

Manual para incluir casos de teste e módulos

Como entra um caso novo, um módulo novo e um sistema novo na ferramenta — e, em cada etapa, **quem faz o quê**.

Para usar a ferramenta, veja o [Manual do usuário](#).

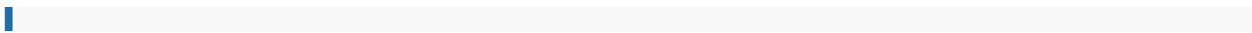
1. Primeiro: quem precisa de IA e quem não precisa

Um caso passa por três estágios. Só um deles exige quem saiba programar.

Estágio	O que é	Quem faz
1. Caso descrito	O que se vai testar, com qual critério de aceite	Você. Conhecimento de negócio, não de código
2. Caso catalogado	A ficha no catálogo: ID, módulo, criticidade, tags	Você, por um comando — sem IA ou solicite a uma IA
3. Caso automatizado	O arquivo <code>.spec.ts</code> que executa e confere	IA ou desenvolvedor. É código

Três consequências práticas:

- **Cadastrar não precisa de IA.** Um caso catalogado e ainda não automatizado já é útil: aparece no catálogo, entra nas contas de cobertura e sai no relatório como roteiro manual. Mas, pode utilizar uma de desejar.
- **Automatizar precisa.** O `.spec.ts` não é gravação de cliques: ele mede, confere o resultado na base e trata as armadilhas da tela. Hoje quem escreve é a IA, a partir da sua descrição.
- **Depois de escrito, roda sem IA para sempre** — e sem custo de token. O trabalho caro acontece uma vez por caso.



O que determina a qualidade do caso automatizado é o **estágio 1**. Um caso descrito como "verificar a tela" vira automação que só confere que a tela abriu. O critério de aceite é o que a IA transforma em medição.

2. Como descrever um caso que dá para automatizar

Quatro perguntas. Se as quatro tiverem resposta, o caso é automatizável.

1. **O que o usuário faz?** Caminho até a tela e os passos, na ordem.
2. **O que caracteriza sucesso?** O que precisa aparecer na tela e o que precisa ficar gravado.
3. **De que dado ele precisa?** Qual ligação, conta, carnê ou cadastro serve — e o que torna um registro elegível.
4. **Como se desfaz?** Se o caso grava algo que muda a operação — conta paga, lote, agrupamento — ele precisa reverter ao final.

Exemplo de descrição suficiente:

“Agrupar duas contas vencidas da mesma ligação. Sucesso: o agrupamento é criado, o valor do código de barras é igual à soma da coluna Total da revisão, e as duas contas aparecem como agrupadas na base. Precisa de ligação com duas contas vencidas, não retidas e não negociadas. Desfazer: desagrupar ao final.

3. Anatomia de um caso no catálogo

O catálogo é o arquivo `catalogo/testes.json`. Cada caso é uma ficha:

```
{  
  "id": "CT-SSB-CON-010",
```

```
"titulo": "Busca pelo painel: consulta por CDC",
"produto": "SSB",
"sistema": "balcao",
"modulo": "Consultas",
"funcionalidade": "Consultas - Busca pelo painel inicial",
"demanda": "REGRESSAO",
"tipo": "regressao",
"criticidade": "Alta",
"categoria": "principal",
"tags": ["busca", "painel", "cdc"],
"spec": "tests/biblioteca/ssb/consultas/CT-SSB-CON-010-busca-pelo-painel.spec.ts",
"automatizado": true,
"escopo": "produto",
"origem": "checklist-balcao-release / Crítico item 1.1",
"ativo": true
}
```

Campos obrigatórios

Campo	O que é
id	Identificador único — veja a convenção abaixo
titulo	O que o caso mede, numa frase
produto	SSB
sistema	balcao, baixa, hidrometria, usuarios
modulo	Módulo dentro do sistema (Consultas, Arrecadacao, ...)
funcionalidade	A tela ou fluxo específico
demanda	REGRESSAO ou o número do IC/RM que originou
tipo	regressao ou funcional
criticidade	Alta, Media ou Baixa
categoria	principal ou alternativo
tags	Palavras para filtrar depois
spec	Caminho do arquivo de automação (vazio enquanto não houver)
automatizado	true só quando o spec existe e roda
escopo	produto (vale para todos) ou cliente

Campo	O que é
<code>origem</code>	De onde o caso veio — checklist, demanda, reunião
<code>ativo</code>	<code>false</code> aposenta sem apagar o histórico

Campos opcionais que mudam o comportamento

Campo	Quando usar
<code>ambienteObrigatorio</code> + <code>motivoAmbienteObrigatorio</code>	O caso só roda num cliente. Ex.: penhora, que só existe em Poços. O motivo é obrigatório junto
<code>ambientePreferido</code>	Roda melhor num cliente, mas aceita outro
<code>aplicavelQuando</code>	Condição medida que torna o caso aplicável. Se não vale, o caso sai como não se aplica , com a condição medida, em vez de reprovar
<code>dependencias</code>	IDs de casos que precisam rodar antes
<code>severidadeSeFalhar</code>	Sobrepõe a criticidade na hora de classificar o defeito
<code>observacao</code>	Tudo que quem for automatizar precisa saber

Convenção de ID

CT-SSB-CON-010

|

|

|

└─ número sequencial dentro do módulo

|

|

└─ três letras do módulo (CON=Consultas, ARR=Arrecadação, BXA=Baixa...)

|

└─ produto

└─ caso de teste

Deixe espaço entre blocos (010, 011, 012... e o bloco seguinte em 020) — assim dá para inserir caso relacionado depois sem renumerar nada.

4. Incluir um caso novo

Não edite `catalogo/testes.json` na mão. O arquivo é grande e, quando vários casos são escritos ao mesmo tempo, abrir o arquivo inteiro e regravar faz um autor apagar o trabalho do outro sem erro nenhum. Use o comando, que grava sob trava e mescla só os campos informados:

```
npx tsx scripts/_catalogo-upsert.ts --id CT-SSB-CON-020 --campos '{"titulo":"Consulta por telefone com DDD e sem DDD","produto":"SSB","sistema":"balcao","modulo":"Consultas","funcionalidade":"Consultas - Por telefone","demanda":"REGRESSAO","tipo":"regressao","criticidade":"Media","categoria":"principal","tags":["busca","telefone"],"spec":"","automatizado":false,"escopo":"produto","origem":"checklist-balcao-release / item 1.9","ativo":true}'
```

O mesmo comando serve para **alterar** um caso: informe só o campo que muda.

```
npx tsx scripts/_catalogo-upsert.ts --id CT-SSB-CON-020 --campos '{"criticidade":"Alta"}'
```

Confira que entrou:

```
npm run catalogo -- --modulo Consultas
```

5. Declarar a massa de que o caso precisa

Se o caso depende de um registro com característica específica, declare isso em `catalogo/massa.json` como um **pedido** — assim a ferramenta encontra o registro sozinha, em qualquer cliente, em vez de depender de um CDC fixo que some.

Cada pedido tem nome, descrição, o **porquê** (por que esse recorte importa), as colunas e a consulta que o encontra:

```
{
  "nome": "ligacao-com-duas-contas-vencidas",
  "produto": "SSB",
  "descricao": "Ligação com ao menos duas contas vencidas, não retidas e não negociadas.",
  "porque": "Agrupar exige duas contas elegíveis; conta retida muda o resultado e viraria defeito falso.",
  "colunas": ["cdc", "conta1", "conta2"],
  "sql": "select ..."
}
```

Regra do projeto: **falta de massa não bloqueia caso**. Antes de desistir, a ferramenta procura o registro em outro cliente configurado e, quando o cenário permite, cria a massa pela própria interface.

6. Automatizar o caso

O arquivo de automação vive em:

```
tests/biblioteca/<produto>/<modulo>/CT-SSB-XXX-000-nome-curto.spec.ts
```

É o estágio que precisa de IA ou de um desenvolvedor. O que esse arquivo tem de fazer, além de clicar:

- 1. **Usar o page object do módulo** (`src/pages/ssb/<modulo>.page.ts`), não seletores soltos.
- 2. **Pedir a massa pelo nome do pedido**, nunca fixar um CDC no código.
- 3. **Conferir o resultado na base**, e não só na tela.
- 4. **Reverter** o que alterou a operação.
- 5. **Bloquear com a causa medida** quando não der para testar — jamais acusar o produto por falta de instrumento.

As armadilhas que já viraram regra

Cada uma dessas gerou, um dia, um defeito falso. Hoje são regra escrita no código:

Armadilha	O que acontece	O que o spec tem de fazer
Grid paginado	A tabela mostra 10 por página; ler só a visível "prova" que o registro não existe	Ampliar a página e conferir o indicador de truncado
Aviso efêmero	A faixa de mensagem dura ~4 segundos	Deixar a captura de avisos ligada antes da ação
Campo obrigatório	Botão nasce desabilitado; julgar aí acusa o produto	Preencher os obrigatórios e só então julgar o botão
Cadeia de modais	Até três modais em sequência; não responder um faz parecer que nada aconteceu	Responder a cadeia inteira
Sonda vazia	Inventário vazio parece "a tela não tem isso"	Se a leitura voltou vazia, bloquear dizendo isso

Depois de escrever, rode só o caso novo

```
npm run validar-versao -- --ambiente guaratingueta-qa --sistema balcao --test CT-SSB-CON-020
```

E só então marque como automatizado:

```
npx tsx scripts/_catalogo-upsert.ts --id CT-SSB-CON-020 --campos
'{"spec":"tests/biblioteca/ssb/consultas/CT-SSB-CON-020-consulta-por-
telefone.spec.ts","automatizado":true}'
```

7. Incluir um módulo novo

Módulo é um bloco de telas dentro de um sistema que já existe — por exemplo, Leituras dentro da Hidrometria. A ordem que funciona:

1. **Levantar as telas** do módulo no menu, em um cliente que o utilize.
2. **Escolher o prefixo de ID** (três letras livres) e reservar a faixa numérica.
3. **Descrever os casos** (seção 2) e **catalogá-los** sem automação (seção 4) — isso já dá cobertura visível e roteiro manual.
4. **Declarar os pedidos de massa** do módulo (seção 5). Massa de outro módulo quase nunca serve.
5. **Criar o page object** em `src/pages/ssb/<módulo>.page.ts`, com os caminhos e as leituras da tela.
6. **Automatizar caso a caso**, do mais crítico para o menos.
7. **Conferir em quais clientes o módulo é publicado** e configurar esses ambientes — um módulo que só existe em dois clientes precisa que esses dois estejam configurados, ou "não configurado" vira "funcionalidade não existe".

8. Incluir um sistema novo

Sistema é um produto inteiro — Balcão, Baixa, Hidrometria, Usuários. Para acrescentar outro:

1. **Descobrir a rota** em que ele é publicado (`/balcao_ssb`, `/baixa`, `/ssb_hidrometria`...) e confirmar que responde `Api/AppInfo`.
2. **Acrescentar o bloco em cada ambiente** que o publica, em `config/ambientes/<cliente>.json`, dentro de `sistemas`:

```
"nomedossistema": {
  "produto": "SSB",
  "nome": "Nome que aparece na interface",
  "rota": "/rota/index.html",
  "login": { "rota": "/rota/index.html", "seletorUsuario": "#username", "seletorSenha":
"#password", "seletorEntrar": "button[type='submit']", "seletorPosLogin": "ul.page-sidebar-
menu" },
```

```
"api": { "autenticacao": "storage", "base": "/rota", "chaveStorage": "ls.authorizationData",  
"caminhoToken": "token" }  
}
```

3. **Reiniciar o servidor** — o sistema novo passa a aparecer no passo 1 da aba *Executar*.
4. Seguir, daí em diante, os passos de módulo novo (seção 7).

9. Regras do projeto, que valem para todo caso novo

1. **Nunca bloquear por massa, parâmetro, ambiente ou permissão** sem antes tentar: criar a massa, procurar o cliente que usa a funcionalidade, trocar de base ou conceder a permissão.
2. **Medir antes de julgar.** Nada de concluir pelo que "parece" na tela — nem ausência, nem sucesso.
3. **Caso bloqueado diz de quem depende destravar:** ambiente, base ou ferramenta.
4. **Reverter o que altera a operação.** Cadastro de teste inativo pode ficar; conta paga, lote e agrupamento, não.
5. **Envio para fora permanece bloqueado**, mesmo em base de homologação.
6. **Um caso, um arquivo, um ID.** Casos irmãos podem dividir o mesmo spec, desde que cada um tenha seu ID e seu resultado próprio.

Revisão #1

Criado 2026-10-20 14:17:21 UTC por Stephanie Correa

Atualizado: 2026-10-20 14:18:58 UTC por Stephanie Correa