

QUALIDADE DE SOFTWARE

# Official components for Angular

Refatoração de Code Smells - Angular

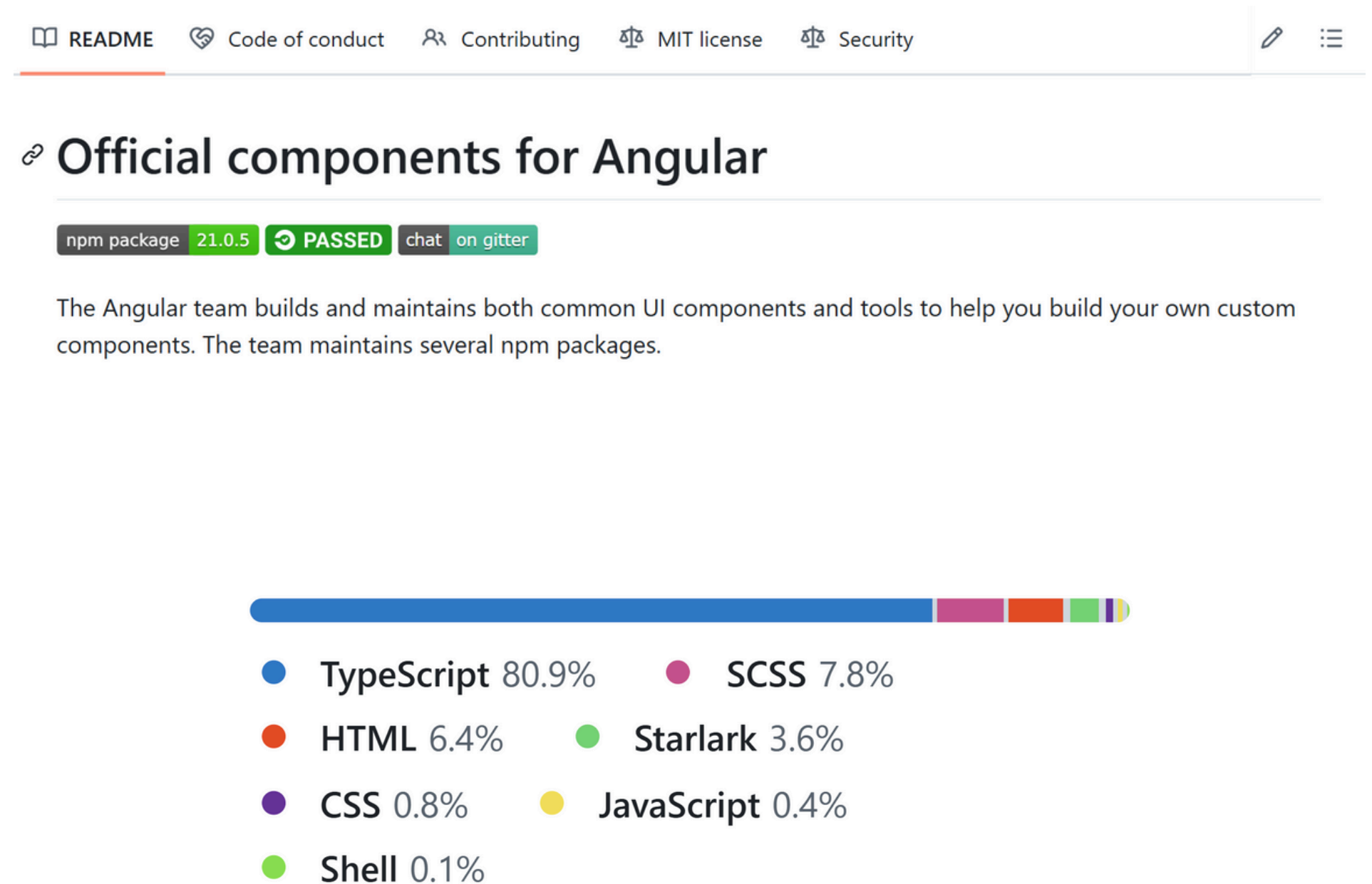
Daniel Chistopher  
Miquéias Bento

# Sobre o projeto

Trata-se de uma biblioteca oficial de componentes UI para Angular mantida pela equipe do framework. Ele contém uma grande variedade de componentes prontos, como botões, abas, seletores, checkboxes e tabelas, amplamente utilizados em aplicações Angular.

Algumas características:

- Grande quantidade de código e diversidade de padrões;
- Alta detecção de code smells.
- Possui uma alta complexidade para a aplicação das técnicas de refatoração e validação da ferramenta de detecção de code smells.



# A detecção de smells

O detector de code smells identificou 993 violações de qualidade distribuídas em 576 arquivos do projeto. O code smell mais frequente é o uso indiscriminado do tipo **Direct DOM Manipulation**.

Foram identificadas as seguintes ocorrências:

- 357 **Direct DOM Manipulation**,
- 103 **Inheritance instead of Composition**,
- 102 **Large Component**,
- 103 **Overusing Any Type**,
- 07 **Too Many Inputs**,
- 309 **Large File**,
- 12 **Excessive Parent-to-Child Communication**.

# Overusing Any Type - Refatoração

O code smell **Overusing Any Type** ocorre quando variáveis, parâmetros ou retornos de funções usam o tipo genérico any em TypeScript de forma excessiva. Isso enfraquece a verificação de tipos do compilador, reduz a segurança e aumenta a chance de erros em tempo de execução.

Antes

```
const options = allOptions.filter(o => o.searchTerm().startsWith(text));
if (popup instanceof ComboboxListboxPattern) {
  (popup.inputs.items as WritableSignal<any[]>).set(options);
} else if (popup instanceof ComboboxTreePattern) {
  (popup.inputs.allItems as WritableSignal<any[]>).set(options);
}
```

Depois

```
78     const options = allOptions.filter(o => o.searchTerm().startsWith(text));
79     if (popup instanceof ComboboxListboxPattern) {
80 +   (popup.inputs.items as WritableSignal<TestOption[]>).set(options as
      TestOption[]);
81   } else if (popup instanceof ComboboxTreePattern) {
82 +   (popup.inputs.allItems as WritableSignal<TreeItemPattern<string>
      []>).set(options as TreeItemPattern<string>[]);
83   }
```

# Overusing Any Type - Refatoração

## Métodos de refatoração

A refatoração desse smell envolve substituir any por tipos específicos ou interfaces mais apropriadas, melhorando a legibilidade, a robustez e a manutenibilidade do código, além de aproveitar os benefícios de tipagem do TypeScript.

## Dificuldades

O maior desafio foi a alta recorrência do problema em um único arquivo e a necessidade de entender o domínio dos dados envolvidos, o que exigiu leitura de contratos internos e, em alguns casos, a criação de novos tipos.

# Too Many Inputs - Refatoração

O code smell **Too Many Inputs** acontece quando um componente Angular declara um número excessivo de propriedades com o decorador `@Input()`. Isso torna a interface pública do componente extensa, complexa e mais difícil de usar e manter.

Antes

```
@Component({ ... })
export class MatSelect {
  @Input() panelClass: string;
  @Input() disabled: boolean;
  @Input() disableRipple: boolean;
  @Input() tabIndex: number;
  @Input() hideSingleSelectionIndicator: boolean;
  @Input() placeholder: string;
  @Input() required: boolean;
  @Input() multiple: boolean;
  @Input() disableOptionCentering: boolean;
  @Input() typeaheadDebounceInterval: number;
  @Input() sortComparator: ...;
  @Input() id: string;
  @Input('aria-label') ariaLabel: string;
  @Input('aria-labelledby') ariaLabelledby: string;
  @Input() errorStateMatcher: ...;
  // [...]
}
```

# Too Many Inputs - Refatoração

O code smell **Too Many Inputs** acontece quando um componente Angular declara um número excessivo de propriedades com o decorador `@Input()`. Isso torna a interface pública do componente extensa, complexa e mais difícil de usar e manter.

Depois

```
@Component({ ... })
export class MatSelect {
  @Input()
  get config(): MatSelectConfig { ... }
  set config(c: MatSelectConfig) { ... }

  @Input()
  get state(): MatSelectState { ... }
  set state(s: MatSelectState) { ... }

  @Input()
  get aria(): MatSelectAria { ... }
  set aria(a: MatSelectAria) { ... }
  // [...]
}
```

# Too Many Inputs - Refatoração

## Métodos de refatoração

Para refatorar esse smell, os inputs relacionados são agrupados em objetos de configuração coerentes, reduzindo o número de entradas expostas e melhorando a coesão e a simplicidade da API dos componentes, sem alterar seu comportamento funcional.

## Dificuldades

O principal desafio foi garantir que o comportamento original fosse mantido, já que algumas propriedades acionavam lógicas internas quando eram alteradas. Por isso, foi necessário mapear cuidadosamente esses valores e preservar as ações já existentes.



# Direct DOM Manipulation - Refatoração

O Direct DOM Manipulation acontece quando um componente Angular acessa ou altera o DOM diretamente, usando APIs. Isso ignora os mecanismos que o próprio Angular oferece para lidar com a interface.

Antes

```
ngOnInit() {  
  const container = document.querySelector('#scene-content-container') as HTMLElement;  
  if (container) {  
    container.style.transform = `scale(${this.scale})`;   
    container.style.transformOrigin = 'top left';  
  }  
  this.scene().createComponent(this.component);  
}
```

Depois

```
ngOnInit() {  
  this.scene().createComponent(this.component);  
}
```

```
<div id="scene-content-container"  
  [style.transform]="scale(' + scale + ')"  
  [style.transformOrigin]="top left">  
  <ng-template #scene />  
</div>
```

# Direct DOM Manipulation - Refatoração

## Métodos de refatoração

A refatoração consiste em substituir essas manipulações por data binding, controle de estado e APIs recomendadas pelo próprio framework.

## Dificuldades

A principal dificuldade foi o forte acoplamento com o DOM, tendo que modificar trechos de outros arquivos HTML, e a recorrência em vários pontos do mesmo arquivo.

# Large File - Refatoração

É um code smell que ocorre quando um **único arquivo** cresce demais, concentrando muitas responsabilidades, regras e detalhes

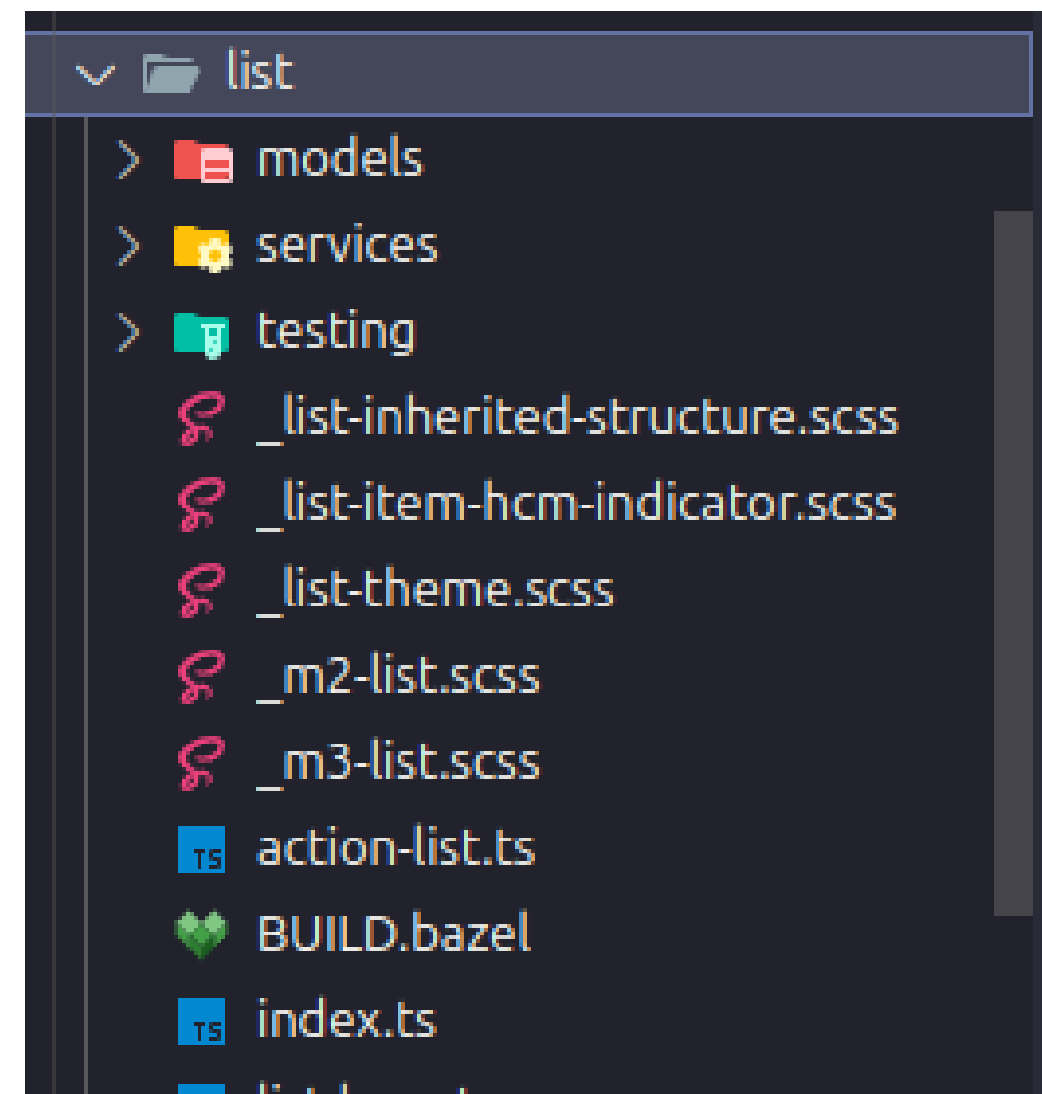
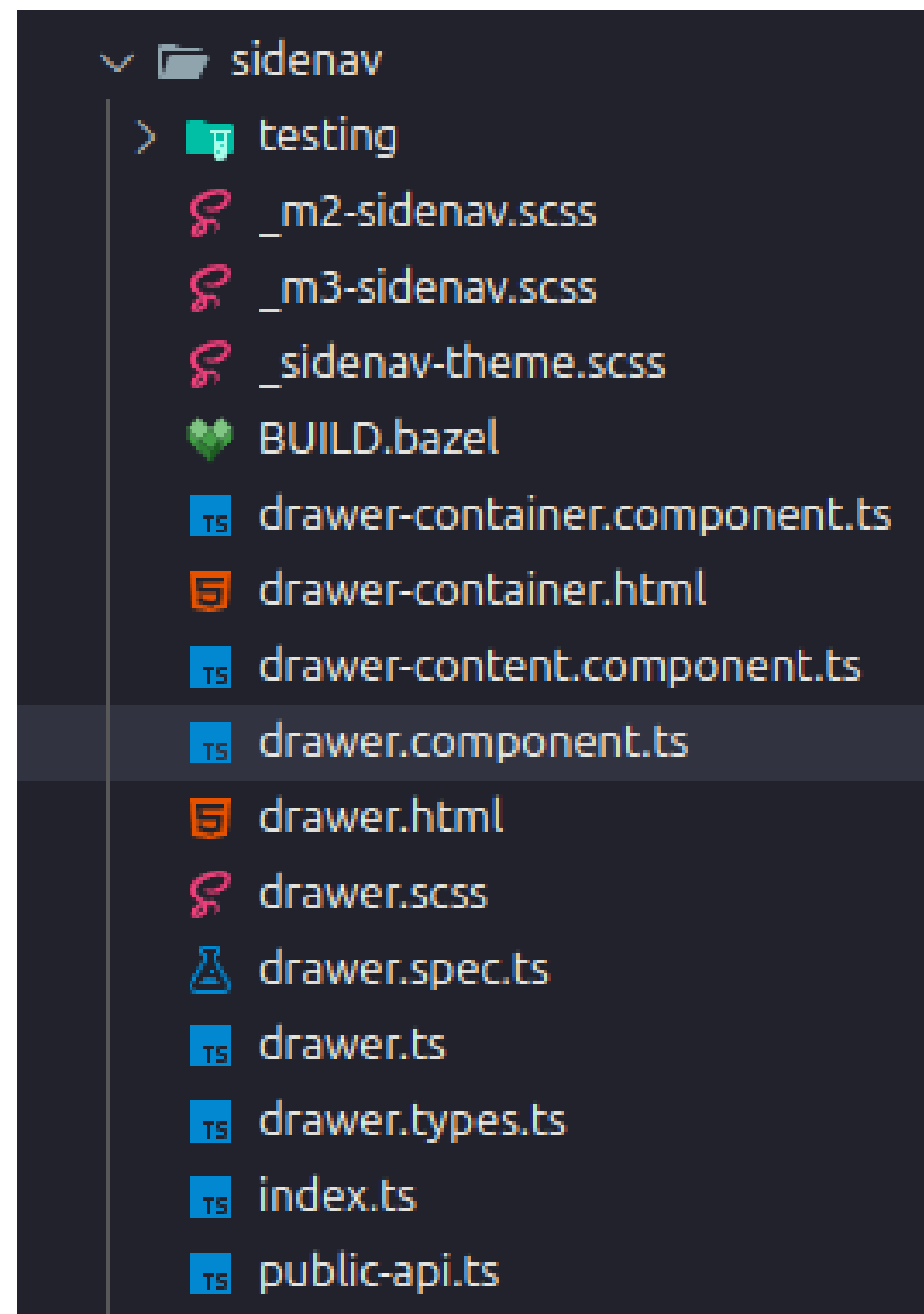
Antes

```
68  /** Configures whether drawers should use auto sizing by default. */
69  export const MAT_DRAWER_DEFAULT_AUTOSIZE = new InjectionToken<boolean>(
70 > 'MAT_DRAWER_DEFAULT_AUTOSIZE', ...
74  },
75  );
76
77  /**
78   * Used to provide a drawer container to a drawer while avoiding circular references.
79   * @docs-private
80   */
81  export const MAT_DRAWER_CONTAINER = new InjectionToken('MAT_DRAWER_CONTAINER');
82
83 > @Component({ ...
100  })
101 > export class MatDrawerContent extends CdkScrollable implements AfterContentInit { ...
139  }
140
141  /**
142   * This component corresponds to a drawer that can be opened on the drawer container.
143   */
144 > @Component({ ...
167  })
168 > export class MatDrawer implements AfterViewInit, OnDestroy { ...
667  }
668
669  /**
670   * `<mat-drawer-container>` component.
671   *
672   * This is the parent component to one or two `<mat-drawer>`s that validates the state internally
673   * and coordinates the backdrop and content styling.
674   */
675 > @Component({ ...
693  })
694 > export class MatDrawerContainer implements AfterContentInit, DoCheck, OnDestroy { ...
1064  }
1065
```

# Large File - Refatoração

É um code smell que ocorre quando um **único arquivo** cresce demais, concentrando muitas responsabilidades, regras e detalhes

Depois



# Large File - Refatoração

## Métodos de refatoração

A refatoração desse code smell pode ser feito de diferentes maneiras, os métodos que mais usei foram o *extract* class, que consiste em separar responsabilidades em classes menores e mais coesas, e o *module decomposition*, que consiste em separar em camadas o arquivo de forma que deixe a sua responsabilidade mais explícita.

## Dificuldades

A maior dificuldade foi de lidar com arquivos de mais de 800 linhas de código no qual estava totalmente acoplado. Entender o código com muita dificuldade de leitura e imaginar como separá-lo foi desafiador.

# Large Component - Refatoração

Large Component é um code smell **específico de UI** que acontece quando um único componente assume responsabilidades demais, indo além de renderização e orquestração.

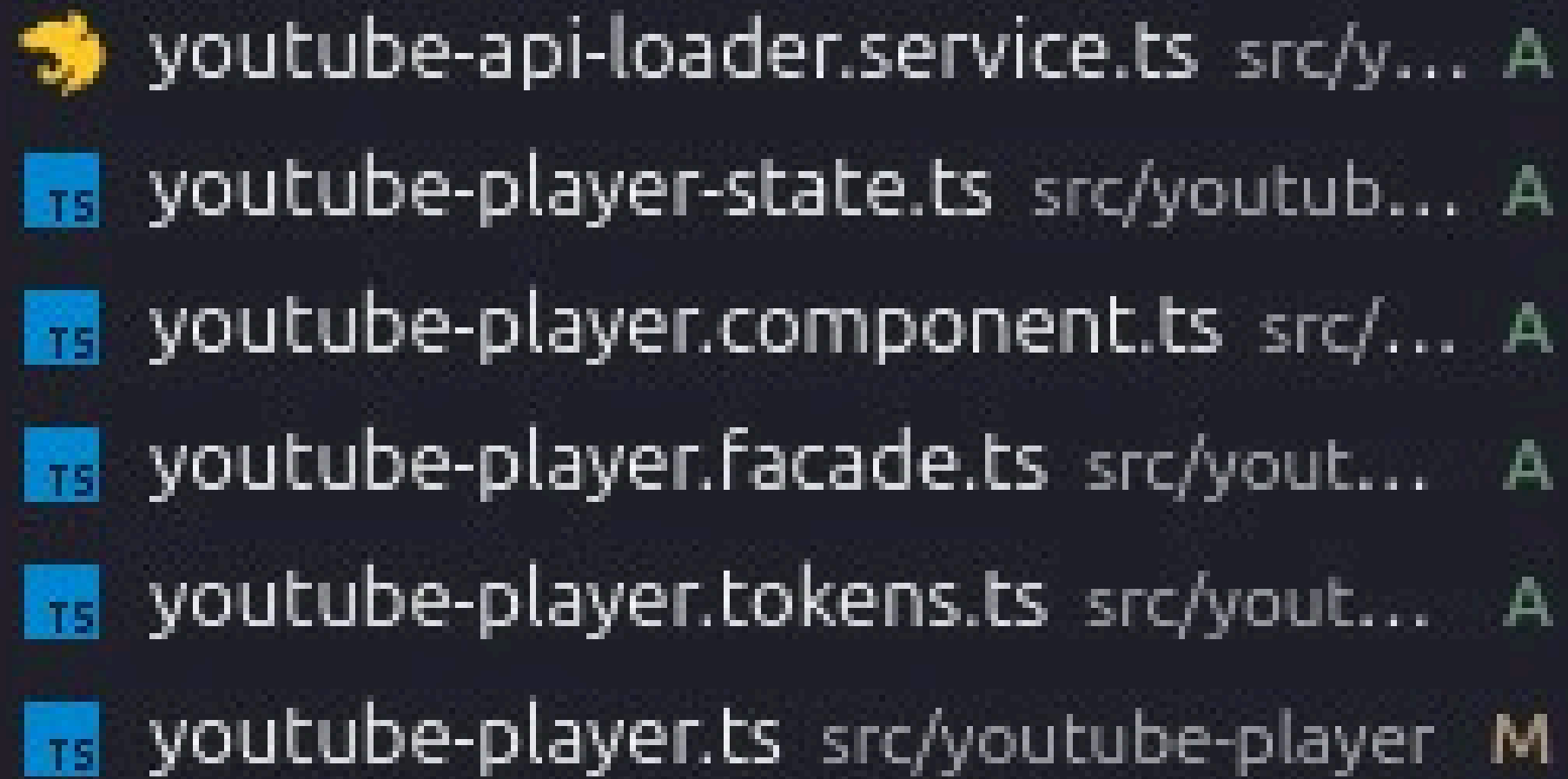
Antes

```
90 /** Coercion function for time values. */
91 function coerceTime(value: number | undefined): number | undefined {
92   return value == null ? value : numberAttribute(value, 0);
93 }
94
95 /**
96  * Equivalent of `YT.PlayerState` which we can't use, because it's meant to
97  * be read off the `window` which we can't do before the API has been loaded.
98  */
99 enum PlayerState {
100   UNSTARTED = -1,
101   ENDED = 0,
102   PLAYING = 1,
103   PAUSED = 2,
104   BUFFERING = 3,
105   CUED = 5,
106 }
107
108 /**
109  * Angular component that renders a YouTube player via the YouTube player
110  * iframe API.
111  * @see https://developers.google.com/youtube/iframe\_api\_reference
112  */
113 > @Component({...
134   })
135 > export class YouTubePlayer implements AfterViewInit, OnChanges, OnDestroy {...
742   }
743
744   let apiLoaded = false;
745
746   /** Loads the YouTube API from a specified URL only once. */
747 > function loadApi(nonce: string | null): void {...
780   }
```

# Large Component - Refatoração

Large Component é um code smell **específico de UI** que acontece quando um único componente assume responsabilidades demais, indo além de renderização e orquestração.

Depois



```
youtube-api-loader.service.ts src/y... A
TS youtube-player-state.ts src/youtub... A
TS youtube-player.component.ts src/... A
TS youtube-player.facade.ts src/yout... A
TS youtube-player.tokens.ts src/yout... A
TS youtube-player.ts src/youtube-player M
```

# Large Component - Refatoração

## Métodos de refatoração

O método principal de refatoração desse code smell são o *Extract Presentational Component*, que se baseia em extrair partes reutilizáveis do código original; e aplicar o *Composition Pattern*, que consiste em construir o componente grande **composto** com componentes menores e independentes, ao invés de concentrar tudo em um único componente.

## Dificuldades

Haviam muitos componentes que juntavam regras de negócio com UI, fazer a separação foi desafiador.



# Refatoração geral

**10** → [ANY] Overusing Any Type

**7** → [TMI] Too Many Inputs

**3** → [DOM] Direct DOM Manipulation

**10** → [LF] Large File

**10** → [LC] Large Component

## QUALIDADE DE SOFTWARE

**Agradecemos  
pela  
atenção**



UNIVERSIDADE  
FEDERAL DO CEARÁ  
CAMPUS QUIXADÁ