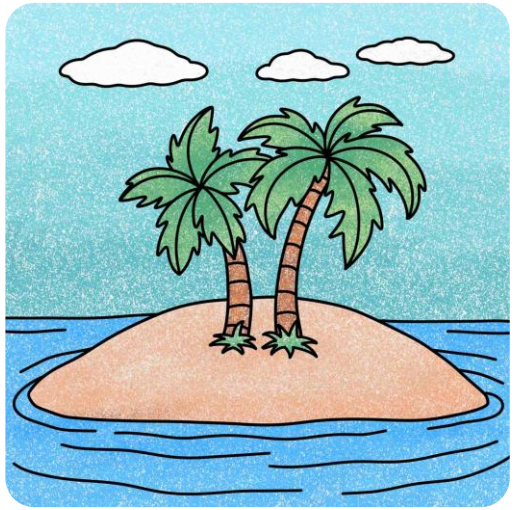
Imagem retirada de "Pinterest"



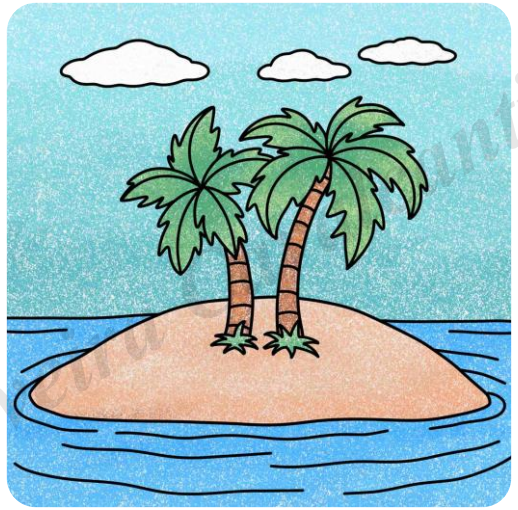
- Design System
- Infra
- Validações
- Autenticação

Imagem retirada de "Pinterest"

Time de Checkout



Time de Produto



Time de Pagamento

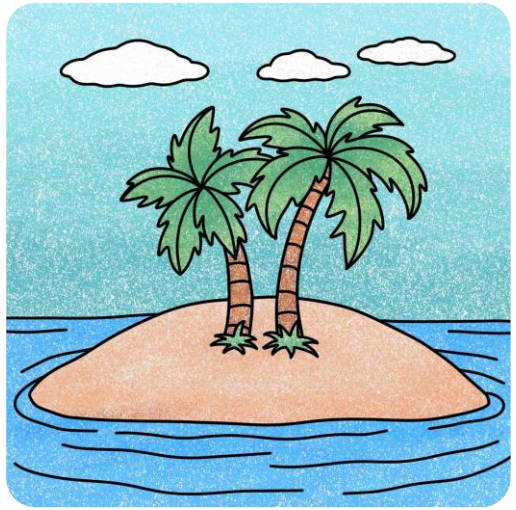
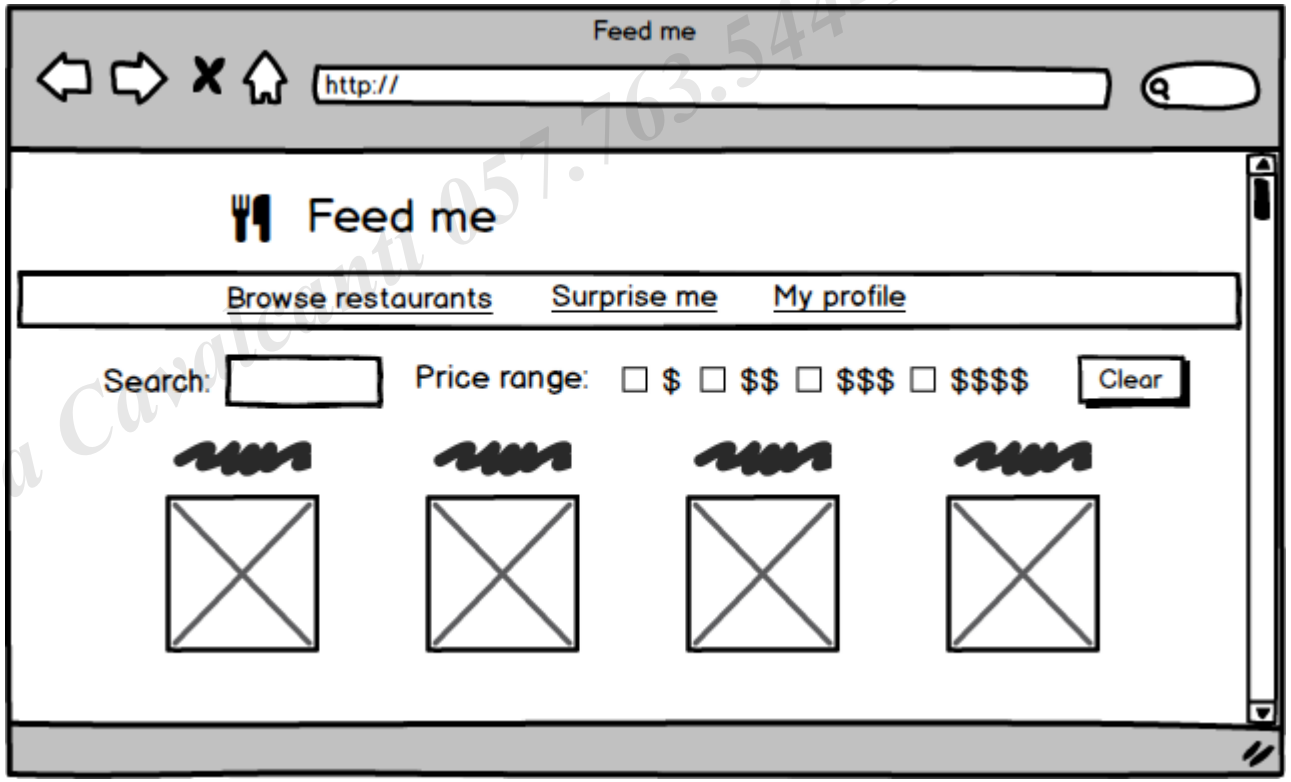


Imagem retirada de "HelloArtsy"

Estudo de caso

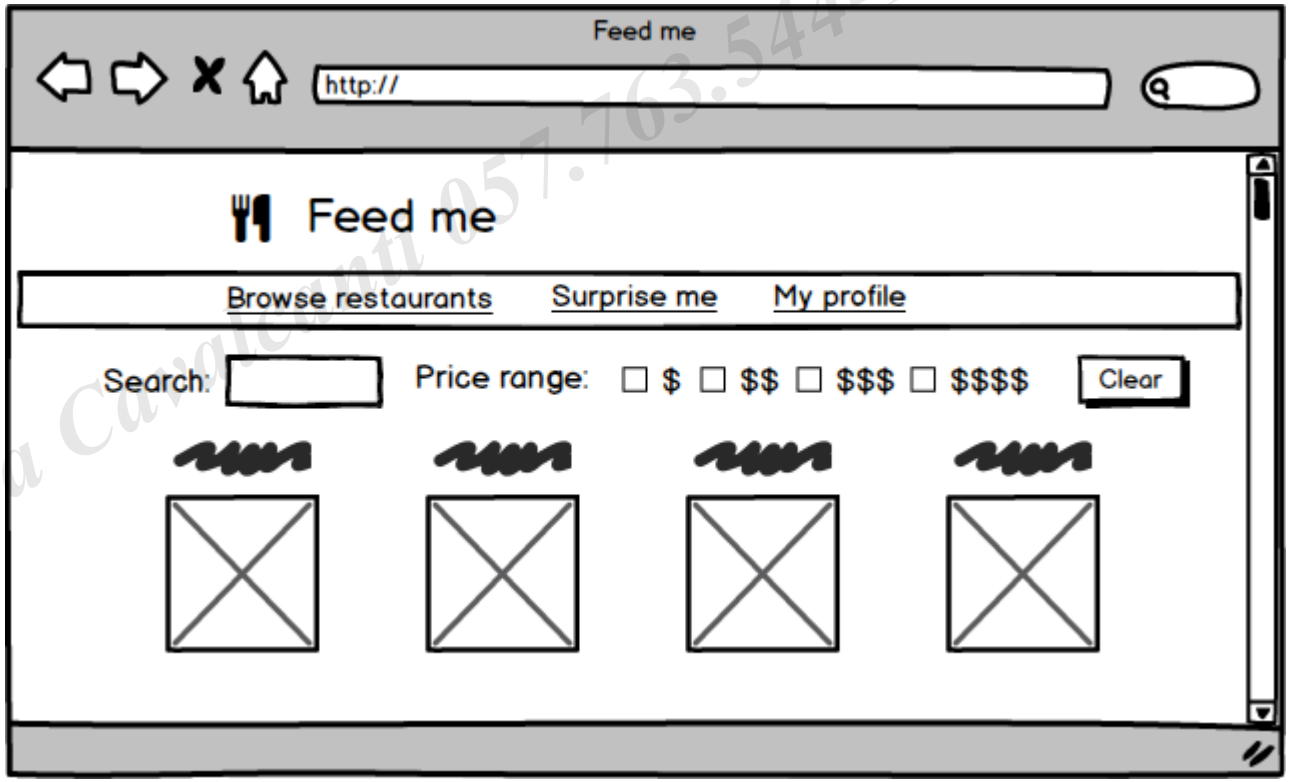
Feed me

- Identificar áreas independentes → Navegação, busca, filtros e listagem de restaurantes.
- Definir responsabilidades → Cada micro frontend deve ter um domínio claro.
- Criar um container → Responsável por carregar os micro frontends dinamicamente.
- Gerenciar comunicação → Como os micro frontends compartilham dados (ex: filtros aplicados).



Feed me

- Escolher a estratégia de composição → Múltiplos repositórios ou um único monorepo.
- Planejar deploys independentes → Cada micro frontend deve ser atualizado sem impactar os outros.
- Garantir coesão visual → Uso de um design system ou compartilhamento de estilos.



Dividindo para conquistar

- A estrutura visual da página pode guiar a divisão dos micro frontends.
- Seções como navegação e conteúdo podem ser independentes.
- Cada time desenvolve, testa e faz deploy sem afetar os outros.
- Isso traz mais flexibilidade e escalabilidade.

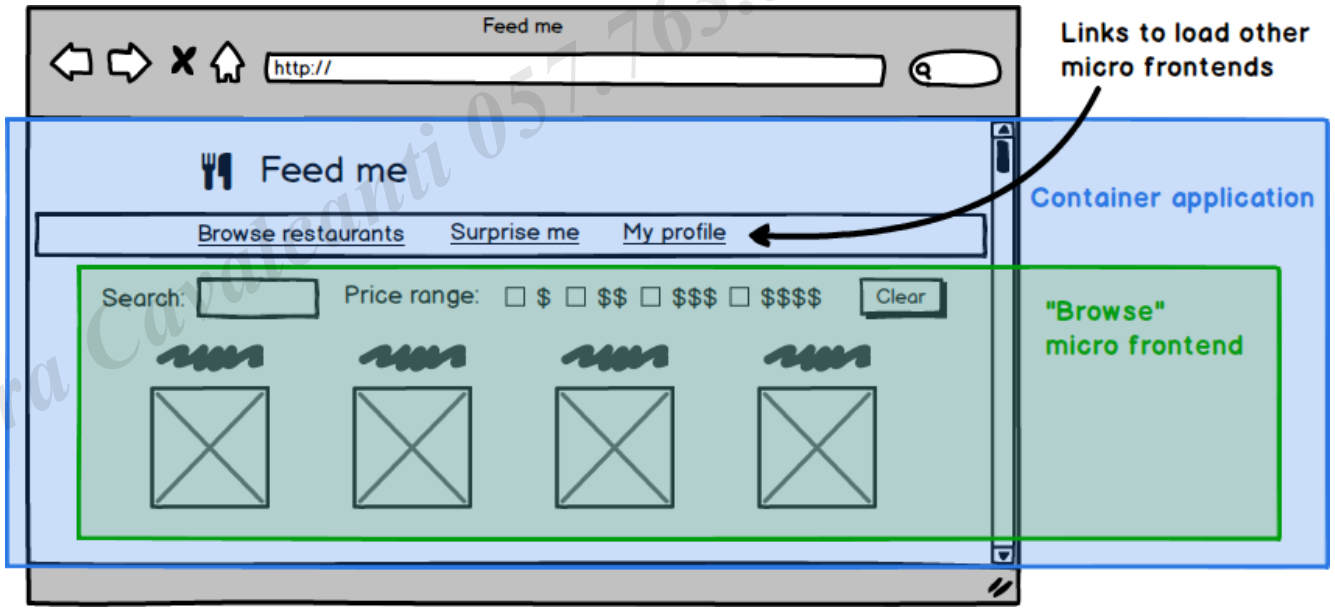


Imagem retirada de "Micro Frontends", artigo do blog Martin Fowler.

iFrames

Run-time

Segurança e boas práticas

Estilos em Micro Frontends

- Cada MF pode ter seu próprio design system e estilos.
- O maior desafio é evitar conflitos globais entre os MFs.
- IFrames isolam estilos por padrão, evitando vazamento de CSS.
- Se não usar IFrames, prefira:
  - Shadow DOM para escopo isolado.
  - Prefixos em classes para evitar colisões.
- Evite dependências globais como bootstrap.css carregado no shell.

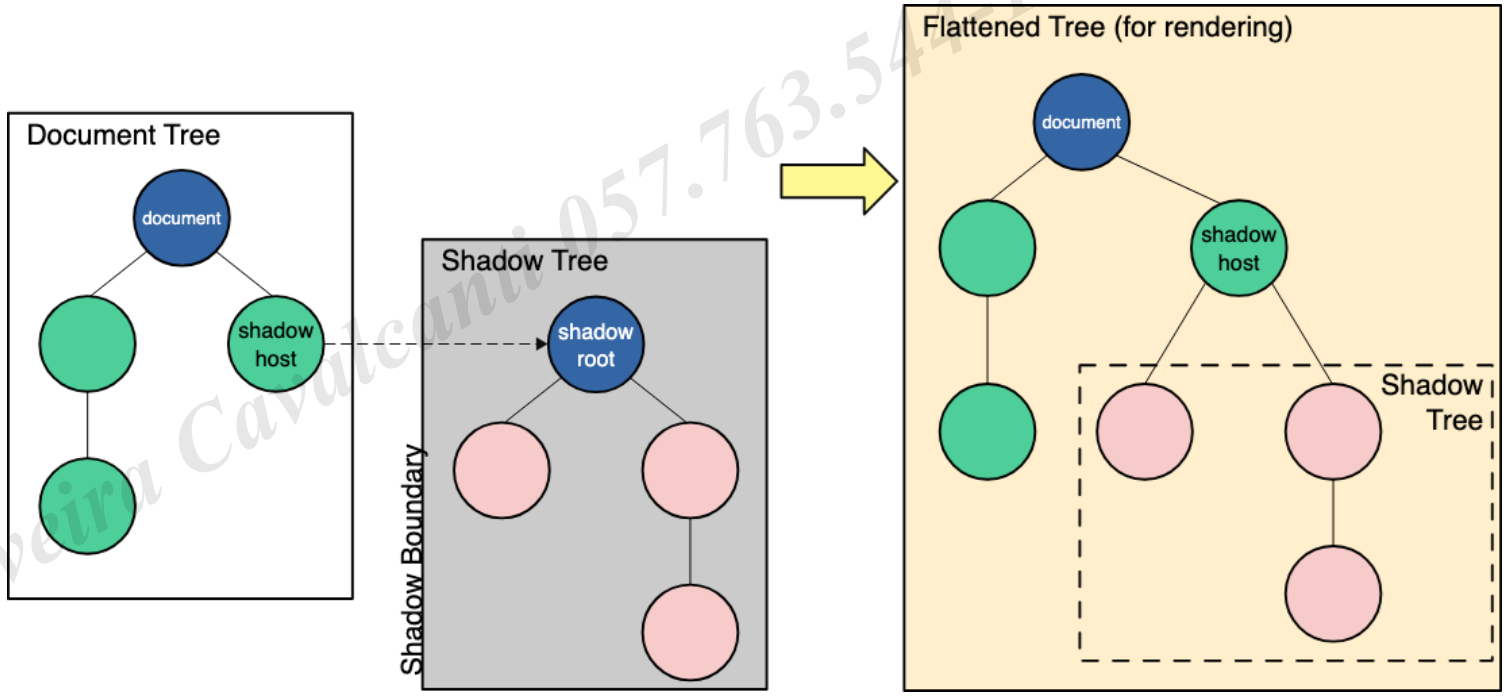


Imagem retirada de "Mozilla"

Cuidados ao usar Import Maps

- Import Maps carregam módulos globais, o que pode gerar dependências indesejadas.
- Se dois MFs importam a mesma lib, podem acabar compartilhando uma versão errada.
- Uma mudança no Import Map do Shell pode quebrar um MF sem aviso.
- Evite expor pacotes internos que não deveriam ser compartilhados entre MFs.
- Sempre versionar as libs no Import Map para evitar incompatibilidades.

```
{
  "imports": {
    "mf-search": "http://localhost:8000/src/main.js",
    "mf-order": "http://localhost:8001/src/main.js"
  }
}
```

Segurança na comunicação entre IFrames

- IFrames isolam execução, impedindo acesso direto ao código de outros MFs.
- Use window.postMessage() para trocar dados com segurança.
- Sempre valide a origem da mensagem (event.origin) antes de processar.
- Não confie em mensagens externas sem validação
- Evite enviar dados sensíveis

```
1  window.addEventListener('message', (event) => {
2    if (event.origin !== 'https://trusted-mf.com') {
3      return
4    };
5  });
```

Segurança no uso de Import Maps

- Import Maps permitem definir caminhos para módulos remotos.
- Um atacante pode tentar substituir um módulo se houver brechas no CDN.
- Sempre carregue módulos de origens confiáveis.
- Defina Content Security Policy (CSP) para restringir a execução de scripts remotos.
- Não exponha APIs sensíveis nos módulos compartilhados pelo Import Map.
- Mantenha um controle de versões para evitar que mudanças no Import Map quebrem a aplicação.

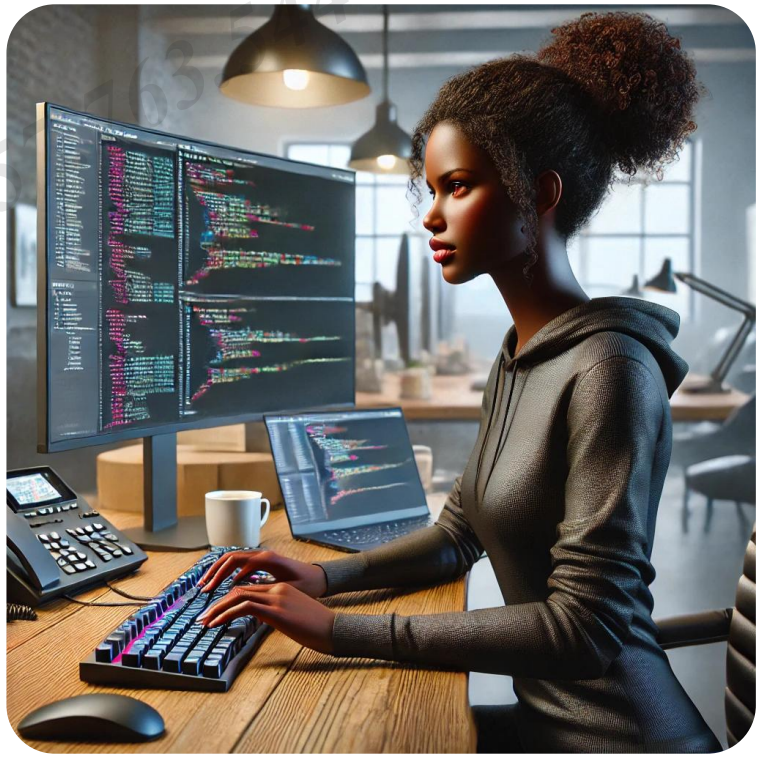


Imagem gerada por IA

MBA USP ESALQ

Obrigado!

Vinicios Neves | Linkedin