

QUALIDADE DE SOFTWARE

Refatorando code smell em projetos React: Ant Design

Dupla: Daniel Ulisses, Kleyton Ferreira

O Projeto

O **Ant Design** é uma biblioteca de interface (UI) voltada para a criação de aplicações web modernas, especialmente em ambientes corporativos. Seu foco está em oferecer uma experiência visual consistente, elegante e altamente funcional, atendendo às necessidades de aplicações de médio e grande porte.

Construído com:

- TypeScript
- React
- Baseado em CSS in JS

O Projeto

The image displays a collection of user interface (UI) design components and examples from the AFFiNE library. The components include:

- Buttons:** Primary Button, Default Button, Danger Button.
- Form Elements:** Option A dropdown, Date input (2022-11-18), Switches, Radio buttons.
- Modals:** Basic Modal (with content about using Modals for user feedback), Popover Title (with content about floating cards).
- Alerts:** Popconfirm (with Cancel and OK buttons), Prompt Text (with message "This is a normal message").
- Progress Indicators:** Progress bars (blue, green, red) showing 50%, 100%, and 50% completion respectively, along with circular progress indicators (68%, success, failed).
- Table Headers:** Ant Design Title 1, 2, 3, 4, each with a title and subtitle "Ant Design, a design language for background applications".
- Step Progress:** Step1, Step2, Step3, Step1, Step2, Step3, Step1, Step2.
- Calendars:** A weekly calendar showing dates from last week to 6 months later, with specific dates like 19th highlighted.
- Tab Navigation:** Tab1, Tab2, Tab3, with Tab1 active.
- Dropdown Menus:** A dropdown menu with items like Previous and Next.
- Help and Info:** Help info (with a list of items: Create a services site 2015-09-01, Create a services site 2015-09-01, Create a services site 2015-09-01, Create a services site 2015-09-01), Success, Error, and a list of 5 items.
- File Management:** Icons for file operations like Open, Save, Copy, Paste, Delete, and Undo/Redo.

A detecção de smells

A análise estática identificou 2.909 violações de qualidade. O code smell mais frequente é o uso indiscriminado do tipo Non-Null Assertions.

Foram identificadas as seguintes ocorrências:

- 1.493 do tipo Non-Null Assertions (NNA).
- 711 do tipo ANY.
- 261 do tipo Large Component
- 233 do tipo Too Many Props
- 83 do tipo Direct DOM Manipulation.
- 52 do tipo Uncontrolled Components
- 45 do tipo Multiple Booleans for State (MBS).
- 21 do tipo Missing Union Type (MUT).
- 09 do tipo Overly Flexible Props (OFP).
- 01 Force Update

Refatoração - Any Type

O code smell Any Type ocorre quando se utiliza o tipo any para desativar propositalmente a verificação de tipos do compilador. Isso anula a segurança e os benefícios do TypeScript, mascarando a estrutura real dos dados e permitindo que erros de contrato passem despercebidos até o tempo de execução

Antes

```
43 type Color = Extract<GetProp<ColorPickerProps, 'value'>, string | { cleared: any }>;
```

Depois

```
43 type Color = Extract<GetProp<ColorPickerProps, 'value'>, string | { cleared: boolean }>;
```

As refatorações: Non-Null Assertion - NNA

O code smell Non-Null Assertion ocorre quando se utiliza o operador ! (conhecido como bang operator) para forçar o compilador a ignorar a possibilidade de um valor ser nulo ou indefinido.

Antes

```
110  return {
111    label: isLink ? (
112      <Link to={getLocalizedPathname(linkPath!, isZhCN(pathname), search)}>
113        <FormattedMessage id={id} />
114      </Link>
115    ) : (
116      <FormattedMessage id={id} />
117    ),
118    icon,
119    key: key || i,
120    extra: showBadge ? (showBadge() ? badge : null) : extra,
121  };
122};
```

Depois

```
110  return {
111    label:
112    isLink && linkPath ? (
113      <Link to={getLocalizedPathname(linkPath, isZhCN(pathname), search)}>
114        <FormattedMessage id={id} />
115      </Link>
116    ) : (
117      <FormattedMessage id={id} />
118    ),
119    icon,
120    key: key || i,
121    extra: showBadge ? (showBadge() ? badge : null) : extra,
122  };
123};
```

As refatorações: Multiple Booleans for State - MBS

Multiple Booleans for State é um code smell onde se utilizam várias variáveis booleanas independentes para gerenciar estados que deveriam ser mutuamente exclusivos.

Antes

```
const [isLineClampSupport, setIsLineClampSupport] = React.useState(false);
const [isTextOverflowSupport, setIsTextOverflowSupport] = React.useState(false);

const [isJsEllipsis, setIsJsEllipsis] = React.useState(false);
const [isNativeEllipsis, setIsNativeEllipsis] = React.useState(false);
const [isNativeVisible, setIsNativeVisible] = React.useState(true);
```

Depois

```
type EllipsisState = {
  lineClampSupport: boolean;
  textOverflowSupport: boolean;
  jsEllipsis: boolean;
  nativeEllipsis: boolean;
  nativeVisible: boolean;
};

const [ellipsisState, setEllipsisState] = React.useState<EllipsisState>({
  lineClampSupport: false,
  textOverflowSupport: false,
  jsEllipsis: false,
  nativeEllipsis: false,
  nativeVisible: true,
});
```

As refatorações: Direct DOM Manipulation

O code smell de Direct DOM Manipulation ocorre quando o código ignora o ciclo de vida do React e acessa ou modifica diretamente o DOM do navegador (usando métodos como `document.getElementById`, `appendChild`, etc.). Isso quebra o princípio declarativo do React, pode causar vazamentos de memória (memory leaks) e torna os testes menos confiáveis, pois o React perde o controle sobre aqueles elementos.

Antes

```
const existScript = document.getElementById(scriptId) as HTMLScriptElement;
```

Depois

```
const scriptRef = useRef<HTMLScriptElement | null>(null);
if (scriptRef.current?.dataset.loaded) {
  openHituCodeBlockFn();
  return;
}
```

Principais dificuldades

Durante o processo de refatoração foram identificados as seguintes dificuldades:

- Contexto do projeto;
 - Para refatorar alguns code smells, era preciso entender um pouco sobre o que o trecho do código indicava, para que a refatoração pudesse ocorrer de uma forma que não afetasse outras partes do código. Por conta disso, era necessário um estudo para entender o contexto do projeto.
- Entender o que o TypeScript dizia
 - Em alguns casos, os desenvolvedores “usaram” os code smells como uma forma de silenciar o que o TypeScript dizia (como o uso do any), então para resolver esses smells, era necessário ver o que a linguagem sinalizava, entender e corrigir o problema.
- Dependências de bibliotecas
 - Alguns casos de smells desse projeto se manifestavam em decorrência de dependência de bibliotecas externas que não possuíam tipagem completa, por exemplo. Nesses casos, foi necessário buscar entender o como as bibliotecas funcionavam para criar tipos compatíveis.

Aprendizados

Para conseguir refatorar os smells foi necessário buscar soluções que iam além dos conhecimentos que já tínhamos:

- Uso do Partial<T>
 - Transforma as propriedades de um tipo em opcionais
- Record<K, V>
 - Tipo utilitário do TypeScript que usa chave e valor

QUALIDADE DE SOFTWARE

OBRIGADO!



**UNIVERSIDADE
FEDERAL DO CEARÁ**
CAMPUS QUIXADÁ