EQUIPE :

JÚLIO MORAIS
DOUGLAS EDUARDO

QUALIDADE DE SOFTWARE

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

CODE SMELLS EM FRAMEWORKS FRONT-END

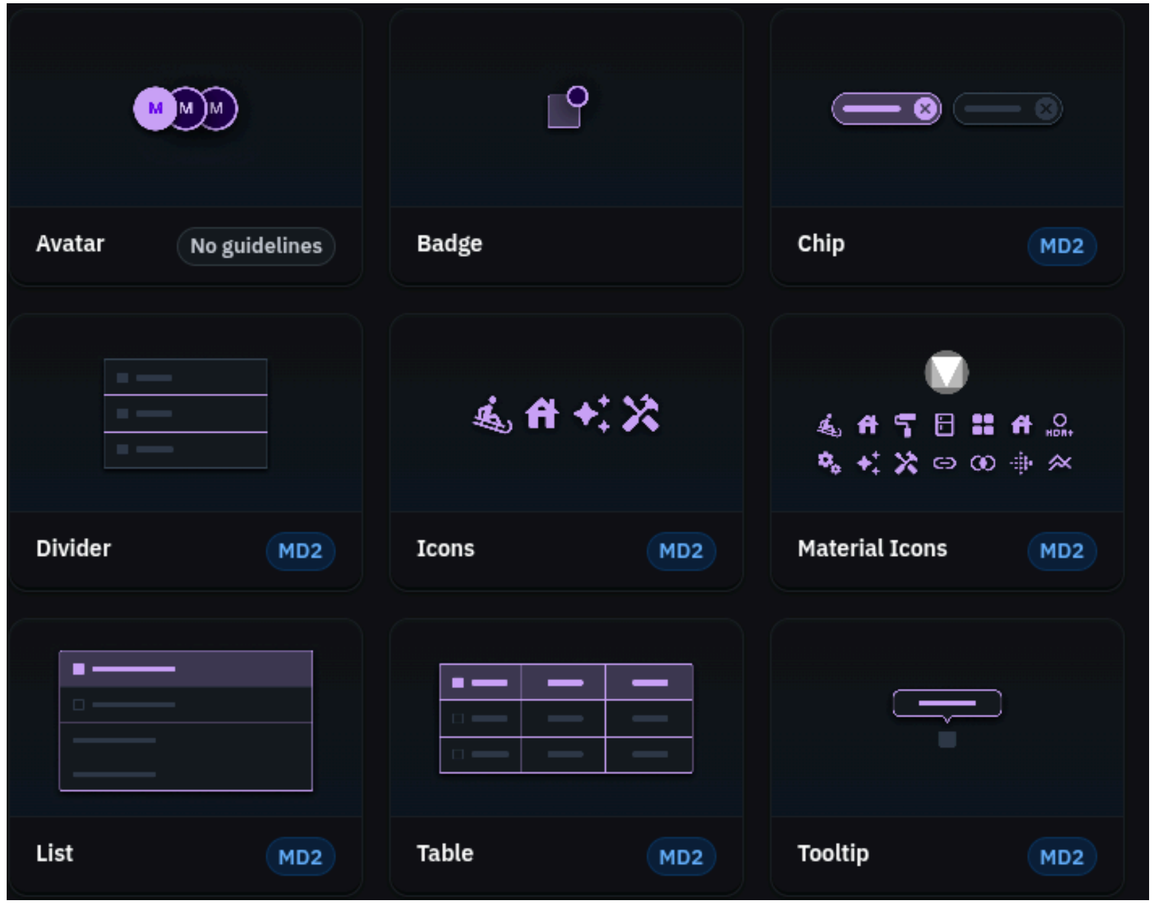
MUI

MATERIAL UI (MUI)

Material UI é uma biblioteca de componentes React de código aberto que implementa o Material Design do Google.

MATERIAL UI (MUI)

Material UI



ARQUIVOS COM SMELLS

(index)	file	MUT	MBS	ANY	EIV	NNA	OFP
0	'AlertVariousStates.tsx'	'N'	'N'	1	0	0	0
1	'GitHubLabel.tsx'	'N'	'N'	1	0	0	0
2	'Virtualize.tsx'	'N'	'N'	1	0	0	0

-
-
-

272	'describeSkipIf.tsx'	'N'	'N'	1	0	0	0
273	'components.spec.tsx'	'N'	'N'	1	0	1	0
274	'theme-scoping.test.tsx'	'N'	'N'	1	0	0	0

QUANTIDADE DETECTADA POR TIPO

MUT	MBS	ANY	EIV	NNA	OFP
6	2	219	0	116	12

SMELLS RESOLVIDOS

MUT	ANY	NNA	OFP
1. **DesignKits.tsx**	1. **i18n.tsx** 2. **Badge.tsx** 3. **Breadcrumbs.tsx** 4. **MenuItem.tsx** 5. **NoSsr.tsx** 6. **Portal.tsx** 7. **SliderValueLabel.tsx** 8. **useSlot.test.tsx** 9. **useSlotProps.test.tsx** 10. **theme-scoping.test.tsx**	1. **FormattedInputs.tsx** 2. **DesignKits.tsx** 3. **CodeCopy.tsx** 4. **MarkdownElement.tsx** 5. **AccordionDetails.tsx** 6. **Alert.tsx** 7. **AspectRatio.tsx** 8. **AutocompleteListbox.tsx** 9. **AutocompleteOption.tsx** 10. **Avatar.tsx**	1. **Copyright.tsx** 2. **dashboard-template-theme.tsx** 3. **LicenseModelContext.tsx** 4. **PrioritySupportContext.tsx** 5. **SearchButton.tsx** 6. **Markdown.tsx** 7. **HighlightedCodeWithTabs.tsx** 8. **Autocomplete.test.tsx** 9. **FocusTrap.test.tsx** 10. **ThemeProviderWithVars.tsx**
	1. **AlertVariousStates.tsx** 2. **GitHubLabel.tsx** 3. **InputFormProps.tsx** 4. **InputReactImask.tsx** 5. **SelectCustomValueAppearance.tsx**	1. **LinearProgressCountUp.tsx** 2. **MenuListComposition.tsx** 3. **ServerModal.tsx**	1. **useSlotProps.test.tsx**

REFATORAÇÕES

ANY-TYPES

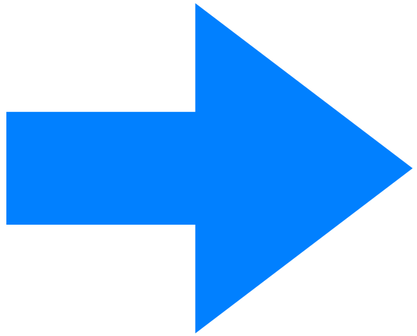
- Any é um tipo que desativa a verificação de tipos do TypeScript.
- Um valor do tipo any pode ser qualquer coisa e aceitar qualquer operação.

REFATORAÇÕES ANY

Type Assertion Específica (as TipoConcreto em vez de as any)

Você diz explicitamente ao TypeScript qual é o tipo real, em vez de desligar o sistema de tipos com any

```
const user = response as any;
user.nome.toUpperCase();
user.id.toFixed();
```



```
type User = {
  id: number;
  nome: string;
};

const user = response as User;
user.nome.toUpperCase(); // OK
user.id.toFixed();       // OK
```

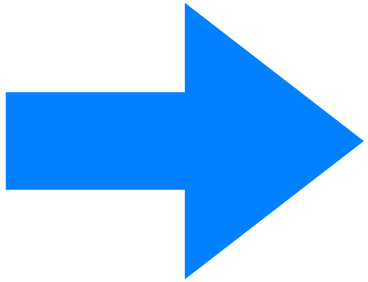
codigo exemplo

REFATORAÇÕES ANY

Union Types para valores com possibilidades conhecidas

Você limita os valores possíveis a um conjunto específico.

```
let status: any;
status = "ativo";
status = 99;
status = false;
```



```
type Status = "ativo" | "inativo" | "pendente";

let status: Status;
status = "ativo";   // ✅
status = "cancelado"; // ❌
```

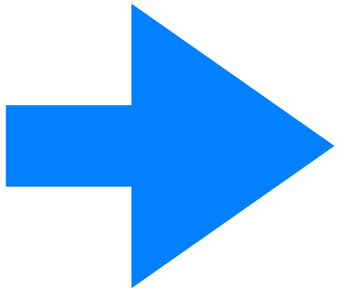
codigo exemplo

REFATORAÇÕES ANY

Interface/Type Definitions para estruturas complexas

Você descreve exatamente a estrutura de um objeto.

```ts
const product: any = {
  id: 1,
  price: 99.9
};
```



```ts
interface Product {
  id: number;
  name: string;
  price: number;
}

const product: Product = {
  id: 1,
  name: "Mouse",
  price: 99.9
};
```

OU

```ts
type Product = {
  id: number;
  name: string;
  price: number;
};
```

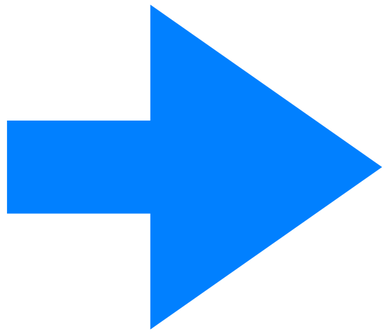
codigo exemplo

REFATORAÇÕES ANY

Satisfies Operator para validação em tempo de desenvolvimento

Garante que um objeto satisfaça um tipo, sem perder os tipos literais internos.

```
const config: Config = {
  mode: "dark",
  retries: 3
};
```



```
type Config = {
  mode: "dark" | "light";
  retries: number;
};

const config = {
  mode: "dark",
  retries: 3
} satisfies Config;
```

codigo exemplo

REFATORAÇÕES

NOT NULL ASSERTIONS

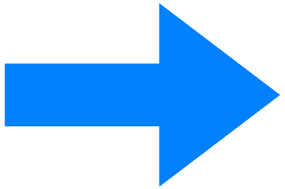
- Non-Null Assertion é o uso do operador !.
- Ele informa ao TypeScript que um valor não é null nem undefined.
- O compilador confia no desenvolvedor e não realiza verificações.

REFATORAÇÕES NNA

Optional Chaining (?.)

Só acessa propriedades se o valor existir

```
user!.profile!.avatar!.url;
```



```
const avatarUrl = user?.profile?.avatar?.url;
```

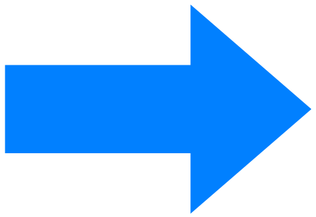
codigo exemplo

REFATORAÇÕES NNA

Verificações Condicionais

Garante a existência antes de usar

```
function printEmail(user?: User) {
  console.log(user!.email);
}
```



```
function printEmail(user?: User) {
  if (!user) return;
  console.log(user.email);
}
```

OU

```
if (!user) {
  throw new Error("Usuário não encontrado");
}

console.log(user.email);
```

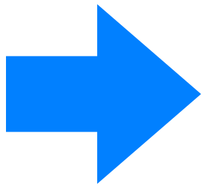
codigo exemplo

REFATORAÇÕES NNA

Nullish Coalescing (??)

Definir um valor padrão pra caso não exista

```
const nameLength = user!.name!.length;
```



```
const nameLength = (user?.name ?? "").length;
```

OUTRO USO

```
const retryCount = config.retries ?? 3;
```

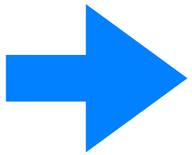
codigo exemplo

REFATORAÇÕES NNA

Combinação: ?. + ??

Checka se existe, mas também define um valor padrão caso não exista

```
const url = user!.profile!.avatar!.url;
```



```
const url = user?.profile?.avatar?.url ?? "/default-avatar.png";
```

codigo exemplo

REFATORAÇÕES

OFP

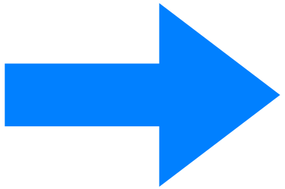
- OFP acontece quando um parâmetro é muito genérico, normalmente usando:
  - any
  - object
- Isso reduz a segurança de tipos e dificulta o entendimento do código.

REFATORAÇÕES OFP

Tipagem explícita de parâmetros

Substituir parâmetros genéricos por tipos bem definidos.

```
function handleData(data: any) { }
```



```
interface UserData {
  id: number;
  name: string;
}

function handleData(data: UserData) { }
```

codigo exemplo

REFATORAÇÕES

MUT

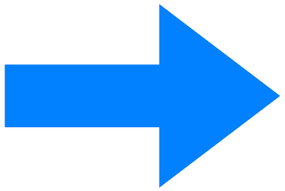
- MUT acontece quando um tipo complexo (união de strings literais ou genéricos) é definido diretamente inline no código e repetido em múltiplos locais.

REFATORAÇÕES MUT

Extração de Tipo para Centralizar Uniões (Extract Type)

Tipos de união literais ou genéricos são definidos inline e repetidos, tornando a manutenção difícil e propensa a erros.

```
prop: 'figma' | 'sketch';
```



```
type ToolBrand = 'figma' | 'sketch';

prop: ToolBrand;
```

codigo exemplo

PERGUNTAS?
COMENTÁRIOS?