

Refatoração de Code Smells em Frameworks Front-end

Análise e Melhoria do Projeto ng-bootstrap

Autores: Sávio de Carvalho e Renan Soares

Trabalho da disciplina de Qualidade de Software - UFC



O Projeto Selecionado: ng-bootstrap

O ng-bootstrap é uma **biblioteca de componentes Angular** que implementa o framework Bootstrap CSS utilizando exclusivamente o TypeScript. Ele oferece uma coleção robusta de widgets Angular que não dependem do JavaScript original do Bootstrap, focando em segurança e integração com o ecossistema Angular.

O **objetivo central** do nosso trabalho foi identificar e corrigir code smells presentes em sua base de código, visando aprimorar a manutenibilidade, legibilidade e segurança do projeto.



Metodologia e Code Smells Identificados

1

Processo de Análise

Utilizamos a ferramenta de detecção de code smells desenvolvida pelo Maykon (angular-code-smells)

2

Code Smells Selecionados

Focamos em quatro tipos de code smells que encontrados no framework ng-bootstrap, são eles:

- Manipulação Direta do DOM
- Uso Excessivo de Any Type
- Herança em vez de Composição
- Arquivo Grande / God Class

Refatoração 1: Manipulação Direta do DOM

(Refatoração conduzida por Sávio)

O Problema

O Angular introduz o `Renderer2` para ser a camada segura e abstrata de manipulação do DOM. Ignorar isso acopla o seu código ao ambiente do navegador. A manipulação direta do DOM é mais difícil de testar e pode introduzir vulnerabilidades de segurança (ex: ataques XSS, se não for cuidadosa). O código pode falhar ao ser executado em ambientes que não são navegadores (ex: **Angular Universal** para Server-Side Rendering (SSR), onde não existe `document` ou `window`).

Arquivos afetados: `alert.ts`, `carousel.ts`, `modal-window.ts`, `modal-window.spec.ts`, `nav-outlet.ts`.

Solução Aplicada

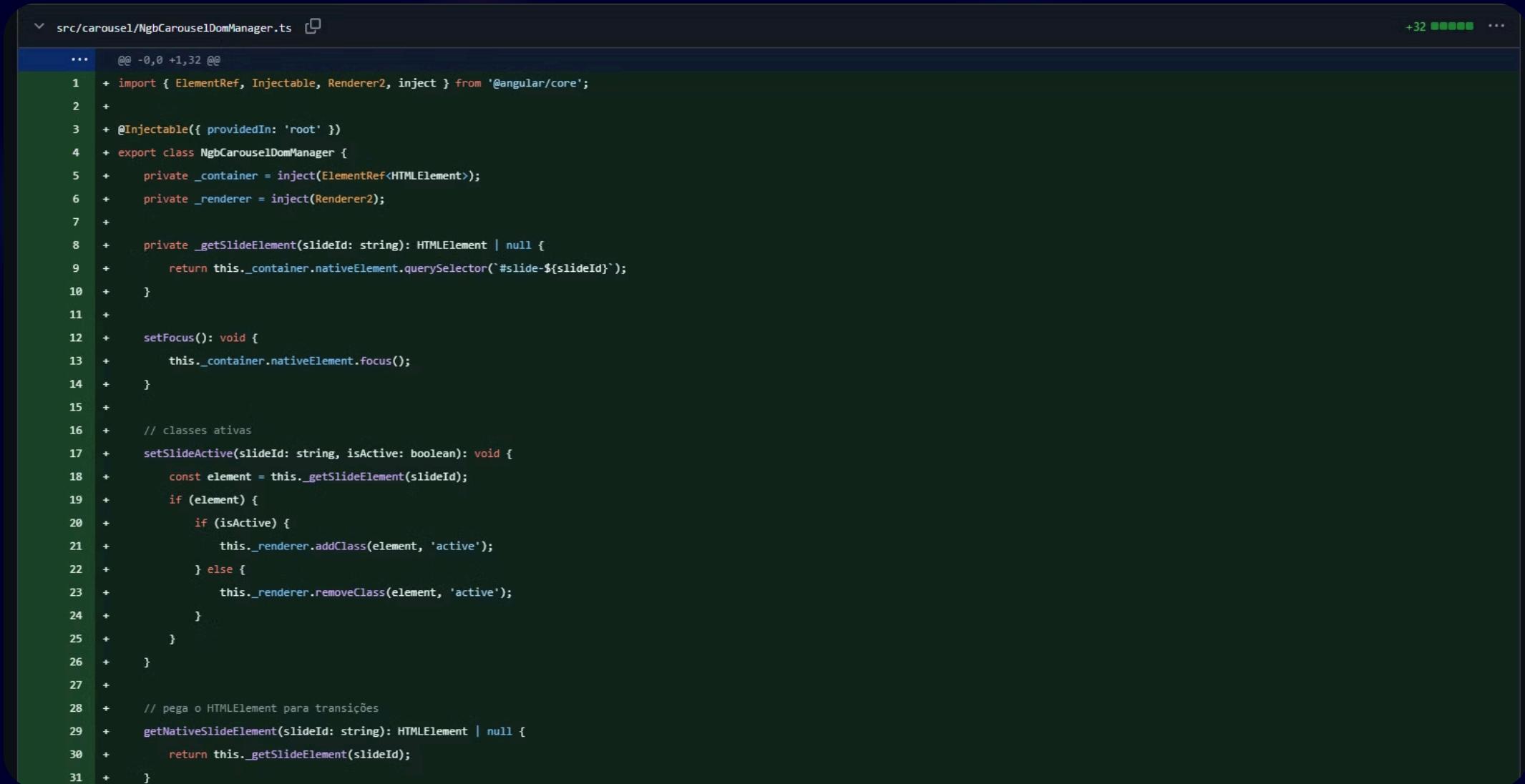
Movido a lógica de manipulação do DOM e a injeção de `ElementRef` para uma classe auxiliar (como por exemplo: `NgbModalWindowDomHelper`), tornando o componente principal mais limpo. Em alguns casos, foi criado *wrappers* ou métodos seguros (ex: o método `setClasses()`) para substituir o acesso direto à `nativeElement.classList`. Com isso, foi garantido que o `Renderer2` fosse usado sempre que possível dentro dos novos serviços de baixo nível para manipulação de classes, estilos e atributos, pois ele é a API agnóstica de plataforma do Angular.

Refatoração 1: Manipulação Direta do DOM

The screenshot shows a code diff interface comparing two versions of the `NgbCarousel` component's TypeScript file. The left column shows the original code (version 140), and the right column shows the refactored code (version 141). The changes are color-coded: red for deleted lines and green for added lines.

```
src/carousel/carousel.ts
@@ -334,7 +334,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
    }
    ...
    this.slides.changes.pipe(takeUntilDestroyed(this._destroyRef)).subscribe(() => {
@@ -334,7 +334,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
        this._transitionIds?.forEach((id) => ngbCompleteTransition(this._getSlideElement(id)));
@@ -337,7 +337,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
        this._transitionIds?.forEach((id) => ngbCompleteTransition(this._domManager.getNativeSlideElement(id)));
        this._transitionIds = null;
@@ -340,7 +340,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
        this._cd.markForCheck();
@@ -345,11 +345,10 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
    {
        mixedReadWrite: () => {
            for (const { id } of this.slides) {
@@ -348,7 +348,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
                const element = this._getSlideElement(id);
@@ -349,7 +349,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
                if (id === this.activeId) {
@@ -350,7 +350,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
                    element.classList.add('active');
@@ -351,7 +351,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
                    this._domManager.setSlideActive(id, true);
@@ -352,7 +352,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
                } else {
@@ -353,7 +353,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
                    element.classList.remove('active');
@@ -354,7 +354,7 @@ export class NgbCarousel implements AfterContentChecked, AfterContentInit, After
                    this._domManager.setSlideActive(id, false);
    }
}
```

Refatoração 1: Manipulação Direta do DOM



```
@@ -0,0 +1,32 @@
1+ import { ElementRef, Injectable, Renderer2, inject } from '@angular/core';
2+
3+ @Injectable({ providedIn: 'root' })
4+ export class NgbCarouselDomManager {
5+   private _container = inject(ElementRef<HTMLElement>);
6+   private _renderer = inject(Renderer2);
7+
8+   private _getSlideElement(slideId: string): HTMLElement | null {
9+     return this._container.nativeElement.querySelector(`#slide-${slideId}`);
10+   }
11+
12+   setFocus(): void {
13+     this._container.nativeElement.focus();
14+   }
15+
16+   // classes ativas
17+   setSlideActive(slideId: string, isActive: boolean): void {
18+     const element = this._getSlideElement(slideId);
19+     if (element) {
20+       if (isActive) {
21+         this._renderer.addClass(element, 'active');
22+       } else {
23+         this._renderer.removeClass(element, 'active');
24+       }
25+     }
26+   }
27+
28+   // pega o HTMLElement para transições
29+   getNativeSlideElement(slideId: string): HTMLElement | null {
30+     return this._getSlideElement(slideId);
31+   }

```

Refatoração 2: Uso Excessivo de Any Type

(Refatoração conduzida por Sávio)

O Problema

A principal vantagem do TypeScript (capturar erros antes da execução) é eliminada. Erros de digitação (`myObject.leght` em vez de `myObject.length`) só serão encontrados em tempo de execução. O código se torna **indocumentado**. Outros desenvolvedores não conseguem inferir a estrutura de dados esperada ou retornada, dificultando a manutenção e a integração. Se a estrutura de dados subjacente mudar, o compilador não notificará o código que usa `any`, levando a falhas silenciosas.

Arquivos afetados: `ngb-date.ts`, `timepicker.ts`,
`utils/transition/ngbTransition.ts`, `utils/popup.ts`, `ngb-date.ts`.

Solução Aplicada

A solução envolveu a **definição de interfaces e tipos explícitos**, como `NgbTimeStruct`, para garantir que os dados manipulados possuam uma estrutura bem definida. Para objetos genéricos ou dados com estrutura variável, optamos pelo uso de `Record<string, unknown>`, que ainda fornece algum nível de tipagem segura sem ser excessivamente restritivo.

Refatoração 2: Uso Excessivo de Any Type

```
@@ -39,7 +39,7 @@ export class NgbTimeStructAdapter extends NgbTimeAdapter<NgbTimeStruct> {
39 39     */
40 40     fromModel(time: NgbTimeStruct | null): NgbTimeStruct | null {
41 41         return time && isInteger(time.hour) && isInteger(time.minute)
42 42 -         ? { hour: time.hour, minute: time.minute, second: isInteger(time.second) ? time.second : <any>null }
42 42 +         ? { hour: time.hour, minute: time.minute, second: isInteger(time.second) ? time.second : null }
43 43         : null;
44 44     }
45 45
@@ -48,7 +48,7 @@ export class NgbTimeStructAdapter extends NgbTimeAdapter<NgbTimeStruct> {
48 48     */
49 49     toModel(time: NgbTimeStruct | null): NgbTimeStruct | null {
50 50         return time && isInteger(time.hour) && isInteger(time.minute)
51 51 -         ? { hour: time.hour, minute: time.minute, second: isInteger(time.second) ? time.second : <any>null }
51 51 +         ? { hour: time.hour, minute: time.minute, second: isInteger(time.second) ? time.second : null }
52 52         : null;
53 53     }
54 54 }
```

Refatoração 2: Uso Excessivo de Any Type

The screenshot shows a code diff interface with a dark theme. The left pane displays the file path: `src/timepicker/ngb-time-struct.ts`. The right pane shows the code changes:

```
@@ -15,5 +15,5 @@ export interface NgbTimeStruct {  
15   15     /**  
16   16       * The second in the `'[0, 59]'` range.  
17   17     */  
18 -   second: number;  
18 +   second: number | null;  
19   19 }
```

The line `second: number;` has been modified to `second: number | null;`. The original line is highlighted in red, and the modified line is highlighted in green.

Refatoração 3: Herança em vez de Composição

(Refatoração conduzida por Renan)

O Problema: Acoplamento Rígido

Identificamos casos onde componentes estendiam classes base **apenas para acessar dados ou funcionalidades auxiliares**. Essa abordagem cria um acoplamento rígido, dificultando a reutilização independente das funcionalidades e a evolução do código.

Arquivos afetados: `accordion-overview`, `nav-overview`, `api-page`, `datepicker-calendars`, `datepicker-overview`, `pagination-overview`, `scrollspy-overview`, `nav.ts`, `table-overview`, `toast-overview`.

Solução Aplicada

A estratégia foi substituir a herança por **Injeção de Dependência**. Ao invés de estender, os componentes agora recebem as dependências de que precisam via construtor, utilizando `inject(COMPONENT_DATA)`.

A composição permite construir objetos complexos combinando partes reutilizáveis, sem depender de hierarquias rígidas. A classe só recebe o que precisa.

Refatoração 3: Herança em vez de Composição

```
- import { NgbdOverviewPage } from '../../../../../shared/overview-page/overview-page.class';
+ import { COMPONENT_DATA } from '../../../../../tokens';
- export class NgbdAccordionOverviewComponent extends NgbdOverviewPage {
+ export class NgbdAccordionOverviewComponent {
private _component = inject(COMPONENT_DATA);

get overview() {
    return this._component.overview;
}
```

demo/src/app/components/accordion/overview/accordion-overview.component.ts

O componente deixou de depender de *toda a lógica* da classe pai. Agora depende apenas do que realmente usa: o objeto COMPONENT_DATA

Refatoração 4: Arquivo Grande (Large File)

(Refatoração conduzida por Renan)

O Problema: Múltiplas Responsabilidades

Arquivos excessivamente grandes ou classes "Deus" acumulavam **múltiplas responsabilidades**, tornando-os difíceis de ler, testar e manter. A modificação de uma parte pequena do código poderia ter impactos inesperados em outras áreas.

Arquivos afetados: collapse.spec, datepicker-integration, datepicker-tools, hebrew.ts, modal-stack, rating.ts, scrollspy.ts, scrollspy-service, positioning.ts, api-doc.

Solução: Separation of Concerns

Foram aplicadas refatorações estruturais para quebrar arquivos monolíticos em módulos coesos e especializados.

Onde tinha lógica misturada com visualização, foram extraídos para serviços, utils.

Arquivos que possuíam múltiplas diretivas (ex scrollspy), foram separadas em seu próprio arquivo.

Refatoração 4: Arquivo Grande (Large File)

The image shows a code editor with four tabs open, illustrating a refactoring process for a large file. The tabs are:

- `src/scrollspy/scrollspy.ts`: The original file containing the `NgbScrollSpyRef` interface and the `NgbScrollSpyItem` class.
- `src/scrollspy/scrollspy-ref.ts`: A new file where the `NgbScrollSpyRef` interface is moved, along with its methods `get active()` and `scrollTo`.
- `src/scrollspy/scrollspy-fragment.ts`: A new file containing the `NgbScrollSpyFragment` directive.
- `src/scrollspy/scrollspy-options.ts`: A new file defining the `NgbScrollSpyProcessChanges` type and its properties.

The code editor highlights the changes made in each file, showing the moved code and the resulting structure. The `scrollspy-ref.ts` file contains the following code:

```
... @@ -0,0 +1,13 @@
1 + import { Observable } from 'rxjs';
2 + import { NgbScrollToOptions } from './scrollspy-options';
3 +
4 + /**
5 + * Common interface for the scroll spy API.
6 + */
7 + @internal
8 + */
9 + export interface NgbScrollSpyRef {
10 +   get active(): string;
11 +   get active$(): Observable<string>;
12 +   scrollTo(fragment: string | HTMLElement, options?: NgbScrollToOptions): void;
13 + }
```

The `scrollspy-fragment.ts` file contains the following code:

```
... @@ -0,0 +1,38 @@
1 + import { AfterViewInit, DestroyRef, Directive, inject, Input } from '@angular/core';
2 + import { NgbScrollSpy } from './scrollspy';
3 +
4 + /**
5 + * A directive to put on a fragment observed inside a scrollspy container.
6 + */
7 + @since 15.1.0
8 + */
9 + @Directive({
10 +   selector: '[ngbScrollSpyFragment]',
11 +   host: {
12 +     '[id]': 'id',
13 +   }
14 + })
```

The `scrollspy-options.ts` file contains the following code:

```
... @@ -0,0 +1,10 @@
1 + import { ChangeDetectorRef } from '@angular/core';
2 + import { NgbScrollSpyRef } from './scrollspy';
3 +
4 + export type NgbScrollSpyProcessChanges = {
5 +   state: {
6 +     entries: IntersectionObserverEntry[];
7 +     rootElement: HTMLElement;
8 +     fragments: Set<Element>;
9 +     scrollSpy: NgbScrollSpyRef; // REFACTOR: Usando interface em vez da classe concreta
10 +     options: NgbScrollSpyOptions;
11 +   },
12 +   changeActive: (active: string) => void,
13 + }
```

Desafios Enfrentados

1

Dependências Circulares e Modularização

Ao dividir arquivos grandes, surgiu o desafio de resolver dependências circulares entre diretivas e serviços que antes conviviam no mesmo arquivo. A refatoração exigiu extrair interfaces, mover tipos para módulos auxiliares e reorganizar imports, garantindo que cada parte pudesse ser carregada de forma independente sem gerar ciclos ou erros de compilação.

2

Tipagem em Bibliotecas Internas

Substituir o `any` exigiu **compreensão profunda dos tipos internos do Angular** e de suas dependências. Encontrar os tipos corretos para garantir a segurança sem introduzir complexidade excessiva foi um processo minucioso de investigação e experimentação.

3

Animações e Acesso ao DOM

Manter as **animações complexas** do ng-bootstrap funcionando corretamente sem acesso direto ao elemento nativo do DOM foi um ponto crítico. Foi necessário abstrair cuidadosamente as interações para respeitar a arquitetura do Angular e evitar regressões visuais.

Resultados e Conclusão

Impacto da Refatoração

A refatoração focou na melhoria estrutural e funcional, com muitos ganhos significativos, incluindo:

- Maior coesão entre os módulos.
- Menor acoplamento entre os componentes.
- Melhora substancial na testabilidade.
- Aumento da legibilidade e manutenibilidade.

Essas melhorias contribuem diretamente para a **longevidade e adaptabilidade** do projeto ng-bootstrap.

Aprendizados Chave

- A importância de **preferir composição a herança**, principalmente em componentes que só compartilham dados.
- O papel da **coesão** na legibilidade: após dividir arquivos grandes, as responsabilidades ficaram mais evidentes.
- Como pequenos smells podem gerar **acoplamentos invisíveis**, como dependências circulares que só se revelam ao modularizar.
- Que **tipagem forte** não é burocracia: ela reduz incertezas e facilita a navegação no código.