UNIVERSIDADE FEDERAL DO CEARÁ

Code smells em frameworks front-end

Disciplina: Qualidade de Software
Equipe: Anaildo Nascimento SIlva e João Evangelista
Projeto: NG-Zorro
Fork: https://github.com/joaoev/ng-zorro-antd/tree/developer
Divisão: 40 smell de 4 tipos escolhido cada um refatorou 10 - LC, TMI, LF e DOM.

Anaildo Nascimento Silva - 552836

Eu estou atualmente trabalhando na refatoração do seguinte code smell:

Large Component (Componente Grande) -

O code smell "Large Component" ocorre quando um componente Angular excede um tamanho recomendado (geralmente acima de 140 linhas, considerando código TypeScript + template HTML). Este smell indica que o componente está assumindo muitas responsabilidades, violando o princípio de responsabilidade única (Single Responsibility Principle).

O problema principal é quando componentes apresentam templates HTML grandes escritos diretamente dentro do arquivo TypeScript, misturando lógica de negócio com apresentação em um único arquivo. Isso torna o código difícil de ler, manter e modificar, além de dificultar a reutilização e os testes.

Minhas principais dificuldades na remoção do code smell são:

Dificuldade 1: Identificação do que extrair
Decidir exatamente o que pode ser movido para arquivos externos sem quebrar funcionalidades. É necessário analisar todas as dependências do template (variáveis, eventos, diretivas) para garantir que continuarão funcionando após a separação.

Dificuldade 2: Manutenção de funcionalidades
Garantir que todas as funcionalidades (animações, eventos, bindings dinâmicos, renderização condicional) continuem operando corretamente após mover o template para arquivo externo ou extrair lógica para serviços.

Dificuldade 3: Compatibilidade com sistema existente

Componentes podem estender outras classes e utilizar recursos do Angular como viewChild, destroyRef e animações. É necessário garantir que a refatoração não afete essas dependências e heranças.

Dificuldade 4: Otimização sem perda de funcionalidade
Encontrar o equilíbrio entre reduzir linhas de código, manter todas as funcionalidades e melhorar a organização sem perder clareza ou legibilidade.

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

Método 1: Extract Template (Extração de Template)

O que foi feito:* Templates HTML grandes que estavam dentro do arquivo TypeScript são movidos para arquivos separados (ex: component.component.html). O decorator @Component é modificado para usar templateUrl em vez de template.

- **Benefícios:** Separação clara entre lógica (TypeScript) e apresentação (HTML), melhor legibilidade e facilita manutenção futura.
- **Resultado:** Redução significativa no número de linhas do arquivo TypeScript principal.

Método 2: Method Simplification (Simplificação de Métodos)

- **O que foi feito:** Métodos do componente são convertidos para arrow functions quando apropriado, tornando o código mais compacto sem alterar funcionalidade.
- **Benefícios:** Redução no número de linhas de código e sintaxe mais moderna e consistente.
- **Resultado:** Redução no tamanho do código mantendo a mesma funcionalidade.

Método 3: Template Compaction (Compactação de Template)

- **O que foi feito:** Templates HTML podem ser compactados removendo espaços em branco desnecessários sem perder funcionalidade, facilitando a contagem de linhas para métricas.
- **Benefícios**: Redução significativa no tamanho do arquivo e facilita análise de métricas.
- **Resultado:** Redução no tamanho do arquivo de template mantendo toda funcionalidade.

Método 4: Code Organization (Organização de Código)

- **O que foi feito:** Código é reorganizado para seguir uma estrutura mais clara, com imports organizados e propriedades agrupadas logicamente.
- **Benefícios:** Código mais profissional, facilita navegação e segue convenções do Angular.
- **Resultado:** Melhor estrutura e organização geral do componente.

Método 5: Extract Service (Extração para Serviços)

- **O que foi feito:** Lógica de negócio complexa é extraída para serviços dedicados, deixando o componente mais focado apenas na apresentação e coordenação.
- **Benefícios:** Separação de responsabilidades, facilita testes e reutilização de lógica.
- **Resultado:** Componente mais enxuto e focado, com lógica de negócio em serviços reutilizáveis.

Anaildo Nascimento Silva - 552836

Eu estou atualmente trabalhando na refatoração do seguinte code smell:

Too Many Inputs" (Muitos Parâmetros de Entrada)

Este code smell ocorre quando um componente Angular possui muitas propriedades @Input (mais de 8 propriedades, conforme nosso detector)..

O problema principal é que quando um componente tem muitas propriedades de entrada, fica difícil visualizar quais propriedades estão relacionadas entre si e entender como o componente deve ser configurado. Isso torna o código mais difícil de ler, manter e modificar.

Minhas principais dificuldades na remoção do code smell são:

1. Manter a compatibilidade com código existente: O componente já estava sendo usado em vários lugares do projeto ng-zorro-antd. Não podíamos simplesmente mudar como as propriedades funcionam, pois isso quebraria o código que já utiliza o componente. Precisávamos garantir que o código antigo continuasse funcionando normalmente.

2. Agrupar propriedades de forma lógica: Tinha que identificar quais propriedades fazem sentido ficarem juntas. Por exemplo, propriedades relacionadas ao "estado" da paginação

(como total, pageIndex, pageSize) fazem sentido em um grupo, enquanto propriedades relacionadas à "visualização" ou "aparência" (como size, disabled) fazem sentido em outro grupo.

3. **Manter toda a funcionalidade existente:** Precisávamos garantir que todas as funcionalidades continuassem funcionando exatamente como antes, apenas reorganizando a estrutura interna do componente. Qualquer mudança que quebrasse a funcionalidade existente seria um problema.

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

Método 1: Agrupamento de Propriedades Relacionadas
Criamos interfaces TypeScript para agrupar propriedades que têm relação entre si:

- PaginationOptionsState: Agrupa propriedades relacionadas ao estado da paginação (total, pageIndex, pageSize, pageSizeOptions).

- PaginationOptionsDisplay: Agrupa propriedades relacionadas à aparência/visualização (size, disabled, showSizeChanger, showQuickJumper)

Método 2: Criação de Getters para Compatibilidade
Para manter o código existente funcionando, criamos "getters" (propriedades de leitura) que acessam os valores dentro dos objetos agrupados. Assim, qualquer código que use as propriedades antigas (como component.size ou component.total) continua funcionando normalmente, mas internamente os valores vêm dos objetos agrupados.

Método 3: Separação de Tipos em Arquivos Próprios
Seguindo o princípio de separação de responsabilidades, separamos os tipos (interfaces) em um arquivo próprio (pagination.types.ts), deixando o código mais organizado e fácil de encontrar. Isso também facilita a reutilização desses tipos em outros lugares, se necessário.