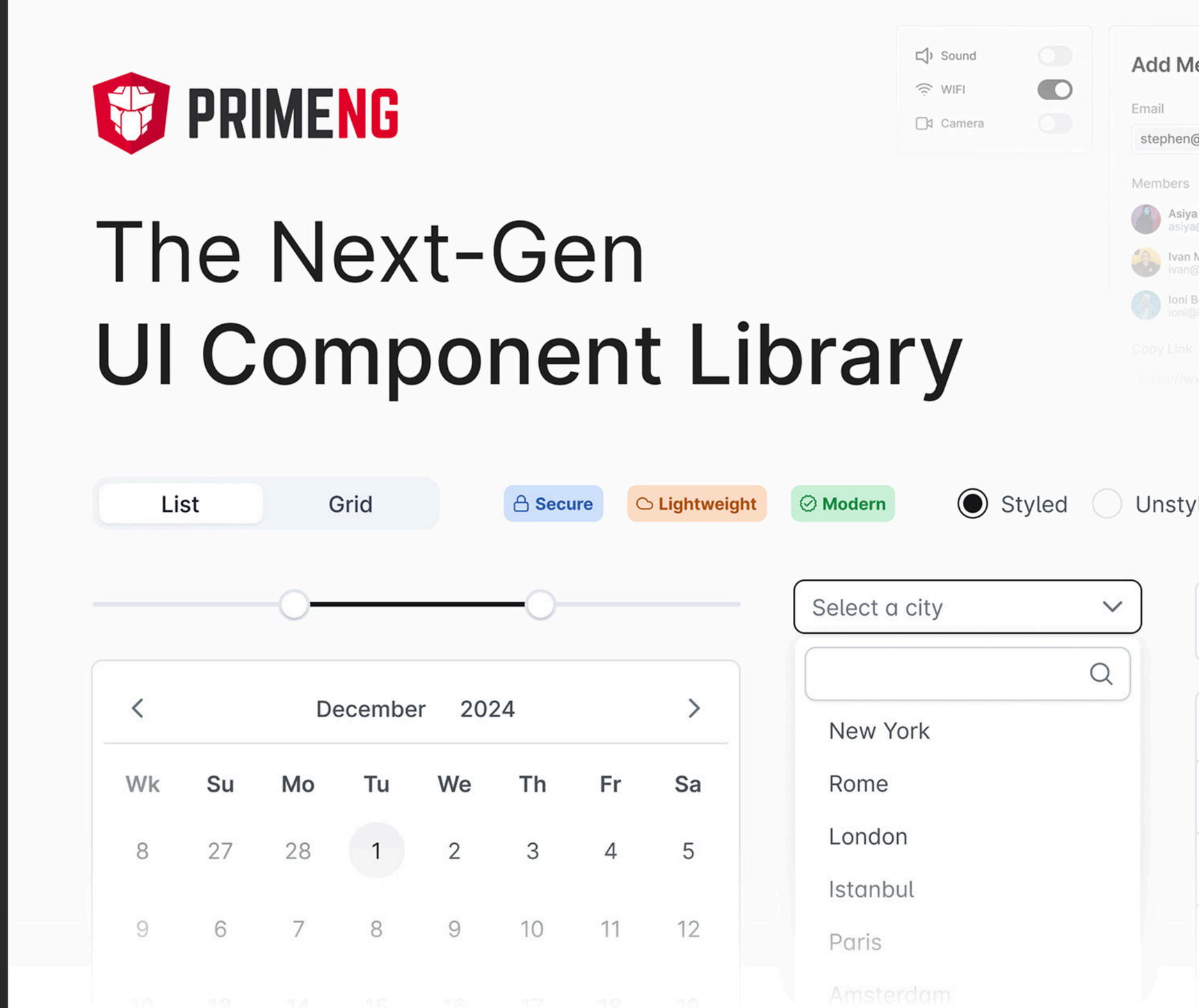
Code Smells em Frameworks Front-end

Ester Ravette e Kelton Martins

O Projeto

Biblioteca de componentes de UI para Angular, baseada em uma infraestrutura **agnóstica ao design**.

+400 milhões de downloads no npm



Categorias de Code Smells

1

DOM

Manipulação direta do DOM.

2

TMI

Componente com excesso de inputs.

3

LF

Arquivo com tamanho excessivo.

4

IIC

Uso inadequado de herança em vez de composição.

5

LC

Componente com responsabilidades excessivas.

6

EPC

Comunicação excessiva entre componentes pai e filho.

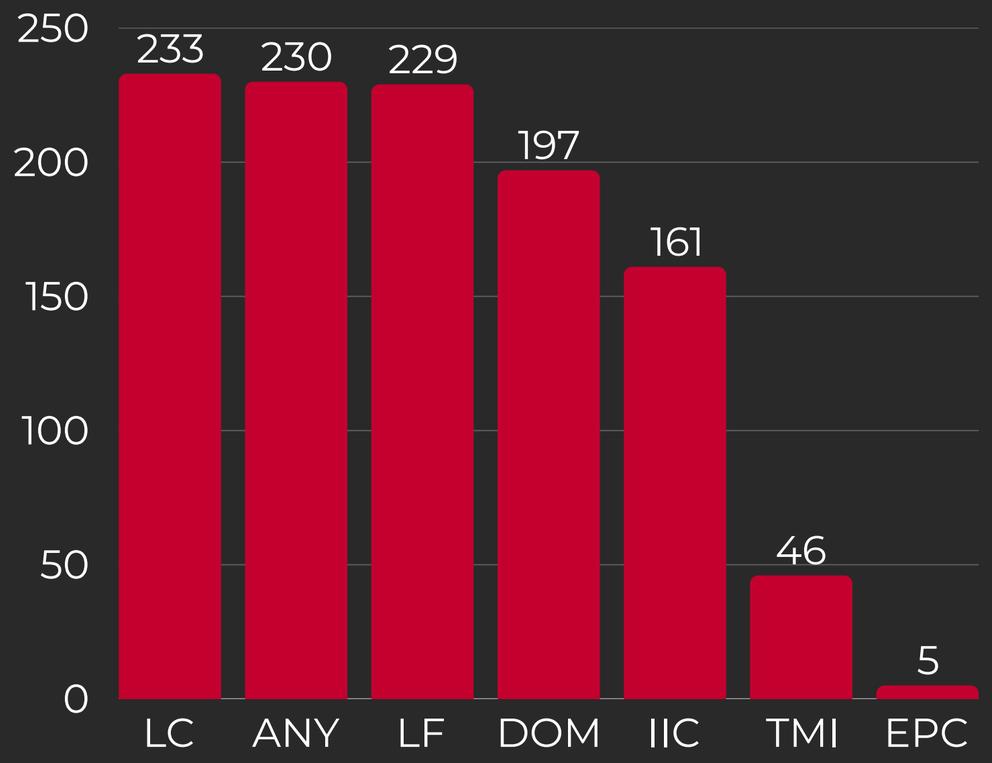
7

ANY

Uso excessivo de any, comprometendo a segurança de tipos.

502
arquivos

1.601
ocorrências



Code Smells Refatorados

DOM	IIC	ANY	TMI
10	10	11	10

Arquivos Refatorados

1. animateonscroll.ts
2. autofocus.ts
3. badge.ts
4. bind.ts
5. card.ts
6. deferreddemo.ts
7. ripple.ts
8. terminal.ts

DOM

1. avatar.ts
2. animateonscroll.ts
3. badge.ts
4. timeline.ts
5. toggleswitch.ts
6. iconfield.ts
7. buttongroup.ts
8. divider.ts
9. floatlabel.ts
10. fluid.ts

IIC

1. index.ts
2. disableddoc.ts
3. filleddoc.ts

ANY

1. app.code.ts
2. app.docapitable.ts
3. button.ts
4. carousel.ts
5. checkbox.ts
6. colorpicker.ts
7. confirmdialog.ts
8. contextmenu.ts
9. dataview.ts
10. drawer.ts

TMI

Code Smell: **DOM**

O código acessava diretamente o nativeElement, acoplando o componente ao DOM do navegador e quebrando a compatibilidade com SSR. A refatoração substituiu esse acesso pelo Renderer2, que abstrai a manipulação do DOM, garantindo que o Angular gerencie a manipulação de forma segura em qualquer ambiente.

```
onAfterContentChecked() {
    if (this.autofocus === false) {
        this.host.nativeElement.removeAttribute('autofocus');
    } else {
        this.host.nativeElement.setAttribute('autofocus', true);
    }

    if (!this.focused) {
        this.autoFocus();
    }
}
```

ANTES

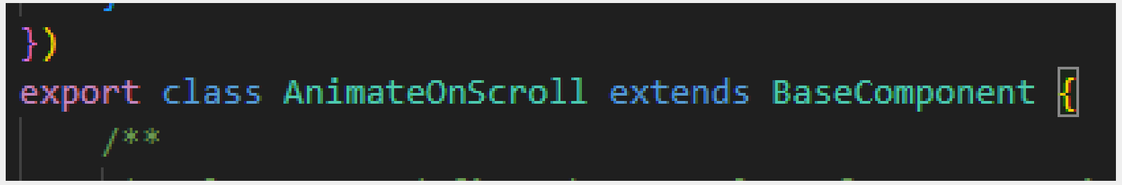
```
onAfterContentChecked() {
    if (this.autofocus === false) {
        // uso do Renderer2 em vez de this.host.nativeElement.removeAttribute
        this.renderer.removeAttribute(this.host.nativeElement, 'autofocus');
    } else {
        // uso do Renderer2 em vez de this.host.nativeElement.setAttribute
        this.renderer.setAttribute(this.host.nativeElement, 'autofocus', 'true');
    }

    if (!this.focused) {
        this.autoFocus();
    }
}
```

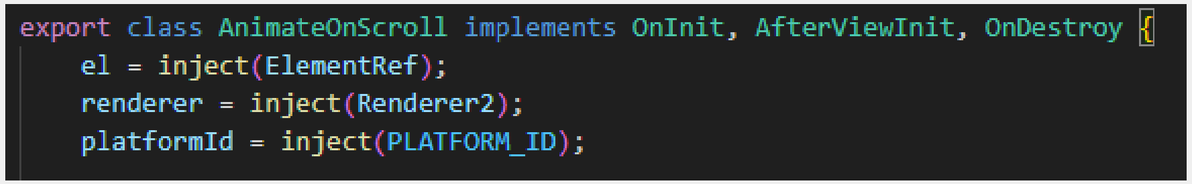
DEPOIS

Code Smell: **IIC**

O componente utilizava herança de uma classe base apenas para acessar utilitários de estilo, criando um acoplamento rígido e ocultando dependências reais. A refatoração removeu a herança e adotou composição através de injeção de dependência explícita (inject), tornando o componente mais desacoplado, legível e fácil de manter.



ANTES



DEPOIS

Code Smell: **ANY**

O código utilizava o tipo genérico any indiscriminadamente, anulando a verificação de segurança do TypeScript. A refatoração foi substituir essas ocorrências por tipos específicos, recuperando a segurança de tipos, previsibilidade e prevenindo erros silenciosos.

```
items: any[] | undefined;

selectedItem: any;

suggestions: any[] | undefined;
```

ANTES

```
items: string[] | undefined;

selectedItem: string | undefined;

suggestions: string[] | undefined;
```

DEPOIS

Code Smell: **TMI**

O componente possuía muitos inputs isolados. A refatoração foi agrupar em uma interface de configuração, simplificando e melhorando a organização dos dados.

```
export class AppCode {
    @Input() code!: Code;

    @Input() service!: any;

    @Input() selector!: string;

    @Input() extFiles: ExtFile[] = [];

    @Input() routeFiles: RouteFile[] = [];

    @Input({ transform: booleanAttribute }) hideToggleCode: boolean = false;

    @Input({ transform: booleanAttribute }) hideCodeSandbox: boolean = true;

    @Input({ transform: booleanAttribute }) hideStackBlitz: boolean = false;

    @Input({ transform: booleanAttribute }) importCode: boolean = false;
```

ANTES

```
//interface pra agrupar e diminuir os inputs.
export interface AppCodeConfig {
    hideToggleCode?: boolean;
    hideCodeSandbox?: boolean;
    hideStackBlitz?: boolean;
    importCode?: boolean;
}
```

```
export class AppCode {
    @Input() code!: Code;

    @Input() service!: any;

    @Input() selector!: string;

    @Input() extFiles: ExtFile[] = [];

    @Input() routeFiles: RouteFile[] = [];

    //agrupamento de inputs usando a interface.
    @Input() config: AppCodeConfig = {
        hideToggleCode: false,
        hideCodeSandbox: true,
        hideStackBlitz: false,
        importCode: false
    };
```

DEPOIS

Obrigado!