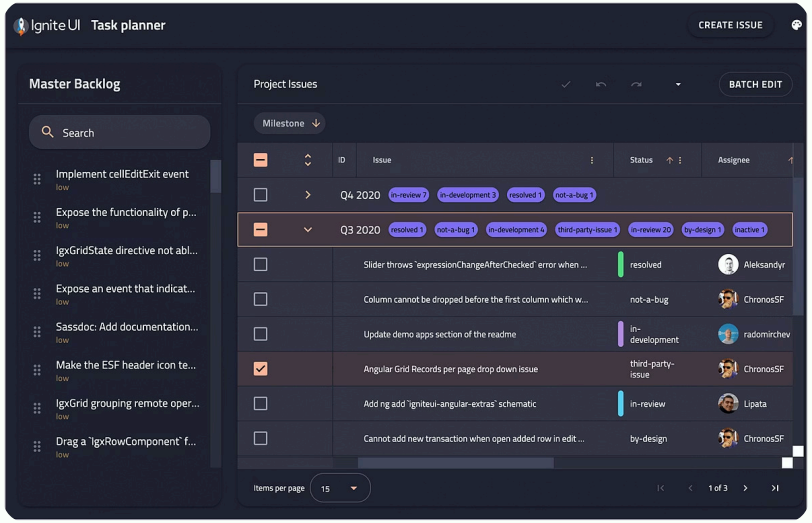
UFC

Refatoração de code smells no Frameworks Ignite UI

Disciplina de Qualidade de Software

José Iago da Silva Lima

Universidade Federal do Ceará



O Projeto Alvo: Ignite UI for Angular

O Ignite UI for Angular é uma biblioteca robusta de componentes UI para o desenvolvimento de aplicações web . Sua praticidade e abrangência o tornam um excelente estudo de caso para análise e refatoração de code smells em um ambiente de framework real.

Explore o fork do projeto para ver as refatorações: https://github.com/lagoLSJ/igniteui-angular

Detecção de Code Smells

Para identificar os code smells foi utilizado uma ferramenta de detecção

```
$ npx tsx lib/index.ts ~/dev/igniteui-angular/projects/igniteui-angular/date-picker/src/date-range-picker/
C:\Users\Iago\dev\angular-code-smells
```

(index)	file	DOM	IIC	LC	ANY	TMI	LF	EPC
0	'..\\igniteui-angular\\projects\\igniteui-angular\\date-picker\\src\\date-range-picker\\date-range-picker-inputs.common.ts'	1	1	1	0	0	0	0
1	'..\\igniteui-angular\\projects\\igniteui-angular\\date-picker\\src\\date-range-picker\\date-range-picker.component.spec.ts'	1	1	0	1	0	1	0
2	'..\\igniteui-angular\\projects\\igniteui-angular\\date-picker\\src\\date-range-picker\\date-range-picker.component.ts'	1	1	1	1	0	1	1
3	'..\\igniteui-angular\\projects\\igniteui-angular\\date-picker\\src\\date-range-picker\\predefined-ranges\\predefined-ranges-area-component.spec.ts'	1	0	0	0	0	0	0
4	'..\\igniteui-angular\\projects\\igniteui-angular\\date-picker\\src\\date-range-picker\\predefined-ranges\\predefined-ranges-area.component.ts'	0	0	0	1	0	0	0

Quantitativos

Total de smells encontrados : 1305

DOM	IIC	LC	ANY	TMI	LF	EPC
293	130	181	280	10	369	42

Tipos de smells escolhidos e quantidade refatorada:

- DOM | 10
- LF | 10+
- ANY | 10+
- TMI | 10

Code Smell: Any Type

O que é o Any Type?

O uso excessivo do tipo `any` no TypeScript anula os benefícios da tipagem estática, reduzindo a segurança do código e aumentando a probabilidade de erros em tempo de execução. Ele compromete a clareza e a manutenibilidade do sistema.

Refatoração: Interfaces Tipadas e Union Types

A substituição de `any` por **Interfaces Tipadas** e **Union Types** garante a segurança e a clareza do código.

src/app/shared/remote.service.ts

```
// Antes: any
public remotePagingData:BehaviorSubject<any[]>;

// Depois: Interface Tipada
export interface Product {
    ProductID: number;
    ProductName: string;
    UnitPrice: number;
    UnitsInStock: number;
}

public remotePagingData:BehaviorSubject<Product[]>;
```

A tipagem adequada melhora a detecção de erros em tempo de compilação, facilitando o desenvolvimento e a manutenção.

Code Smell: Too Many Inputs (TMI)

Problema: Excesso de @Input()

Componentes com mais de 8 decoradores `@Input()` tornam-se difíceis de entender e manter. Isso indica alta complexidade e baixa coesão, comprometendo a legibilidade e a capacidade de reutilização.

Desafio: Agrupamento

Agrupar inputs sem quebrar os *bindings* existentes nos templates é um desafio. A modificação de um componente central pode gerar um efeito cascata de alterações em toda a aplicação.

Solução: Interfaces Contextuais

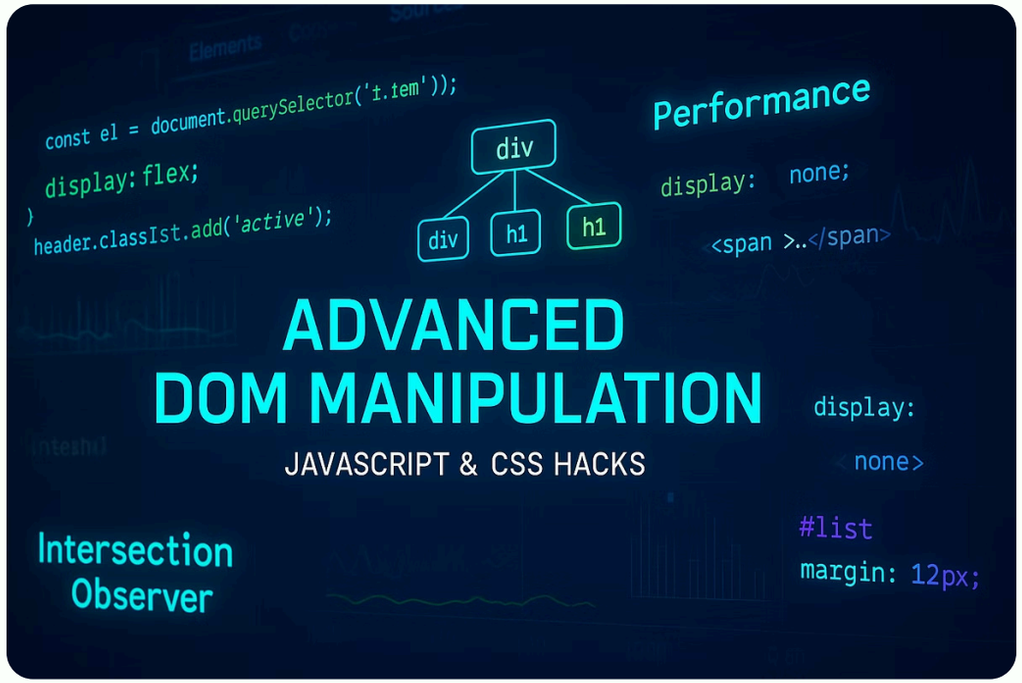
A refatoração envolve agrupar os inputs em interfaces tipadas contextuais. Isso reduz a quantidade de `@Input()`, melhora a organização e a clareza do componente, e facilita a manutenção futura.

```
projects/igniteui-angular/chips/src/chips/chip.config.ts
import { TemplateRef } from '@angular/core';
import { IChipResourceStrings } from 'igniteui-angular/core';

export type IgxChipTypeVariantConfig = 'primary' | 'info' | 'success' | 'warning' | 'danger' | null;

export interface IgxChipConfig {
    variant?: IgxChipTypeVariantConfig;
    id?: string;
    tabIndex?: number | null;
    data?: any;
    draggable?: boolean;
    animateOnRelease?: boolean;
    hideBaseOnDrag?: boolean;
    removable?: boolean;
    removeIcon?: TemplateRef<any>;
    selectable?: boolean;
    selectIcon?: TemplateRef<any>;
    class?: string;
    disabled?: boolean;
    selected?: boolean;
    color?: string;
    resourceStrings?: IChipResourceStrings;
```

Code Smell: DOM Manipulation

O Problema da Manipulação Direta

Acesso direto ao **DOM** (`nativeElement`, `window`, `document`) em componentes Angular quebra o padrão do framework. Isso dificulta a testabilidade, introduz dependências globais e impede a renderização do lado do servidor (SSR).



Refatoração: Bindings e Diretivas

A solução consiste em mover a lógica visual para **bindings** (propriedades, eventos) e **diretivas** (estruturais, de atributo). Isso permite que o Angular gerencie o **DOM** de forma otimizada e portável, eliminando as dependências globais e mantendo a reatividade.

```
// Antes:
constructor(private el: ElementRef) {
  this.el.nativeElement.style.backgroundColor = 'blue';
}

// Depois (usando HostBinding):
@HostBinding('style.backgroundColor')
backgroundColor = 'blue';
```

Code Smell: Large File (LF)

Identificação: Arquivos Volumosos

Arquivos com mais de 240 linhas de código e múltiplas responsabilidades são um indicativo de **Large File**. Essa característica compromete a legibilidade, dificulta a depuração e aumenta a complexidade de manutenção.

Refatoração: Modularização

A estratégia de refatoração consiste em dividir o código em partes coesas, com responsabilidades bem definidas. Isso envolve a criação de novos serviços, *helpers* e utilitários, promovendo o princípio da responsabilidade única.

Benefícios da Divisão

A modularização não apenas reduz o tamanho dos arquivos, mas também melhora a organização do projeto, facilita a reutilização de código e otimiza o trabalho em equipe, tornando o código mais escalável.

Componentes menores e bem definidos são mais fáceis de testar, manter e evoluir ao longo do tempo.

Conclusão e Resultados da Refatoração

Melhora na Legibilidade e Manutenção

A eliminação de **Code Smells** como `any type` e **Large File** resultou em um código mais claro, fácil de entender e, consequentemente, mais simples de manter e evoluir.

Aumento da Segurança e Robustez

Com a tipagem estrita e a manipulação controlada do **DOM**, a aplicação se tornou mais robusta, com menos chance de erros em tempo de execução e maior segurança nas operações.

Facilidade de Testabilidade

Componentes menores e com responsabilidades únicas são intrinsecamente mais fáceis de testar, promovendo a criação de suítes de testes mais eficazes e confiáveis.

A refatoração de **Code Smells** é um investimento contínuo na qualidade do software, garantindo um produto final mais sustentável e de alto desempenho.

Obrigado!

Universidade Federal do Ceará

Qualidade de Software: Construindo o Futuro da Engenharia