



UNIVERSIDADE
FEDERAL DO CEARÁ

Code smells em frameworks front-end

Disciplina: Qualidade de Software

Equipe: Ester Holanda Ravette (556506) e Kelton Martins Nojosa de Oliveira (571601)

Projeto: <https://github.com/primefaces/primeng>

Fork: <https://github.com/esterravette/primeng>

1. DOM

Eu estou atualmente trabalhando na refatoração do seguinte code smell:

```
packages/primeng/src/animateonscroll/animateonscroll.ts  
packages/primeng/src/autofocus/autofocus.ts  
packages/primeng/src/badge/badge.ts  
packages/primeng/src/bind/bind.ts  
packages/primeng/src/card/card.ts  
packages/primeng/src/ripple/ripple.ts (2 code smells refatorados)  
packages/primeng/src/terminal/terminal.ts (2 code smells refatorados)  
apps/showcase/components/demo/deferreddemo.ts
```

Esse code smell ocorre principalmente pelo acesso e modificação direta do DOM, além de funções utilitárias externas que manipulam o DOM fora da abstração do Angular, o que compromete o desacoplamento, a compatibilidade com SSR e a segurança da aplicação.

Minhas principais dificuldades na remoção do code smell são:

- Substituir leituras repetidas do DOM por referências mantidas em memória, cuidando para que essas referências fossem liberadas corretamente ao destruir o componente, evitando vazamentos.
- Ajustar a aplicação dinâmica de classes, estilos, atributos e eventos sem alterar o comportamento existente e sem deixar estados antigos acumulados no DOM.
- Garantir que o código que manipula o DOM só fosse executado quando o elemento estivesse disponível e em ambiente de navegador, evitando erros e problemas em SSR.



UNIVERSIDADE FEDERAL DO CEARÁ

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
A refatoração consistiu em substituir manipulações diretas do DOM por abordagens recomendadas pelo Angular, incluindo:

- Uso do Renderer2 (setStyle, removeStyle, addClass, removeClass, setAttribute, removeAttribute, setProperty, createElement, appendChild);
- Cache de referências DOM em propriedades da classe para evitar leituras repetidas e demoradas;
- Implementação de lógicas de comparação (diffing) para estilos, atributos e eventos;
- Uso de verificações de plataforma (isPlatformBrowser) e null checks para segurança em SSR;
- Melhoria da tipagem com ElementRef<HTMLElement> e tratamento seguro de APIs do navegador;
- Limpeza explícita de referências e listeners para evitar vazamentos de memória.*

2. IIC

Eu estou atualmente trabalhando na refatoração do seguinte code smell:

packages/primeng/src/avatar/avatar.ts

packages/primeng/src/animateonscroll/animateonscroll.ts

packages/primeng/src/badge/badge.ts

packages/primeng/src/timeline/timeline.ts

packages/primeng/src/toggleswitch/toggleswitch.ts

packages/primeng/src/iconfield/iconfield.ts

packages/primeng/src/buttongroup/buttongroup.ts

packages/primeng/src/divider/divider.ts

packages/primeng/src/floatlabel/floatlabel.ts

packages/primeng/src/fluid/fluid.ts

Esse code smell ocorre principalmente pelo uso desnecessário de herança (extends BaseComponent e extends BaseEditableHolder) apenas para reutilizar utilitários de estilo, configuração global ou lógica de formulário. Essa abordagem cria acoplamento forte,



UNIVERSIDADE FEDERAL DO CEARÁ

oculta dependências reais dos componentes, dificulta a manutenção, a testabilidade e pode gerar ciclos de importação.

Minhas principais dificuldades na remoção do code smell são:

- Identificar exatamente quais funcionalidades da classe base eram realmente utilizadas, garantindo que a remoção da herança não quebrasse templates, host bindings ou comportamentos existentes.
- Reimplementar localmente lógicas de estilização dinâmica e PassThrough (como cx, cn, sx, ptm, ptms) que antes eram abstraídas pela classe pai.
- No caso de componentes de formulário, reimplementar explicitamente interfaces que antes vinham prontas pela herança, como `ControlValueAccessor`, mantendo o comportamento esperado.
- Garantir que a inicialização e o acesso a Inputs e estilos continuassem funcionando corretamente sem depender da infraestrutura da classe base.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:

- Remoção da herança inadequada (`extends BaseComponent` e `extends BaseEditableHolder`);
- Injeção de dependência direta;
- Implementação local de utilitários de classe;
- Criação de getters locais para gerenciar classes CSS, estilos e atributos do host;
- Implementação explícita de interfaces de ciclo de vida e formulários;
- Simplificação de componentes onde a herança era superficial ou desnecessária (caso do `buttongroup.ts`).

3. ANY

Eu estou atualmente trabalhando na refatoração do seguinte code smell:

```
apps/showcase/components/doc/helpers/index.ts  
apps/showcase/doc/autocomplete/disabledldoc.ts  
apps/showcase/doc/autocomplete/filledldoc.ts
```

Esse code smell ocorre pelo uso excessivo do tipo `any`, que desativa a verificação



UNIVERSIDADE FEDERAL DO CEARÁ

estática de tipos do TypeScript. Que acaba anulando os benefícios da linguagem, mascarando erros de contrato que só aparecem em tempo de execução e tornando o código frágil a refatorações futuras.

Minhas principais dificuldades na remoção do code smell são:

- Descobrir o formato real do dado, se é um array, string, objeto ou numero.
- De onde ele vinha? ex: input, api, evento, serviço.
- rastrear o fluxo daquele dado no código por que ele pode ser um input, ai tem que ser verificado se o resto do código vai aceitar aquele tipo.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:

- Substituição any por unknown, o que acaba forçando a verificação do tipo antes do uso, ajuda evitando acessos inseguros a propriedades.
- tipagem especifica de tipos primitivos, alteração de tipos mais genéricos como any[] para tipos mais concretos string[]
- em alguns casos a criação de uma interface explícita, pra dizer exatamente como aquele objeto tem que ser.

3. TMI

Eu estou atualmente trabalhando na refatoração do seguinte code smell:

apps/showcase/components/doc/app.code.ts
apps/showcase/components/doc/app.docapitable.ts
packages/primeng/src/button/button.ts
packages/primeng/src/carousel/carousel.ts
packages/primeng/src/checkbox/checkbox.ts
packages/primeng/src/colorpicker/colorpicker.ts
packages/primeng/src/confirmdialog/confirmdialog.ts
packages/primeng/src/contextmenu/contextmenu.ts
packages/primeng/src/dataview/dataview.ts
packages/primeng/src/drawer/drawer.ts



UNIVERSIDADE FEDERAL DO CEARÁ

Esse code smell ocorre quando um componente expõe um número excessivo de propriedades de entrada (@Input). Isso viola o princípio de encapsulamento, aumenta a complexidade cognitiva, polui os templates HTML e torna a manutenção e os testes unitários mais difíceis, pois o componente assume muitas responsabilidades de configuração.

Minhas principais dificuldades na remoção do code smell são:

- identificar o que iria afetar no código mexer naquele input.
- manipular todo o código após juntar todos eles dentro de uma interface.
- verificar se era necessário modificação no template HTML.
- e garantir que a reatividade do angular continue funcionando corretamente.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:

- substituição de múltiplos @inputs por um input tipado com uma nova interface
- criação de novas interfaces de configuração, ex: AppConfig, DocApiTableData.
- implementação de getters pra verificar o objeto de configuração e retornar valores caso a propriedade não seja definida.