UNIVERSIDADE FEDERAL DO CEARÁ

Code smells em frameworks front-end

Disciplina: Qualidade de Software
Equipe: Lais Queiroz e Evynne Avelino
Projeto: Payload
Fork:https://github.com/EvynneAv/payload

Eu estou atualmente trabalhando na refatoração do seguinte code smell: Any Type no arquivo: examples/auth/src/app/(app)/_components/Input/index.tsx, no caso havia dois smells no arquivo
Encontrar uma tipagem adequada para substituir os Any
Eu estou usando os seguintes métodos de refatoração para remover o code smell: Substituir o tipo Any por um tipo mais específico, tornando o código mais seguro.

Eu estou atualmente trabalhando na refatoração do seguinte code smell: Arquivo:\payload\examples\auth\src\app\(app)_components\RichText\index.tsx Tipo: Any Type Linha(s): 6-6 Arquivo :payload\examples\localization\src\blocks\Content\Component.tsx Tipo: Any type Linha(s): 35-35 Arquivo : payload\examples\localization\src\blocks\Form\Checkbox\index.tsx Tipo: Any type Linha(s):Lines 17-17 Arquivo : payload\examples\localization\src\blocks\Form\Checkbox\index.tsx Tipo: Any type Linha(s):Lines 20-20 Arquivo :payload\examples\localization\src\blocks\Form\Checkbox\index.tsx Tipo: Any type Linha(s):Lines 22-22
Minhas principais dificuldades na remoção do code smell são: Não saber a estrutura exata de “errors” e piorar a situação ao invés de corrigir o smell

UNIVERSIDADE FEDERAL DO CEARÁ

Eu estou usando os seguintes métodos de refatoração para remover o code smell: trocar o any por um tipo já pronto do react-hook-form.

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo: payload\examples\localization\src\components\LivePreviewListener\index.tsx
Tipo de smell: Non Null Assertions
Lines 9-9

Minhas principais dificuldades na remoção do code smell são:
às vezes esse componente precisa do serverURL, tem que se decidir o comportamento quando a env não existe.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
A forma mais fácil de tirar isso é não assumir que existe. Se trata o caso em que a env não veio.
Antes:

examples > localization > src > components > LivePreviewListener > index.tsx > ...

```tsx
'use client'
import { RefreshRouteOnSave as PayloadLivePreview } from '@payloadcms/live-preview-react'
import { useRouter } from 'next/navigation'
import React from 'react'

export const LivePreviewListener: React.FC = () => {
  const router = useRouter()
  return (
    <PayloadLivePreview refresh={router.refresh} serverURL={process.env.NEXT_PUBLIC_SERVER_URL!} />
  )
}

```

Depois:

examples > localization > src > components > LivePreviewListener > index.tsx > ...

```tsx
'use client'

import { RefreshRouteOnSave as PayloadLivePreview } from '@payloadcms/live-preview-react'
import { useRouter } from 'next/navigation'
import React from 'react'

export const LivePreviewListener: React.FC = () => {
  const router = useRouter()

  const serverURL = process.env.NEXT_PUBLIC_SERVER_URL

  if (!serverURL) {
    // pode retornar null (não renderiza) ou mostrar um aviso
    return null
  }

  return <PayloadLivePreview refresh={router.refresh} serverURL={serverURL} />
}

```

UNIVERSIDADE
FEDERAL DO CEARÁ

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s):payload\examples\localization\src\components\Media\ImageMedia\index.tsx
Tipo de smell:(2) Non Null Assertions
Linha(s): 44-44 45-45

Minhas principais dificuldades na remoção do code smell são: Quando tento tirar o !, eu percebo que fullWidth e fullHeight podem não existir em alguns casos. Isso me obriga a decidir o que fazer quando esses valores vêm undefined, e existe o risco de quebrar o layout se eu simplesmente assumir que eles sempre estão lá.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
Colocando uma verificação antes de usar os valores. Só atribuo width e height se eles realmente existirem, em vez de forçar com !. Assim eu elimino o Non-Null Assertion e deixo o código mais seguro.
Antes:

```
33    let src: StaticImageData | string = srcFromProps || ''
34
35    if (!src && resource && typeof resource === 'object') {
36      const {
37        alt: altFromResource,
38        filename: fullFilename,
39        height: fullHeight,
40        url,
41        width: fullWidth,
42      } = resource
43
44      width = fullWidth!
45      height = fullHeight!
46      alt = altFromResource
47
48      src = `${process.env.NEXT_PUBLIC_SERVER_URL}${url}`
49    }
50
51    // NOTE: this is used by the browser to determine which image to download at different screen sizes
52    const sizes = sizeFromProps
53      ? sizeFromProps
54      : Object.entries(breakpoints)
55          .map(([, value]) => `(max-width: ${value}px) ${value}px`)
56          .join(', ')
57
```

Depois:

```
35    if (!src && resource && typeof resource === 'object') {
36      const {
37        alt: altFromResource,
38        filename: fullFilename,
39        height: fullHeight,
40        url,
41        width: fullWidth,
42      } = resource
43
44      if (typeof fullWidth === 'number') width = fullWidth
45      if (typeof fullHeight === 'number') height = fullHeight
46      alt = altFromResource
47
48      src = `${process.env.NEXT_PUBLIC_SERVER_URL}${url}`
49    }
50
51    // NOTE: this is used by the browser to determine which image to download at different screen sizes
52    const sizes = sizeFromProps
53      ? sizeFromProps
54      : Object.entries(breakpoints)
55          .map(([, value]) => `(max-width: ${value}px) ${value}px`)
56          .join(', ')
```

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s):payload\packages\plugin-ecommerce\src\react\provider\index.tsx
Tipo de smell:Non Null Assertions
Linha(s): 123-125

Minhas principais dificuldades na remoção do code smell são:Removi o operador ! para não assumir que o dado sempre existe. Agora, o código me obriga a tratar casos undefined

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
Substituí o operador ! por verificações explícitas para não assumir que o dado sempre existirá. Ao tratar os casos de undefined logo no início, garanto que a lógica só avance com dados válidos.
Antes:

UNIVERSIDADE FEDERAL DO CEARÁ

```
 * The secret for accessing guest carts without authentication.
 * This is generated when a guest user creates a cart.
 */
const [cartSecret, setCartSecret] = useState<string | undefined>(undefined)
const [cart, setCart] = useState<CartsCollection>()

const [selectedCurrency, setSelectedCurrency] = useState<Currency>(
  () =>
    currenciesConfig.supportedCurrencies.find(
      (c) => c.code === currenciesConfig.defaultCurrency,
    )!,
)

const [selectedPaymentMethod, setSelectedPaymentMethod] = useState<null | string>(null)

const cartQuery = useMemo(() => {
  const priceField = `priceIn${selectedCurrency.code}`

  const baseQuery = {
    depth: 0,
    populate: {
      products: {
```

Depois:

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s): payload\packages\plugin-import-export\src\components\SortBy\index.tsx
Tipo de smell:(3) Missing Union Type Abstraction
Linha(s): Lines 32-32
Lines 96-96
Lines 117-117

Minhas principais dificuldades na remoção do code smell são:
O TS não consegue prever com exatidão todos os formatos do valor. Se eu mantenho a tipagem genérica (como string), sacrifico a segurança do código; por outro lado, se eu restrinjo demais o tipo, corro o risco de causar quebras em partes do sistema que ainda operam com outros valores.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
Eu resolvo criando um tipo união (union type) com os valores possíveis (ex.: 'asc' | 'desc') e reutilizo esse tipo nos pontos do código. Quando chega algo “de fora” (ex.: query string), eu converto/normalizo antes de usar, em vez de ficar fazendo cast solto.
Antes linha 32:

```
25
26  // The "sort" text field that stores 'title' or '-title'
27  const { setValue: setSort, value: sortRaw } = useField<string>()
28
29  // Sibling order field ('asc' | 'desc') used when writing sort on change
30  const { value: sortOrder = 'asc' } = useField<string>({ path: 'sortOrder' })
31  // Needed so we can initialize sortOrder when SortOrder component is hidden
32  const { setValue: setSortOrder } = useField<'asc' | 'desc'>({ path: 'sortOrder' })
33
34  const { value: collectionSlug } = useField<string>({ path: 'collectionSlug' })
35  const { query } = useListQuery()
36  const { getEntityConfig } = useConfig()
37  const { collection } = useImportExport()
38
```

Depois:

```
27  const { setValue: setSort, value: sortRaw } = useField<string>()
28
29  // Sibling order field ('asc' | 'desc') used when writing sort on change
30  const { value: sortOrder = 'asc' } = useField<string>({ path: 'sortOrder' })
31  // Needed so we can initialize sortOrder when SortOrder component is hidden
32  type SortOrder = 'asc' | 'desc'
33
34  const { value: sortOrder = 'asc' } = useField<SortOrder>({ path: 'sortOrder' })
35
36
37  const { value: collectionSlug } = useField<string>({ path: 'collectionSlug' })
38  const { query } = useListQuery()
39  const { getEntityConfig } = useConfig()
40  const { collection } = useImportExport()
```

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s): payload\packages\ui\src\forms\Form\index.tsx
Tipo de smell:(6) Multiple Booleans For State
Linha(s):
Lines 108-108
Lines 110-110
Lines 116-116
Lines 119-119
Lines 125-125
Lines 150-150

Minhas principais dificuldades na remoção do code smell são:

Eu estou usando os seguintes métodos de refatoração para remover o code smell:Eu tento transformar booleanos em um estado mais significativo, juntando numa estrutura única. Aqui, em vez de um ref booleano, eu guardo um “evento”/status

Antes:

```
106
107       const [disabled, setDisabled] = useState(disabledFromProps || false)
108       const [isMounted, setIsMounted] = useState(false)
109
110       const [submitted, setSubmitted] = useState(false)
111
112       /**
113        * Tracks wether the form state passes validation.
114        * For example the state could be submitted but invalid as field errors have been returned.
115        */
116       const [isValid, setIsValid] = useState(true)
117       const [initializing, setInitializing] = useState(initializingFromProps)
118
119       const [processing, setProcessing] = useState(false)
120
121       /**
122        * Determines whether the form is processing asynchronously in the background, e.g. autosave is running.
123        * Useful to determine whether to disable the form or queue other processes while in flight, e.g. disable manual submits while an autosave is running.
124        */
125       const [backgroundProcessing, _setBackgroundProcessing] = useState(false)
126
127       /**
128        * A ref that can be read within the `setModified` interceptor.
129        * Dependents of this state can read it immediately without needing to wait for a render cycle.
130        */
131       const backgroundProcessingRef = useRef(backgroundProcessing)
132
133       /**
134        * Flag to track if the form was modified _during a submission_, e.g. while autosave is running.
135        * Useful in order to avoid resetting `modified` to false wrongfully after a submit.
136        * For example, if the user modifies a field while the a background process (autosave) is running,
137        * we need to ensure that after the submit completes, the `modified` state remains true.
138        */
139       const modifiedWhileProcessingRef = useRef(false)
140
141       /**
142        * Intercept the `setBackgroundProcessing` method to keep the ref in sync.
143        * See the `backgroundProcessingRef` for more details.
144        */
145       const setBackgroundProcessing = useCallback((backgroundProcessing: boolean) => {
146         backgroundProcessingRef.current = backgroundProcessing
147         _setBackgroundProcessing(backgroundProcessing)
148       }, [])
149
```

Depois:

UNIVERSIDADE
FEDERAL DO CEARÁ

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s):payload\packages\plugin-seo\src\index.tsx
Tipo de smell:(4)
Linha(s): Missing Union Type Abstraction

Minhas principais dificuldades na remoção do code smell são: Eu fiz e refiz o código várias vezes mas mesmo assim não passava no sniff

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
Antes: Criando uma constante com a lista e derivando o tipo dela, então o TS gera o union automaticamente

```
128            }
129          }
130
131          return collection
132        }) || [],
133      endpoints: [
134        ...(config.endpoints ?? []),
135        {
136          handler: async (req) => {
137            const data: Omit<
138              Parameters<GenerateTitle>[0],
139              'collectionConfig' | 'globalConfig' | 'req'
140            > = await req.json?.()
141
142            const reqData = data ?? req.data
143
144            const result = pluginConfig.generateTitle
145              ? await pluginConfig.generateTitle({
146                  ...data,
```

139 Depois:

```
132        }) || [],
133      endpoints: [
134        ...(config.endpoints ?? []),
135        {
136          handler: async (req) => {
137            const generateTitleInjectedKeys = ['collectionConfig', 'globalConfig', 'req'] as const
138  type GenerateTitleInjectedKey = (typeof generateTitleInjectedKeys)[number]
139
140
141            const data: Omit<Parameters<GenerateTitle>[0], GenerateTitleInjectedKey> =
142    await req.json?.()
143
144
145
146            const reqData = data ?? req.data
147
```

163 antes:

```
157            path: '/plugin-seo/generate-title',
158          },
159          {
160            handler: async (req) => {
161              const data: Omit<
162                Parameters<GenerateTitle>[0],
163                'collectionConfig' | 'globalConfig' | 'req'
164              > = await req.json?.()
165
166              const reqData = data ?? req.data
167
168              const result = pluginConfig.generateDescription
```

UNIVERSIDADE FEDERAL DO CEARÁ

```
            return new Response(JSON.stringify({ result }), { status: 200 })
          },
          method: 'post',
          path: '/plugin-seo/generate-title',
        },
        {
          handler: async (req) => {
            const generateTitleInjectedKeys = ['collectionConfig', 'globalConfig', 'req'] as const
            type GenerateTitleInjectedKey = (typeof generateTitleInjectedKeys)[number]

            const data: Omit<Parameters<GenerateTitle>[0], GenerateTitleInjectedKey> =
await req.json?.()


            const reqData = data ?? req.data

            const result = pluginConfig.generateDescription
```

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s):
C:\Users\rzgrz\Desktop\Quality\payload\examples\form-builder\src\components\Blocks\Form\Checkbox\index.tsx
Tipo de smell: Any Type
Linha(s): 15, 18, 19

Minhas principais dificuldades na remoção do code smell são: O {[x: string]: any} escondia a real estrutura esperada.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
Antes:
Props como interseção de tipos + objetos genéricos

```
  getValues: any
  register: UseFormRegister<any & FieldValues>
  setValue: any
} & CheckboxField
```

- Objeto genérico com any

```
FieldErrorsImpl<{
  [x: string]: any
}>
```

Depois:
Interface explícita com tipos específicos

```
interface CheckboxProps extends CheckboxField {
  errors: Partial<FieldErrorsImpl<FormData>>
  getValues: UseFormGetValues<FormData>
  register: UseFormRegister<FormData>
  setValue: UseFormSetValue<FormData>
}
```

- Tipo explícito com possibilidades conhecidas

```
type FormData = {
  [key: string]: boolean | string | number | null
}
```

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s): payload\packages\next\src\views\Login\LoginField\index.tsx
Tipo de smell: Any Type
Linha(s): 61

Minhas principais dificuldades na remoção do code smell são: Não estava claro o que cada parâmetro genérico controlava.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
Antes: Múltiplos 'any' e tipo parcial

```
const passesEmail = email(
  value,
  options as ValidateOptions<any, { username?: string }, any, any>,
)
```

Depois: encapsulamento de tipos e complexidade em funções tipadas.

```
validate={(value, options) => {
  const passesUsername = validateUsername(value, options)
  const passesEmail = validateEmail(value, options)

  if (!passesEmail && !passesUsername) {
      return t('validation:invalidEmailOrUsername', {
      emailError: passesEmail,
      usernameError: passesUsername
    })
  }
```

UNIVERSIDADE FEDERAL DO CEARÁ

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s): payload\packages\plugin-import-export\src\components\SortBy\index.tsx
Tipo de smell: Missing Union Type Abstraction
Linha(s): 32, 96, 117

Minhas principais dificuldades na remoção do code smell são: Escolher entre criar uma abstração no próprio arquivo ou utilizar um arquivo secundário que já estava sendo importado e que tinha códigos de funções auxiliares e preservar o comportamento para não quebrar o código.

Eu estou usando os seguintes métodos de refatoração para remover o code smell: Utilizei o método de extração de de tipos criando uma função de validação para depois substituir e remover o smell.
Antes:

```
const order: 'asc' | 'desc' = isDesc ? 'desc' : 'asc'

const next = applySortOrder(option.value, String(sortOrder) as 'asc' | 'desc')
setSort(next)

const { setValue: setSortOrder } = useField<'asc' | 'desc'>({ path: 'sortOrder' })
```

Depois:

```
const direction = toSortDirection(sortOrder, 'asc')
const next = applySortOrder(option.value, direction)
setSort(next)

const isDesc = isDescendingSort(qsSort)
const base = stripSortDash(source)
const order: SortDirection = isDesc ? 'desc' : 'asc'

  // Sibling order field ('asc' | 'desc') used when writing sort on change
  const { value: sortOrderRaw } = useField<SortDirection>({ path: 'sortOrder' })

  const sortOrder: SortDirection = toSortDirection(sortOrderRaw, 'asc')

  // Needed so we can initialize sortOrder when SortOrder component is hidden
  const { setValue: setSortOrder } = useField<SortDirection>({ path: 'sortOrder' })
```

```ts
export type SortDirection = 'asc' | 'desc'

export function isValidSortDirection(value: unknown): value is SortDirection {
  return value === 'asc' || value === 'desc'
}

export function toSortDirection(value: unknown, fallback: SortDirection = 'asc'): SortDirection {
  return isValidSortDirection(value) ? value : fallback
}

export function toggleSortDirection(current: SortDirection): SortDirection {
  return current === 'asc' ? 'desc' : 'asc'
}

export function toggleSortDirection(current: SortDirection): SortDirection {
  return current === 'asc' ? 'desc' : 'asc'
}
```

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s): payload\packages\ui\src\elements\BulkUpload\FormsManager\index.tsx
Tipo de smell: Multiple Booleans For State
Linha(s): 115 , 117-121, 141

Minhas principais dificuldades na remoção do code smell são: Garantir que a refatoração não mudasse o comportamento da aplicação, por se tratar de um fluxo assíncrono de inicialização que depende de múltiplos efeitos (useEffect, useCallback) e possui chamadas a funções remotas.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
Booleans que representavam fases do mesmo processo (hasInitializedState, hasInitializedDocPermissions, isInitializing) foram substituídos por um único estado explícito (initializationStatus).
Antes:

```ts
115     const [hasSubmitted, setHasSubmitted] = React.useState(false)
116     const [docPermissions, setDocPermissions] = React.useState<SanitizedDocumentPermissions>()
117     const [hasSavePermission, setHasSavePermission] = React.useState(false)
118     const [hasPublishPermission, setHasPublishPermission] = React.useState(false)
119     const [hasInitializedState, setHasInitializedState] = React.useState(false)
120     const [hasInitializedDocPermissions, setHasInitializedDocPermissions] = React.useState(false)
121     const [isInitializing, setIsInitializing] = React.useState(false)
```

```ts
141     const [isUploading, setIsUploading] = React.useState(false)
```

Depois:

```ts
type InitializationStatus = 'idle' | 'initializing' | 'ready'
```

```ts
118   const [initializationStatus, setInitializationStatus] = React.useState<InitializationStatus>('idle')
119   const isInitializing = initializationStatus === 'initializing'
```

UNIVERSIDADE FEDERAL DO CEARÁ

```
153   const initializeSharedDocPermissions = React.useCallback(async () => {
154     setInitializationStatus('initializing')
155
```

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s): payload\packages\ui\src\providers\ServerFunctions\index.tsx
Tipo de smell: Missing Union Type Abstraction e Any Type
Linha(s): 37, 43, 49, 55, 59

Minhas principais dificuldades na remoção do code smell são: Com relação a refatoração do code smell de MUT a minha principal dificuldade foi em analisar não somente a repetição semântica do código mas também a sintática já que a duplicação era tanto no formato das funções como também na utilização dos argumentos (Omit<..., 'clientConfig' | 'req'>). Já na refatoração do smell de Any Type a dificuldade foi tratar o data de modo que não comprometesse a função da aplicação já que não dá para saber exatamente ao certo qual tipo exato de dados que o argumento irá receber, então era necessário utilizar de um tipo que conseguisse ser genérico e também funcional e seguro para o Typescript.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:
O conceito de “argumentos expostos ao client” foi abstraído em um tipo único, eliminando duplicações.
Antes:

```
34  - type GetFormStateClient = (
35  -   args: {
36  -     signal?: AbortSignal
37  -   } & Omit<BuildFormStateArgs, 'clientConfig' | 'req'>,
38  - ) => ReturnType<typeof buildFormStateHandler>
```

Depois:

```
34  + type ClientExposedArgs<T> = Omit<T, 'clientConfig' | 'req'>
```

```
36  + type ServerFunctionClientWithSignal<Args, Result> = (
37  +   args: { signal?: AbortSignal } & ClientExposedArgs<Args>
38  + ) => Result
```

```
50  + type GetTableStateClient = ServerFunctionClientWithSignal<
51  +   BuildTableStateArgs,
52  +   ReturnType<typeof buildTableStateHandler>
```

Any Type:

UNIVERSIDADE
FEDERAL DO CEARÁ

```
58  60    export type RenderDocumentResult = {
59        -   data: any
    61    +   data: unknown
60  62      Document: React.ReactNode
61  63      preferences: DocumentPreferences
62  64    }
```

Eu estou atualmente trabalhando na refatoração do seguinte code smell:
Arquivo(s):
Tipo de smell:
Linha(s):

Minhas principais dificuldades na remoção do code smell são:

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

Depois: