como usar o Design Pattern Builder para lidar com Discriminated Unionsa no seu JSX

```tsx
export const Feed = ({ data, type }: DataFeedProps) => {
  return (
    <div>
      {type === PostFeedType.CAROUSEL &&
        <PostFeedSlider posts={data} />
      }
      {type === PostFeedType.IMAGE &&
        <PostFeedImage post={data} />
      }
      {type === PostFeedType.VIDEO &&
        <PostFeedVideo post={data} />
       }
    </div>
  )
}
```

Isaac Gomes

o que é Discriminated Union ?

```ts
type DataFeedProps =
  | {
      type: PostFeedType.CAROUSEL
      data: Array<ImageFeedProps | VideoFeedProps>
    }
  | {
      type: PostFeedType.VIDEO
      data: VideoFeedProps
    }
  | {
      type: PostFeedType.IMAGE
      data: ImageFeedProps
    }
```

note que para cada retorno temos um valor de enum/string associado que permite a identificar o tipo de retorno

Isaac Gomes

TS

o que é Discriminated Union ?

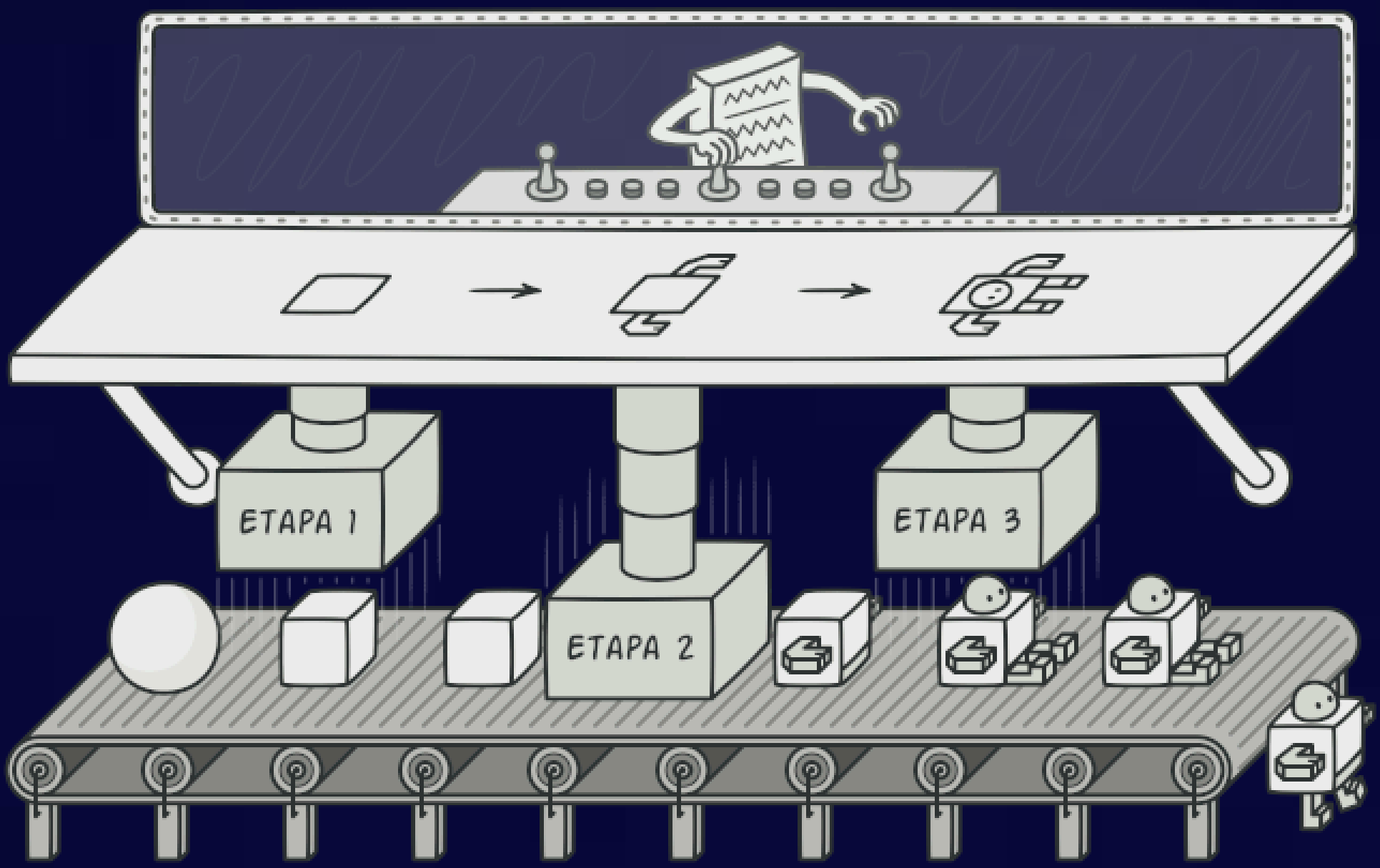
```ts
const Example = ({ data, type }: DataFeedProps) => {
  if (type === PostFeedType.CAROUSEL) {
    return data.map(item => {
      if (item.type === PostFeedType.IMAGE) {
        return PostFeedType.IMAGE
      }
      if (item.type === PostFeedType.VIDEO) {
        return PostFeedType.VIDEO
      }
    })
  }
}
```

quando realizamos a verificação temos os tipos de acordo com o valor associado

Isaac Gomes

TS

o que é Design Pattern Builder ?



O Design Pattern Builder é um padrão de design criacional que é usado para construir objetos complexos passo a passo

Isaac Gomes

o que é Design Pattern Builder ?

```
export class ValidationBuilder {
  private constructor(
    private readonly fielName: string,
    private readonly validations: FieldValidation[])
  {}

  static field(fieldName: string): ValidationBuilder {
    return new ValidationBuilder(fieldName, [])
  }

  required(): ValidationBuilder {
    this.validations.push(new RequiredFieldValidation(this.fielName))
    return this
  }

  email(): ValidationBuilder {
    this.validations.push(new EmailValidation(this.fielName))
    return this
  }

  min(lenght: number): ValidationBuilder {
    this.validations.push(new MinLenthValidation(this.fielName, lenght))
    return this
  }

  build(): FieldValidation[] {
    return this.validations
  }
}
```

Isaac Gomes

o que é Design Pattern Builder ?

```
ValidationBuilder
      .field('name')
      .required()
      .min(5)
      .build(),
ValidationBuilder
      .field('email')
      .required()
      .email()
      .build(),
```

veja que com um Builder temos a criação de objetos complexos de maneira flexível

Isaac Gomes

pq não trocar os ifs por um objeto literal?

```
const postFeedComponents:
    Record<PostFeedType, React.ReactElement> = {
    [PostFeedType.CAROUSEL]: <PostFeedSlider posts={data} />,
    [PostFeedType.IMAGE]: <PostFeedImage post={data} />,
    [PostFeedType.VIDEO]: <PostFeedVideo post={data} />,
}
```

porque não conseguimos fazer Discriminated Union através de objetos literais para fazer precisamos fazer verificações quer seja através de if ou case

Isaac Gomes

como podemos aplicar um builder nesse caso?

```
export const Feed = (post: DataFeedProps) => {
  return match(post)
    .with({ type: PostFeedType.CAROUSEL }, res => <PostFeedSlider posts={res.data} />)
    .with({ type: PostFeedType.IMAGE }, res => <PostFeedImage post={res.data} />)
    .with({ type: PostFeedType.VIDEO }, rest => <PostFeedVideo post={rest.data} />)
    .otherwise(() => <div>default</div>)
}
```

para criar condicionais com builder podemos usar a lib TS-Pattern ele permite a criação de condicionais com base no padrão builder

Isaac Gomes

mas sera que vale a pena instalar uma lib só para fazer isso?

```
function checkOrderStatus(order: Order): string {
  return match(order)
    .with({
        status: OrderStatus.PENDING,
        paymentType: PaymentType.CREDIT_CARD,
        totalAmount: (amount) => amount >= 100
    },
    () => {
  return 'Pedido pendente, pagamento com cartão de crédito e valor alto.';
})
.with({
    status: OrderStatus.PROCESSING,
    paymentType: PaymentType.PAYPAL,
    totalAmount: (amount) => amount >= 50 && amount < 100
    },
    () => {
  return 'Pedido em processamento, pagamento com PayPal e valor médio.';
})
.exhaustive()
}
```

como tudo DEPENDE... 90% dos casos NÃO

Isaac Gomes

TS

ta e como fica os 10%?

```tsx
const PostFeedSlider = ({ posts }: { posts: Array<ImageFeedProps | VideoFeedProps> }) => {
    return (
        <>
            {posts.map(post =>
                match(post)
                    .with({ type: PostFeedType.IMAGE }, res => <PostFeedImage post={res} />)
                    .with({ type: PostFeedType.VIDEO }, rest => <PostFeedVideo post={rest} />),
            )}
        </>
    )
}

export const Feed = (post: DataFeedProps) => {
    return match(post)
        .with({ type: PostFeedType.CAROUSEL }, res => <PostFeedSlider posts={res.data} />)
        .with({ type: PostFeedType.IMAGE }, res => <PostFeedImage post={res.data} />)
        .with({ type: PostFeedType.VIDEO }, rest => <PostFeedVideo post={rest.data} />)
        .otherwise(() => <div>default</div>)
}
```

casos onde faz sentido seu uso é em sistemas muito flexíveis como SaaS, pois, geralmente temos como no exemplo acima if dentro de if e pode ser um jeito de driblar isso

Isaac Gomes

vantagem do TS-pattern podemos usar o exhaustive para garantir que todos os casos foram cobertos

```ts
export const Feed = (post: DataFeedProps) => {
    return match(post)
        .with({ type: PostFeedType.CAROUSEL }, res => <PostFeedSlider posts={res.data} />)
        .with({ type: PostFeedType.IMAGE }, res => <PostFeedImage post={res.data} />)
        .exhaustive()
}
```

```ts
// Essa expressão não pode ser chamada.
// O tipo 'NonExhaustiveError
// <{ type: PostFeedType.VIDEO; data: VideoFeedProps; }>'
// não tem assinaturas de chamada.ts(2349)
// (property) exhaustive: NonExhaustiveError<{
//     type: PostFeedType.VIDEO;
//     data: VideoFeedProps;
// }>
```

Isaac Gomes

mas, 90% dos casos o switch faz o trabalho bem...

```
function renderPostComponent(post: DataFeedProps) {
    switch (post.type) {
      case PostFeedType.CAROUSEL:
        return <PostFeedSlider posts={post.data} />;
      case PostFeedType.IMAGE:
        return <PostFeedImage post={post.data} />;
      case PostFeedType.VIDEO:
        return <PostFeedVideo post={post.data} />;
      default:
        return <div />
    }
}
```

mas e ai o que achou do TS-pattern?

Isaac Gomes

Gostou?

Curta

Compartilhe

Salve

Isaac Gomes