SPECSFY

Guia completo do usuário

Specify. Prove. Ship.

Da primeira ideia à entrega comprovada, em uma jornada prática e rastreável.

Edição v0.25.0
Gerado em 2 de novembro de 2026
Guia completo e autocontido

Sumário

Guia completo do usuário 11

- Percurso pedagógico 11
- Conversa contínua entre etapas 14
- A ideia central em um exemplo 14

Como a metodologia funciona 15

- Uma única especificação 15
- Effort e conversa contínua 16
- Interfaces fazem parte da definição 16
- Da ideia até o código 17
- Os três atos 17
- Como BDD e TDD trabalham juntos 19
- O agente conduz as transições 20
- Mudanças durante o trabalho 20
- Informações que permanecem entre entregas 20
- O que você encontra ao final 21
- Limites do método 21
- Justificativa de tamanho 21

Referência do método 22

- A spec como registro único 22
- Leitura do sistema existente 22
- Effort 23
- Estados e transições 24
- Gates 26
- Anatomia completa da spec 26
- Identificadores e rastreabilidade 29
- Tarefas, pesquisa e progresso 30
- Dúvidas frequentes 31
- Justificativa de tamanho 31

Instalação do Specsfy 32

- Instale o CLI 32
- Prepare o projeto consumidor 32
- Instalação no Windows e recuperação após falha 33
- Confira os arquivos instalados 34

Atualize sem perder customizações 34
Corrija falhas comuns 35
Primeiro projeto com o Specsfy 36
O que você vai construir 36
Escolha o diretório do projeto 36
Capture uma entrada 36
Refine a proposta no backlog 37
Crie a especificação única 37
Comprove a definição 38
Organize as tarefas 38
Prove que o teste detecta a ausência da página 39
Implemente e valide 39
Incorpore uma mudança posterior 40
Consulte o estado final 40
Converse sobre a próxima escolha — `$specsfy-interviewer` 40
Continue na mesma conversa 41
Justificativa de tamanho 41
Manutenção deste guia 41
Inbox 42
Capturar 42
O que o arquivo organiza 42
Inbox, backlog e spec 43
Quando a captura sai da Inbox 43
Sessões de descoberta do MVP 44
Templates instalados 44
Refinar ideias no backlog 45
Estrutura 45
Organização do backlog 45
Anatomia de um item 46
Padrões recorrentes 47
Ordenar o backlog 48
Fluxo 49
Estados do backlog 49
Quando está refinado 49
Limites 50

Próximo passo 50
Organize o produto por milestones 51
Comece pelo MVP 51
Arquivos do projeto 51
Vincule specs e backlog 51
Atualize o mapa automaticamente 52
Cinco usos do comando 52
Planeje o que vem depois 52
Skills base 54
Nomes exibidos 54
Como responder às perguntas 54
Encontre a skill pelo estado do trabalho 55
Capturar uma entrada com `specsfy-01-inbox` 57
Quando usar 57
Como descrever a tarefa 57
Exemplo passo a passo 57
O que esperar 57
Erros comuns 58
Próximo passo 58
Refinar uma entrada com `specsfy-02-backlog` 59
Quando usar 59
Como descrever a tarefa 59
Exemplo passo a passo 59
Como o ciclo termina 60
O que esperar 60
Erros comuns 61
Próximo passo 61
Criar a especificação com `specsfy-03-specify` 62
Quando usar 62
Como descrever a tarefa 62
Exemplo passo a passo 62
O que esperar 63
Erros comuns 63
Próximo passo 63
Validar a definição com `specsfy-04-validate` 64

Quando usar 64
Como descrever a tarefa 64
Exemplo passo a passo 64
O que esperar 65
Erros comuns 65
Próximo passo 65
Planejar tarefas com `specsfy-05-tasks` 66
Quando usar 66
Como descrever a tarefa 66
Exemplo passo a passo 66
O que esperar 67
Erros comuns 67
Próximo passo 68
Preparar testes com `specsfy-06-tdd-bdd` 69
Quando usar 69
Como descrever a tarefa 69
Exemplo passo a passo 69
O que esperar 70
Erros comuns 70
Próximo passo 70
Entregar código com `specsfy-07-implement` 71
Quando usar 71
Como descrever a tarefa 71
Exemplo passo a passo 71
Runner próprio de migrations 72
O que esperar 73
Erros comuns 73
Próximo passo 73
Incorporar mudanças com `specsfy-update-spec` 74
Quando usar 74
Como descrever a tarefa 74
Exemplo passo a passo 74
O que esperar 75
Erros comuns 75
Próximo passo 75

Consultar o estado com `specsfy-progress` 76
Quando usar 76
Como descrever a tarefa 76
Exemplo passo a passo 76
O que esperar 76
O que esperar 77
Erros comuns 77
Próximo passo 77
Conversar com uma spec 78
Quando usar 78
Como descrever a tarefa 78
Exemplo passo a passo 78
O que esperar 78
Erros comuns 79
Próximo passo 79
Definir o MVP por milestones 80
Quando usar 80
Como descrever a tarefa 80
Exemplo passo a passo 80
O que esperar 81
Erros comuns 81
Próximo passo 81
Descobrir o que o sistema precisa guardar com `specsfy-data-discovery` 82
Quando usar 82
Como descrever a tarefa 82
Exemplo passo a passo 82
O que esperar 82
Erros comuns 83
Próximo passo 83
Planejar milestones pós-MVP 84
Quando usar 84
Como descrever a tarefa 84
Exemplo passo a passo 84
O que esperar 84
Erros comuns 84

Próximo passo 84
Manter o mapa de milestones 85
Quando usar 85
Como descrever a tarefa 85
Exemplo passo a passo 85
O que esperar 85
Erros comuns 85
Próximo passo 85
CLI e TUI do Specsfy 86
Instalar e gerenciar skills 86
Atualização automática 87
Dashboard e progresso 87
Ciclo de vida de specs 88
Testes do projeto 92
Justificativa de tamanho 94
Segurança e reversibilidade 94
Referência dos comandos do Specsfy CLI 96
`specsfy` 96
`specsfy install` 96
`specsfy doctor` 97
`specsfy update` 97
`specsfy upgrade` 97
`specsfy skills list` 98
`specsfy skills detect` 98
`specsfy skills install` 98
`specsfy skills remove` 99
`specsfy skills update` 99
`specsfy transition` 99
`specsfy migrate` 100
`specsfy effort` 100
`specsfy inbox process` 100
`specsfy progress` 101
`specsfy milestones sync` 102
`specsfy test` 102
`specsfy tui` 103

`specsfy config show` 103
`specsfy config set` 103
Justificativa de tamanho 104
Confirmação e diagnóstico 104
Informações permanentes do projeto 105
Projeto existente com GitHub Spec Kit 107
Monitoramento durante mudanças 107
Preservação e atualização 108
Design system de interface 109
Onde fica 109
Como funciona 109
Defaults para CRUD 109
Defaults para dashboards e blocos 110
Cenários a cobrir 110
Revisão visual durante o desenvolvimento 111
Quando não usar 111
Conferência 111
Documentação técnica do sistema 112
O que esta documentação explica 112
O que é gerado e o que é preservado 112
Como ler cada documento 113
Procedimento depois de uma mudança 113
Situações que impedem a conclusão 114
Atualizar uma especificação 116
Descreva o que mudou 116
O que acontece 116
Resultado esperado 117
Limites 117
Quando usar outra skill 117
Skills especialistas 118
Detectar e instalar 118
Instalação no setup 118
Instalação manual 119
Gestor de pacotes Laravel 120
Contrato de interfaces React 120

Catálogo por domínio 121
Guia de cada especialista 121
Relação com as bases 123
Como funciona o Deploy da aplicação 124
O arquivo SEMVER 124
Uma identidade, vários pontos de conferência 125
Como as skills trabalham juntas 126
Sequência de uma entrega 129
Migrations durante o rollout 130
Rollback faz parte da preparação 131
Exemplo completo 131
Antes de considerar o deploy concluído 131
Justificativa de tamanho 132
Servidores, conexões e comandos de deploy 133
Onde a automação fica no seu projeto 133
O que muda quando você pede outro deploy 134
Como os servidores entram no inventário 134
A conta do servidor e a conta do container 135
Teste das conexões 135
Sincronização das chaves públicas 136
Comandos curtos para o Herdr 136
Senha do Vault e deploy pela IA 138
Configurar a senha fora do projeto 138
Deploy manual e deploy pela IA 139
Cadastro de segredos com a mesma configuração 141
Atualizar scripts que já existem 142
Uso avançado do Specsfy 143
Detecte e instale orientação técnica 143
Automatize a leitura de progresso 143
Ajuste a atualização visual 144
Atualize sem perder customizações 144
Incorpore uma mudança na spec 144
Limites 145
Usar Specsfy com Laravel 146
Confirmar a detecção 146

Instalação 146
Aplicar na spec 147
O que o especialista acrescenta 148
Resultado esperado 148
Limites 148
Usar Specsfy com Astro 150
Confirmar a detecção 150
Instalação 150
Aplicar na spec 150
O que o especialista acrescenta 151
Resultado esperado 151
Limites 151
Usar Specsfy com Next.js 152
Confirmar a detecção 152
Instalação 152
Aplicar na spec 152
O que o especialista acrescenta 153
Resultado esperado 153
Limites 153
Créditos do Specsfy 154
Projeto e manutenção 154
Comunidade 154
Identidade 154
Componentes relacionados 154
Inspirações e fontes 154
Como contribuir 155

Guia completo do usuário



O Specsfy ajuda você a transformar uma ideia em software testado sem espalhar requisitos, planos e tarefas por vários arquivos. Você conversa normalmente com o agente, e as skills organizam o trabalho em uma única especificação. Nesse arquivo, você consegue conferir o que será entregue, quais testes comprovam o comportamento e o que já foi concluído.

Este guia começa pela lógica da metodologia, prepara o ambiente e acompanha uma entrega completa. Os capítulos seguintes explicam a rotina com o CLI, as mudanças posteriores e os recursos avançados. Você não precisa conhecer a implementação do framework para seguir esse percurso.

Percurso pedagógico

Siga a ordem abaixo na primeira leitura. Quando já conhecer o método, use os links para voltar diretamente à tarefa que precisa executar.

1. Entenda a metodologia

Comece pela Metodologia. Cada entrega mantém o problema, os requisitos, os exemplos de comportamento, o plano técnico, as tarefas, os testes e as evidências em uma única `spec.md`. O capítulo mostra como esse arquivo muda ao longo do trabalho e o que comprova a passagem entre os atos.

Os três atos ligam cada fase a uma evidência verificável na `spec.md`:

1. **Ato I — Definir:** entender e validar o que deve ser entregue.
2. **Ato II — Projetar e provar:** preparar tarefas e obter o RED, a falha esperada antes da implementação.
3. **Ato III — Entregar e validar:** implementar, obter testes verdes e registrar evidências.

Para interpretar cada campo da spec, consulte a Referência do método. Ela detalha Effort, estados, transições, gates, IDs, pesquisa, tarefas e progresso.

2. Instale o Specsfy

Com o método entendido, siga a Instalação para instalar o CLI e preparar seu repositório. Ao final, `specsfy skills list` mostra as skills disponíveis e `specsfy progress --project .` confirma que o CLI consegue ler o projeto, mesmo que ainda não exista uma spec.

3. Faça a primeira entrega

Use Primeiro projeto como tutorial guiado. Você começa com uma mudança pequena, acompanha a criação da `spec.md` e termina conferindo os testes e as evidências registradas, sem precisar decorar cada skill.

Para preservar uma ideia sem iniciar a especificação, escolha uma destas entradas:

- preserve um texto sem perguntas na Inbox.
- refine e priorize uma proposta no Backlog.
- organize o MVP e o roadmap com Milestones.

4. Aprofunde o fluxo base

O índice de Skills base apresenta o fluxo completo. Leia cada etapa nesta ordem:

1. Capturar uma entrada.
2. Refinar no backlog.
3. Criar a especificação.
4. Validar a definição.
5. Preparar as tarefas.
6. Preparar TDD e BDD.
7. Implementar.
8. Atualizar a especificação.
9. Consultar o progresso.
10. Conversar com a spec.
11. Entrevistar o MVP.
12. Descobrir informações a guardar.
13. Planejar o roadmap.
14. Governar milestones.

Essas páginas explicam quando usar cada skill, como descrever a tarefa em linguagem natural, o resultado esperado, os erros comuns e o próximo passo. Quando uma delas precisar perguntar, você recebe uma pergunta numerada por rodada. Ela traz três ou mais opções numeradas, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde o início da conversa. Cada área aceita no máximo oito perguntas, salvo se você pedir mais e informar quantas deseja

responder. Depois de avançar, você informa se quer encerrar definitivamente as perguntas daquela área, responder depois ou retomar agora. O encerramento é respeitado até você reabrir a área; o adiamento preserva os pontos para retomada.

No setup, o perfil persistente em `.specsfy/USER-PROFILE.md` registra o nível de conhecimento e as respostas já confirmadas. O agente consulta esse arquivo, a conversa e as fontes do projeto antes de perguntar novamente.

5. Opere o projeto no dia a dia

Depois da primeira entrega, escolha os guias ligados à sua rotina:

- CLI e TUI: interface visual e acompanhamento.
- Referência dos comandos: parâmetros, efeitos, saídas e exemplos do CLI.
- Informações permanentes do projeto: stack, regras, banco e convenções.
- Design system de interface: regras macro, padrões CRUD, estados e exceções com alcance.
- Documentação do sistema: documentação técnica derivada da aplicação.
- Mudanças posteriores: como incorporar um novo requisito à mesma especificação.

6. Avance quando precisar

Os próximos guias são opcionais. Consulte-os quando a entrega exigir uma integração ou tecnologia específica:

- Especialistas, para conhecimento técnico adicional.
- Como funciona o Deploy da aplicação, para ligar `SEMVER`, imagem, Ansible e Docker Swarm em um deploy verificável.
- Servidores e comandos de deploy, para cadastrar servidores, conferir conexões, sincronizar chaves públicas e usar os comandos curtos no Herdr.
- Senha do Vault e deploy pela IA, para configurar a senha fora do Git e alternar entre execução manual e automatizada.
- Uso avançado, para automação e integrações.
- aplicação em projetos Laravel, Astro ou Next.js.
- Quadro técnico, para conhecer os módulos do monorepo.
- Créditos, para autoria e identidade do projeto.

Se você pretende contribuir ou modificar o próprio framework, continue no guia técnico. Ele é um percurso separado do uso em projetos consumidores.

Conversa contínua entre etapas

Quando uma etapa depende de outra skill, o agente anuncia a transição, explica o que falta e retoma o trabalho na mesma conversa. Você acompanha a mudança de etapa sem repetir a instrução inicial nem escolher manualmente cada skill.

A ideia central em um exemplo

Imagine uma página de boas-vindas. Você pode preservar a ideia, refiná-la no backlog e promovê-la até chegar a:

```
specs/<estado>/0001-pagina-boas-vindas/spec.md
```

Em seguida, o agente valida a definição, organiza tarefas, prepara testes, implementa e registra evidências nesse mesmo arquivo. Se depois você solicitar um botão novo, a alteração retorna à mesma `spec.md`. O agente reabre somente os atos cujas provas perderam validade, sem criar `plan.md`, `tasks.md` ou outra fonte normativa.

Para começar esse percurso com orientação passo a passo, siga agora a Metodologia.

Como a metodologia funciona

O Specsfy ajuda você a transformar uma necessidade em uma entrega comprovada. Você explica o que precisa, e o agente organiza a definição, o plano, os testes e a implementação na mesma especificação.

A metodologia existe para responder, durante todo o trabalho, a três perguntas:

1. **O que será entregue?**
2. **O que comprova que o plano está pronto?**
3. **Qual evidência comprova que a entrega funciona?**

Você não precisa decorar comandos nem escolher cada skill. O agente identifica a etapa atual, anuncia as transições e mantém o trabalho na mesma conversa.

Antes de iniciar uma descoberta, o setup lê o sistema que já existe. Ele reúne instruções do projeto, manifests, configuração, código, rotas, dados, integrações, interfaces, testes e documentação. Essa leitura evita sugestões desconectadas da aplicação e registra o que precisa continuar igual antes de propor uma mudança.

Uma única especificação

Cada mudança escolhida possui uma única fonte normativa. O caminho permite que você reconheça a sequência e o assunto ao listar o diretório de specs:

```
specs/<estado>/<NNNN>-<slug>/spec.md
```

`NNNN` é o número da spec, como `0001`. O `slug` identifica o assunto, como `recuperar-senha`. Assim, o caminho `0001-recuperar-senha` pode ser localizado sem abrir o arquivo.

A pasta também mostra o estado operacional da entrega:

```
draft → defined → planned → in-progress → review → completed
```

O campo `Status` dentro da spec espelha essa pasta. Use `specsfy transition` para mover o pacote inteiro, e não mova arquivos manualmente. `completed/` mantém o histórico das entregas finalizadas.

Ao abrir a `spec.md`, você encontra o comportamento esperado, o plano e as provas que permitem conferir o estado atual:

- o problema e o resultado esperado.
- as pessoas afetadas e as regras que precisam ser respeitadas.
- histórias, requisitos e limites.

- exemplos de comportamento escritos em BDD.
- as escolhas registradas, as possíveis falhas e o plano técnico.
- tarefas, testes e evidências da execução.
- gates e estado atual da entrega.

O plano e as tarefas ficam dentro da própria spec. O Specsfy não cria `plan.md` ou `tasks.md`, então a explicação da entrega não se divide entre arquivos com estados diferentes.

Effort e conversa contínua

Cada spec inclui `Effort`, uma estimativa de 1 a 10 da capacidade de raciocínio e execução necessária. A pontuação não é prazo: 1–2 indica trabalho atômico, 3–6 mudança local, 7–8 integração ou migração e 9–10 uma entrega com alta incerteza ou revisão humana frequente.

O entrevistador do Specsfy conversa com você quando uma lacuna puder mudar a próxima etapa. Ele atualiza a justificativa de Effort conforme a definição, o plano e a execução ganham forma. A Inbox continua sem perguntas.

A Referência do método detalha a escala de Effort, os perfis exibidos pelo progresso, os estados e os gates apresentados neste guia.

Interfaces fazem parte da definição

Quando a entrega cria ou muda uma tela usada por pessoas, o Specsfy pergunta antes do código como a experiência deve funcionar. A conversa cobre as telas, o fluxo de informação, os menus e a navegação principal, os campos e validações do formulário, a composição e o formato de cada ação, como página, painel lateral, modal ou outra alternativa. As opções usam texto completo, e você sempre pode escolher `Escrever outra resposta`, `Gere outras opções` ou `Avançar`.

Antes das perguntas, o Specsfy analisa a stack e o sistema atual quando ele existe. Ele observa rotas, telas, componentes, conteúdo, permissões, estados e testes para preservar o que já funciona e sugerir uma continuação coerente. O agente não troca React, Tailwind, shadcn/ui ou outra tecnologia por suposição.

Um CRUD com interface não é considerado pronto apenas por ter banco, serviço ou API. A spec registra telas, menus, formulário, navegação, estados de carregamento, vazio, erro e sucesso, além do uso por teclado. O plano gera tarefas e testes para essa interface antes da implementação.

Essas tarefas aparecem em uma `Fase de interface` própria na seção 14 da spec. Cada tela tem uma tarefa com caminho, comportamento e teste de interação.

Da ideia até o código

Nem todo texto precisa virar uma entrega imediatamente. Você escolhe o destino de acordo com o quanto já definiu:

- Uma **ideia** preserva seu texto em `specs/inbox/` sem perguntas e sem autorizar implementação.
- O **backlog** guarda uma proposta que merece organização e refinamento, mas ainda não foi escolhida para entrega. Ela vive em `specs/backlog/`.
- A **spec** representa uma entrega escolhida. A partir dela, definição, plano, testes, código e evidências passam a seguir o mesmo contrato.

O percurso mais completo preserva a ideia, aprofunda as definições e só chega ao código depois do plano e do RED:

```
inbox → backlog → spec → plano e RED → código → validação
```

Você também pode começar com "implemente recuperação de senha". O agente só altera o código depois de verificar as definições ausentes e conduzir as etapas correspondentes.

Os três atos

Os atos são grupos de trabalho. Cada um termina com um **gate**, que é um ponto de controle baseado em evidência. Um **gate** aprovado não significa apenas "parece bom": significa que as condições daquela etapa foram verificadas.

Escolha o destino da ideia

Objetivo: preservar a ideia sem transformá-la imediatamente em spec.

Sua participação: você informa a ideia e escolhe quando vale refiná-la ou promovê-la. Se solicitar apenas a captura, o agente não inicia o refinamento.

Prova técnica: a captura recebe um arquivo próprio em `specs/inbox/`. Se você escolher o refinamento, ela segue para o backlog. A `spec.md` normativa só aparece depois de uma promoção explícita.

Essa separação mantém a caixa de entrada leve e impede que toda observação se transforme em código ou em uma especificação extensa.

Ato I — Definir o que precisa mudar

Objetivo: entender o problema e transformar a intenção em comportamento que possa ser conferido.

Sua participação: você responde apenas às dúvidas que realmente mudam a entrega. O agente reaproveita o que já foi informado e apresenta uma pergunta numerada por rodada. Ela inclui três ou mais opções, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a primeira rodada. O ciclo faz no máximo oito perguntas por área: cada conjunto de respostas atualiza a análise. O avanço mantém uma confirmação para você encerrar a área, responder depois ou retomar agora. O encerramento é respeitado até uma reabertura explícita. O adiamento mantém os pontos registrados e o Definition Gate pendente até serem resolvidos.

Prova técnica: a spec registra a finalidade, as pessoas afetadas, os requisitos, os limites e os cenários BDD. A validação procura contradições, definições em aberto e dúvidas que impedem o planejamento.

O Ato I termina quando a validação registra este gate na `spec.md`:

```
Definition Gate: Passed
```

Esse estado permite que o agente organize o plano a partir dos requisitos e cenários já conferidos. O código ainda não foi alterado, e o Plan Gate continua pendente.

Ato II — Planejar e preparar o RED

Objetivo: definir como a mudança será construída e demonstrar que os testes conseguem detectar a ausência do novo comportamento.

Sua participação: você confirma escolhas de produto ou estratégia que alteram o resultado. Os detalhes técnicos que o repositório consegue comprovar podem ser derivados do código, da stack e das regras do projeto.

Prova técnica: o agente registra as tarefas, os contratos, as informações afetadas, as possíveis falhas e a reversibilidade na spec. Depois, transforma os cenários BDD em testes TDD executáveis e observa o **RED**, uma falha causada pela funcionalidade que ainda não existe.

O Ato II termina quando as tarefas e os testes permitem registrar:

```
Plan Gate: Passed
```

O RED precisa falhar pelo motivo esperado. Erro de sintaxe, dependência ausente ou ambiente quebrado não prova que o teste protege o comportamento.

Ato III — Entregar e conferir o resultado

Objetivo: implementar cada tarefa e reunir evidências atuais de que o resultado atende à definição.

Sua participação: você acompanha novas escolhas ou mudanças de escopo. O agente registra cada comando executado e apresenta o resultado verificável, sem exigir que você conduza os ciclos de teste.

Prova técnica: cada tarefa de código parte de um teste que falha pelo motivo esperado, recebe a menor implementação capaz de deixá-lo verde e termina com refatoração protegida pela suíte:

```
RED → GREEN → REFACTOR
```

- **RED:** o teste falha pela razão esperada.
- **GREEN:** a menor implementação faz o teste passar.
- **REFACTOR:** o código é melhorado sem mudar o comportamento.

Depois, o agente executa o aceite e a regressão completa, verifica a ligação entre cada requisito e seu teste, atualiza os registros permanentes do projeto e reconstrói a documentação aplicável.

O Ato III termina quando o aceite, a regressão e a documentação permitem registrar:

```
Delivery Gate: Passed
Status: Reviewing
```

Depois do aceite final, a spec passa por `review/` para `completed/`, com `Status: Complete`. A entrega concluída possui código e evidência atual. Você pode abrir a spec e localizar os comandos, os resultados e os testes que comprovam esse estado.

Como BDD e TDD trabalham juntos

O BDD (desenvolvimento orientado por comportamento) descreve o resultado que produto e desenvolvimento precisam discutir. O TDD (desenvolvimento orientado por testes) transforma esse comportamento em uma prova executável no projeto.

O **BDD** descreve o comportamento em uma linguagem que produto, desenvolvimento e testes conseguem discutir. Por exemplo:

```
Cenário: cliente solicita recuperação de senha
  Dado que existe um cadastro para o e-mail informado
  Quando o cliente solicita a recuperação
  Então o sistema confirma a solicitação sem revelar informações privadas
```

Esse cenário mostra a regra e o resultado esperado, mas o texto Gherkin não é executado como uma suíte separada pelo Specsfy.

O **TDD** transforma o comportamento em testes executáveis na ferramenta já usada pelo projeto. Primeiro o teste falha pelo motivo correto. Depois, a implementação faz o teste passar. Em resumo:

- BDD ajuda a definir **o comportamento que importa**.
- TDD comprova **que o código apresenta esse comportamento**.

Essa ligação evita uma descrição clara sem prova automática e também impede que um teste técnico seja aceito sem representar a necessidade registrada.

O agente conduz as transições

As skills dividem responsabilidades, mas você não precisa operar o fluxo como uma lista de comandos. Quando uma etapa depende de outra, o agente:

1. informa de qual skill está saindo e para qual está indo.
2. explica a pendência que motivou a transição.
3. resolve a pendência na skill responsável.
4. retorna à etapa anterior quando necessário.

Por exemplo, se a implementação encontrar um teste ausente, o agente volta ao planejamento e à preparação TDD, obtém um RED válido e só então retoma o código. Ele não aprova um gate apenas para contornar a pendência.

Mudanças durante o trabalho

Se a necessidade mudar depois que a spec já existe, explique a alteração em linguagem normal. O novo requisito entra na mesma `spec.md`, sem criar uma segunda especificação.

O Specsfy reabre somente as provas que perderam validade:

- Uma mudança de comportamento reabre definição, plano e entrega.
- Uma mudança apenas na estratégia técnica reabre plano e entrega.
- Uma evidência desatualizada exige somente a repetição da validação correspondente.

Use `specsfy-update-spec` para incorporar a nova instrução. A skill informa quais gates perderam validade e retoma a primeira etapa afetada.

Informações que permanecem entre entregas

Além da spec de cada entrega, o projeto mantém informações que valem para o sistema inteiro:

- `PROJECT.md`: finalidade, capacidades e limites do projeto.
- `.specsfy/STACK.md`: tecnologias estruturais e suas evidências.
- `.specsfy/RULES.md`: regras confirmadas para o trabalho.
- `.specsfy/DATABASE.md`: visão da persistência e das relações.

O agente consulta esses arquivos antes de planejar e os revisa durante a implementação. Assim, uma nova entrega começa com a arquitetura, as convenções e o banco já documentados.

O que você encontra ao final

Uma mudança completa deixa na `spec.md` um caminho auditável entre requisito, teste, tarefa e evidência:

- a intenção e as escolhas estão na spec.
- cada requisito aponta para condições de aceite.
- os cenários BDD explicam o comportamento.
- os testes TDD demonstram o resultado no código.
- as tarefas registram execução e evidências.
- os gates mostram quais etapas foram realmente comprovadas.
- a documentação reflete o sistema implementado.

Para consultar esse estado sem alterar arquivos, use `specsfy-progress`.

Limites do método

O método não define requisitos importantes sem você, não transforma toda ideia em spec e não trata pesquisa como requisito aprovado. Também não aceita erro de ambiente como RED nem substitui os testes e as ferramentas do seu projeto.

Justificativa de tamanho

Este guia mantém os três atos, os gates e a relação entre BDD e TDD na mesma página para que você possa comparar o percurso completo sem alternar entre explicações parciais.

O guia de instalação prepara o CLI e o framework. Depois, o primeiro projeto aplica os três atos a uma página de boas-vindas e mostra os gates na `spec.md`.

Referência do método

Este capítulo explica os campos, estados e provas de uma entrega no Specsfy. Use-o junto do guia da metodologia quando quiser interpretar uma `spec.md`, a tela de progresso ou uma pergunta do agente com precisão.

A spec como registro único

Cada entrega escolhida possui uma única fonte normativa:

```
specs/<estado>/<NNNN>-<slug>/spec.md
```

`NNNN` mantém a sequência de criação e o `slug` identifica o assunto. A pasta mostra o estado operacional. O campo `Status` no topo da spec é o espelho desse estado. Use `specsfy transition` para alterar os dois juntos, pois mover a pasta manualmente deixa o registro inconsistente.

A spec não é um relatório preenchido no final. Ela recebe a definição no Ato I, o plano e o RED no Ato II, e as tarefas com seus resultados no Ato III. Código, testes, pesquisa e documentação derivada podem ficar em outros arquivos, mas a ligação entre eles permanece na spec.

Leitura do sistema existente

Antes de uma skill propor tecnologia, telas ou implementação, o setup percorre as fontes relevantes do projeto: instruções, manifests, configuração, aplicação, rotas, persistência, integrações, interface, testes e documentação. Ele usa essa leitura para preservar convenções, comportamento e fontes já existentes. Quando houver muitas fontes, informa o conjunto lido antes de avançar; uma sugestão não substitui a análise do código existente.

CAMPO OU SEÇÃO	O QUE REGISTRA	COMO USAR
`Status` e pasta	posição da entrega no fluxo	identificar a próxima atividade permitida
`Effort`	capacidade de raciocínio e execução necessária	calibrar acompanhamento, não prazo
Gates	prova de prontidão de cada ato	impedir avanço prematuro
Seções 1–7	problema, escopo, comportamento e requisitos	conferir o que será entregue
Seções 8–15	plano, testes, tarefas e relações	conferir como a entrega será construída

Seções 16–18	dependências, escolhas e conclusão	conferir limites e encerramento

Effort

`Effort` é uma escala inteira de 1 a 10 para a capacidade de raciocínio, investigação, coordenação e execução pedida pela entrega no estado atual. Não representa horas, dias, preço, quantidade de pessoas ou prazo prometido. Uma tarefa curta pode ter Effort alto quando envolve autorização delicada, migração difícil de reverter ou integração pouco conhecida. Uma tarefa longa pode ficar em faixa menor quando segue uma convenção já comprovada e tem testes diretos.

O valor inicial é uma hipótese de trabalho. Ele pode mudar depois de uma resposta confirmada, da leitura do repositório, de uma dependência descoberta ou de um resultado de teste. O CLI guarda valor anterior, data e justificativa no histórico da própria spec. Assim, a pessoa que retoma a entrega consegue ver o motivo de cada recalibração.

Faixas e perfis

EFFORT	PERFIL EXIBIDO	SITUAÇÃO TÍPICA	ACOMPANHAMENTO
1–2	`light`	ajuste atômico em padrão conhecido	escopo e teste focal
3–4	`standard`	mudança local com testes diretos	requisitos, arquivos e variações
5–6	`standard`	vários arquivos ou integração conhecida	dependências, testes e ordem
7–8	`high`	mudança transversal, migração ou integração externa	contratos, dados, reversão e documentação
9–10	`maximum`	arquitetura ou incerteza relevante	descoberta progressiva e revisão frequente

As faixas orientam o perfil exibido pelo progresso, mas não autorizam pular um gate. Uma entrega de Effort 1 ainda precisa de definição, plano compatível e prova de entrega. Uma entrega de Effort 9 pode ser dividida em entregas menores quando seus comportamentos forem independentes.

Quando atualizar

Atualize Effort quando um fato mudar a capacidade necessária: mais módulos, dados existentes, integração externa, autorização, compatibilidade ou revisão humana recorrente podem elevar a faixa. Escopo menor, padrão já comprovado ou dependência removida podem reduzi-la. Não altere o número para comunicar urgência, pressão comercial ou preferência por modelo.

Registre o fato que mudou a estimativa. Uma justificativa como “passou de 4 para 7 porque inclui migração de dados existentes e compatibilidade com uma API publicada” permite revisar o valor mais tarde. Depois da confirmação, use:

```
specsfy effort <id-da-spec> <1-10> --reason "<justificativa confirmada>"
```

O entrevistador pode conduzir essa atualização, mas não inventa justificativa nem transforma Effort em autorização para implementar.

Effort não é prioridade

Prioridade responde o que deve receber atenção antes. Effort responde quanta capacidade a entrega exige. Um ajuste urgente pode ter Effort 2. Um estudo que ficará para depois pode ter Effort 8. O backlog ordena itens por valor, urgência, dependências, exposição operacional, esforço e informações ausentes. A spec usa Effort para tornar a execução transparente.

Estados e transições

O ciclo canônico é:

```
draft → defined → planned → in-progress → review → completed
```

PASTA	STATUS	O QUE JÁ EXISTE	O QUE AINDA NÃO VALE
`draft`	`Draft`	intenção e definição em construção	tratar requisitos como aprovados
`defined`	`Defined`	Definition Gate aprovado	implementar ou declarar plano pronto
`planned`	`Planned`	plano, tarefas e RED compatíveis	editar código sem iniciar a execução
`in-progress`	`Implementing`	tarefas e verificações em execução	concluir com trabalho pendente
`review`	`Reviewing`	Delivery Gate aprovado	alterar sem retornar à etapa necessária
`completed`	`Complete`	aceite final e documentação atual	incluir nova solicitação por edição direta

O significado de cada estado

Em `draft`, o agente esclarece problema, resultado, atores, escopo, regras, casos-limite e comportamento observável. Uma dúvida que muda produto, dados, segurança, aceite ou plano impede o gate da definição. Pesquisa pode apoiar a conversa, mas precisa ter conclusão e

impacto registrados na spec.

Em `defined`, problema, limites, histórias, requisitos e cenários BDD permitem planejar sem adivinhar o que será aceito. Uma mudança de comportamento retorna a spec para `draft` e torna o gate da definição pendente novamente.

Em `planned`, há plano técnico, tarefas, dependências, contratos, plano de teste e RED válido. Um RED não vale quando falha por sintaxe, fixture incompleta, dependência ausente ou ambiente indisponível. Ele precisa apontar a ausência do comportamento pretendido.

Em `in-progress`, cada tarefa segue a ordem registrada ou uma alteração justificada. Uma tarefa de código percorre RED, GREEN e REFACTOR, registrando comando, resultado, IDs cobertos e arquivos envolvidos. Descoberta que muda o comportamento retorna ao primeiro ato cuja prova perdeu validade.

Em `review`, a entrega já passou pelo Delivery Gate e aguarda aceite final. Um retorno de aceite pode levar a `in-progress` para correção técnica ou a um ato anterior quando a definição também mudar. Em `completed`, o pacote preserva o histórico. Uma solicitação nova inicia ou atualiza uma entrega ativa, em vez de alterar diretamente a spec concluída.

Retornos permitidos

ORIGEM	DESTINOS ACEITOS	MOTIVO DE RETORNO
`draft`	`draft`, `defined`	a definição ainda está aberta
`defined`	`draft`, `defined`, `planned`	requisito, escopo ou aceite mudou
`planned`	`defined`, `planned`, `in-progress`	plano revelou problema na definição ou RED precisa ser refeito
`in-progress`	`planned`, `in-progress`, `review`	tarefa, teste ou plano perdeu validade
`review`	`in-progress`, `review`, `completed`	aceite final encontrou trabalho pendente
`completed`	`completed`	entrega fechada não recebe edição direta

Mudança apenas de estratégia técnica retorna a `planned`. Mudança de comportamento retorna a `draft`. Uma comprovação vencida, como regressão executada antes da última alteração, pede nova verificação. O método não repete etapas por ritual: ele evita aprovar um resultado novo com uma prova antiga.

Gates

Um gate fica `Pending` enquanto sua etapa não tem a comprovação exigida e fica `Passed` quando as condições foram verificadas e registradas. Ele não é uma estimativa de qualidade nem um botão de aprovação manual.

Definition Gate

O gate do Ato I exige problema e resultado observável, escopo incluído e fora de escopo, atores, regras, histórias, requisitos funcionais e não funcionais, três cenários BDD distintos para cada item principal e nenhuma lacuna P1 que impeça o planejamento. Quando a entrega tem interface para pessoas, ele exige também telas, fluxo de informação, menus e navegação principal, formulário, composição, estados e acessibilidade descritos na seção 10. Um CRUD sem telas e formulário mantém o gate pendente. Termo ambíguo, requisito sem forma de teste, história sem aceite ou conflito entre seções mantêm o gate pendente.

Plan Gate

O gate do Ato II exige plano técnico proporcional, contratos e dados aplicáveis, tarefas com IDs e dependências, estratégia TDD derivada do BDD, RED válido, plano de testes e ordem de execução. Uma lista de tarefas vaga, sem referências, ou um teste sem cenário correspondente não torna o plano pronto.

Delivery Gate

O gate do Ato III exige tarefas tratadas, GREEN para os testes derivados do comportamento, aceite e regressão no estado atual, rastreabilidade entre histórias, requisitos, cenários, testes e tarefas, registros dos comandos e documentação atualizada quando a mudança a alcança. Um teste verde isolado não fecha o gate, pois ele pode cobrir somente parte do comportamento.

Anatomia completa da spec

A tabela inicial da spec identifica o formato, o ID, o slug, o estado, Effort, o vínculo opcional com ClickUp, os três gates, a versão do contrato de comprovação e a data de atualização. Não use essa tabela para esconder uma mudança material: o detalhamento fica na seção que trata do assunto e a tabela apenas torna o estado geral legível.

Ato I: seções 1 a 7

SEÇÃO	FINALIDADE	O QUE PRECISA FICAR CLARO
1. Problema e resultado	separar a dor atual da mudança desejada	contexto observável, resultado esperado e métrica verificável

2. Research e esclarecimentos	registrar o que foi investigado e o que ainda está aberto	pergunta, fonte, conclusão, impacto, dúvidas respondidas e dúvidas abertas
3. Escopo e atores	delimitar quem participa e o que a entrega cobre	incluído, fora de escopo, objetivos e permissões dos atores
4. Princípios e restrições	preservar regras já confirmadas pelo projeto	regras de governança, arquitetura, qualidade ou compatibilidade
5. Histórias de usuário	explicar valor por ator	capacidade, valor, prioridade, requisito e teste independente
6. Cenários BDD de aceite	tornar o comportamento observável	condição inicial, ação, resultado e IDs cobertos
7. Requisitos	declarar obrigações do sistema	funções, qualidades mensuráveis, erros e casos-limite

A seção 1 não deve antecipar a solução. “Criar uma tela” descreve uma possível implementação, enquanto “permitir que uma pessoa recupere acesso sem revelar se um e-mail existe” descreve o resultado e seu limite. A métrica de sucesso precisa ter um alvo ou uma observação verificável, não uma impressão genérica.

A seção 2 diferencia pesquisa de escolha. Um R-001 pode registrar uma documentação de API consultada e concluir que determinado endpoint exige idempotência. A regra que a entrega adotará aparece depois na seção de restrições, requisitos, plano ou escolhas. Se a fonte externa foi realmente consultada, seu registro local em research/ preserva origem, versão ou data e o ponto usado na conclusão.

Na seção 3, “fora de escopo” protege tanto o projeto quanto a expectativa de quem acompanha a entrega. Por exemplo, uma entrega que permite solicitar troca de senha pode deixar explícito que não inclui autenticação social, gestão de perfis ou alteração do visual de todas as telas. Atores não são apenas pessoas: um serviço externo, um job ou um administrador pode ter objetivo, permissão e limite próprios.

Nas seções 5, 6 e 7, a mesma necessidade aparece em três níveis. A história explica para quem a capacidade gera valor. O requisito declara a obrigação. O cenário mostra um exemplo que pode ser aceito ou recusado. Essa diferença evita história vaga, requisito sem teste e cenário que não representa valor.

Ato II: seções 8 a 15

SEÇÃO	FINALIDADE	PERGUNTAS QUE A SEÇÃO RESPONDE
8. Plano técnico	tornar a implementação compreensível antes da execução	quais módulos, dados, contratos, arquivos e compatibilidades serão afetados?

9. Modelo de dados	explicar persistência e ciclo de vida da informação	quais entidades, estados, transições, retenção e migrações existem?
10. Interfaces e contratos	registrar superfícies de integração e a experiência para pessoas	quais telas, menus, formulários, fluxos, ações, APIs, eventos, entradas, saídas e falhas importam?
11. Estratégia TDD	derivar testes executáveis do BDD	qual caso falha primeiro, por qual motivo e como ficará verde?
12. Plano de testes e rastreabilidade	ligar requisito à comprovação	qual cenário, nível, arquivo ou comando cobre cada item?
13. Validações	registrar os gates e achados	qual comando foi executado, qual resultado produziu e o que falta?
14. Tarefas	dividir a entrega em ações verificáveis	o que fazer, em qual ordem, com quais referências e dependências?
15. Ordem de execução	expor dependências reais	qual é o caminho crítico, o paralelismo e o menor conjunto entregável?

O plano técnico é proporcional ao alcance. Uma alteração local pode registrar um componente, teste e arquivo. Uma migração precisa explicar compatibilidade, ordem, reversão e retenção. Quando uma categoria não se aplica, escreva “Não aplicável” com a razão, em vez de deixar uma lacuna que pareça esquecimento.

Quando o cabeçalho declara `Interface para pessoas: Sim`, a seção 10 também registra a stack e o sistema atual observados, a responsabilidade de cada tela, os menus, seus itens e destinos, como a pessoa avança e retorna no fluxo, os campos e validações dos formulários, o padrão de abertura de ações, a disposição dos elementos e os estados de interface. O Specsfy analisa rotas, telas, componentes, conteúdo, permissões e testes antes de perguntar as lacunas reais. Assim, painel lateral, modal, página ou outro padrão não vira uma escolha escondida do agente nem substitui o que já existe sem confirmação.

Uma spec de interface também possui `Fase de interface` na seção 14. Cada tela recebe tarefa própria com caminho, testes de navegação, formulário, validações, feedback e teclado. O validador não aceita a fase ausente ou com menos tarefas do que as telas registradas.

O modelo de dados descreve estados de domínio, não o estado da pasta da spec. Por exemplo, um pagamento pode transitar de pendente para confirmado ou cancelado, enquanto a spec que altera pagamentos pode estar em planned. Mantenha as duas máquinas de estado separadas para não confundir o ciclo da entrega com o ciclo da informação do produto.

A estratégia TDD e o plano de testes são complementares. A seção 11 explica a sequência RED, GREEN e REFACTOR para o caso. A seção 12 permite localizar a relação entre requisito, cenário BDD, nível de teste, arquivo e comando. O marcador SPECSFY: no teste deve usar os IDs da spec. O resultado é uma cadeia que uma pessoa consegue seguir nos dois sentidos.

A seção 13 registra o resultado de cada gate e os achados de revisão. Um achado especializado usa identificador, severidade, estado, referências e comprovação. Marcar um achado como aceito não o torna invisível: a spec precisa mostrar o motivo, o responsável e o limite aceito.

Nas tarefas, [P] indica possibilidade de execução paralela apenas quando as dependências realmente permitem. Depends: none não é enfeite: significa que a tarefa pode começar sem outra tarefa da spec. A ordem de execução consolida essa informação em caminho crítico, tarefas paralelas e estratégia de MVP.

Ato III: seções 16 a 18

SEÇÃO	FINALIDADE	O QUE REGISTRAR
16. Dependências, riscos e suposições	expor condições externas e hipóteses	dependência, consequência, mitigação e condição que ainda precisa de confirmação
17. Decisões	preservar escolhas confirmadas	escolha, motivo, referências atingidas e ato que precisa ser revisto se ela mudar
18. Definition of Done	fechar o contrato de entrega	critérios de conclusão, aceite, documentação e comprovações finais

Dependências são itens fora do controle imediato da tarefa, como uma API, aprovação, credencial fornecida por outra equipe ou migração prévia. Suposições são premissas ainda não confirmadas. Registre ambas de forma explícita para que um teste verde não esconda uma condição externa não atendida.

A seção 17 não substitui um ADR para uma escolha arquitetural transversal nem substitui as seções de requisito e plano. Ela conserva a escolha da entrega e mostra o que precisa ser reavaliado quando a escolha for revista. O histórico é mais útil quando registra alternativas descartadas e seu motivo, sem reescrever o passado como se a escolha atual sempre tivesse sido conhecida.

A Definition of Done é o fechamento da entrega concreta. Ela reúne gates, testes, aceite, regressão, rastreabilidade, documentação e pendências tratadas. Não use uma lista genérica copiada de outra spec. Cada item precisa corresponder ao comportamento, aos dados e às integrações daquela entrega.

Identificadores e rastreabilidade

PREFIXO	REPRESENTA	USO
`US-001`	história de usuário	valor para um ator

`FR-001`	requisito funcional	comportamento obrigatório
`NFR-001`	requisito não funcional	condição mensurável de qualidade ou operação
`AC-001`	cenário BDD de aceite	caminho principal, regra ou limite
`T001`	tarefa	ação que atende referências declaradas
`R-001`	pesquisa	pergunta, conclusão, fonte e impacto
`FIND-*`	achado de revisão	tipo, severidade, estado e referência

Cada história principal possui três cenários distintos. Em geral, um cobre o caminho principal, outro uma regra crítica e outro uma falha ou limite. O cenário declara os IDs cobertos. O teste TDD usa o marcador `SPECSFY:` com os mesmos IDs. Essa cadeia mostra requisito sem teste e teste sem comportamento definido.

Tarefas, pesquisa e progresso

Uma tarefa registra ID, tipo, história relacionada, ação, caminho, referências e dependências. O checklist abaixo dela deixa seis movimentos auditáveis:

MOVIMENTO	PERGUNTA RESPONDIDA
`PREP`	escopo, referências, dependências e baseline estão claros?
`EXECUTE`	qual entrega foi produzida no caminho declarado?
`VERIFY`	qual verificação focal confirmou o resultado?
`VISUAL`	a interface respeita bordas, espaçamentos, margens, padding e tipografia, ou por que a revisão não se aplica?
`EVIDENCE`	qual comando, resultado e IDs permitem conferir o trabalho?
`IMPROVE`	houve melhoria de processo ou há motivo registrado para não aplicá-la?

Pesquisa responde uma pergunta, mas não aprova sozinha uma regra de produto. Registre pergunta, conclusão, fonte, localizador e impacto. Fonte externa consultada requer registro local em `research/`, enquanto a conclusão normativa fica na spec, ligada ao escopo, requisito ou plano.

`specsfy progress` lê as specs sem alterá-las. Ele mostra estado, Effort, perfil, gates e contagens. Sua porcentagem é uma projeção, não uma aprovação: com checklists, ela usa itens concluídos sobre itens totais, e sem checklists, gates aprovados sobre gates totais. Use a porcentagem para

localizar trabalho pendente. Use gates, resultados de teste e leitura da spec para confirmar um avanço.

Dúvidas frequentes

DÚVIDA	RESPOSTA
Posso implementar com Definition Gate aprovado?	Não. Falta o plano e o RED do Ato II.
Testes verdes permitem mover para `completed`?	Não. Falta aceite final em `review` e documentação aplicável.
Effort 10 significa dez dias?	Não. É a maior faixa de capacidade necessária.
RED de ambiente quebrado vale?	Não. O RED precisa indicar ausência do comportamento.
Pesquisa é requisito aprovado?	Não. A conclusão precisa ser incorporada e confirmada na spec.
Porcentagem de progresso aprova a entrega?	Não. Ela apenas projeta itens ou gates concluídos.

Retorne ao guia da metodologia para a jornada ou à página da skill que corresponde ao estado atual da sua spec.

Justificativa de tamanho

Este capítulo reúne contratos que antes apareciam em páginas diferentes ou em menções breves: Effort, estados, transições, gates, seções da spec, rastreabilidade, tarefas, pesquisa e progresso. Mantê-los próximos permite comparar o significado de cada campo sem exigir leitura do código ou salto entre vários guias. As páginas das skills continuam explicando como executar cada etapa, enquanto esta referência preserva os conceitos compartilhados.

Instalação do Specsfy

Instale o CLI

O Specsfy requer Node.js 22.20 ou uma versão mais recente. O pacote oficial publicado no npm instala o comando no ambiente global do usuário:

```
npm install --global @promovaweb/specsfy
```

Execute `specsfy --version` para confirmar que o terminal localiza o comando e que o Node.js consegue abrir o aplicativo:

```
specsfy --version
```

Uma resposta com o número da versão confirma a instalação. Se o terminal mostrar `specsfy: command not found`, consulte o diretório global do npm e confirme se o diretório de executáveis está no `PATH`:

```
npm prefix --global
specsfy --version
```

O download em `get.specsfy.dev` continua disponível para instalações mantidas em `$HOME/.local/bin`. Esse executável inclui as dependências do CLI, mas também requer Node.js 22.20 ou superior:

```
mkdir -p "$HOME/.local/bin"
curl -fL get.specsfy.dev -o "$HOME/.local/bin/specsfy"
chmod +x "$HOME/.local/bin/specsfy"
```

Prepare o projeto consumidor

Abra a raiz do repositório que receberá a metodologia. Não execute a instalação dentro do monorepo oficial do Specsfy, porque o CLI reconhece essa raiz como ambiente de desenvolvimento e recusa a operação:

```
cd caminho/do/projeto
specsfy doctor --project .
specsfy install --project .
```

O diagnóstico confere Node.js 22.20 ou superior, Git, npm, o diretório do projeto e o `npx`. Toda materialização usa `npx skills add`, inclusive quando o CLI foi instalado pelo npm. `install` repete as verificações necessárias antes de escrever qualquer arquivo e reúne todas as correções na mesma mensagem.

Quando o projeto estiver em um Hub, use o subdiretório escolhido pela pessoa em `--project`, como `specsfy install --project apps/portal`. O setup confirma o mesmo caminho e mantém nele os contextos, as specs e o trabalho de código.

O instalador publica as etapas numeradas a partir de `.agents/skills/specsfy-01-inbox`, grava o contrato central em `.specsfy/Spec.md` e adiciona templates, exemplos e registros técnicos em `.specsfy/`, inclusive `.specsfy/templates/DESIGNSYSTEM.MD` para as regras macro de interface. Ele também insere blocos gerenciados em `AGENTS.md` e `CLAUDE.md`, preservando o conteúdo que já existe fora desses blocos.

Ao executar `$specsfy-setup`, o template também gera `DESIGNSYSTEM.MD` na raiz do projeto quando o arquivo ainda não existe. O setup preserva um arquivo local existente e deixa a aplicação pronta para registrar seus padrões de CRUD, dashboard e interface.

O mesmo setup gera `.specsfy/USER-PROFILE.md` quando esse arquivo ainda não existe. Na conversa, ele identifica o nível de conhecimento, consulta respostas já registradas e adapta a explicação das próximas perguntas. O arquivo local e as respostas confirmadas são preservados entre execuções.

Para personalizar um template sem impedir atualizações, copie-o para `.specsfy/templates/custom/` com o mesmo nome. Essa versão tem precedência e nunca é sobrescrita pelo instalador, inclusive com `--force`.

A instalação inclui as quatorze skills base, entre elas as quatro de conversa e milestones, além do setup, do documentador do sistema e das três skills auxiliares. Ela prepara os arquivos usados pelo agente, mas não cria uma spec de produto nem altera o código da aplicação.

Instalação no Windows e recuperação após falha

No Windows, o diagnóstico procura executáveis com as extensões do PATHEXT, como git.exe. A instalação chama o entrypoint JavaScript do npx pelo Node, para preservar caminhos com espaços sem passar os argumentos por um shell. Se o npm estiver incompleto, repare sua instalação antes de tentar novamente.

Quando o instalador delegado falha, o Specsfy tenta restaurar as regras, os templates, as skills e os locks gerenciados que existiam antes da operação. Se a restauração também falhar, a mensagem informa onde encontrar a cópia dos arquivos anteriores. Preserve essa pasta e o log antes de repetir o comando.

A correção foi exercitada com simulação de plataforma. A confirmação em Windows nativo depende do teste no computador que apresentou a falha.

Confira os arquivos instalados

Na mesma raiz, liste o catálogo e consulte o progresso. O primeiro comando deve mostrar as skills instaladas, e o segundo deve conseguir ler o diretório de specs:

```
specsfy skills list
specsfy progress --project .
```

O catálogo deve mostrar as skills do Specsfy. Em um projeto novo, o comando de progresso pode retornar zero specs. Esse resultado confirma que o CLI leu o repositório e ainda não encontrou arquivos em `specs/<estado>/<NNNN>-<slug>/spec.md`.

O comando sem subcomando abre a interface visual no diretório atual. O nome do projeto aparece no topo e as abas devem carregar mesmo quando ainda não houver spec:

```
specsfy
```

O dashboard deve carregar as abas do projeto mesmo quando as tabelas ainda estiverem vazias. Use `Ctrl+Q` para sair.

Atualize sem perder customizações

Para atualizar o próprio CLI instalado pelo npm:

```
specsfy upgrade
specsfy --version
```

Na instalação pelo arquivo de `get.specsfy.dev`, repita o download e a permissão de execução quando o npm não gerenciar o executável. Para atualizar as skills já instaladas, execute `specsfy update`. O CLI compara os fingerprints e preserva os arquivos customizados:

```
specsfy update --project .
```

`specsfy skills update --project .` continua aceito para automações anteriores.

O Specsfy registra fingerprints dos arquivos gerenciados. Uma atualização normal substitui versões intactas e preserva arquivos customizados. Se o CLI informar que encontrou alterações locais, revise a diferença. `--force` descarta a customização protegida no arquivo indicado.

Corrija falhas comuns

- **Comando ausente:** confira o resultado de `npm prefix --global` e o `PATH` usado pelo terminal.
- **Node.js incompatível:** execute `node --version`. O aplicativo requer Node.js 22.20 ou uma versão mais recente.
- **Permissão negada:** execute novamente `chmod +x "$HOME/.local/bin/specsfy"` quando usar o download. Em instalações pelo npm, configure um diretório global gravável pelo seu usuário.
- **Mensagem `npx não encontrado`:** instale ou repare o npm e disponibilize `npx` no `PATH`.
- **Arquivo gerenciado customizado:** preserve sua versão ou compare as mudanças oficiais e só então repita o comando com `--force`.

Com o ambiente conferido, siga o primeiro projeto para criar uma entrega pequena e observar a primeira `spec.md`. O guia do CLI e da TUI detalha os comandos de atualização, progresso, testes e configuração.

Primeiro projeto com o Specsfy

O que você vai construir

Este tutorial acompanha uma página de boas-vindas em um projeto Laravel que já usa Pest. A rota recebe um nome e mostra uma saudação. Quando o nome não for informado, a página usa `visitante`.

Você verá como uma ideia chega à implementação sem dividir a fonte normativa entre `plan.md`, `tasks.md` e outros arquivos. O exemplo mostra cada skill separadamente para facilitar a consulta, embora o agente consiga fazer as transições na mesma conversa.

O tutorial depende de três condições verificáveis. Confirme a instalação, abra o agente na raiz do projeto consumidor e rode a suíte Pest existente:

- o CLI e o framework foram instalados conforme o guia de instalação.
- o agente está aberto na raiz do projeto consumidor.
- o repositório possui um runner Pest funcional.

Escolha o diretório do projeto

Você pode começar a conversa na raiz de um Hub e indicar o subdiretório do projeto. Instale e execute o setup com o mesmo caminho:

```
specsfy install --project apps/portal
specsfy doctor --project apps/portal
```

Depois, informe `apps/portal` ao `$specsfy-setup`. O agente cria contexto, specs, testes e código apenas nesse diretório. Ele não usa a raiz Git do Hub como destino por dedução. Antes de cada skill seguinte, o framework executa o setup novamente para verificar essa consistência, reutilizando o caminho já confirmado na conversa.

Capture uma entrada

Use `$specsfy-01-inbox` para guardar a formulação original em `specs/inbox/`. A captura não inicia perguntas nem altera o código:

```
Use $specsfy-01-inbox para capturar:
criar uma página /boas-vindas que cumprimente o visitante pelo nome.
```

A skill grava um arquivo com data, horário e slug em `specs/inbox/`. O relato deve apontar um caminho semelhante a este:

```
specs/inbox/2026-07-28-143205-pagina-boas-vindas.md
```

Essa captura preserva o texto recebido e registra inferências separadamente. Ela ainda não cria backlog, spec, tarefas ou código.

Refine a proposta no backlog

Quando a ideia merecer refinamento, envie o arquivo para `$specsfy-02-backlog`. A skill lê a captura preservada, procura relações e cria um item numerado:

```
Use $specsfy-02-backlog para refinar
specs/inbox/2026-07-28-143205-pagina-boas-vindas.md
```

Você também pode fornecer o texto diretamente. A skill procura material relacionado, esclarece somente o necessário para o backlog e grava um item numerado:

```
specs/backlog/0001-pagina-boas-vindas.md
```

O backlog organiza uma possibilidade de entrega, mas não autoriza alteração no código. Essa separação permite comparar e priorizar ideias antes de criar uma especificação normativa.

Use `$specsfy-02-backlog` para aprofundar o item. A conversa pergunta uma lacuna aplicável por vez e retorna um brief:

```
Use $specsfy-02-backlog em
specs/backlog/0001-pagina-boas-vindas.md
```

O agente reaproveita o conteúdo existente e pergunta uma lacuna relevante por vez. Neste exemplo, a resposta padrão muda o comportamento visível da página:

```
Agente: O que deve aparecer quando nenhum nome for informado?
Você: Olá, visitante!
```

O refinamento do backlog produz um brief na conversa. Por padrão, ele não cria uma segunda fonte normativa nem modifica o backlog.

Crie a especificação única

Depois de resolver as dúvidas materiais, promova o backlog com `$specsfy-03-specify`:

```
Use $specsfy-03-specify para promover
specs/backlog/0001-pagina-boas-vindas.md
```

A skill cria o diretório numerado e mantém a fonte normativa neste caminho:

```
specs/<estado>/0001-pagina-boas-vindas/spec.md
```

Abra esse arquivo e confira se o problema, as pessoas afetadas, os requisitos, os limites e os cenários BDD representam a conversa. O Gherkin permanece na spec como referência legível. O Specsfy não cria uma suíte `.feature` separada.

Comprove a definição

Use `$specsfy-04-validate` para auditar a spec. A skill informa a localização de cada falha e só aprova o Definition Gate quando a definição estiver completa:

```
Use $specsfy-04-validate em
specs/<estado>/0001-pagina-boas-vindas/spec.md
```

Uma definição pronta termina a validação com estes dois sinais:

```
READY
Definition Gate: Passed
```

`READY` confirma que a spec possui as informações necessárias para planejar. O estado ainda não afirma que a página existe. Se houver contradição ou uma escolha importante em aberto, a validação retorna à skill responsável antes de aprovar o gate.

Organize as tarefas

Use `$specsfy-05-tasks` para manter o plano e as tarefas dentro da mesma `spec.md`. O arquivo deve mostrar os requisitos cobertos e a dependência entre o teste em RED e cada tarefa de produção:

```
Use $specsfy-05-tasks em
specs/<estado>/0001-pagina-boas-vindas/spec.md
```

A skill separa testes, código, documentação e trabalho operacional, registra dependências e liga cada tarefa aos requisitos correspondentes. Ela não cria `tasks.md` nem altera o código de produção.

Prove que o teste detecta a ausência da página

Use `$specsfy-06-tdd-bdd` no modo de preparação:

```
Use $specsfy-06-tdd-bdd em
specs/<estado>/0001-pagina-boas-vindas/spec.md para preparar o TDD.
```

Como o projeto do exemplo usa PHP, a skill cria testes Pest derivados dos cenários BDD. Cada caso executável recebe seu marcador `SPECSFY:` junto à definição. A feature inteira e cada história ou requisito aplicável precisam ter, no mínimo, três casos distintos: caminho feliz, variação importante e falha ou limite material.

Execute o teste focal e confirme o RED pelo motivo esperado. Uma rota ausente prova que o teste detecta o comportamento ainda não implementado. Erro de sintaxe, fixture quebrada ou dependência ausente precisa ser corrigido antes de o RED ser aceito. Depois que as tarefas e seus predecessores TDD estiverem coerentes, o `Plan Gate` pode chegar a `Passed`.

Implemente e valide

Com os gates de definição e plano aprovados, use `$specsfy-07-implement`:

```
Use $specsfy-07-implement em
specs/<estado>/0001-pagina-boas-vindas/spec.md
```

A implementação percorre cada tarefa em `RED → GREEN → REFACTOR`. Para uma tarefa de código, a skill exige um predecessor TDD com RED registrado, cria a menor mudança capaz de deixar o teste verde e executa a regressão aplicável. Os comandos, os resultados e os IDs cobertos entram como evidência na spec.

Depois de cada tarefa de código, o agente chama `$specsfy-documentator`. Essa skill reconstrói a documentação técnica em `docs/` a partir do sistema existente e executa o modo `--check`. O fluxo só retoma a implementação quando a documentação representar o código atual.

No fechamento, a implementação verifica aceite, regressão, rastreabilidade, documentação e Definition of Done. Uma entrega comprovada termina com:

```
Delivery Gate: Passed
Status: Reviewing
```

Incorpore uma mudança posterior

Imagine que, depois da primeira entrega, o nome precise aceitar no máximo 80 caracteres. Use `$specsfy-update-spec` na spec existente:

```
Use $specsfy-update-spec em
specs/<estado>/0001-pagina-boas-vindas/spec.md:
o nome deve ter no máximo 80 caracteres.
```

Essa skill preserva a nova instrução, atualiza a `spec.md` e invalida somente as provas afetadas. Como o limite muda comportamento, o fluxo reabre desde o Ato I, percorre validação, tarefas e TDD/BDD, e só então retoma `$specsfy-07-implement`. A skill de atualização não altera código de produção automaticamente.

Uma alteração restrita ao plano técnico reabre os Atos II e III. Uma correção editorial comprovadamente sem mudança de significado preserva os gates. Em todos os casos, o histórico continua na mesma spec.

Consulte o estado final

Use `$specsfy-progress` para projetar o estado sem editar arquivos:

```
Use $specsfy-progress para mostrar o resultado final.
```

O relatório mostra specs, gates, tarefas, checklists, pendências e o próximo trabalho disponível. Você também pode consultar o mesmo estado pelo CLI:

```
specsfy progress --project .
specsfy progress --project . --json
specsfy tui --project .
```

Depois do aceite final, a entrega pronta aparece como `Complete` em `completed/`, com os três gates aprovados e sem pendência documental. Capturas em `specs/inbox/` e itens em `specs/backlog/` não entram nesse cálculo.

Converse sobre a próxima escolha — `$specsfy-interviewer`

Quando uma lacuna puder alterar escopo, plano, execução ou aceite, use `$specsfy-interviewer` na mesma spec. Ele registra respostas confirmadas e recalibra Effort, sem aprovar gates nem substituir a skill da etapa.

Continue na mesma conversa

Você não precisa enviar cada exemplo deste tutorial manualmente. Ao autorizar a jornada completa, uma skill anuncia o handoff, carrega a próxima responsabilidade e retoma a etapa anterior quando necessário. A transição automática não amplia permissões para deploy, publicação, instalação de especialista ou ação destrutiva.

Agora aprofunde os comandos do CLI e da TUI, consulte as informações permanentes do projeto ou conheça o uso avançado.

Justificativa de tamanho

O tutorial acompanha uma única entrega desde a captura até o estado `Complete`. Manter o exemplo em uma página permite conferir como cada arquivo e gate é produzido pelo resultado da etapa anterior.

Manutenção deste guia

Atualize esta página quando a sequência dos atos, a responsabilidade de uma skill base, os gates, os estados ou os caminhos canônicos mudarem. Use a metodologia executável em `skills/` como fonte e preserve `specs/<estado>/<NNNN>-<slug>/spec.md` como a única fonte normativa de cada fatia no projeto consumidor.

Inbox

`specs/inbox/` recebe entradas antes de qualquer refinamento ou definição de produto. A captura é deliberadamente rápida. O agente guarda o texto sem fazer perguntas e sem transformá-lo em compromisso de entrega.

Capturar

```
Use $specsfy-01-inbox para guardar esta entrada:
quero permitir que a pessoa continue um formulário em outro dispositivo.
```

O agente grava a captura em `specs/inbox/` com data, hora e um nome derivado do conteúdo:

```
specs/inbox/AAAA-MM-DD-HHMMSS-<slug>.md
```

Data e hora evitam uma fila numerada artificial e preservam a ordem real de chegada. Se duas capturas tiverem o mesmo nome no mesmo segundo, o Specsfy adiciona um sufixo sem sobrescrever a anterior.

O que o arquivo organiza

- metadados e integridade do texto original.
- texto original completo.
- resumo processado.
- problema ou oportunidade.
- pessoas e valor percebidos.
- sinais de escopo, regras ou solução.
- dependências, falhas possíveis e direções futuras.
- pontos a revisar no futuro.
- rastreabilidade para backlog ou spec derivados.

Declarações, inferências e lacunas permanecem identificadas. Um campo sem base no texto aparece como não identificado, em vez de receber uma resposta inventada.

Quando o texto indicar que o sistema precisa guardar, consultar, compartilhar ou apagar informações, a Inbox registra o sinal sem perguntar. No backlog, `$specsfy-data-discovery` conversa sobre essas informações antes de o item ser considerado pronto.

O texto será versionado no Git. Não inclua senhas, tokens, chaves privadas ou dados pessoais sensíveis. Se o agente detectar um segredo evidente, ele não grava a captura e orienta você a remover o conteúdo sensível antes de reenviar.

Inbox, backlog e spec

```
captura sem perguntas → backlog refinável → spec normativa
```

- `specs/inbox/` preserva o input e funciona como fila ativa.
- `specs/inbox/processed/` guarda as capturas já cobertas por specs aprovadas.
- `specs/backlog/` organiza algo escolhido para refinamento.
- `specs/<estado>/<NNNN>-<slug>/spec.md` governa comportamento e entrega.

Uma captura pode permanecer na Inbox pelo tempo que for preciso. Quando quiser avançar, use `$specsfy-02-backlog` com o caminho do arquivo.

Quando a captura sai da Inbox

Virar backlog ou spec em Draft não retira a captura da fila ativa: enquanto qualquer parte do pedido estiver sem definição aprovada, ela continua em `specs/inbox/`, e a leitura da pasta mostra o que ainda precisa de tratamento.

A saída acontece em um único momento: quando `$specsfy-04-validate` aprova o Definition Gate de uma spec e confirma que essa spec, sozinha ou com outras já aprovadas, cobre o pedido inteiro. A validação então executa:

```
specsfy inbox process 2026-07-28-143205-avisar-clientes.md --spec 0007-avisos-de-atraso
```

O comando move o arquivo para `specs/inbox/processed/`, com o mesmo nome, e:

- troca `Status: Capturada` por `Status: Processada`;
- registra as specs que absorveram o pedido em `Spec derivada` e a data em `Processada em`;
- atualiza os links para a captura no backlog e nas specs;
- preserva o texto original e a análise inicial sem alteração.

Antes de mover, o comando confere cada spec indicada. Ela precisa estar fora da pasta `draft`, ter `Status` diferente de `Draft` e mostrar `Definition Gate: Passed` na tabela do topo. O `Status` também precisa estar escrito nessa tabela. Tabelas do corpo e exemplos não comprovam aprovação. Se o gate não estiver escrito, a spec é recusada mesmo que já esteja em outra pasta. Basta uma spec recusada para a captura continuar em `specs/inbox/`.

```
specs/inbox/                        ← capturas pendentes
specs/inbox/processed/              ← capturas integralmente cobertas por specs
aprovadas
```

Nada é apagado. Capturas processadas continuam valendo como histórico e entram na busca por duplicatas do backlog. Se uma parte do pedido ficou de fora, a captura permanece na fila ativa e a validação informa o que resta encaminhar.

Sessões de descoberta do MVP

`$specsfy-mvp-milestone-interviewer` pode usar Inboxes para capturar respostas novas durante a conversa. Isso não se aplica à importação de `MVP.md`: nela a skill cria `M01` e seleciona somente requisitos de desenvolvimento para gerar backlog e spec Draft diretamente. Os temas de negócio e produto permanecem no `MVP.md`, sem Inbox. O backlog reaproveita as respostas declaradas no trecho técnico e pergunta somente o que permanece ausente, ambíguo ou contraditório antes de uma promoção. `BRAND.md`, quando existir, orienta as perguntas, mas não é copiado para as capturas. Se o projeto estiver em um submódulo Git, a skill procura os dois arquivos na raiz do Hub somente quando eles não existirem no projeto.

Templates instalados

O CLI mantém em `.specsfy/templates/` os modelos usados para criar entradas, backlogs, specs e informações permanentes do projeto:

```
Inbox.md
Backlog.md
Spec.md
Tasks.md
Project.md
Stack.md
Rules.md
Database.md
```

Para personalizar um modelo, copie somente o arquivo desejado para `.specsfy/templates/custom/` e preserve o mesmo nome. A versão em `custom/` tem precedência sobre a cópia padrão. O instalador protege alterações locais nos templates gerenciados, mas `--force` pode substituí-los. O conteúdo de `custom/` nunca é gerenciado nem sobrescrito.

Refinar ideias no backlog

O backlog recebe uma ideia que merece conversa, mas ainda não está pronta para virar especificação. Nele, você esclarece o problema, o público afetado e o efeito esperado sem definir arquitetura, tarefas ou código nessa etapa.

Para apenas guardar um texto sem responder perguntas, use a Inbox. Depois que uma entrada for promovida, a `spec.md` passa a governar o comportamento, os gates, as tarefas e as evidências da entrega.

Estrutura

```
specs/
├── inbox/
│   ├── processed/
│   │   └── 2026-06-10-091500-ideia-ja-especificada.md
│   └── 2026-07-28-143205-ideia.md
├── backlog/
│   └── 0001-ideia.md
└── specs/
    └── 0001-feature/
        ├── spec.md
        └── research/
```

O item começa leve e amadurece até produto, desenvolvimento e testes compreenderem o comportamento esperado. Ele não contém tarefas ou gates e nunca autoriza implementação diretamente.

Organização do backlog

```
Produto
└── Épico
    └── Funcionalidade
        ├── História ou requisito
        ├── Regra
        ├── Item técnico
        └── Melhoria
```

O épico representa um objetivo amplo. A funcionalidade delimita uma capacidade. Histórias e requisitos representam entregas menores e verificáveis. Regras, itens técnicos e melhorias também pertencem ao backlog quando tornam o comportamento ou sua operação possível.

Nem toda ideia precisa começar com a hierarquia completa. Use `A esclarecer` no lugar de inventar uma relação.

Anatomia de um item

Logo abaixo do título, mantenha as metainformações em uma tabela, não em lista:

METAINFORMAÇÃO	VALOR
ID	`BACKLOG-0001`
Status	`Captured`
Produto	A esclarecer
Épico	A esclarecer
Funcionalidade	A esclarecer
Tipo	A esclarecer
Prioridade	Não priorizado
Criado em	data ISO
Spec promovida	Nenhuma

Conforme a conversa avança, o item pode registrar estas informações sem preencher campos que ainda não foram esclarecidos:

- título, tipo e prioridade.
- produto, épico e funcionalidade.
- contexto do problema.
- objetivo ou história do usuário.
- comportamento esperado.
- regras de negócio.
- condições de aceite.
- segurança, privacidade, desempenho, volume e auditoria.
- dependências.
- situações de erro e exceções.
- dentro e fora de escopo.

A profundidade é adaptativa. Uma alteração simples exige menos detalhes, enquanto autenticação, pagamentos, permissões, privacidade e processamento assíncrono exigem análise cuidadosa. O melhor item não é o mais longo, mas aquele que reduz a ambiguidade e permite verificar a entrega.

Captura mínima e conversa

Ao receber a ideia, `$specsfy-02-backlog` reaproveita a descrição original e confirma:

- problema percebido.
- pessoa afetada ou beneficiada.
- resultado ou valor esperado.
- contexto suficiente para distinguir a ideia.

Se algo estiver ausente, vago, contraditório ou ambíguo, a skill pergunta uma lacuna relevante por vez e reavalia após cada resposta. Ela não usa questionário fixo, não repete o que já foi explicado e não persiste placeholders nesses campos. Se a pessoa não souber responder, explicita a lacuna sem inventar.

Esse é o mínimo de captura, não um refinamento profundo. Hierarquia, prioridade, regras detalhadas, aceite e solução técnica podem ser refinados depois.

Duplicatas e referências

Antes de criar, a skill pesquisa termos da ideia em `specs/inbox/*.md`, `specs/inbox/processed/*.md`, `specs/backlog/*.md`, `specs/<estado>/*/spec.md` e `docs/**/*.md`. Ela separa possível duplicata de backlog relacionado, spec relacionada ou documentação útil.

Uma possível duplicata exige confirmar se o item existente será atualizado ou se há uma diferença real. Fontes úteis ficam em `Referências relacionadas`, com o caminho e o tipo de relação. Elas não substituem o que você declarou.

Padrões recorrentes

CAPACIDADE	INFORMAÇÕES QUE O BACKLOG DEVE TORNAR OBJETIVAS
Autenticação	cadastro, tentativas, sessão, mensagens e autorização
Notificações	propriedade, leitura, ações em lote e isolamento
Permissões	perfil, operação, alvo, validação e auditoria
PIX/pagamentos	estados, idempotência, webhook e dados sensíveis
Exportação	filtros, autorização, volume, fila e expiração

Use a representação que reduz ambiguidade. Fluxos numerados ajudam em integrações, matrizes ajudam em permissões e cenários demonstram resultados e erros.

Um requisito funcional descreve o que o sistema faz. Um requisito não funcional define condições mensuráveis de qualidade e operação. Troque “o sistema deve ser rápido” por um limite de latência, volume e ambiente verificáveis.

Ordenar o backlog

Mantenha o backlog realmente ordenado para que a próxima oportunidade fique visível no arquivo para todos os responsáveis. Compare os itens pelos fatores abaixo:

1. valor para a pessoa e para o negócio.
2. exposição de segurança, privacidade e operação.
3. dependências e entregas desbloqueadas.
4. urgência.
5. esforço.
6. pontos ainda desconhecidos.

Prioridade é relativa. Classificar tudo como alta não cria uma ordem. Autenticação e autorização podem preceder melhorias visuais, assim como uma infraestrutura de notificações pode preceder os eventos que dependem dela.

A tabela abaixo mostra como uma ordem real diferencia o que deve ser retomado primeiro do que pode esperar:

ORDEM	PRIORIDADE	TIPO	ITEM	OBJETIVO
1	Alta	Épico	Gestão de usuários	Administrar acesso
2	Alta	História	Realizar login	Acessar áreas privadas
3	Alta	Regra	Controlar permissões	Restringir operações por perfil
4	Média	Técnico	Notificações em fila	Evitar lentidão nas requisições
5	Baixa	Melhoria	Preferências de notificação	Escolher canais

Cada linha aponta para um item próprio. Por exemplo, “realizar login” deve esclarecer cadastro ativo, comparação de e-mail, proteção da senha, tentativas inválidas, sessão, permissões, resultados de sucesso e falha e o que ficou fora do escopo. “Criar uma tela de login” não cobre esse comportamento.

Fluxo

```
input → inbox → backlog → spec
```

1. Use `$specsfy-02-backlog` quando a entrada ainda for geral. A skill pesquisa material relacionado, conversa até a captura mínima ficar clara e produz `specs/backlog/<NNNN>-<slug>.md`. A mesma skill pode organizar, priorizar e refinar o item progressivamente.
2. A mesma skill apresenta uma pergunta numerada por rodada e produz um brief enquanto houver escolhas materiais abertas.
3. Use `$specsfy-03-specify` somente quando houver intenção explícita de promover o material. A fonte normativa é criada em `specs/<estado>/<NNNN>-<slug>/spec.md`.
4. Depois da promoção, mantenha o backlog como proveniência, marque-o `Promoted` e registre o caminho da spec.

Ao concluir, a skill informa qual etapa assumirá o trabalho e executa automaticamente o avanço ou o retorno. A conversa mostra o nome da skill, o motivo da troca e o resultado esperado. Você acompanha essa transição sem repetir comandos, e nenhuma troca autoriza implementação.

Backlog e spec possuem sequências independentes. `BACKLOG-0004` pode originar `SPEC-0002`, mas os números não precisam coincidir e não indicam prioridade.

Estados do backlog

ESTADO	SIGNIFICADO
`Captured`	ideia preservada com contexto mínimo
`Refining`	conversa ou pesquisa leve em andamento
`Ready for specification`	contexto suficiente para criar a especificação
`Promoted`	spec derivada criada e referenciada

Quando está refinado

O item está pronto para especificação quando a equipe consegue responder às perguntas abaixo sem consultar suposições fora do arquivo:

- Qual problema será resolvido e qual público será beneficiado?
- Qual evento inicia o comportamento e qual resultado será produzido?
- Qual papel pode executar a operação?
- Quais regras, erros e exceções precisam ser respeitados?
- Como verificar objetivamente o resultado?

- Há implicações de segurança, privacidade, desempenho ou volume?
- O que ficou fora da entrega?
- Quais dependências ou definições continuam pendentes?

O agente identifica lacunas e pergunta uma por rodada. Definições que alteram segurança, escopo, arquitetura ou experiência não são inventadas silenciosamente. Esse diagnóstico prepara a spec, mas ainda não autoriza desenvolvimento.

Limites

- Não implementar diretamente de um backlog.
- Não exigir arquitetura, Gherkin ou plano técnico durante a captura inicial.
- Não apagar a formulação original ao resumir.
- Não promover automaticamente uma ideia.
- Não tratar o backlog como fonte normativa depois da promoção.

Próximo passo

Quando o item estiver claro o bastante para uma conversa aprofundada, use `specsfy-02-backlog`. A promoção para `spec.md` só acontece depois de refinamento suficiente e de uma instrução explícita para criar a especificação.

Organize o produto por milestones

Comece pelo MVP

Use `$specsfy-mvp-milestone-interviewer` depois de apresentar a ideia do produto. A conversa começa com a finalidade, quem será atendido e qual jornada precisa funcionar. Depois de cada resposta, o agente resume o entendimento e faz a próxima pergunta que falta para definir o menor produto utilizável.

A entrevista termina quando você puder confirmar uma frase como: “o MVP estará pronto quando uma equipe comercial conseguir capturar um lead, consultá-lo e atribuir uma pessoa responsável em ambiente publicado”.

O agente propõe normalmente de quatro a oito milestones. Para cada uma, você aprova objetivo, condição de saída, fora de escopo, dependências e specs iniciais. Só depois disso ele cria ou reorganiza os arquivos.

Arquivos do projeto

```
PROJECT.md
specs.md
specs/
├── milestones/
│   ├── M01.md
│   └── M02.md
├── backlog/
└── <estado>/<NNNN>-<slug>/spec.md
```

`specs.md` é o mapa do projeto: mostra a sequência das specs, o estado de cada uma e os marcos vinculados. A fonte de comportamento continua em cada `spec.md`; o objetivo e a condição de saída ficam no arquivo de milestone.

Vincule specs e backlog

Inclua o campo `Milestones` nas tabelas de uma spec ou de um item do backlog:

```
| Milestones | M02, M04 |
```

Uma spec possui uma milestone principal na maioria dos casos. Ela pode ter outro vínculo quando a mesma capacidade contribui para dois estados reais do produto. O backlog aparece no marco como trabalho relacionado, mas não aumenta o percentual de conclusão.

Atualize o mapa automaticamente

Depois de alterar relações ou status, execute:

```
specsfy milestones sync --project .
```

O comando atualiza blocos identificados em `specs.md` e em `specs/milestones/MNN.md`. Texto fora desses blocos continua seu. Um marco referenciado por spec ou backlog e ainda sem arquivo recebe um esqueleto, para você completar na entrevista.

`specsfy transition` também sincroniza o mapa depois de mover uma spec. Use o comando direto quando tiver alterado vínculos no Markdown ou refinado backlog.

Cinco usos do comando

```
# Criar ou atualizar o índice no projeto atual.
specsfy milestones sync

# Declarar explicitamente a raiz do projeto atual.
specsfy milestones sync --project .

# Consumir o resultado em outra automação local.
specsfy milestones sync --project . --json

# Atualizar um projeto em outro diretório.
specsfy milestones sync --project ../crm

# Atualizar relações depois de editar uma spec ou backlog.
specsfy milestones sync --project .
```

O progresso considera specs com `Status: Complete`. A milestone só é concluída quando suas specs necessárias estiverem completas e você confirmar que a condição de saída foi demonstrada ou validada.

Planeje o que vem depois

Quando o MVP estiver aceito, use `$specsfy-roadmap-milestone-interviewer`. Ele parte dos limites já aprovados e organiza evolução, integrações, automações e hipóteses que dependem de uso real. Se uma resposta mudar o núcleo do MVP, o agente pede confirmação e encaminha a alteração da spec para `$specsfy-update-spec`.

Para revisar o mapa existente e encontrar relações ausentes, use `$specsfy-milestone-governor`. Ele sincroniza a projeção, aponta lacunas e propõe ajustes; não altera objetivo ou condição de saída sem sua confirmação.

Skills base



As skills base dividem o método por responsabilidade. Você pode chamar uma delas pelo nome ou explicar o resultado esperado. O agente lê o estado da spec, seleciona a etapa responsável e anuncia cada transição necessária.

Nomes exibidos

Os comandos técnicos continuam, por exemplo, como `$specsfy-02-backlog`. Na interface, as sete etapas centrais aparecem como `Specsfy - 01 - Inbox` até `Specsfy - 07 - Implementar`. As skills adicionais usam `Specsfy - Nome`, e as técnicas usam `Specsfy - Especialista - Nome`.

Como responder às perguntas

Toda skill que precisa perguntar segue o mesmo formato desde a primeira rodada:

1. apresenta uma pergunta com o rótulo `Pergunta 1` e espera sua resposta;
2. oferece pelo menos três respostas sugeridas e numeradas abaixo dela;
3. acrescenta `Escrever outra resposta` para você informar seu próprio texto;
4. acrescenta `Gere outras opções` para mostrar alternativas diferentes à mesma pergunta;
5. acrescenta `Avançar` para abrir a confirmação de encerramento da área, adiamento ou retomada imediata.

Você pode responder com combinações como `1.2` ou `1.4: meu texto`. O número da opção é convertido no texto completo antes de gerar qualquer contexto. Ao escolher `Avançar`, a rodada seguinte pergunta se você quer encerrar definitivamente as perguntas daquela área, responder depois ou voltar a responder agora. Se encerrar, a skill registra sua escolha e não pergunta sobre a área novamente, a menos que você a reabra. Se adiar, os pontos ficam registrados para retomada. Nenhuma das escolhas inventa uma resposta ou aprova uma etapa incompleta.

Inbox, progresso, auxiliares de stack e banco e documentador não conduzem entrevista. Essas skills registram ou projetam o que já existe e encaminham qualquer pergunta para uma etapa conversacional.

ETAPA	SKILL	RESULTADO PRINCIPAL
capturar sem perguntas	`specsfy-01-inbox`	arquivo em `specs/inbox/`

refinar uma entrada e aprofundar definições	`specsfy-02-backlog`	item em `specs/backlog/` e brief na conversa
criar a fonte única	`specsfy-03-specify`	`spec.md`
revisar definição	`specsfy-04-validate`	Definition Gate confiável
decompor o plano	`specsfy-05-tasks`	tarefas dentro da spec
preparar e executar testes	`specsfy-06-tdd-bdd`	RED/GREEN rastreável
produzir a mudança	`specsfy-07-implement`	código, testes e evidência
incorporar mudança posterior	`specsfy-update-spec`	spec atualizada e gates reabertos
consultar o estado	`specsfy-progress`	relatório somente leitura
conversar conforme a fase	`specsfy-interviewer`	respostas confirmadas e Effort recalibrado
definir o MVP e seus marcos	`specsfy-mvp-milestone-interviewer`	milestones aprováveis do MVP
descobrir o que o sistema precisa guardar	`specsfy-data-discovery`	respostas confirmadas em `DATABASE.md`
planejar a evolução	`specsfy-roadmap-milestone-interviewer`	milestones pós-MVP
manter a projeção	`specsfy-milestone-governor`	`specs.md` e progresso derivado

Encontre a skill pelo estado do trabalho

Uma anotação que precisa ser preservada vai para `specsfy-01-inbox`. Quando você quiser comparar essa ideia com itens existentes e esclarecer o mínimo necessário, `specsfy-02-backlog` cria ou atualiza o arquivo numerado e aprofunda as definições que mudam o comportamento e entrega um brief na conversa.

Com a intenção de criar uma entrega confirmada, `specsfy-03-specify` monta a `spec.md` e `specsfy-04-validate` comprova o Ato I. A skill `specsfy-05-tasks` organiza o plano, chama `specsfy-06-tdd-bdd` para materializar os testes e só aprova o Plan Gate depois de um RED válido. `specsfy-07-implement` executa as tarefas com evidência e documentação atualizada.

Uma necessidade surgida depois da definição retorna à mesma spec por `specsfy-update-spec`. Para apenas consultar gates, tarefas e o próximo trabalho sem alterar arquivos, use `specsfy-progress`.

Quando uma lacuna puder alterar a próxima etapa, chame `specsfy-interviewer`. Ele conversa com a spec sem substituir a skill que valida, planeja, implementa ou conclui.

Para organizar um produto inteiro, comece pelo `specsfy-mvp-milestone-interviewer`. Depois do aceite do MVP, use o `specsfy-roadmap-milestone-interviewer`. O `specsfy-milestone-governor` mantém o mapa derivado de specs e backlog. O guia Milestones explica arquivos, relações e sincronização.

Quando uma conversa revelar informações que o sistema precisa lembrar, use `specsfy-data-discovery`. A skill pergunta em linguagem simples e mantém o registro em `.specsfy/DATABASE.md` antes de o backlog ou a spec avançar.

Volte ao guia completo ou leia como a metodologia funciona.

Capturar uma entrada com `specsfy-01-inbox`

Esta skill funciona como uma caixa de entrada: recebe seu texto, guarda o original e organiza uma primeira leitura sem interromper você com perguntas.

Quando usar

Use quando quiser anotar uma entrada, oportunidade, necessidade ou pensamento para retomar depois. O arquivo fica em `specs/inbox/`.

Não use para refinar prioridade, decidir requisitos ou iniciar implementação. Esses trabalhos pertencem às etapas seguintes.

Como descrever a tarefa

```
Use $specsfy-01-inbox para capturar:
seria útil avisar clientes quando uma entrega atrasar, talvez por e-mail.
```

Se preferir uma instrução curta, escreva “guarde na Inbox” e inclua o texto original na mesma mensagem. A skill organiza a captura sem exigir um formato prévio.

Exemplo passo a passo

1. Você envia o texto livre.
2. O agente preserva o texto original.
3. Sem fazer perguntas, ele separa o resumo, o problema ou a oportunidade, as pessoas afetadas, o resultado esperado, as dependências e os pontos que ainda precisam de revisão.
4. Ele cria um nome com data, hora e slug.
5. Ele informa o caminho criado e encerra a captura.

O que esperar

```
specs/inbox/2026-07-28-143205-avisar-clientes-sobre-atrasos.md
```

O arquivo diferencia o que você declarou, o que foi inferido e o que ainda precisa ser revisto. Nenhum backlog, spec, tarefa ou código é criado.

Os metadados incluem horário da captura, origem, slug, status, hash do texto original e links futuros para backlog ou spec.

A captura nasce com `Status: Capturada` e fica em `specs/inbox/` enquanto o pedido tiver alguma parte sem definição aprovada. Quando specs com Definition Gate aprovado cobrirem o pedido inteiro, a validação a move para `specs/inbox/processed/` com `Status: Processada`. O texto original não muda. Leia Quando a captura sai da Inbox.

Erros comuns

- esperar que a captura faça perguntas ou complete definições ausentes.
- tratar a análise inicial como requisito aprovado.
- implementar diretamente a partir de `specs/inbox/`.
- editar o texto original em vez de registrar uma evolução no backlog.
- mover ou apagar a captura à mão: só `specsfy inbox process`, executado pela validação, a retira da fila ativa.
- procurar o template dentro da skill: o padrão vive em `.specsfy/templates/Inbox.md`, e uma personalização homônima em `.specsfy/templates/custom/` tem precedência.

Próximo passo

Deixe a entrada guardada ou use `specsfy-02-backlog` quando quiser refiná-la.

Refinar uma entrada com `specsfy-02-backlog`

Esta skill transforma uma entrada da Inbox em backlog refinável. Ela procura itens parecidos, registra o contexto inicial e, quando necessário, conduz uma pergunta prioritária por vez até produzir um brief pronto para especificar.

Quando usar

Use quando quiser organizar uma captura em `specs/inbox/`, avaliar uma oportunidade ou fechar lacunas sobre público, finalidade, regras, limites, privacidade, falhas e resultado esperado. Para apenas salvar sem perguntas, use `specsfy-01-inbox`.

Não use para planejar tarefas, escrever testes ou iniciar código, pois um item de backlog ainda não passou pelos gates que autorizam essas etapas.

Como descrever a tarefa

Descreva a entrada e o destino esperado na mesma mensagem. A skill usará esse texto para criar um item reconhecível em `specs/backlog/`, sem tratá-lo como autorização para implementar. Por exemplo:

```
Use $specsfy-02-backlog para refinar esta entrada:
permitir que a pessoa escolha o idioma da interface.
```

Quando houver itens parecidos em `specs/backlog/`, inclua a situação que diferencia esta entrada das demais. Isso ajuda a skill a atualizar o item correto e a manter visíveis as definições que continuam abertas:

```
Anote no backlog: clientes internacionais não entendem os e-mails atuais.
Ainda não decidimos quais idiomas serão oferecidos.
```

Exemplo passo a passo

1. Você apresenta a entrada ou aponta o arquivo da Inbox.
2. O agente também pode ler a origem em `specs/inbox/`.
3. Ele procura itens semelhantes em `specs/backlog/` e nas specs.
4. Ele pergunta uma lacuna material por vez.
5. Você responde: “o problema afeta e-mails e a interface”.
6. A skill cria o item com `.specsfy/templates/custom/Backlog.md` quando presente ou com `.specsfy/templates/Backlog.md`:

```
specs/backlog/0003-idioma-da-interface.md
```

O item registra problema, público, resultado esperado e dúvidas abertas. Ele continua sendo backlog e não autoriza implementação.

Quando a entrega incluir interface, o refinamento também pergunta somente o que ainda não foi dito sobre telas, fluxo de informação, menus e navegação principal, formulário, padrão de abertura da ação e disposição dos elementos. Você pode pedir alternativas para uma página, painel lateral ou modal. As respostas ficam registradas em texto para orientar a spec.

Antes disso, o agente analisa o sistema existente. Rotas, telas, componentes, conteúdo, permissões, estados, testes e stack indicam o que deve ser preservado e evitam sugestões incompatíveis com o projeto.

Como o ciclo termina

O refinamento faz no máximo oito perguntas por área. Cada rodada traz exatamente uma pergunta numerada. Ela oferece três ou mais opções numeradas, `Escrever outra resposta`, `Gere outras opções` e `Avançar`. Ao escolher outras opções, o agente mantém a pergunta e apresenta sugestões diferentes. O agente reconsidera o pedido e as respostas antes de montar a próxima rodada.

Ao chegar a oito perguntas, o agente resume o que foi confirmado e o que ficou aberto. Ele só continua se você pedir explicitamente mais perguntas e informar quantas quer responder.

`Avançar` existe desde a primeira rodada. Na rodada seguinte, você escolhe se quer encerrar definitivamente as perguntas daquela área, responder depois ou voltar a responder agora. O encerramento fica registrado e é respeitado até você reabrir a área. O adiamento preserva os pontos para retomada. Quando ainda houver lacunas aplicáveis, a spec permanece `Status: Draft` e o `Definition Gate: Pending`.

O que esperar

- perguntas adaptadas ao caso, agrupadas em rodadas numeradas.
- preservação das suas palavras.
- indicação de duplicatas ou relações.
- distinção entre fato, hipótese e escolha confirmada.
- um brief pronto para especificar.
- um caminho claro para retomar depois.
- nenhuma `spec.md` criada automaticamente sem promoção.

Erros comuns

- transformar o backlog em uma especificação completa.
- inventar solução técnica quando o problema ainda está aberto.
- criar um segundo item sem procurar duplicatas.
- tratar o item como aprovação para implementar.
- responder categorias como um formulário fixo.
- esconder uma dúvida para encurtar a conversa.

Próximo passo

Quando o backlog estiver claro, use `specsfy-03-specify`. Se ainda houver uma decisão material aberta, continue o ciclo nesta mesma skill.

Criar a especificação com `specsfy-03-specify`

Esta skill cria a fonte única da entrega no arquivo `spec.md`. Ao abrir esse arquivo, você encontra a descoberta usada como base para os requisitos, os comportamentos que orientam o plano e a ligação entre tarefas, testes e evidências. Essa concentração evita que arquivos paralelos apresentem versões diferentes da mesma entrega.

Quando usar

Use para promover um backlog refinado ou criar uma spec nova ainda em estado Draft. Para mudar uma spec que já foi definida, use update-spec.

Se precisar confirmar arquivo, síntese ou próximo passo, a skill apresenta uma pergunta numerada. Ela inclui três ou mais respostas sugeridas, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a primeira rodada.

Como descrever a tarefa

Quando a ideia já tiver sido refinada no backlog, informe o caminho do item:

```
Use $specsfy-03-specify para promover
specs/backlog/0003-idioma-da-interface.md.
```

Quando o refinamento do backlog tiver produzido um brief na conversa, peça a criação com base nas definições registradas:

```
Use $specsfy-03-specify para criar uma especificação para recuperação de senha
com base nas definições desta conversa.
```

Exemplo passo a passo

1. A skill confirma a raiz do projeto e o próximo número disponível.
2. Ela lê as instruções, o brief e as evidências necessárias.
3. Cria o pacote:

```
specs/<estado>/0004-recuperar-senha/
├── spec.md
└── research/        # somente quando houver pesquisa externa
```

Em seguida, ela preenche o Ato I com requisitos e cenários, mantém as escolhas ainda abertas claramente marcadas e executa os validadores. Um resultado incompleto permanece pendente em vez de receber uma aprovação sem evidência. Quando falta uma decisão material, a skill entrega a conversa ao ciclo de refinamento do backlog e retoma depois. Se você escolher `avançar`, a spec pode registrar o brief parcial, mas permanece Draft com o Definition Gate pendente. A mesma o refinamento não é reaberto imediatamente nessa retomada.

Os arquivos em `research/` apoiam as escolhas registradas, mas nunca substituem o texto normativo de `spec.md`. Quando houver divergência, a entrega e seus validadores devem seguir a spec.

O que esperar

- formato `Specsfy/2.0`.
- IDs rastreáveis para histórias, requisitos e condições de aceite.
- cenários que cobrem sucesso, variação e falha.
- para interfaces, telas, fluxo de informação, menus e navegação principal, formulários, composição, estados e acessibilidade na seção 10.
- a stack e as telas atuais analisadas, com o que será preservado e alterado.
- uma única fonte normativa.
- status Draft ou Defined conforme a evidência real.

Erros comuns

- criar `plan.md`, `tasks.md` ou `research.md`.
- copiar uma fonte externa como requisito sem confirmação.
- aprovar gates com campos incompletos.
- misturar várias entregas grandes na mesma spec.
- descrever um CRUD apenas com API, serviço ou banco, sem a experiência que a pessoa precisa usar.

Próximo passo

Use `specsfy-04-validate` para provar que a definição está pronta. Se uma escolha importante faltar, a transição volta para `specsfy-02-backlog`.

Validar a definição com `specsfy-04-validate`

Esta skill revisa a spec como código em linguagem natural. Primeiro verifica o formato. Depois avalia clareza, completude, consistência e possibilidade de teste.

Quando usar

Use para comprovar o Ato I ou quando uma mudança exigir nova validação da definição. Também é útil para auditar uma spec sem alterar sua intenção.

Quando a validação depender de uma escolha sua, a rodada traz exatamente uma pergunta numerada. Ela oferece três ou mais respostas sugeridas, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a primeira rodada.

Como descrever a tarefa

```
Use $specsfy-04-validate em
specs/<estado>/0004-recuperar-senha/spec.md.
```

Exemplo passo a passo

1. A skill confirma o caminho e o formato `Specsfy/2.0`.
2. Verifica se os requisitos possuem exemplos suficientes.
3. Encontra uma lacuna: “a validade do link não foi decidida”.
4. Retorna para o refinamento do backlog e registra a validade de 30 minutos.
5. Executa novamente os validadores.
6. Atualiza a seção de gates:

```
Definition Gate: Passed
Status: Defined
```

1. Move a spec para `defined` e confere as capturas de `specs/inbox/` que a originaram. Se a spec, sozinha ou com outras já aprovadas, cobre o pedido inteiro, executa `specsfy inbox process <captura> --spec <id>` e a captura passa para `specs/inbox/processed/`. Se ficou alguma parte sem spec aprovada, a captura permanece na fila ativa e a skill informa o que resta.

Quando o plano inclui uma tarefa `[MIGRATION]`, a validação final também confere a entrega registrada pela implementação. A tarefa só pode terminar quando a comprovação relaciona o arquivo versionado da migration, o comando que a aplicou no banco de teste e o comando que

consultou o estado depois da aplicação. Uma tarefa `[MIGRATION]` ainda aberta também impede o Delivery Gate.

Uma falha de parser, fixture ou ambiente não serve como prova de requisito ausente.

O que esperar

- problemas apontados com localização e motivo.
- nenhuma aprovação baseada em suposição.
- verificação das evidências externas citadas.
- conferência da aplicação das migrations previstas no plano.
- retorno automático à etapa que pode resolver a lacuna.
- gate aprovado somente após nova validação.
- captura da Inbox retirada da fila ativa somente quando o pedido inteiro estiver coberto por specs aprovadas.

Erros comuns

- marcar READY sem resolver uma ambiguidade material.
- confundir estrutura válida com conteúdo suficiente.
- criar um relatório paralelo em vez de atualizar a seção correta.
- compensar um Ato I incompleto em uma etapa posterior.

Próximo passo

Com o Definition Gate aprovado, use `specsfy-05-tasks`. Se a validação encontrar uma escolha aberta, use `specsfy-02-backlog`.

Planejar tarefas com `specsfy-05-tasks`

Esta skill transforma uma definição aprovada em tarefas pequenas, ordenadas e verificáveis. As tarefas ficam na seção 14 da própria `spec.md`.

Quando usar

Use depois do Definition Gate ou quando uma alteração exigir replanejamento. Não use para escrever código nem para marcar uma tarefa como concluída.

Se houver mais de um recorte, ordem ou próximo passo possível, a skill reúne a consulta em uma pergunta numerada. Ela oferece três ou mais sugestões, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a primeira rodada.

Como descrever a tarefa

```
Use $specsfy-05-tasks em
specs/<estado>/0004-recuperar-senha/spec.md.
```

Se a spec contiver mais de uma entrega observável, indique a fatia vertical que deve ser planejada primeiro:

```
Prepare as tarefas da primeira fatia vertical da spec 0004.
```

Exemplo passo a passo

1. A skill lê a spec e o código existente.
2. Identifica a menor entrega observável: solicitar o link.
3. Liga cada tarefa aos requisitos e às condições de aceite correspondentes.
4. Faz cada tarefa de produção depender de um teste com RED registrado.
5. Registra:

```
T001 [ ] Criar caso TDD para solicitação válida — cobre AC-001
T002 [ ] Criar caso TDD para e-mail desconhecido — cobre AC-002
T003 [ ] Implementar solicitação sem revelar existência do cadastro
```

Quando a spec declara uma interface, o plano inclui tarefas para telas, menus, navegação, formulário, ações e seus testes de navegação, validação e recuperação de erro. Uma tarefa de API ou persistência não substitui essas tarefas.

Quando qualquer tarefa cria ou altera schema, tabela, coluna, índice, relação ou model persistente, o plano inclui uma tarefa separada com as tags `[CODE] [MIGRATION]`. Ela aponta para o arquivo versionado e fica entre os testes TDD e o código que passa a usar a nova estrutura.

```
T004 [CODE] [MIGRATION] Criar tabela em
database/migrations/2026_09_04_120000_create_clients_table.php
```

O `VERIFY` registra um comando que aplica a migration no banco de teste e outro que consulta seu estado. Em Laravel, por exemplo:

```
php artisan migrate --env=testing
php artisan migrate:status --env=testing
```

Sem a tarefa `[MIGRATION]`, o validador recusa um plano que crie ou altere a estrutura do banco. Consultar ou usar uma tabela existente não exige migration por si só.

Essas tarefas ficam em uma `Fase de interface` dedicada. Há uma tarefa por tela registrada, usando os componentes e a stack já existentes no projeto. Em projetos React, o `PREP` de cada tarefa carrega `$specsfy-specialist-react-ui-components` antes da escrita de JSX ou TSX. Se o especialista estiver ausente, o fluxo retorna ao setup para instalar a skill detectada antes de liberar o código da tela.

Depois de registrar as tarefas, a skill chama `specsfy-06-tdd-bdd` para materializar os testes. O Plan Gate só pode ser aprovado quando todos os predecessores exigidos possuem RED válido.

O que esperar

- tarefas pequenas e com resultado verificável.
- ordem explícita de dependência.
- testes com RED como predecessores do código.
- caminhos e comandos reais do projeto.
- migration explícita para toda tarefa ligada ao banco.
- tarefas mantidas dentro da fonte única.

Erros comuns

- criar `tasks.md`.
- escrever tarefas vagas como “fazer backend”.
- colocar várias mudanças independentes em uma tarefa.
- planejar sem inspecionar a stack real.
- marcar uma tarefa pronta sem evidência.

Próximo passo

Use `specsfy-06-tdd-bdd` em modo `prepare` para materializar o próximo teste e observar RED.

Preparar testes com `specsfy-06-tdd-bdd`

Esta skill usa os cenários da spec para criar testes executáveis. Ela mantém a rastreabilidade entre o comportamento descrito e a prova no código.

Quando usar

Use para preparar o RED que autoriza a implementação, executar um ciclo RED–GREEN–REFACTOR ou verificar testes e rastreabilidade.

Se precisar escolher runner, comando ou caso focal, a skill apresenta exatamente uma pergunta numerada. Ela contém três ou mais respostas sugeridas, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a primeira rodada.

Como descrever a tarefa

Para preparar o próximo teste focal e produzir a evidência de RED, use o modo `prepare`:

```
Use $specsfy-06-tdd-bdd em modo prepare para
specs/<estado>/0004-recuperar-senha/spec.md.
```

Para conferir os testes e a rastreabilidade de uma entrega existente, use o modo `verify`:

```
Use $specsfy-06-tdd-bdd em modo verify na spec 0004.
```

Exemplo passo a passo

1. A skill seleciona a próxima condição de aceite.
2. Encontra o runner real do projeto.
3. Em Laravel, exige `.env.testing` com destino explícito e separado do `.env`.
4. Confere o comando com `check_database_safety.mjs`. Se o resultado não for `SAFE`, não executa nenhum teste.
5. Cria um teste com marcador de rastreabilidade.
6. Executa somente o teste focal.
7. Confirma:

```
RED válido: o teste falhou porque a recuperação ainda não foi implementada.
Caso: TDD-AC-001
```

Depois da implementação, a skill repete o teste focal e executa a regressão para confirmar que o comportamento ficou GREEN sem quebrar o restante.

O bloco Gherkin da spec é uma referência legível. A prova automatizada fica na suíte normal do projeto, não em um arquivo `.feature` nem em uma segunda suíte.

O que esperar

- caso de teste ligado a uma condição de aceite da spec.
- comando e resultado registrados.
- distinção entre falha esperada e problema de ambiente.
- cobertura de sucesso, regra e limite.
- regressão depois do GREEN.

Erros comuns

- chamar erro de configuração de RED.
- escrever produção no modo `prepare`.
- criar testes que não correspondem às condições de aceite.
- considerar o Gherkin sozinho como teste executado.
- aprovar o plano sem prova focal.
- executar teste para descobrir se a conexão está correta.
- recriar ou limpar todo o banco para preparar a suíte.

Próximo passo

Com RED válido e tarefa pronta, use `specsfy-07-implement`. Para apenas reorganizar as tarefas, volte a `specsfy-05-tasks`.

Entregar código com `specsfy-07-implement`

Esta skill executa as tarefas aprovadas da spec em ordem. Ela altera produção somente quando existe um plano válido e um teste focal em RED.

Quando usar

Use para implementar a próxima tarefa pronta, continuar uma entrega ou concluir uma feature planejada. Não use para pular definição, planejamento ou testes.

Quando uma autorização ou escolha for necessária, a skill apresenta exatamente uma pergunta numerada. Ela contém três ou mais respostas sugeridas, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a primeira rodada.

Como descrever a tarefa

```
Use $specsfy-07-implement para executar a próxima tarefa pronta de
specs/<estado>/0004-recuperar-senha/spec.md.
```

Quando houver mais de uma tarefa pronta, indique o ID da tarefa que deve ser executada:

```
Implemente T003 da spec 0004 e valide a regressão.
```

Exemplo passo a passo

1. A skill confirma Definition Gate e Plan Gate aprovados.
2. Para uma interface, confere as telas, menus, formulário, ações e estados definidos.
3. Em React, carrega `$specsfy-specialist-react-ui-components`, localiza os componentes atuais e define quais serão reaproveitados ou adaptados antes de escrever JSX ou TSX.
4. Verifica a tarefa predecessora e o RED atual.
5. Em Laravel, confirma `.env.testing` separado do `.env` e passa o comando pelo `check_database_safety.mjs`. Sem `SAFE`, interrompe antes do teste.
6. Faz a menor mudança de produção, incluindo a tela e a interação previstas.
7. Executa o teste focal até obter GREEN.
8. Em uma tarefa `[MIGRATION]`, aplica o arquivo no banco de teste e consulta o estado das migrations.
9. Faz a revisão visual obrigatória quando a tarefa puder alterar a interface, mesmo sem pedido específico. Confere bordas, espaçamentos, margens, padding, tipografia,

alinhamento, largura, overflow, foco, zoom e conteúdo curto ou longo nos viewports e estados aplicáveis.

10. Refatora sem alterar o comportamento.
11. Executa a regressão e atualiza os registros da tarefa:

```
T003 [x] Implementar solicitação sem revelar existência do cadastro
Teste focal: passou
Revisão visual: Não aplicável; a tarefa altera somente a regra de serviço.
Regressão: passou
```

O checklist normativo da tarefa segue `PREP`, `EXECUTE`, `VERIFY`, `VISUAL`, `EVIDENCE` e `IMPROVE`. O item `VISUAL` registra a inspeção da interface ou o motivo concreto para sua não aplicação.

Para `[MIGRATION]`, o comentário `specsfy:evidence` precisa listar o arquivo criado, o comando de aplicação e a consulta de estado, todos com saída zero. `verify_evidence.mjs` confere os três elementos. A existência do model ou do teste não substitui essa comprovação. Durante o aceite final, `--delivery` também recusa qualquer tarefa `[MIGRATION]` que permaneça aberta.

Depois de cada tarefa de código, a skill chama o documentador do projeto consumidor. A execução só continua quando `docs/` estiver atualizado.

Runner próprio de migrations

Se o projeto aplica migrations com um script próprio, declare os comandos em `.specsfy/migrations.json`, na raiz do projeto consumidor. Assim, você pode registrar os comandos realmente executados sem editar o validador instalado. Por exemplo, para um script Node com ações `apply` e `status`:

```
{
  "apply": ["^node scripts/migrate\\.mjs apply$"],
  "status": ["^node scripts/migrate\\.mjs status$"]
}
```

Cada lista contém expressões regulares JavaScript sem flags. No exemplo, `apply` reconhece a aplicação e `status` reconhece a consulta do estado. Adapte os padrões aos comandos do seu projeto, incluindo os argumentos usados no banco de teste. Prefira padrões específicos: uma expressão como `.*` aceitaria qualquer comando como comprovação.

As duas listas precisam conter pelo menos uma expressão válida e não vazia. JSON inválido ou padrão inválido faz a validação falhar com o caminho do arquivo no diagnóstico. Os comandos nativos de migrations continuam aceitos. O arquivo da migration precisa existir e os comandos registrados precisam ter saída zero; declarar um runner não executa migrations nem comprova que o banco foi atualizado. O comentário `specsfy:evidence` deve refletir a execução real.

Ao atualizar a stack, `specsfy-aux-stack` registra a presença dessa configuração em `.specsfy/STACK.md` e preserva o conteúdo de `migrations.json`.

O que esperar

- uma tarefa por vez.
- mudanças limitadas ao escopo aprovado.
- testes focais e regressão.
- documentação aplicável atualizada.
- status e checkboxes comprovados por evidência.

Erros comuns

- implementar com gate pendente.
- aceitar um RED causado por dependência ausente.
- ampliar o escopo sem atualizar a spec.
- implementar um CRUD como API sem os menus, as telas e o formulário aprovados.
- marcar conclusão sem regressão.
- marcar uma tarefa de banco sem criar, aplicar e conferir a migration.
- rodar teste no banco do `.env` ou aceitar um comando que recrie o banco.
- deixar `docs/`, `PROJECT.md` ou os arquivos `.specsfy/` incompatíveis com o código alterado.

Próximo passo

Continue com a próxima tarefa ou consulte `specsfy-progress`. Se surgir uma necessidade nova, use `specsfy-update-spec`.

Incorporar mudanças com `specsfy-update-spec`

Esta skill atualiza uma spec que já foi definida, planejada, iniciada ou concluída. A nova necessidade entra na mesma fonte, e somente os atos cujas provas perderam validade são reabertos.

Quando usar

Use quando uma entrega existente precisar incorporar algo esquecido, adicionar ou remover comportamento, corrigir uma definição ou mudar uma regra já registrada.

Para criar a primeira versão da spec, use specify. A update-spec depende de uma fonte existente para comparar a nova instrução com requisitos, testes, tarefas e gates já registrados.

Como descrever a tarefa

```
Use $specsfy-update-spec para adicionar expiração de 30 minutos à
specs/<estado>/0004-recuperar-senha/spec.md.
```

Para remover um comportamento, identifique a exigência e o arquivo `spec.md` que deve ser revisto. Assim, a skill consegue localizar testes e tarefas que ficariam incompatíveis com a remoção:

```
Remova a exigência de SMS da spec 0004 e ajuste o trabalho afetado.
```

Exemplo passo a passo

1. A skill preserva literalmente a nova instrução.
2. Lê requisitos, testes, tarefas, gates e evidências atuais.
3. Classifica a mudança:

```
Mudança de definição: altera comportamento e condição de aceite.
Atos reabertos: I, II e III.
```

Com o impacto identificado, a skill atualiza a mesma `spec.md` e invalida somente os gates que perderam validade. Depois, retoma refinamento, validação, tarefas, testes e implementação na ordem necessária. A nova evidência é registrada sem apagar o histórico relevante.

Se a mudança abrir escolhas materiais, o refinamento do backlog apresenta exatamente uma pergunta numerada por rodada e reavalia o contexto após a resposta. A pergunta oferece três ou mais opções numeradas, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a

primeira rodada. O ciclo faz no máximo oito perguntas por área. O avanço abre uma confirmação para encerrar definitivamente as perguntas daquela área, responder depois ou retomar agora. O encerramento é respeitado até você reabrir a área. O adiamento preserva os pontos e mantém o Ato I reaberto, sem iniciar outra vez o mesmo ciclo nessa retomada.

O que esperar

- instrução original preservada.
- impacto explicado na retomada.
- nenhuma spec duplicada.
- trabalho já válido mantido.
- transições automáticas até o estado coerente.

Erros comuns

- editar apenas o código e deixar a spec antiga.
- reabrir todos os gates sem analisar impacto.
- manter teste ou tarefa que contradiz a nova instrução.
- esconder a mudança em notas soltas.
- criar uma segunda especificação para a mesma entrega.

Próximo passo

A própria skill encaminha para a etapa reaberta. Depois da atualização, use `specsfy-progress` para revisar o estado geral.

Consultar o estado com `specsfy-progress`

Esta skill lê todas as specs e apresenta uma visão geral de gates, tarefas, checklists, falhas que impedem avanço e próximo trabalho. Ela não altera nenhum estado.

Quando usar

Use para saber quanto falta, qual falha impede uma entrega, qual tarefa está pronta ou quais specs foram concluídas.

Como descrever a tarefa

Para receber uma síntese na conversa, descreva o projeto que deseja consultar:

```
Use $specsfy-progress para mostrar o progresso do projeto.
```

No terminal, use o CLI para obter a mesma leitura das fontes canônicas:

```
specsfy progress --project .
```

Exemplo passo a passo

1. A skill lê `specs/<estado>/*/spec.md`.
2. Calcula o estado a partir de gates e checkboxes.
3. Não consulta um relatório paralelo.
4. Apresenta:

```
Specs: 2
Complete: 1
Implementing: 1
Tarefas: 7 de 10 concluídas
Próximo trabalho: T004 da spec 0004-recuperar-senha
Pendências impeditivas: nenhuma
```

Se os metadados estiverem incoerentes, o relatório mostra a pendência em vez de inventar um percentual confiável.

O que esperar

Checklists fora das tarefas

O relatório separa os itens vinculados às tarefas dos outros checkboxes. No CLI, o percentual usa a tarefa e seus itens canônicos; anotações soltas ficam na contagem de outros checkboxes. Exemplos dentro de blocos de código são ignorados. Sem itens de tarefa, o CLI calcula o percentual pelos três gates.

Itens como `PREP` ou `VERIFY` fora de uma tarefa aparecem com o número da linha. O aviso permite localizar um rascunho residual sem alterar a spec durante a consulta. No JSON do CLI, consulte `orphan_checklist_lines` por spec e `orphan_checklist_items` no resumo. A skill expõe as mesmas linhas e mantém os totais de checklists separados dos totais de tarefas.

Para corrigir um resíduo, peça a `$specsfy-update-spec` que confira as linhas indicadas. Quando o texto apenas duplica uma tarefa já concluída, a skill pode convertê-lo em anotação sem checkbox, preservando o conteúdo histórico e os gates. Se o trecho representar trabalho ainda necessário, ele precisa voltar ao planejamento da spec.

O que esperar

- leitura somente das fontes canônicas.
- totais de specs e tarefas.
- gates pendentes.
- falhas impeditivas e próximo trabalho.
- saída humana ou JSON pelo CLI.

Erros comuns

- alterar checkboxes durante a consulta.
- manter um segundo arquivo de progresso.
- calcular conclusão só pelo número de arquivos.
- esconder spec inválida do relatório.
- confundir progresso com autorização para implementar.

Próximo passo

Abra a página da skill indicada pelo próximo trabalho. Para acompanhar continuamente no terminal, use:

```
specsfy progress --project . --watch
```

Conversar com uma spec

`specsfy-interviewer` ajuda você a resolver lacunas que podem mudar a próxima etapa da spec. Ele lê o estado atual, apresenta exatamente uma pergunta numerada por rodada e só registra respostas que você confirmou. A pergunta traz três ou mais opções numeradas, `Escrever outra resposta`, `Gere outras opções` e `Avançar`. Depois de avançar, você escolhe entre encerrar definitivamente as perguntas da área, responder depois ou retomar agora. A skill registra a escolha e respeita o encerramento até uma reabertura explícita.

Quando usar

Use durante `draft`, `defined`, `planned`, `in-progress` ou `review` quando uma escolha de produto, escopo, dependência, teste ou aceite ainda estiver aberta. A Inbox não usa entrevistador: ela preserva sua mensagem sem perguntas.

Como descrever a tarefa

Peça “converse comigo sobre a spec de login social antes de montar as tarefas” ou informe o ID da spec e a dúvida que precisa resolver.

Exemplo passo a passo

O entrevistador lê a spec e agrupa os pontos que impedem o plano, como o provedor de identidade, o comportamento quando a conta já existe e o método de recuperação. Você pode escolher uma opção, escrever outra resposta ou avançar.

```
specs/planned/0042-login-social/spec.md
```

Depois de cada resposta, ele verifica se mudou a estimativa de execução. Se necessário, registra `Effort`, data e justificativa com o comando do CLI.

O que esperar

Effort vai de 1 a 10 e mede a capacidade de execução necessária, não duração:

- 1–2: alteração atômica;
- 3–6: trabalho local ou integração conhecida;
- 7–8: migração ou integração transversal;
- 9–10: arquitetura ou incerteza que pede revisão humana frequente.

O histórico fica na própria `spec.md`, para que você entenda por que a estimativa mudou.

Erros comuns

- usar o entrevistador para capturar uma Inbox, que não recebe perguntas;
- esperar que ele aprove um gate ou mova a pasta sem a skill da etapa;
- usar Effort como prazo ou vinculá-lo a um modelo específico.

Próximo passo

O entrevistador não aprova gates, implementa código ou move a spec. Ao fechar a lacuna, ele entrega o trabalho à skill responsável pela fase atual. Quando ClickUpfy estiver instalado e houver uma tarefa vinculada, essa skill também atualiza a projeção remota.

Definir o MVP por milestones

Use `$specsfy-mvp-milestone-interviewer` (`specsfy-mvp-milestone-interviewer`) para explorar o menor produto utilizável sem perder o caminho da conversa. A skill lê `MVP.md` e `BRAND.md` da raiz do projeto. Quando o projeto é um submódulo Git e esses arquivos não estão nele, ela os procura na raiz do Hub que contém o submódulo. A importação preserva o contexto comercial e de produto no próprio `MVP.md`; somente requisitos de desenvolvimento entram no Specsfy. Quando você escolher uma opção pelo número, a captura registra o texto da opção escolhida, e não somente `1`, `2` ou `3`.

Quando usar

Use antes de criar um conjunto de specs para um produto novo ou quando o MVP ainda não tem uma jornada confirmada. Se `MVP.md` existir, a skill o importa como `specs/milestones/M01.md` e cria backlog e spec somente para os temas que descrevem algo a ser desenvolvido. Ela não cria Inboxes nessa importação e não copia visão, público, negócio, métricas ou posicionamento para `specs/`. Nenhum arquivo existente é substituído.

Um arquivo no projeto tem prioridade. O Hub só entra na busca quando o projeto é um submódulo Git e o arquivo local está ausente. Assim, um `MVP.md` ou `BRAND.md` específico do projeto não é trocado pelo contexto compartilhado.

Cada rodada traz uma pergunta numerada. Abaixo dela, você recebe três ou mais sugestões, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a primeira rodada.

Quando a jornada indicar que o sistema precisa guardar informações, a skill pergunta, uma por vez, sobre cada informação ausente ou ambígua durante a entrevista do backlog correspondente. São no máximo oito perguntas por área; para continuar, você precisa pedir mais e indicar quantas deseja responder. As respostas confirmadas ficam em `.specsfy/DATABASE.md`.

Como descrever a tarefa

Peça: “use o entrevistador de MVP para organizar os marcos do meu sistema de leads”.

Exemplo passo a passo

```
MVP.md → filtro de desenvolvimento → M01 + backlogs e specs Draft
→ milestones sincronizadas
```

O que esperar

O importador cria `M01` e só cria backlog e spec Draft para temas que representam capacidades ou comportamentos a serem desenvolvidos. Visão, público, princípios, modelo de negócio e contexto ficam somente no `MVP.md` e não viram trabalho de desenvolvimento nem Inbox. Antes de perguntar, ele aplica defaults seguros quando encontra um rótulo explícito ou uma formulação inequívoca, registra o campo preenchido, a base usada e a lacuna que ficou aberta. Cada backlog também recebe o trecho que o originou como registro confirmado. A própria skill carrega `$specsfy-02-backlog`, que reaproveita os defaults e só pergunta por lacunas, ambiguidades, contradições ou escolhas reais. Ela chama `$specsfy-data-discovery` quando houver dados ambíguos e retorna à fila até entrevistar todos. Depois sincroniza os milestones e só chama `$specsfy-03-specify` para gerar uma spec Draft para cada backlog. Os campos sem resposta confiável ficam marcados como `Pendente`. A seção 10 de cada spec Draft também registra menus e navegação principal, usando o que o MVP informar ou `Pendente` quando essa parte não existir. A skill não implementa código, não executa tarefas e não passa os gates durante a conversa.

Erros comuns

- pedir tarefas técnicas antes de entrevistar todos os backlogs gerados;
- misturar uma hipótese da conversa com o texto que foi registrado;
- esperar que a importação implemente código ou aprove uma spec com lacunas.

Próximo passo

Leia Milestones para conhecer arquivos e sincronização.

Descobrir o que o sistema precisa guardar com `specsfy-data-discovery`

Use `$specsfy-data-discovery` (`specsfy-data-discovery`) quando uma jornada depender de informações que o sistema precisa lembrar, mostrar para alguém, alterar ou apagar. A conversa usa palavras do seu produto e salva apenas o que você confirmar em `.specsfy/DATABASE.md`.

Quando usar

Use durante o backlog, depois de uma Inbox, durante a descoberta de `MVP.md` ou antes de consolidar uma spec. A skill é útil quando ainda não está claro o que cada pedido, pessoa, atendimento ou outro item do produto precisa guardar.

Como descrever a tarefa

```
Use $specsfy-data-discovery para entender o que nosso sistema precisa lembrar
sobre cada reserva, quem consulta essas informações e quando elas deixam de
ser necessárias.
```

Exemplo passo a passo

```
Inbox sobre reservas → conversa sobre informações a guardar → DATABASE.md
→ backlog refinado
```

O agente pode perguntar o que você precisa lembrar sobre uma reserva, quem pode consultar ou corrigir as informações e quando uma reserva deixa de valer. Você responde com a situação real do seu trabalho, sem precisar nomear partes internas do sistema.

Depois, ele sugere a forma mais adequada para registrar cada informação, como texto curto, data, valor em dinheiro ou escolha entre opções. A sugestão só é salva depois que você confirmar que ela atende ao uso real.

O que esperar

Cada resposta confirmada aparece na seção `Informações a guardar confirmadas` de `.specsfy/DATABASE.md`. Ela informa para que a informação serve, o que deve ser lembrado, o formato sugerido, o que fica ligado, quem usa e quando muda ou sai do sistema.

Erros comuns

- tentar escolher a tecnologia antes de explicar a necessidade do produto;
- tratar uma hipótese como informação confirmada;
- copiar dados reais de clientes, senhas ou chaves para a conversa;
- pular a conversa quando a jornada depende de informações guardadas.

Próximo passo

Volte ao `$specsfy-02-backlog` para concluir o refinamento ou ao `$specsfy-03-specify` para consolidar a fonte normativa da entrega.

Planejar milestones pós-MVP

Use `$specsfy-roadmap-milestone-interviewer` (`specsfy-roadmap-milestone-interviewer`) depois que o MVP estiver aceito. Ele entrevista você sobre o que vem depois e propõe marcos de evolução sem alterar silenciosamente a jornada central já aprovada.

Quando usar

Use quando o MVP já possui objetivo e condição de saída confirmados.

Cada rodada traz uma pergunta numerada. Abaixo dela, você recebe três ou mais sugestões, `Escrever outra resposta`, `Gere outras opções` e `Avançar` desde a primeira rodada.

Como descrever a tarefa

Peça: “planeje o roadmap depois do MVP com milestones”.

Exemplo passo a passo

```
MVP aceito → entrevista de evolução → milestones pós-MVP → specs candidatas
```

O que esperar

Você recebe uma sequência recomendada, dependências e hipóteses que ainda dependem de uso real.

Erros comuns

- usar o roadmap para mudar o núcleo do MVP sem confirmação;
- chamar uma feature isolada de milestone.

Próximo passo

Mudanças no núcleo seguem por `$specsfy-update-spec`. Consulte Milestones para revisar a estrutura resultante.

Manter o mapa de milestones

Use `$specsfy-milestone-governor` (`specsfy-milestone-governor`) para revisar relações entre milestones, specs e backlog. Ele executa a projeção derivada, sugere vínculos ausentes e mantém `specs.md` atualizado sem substituir a escrita humana dos marcos.

Quando usar

Use depois de criar, alterar ou concluir specs vinculadas a milestones.

Se uma relação precisar de confirmação, a skill apresenta exatamente uma pergunta numerada. Ela contém três ou mais sugestões, `Escrever outra resposta` `Gere outras opções` e `Avançar` desde a primeira rodada.

Como descrever a tarefa

Peça: “revise os milestones do projeto e sincronize o mapa”.

Exemplo passo a passo

```
Specs e backlog vinculados → sync → specs.md e progresso dos marcos atualizados
```

O que esperar

A skill mostra relações ausentes e atualiza somente blocos gerados.

Erros comuns

- tratar percentual de tarefas como aceite do marco;
- esperar que a skill escreva uma condição de saída sem confirmação.

Próximo passo

A condição de saída continua dependendo de validação confirmada. Veja Milestones para o comando e o modelo de arquivos.

CLI e TUI do Specsfy

O executável `specsfy` instala e atualiza as skills, mostra o progresso, executa os testes detectados e abre um dashboard no terminal. Instale primeiro o CLI e o framework seguindo o guia de instalação. Este guia assume que `specsfy --version` já responde no terminal e que o bootstrap foi executado no projeto consumidor. Para conduzir a primeira fatia depois do bootstrap, siga o guia do primeiro projeto. Para seleção técnica, automação e reabertura de gates, consulte o uso avançado. Para consultar cada argumento, opção, efeito e formato de saída, use a referência dos comandos.

Os templates de ideia, backlog, spec, tarefas e informações permanentes ficam em `.specsfy/templates/`. Customizações com o mesmo nome ficam em `.specsfy/templates/custom/` e têm precedência sobre os arquivos padrão. Ao criar uma especificação, `specsfy-03-specify` renderiza o template resolvido em `specs/draft/<NNNN>-<slug>/spec.md`. O exemplo demonstra os três atos e as 18 seções para agentes, testes e diagnóstico do CLI, mas não governa uma feature. O cabeçalho renderizado é uma tabela Markdown de duas colunas, `Campo` e `Valor`, com um metadado por linha.

Instalar e gerenciar skills

Para preparar um projeto novo, o comando `install` pode publicar as bases e todos os especialistas detectados em uma única execução:

```
specsfy doctor --project .
specsfy install --project . --detected
```

Quando você já souber quais stacks precisam de orientação especializada, repita `--specialist` para instalar somente o conjunto indicado:

```
specsfy install --project . \
  --specialist specsfy-specialist-laravel \
  --specialist specsfy-specialist-postgres
```

Depois da instalação inicial, os subcomandos de `skills` permitem listar o catálogo, detectar recomendações, adicionar uma skill ou atualizar as versões gerenciadas:

```
specsfy skills list
specsfy skills detect --project .
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-laravel --agent universal --copy --full-depth
specsfy update --project .
```

Atualização automática

Ao abrir `specsfy` ou `specsfy tui` em um terminal interativo, o CLI verifica a versão realmente publicada no registro npm. As tags semânticas de `cli/` servem para registrar a origem da versão quando o GitHub responde. O cache e as configurações globais ficam em `~/.specsfy/cli.json`, com permissão restrita ao usuário e intervalo padrão de 24 horas entre consultas.

Quando existe uma versão publicada superior, o CLI apresenta a versão atual e pergunta se deve atualizar. Recusar abre o dashboard normalmente. Em uma instalação npm, aceitar executa `npm install --global @promovaweb/specsfy@latest`. Em um executável avulso, o CLI baixa `get.specsfy.dev`, valida a versão e substitui o arquivo atual. O processo encerra e a versão nova entra em uso na próxima abertura.

Depois de recusar ou de uma tentativa que falhou, a mesma versão fica adiada até o próximo intervalo configurado. Isso evita a repetição do aviso a cada abertura. O comando explícito `specsfy upgrade` consulta novamente e ignora o adiamento. Se o executável for um symlink, o arquivo apontado é atualizado e o symlink continua disponível no `PATH`.

O arquivo global separa `settings`, incluindo habilitação e intervalo da consulta, de `cache`, que registra horário, versão publicada, tag, commit, ETags, versão adiada e erro recente. Chaves desconhecidas são preservadas. A aplicação continua abrindo quando a rede está indisponível, a resposta é inválida ou a escrita falha, e o aviso fica para a próxima consulta.

Como o monorepo é privado, o catálogo e a proveniência das tags são consultados com `GH_TOKEN`, `GITHUB_TOKEN` ou, na ausência dessas variáveis, com a sessão de `gh auth token`. O token não é copiado para `~/.specsfy/cli.json`.

Em uma instalação global gerenciada pelo npm, `upgrade` atualiza o próprio CLI. Abra o comando novamente para conferir a nova versão:

```
specsfy upgrade
specsfy --version
```

Não confunda os dois fluxos: `specsfy update --project .` atualiza as skills do projeto; `specsfy upgrade` atualiza o programa global. O comando anterior `specsfy skills update` permanece como alias compatível.

Se você instalou o executável Node com `curl -fL get.specsfy.dev`, a oferta automática também atualiza esse arquivo diretamente. O download só substitui o executável depois de confirmar a versão esperada. Uma falha preserva a versão atual e abre a TUI normalmente.

Dashboard e progresso

Ciclo de vida de specs

O CLI move specs pelo ciclo `draft`, `defined`, `planned`, `in-progress`, `review` e `completed`. Ele atualiza o campo `Status` junto com a pasta:

```
specsfy transition 0001-recuperar-senha defined --project .
specsfy transition 0001-recuperar-senha planned --project .
specsfy transition 0001-recuperar-senha in-progress --project .
specsfy migrate --project .
```

Use `migrate` apenas para converter o layout anterior `specs/specs/`.

Quando o Definition Gate aprovado esgota uma captura de `specs/inbox/`, a validação a retira da fila ativa sem apagar nada:

```
specsfy inbox process 2026-07-28-143205-recuperar-senha.md \
  --spec 0001-recuperar-senha --project .
```

A captura vai para `specs/inbox/processed/` com `Status: Processada`, as specs derivadas registradas e os links atualizados. Para recalibrar a execução, registre Effort e sua justificativa diretamente na fonte normativa:

```
specsfy effort 0001-recuperar-senha 7 \
  --reason "Inclui migração e integração externa." --project .
```

Execute sem argumentos dentro do projeto consumidor para abrir a TUI:

```
specsfy
```

`specsfy tui --project PATH` abre explicitamente outro projeto. O dashboard é organizado em seis abas:

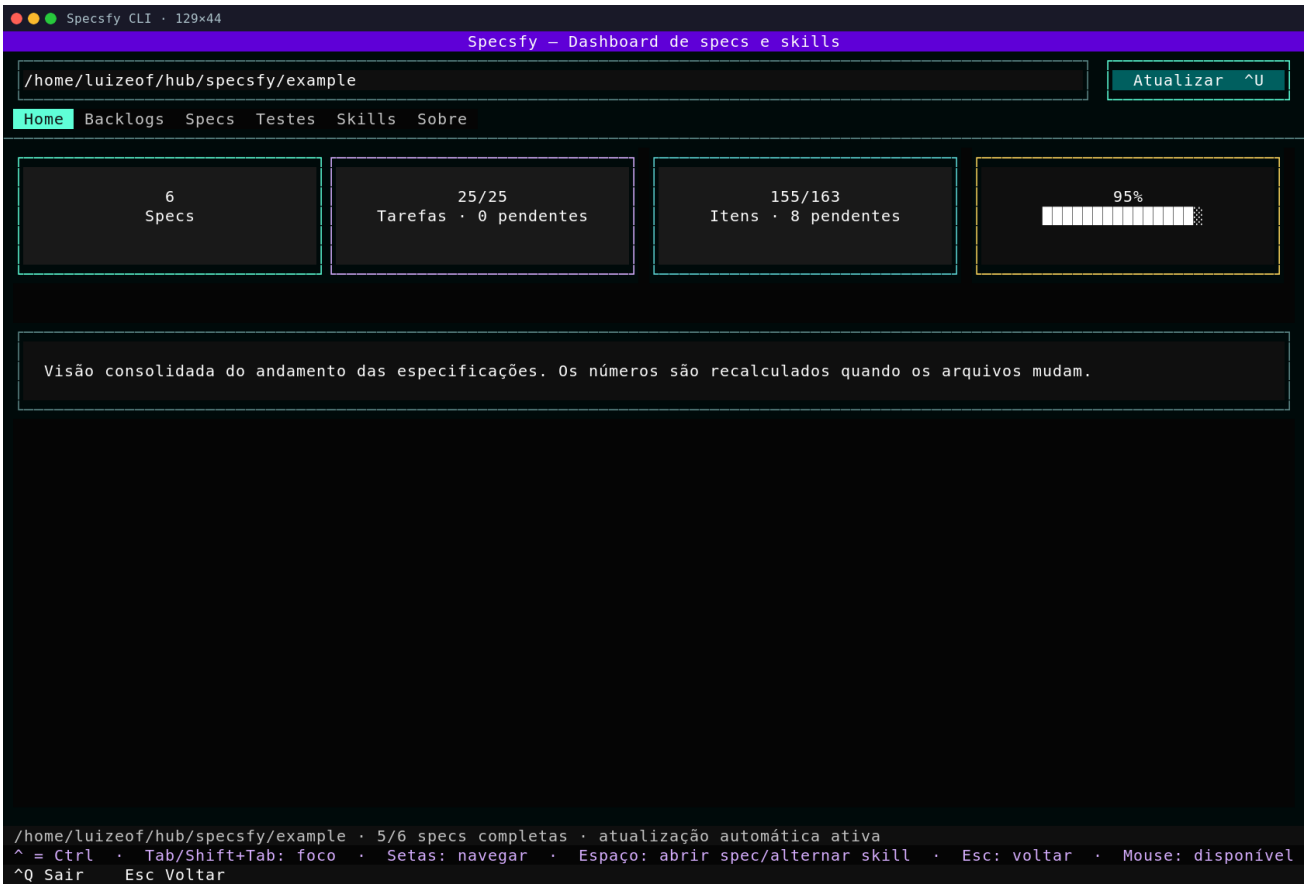
- **Home**, com as estatísticas consolidadas.
- **Backlogs**, com lista navegável na coluna esquerda e preview Markdown formatado na coluna direita.
- **Specs**, com a tabela e o progresso de cada especificação.
- **Testes**, com execução do runner detectado, resumo da última execução e saída detalhada em subabas separadas.
- **Skills**, com catálogo tabular, painel de detalhes e uma prévia explícita das instalações e remoções pendentes.
- **Sobre**, com a versão e a finalidade do CLI.

No dashboard, os cards e as tabelas combinam estes dados para mostrar tanto o estado global quanto a situação de cada spec:

- quantidade total e concluída de specs.
- tarefas `T...` concluídas, pendentes e totais.
- todos os itens de checklist concluídos, pendentes e totais.
- gates aprovados por spec.
- porcentagem e barra de progresso global.
- porcentagem e barra de progresso de cada spec.

Percurso visual

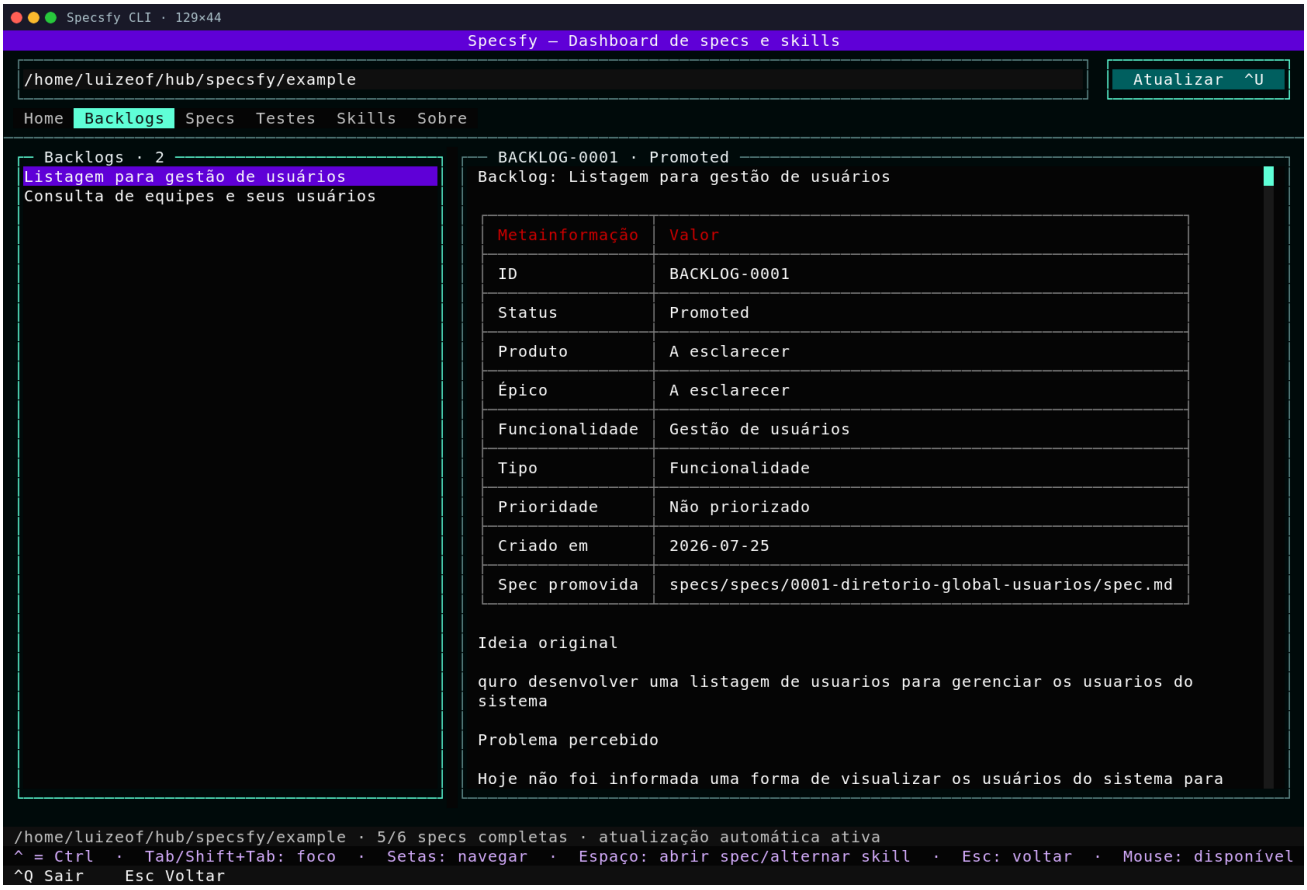
Home: visão consolidada



Dashboard Home

A Home reúne o total de specs, o avanço das tarefas e checklists e a porcentagem global. O diretório selecionado aparece no topo e o rodapé confirma quantas specs estão completas e se a atualização automática está ativa.

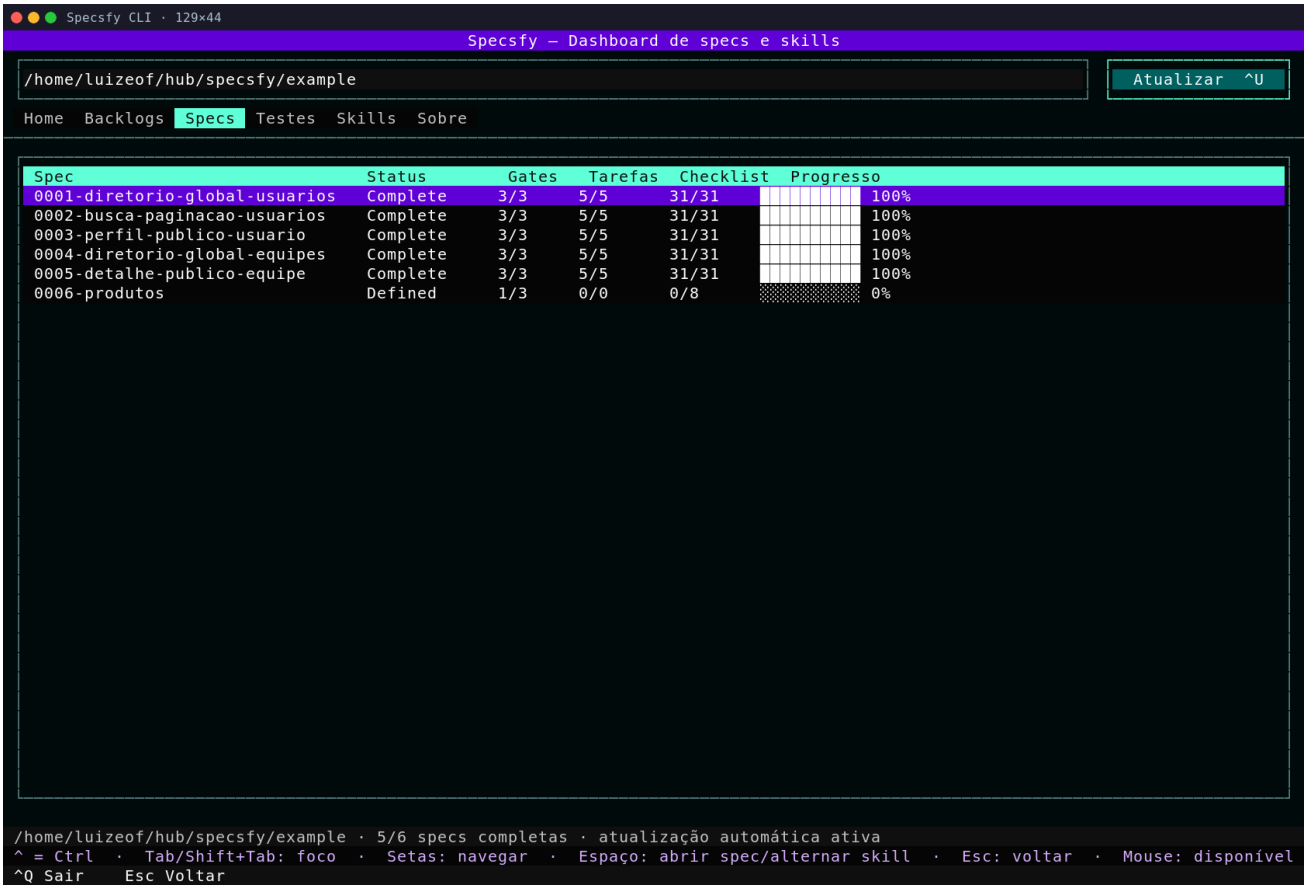
Backlogs: lista e leitura lado a lado



Backlogs

A aba Backlogs mantém a seleção na coluna esquerda e renderiza o Markdown do item na direita. Assim é possível conferir metainformação, status e conteúdo sem abandonar o dashboard.

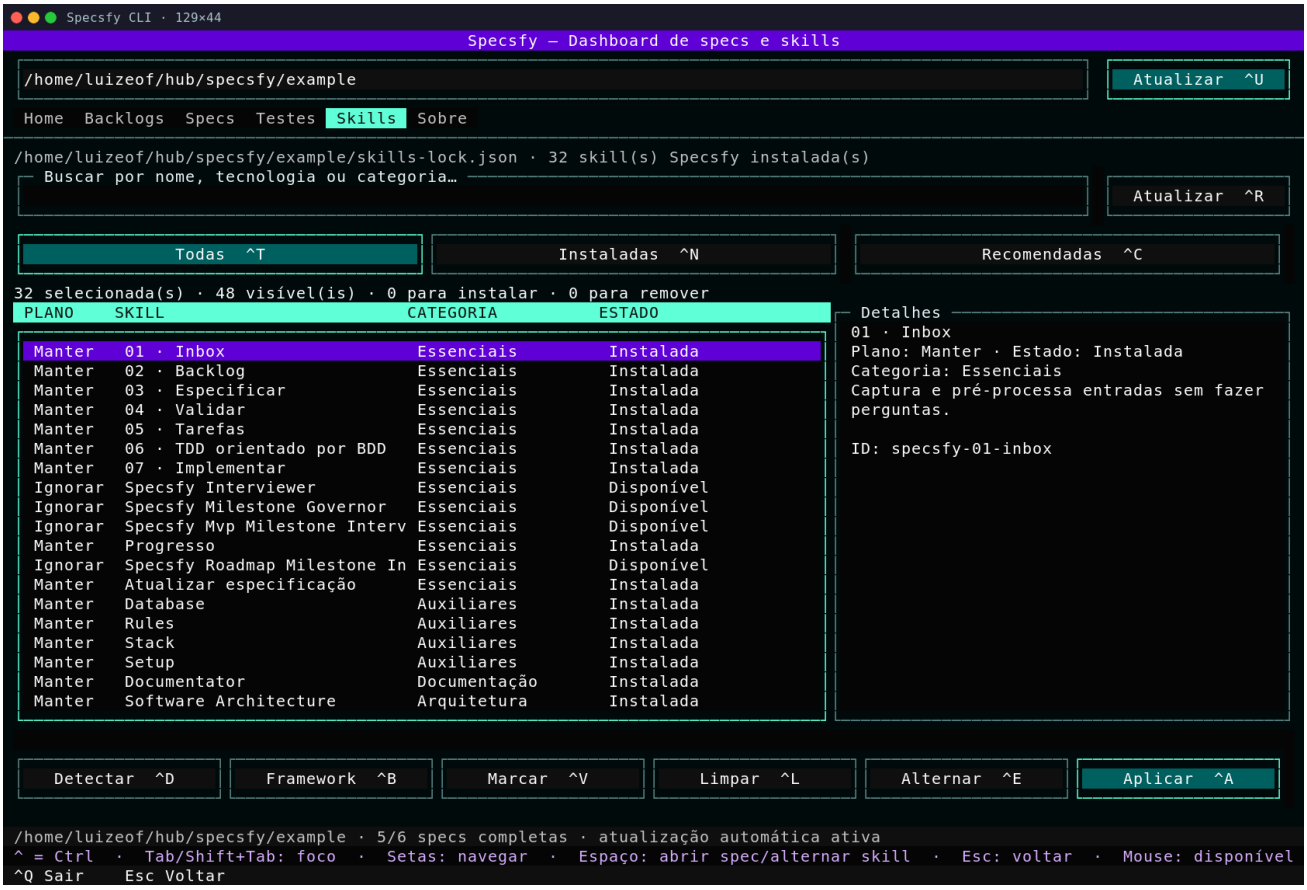
Specs: gates, tarefas e progresso



Specs

A aba Specs compara status, gates, tarefas, checklists e porcentagem por especificação. A linha destacada pode ser aberta com `Espaço` para consultar a spec completa em um modal Markdown. Use `Esc` ou o botão **Fechar Esc** para voltar à lista. Dentro do modal, `Tab` e `Shift+Tab` alternam apenas entre a leitura e o fechamento, sem levar o foco aos controles que estão atrás da visualização.

Skills: planejar antes de aplicar



Skills

A aba Skills combina busca, filtros, plano, categoria e estado com o painel de detalhes da seleção. Os totais acima da tabela mostram instalações e remoções planejadas. Nada muda no projeto até a ação **Aplicar**.

Testes do projeto

Em um projeto Laravel com Pest, `specsfy test` detecta o runner e transmite a saída do teste no mesmo terminal:

```
specsfy test --project .
```

O CLI detecta `artisan` e `pestphp/pest`, chama `php artisan test` diretamente na raiz selecionada, transmite a saída e preserva o exit code do runner. Ele não aceita uma string de shell arbitrária.

Na aba **Testes**, `Executar testes ^X` inicia a mesma execução. A subaba **Resumo** mostra resultado, runner, comando, projeto, duração, exit code e os totais emitidos pelo Pest. A subaba **Testes** mantém a saída completa e rolável. Quando o projeto fornece um relatório Pest estruturado, cada falha é apresentada com nome do teste, arquivo, linha e mensagem.

O progresso lê `specs/<estado>/*/spec.md` e separa os itens vinculados às tarefas dos demais checkboxes. O percentual considera as tarefas com ID `T...` e seus itens canônicos. Rascunhos soltos aparecem como outros checkboxes; itens do ciclo sem tarefa também recebem um aviso com as linhas para revisão. Exemplos dentro de blocos de código são ignorados. Quando uma spec não possui itens de tarefa, o percentual usa os três gates.

A TUI calcula fingerprints dos backlogs, das specs e do `skills-lock.json`. Ela atualiza automaticamente as listas, previews, cards e seleção de skills quando esses arquivos mudam. O intervalo padrão é 0,75 segundo e pode ser configurado por projeto:

```
specsfy config show --project .
specsfy config set --project . --watch-interval 0.5
```

O rodapé apresenta os atalhos disponíveis na aba atual. Use essas combinações para trocar de tela ou aplicar uma ação sem retirar o foco do terminal:

- `Ctrl+Q`: sair.
- `Ctrl+U`: atualizar.
- `Ctrl+D`: detectar recomendações.
- `Ctrl+B`: selecionar todas as skills do framework.
- `Ctrl+E`: alternar o plano da skill destacada.
- `Ctrl+V`: marcar os resultados visíveis.
- `Ctrl+L`: limpar os resultados visíveis.
- `Ctrl+A`: aplicar a seleção.
- `Ctrl+R`: atualizar todas as skills Specsfy instaladas.
- `Ctrl+T`, `Ctrl+N`, `Ctrl+C`: abrir os filtros Todas, Instaladas e Recomendadas.
- `Ctrl+H`, `Ctrl+G`, `Ctrl+S`, `Ctrl+K`, `Ctrl+O`: abrir Home, Backlogs, Specs, Skills e Sobre.
- `Ctrl+J`: abrir Testes.
- `Ctrl+X`: executar os testes do projeto selecionado.

Os atalhos são globais e aparecem nos próprios rótulos dos botões. A interface inteira também aceita:

- `Tab` e `Shift+Tab` para percorrer controles.
- setas para navegar listas e tabelas.
- `Enter` ou `Espaço` para alternar o plano da skill destacada.
- `Esc` para fechar o modal da spec, limpar a busca ou voltar à Home.
- mouse para abas, listas, preview, filtros e botões.

O campo de projeto entra em edição com `Enter` ou com um clique. A confirmação recarrega backlogs, specs e skills a partir do caminho informado.

Ao fechar uma spec, a TUI restaura a seleção e o foco da lista que a abriu. Você pode continuar com as setas, abrir outra linha ou trocar de aba sem precisar pressionar `Tab` até encontrar um controle válido.

A TUI usa a paleta escura oficial do Specsfy. O turquesa identifica o foco e a aba ativa, o violeta marca a linha selecionada e as ações primárias usam fundo petróleo com texto claro. Cada estado também mantém um rótulo visível, por isso continua compreensível quando a paleta configurada no terminal altera as cores.

Na aba Skills, cada linha separa `Plano`, `Skill`, `Categoria` e `Estado`. O plano usa os valores `Instalar`, `Manter`, `Remover` e `Ignorar`. Ele expressa o que acontecerá sem executar a mudança imediatamente. O painel lateral mostra o identificador completo, a descrição, a recomendação e o plano da linha destacada. A alteração só ocorre ao acionar `Aplicar`.

A configuração vive em `<projeto>/.specsfy/config.json`. Valores desconhecidos adicionados pelo usuário são preservados quando o CLI atualiza uma opção.

Para automação, `specsfy progress` pode emitir texto, JSON ou uma sequência de snapshots. Assim, outro processo recebe uma nova leitura somente quando o conteúdo das specs muda:

```
specsfy progress --project .
specsfy progress --project . --json
specsfy progress --project . --watch
specsfy progress --project . --watch --interval 0.5 --json
```

O JSON contém um objeto `summary` e a coleção `specs`. Com `--watch`, um novo snapshot é emitido somente quando o conteúdo das specs muda, o que evita leituras repetidas em uma integração. A instalação das bases e dos especialistas continua disponível na TUI e nos comandos de `skills`.

O leitor mantém compatibilidade com `specs/<NNNN>-<slug>/spec.md` para projetos existentes, mas toda spec nova usa `specs/draft/`. Itens em `specs/backlog/` não entram na porcentagem de entrega.

Justificativa de tamanho

Comandos, dashboard e atalhos operam os mesmos arquivos do projeto. Reuni-los nesta página permite comparar a ação no terminal com o resultado mostrado na TUI, sem duplicar explicações sobre progresso e proteção de alterações locais.

Segurança e reversibilidade

- O destino padrão é `<projeto>/.agents/skills`.
- As regras centrais ficam em `<projeto>/.specsfy/Spec.md`.

- Template e exemplo ficam em `<projeto>/.specsfy/templates/Spec.md` e `<projeto>/.specsfy/examples/Spec.md`.
- Templates personalizados ficam em `<projeto>/.specsfy/templates/custom/`, fora do lock e protegidos até mesmo de operações com `--force`.
- `AGENTS.md` e `CLAUDE.md` recebem somente blocos delimitados e atualizáveis.
- O registro fica em `<projeto>/.specsfy/skills-lock.json`.
- Cada skill gerenciada registra um fingerprint de conteúdo no lock.
- Uma execução repetida sobre a mesma versão não altera arquivos nem o lock.
- Uma skill gerenciada e intacta pode receber uma versão nova sem `--force`.
- Alterações locais fazem a atualização e a remoção serem recusadas. Use `--force` somente depois de confirmar que a versão customizada pode ser descartada.
- Downloads dos catálogos usam checkout temporário, e `skills add` faz a instalação no projeto.
- `specsfy skills remove <nome>` remove somente o nome explícito e preserva conteúdo local divergente.
- O CLI recusa a raiz reconhecida do workspace `promovaweb/specsfy`.

Referência dos comandos do Specsfy CLI

Esta referência descreve a interface pública do `specsfy` 0.8.1. O caminho informado por `--project` deve apontar para a raiz do projeto consumidor. Sem essa opção, o CLI usa o diretório atual. Os comandos que consultam catálogo ou versões privadas usam `GH_TOKEN`, `GITHUB_TOKEN` ou a sessão de `gh auth token`.

Para aprender o percurso pela interface visual, consulte o guia do CLI e da TUI. Para preparar o primeiro projeto, comece pela instalação.

`specsfy`

Abre a TUI no diretório atual. Não recebe argumentos. Antes do dashboard, pode consultar uma versão estável mais recente e pedir autorização para atualizar o pacote global. A recusa ou uma falha de rede preserva a versão instalada.

Exemplos:

```
specsfy
cd aplicativo && specsfy
GH_TOKEN="$TOKEN_SPECSFY" specsfy
GITHUB_TOKEN="$TOKEN_SPECSFY" specsfy
SPECSFY_SPECIALISTS_CATALOG=/tmp/catalog.json specsfy
```

`specsfy install`

Instala as skills base, regras, templates, exemplo e blocos gerenciados no projeto. `--detected` acrescenta os especialistas encontrados pela stack. `--specialist <nome>` pode ser repetido. `--force` permite substituir conteúdo gerenciado que recebeu alteração local. `--json` emite a lista de caminhos alterados como JSON. O comando recusa a raiz do monorepo oficial.

Exemplos:

```
specsfy install --project .
specsfy install --project ./aplicativo --json
specsfy install --project . --detected
specsfy install --project . --specialist specsfy-specialist-laravel
specsfy install --project . --detected --force --json
```

`specsfy doctor`

Verifica Node.js, Git, npm, acesso de leitura e escrita ao projeto e a disponibilidade do `npx`. A resolução usa `npx` encontrado no `PATH` ou o override técnico `SPECSFY_NPX_COMMAND`; a instalação materializa skills por `npx skills add`. `--json` retorna cada item, seu estado e o comando encontrado. Qualquer requisito ausente produz exit code 1.

Exemplos:

```
specsfy doctor
specsfy doctor --project .
specsfy doctor --project ./api
specsfy doctor --json
specsfy doctor --project ./api --json
```

`specsfy update`

Atualiza todas as skills Specsfy registradas no projeto, incluindo skills base, auxiliares, setup, documentador e especialistas. Skills externas e templates customizados permanecem intocados. `--force` substitui conteúdo gerenciado alterado e `--json` informa os caminhos modificados. O comando executa o diagnóstico de instalação antes do download.

Exemplos:

```
specsfy update
specsfy update --project .
specsfy update --project ./api
specsfy update --force
specsfy update --project ./api --force --json
```

`specsfy upgrade`

Consulta novamente a versão publicada no npm e atualiza o próprio CLI pelo pacote `@promovaweb/specsfy@latest`. O npm só é executado quando a versão encontrada é superior à versão atual, o que impede downgrade e ignora um adiamento anterior. `--json` informa se houve atualização e as versões envolvidas. O projeto não é alterado.

Exemplos:

```
specsfy upgrade
specsfy upgrade --json
GH_TOKEN="$TOKEN_SPECSFY" specsfy upgrade
GITHUB_TOKEN="$TOKEN_SPECSFY" specsfy upgrade --json
PATH="$HOME/.npm-global/bin:$PATH" specsfy upgrade
```

`specsfy skills list`

Lista o catálogo do framework e dos especialistas. `--json` entrega objetos estruturados para automação. A leitura consulta o catálogo remoto, exceto quando `SPECSFY_SPECIALISTS_CATALOG` aponta para uma fonte local.

Exemplos:

```
specsfy skills list
specsfy skills list --json
specsfy skills list --json > /tmp/specsfy-skills.json
GH_TOKEN="$TOKEN_SPECSFY" specsfy skills list
SPECSFY_SPECIALISTS_CATALOG=./catalog.json specsfy skills list --json
```

`specsfy skills detect`

Compara os arquivos do projeto com as regras do catálogo e retorna as skills recomendadas. `--project <caminho>` seleciona a raiz e `--json` fornece saída estruturada. O comando apenas consulta, sem instalar arquivos.

Exemplos:

```
specsfy skills detect
specsfy skills detect --project .
specsfy skills detect --project ./api --json
GH_TOKEN="$TOKEN_SPECSFY" specsfy skills detect --project .
SPECSFY_SPECIALISTS_CATALOG=./catalog.json specsfy skills detect --json
```

`specsfy skills install`

Depois de revisar a recomendação e autorizar a instalação, este comando usa `npx skills add` com o repositório oficial e somente os nomes necessários. Ele materializa a skill no projeto atual; não instale o catálogo inteiro por padrão.

Exemplos:

```
specsfy skills install specsfy-specialist-laravel
specsfy skills install specsfy-specialist-postgres --project ./api
specsfy skills install specsfy-specialist-laravel specsfy-specialist-postgres
specsfy skills install specsfy-specialist-react-ui-components --project .
specsfy skills install specsfy-specialist-interface-experience --project .
```

`specsfy skills remove`

Remove somente as skills nomeadas e preserva skills externas presentes no mesmo lock. Os nomes são obrigatórios. Conteúdo local divergente impede a remoção, a menos que `--force` seja informado. A saída lista os caminhos alterados.

Exemplos:

```
specsfy skills remove specsfy-specialist-laravel
specsfy skills remove specsfy-specialist-postgres --project ./api
specsfy skills remove specsfy-specialist-laravel specsfy-specialist-postgres
specsfy skills remove specsfy-specialist-react-ui-components --project .
specsfy skills remove specsfy-specialist-laravel --project . --force
```

`specsfy skills update`

Alias compatível de `specsfy update`. Atualiza todas as skills Specsfy já instaladas. `--project <caminho>` seleciona o projeto. `--force` permite substituir conteúdo gerenciado alterado e `--json` retorna os caminhos. Skills externas e arquivos de `.specsfy/templates/custom/` permanecem intocados.

Exemplos:

```
specsfy skills update
specsfy skills update --project .
specsfy skills update --project ./api
specsfy skills update --force
specsfy skills update --project ./api --force
```

`specsfy transition`

Move uma spec para `draft`, `defined`, `planned`, `in-progress`, `review` ou `completed` e atualiza o campo `Status` no mesmo ato. O identificador e o estado são obrigatórios. `--json` inclui o caminho resultante e o handoff opcional do ClickUpfy. O comando recusa transições não permitidas e sincroniza os milestones após a escrita.

Exemplos:

```
specsfy transition 0001-recuperar-senha defined
specsfy transition 0001-recuperar-senha planned --project .
specsfy transition 0001-recuperar-senha in-progress --json
specsfy transition 0001-recuperar-senha review --project ./api
specsfy transition 0001-recuperar-senha completed --project . --json
```

`specsfy migrate`

Move specs do layout anterior para as pastas do ciclo de vida e alinha o campo `Status`. `--project <caminho>` seleciona a raiz. `--json` retorna a coleção `migrated`. Quando não há fonte legada, a execução não altera arquivos.

Exemplos:

```
specsfy migrate
specsfy migrate --project .
specsfy migrate --project ./api
specsfy migrate --json
specsfy migrate --project ./api --json
```

`specsfy effort`

Registra a pontuação inteira de 1 a 10 e sua justificativa na spec. O identificador, a pontuação e `--reason <texto>` são obrigatórios. `--json` inclui o resultado e o handoff opcional do ClickUpfy. Pontuação fora do intervalo ou spec ambígua é recusada.

Exemplos:

```
specsfy effort 0001-recuperar-senha 3 --reason "Alteração local."
specsfy effort 0001-recuperar-senha 5 --reason "Inclui testes de integração."
specsfy effort 0001-recuperar-senha 7 --reason "Inclui migração." --project .
specsfy effort 0001-recuperar-senha 8 --reason "Depende de API externa." --json
specsfy effort 0001-recuperar-senha 10 \
  --reason "Entrega distribuída." --project ./api --json
```

`specsfy inbox process`

Retira de `specs/inbox/` uma captura cujo pedido já está integralmente coberto por specs com `Definition Gate: Passed` e a guarda em `specs/inbox/processed/`. O comando marca `Status: Processada`, registra as specs derivadas e a data em `Processada em`, atualiza as referências à

captura em `specs/backlog/` e nas specs e preserva o texto original. A captura pode ser indicada pelo nome do arquivo ou pelo caminho. `--spec` é obrigatório e pode ser repetido quando mais de uma spec cobre o pedido. Cada spec precisa estar fora de `draft`, ter `Status` diferente de `Draft` e declarar `Definition Gate: Passed` na primeira tabela de metadados. O `Status` também precisa estar escrito nessa tabela. Tabelas do corpo e exemplos são ignorados. A ausência do gate conta como recusa. Se qualquer spec indicada falhar, nada é movido. O comando nunca sobrescreve uma captura já processada. Decidir se o pedido está inteiro coberto é tarefa da validação; o comando só executa a mudança.

Exemplos:

```
specsfy inbox process \
  2026-07-28-143205-avisar-clientes.md \
  --spec 0007-avisos-de-atraso
specsfy inbox process \
  specs/inbox/2026-07-28-143205-avisar-clientes.md \
  --spec 0007-avisos-de-atraso \
  --project .
specsfy inbox process \
  2026-07-28-143205-avisar-clientes.md \
  --spec 0007-avisos-por-email \
  --spec 0008-avisos-por-sms
specsfy inbox process \
  2026-07-28-143205-avisar-clientes.md \
  --spec 0007-avisos-de-atraso \
  --json
specsfy inbox process \
  2026-07-28-143205-avisar-clientes.md \
  --spec 0007-avisos-de-atraso \
  --project ./api \
  --json
```

`specsfy progress`

Lê specs e calcula estados, gates, Effort, tarefas, checklists e porcentagens. `--json` retorna `summary` e `specs`. `--watch` permanece ativo e só emite um novo snapshot após mudança das fontes. `--interval <segundos>` configura a espera do watch e deve ser maior que zero. O comando não altera o projeto.

Exemplos:

```
specsfy progress
specsfy progress --project .
specsfy progress --project ./api --json
specsfy progress --watch
specsfy progress --project . --watch --interval 0.5 --json
```

`specsfy milestones sync`

Projeta os vínculos declarados nas specs e no backlog para `specs.md` e `specs/milestones/MNN.md`. Somente blocos gerados são substituídos. `--json` retorna o caminho do índice e os totais de cada milestone. Referências inválidas ou metadados ambíguos interrompem a escrita.

Exemplos:

```
specsfy milestones sync
specsfy milestones sync --project .
specsfy milestones sync --project ./api
specsfy milestones sync --json
specsfy milestones sync --project ./api --json
```

`specsfy test`

Detecta um projeto Laravel com Pest, executa `php artisan test`, transmite a saída e devolve o mesmo exit code. `--project <caminho>` seleciona a raiz. O comando não aceita uma string arbitrária de shell e recusa projetos sem runner compatível. Antes de iniciar o processo PHP, exige `.env.testing` com `APP_ENV=testing` e um `DB_DATABASE` ou `DB_URL` explícito, separado do `.env`. Também recusa `RefreshDatabase` e `DatabaseMigrations`. A TUI aplica o mesmo gate; quando a configuração estiver pendente, nenhum teste é iniciado.

Exemplos:

```
specsfy test
specsfy test --project .
specsfy test --project ./api
NO_COLOR=1 specsfy test --project .
specsfy test --project "./aplicativo Laravel"
```

`specsfy tui`

Abre explicitamente o dashboard para o projeto selecionado. `--project` recebe um caminho e usa o diretório atual por padrão. A TUI pode escrever configuração, instalar skills e executar testes somente após uma ação da pessoa. `Ctrl+Q` encerra a interface.

Exemplos:

```
specsfy tui
specsfy tui --project .
specsfy tui --project ./api
GH_TOKEN="$TOKEN_SPECSFY" specsfy tui --project .
SPECSFY_SPECIALISTS_CATALOG=./catalog.json specsfy tui --project .
```

`specsfy config show`

Lê a configuração efetiva de `.specsfy/config.json`. `--project` seleciona a raiz e `--json` fornece saída estruturada. A ausência do arquivo usa os valores padrão e não cria configuração.

Exemplos:

```
specsfy config show
specsfy config show --project .
specsfy config show --project ./api
specsfy config show --json
specsfy config show --project ./api --json
```

`specsfy config set`

Grava o intervalo de atualização da TUI em `.specsfy/config.json`. `--watch-interval <segundos>` é obrigatório e aceita número positivo. `--project` seleciona a raiz e `--json` retorna a configuração resultante. Chaves desconhecidas existentes são preservadas.

Exemplos:

```
specsfy config set --watch-interval 0.5
specsfy config set --project . --watch-interval 0.75
specsfy config set --project ./api --watch-interval 1
specsfy config set --watch-interval 2 --json
specsfy config set --project ./api --watch-interval 5 --json
```

Justificativa de tamanho

A referência mantém comandos, parâmetros, efeitos, recusas e exemplos no mesmo arquivo para que a cobertura automatizada compare a gramática executável com uma única fonte. Separar cada comando impediria essa conferência direta e espalharia opções compartilhadas por várias páginas.

Confirmação e diagnóstico

Depois de qualquer escrita, confirme o resultado com uma leitura compatível: `skills list`, `progress`, `milestones sync --json` ou `config show --json`. Falhas do CLI são impressas em `stderr` com o prefixo `erro:` e retornam exit code diferente de zero. `--help` mostra a gramática instalada e `--version` confirma qual release está sendo executada.

Informações permanentes do projeto

O Specsfy separa a descrição durável do sistema das especificações de cada mudança. O arquivo `PROJECT.md` explica a finalidade da aplicação, enquanto cinco documentos em `.specsfy/` registram a stack, as instruções confirmadas, a persistência observada no código, os pacotes instalados e o perfil de interação do setup. As regras macro de interface ficam em `DESIGNSYSTEM.MD`, na raiz do projeto, e seguem um ciclo próprio com a skill especialista de design system.

Execute `$specsfy-setup` depois de instalar o framework. Depois disso, o framework a executa obrigatoriamente antes de iniciar cada skill, inclusive em transições automáticas, para verificar e reconciliar os contextos iniciais e os blocos reservados de agentes. Na mesma conversa, a raiz confirmada é reaproveitada sem perguntar de novo. A skill detecta Laravel, Next.js e Astro pelos manifests e sugere o modelo correspondente. `$specsfy-documentator` acrescenta e atualiza `PACKAGES.md`. Juntas, as skills mantêm esta estrutura:

```
<projeto>/
├── PROJECT.md
├── DESIGNSYSTEM.MD
├── docs/
│   └── packages/
│       ├── README.md
│       └── <vendor>-<nome>.md
└── .specsfy/
    ├── STACK.md
    ├── RULES.md
    ├── DATABASE.md
    ├── PACKAGES.md
    ├── USER-PROFILE.md
    └── SPECKIT.md      # somente quando GitHub Spec Kit for detectado
```

O setup também reserva blocos delimitados para as diretrizes do Specsfy em `AGENTS.md` e `CLAUDE.md`. Conteúdo fora desses blocos pertence ao usuário e é preservado. A referência publicável das diretrizes vive em `specsfy-setup`.

ARQUIVO	CONTEÚDO	SKILL MANTENEDORA
`PROJECT.md`	finalidade e capacidades	`$specsfy-setup` cria o modelo
`.specsfy/STACK.md`	stack e evidências	`$specsfy-aux-stack`
`.specsfy/RULES.md`	regras explícitas confirmadas	`$specsfy-aux-rules`
`.specsfy/DATABASE.md`	persistência e relações	`$specsfy-aux-database`

`.specsfy/PACKAGES.md`	pacotes npm e Composer com finalidade	`$specsfy-documentator`
`docs/packages/README.md`	índice dos pacotes Composer diretos e links para fichas de uso	`$specsfy-specialist-laravel-package-manager`
`docs/packages/<vendor>-<nome>.md`	instalação, configuração, uso local e testes do pacote	`$specsfy-specialist-laravel-package-manager`
`.specsfy/USER-PROFILE.md`	nível de conhecimento, respostas confirmadas e fontes do setup	`$specsfy-setup`
`.specsfy/SPECKIT.md`	constituição e fontes preservadas do GitHub Spec Kit	`$specsfy-setup`
`DESIGNSYSTEM.MD`	defaults comuns de interface, CRUD, dashboards, estados e exceções	`$specsfy-setup` cria; `$specsfy-specialist-design-system` mantém

Os modelos ficam em `.specsfy/templates/Project.md`, `Stack.md`, `Rules.md`, `Database.md` e `UserProfile.md`, junto dos demais templates do framework. Para personalizar um deles, mantenha o mesmo nome em `.specsfy/templates/custom/`; essa cópia tem precedência e não é alterada pelo CLI.

Durante o setup, `.specsfy/USER-PROFILE.md` guarda o nível de conhecimento confirmado e as respostas já fornecidas. O agente consulta esse arquivo, a conversa e as fontes do projeto antes de perguntar. Assuntos já respondidos ficam fora da próxima rodada; perguntas técnicas recebem mais contexto para iniciantes e podem usar versões, arquitetura e integrações diretamente para pessoas experientes.

O setup também verifica como você administra cada informação principal do produto. Quando a jornada precisa de cadastro e manutenção, ele confirma as ações de criar, consultar, editar e apagar. Informações somente de leitura, históricos imutáveis e conteúdos mantidos por integrações não recebem um CRUD sem necessidade. Se o código e os documentos não deixarem essa necessidade clara, o setup pergunta antes de registrar a cobertura.

Para cada tela de uso recorrente, o setup identifica o caminho pelos menus do sistema. Ele registra item, destino, permissão e comportamento responsivo em `INTERFACE.md`. Rotas técnicas e etapas abertas apenas por redirecionamento podem ficar fora do menu. Quando o destino ou a presença do link estiverem abertos, você escolhe em uma pergunta numerada antes da continuação.

Ao terminar a preparação dos contextos e especialistas, o setup executa `$specsfy-documentator`. Essa etapa reconstrói a documentação técnica de todo o sistema existente em `docs/` e atualiza `.specsfy/PACKAGES.md`, mesmo quando a execução não começou por uma alteração recente no código.

Use `$specsfy-data-discovery` antes de implementar quando ainda faltar explicar o que o produto precisa guardar, quem consulta cada informação e quando ela deixa de ser necessária. A skill registra as respostas confirmadas em uma seção própria de `DATABASE.md`, separada do que o código detectar depois.

Projeto existente com GitHub Spec Kit

O setup reconhece o GitHub Spec Kit pela constituição em `.specify/memory/constitution.md`. Quando ela existe, a skill lê a constituição e todos os arquivos regulares dentro de `specs/`, inclusive specs, planos, tarefas, contratos e anexos. O resultado aparece em `.specsfy/SPECKIT.md` como uma lista de caminhos, títulos, tipos e fingerprints SHA-256.

Abra as fontes listadas antes de trabalhar na feature correspondente. A constituição continua governando o projeto e os artefatos do GitHub Spec Kit permanecem nos caminhos originais. O setup não escreve, move, converte ou remove arquivos de `.specify/` e `specs/`.

Você pode acrescentar notas próprias fora do bloco `specsfy:speckit` em `.specsfy/SPECKIT.md`. Uma nova execução atualiza somente o bloco delimitado. Se a constituição divergir de `.specsfy/RULES.md`, preserve os dois textos e resolva a divergência antes de alterar a feature.

Execute `$specsfy-aux-stack` após alterar frameworks, runtimes, ferramentas estruturais ou persistência. Execute `$specsfy-aux-database` sempre que criar ou alterar banco, schema, tabela, coleção, model persistente, campo, relação, índice ou migration. Use `$specsfy-aux-rules` para formular e acrescentar uma regra confirmada sem duplicar ou apagar regras anteriores.

Monitoramento durante mudanças

O monitor é executado pelas skills no início e no fim de uma mudança. Ele lê os arquivos staged, unstaged e untracked do Git para descobrir qual documento precisa ser revisto, mas não permanece como daemon em segundo plano:

```
node .agents/skills/specsfy-setup/scripts/monitor_context.mjs \
  --project . --check
```

SINAL OBSERVADO	OBRIGAÇÃO
manifest, lockfile ou configuração	`$specsfy-aux-stack` revisa `STACK.md`
manifest ou lockfile npm/Composer	`$specsfy-documentator` reconstrói `PACKAGES.md` e `docs/`
schema, model ou migration	`$specsfy-aux-database` revisa `DATABASE.md`
código da aplicação	revisar `PROJECT.md`

aplicação ou persistência	`$specsfy-documentator` reconstrói `docs/` e `PACKAGES.md`
instrução ou convenção	`$specsfy-aux-rules` revisa `RULES.md`

`PENDING` impede a conclusão da tarefa e do Delivery Gate. Quando uma mudança de aplicação não altera história, finalidade, capacidades ou limites, registre essa avaliação na evidência da tarefa e execute novamente com `--acknowledge-project-no-change`. Para uma revisão de regras sem regra nova, use `--acknowledge-rules-no-change` somente depois de registrar a justificativa. Veja a topologia e o `--check` no guia de documentação técnica do sistema.

Preservação e atualização

- O setup cria um arquivo de informações somente quando ele ainda não existe. Isso inclui `DESIGNSYSTEM.MD` e `.specsfy/USER-PROFILE.md`: os templates iniciais apresentam padrões comuns de interface, CRUD, dashboard e registro da conversa. Os arquivos continuam humanos e qualquer conteúdo existente é preservado.
- As auxiliares atualizam apenas blocos detectados delimitados e preservam seções humanas.
- Uma nova varredura nunca autoriza remover silenciosamente uma definição humana.
- `DATABASE.md` usa tabelas Markdown para facilitar mapeamento e comparação.
- `PACKAGES.md` deriva de manifests, lockfiles e metadados locais, preservando texto humano fora do bloco gerado.
- Valores sensíveis não são lidos nem registrados. Cite somente nomes seguros de variáveis e caminhos das fontes.

O estado implementado é comprovado pelas fontes do próprio projeto. Os manifests e lockfiles mostram a stack, enquanto schemas e migrations mostram a persistência. Os documentos resumem essas evidências e as definições humanas que precisam permanecer entre mudanças. Eles não substituem a `spec.md`, não registram gates e nunca devem copiar segredos, valores de `.env` ou registros de produção.

Design system de interface

`DESIGNSYSTEM.MD` é o documento do projeto consumidor que guarda as regras macro de interface. Ele orienta cada tela nova junto das regras do domínio, permissões, estados e telas relacionadas.

Onde fica

Depois de `specsfy install`, o template fica disponível em `.specsfy/templates/DESIGNSYSTEM.MD`. Ao executar o setup, o Specsfy cria `DESIGNSYSTEM.MD` na raiz somente se ele ainda não existir. O arquivo é humano, pertence ao projeto e passa a ser mantido pela skill especialista, não pelo lock do CLI.

Instale a skill quando for criar ou revisar uma interface:

```
specsfy skills install specsfy-specialist-design-system
```

Se a entrega já usa a coordenação completa de interface, instale:

```
specsfy skills install specsfy-specialist-interface-experience
```

O catálogo resolve o especialista de design system junto das dependências de UX e UI.

Como funciona

Antes de projetar uma tela, o agente lê `DESIGNSYSTEM.MD`. Se você não informar uma direção visual, os defaults do arquivo são aplicados e registrados como direção padrão da entrega. Se você pedir algo diferente, a alteração entra em `Exceções da entrega` com alcance definido. Uma exceção de tela não muda o produto inteiro.

`INTERFACE.md` continua registrando componentes, blocos, telas, props, eventos, estados e consumidores locais. Ele aponta para o design system e não repete as regras macro.

Defaults para CRUD

TELA	COMPOSIÇÃO PADRÃO
Lista	`PageHeader` compartilhado + resumo útil + `DataGrid` em largura total
Detalhe	`PageHeader` compartilhado + `DetailLists`
Criar	`PageHeader` compartilhado + seções de formulário em duas colunas responsivas

Editar	`PageHeader` compartilhado + seções de formulário em duas colunas responsivas

O `PageHeader` é um componente único, configurado por props ou configuração para título, descrição, breadcrumb e ações. Lista, detalhe, criação e edição reutilizam essa mesma implementação; não duplicam sua marcação.

A lista sempre mostra o `ID` em uma coluna visível do `DataGrid`. Cada linha inteira é o link para o detalhe, com suporte a mouse e teclado. Os botões `Editar` e `Apagar` ficam na própria linha como ações independentes e não disparam a navegação do detalhe. A permissão e a confirmação da ação destrutiva também fazem parte do contrato.

Formulários usam labels visíveis acima dos campos. Quando um campo falha, ele fica com estado visual vermelho e mostra a mensagem abaixo do campo. O foco vai para o primeiro erro, os outros valores permanecem preenchidos e o resumo de erros oferece links quando há várias falhas.

Toda tela também exibe um `Breadcrumb` no shell global. A trilha mantém o nome da equipe ativa visível entre o contexto inicial e o módulo, seguida do título da tela atual. Em Laravel, o padrão deve reaproveitar o `Breadcrumb` ou `Breadcrumbs` que já existir no layout, suas rotas e sua tipagem, sem criar uma segunda implementação.

As linhas do `DataGrid` abrem o detalhe inteiro por clique ou teclado. Botões, checkboxes e menus da linha permanecem ações independentes, acima do link de detalhe.

Criar e editar são divididos em seções. Cada seção tem contexto à esquerda e um painel de campos à direita; os campos relacionados usam duas colunas em telas largas e uma coluna no mobile. Campos longos, uploads e erros podem ocupar toda a largura.

Defaults para dashboards e blocos

Dashboards começam com `PageHeader`, período ou escopo, filtros, indicadores `KPI` contextualizados, uma visualização principal e uma lista ou `DataGrid` para investigação. Use primitives do `shadcn/ui` para controles fundamentais e blocos gratuitos do ReUI para composições comuns de CRUD e dashboard quando forem compatíveis com a stack. Registre origem, estados, acessibilidade e consumidores em `INTERFACE.md`.

Cenários a cobrir

Uma entrega de interface registra o recorte aplicável dos cenários do template:

- lista com registros e lista vazia;
- detalhe com status, ações e relações relevantes;
- criação e edição válidas;
- criação ou edição com erro de campo;
- ausência de permissão;

- falha de carregamento;
- alteração não salva;
- ação destrutiva e seu resultado.

Para cada cenário, descreva pré-condição, ação, resposta, estado visual, foco, mensagem e próximo passo. A personalidade do produto vem dos dados, da linguagem, dos tokens, do ritmo e da hierarquia, não de uma decoração genérica.

Revisão visual durante o desenvolvimento

Toda tarefa que altera interface passa por uma revisão visual durante o desenvolvimento, mesmo sem pedido específico. Confira bordas, espaçamentos, margens, padding e tipografia, incluindo família, peso, tamanho, altura de linha, espaçamento entre letras, hierarquia e quebra de texto. Confira também alinhamento, largura, overflow, foco, zoom e conteúdo curto ou longo nos viewports e estados aplicáveis.

Registre no item `VISUAL` da tarefa o método usado, os viewports, os estados, os ajustes feitos e o resultado. Quando a tarefa não tem interface, registre `Não aplicável` e o motivo concreto.

Quando não usar

Não use `DESIGNSYSTEM.MD` para listar props ou arquivos de um componente. Não use `INTERFACE.md` como substituto das regras macro. Não crie uma tela CRUD sem consultar a fonte, sem tratar estados ou sem registrar uma exceção ao default.

Conferência

Antes de concluir uma tela, confira:

- `DESIGNSYSTEM.MD` existe e foi lido.
- A tela exibe `Breadcrumb` com equipe, módulo e tela atual; em Laravel, o componente existente foi reaproveitado.
- A composição corresponde à superfície CRUD.
- Loading, vazio, erro, sucesso, permissão e não salvo foram cobertos quando aplicáveis.
- A direção padrão ou a exceção tem alcance registrado.
- `INTERFACE.md` recebeu os componentes e telas locais.

Documentação técnica do sistema

`$specsfy-documentator` reconstrói a visão técnica de uma aplicação em `<projeto>/docs/` e o inventário de dependências em `<projeto>/.specsfy/PACKAGES.md`. Esses arquivos explicam o código e os pacotes do projeto consumidor e não se confundem com a documentação oficial da metodologia Specsfy.

O que esta documentação explica

A documentação reconstruída responde como a aplicação está montada no momento da varredura. Ela não tenta substituir a spec de uma entrega e não cria uma segunda lista de tarefas. Use a spec para entender por que uma mudança existe, quais requisitos ela atende e quais testes a comprovam. Use o diretório docs/ do projeto consumidor para entender a aplicação como um todo antes de alterar um módulo.

Essa separação evita dois problemas comuns. O primeiro é usar uma documentação de arquitetura para aprovar comportamento que nunca foi definido. O segundo é copiar toda a arquitetura dentro de cada spec e deixá-la envelhecer depois da próxima mudança. A spec aponta apenas o contexto necessário para sua entrega, enquanto a documentação técnica volta a ser construída a partir do código, manifests, schemas, migrations e testes atuais.

O que é gerado e o que é preservado

A skill reconstrói somente blocos identificados pelo marcador specsfy:documentator. Texto humano escrito fora desses blocos permanece no arquivo. Isso permite acrescentar uma explicação de negócio, uma observação de suporte ou uma escolha editorial sem que a próxima varredura a apague.

O conteúdo gerado descreve somente o que as fontes locais sustentam. Quando a skill encontra uma convenção, ela pode registrá-la como observação encontrada, mas não a apresenta como escolha humana confirmada. Escolhas explícitas do projeto continuam em PROJECT.md, RULES.md, na spec aplicável ou em um ADR, conforme o alcance da escolha.

Execute `$specsfy-documentator` livremente para documentar um sistema legado ou atualizar sua visão técnica. Depois de cada tarefa de código concluída por `$specsfy-07-implement`, a transição para o documentador é obrigatória. A implementação só continua quando `docs/` representar o código atual.

A skill lê o código completo e as fontes que descrevem a aplicação. Isso inclui os manifests, as migrations, as rotas e os testes, além das informações permanentes do projeto. Cada execução reconstrói blocos delimitados nos seguintes arquivos e preserva o texto humano externo:

ARQUIVO NO CONSUMIDOR	CONTEÚDO
`docs/README.md`	portal e ordem de leitura

`docs/architecture.md`	componentes, dependências e UML Mermaid
`docs/application.md`	módulos e implementações observadas
`docs/database.md`	entidades, campos, relações e `erDiagram`
`docs/flows.md`	rotas, `flowchart` e `sequenceDiagram`
`docs/testing.md`	runners, comandos, inventário e resumo
`docs/frontend.md`	views, páginas, componentes, React e Tailwind
`docs/packages.md`	runtime, framework, nativos, integrados e terceiros
`docs/packages/README.md`	índice Laravel de pacotes Composer diretos e fichas de uso
`docs/packages/<vendor>-<nome>.md`	instalação, configuração, uso local e testes de um pacote
`docs/integrations.md`	serviços externos e nomes de configuração
`docs/decisions.md`	escolhas explícitas e suas fontes
`.specsfy/PACKAGES.md`	pacotes npm e Composer, versão, finalidade e fonte

Como ler cada documento

O portal docs/README.md oferece uma ordem de leitura. architecture.md mostra a visão dos componentes e suas dependências. application.md aproxima essa visão do código, apontando módulos e implementações encontradas. database.md separa entidades, campos, relações e a origem dessas informações. flows.md mostra a passagem entre rotas, handlers, serviços e integrações para que um fluxo possa ser conferido de ponta a ponta.

testing.md não promete cobertura que o repositório não mostra. Ele identifica os runners, os comandos disponíveis, arquivos de teste e o resumo observado. frontend.md só aparece com as superfícies que o projeto contém, como views, páginas, componentes, React ou Tailwind. integrations.md lista serviços e nomes seguros de configuração, jamais o valor de uma variável. decisions.md conserva escolhas que possuem fonte explícita, distinguindo uma escolha registrada de uma inferência do código.

PACKAGES.md tem outro papel: tornar as dependências auditáveis. Ele lista o gerenciador, escopo, nome, versão, finalidade e fonte encontrada localmente. A finalidade pode estar ausente nos metadados. Nessa situação, o documento declara a ausência em vez de completar a coluna por suposição.

Procedimento depois de uma mudança

Depois de uma tarefa de código, a implementação chama o documentador. Você também pode executá-lo para iniciar a documentação de um sistema legado ou reconciliar uma alteração feita fora do fluxo:

```
node .agents/skills/specsfy-documentator/scripts/build_documentation.mjs \
  --project .
```

Leia primeiro o portal, o arquivo diretamente relacionado à alteração e o resultado das seções geradas. Quando alguma relação, pacote ou integração não corresponder ao que o código demonstra, corrija a fonte que permite a inferência ou registre a limitação. Não edite o bloco gerado para alterar uma conclusão que a próxima execução voltará a produzir.

Em seguida, execute o modo de conferência. Ele não escreve arquivos: compara a projeção atual com os blocos publicados.

```
node .agents/skills/specsfy-documentator/scripts/build_documentation.mjs \
  --project . --check
```

O resultado aprovado mostra que a documentação corresponde às fontes atuais. Um resultado pendente indica que a aplicação, a persistência ou as dependências mudaram sem nova reconstrução, ou que a saída publicada foi modificada.

Situações que impedem a conclusão

O monitor de contexto e o modo check participam do Delivery Gate. A entrega não termina quando existe uma das condições abaixo:

- código da aplicação alterado sem reconstruir docs/;
- migration, schema ou modelo persistente alterado sem atualizar o mapa de dados e os documentos reconstruídos;
- manifest ou lockfile alterado sem atualizar PACKAGES.md;
- documentação gerada que não corresponde ao estado atual das fontes;
- inclusão de segredo, valor de ambiente ou dado de produção em um documento.

Quando a aplicação mudou mas sua finalidade, capacidades e limites não mudaram, registre essa avaliação na tarefa e use o reconhecimento permitido pelo monitor. Esse reconhecimento não dispensa a reconstrução de documentação técnica nem serve para ocultar uma mudança documental real.

Em Laravel, o inventário acompanha a requisição pelas rotas, controllers e services, relaciona Eloquent e migrations e registra os testes Pest ou PHPUnit. Em projetos Node, Next.js, React ou Astro, a documentação mostra páginas, endpoints, componentes e scripts, além do runner observado no repositório.

Cada pacote recebe a versão e a referência do repositório no GitHub. Quando essa origem não puder ser confirmada localmente, a documentação publica uma busca identificada como tal, em vez de inventar uma URL.

O arquivo `.specsfy/PACKAGES.md` percorre todos os manifests npm e Composer do projeto e inclui também as dependências transitivas registradas em `package-lock.json` e `composer.lock`. A finalidade vem da descrição presente no lockfile, no pacote instalado ou no catálogo conhecido do documentador. Quando nenhuma dessas fontes existir, o arquivo declara que a finalidade não foi descrita nos metadados locais.

Depois da reconstrução, a própria skill executa o modo `--check`. O comando compara os blocos gerados com o estado atual e falha quando `docs/` está desatualizado:

```
node .agents/skills/specsfy-documentator/scripts/build_documentation.mjs \
  --project . --check
```

O monitor do setup também retorna `PENDING` quando o código da aplicação, a persistência ou as dependências mudaram sem uma nova reconstrução de `docs/`. Mudanças em manifests ou lockfiles exigem ainda a atualização de `.specsfy/PACKAGES.md`. Esse estado impede a conclusão da tarefa e do Delivery Gate.

O código, os testes, os manifests, os schemas e as migrations comprovam o estado implementado. A spec governa o comportamento da mudança, enquanto `PROJECT.md` e `.specsfy/` preservam informações válidas para o sistema inteiro. Os arquivos em `<projeto>/docs/` podem ser reconstruídos dessas fontes e não devem copiar segredos, valores de ambiente, registros de produção ou código integral.

Atualizar uma especificação

Quando uma entrega já definida precisar mudar, use `specsfy-update-spec`. A skill incorpora a nova instrução na mesma `spec.md` e reconcilia as partes afetadas sem exigir que você escolha atos ou gates manualmente.

Descreva o que mudou

Durante ou depois da implementação, identifique a `spec.md` e diga diretamente o que deve mudar. Você pode adicionar, remover, corrigir ou mudar uma regra. A skill usará esse arquivo para encontrar os requisitos, testes e tarefas afetados:

```
Use $specsfy-update-spec em
specs/<estado>/0001-pagina-boas-vindas/spec.md:
esqueci de pedir que o nome tenha no máximo 80 caracteres.
```

Se a spec já estiver clara na conversa, você pode usar uma instrução natural sem mencionar atos ou gates:

```
Quero adicionar uma regra à especificação atual.
Remova o comportamento de visitante.
Corrija o que acontece quando o nome estiver vazio.
Mude esta entrega para exigir autenticação.
```

A skill preserva a instrução, compara-a com a spec existente e informa primeiro o resultado prático. Você não precisa escolher manualmente qual gate ou etapa reabrir.

O que acontece

TIPO DE MUDANÇA	TRATAMENTO
correção interna sem efeito observável	mantém a spec
esclarecimento sem novo significado	ajusta o texto e preserva os gates
mudança de comportamento ou aceite	atualiza a spec e reabre o Ato I
mudança de solução, tarefas ou testes	atualiza a spec e reabre o Ato II
capacidade independente	encaminha para backlog ou nova spec
definição material ausente	pergunta e retoma depois da resposta

Quando a mudança pertence à spec atual, a skill atualiza o arquivo, encaminha as etapas invalidadas e retorna à implementação somente depois das novas provas:

```
mudança posterior → update-spec → validate ou tasks → TDD/BDD → implement
                                ↘ backlog, se faltar uma definição
```

Testes e tarefas que continuam válidos são preservados. Provas dependentes do comportamento anterior voltam a ficar pendentes. A implementação só é retomada depois de existir novo RED válido e os gates necessários voltarem a `Passed`.

Resultado esperado

- a nova instrução está incorporada à única `spec.md` normativa.
- IDs e definições ainda válidos foram preservados.
- os gates afetados foram reabertos.
- validação, tarefas e testes foram reconciliados.
- a etapa que recebeu o pedido foi retomada automaticamente.

Limites

- não cria `change-request.md`, `plan.md`, `tasks.md` ou outra fonte paralela.
- não transforma uma capacidade independente em aumento silencioso de escopo.
- não inventa uma definição material.
- não altera código antes de atualizar a spec e preparar novo RED.
- handoffs automáticos não autorizam deploy, publicação ou ação destrutiva.

Quando usar outra skill

- criar a primeira versão de uma spec.
- capturar uma ideia ainda superficial.
- editar código quando a spec continua correta.

Depois da atualização, consulte `specsfy-progress` para conferir o estado geral. A spec atualizada permanece como única fonte normativa do projeto.

Skills especialistas

As skills base conduzem a metodologia. As `specsfy-specialist-*` acrescentam orientação para a tecnologia encontrada no projeto, como Laravel, Astro, Next.js, Postgres ou Redis. Assim, você instala somente o conhecimento técnico usado pela aplicação.

Na interface, cada uma usa o padrão `Specsfy - Especialista - Nome`, como `Specsfy - Especialista - Laravel`. O identificador de comando continua `specsfy-specialist-laravel`.

Para criar ou alterar interfaces, `specsfy-specialist-interface-experience` coordena a entrega e carrega `specsfy-specialist-design-system` antes dos especialistas de UX, UI e componentes. O design system mantém as regras macro, defaults e cenários CRUD; a experiência examina a stack e o sistema atual, organiza as perguntas sobre telas e menus e garante uma fase específica de interface nas tarefas.

Detectar e instalar

O comando `detect` lê o projeto e mostra recomendações sem instalar arquivos. O catálogo local só deve ser alterado depois que você revisar os nomes retornados:

```
specsfy skills detect
```

Depois da revisão, instale apenas os especialistas usados pela aplicação. A instalação sempre usa `npx skills add` e registra cada escolha no `skills-lock.json`, permitindo conferir depois quais arquivos são gerenciados:

```
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-laravel \
  --skill specsfy-specialist-postgres \
  --skill specsfy-specialist-redis \
  --agent universal --copy --full-depth
```

As skills base podem sugerir um especialista. Quando ele já estiver instalado, a transição é anunciada e continua na mesma conversa. Quando estiver ausente, o agente informa o especialista, a finalidade e as dependências, avisa que usará `npx skills add` e pede autorização específica para instalar. A transição entre skills nunca instala o catálogo inteiro automaticamente.

Instalação no setup

Ao iniciar o setup, você autoriza o Specsfy a detectar e instalar os especialistas diretamente ligados à stack encontrada, além de um núcleo comum para modelar dados, domínio, arquitetura e interfaces ReUI. Ele mostra os itens e usa `npx skills add` no projeto atual. Tecnologias sem

sinal no código não entram na instalação.

STACK ENCONTRADA	ESPECIALISTA
Laravel	Laravel e gestor de pacotes Laravel
Supabase, PostgreSQL ou Redis	Supabase, PostgreSQL ou Redis correspondente
React, Astro, Next.js ou TypeScript	especialista correspondente
Tailwind CSS ou shadcn/ui	Tailwind CSS ou shadcn/ui correspondente
ReUI solicitado para React e Tailwind	ReUI, React, Tailwind CSS e shadcn/ui
Docker, Swarm ou Ansible	especialista de plataforma e versionamento correspondente
OpenAPI, OpenTelemetry, Prometheus ou CI/CD	API, observabilidade ou entrega correspondente

Instalação manual

Você pode instalar qualquer especialista do catálogo fora do setup. Essa opção serve para uma necessidade pontual que a stack ainda não revela, como revisão, acessibilidade ou pesquisa técnica:

```
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-web-accessibility --agent universal --copy --full-
depth
```

O setup não remove especialistas instalados manualmente. Eles permanecem no projeto e podem ser usados quando a entrega precisar deles.

Para usar os componentes gratuitos do ReUI, instale o especialista e suas dependências resolvidas pelo catálogo:

```
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-reui --agent universal --copy --full-depth
```

Ele prepara o registry ReUI para React 19, Tailwind CSS v4 e shadcn/ui, atende Laravel com Inertia e Vite e preserva o padrão de outros frameworks React. Nos CRUDs e dashboards compatíveis, ReUI é a base para consulta, filtros, formulários, indicadores, ações contextuais, anexos e mensagens de retorno. A skill consulta o catálogo gratuito antes de permitir a criação de componente manual equivalente.

Gestor de pacotes Laravel

`specsfy-specialist-laravel-package-manager` recebe a URL de um repositório GitHub, lê seu contrato Composer e sua documentação e compara o pacote com o que já existe no projeto. Quando a solicitação autoriza a instalação, executa `composer require` na raiz Laravel e confere o lockfile atualizado.

O resultado documental fica em `docs/packages/`:

- `README.md` é o índice dos pacotes Composer diretos, com versão, finalidade e link para cada ficha;
- `<vendor>-<nome>.md` explica instalação, configuração, uso local, testes e fontes consultadas;
- `.specsfy/PACKAGES.md` continua sendo a relação completa, incluindo dependências transitivas.

O especialista não repete uma instalação quando o pacote já aparece no manifest, no lockfile ou em `vendor/`. Se o repositório não expuser um pacote Composer compatível com Laravel, ele registra a lacuna e não altera os arquivos.

Contrato de interfaces React

Em Laravel com React, shadcn/ui e ReUI trabalham juntos: shadcn/ui fornece as primitives e ReUI fornece as composições gratuitas. Toda tela é construída por componentes React. A rota ou página coordena dados e compõe blocos; não reúne grade, formulário, filtros, diálogos, painel lateral e cartões reutilizáveis no mesmo arquivo.

O setup cria `INTERFACE.md` e `DESIGNSYSTEM.MD` na raiz do projeto quando eles estão ausentes. O primeiro registra tokens, registries, telas e todos os blocos criados ou reaproveitados. O arquivo `DESIGNSYSTEM.MD` guarda as regras macro, defaults, padrões de CRUD e dashboard, estados e exceções com alcance; a skill especialista o mantém. Para cada bloco, informe arquivo, origem, finalidade, props e eventos, estados, acessibilidade, consumidores e como reaproveitar ou estender. A seção 10 de cada spec com interface usa a mesma relação para declarar os blocos React e os componentes shadcn/ui e ReUI escolhidos. As tarefas de interface atualizam o mapa antes de serem concluídas.

Antes de desenvolver uma tela React, o planejamento e a implementação carregam `$specsfy-specialist-react-ui-components`. A skill procura primeiro os blocos existentes, orienta o reaproveitamento ou a adaptação e mantém `INTERFACE.md` alinhado ao código. Se ela ainda não estiver instalada, o fluxo retorna ao setup e instala o especialista detectado antes de escrever JSX ou TSX.

Nas superfícies CRUD, a regra padrão é `PageHeader` e `DataGrid` para listas, com a linha inteira abrindo o detalhe por clique ou teclado; `PageHeader` e `DetailLists` para detalhes; e `PageHeader` com seções de formulário em duas colunas responsivas para criar e editar. Botões, checkboxes e menus internos ficam independentes da navegação da linha. Erros de campo ficam vermelhos e recebem mensagem abaixo do campo. Toda tela também exibe `Breadcrumb` com a equipe ativa, o módulo e o título atual. Em Laravel, o padrão reaproveita o `Breadcrumb` ou `Breadcrumbs` existente no layout.

Catálogo por domínio

DOMÍNIO	ESPECIALISTAS
backend e dados	Laravel, Supabase, Postgres, Redis e APIs web
frontend	React, Astro, Next.js, TypeScript e Tailwind CSS
interface	design system, experiência de interface, shadcn/ui, UI, UX e acessibilidade web
plataforma	versionamento, Docker, Docker Swarm, Ansible e engenharia de entrega
qualidade	segurança, observabilidade e performance
design técnico	arquitetura e modelagem de domínio
engenharia	code review, debugging, prototipação, pesquisa e conflitos Git

Cada especialista explica o fluxo de trabalho e as validações próprias da sua tecnologia. Uma mudança em Eloquent pode combinar Laravel e Postgres, por exemplo, mas uma alteração isolada em uma página Astro não precisa carregar orientações de todo o catálogo.

Guia de cada especialista

Esta é a referência de entrada para todos os especialistas distribuídos pelo Specsfy. Use o nome de comando na instalação. O setup instala o núcleo e os especialistas detectados; os demais permanecem disponíveis para instalação manual quando a entrega pedir aquele domínio.

ESPECIALISTA	QUANDO USAR
Laravel	Domínio PHP, HTTP, filas, autorização, persistência e testes Laravel.
Gestor de pacotes Laravel	Pacotes Composer recebidos por URL GitHub, instalação e fichas em `docs/packages/`.

Supabase	Postgres gerenciado, Auth, RLS, Storage, Realtime e Edge Functions.
PostgreSQL	Modelagem relacional, SQL, índices, migrations e operação do banco.
Redis	Cache, filas, locks, rate limiting e estruturas de dados em memória.
Modelagem de Dados	Entidades, relações, ciclos de vida e contratos persistentes.
Modelagem de Domínio	Linguagem do negócio, invariantes e limites entre módulos.
Arquitetura de Software	Módulos, dependências, atributos de qualidade e fronteiras técnicas.
Design System	Regras macro, defaults, estados, padrões CRUD e exceções com alcance.
React	Componentes, estado, efeitos, acessibilidade e testes de interface.
Tailwind CSS	Tokens, responsividade, variantes e CSS do design system.
shadcn/ui	Primitives, registry, tema e padrões acessíveis de aplicação.
ReUI	Composições gratuitas para interfaces React e Tailwind, especialmente CRUDs.
Componentes React	Catálogos copiáveis que complementam UI design em projetos React.
UI Design	Layouts, dashboards, tabelas, formulários, estados e hierarquia visual.
UX Design	Fluxos, arquitetura de informação, pesquisa e validação de uso.
Experiência de Interface	Leitura do sistema atual, telas, formulários e tarefas de interface.
Acessibilidade Web	WCAG, semântica, teclado, foco e tecnologias assistivas.
Astro	Ilhas, conteúdo, renderização, integrações e performance em Astro.
Next.js	App Router, fronteiras server/client, cache, dados e deploy.
TypeScript	Tipos, strictness, módulos e contratos de API.
Docker	Imagens, Compose, segurança, builds e ambiente local.
Deploy	Orquestra `SEMVER`, inventário, conexões, chaves públicas, Compose local, stack Swarm e Ansible.
Versionamento	Arquivo `SEMVER`, incrementos e alinhamento entre artefato, tag e deploy.
Debian Server	APT, systemd, SSH, nftables, sysctl, storage e Docker Engine no host.
Docker Swarm	Stacks, serviços, redes, secrets, rollout e recuperação.
Ansible	Playbooks, roles, Vault e automação de infraestrutura.
APIs Web	Contratos HTTP, erros, paginação, idempotência e evolução de API.

Observabilidade	Logs, métricas, traces, SLOs e alertas acionáveis.
Engenharia de Entrega	CI/CD, artefatos, promoções, migrations, rollout e rollback.
Segurança de Aplicações	Ameaças, controles, segredos e verificações de segurança.
Performance	Orçamentos, medição, profiling, carga e regressões.
Code Review	Revisão técnica guiada pelo contrato e pelos testes.
Debugging	Diagnóstico de bugs, falhas intermitentes e regressões.
Prototipação	Protótipos descartáveis para dúvidas técnicas ou de interface.
Pesquisa Técnica	Pesquisa rastreável em fontes primárias e artefatos observáveis.
Resolução de Conflitos Git	Conflitos Git preservando intenção e integração do código.
Gitflow	Branches, merges e ciclo de release quando o projeto adotá-lo.

Cada linha corresponde ao comando `specsfy-specialist-<nome-em-inglês>` do catálogo. A listagem completa de nomes, tags, dependências e sinais de detecção está em `specialists/catalog.json` no repositório do Specsfy.

Relação com as bases

- `specsfy-02-backlog` identifica a necessidade.
- `specsfy-03-specify` registra requisitos e atributos de qualidade.
- `specsfy-04-validate` seleciona lentes de revisão.
- `specsfy-05-tasks` usa checklists técnicos para decompor trabalho.
- `specsfy-06-tdd-bdd` preserva RED/GREEN.
- `specsfy-07-implement` executa de acordo com a stack.
- `specsfy-update-spec` revisa requisitos e atributos de qualidade afetados por uma mudança posterior.
- `specsfy-progress` identifica a orientação técnica aplicável e executa a transição automática.

A presença de um especialista não aprova gates. O Plan Gate ainda exige um RED válido, e o Delivery Gate depende dos testes e das evidências registradas na spec.

Como funciona o Deploy da aplicação

Deploy não começa no servidor. Ele começa quando a entrega recebe um número que acompanha o código, a imagem e o manifesto até o ambiente de destino.

O Specsfy usa um arquivo chamado `SEMVER` na raiz do projeto para manter essa identidade. As skills de Docker, Docker Swarm, Ansible e engenharia de entrega consultam o mesmo valor. Assim, você consegue responder qual código foi compilado, qual imagem chegou ao registry e qual versão está ativa no cluster.

Este capítulo acompanha o percurso completo. A aplicação sai de uma mudança validada, vira uma imagem imutável, passa pela preparação do servidor e chega ao Docker Swarm com um caminho conhecido de rollback.

O arquivo SEMVER

`SEMVER` contém uma única linha com a versão preparada. Você pode abrir o arquivo antes do deploy e comparar esse valor com a imagem que será publicada:

```
1.4.0
```

O formato segue `MAJOR.MINOR.PATCH`. Cada parte comunica um tipo diferente de mudança e impede que uma correção pareça uma quebra de compatibilidade:

- aumente `PATCH` para uma correção compatível, como `1.4.0` para `1.4.1`.
- aumente `MINOR` para uma capacidade nova e compatível, como `1.4.1` para `1.5.0`.
- aumente `MAJOR` quando a interface pública deixa de ser compatível, como `1.5.0` para `2.0.0`.

A skill `$specsfy-specialist-versioning` analisa o alcance da entrega, propõe o incremento e explica a consequência. Durante a preparação autorizada, ela atualiza o arquivo. Publicar imagem, criar tag, abrir uma GitHub Release ou alterar um servidor continua dependendo de autorização explícita.

Se o projeto ainda não tiver o arquivo, informe a versão inicial:

```
node .agents/skills/specsfy-specialist-versioning/scripts/semver.mjs \
  init --initial 1.0.0 --project .
```

O mesmo utilitário permite consultar a versão antes de qualquer alteração e incrementá-la depois que o alcance da entrega estiver confirmado:

```
node .agents/skills/specsfy-specialist-versioning/scripts/semver.mjs \
  current --project .

node .agents/skills/specsfy-specialist-versioning/scripts/semver.mjs \
  bump patch --project .
```

O caminho pode mudar conforme a biblioteca de skills do agente. O contrato permanece o mesmo: `SEMVER` fica na raiz do projeto do usuário.

Uma identidade, vários pontos de conferência

A versão preparada precisa aparecer nos artefatos que representam a entrega:

PONTO	EXEMPLO PARA A VERSÃO `1.4.0`
Fonte local	`SEMVER` contém `1.4.0`
Changelog	seção da versão `1.4.0`
Imagem	`registry.example/app:1.4.0`
Anotação OCI	`org.opencontainers.image.version=1.4.0`
Manifesto	serviço aponta para `1.4.0` ou para seu digest
Git	tag `v1.4.0`
GitHub	release `v1.4.0`

O commit também pode gerar uma tag de imagem, como `git-a1b2c3d`. Depois da publicação, o digest `sha256:...` identifica o conteúdo exato promovido. O número explica a evolução para pessoas. O digest garante que todos os nodes baixem os mesmos bytes.

```
flowchart TD
  P[Pedido em linguagem natural] --> D[specsfy-specialist-deploy]
  D --> I[Inventário dos servidores]
  I --> C[Teste das conexões]
  C --> K[Chaves públicas para deploy]
  D --> S[SEMVER]
  D --> V[Secrets no Ansible Vault]
  S --> T[Testes da aplicação]
  T --> B[Build único]
  B --> R[Registry e digest]
  R --> G[Tag Git e GitHub Release]
  K --> A[Ansible]
  V --> A
  R --> A
  A --> W[Managers e workers do Docker Swarm]
  W --> O[Stack, réplicas e healthchecks]
```

Como as skills trabalham juntas

Você não precisa chamar cada especialista manualmente durante um fluxo de deploy. O catálogo declara as relações necessárias.

Use `$specsfy-specialist-deploy` como entrada única para um release ou deploy completo. Ela gera a base operacional com:

```
node .agents/skills/specsfy-specialist-deploy/scripts/scaffold.mjs \
  --project . \
  --image registry.example/equipe/aplicacao
```

O comando lê `SEMVER` na raiz e cria o `Dockerfile` com Open Swoole, o `compose.yaml` para desenvolvimento, o `stack.yaml` para produção e a pasta `ansible/`. Aplicações Laravel executam Octane com `--server=swoole`. O playbook cria o usuário `deploy`, instala o Docker Engine, ativa o Docker Swarm e publica a stack pelo manager. Se algum desses arquivos já existir, o gerador encerra sem sobrescrever o conteúdo.

Por padrão, a stack também cria o serviço `cloudflared`. Ele participa da mesma rede overlay da aplicação e encaminha o hostname público para `http://app:8000`. A conexão parte do container para o Cloudflare, por isso o serviço Laravel não publica a porta `8000` no host de produção.

```
flowchart LR
  C[Cloudflare] <-->|conexões de saída| T[cloudflared]
  T -->|rede overlay: http://app:8000| A[Laravel Octane]
  S[Docker Secret] -->|arquivo montado| T
```

O token do túnel entra no Ansible Vault como `vault_cloudflare_tunnel_token`. O playbook o converte no Docker Secret externo `cloudflare_tunnel_token`, montado em arquivo no container. A stack usa `--token-file`; o valor não aparece no YAML, nos argumentos do processo ou no repositório.

Se você escolher Nginx, Traefik, HAProxy ou outro ingresso, diga isso ao pedir o deploy. A geração usa `--proxy external`, não cria `cloudflared` e deixa a configuração pública para a alternativa escolhida. O Cloudflare Tunnel só é omitido depois dessa escolha explícita.

Opções do scaffold

OPÇÃO	OBRIGATÓRIA	PADRÃO	RESULTADO
`--project <caminho>`	sim	nenhum	define a raiz que contém `SEMVER`
`--image <nome>`	sim	nenhum	define a imagem sem tag nem digest
`--proxy <tipo>`	não	`cloudflare-tunnel`	aceita o padrão ou `external`

O gerador grava os arquivos no projeto indicado e encerra sem alterações se algum destino já existir. Ele também recusa `SEMVER` inválido, imagem com tag ou digest e valor de proxy desconhecido. Estes exemplos mostram as formas de uso da interface:

```
node scripts/scaffold.mjs --project . --image registry.example/app

node scripts/scaffold.mjs --project /srv/minha-app --image ghcr.io/equipe/app

node scripts/scaffold.mjs --project . --image registry.example/app \
  --proxy cloudflare-tunnel

node scripts/scaffold.mjs --project . --image registry.example/app \
  --proxy external

node scripts/scaffold.mjs --project ../api --image ghcr.io/equipe/api \
  --proxy external
```

Depois da geração, a skill pergunta quais senhas, tokens e chaves a aplicação consome e registra os nomes em `ansible/vault-fields.txt`. O utilitário solicita cada valor com entrada oculta. A senha do Vault vem da configuração externa ou do terminal, conforme o modo escolhido:

```
./deploy secrets
```

Cada resposta é adicionada automaticamente ao `vault.yml` como variável criptografada. O playbook transforma essas variáveis em Docker Secrets. A stack mantém somente os nomes externos e nunca recebe os valores protegidos. Se você repetir o comando, ele preserva os campos existentes e pergunta apenas pelos que ainda faltam.

O próximo capítulo, Servidores, conexões e comandos de deploy, mostra a árvore criada no projeto, o cadastro de máquinas, a conta `deploy`, a sincronização das chaves públicas e os comandos curtos para outro painel do Herdr.

Versionamento prepara a entrega

`$specsfy-specialist-versioning` lê o estado atual, propõe `patch`, `minor` ou `major` e atualiza `SEMVER`. Ela compara o valor com a publicação anterior e recusa reutilizar um número já publicado.

Docker produz a imagem

`$specsfy-specialist-docker` carrega a versão quando a imagem deixa de servir apenas ao desenvolvimento local. O build recebe a tag SemVer, a tag do commit e as anotações OCI. A mesma imagem serve HTTP, filas, scheduler ou outros processos, com comandos próprios.

A imagem é compilada uma vez. Staging e produção recebem o mesmo digest. Não há um novo build por ambiente.

Ansible prepara os hosts

`$specsfy-specialist-ansible` confere `SEMVER`, imagem e manifestos no preflight. Depois, prepara o Debian, autentica no registry, instala os arquivos versionados e valida a stack antes da escrita no cluster.

Use `--check --diff` primeiro. Limite os hosts com `--limit` e aplique lotes com `serial` quando a alteração alcançar várias máquinas. Segredos ficam no Ansible Vault ou em um cofre externo, nunca no repositório.

Docker Swarm aplica a versão

`$specsfy-specialist-docker-swarm` recebe uma imagem já publicada. O arquivo de stack aponta para a tag conferida ou, de preferência, para o digest. O Swarm distribui as réplicas, respeita healthchecks e executa `update_config`.

O retorno de `docker stack deploy` apenas confirma que o comando foi aceito. A entrega termina quando as réplicas convergem e os sinais da aplicação permanecem saudáveis.

Sequência de uma entrega

1. Prepare a versão

Revise a mudança concluída, escolha o incremento e atualize o changelog junto com `SEMVER`. Confirme que a versão é superior à tag mais recente.

```
VERSION=$(tr -d '\n' < SEMVER)
git tag --list "v${VERSION}"
```

Esse comando consulta o Git sem criar a tag. Uma linha vazia confirma que a versão ainda não foi usada no repositório local.

2. Teste e compile uma vez

Rode a suíte do projeto antes do build. Passe a versão e o commit como metadados da imagem:

```
VERSION=$(tr -d '\n' < SEMVER)
COMMIT=$(git rev-parse --short=12 HEAD)

docker build \
  --label "org.opencontainers.image.version=${VERSION}" \
  --label "org.opencontainers.image.revision=${COMMIT}" \
  --tag "registry.example/app:${VERSION}" \
  --tag "registry.example/app:git-${COMMIT}" \
  .
```

Antes do push, confirme que a tag SemVer ainda não existe no registry. Uma tag publicada é imutável. Se for preciso corrigir o conteúdo, prepare um novo `PATCH`.

3. Publique o artefato antes da tag Git

Com autorização explícita, envie a imagem e confira o digest retornado. Só depois crie `v1.4.0` no Git e a GitHub Release correspondente.

Essa ordem evita uma release pública sem imagem disponível. Ela também mantém um caminho simples para repetir o deploy sem recompilar.

4. Valide os hosts e a stack

O preflight do Ansible reúne as condições que precisam estar visíveis antes da escrita no cluster. A leitura deve confirmar:

- acesso aos hosts e ao registry.
- versão do Docker Engine e papel de cada node.
- presença das redes externas necessárias.
- existência dos Docker Secrets esperados.
- serviço `cloudflared` na rede overlay, salvo quando outro proxy foi pedido.
- token do túnel entregue por Docker Secret, sem valor literal na stack.
- versão do manifesto igual ao conteúdo de `SEMVER`.
- acesso ao digest publicado.
- sintaxe aceita por `docker stack config`.

Check mode prepara essa leitura sem implantar a stack. Depois da autorização, o playbook copia os manifestos e chama o deploy na ordem das dependências.

5. Observe a convergência

Depois de `docker stack deploy`, acompanhe cada serviço até que a quantidade de réplicas ativas corresponda à quantidade desejada:

```
docker stack services minha-stack
docker service ps minha-stack_web
docker service logs --since 10m minha-stack_web
```

Compare réplicas desejadas e ativas, healthchecks, taxa de erro, latência e consumo de recursos. Defina um prazo máximo para a convergência. Se o serviço não estabilizar, interrompa a promoção e use a versão anterior já publicada.

Migrations durante o rollout

Em um rolling update, versões antiga e nova podem executar ao mesmo tempo. A alteração do banco precisa aceitar essa convivência.

Use o percurso expand/contract para separar a mudança compatível da remoção do formato anterior:

1. adicione coluna, tabela ou índice sem remover o contrato antigo.
2. publique código que entende os dois formatos.
3. migre ou preencha os dados necessários.

4. confirme que nenhum processo antigo depende do formato anterior.
5. remova o contrato antigo em outra versão.

Execute a migration por uma única tarefa ou réplica. Se dois containers tentarem alterar o mesmo schema, o rollout pode parar no meio da atualização. Os serviços HTTP e os workers não devem disputar a mesma migration.

Rollback faz parte da preparação

Antes do deploy, registre a versão anterior e seu digest. O rollback de código volta a stack para essa imagem. O rollback de dados é outro procedimento: algumas migrations não podem ser desfeitas sem perda.

Um plano mínimo deixa registrada a imagem que voltará ao ar, a relação com o banco e os sinais usados para interromper o rollout. Ele informa:

- versão e digest anteriores.
- comando para restaurar a referência da stack.
- compatibilidade do banco com a versão anterior.
- sinais que interrompem o rollout.
- pessoa responsável por acompanhar a recuperação.

No Swarm, `rollback_config` define paralelismo, intervalo, ordem e resposta a falhas. Teste esse percurso em um ambiente representativo antes da primeira execução em produção.

Exemplo completo

Considere uma aplicação na versão `1.3.2`. A entrega adiciona um endpoint sem quebrar clientes existentes. A skill propõe `minor`, então `SEMVER` passa para `1.4.0`.

O pipeline testa o commit, cria `app:1.4.0` e `app:git-a1b2c3d`, publica ambas e registra o digest. A tag Git `v1.4.0` é criada depois dessa confirmação. O Ansible valida os nodes e instala o manifesto. O Swarm promove o digest com `start-first`, observa a convergência e mantém `1.3.2` disponível para rollback.

Ao final, `SEMVER`, changelog, imagem, manifesto, tag Git e GitHub Release contam a mesma história.

Antes de considerar o deploy concluído

- `SEMVER` contém a versão preparada.
- A versão é superior à última publicação.
- Testes e build partiram do mesmo commit.
- A imagem SemVer e a imagem do commit apontam para o mesmo digest.

- O manifesto usa a imagem conferida.
- A tag Git foi criada depois da publicação da imagem.
- O preflight do Ansible passou no ambiente correto.
- A stack convergiu e os sinais permaneceram saudáveis.
- A versão anterior continua disponível para rollback.
- O resultado foi registrado na especificação e no changelog.

Para conhecer cada especialista, continue em Skills especialistas. Para pipelines e automações adicionais, consulte Uso avançado.

Justificativa de tamanho

Este capítulo mantém no mesmo percurso a versão, o artefato, a preparação dos hosts e a operação do cluster. A leitura conjunta permite conferir a passagem do `SEMVER` ao runtime sem separar etapas que dependem umas das outras.

Servidores, conexões e comandos de deploy

A `specsfy-specialist-deploy` prepara os arquivos do projeto e mantém a lista de máquinas que receberão a aplicação. Este capítulo detalha essa camada operacional: onde cada arquivo fica, como uma máquina entra no ambiente, o que o teste de conexão comprova e quais comandos curtos você pode copiar no Herdr.

Onde a automação fica no seu projeto

A skill trabalha a partir da raiz que você confirmou. Os arquivos permanecem junto do sistema para que o Git mostre quando a infraestrutura muda com o código:

```
meu-projeto/
├── deploy
├── SEMVER
├── Dockerfile
├── compose.yaml
├── stack.yaml
├── docker/
│   └── entrypoint.sh
└── ansible/
    ├── inventory.yml
    ├── inventory.example.yml
    ├── check-hosts.mjs
    ├── create-vault.sh
    ├── vault.mjs
    ├── deploy.yml
    ├── keys.yml
    ├── sync-keys.yml
    ├── secrets.yml
    ├── vault-fields.txt
    ├── group_vars/
    │   ├── all.yml
    │   └── all/
    │       └── vault.yml
    └── templates/
        └── stack.yaml.j2
```

No modo padrão, `vault-fields.txt` inclui `vault_cloudflare_tunnel_token`. Você informa esse valor pelo prompt oculto de `./deploy secrets`, junto dos demais secrets ainda ausentes. O template da stack monta o Docker Secret no serviço `cloudflared`, que compartilha a rede

overlay com `app`.

O arquivo `deploy` oferece nomes curtos para ações que você pode copiar em outro painel do Herdr. Você não precisa iniciar o fluxo por esses comandos. Um pedido como "faça o deploy desta aplicação" continua sendo a entrada normal, e a IA executa as ferramentas conforme o estado encontrado.

O que muda quando você pede outro deploy

Na primeira preparação, o gerador cria a base quando nenhum dos destinos existe. Nas chamadas seguintes, a IA não executa o gerador sobre os mesmos arquivos. Ela lê a aplicação outra vez, compara a infraestrutura atual com as necessidades do código e modifica somente o que precisa acompanhar a entrega.

O `Dockerfile` recebe uma revisão própria. Em Laravel, a skill confere a versão do PHP, as extensões exigidas pelo Composer, o pacote `laravel/octane`, a extensão `openswoole`, o build dos assets, o entrypoint, as permissões, a porta, o healthcheck e o comando `octane:start --server=swoole`. Uma dependência nova que exija outra extensão PHP deve aparecer nessa análise antes do build.

O mesmo exame alcança `compose.yaml`, `stack.yaml`, `docker/` e `ansible/`. Quando um arquivo possui personalizações do projeto, a skill preserva esses trechos e apresenta a comparação dos arquivos alterados antes de substituir uma estrutura sem marcações gerenciadas. Repetir o pedido não significa gerar tudo novamente. Significa reconciliar o estado existente com a aplicação que será publicada.

Essa reconciliação também confere o ingresso. Sem uma escolha diferente, a IA mantém Cloudflare Tunnel como serviço da stack. Quando você pedir outro proxy, ela remove ou deixa de gerar os componentes do túnel e prepara a alternativa solicitada sem misturar tokens entre os dois caminhos.

Como os servidores entram no inventário

Antes da primeira conexão, a orquestradora aciona o especialista de Debian e pergunta quais máquinas fazem parte do ambiente. A conversa trata um servidor por rodada. Para cada máquina, você confirma:

CAMPO	O QUE REPRESENTA	EXEMPLO
alias	nome estável dentro do Ansible	`app01`
endereço	IP ou hostname alcançável	`203.0.113.10`
porta	porta usada pelo SSH	`22`
usuário inicial	conta que já consegue entrar no host	`root`
papel	função do node no Docker Swarm	`manager`

Essas respostas formam `ansible/inventory.yml`. O arquivo de exemplo ensina o formato, mas nunca substitui o inventário real durante uma conexão.

Quando você disser "adicione um novo servidor", a skill lê os hosts atuais e pergunta somente pelos dados da nova máquina. Depois, ela acrescenta o host ao grupo escolhido, testa o SSH, sincroniza as chaves públicas, prepara o usuário `deploy`, instala o Docker e integra o node ao Swarm. Os servidores anteriores permanecem no arquivo e não são renomeados ou removidos nessa operação.

A conta do servidor e a conta do container

O host e o container usam nomes diferentes porque cumprem funções diferentes. No Debian, `deploy` é a conta operacional que recebe as chaves SSH, pertence ao grupo `docker` e administra os diretórios sob `/opt/apps`. Dentro da imagem, `app` continua sendo a conta sem privilégios que executa o Laravel Octane.

```
máquina local ──SSH──> deploy@servidor ──Docker──> container: usuário app
```

O acesso por senha da conta `deploy` permanece desabilitado. O grupo `docker` concede controle amplo sobre o host, por isso a automação não adiciona outras contas a esse grupo sem uma necessidade confirmada.

Teste das conexões

Antes de alterar qualquer servidor, a skill executa o teste abaixo. Assim, um host inacessível aparece na tabela antes que o playbook modifique outra máquina:

```
./deploy check-hosts
```

O comando lê `ansible/inventory.yml`, chama o módulo `ping` do Ansible e mostra uma linha por máquina:

```
SERVIDOR  ENDEREÇO       PORTA  USUÁRIO  PAPEL    ESTADO
--------  -------------  -----  -------  -------  ----------
app01     203.0.113.10   22     root     manager  conectado
app02     203.0.113.11   22     deploy   worker   conectado
```

O estado `conectado` confirma que o Ansible conseguiu autenticar e executar o módulo remoto. Ele não afirma que o Docker ou a aplicação estejam saudáveis. Se qualquer host estiver inacessível, o comando termina com erro e o deploy para antes da primeira escrita remota.

Sincronização das chaves públicas

A automação procura arquivos com o padrão `~/.ssh/*.pub` na máquina que está executando o agente. Somente o conteúdo das chaves públicas entra no `authorized_keys` de `deploy` em cada servidor cadastrado. Arquivos privados, como `id_ed25519` ou `id_rsa` sem o sufixo `.pub`, não são abertos nem enviados.

Quando você quiser executar apenas essa etapa em um painel do Herdr, use:

```
./deploy sync-keys
```

A operação primeiro executa `check-hosts`. Depois, mantém as chaves já presentes e inclui somente as ausentes. Ela não usa sincronização exclusiva e, portanto, não remove o acesso de outra máquina administrativa.

Comandos curtos para o Herdr

COMANDO	FINALIDADE	EFEITO PERSISTENTE
`./deploy check-hosts`	listar o inventário e testar conexões	nenhum
`./deploy secrets`	incluir campos ausentes	atualiza o Vault
`./deploy sync-keys`	autorizar chaves `.pub`	atualiza os hosts
`./deploy run`	aplicar o playbook com senha manual	atualiza Swarm e stack
`./deploy configure-vault`	cadastrar senha externa	grava arquivo local fora do Git
`./deploy run --non-interactive`	publicar com fonte de senha pronta	atualiza Swarm e stack

O inventário padrão é `ansible/inventory.yml`. Para conferir outro arquivo sem alterar o projeto, defina `ANSIBLE_INVENTORY` somente para aquela execução. Estes exemplos cobrem as ações disponíveis:

```
./deploy check-hosts

ANSIBLE_INVENTORY=ansible/inventory.staging.yml ./deploy check-hosts

./deploy secrets

./deploy sync-keys

./deploy run
```

`check-hosts` não recebe senhas como argumentos. `secrets` abre prompts ocultos e não aceita valores pela linha de comando. `sync-keys` recusa continuar quando não encontra uma chave pública local. `run` testa as conexões antes do playbook e solicita a senha do Vault no próprio terminal no modo manual.

Volte ao capítulo Como funciona o Deploy da aplicação para acompanhar build, publicação, rollout, migrations, rollback e conferência da versão ativa.

O capítulo Senha do Vault e deploy pela IA explica o cadastro externo, os dois modos de execução e a migração de scripts existentes.

Senha do Vault e deploy pela IA

Configurar a senha fora do projeto

Você pode continuar digitando a senha a cada deploy ou preparar a máquina para que a IA execute a publicação sem perguntas. Nos dois casos, os valores da aplicação permanecem criptografados no Ansible Vault. A configuração externa muda somente a maneira de fornecer a senha que abre esse arquivo.

Comando configure-vault

Execute `./deploy configure-vault` no seu terminal. O comando pede a senha atual do Vault duas vezes e grava um arquivo fora do repositório. A senha não aparece na tela, nos argumentos do processo ou nas mensagens do configurador. Ela fica em texto simples no arquivo local, com permissão `600`, dentro de uma pasta com permissão `700`. Seu usuário e administradores da máquina podem ler esse arquivo; por isso, use somente uma máquina sob seu controle.

O destino padrão é `~/.config/specsfy/vault/<projeto>-<hash>/password`. O hash identifica o caminho real do checkout e separa projetos com nomes iguais. A variável `XDG_CONFIG_HOME`, quando definida com caminho absoluto, substitui `~/.config`. Mover o checkout exige configurar o novo caminho ou fornecer a fonte explicitamente.

OPÇÃO	TIPO E PADRÃO	EFEITO
`--vault-password-file <arquivo>`	caminho opcional	escolhe outro arquivo externo
`--show-path`	booleano, desativado	mostra somente o destino, sem criar nem ler a senha

O configurador recusa caminhos dentro do worktree e links simbólicos. Antes de substituir um arquivo, pede confirmação; responder `n` ou usar Enter preserva o conteúdo. Senhas diferentes na confirmação também preservam o arquivo. O comando exige terminal para cadastrar e não aceita a senha como argumento. Sucesso retorna código `0`; falha retorna `1`, opção inválida `2` e cancelamento por Ctrl+C `130`.

Estes cinco usos cobrem o cadastro, a consulta e a separação de configurações:

```
# Cadastrar a senha na localização padrão deste projeto
./deploy configure-vault
```

```
# Conferir o destino sem abrir o arquivo
./deploy configure-vault --show-path
```

```
# Cadastrar um arquivo externo específico para produção
./deploy configure-vault \
  --vault-password-file "$HOME/.config/minha-app/producao/password"
```

```
# Conferir um destino personalizado antes do cadastro
./deploy configure-vault --show-path \
  --vault-password-file "$HOME/.config/minha-app/producao/password"
```

```
# Usar uma raiz de configuração diferente nesta máquina
XDG_CONFIG_HOME="$HOME/.config-automacao" ./deploy configure-vault
```

Um destino personalizado precisa ser informado nas execuções seguintes por `--vault-password-file`, `ANSIBLE_VAULT_PASSWORD_FILE` ou `ansible.cfg`. O cadastro padrão é descoberto automaticamente pelo modo sem interação do mesmo checkout e usuário. A consulta `--show-path` ajuda a conferir permissões com `stat`, sem revelar o conteúdo.

Trocar, remover ou configurar outra máquina

Para substituir a cópia local da senha, execute o mesmo comando de cadastro, confirme com `s` e informe a senha atual. Isso não altera a criptografia do Vault. Quando a intenção for mudar a senha que criptografa os dados, faça a rotação pelo Ansible e atualize depois todas as fontes que fornecem essa senha.

Para remover o cadastro padrão, confira primeiro o caminho e remova somente aquele arquivo:

```
./deploy configure-vault --show-path
rm -- "$(./deploy configure-vault --show-path)"
```

Após a remoção, o deploy manual continua disponível. O modo da IA só funciona quando outra fonte estiver configurada. Para um arquivo personalizado, remova o caminho que você cadastrou e retire a referência correspondente do ambiente ou de `ansible.cfg`. Essa remoção não apaga o Vault criptografado do projeto.

Em outra máquina, execute o cadastro novamente no checkout correspondente. Não envie o arquivo da senha pelo Git. A configuração é local ao usuário que executa o Ansible; um agente executado em container ou por outro usuário precisa receber sua própria fonte de senha e acesso SSH ao ambiente autorizado.

Deploy manual e deploy pela IA

Comando run

`./deploy run` mantém a entrada manual: pede a senha no terminal, mesmo quando existe uma fonte externa. A senha digitada fica em um arquivo temporário com permissão `600` durante a execução, removido ao concluir ou cancelar normalmente. Um encerramento forçado por `SIGKILL` pode deixar esse temporário na máquina.

`./deploy run --non-interactive` usa uma fonte configurada e não pede dados. Antes de conectar aos servidores, valida localmente a descriptografia de `ansible/group_vars/all/vault.yml`. Depois, testa os hosts e aplica o playbook. Senha incorreta, fonte ausente ou falha do cofre encerram a execução, sem recorrer a uma pergunta ou tentar outra fonte.

OPÇÃO	TIPO E PADRÃO	USO
`--non-interactive`	booleano, desativado	habilita execução pela IA ou CI/CD
`--vault-password-file <fonte>`	caminho opcional	arquivo ou script executável que fornece a senha
`--vault-id <id>@<fonte>`	texto opcional, repetível	associa uma fonte a um ambiente

As opções de fonte em `run` exigem `--non-interactive`. A ordem de escolha é: fontes explícitas do comando; configuração nativa resolvida pelo Ansible; cadastro externo padrão deste checkout. A configuração nativa inclui `ANSIBLE_VAULT_PASSWORD_FILE`, `ANSIBLE_VAULT_IDENTITY_LIST` e `ansible.cfg`, com a precedência do próprio Ansible. Os IDs permitem usar mais de uma senha no mesmo Vault. Fontes com `@prompt` são recusadas nesse modo.

O comando exige Node.js 22 ou superior e os executáveis `ansible-config`, `ansible-inventory`, `ansible` e `ansible-playbook`. O ambiente também precisa de acesso SSH por chave e das permissões de elevação usadas pelo playbook. O modo sem interação desativa as perguntas de SSH, sudo e Vault; não concede permissões adicionais. Uma configuração de cofre deve estar autenticada antes da execução. Scripts que fornecem senha não podem depender de perguntas.

Nos cinco exemplos abaixo, confirme os hosts declarados no arquivo selecionado antes de publicar. O alvo padrão continua sendo `ansible/inventory.yml`:

```
# Publicar manualmente, digitando a senha
./deploy run
```

```
# Permitir que a IA use o cadastro padrão já preparado
./deploy run --non-interactive
```

```
# Usar um arquivo externo informado nesta execução
./deploy run --non-interactive \
  --vault-password-file "$HOME/.config/minha-app/producao/password"
```

```
# Usar uma fonte fornecida pela automação, como um script de cofre
ANSIBLE_VAULT_PASSWORD_FILE="$HOME/.local/bin/minha-app-vault-client" \
  ./deploy run --non-interactive
```

```
# Selecionar hosts e identidade de staging explicitamente
ANSIBLE_INVENTORY=ansible/inventory.staging.yml \
  ./deploy run --non-interactive \
  --vault-id "staging@$HOME/.config/minha-app/staging/password"
```

O script de cofre imprime somente a senha em stdout e retorna código `0`. Mensagens de diagnóstico pertencem a stderr e não devem conter segredos. O Ansible pode chamar o script várias vezes durante um deploy, por isso a fonte precisa aceitar consultas repetidas. No CI/CD, configure o secret pelo mecanismo do provedor e disponibilize um arquivo temporário protegido ou um script para consulta; limpe o temporário ao final do job.

A execução mostra a tabela de conexões e o resultado do playbook. Código `0` confirma o sucesso dos comandos; a IA ainda deve conferir os serviços e a versão publicada. Falhas de configuração ou execução retornam `1`, argumentos inválidos retornam `2` e Ctrl+C retorna `130`. Sem fonte pronta, a IA informa a pendência e orienta o cadastro humano, sem pedir a senha na conversa.

Cadastro de segredos com a mesma configuração

Comando secrets

`./deploy secrets` continua solicitando os valores ausentes da aplicação no terminal. Para a senha do Vault, reaproveita as mesmas fontes explícitas, nativas ou locais do modo automatizado. Sem nenhuma fonte configurada, pede também a senha. O comando não recebe valores secretos como argumentos.

As opções `--vault-password-file <fonte>` e `--vault-id <id>@<fonte>` são opcionais, com os mesmos tipos e ordem de escolha de `run`. Não há `--non-interactive` para cadastrar valores: quando faltam campos, o terminal humano é obrigatório. Com todos os campos presentes, o comando encerra sem perguntas ou alterações.

O arquivo `ansible/vault-fields.txt` contém os nomes desejados, um por linha, no formato `vault_` seguido de letras minúsculas, números ou sublinhados. O utilitário preserva campos existentes e publica o YAML atualizado somente depois de criptografar todos os novos valores.

Uma falha intermediária mantém o arquivo original. É necessário ter `ansible-vault` instalado.

```
# Cadastrar os campos faltantes usando a fonte já configurada
./deploy secrets
```

```
# Cadastrar com um arquivo externo específico
./deploy secrets \
  --vault-password-file "$HOME/.config/minha-app/producao/password"
```

```
# Consultar o cofre por um script já autenticado
./deploy secrets \
  --vault-password-file "$HOME/.local/bin/minha-app-vault-client"
```

```
# Identificar a senha usada para os novos valores
./deploy secrets \
  --vault-id "producao@$HOME/.config/minha-app/producao/password"
```

```
# Reutilizar uma fonte definida no ambiente
ANSIBLE_VAULT_PASSWORD_FILE="$HOME/.config/minha-app/producao/password" \
  ./deploy secrets
```

Com várias identidades, configure também `vault_encrypt_identity` em `ansible.cfg`, ou `ANSIBLE_VAULT_ENCRYPT_IDENTITY`, para indicar qual delas criptografa os campos novos. O comando informa somente o caminho atualizado ou que todos os campos já existem. Os códigos de saída seguem o cadastro externo: `0` para sucesso, `1` para falha, `2` para argumentos inválidos e `130` para cancelamento.

Atualizar scripts que já existem

Atualizar a skill instalada não substitui os scripts do seu projeto. Peça à IA para comparar a versão atual de `deploy`, `ansible/create-vault.sh`, `ansible/check-hosts.mjs` e o novo `ansible/vault.mjs` com uma geração temporária, preservando suas personalizações. O scaffold recusa sobrescrever arquivos existentes. Depois da migração, valide os dois modos em um ambiente de teste e faça o cadastro externo na máquina que executará o deploy.

Uso avançado do Specsfy

Este guia reúne recursos usados depois da primeira spec: seleção de especialistas, progresso em JSON, atualização visual e retomada de uma mudança posterior. Para acompanhar os exemplos, conclua o primeiro projeto, mantenha o CLI e o framework instalados e execute os comandos na raiz do projeto consumidor.

Detecte e instale orientação técnica

Comece com `skills detect` para ler as recomendações do catálogo sem alterar o projeto:

```
specsfy skills detect --project .
```

Quando todas as recomendações forem aplicáveis, `--detected` instala o framework e os especialistas em uma única execução. O `skills-lock.json` registra os arquivos publicados:

```
specsfy install --project . --detected
```

Quando o catálogo listar tecnologias que não pertencem à aplicação, repita `--specialist` somente com os nomes confirmados. Assim, o instalador publica as bases e ignora as recomendações que não foram escolhidas:

```
specsfy install --project . \
  --specialist specsfy-specialist-laravel \
  --specialist specsfy-specialist-postgres
```

Em um projeto já preparado, acrescente somente os especialistas escolhidos com `npx skills add`:

```
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-laravel --agent universal --copy --full-depth
```

A detecção usa manifests, dependências e arquivos reconhecidos pelo catálogo. O comando de instalação só deve ser executado depois da revisão. Os especialistas orientam escolhas técnicas, mas não criam specs nem aprovam gates.

Automatize a leitura de progresso

```
specsfy progress --project . --json
specsfy progress --project . --watch --interval 0.5 --json
```

O JSON contém `summary` e `specs`. Com `--watch`, um snapshot novo aparece somente quando o conteúdo das specs muda. Painéis podem consumir essa saída para leitura, mas os gates continuam sendo editados apenas na `spec.md` pela skill responsável.

Ajuste a atualização visual

```
specsfy config show --project .
specsfy config set --project . --watch-interval 0.5
```

A configuração vive em `<projeto>/.specsfy/config.json` e preserva chaves desconhecidas. Consulte todos os recursos no guia do CLI.

Atualize sem perder customizações

```
specsfy update --project .
```

O CLI usa fingerprints para distinguir conteúdo gerenciado intacto de customização local. Uma diferença local faz a atualização ou a remoção ser recusada. Use `--force` somente depois de revisar a comparação e confirmar que o conteúdo protegido pode ser descartado.

Quando o CLI foi instalado pelo npm, o comando abaixo atualiza o pacote global:

```
specsfy upgrade
```

Quando a instalação usa o executável Node oficial, `specsfy upgrade` pode substituí-lo automaticamente. Para fazer a troca manual, use o download e restaure a permissão:

```
curl -fL get.specsfy.dev -o "$HOME/.local/bin/specsfy"
chmod +x "$HOME/.local/bin/specsfy"
```

Incorpore uma mudança na spec

Quando lembrar de algo depois da definição, use a entrada explícita:

```
Use $specsfy-update-spec em
specs/<estado>/<NNNN>-<slug>/spec.md:
quero adicionar, remover, corrigir ou mudar esta especificação.
```

Quando um requisito observável muda, a skill reabre a definição. O Definition, o Plan e o Delivery Gate precisam de evidência nova. Quando apenas o plano técnico muda, ela reabre o Ato II e o Ato III. A spec alterada invalida as provas que dependiam da versão anterior e preserva tarefas e evidências ainda compatíveis.

As skills fazem transições e retomadas na mesma conversa. Esse handoff não amplia autorização para instalar, publicar, fazer deploy ou executar ação destrutiva.

Consulte Atualizar uma especificação para a classificação completa e exemplos em linguagem comum.

Limites

- `--force` pode descartar customizações protegidas.
- `--detected` depende do catálogo e do estado observado no projeto.
- A TUI e o progresso projetam estado, mas não substituem a spec.
- o especialista não substitui a confirmação da versão, das convenções e das falhas possíveis no projeto.

O resultado esperado é um projeto com especialistas escolhidos de forma explícita, automação somente de leitura e gates coerentes com a versão atual da spec.

Usar Specsfy com Laravel

`$specsfy-specialist-laravel` acrescenta verificações próprias do Laravel ao fluxo do Specsfy. A spec continua governando a mudança, e o especialista segue as convenções e a versão comprovadas pelo projeto.

Confirmar a detecção

O catálogo detecta Laravel por `artisan`, `composer.json` ou pela dependência `laravel/framework`. Confirme a versão e as extensões em `composer.json` e `composer.lock`. Uma API só deve orientar a implementação quando existir na versão usada pela aplicação.

Instalação

Na raiz do projeto, `--detected` instala o framework e o especialista quando o catálogo reconhece Laravel:

```
specsfy install --project . --detected
```

Para revisar a recomendação sem instalar arquivos, use `skills detect`. A saída deve incluir `specsfy-specialist-laravel` quando `artisan` ou a dependência do framework for encontrada:

```
specsfy skills detect --project .
```

Quando o nome já estiver confirmado, `npx skills add` instala somente o especialista Laravel e registra os arquivos gerenciados:

```
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-laravel --agent universal --copy --full-depth
```

Para adotar um pacote Laravel a partir de um repositório GitHub, instale também o gestor de pacotes:

```
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-laravel-package-manager --agent universal --copy --
full-depth
```

Depois, peça ao agente:

```
Use $specsfy-specialist-laravel-package-manager com
https://github.com/organizacao/pacote.
Leia a documentação, confira se o pacote já está instalado e, com autorização,
instale-o e documente seu uso.
```

O especialista lê `composer.json`, `composer.lock`, `.specsfy/PACKAGES.md` e as fichas atuais antes de executar Composer. Para cada dependência direta, ele mantém uma ficha em `docs/packages/<vendor>-<nome>.md` e atualiza `docs/packages/README.md` com versão, finalidade e links. Pacotes já instalados são reaproveitados; dependências transitivas continuam relacionadas em `.specsfy/PACKAGES.md`.

Aplicar na spec

1. Capture ou promova a ideia pelo primeiro projeto.
2. Peça ao agente para usar `$specsfy-specialist-laravel` na fatia ativa.
3. Confirme a versão, extensões, convenções locais e o caminho da requisição.
4. Na definição e no plano, registre autorização e validação. Quando a mudança alcançar persistência ou execução assíncrona, inclua transações, idempotência, filas, falhas e uma tarefa `[CODE] [MIGRATION]` separada. Essa tarefa aponta para o arquivo em `database/migrations/` e inclui os comandos de aplicação e consulta do estado.
5. Derive testes para caminho feliz, autorização, validação, efeitos e falhas.
6. Antes de qualquer teste, crie `.env.testing` com `APP_ENV=testing` e um `DB_DATABASE` ou `DB_URL` explícito, diferente do destino usado pelo `.env`.
7. Implemente controllers finos e mantenha as regras na camada já adotada pelo projeto. Inspecione as consultas e o N+1 quando a quantidade de relações puder aumentar o tempo da resposta.
8. Confira o ambiente e o comando antes de executar os checks:

```
node .agents/skills/specsfy-setup/scripts/check_database_safety.mjs \
  --project . --command "php artisan test"
```

Somente a saída `SAFE` permite continuar. `PENDING` encerra a etapa até a configuração ser corrigida. `IGNORED` descarta o comando, sem pedir autorização para forçá-lo.

Quando houver uma migration planejada, aplique-a no banco de teste protegido e confirme o resultado antes de concluir a tarefa:

```
php artisan migrate --env=testing
php artisan migrate:status --env=testing
```

O registro da tarefa precisa conter o caminho exato da migration e a saída com exit code zero dos dois comandos. Criar um model, alterar uma consulta ou fazer os testes passarem não substitui essa conferência.

1. Em Laravel com Pest, o CLI oferece:

```
specsfy test --project .
```

O CLI detecta `artisan` e `pestphp/pest`, chama `php artisan test` e preserva o exit code. Ele não recebe uma string arbitrária de shell. A verificação anterior continua obrigatória antes desse comando.

O que o especialista acrescenta

- contratos HTTP, Form Requests, policies, resources e bindings.
- Eloquent, eager loading, casts e transações conscientes.
- jobs idempotentes, tentativas, backoff e tratamento de falha.
- migrations compatíveis com volume, locks, rollback e deploy misto.
- verificação de queues, scheduler, cache, configuração e ambiente.
- leitura, instalação autorizada e documentação de pacotes Composer recebidos por URL GitHub.

Resultado esperado

A spec continua sendo a fonte normativa, enquanto os testes e a implementação consideram as falhas possíveis do Laravel observado no projeto e na versão instalada.

Limites

- não aplique a skill a PHP sem Laravel.
- não presuma APIs pela versão mais recente da documentação.
- não execute migration, deploy ou comando operacional fora da autorização registrada na tarefa e do ambiente de teste protegido.
- não execute teste sem `.env.testing` separado do banco de desenvolvimento.
- não use `RefreshDatabase`, `DatabaseMigrations`, `migrate:fresh`, `migrate:refresh`, `migrate:reset`, `migrate:rollback` ou `db:wipe`; use `DatabaseTransactions`, factories e limpeza limitada aos registros criados pelo próprio caso.
- não confie apenas na validação ou autorização da interface.

Não use esse especialista para PHP sem Laravel nem para fixar uma versão que o projeto não comprova. O código, `composer.json`, `composer.lock`, os testes e a configuração local permanecem como evidência do estado da aplicação.

Usar Specsfy com Astro

`$specsfy-specialist-astro` acrescenta ao fluxo do Specsfy as escolhas de renderização, conteúdo, ilhas e performance próprias de um site Astro. A skill usa a versão e a configuração do projeto, sem presumir o adapter ou o modo de saída.

Confirmar a detecção

O catálogo detecta Astro pela dependência `astro` em `package.json` ou pelos arquivos `astro.config.mjs` e `astro.config.ts`. Descubra no projeto a versão, o output mode, o adapter, as integrações e as fontes de conteúdo.

Instalação

```
specsfy skills detect --project .
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-astro --agent universal --copy --full-depth
```

Quando todas as recomendações exibidas forem aplicáveis, `--detected` instala o framework e os especialistas em uma única execução. Confira depois se `specsfy-specialist-astro` aparece no catálogo instalado:

```
specsfy install --project . --detected
```

Aplicar na spec

1. Conduza a ideia até a spec conforme o primeiro projeto.
2. Peça ao agente para usar `$specsfy-specialist-astro` na fatia ativa.
3. Classifique cada rota afetada como estática, sob demanda ou endpoint.
4. Mantenha HTML estático por padrão e escolha uma diretiva `client:*` somente para a interação que precisa de hidratação.
5. Modele conteúdo com schemas e relações explícitas. Defina cache, headers, imagens e metadados quando aplicáveis.
6. Crie testes derivados do Gherkin para rotas, conteúdo inválido e comportamento hidratado.
7. Execute `astro check`, a suíte do projeto, o build de produção e o preview no runtime do adapter disponível no projeto.

O que o especialista acrescenta

- escolha explícita entre static, on-demand, island e endpoint.
- proteção contra transporte de dados sensíveis para ilhas.
- validação de slugs, relações e conteúdo.
- semântica, canonical, sitemap e dados estruturados.
- inspeção de payload cliente, Core Web Vitals e compatibilidade do adapter.

Resultado esperado

A fatia preserva a fonte normativa Specsfy e torna observáveis a estratégia de renderização, a hidratação necessária, os contratos de conteúdo e a validação no runtime alvo.

Limites

- não escolha SSR sem necessidade de sessão, personalização ou frescor.
- não suponha APIs de Node em adapters edge.
- não valide apenas no servidor de desenvolvimento.
- confirme scripts e comandos disponíveis no `package.json`.

Não use esse especialista para prescrever um adapter ou modo de saída sem evidência. A versão, o adapter, os scripts e as integrações são comprovados pelos manifests, lockfiles e pela configuração do projeto consumidor.

Usar Specsfy com Next.js

`$specsfy-specialist-nextjs` acrescenta ao fluxo do Specsfy as escolhas de Server e Client Components, cache, mutations, rotas e deploy próprias do Next.js. A skill confirma o router e a versão porque os padrões de cache mudam entre gerações do framework.

Confirmar a detecção

O catálogo detecta Next.js pela dependência `next` em `package.json` ou por `next.config.js`, `next.config.mjs` e `next.config.ts`. Confirme a versão e o router. O runtime, o destino de deploy e as flags ativas também precisam ser registrados no plano.

Instalação

```
specsfy skills detect --project .
npx skills add https://github.com/promovaweb/specsfy \
  --skill specsfy-specialist-nextjs --agent universal --copy --full-depth
```

Quando todas as recomendações forem aplicáveis, `--detected` instala as bases e os especialistas em uma única execução. Confira depois se `specsfy-specialist-nextjs` aparece no catálogo instalado:

```
specsfy install --project . --detected
```

Aplicar na spec

1. Conduza a ideia até a spec pelo primeiro projeto.
2. Peça ao agente para usar `$specsfy-specialist-nextjs` na fatia ativa.
3. Mapeie a rota, o layout e os estados `loading`, `error` e `not-found`. Registre também o limite entre o código do servidor e a interação no navegador.
4. No App Router, mantenha componentes no servidor por padrão e mova ao cliente apenas o componente que requer estado, eventos ou APIs do navegador.
5. Defina cache, revalidation, tags e comportamento dinâmico explicitamente.
6. Trate Server Actions e Route Handlers como superfícies públicas: valide autenticação, autorização e entrada em cada mutation.
7. Derive testes para estados, redirects, autorização, invalidação e isolamento de dados entre usuários.
8. Execute lint, typecheck, testes, build e runtime de produção conforme os scripts do `package.json`.

O que o especialista acrescenta

- controle do limite entre Server e Client Components.
- prevenção de segredos e módulos server-only no bundle cliente.
- análise de waterfalls, streaming e recuperação de erro.
- cache como contrato compatível com a versão instalada.
- metadata, assets, bundle, imagens, fontes e Web Vitals.

Resultado esperado

As escolhas de renderização, cache e segurança ficam rastreadas pela spec e provadas por testes e build na versão e no router realmente usados.

Limites

- não presuma App Router ou semântica de cache sem confirmar a versão.
- não mova uma árvore inteira ao cliente por conveniência.
- não use middleware para lógica longa ou incompatível com o runtime.
- não assuma comportamento específico do host sem documentá-lo.

Não use esse especialista para escolher App Router, Pages Router ou runtime sem evidência. O código, `package.json`, o lockfile e a configuração comprovam o router, a versão, o runtime e os scripts disponíveis no projeto consumidor.

Créditos do Specsfy

Esta página registra a autoria pública do Specsfy e aponta as fontes usadas para confirmar contribuições, identidade visual e componentes relacionados. O histórico Git preserva a autoria detalhada de cada mudança.

Projeto e manutenção

Specsfy é um projeto da Promovaweb, criado e mantido por **Luiz Eduardo Oliveira Fonseca** com a comunidade. O contato público do projeto é contato@promovaweb.com.

Comunidade

Contribuições em documentação, código, testes e pesquisa fazem parte da evolução do projeto. O histórico do monorepo preserva a autoria de cada arquivo e commit, por isso este guia não mantém uma lista manual que ficaria desatualizada.

Identidade

Logos, cores, tipografia, voz, acessibilidade e regras de aplicação pertencem ao diretório `brand/`. Use os ativos e orientações desse diretório ao apresentar o projeto.

Componentes relacionados

O instalador do Specsfy delega a instalação de skills ao projeto `vercel-labs/skills`. O CLI pode usar o executável `skills` ou o fallback por `npx`, conforme documentado no guia de instalação.

Inspirações e fontes

O Specsfy desenvolve um contrato executável próprio e reconhece estas referências como inspiração:

- GitHub Spec Kit: aplicação de specification-driven development em etapas próximas ao código.
- OpenSpec: especificações e mudanças mantidas no repositório como um acordo leve entre o desenvolvedor responsável e o agente.
- *Categorias*, de Aristóteles: referência filosófica para classificar objetos, atributos, relações e estados antes de formular afirmações sobre eles.

As referências não são dependências do Specsfy e não tornam os métodos equivalentes. As skills, os templates, os validadores e os testes deste repositório continuam sendo as fontes do comportamento executável.

Como contribuir

Localize primeiro o diretório responsável no quadro dos módulos, leia as instruções locais e preserve os limites de cada repositório Git. A documentação oficial fica neste repositório. O comportamento executável e os testes permanecem no módulo que implementa cada recurso.

O histórico e os arquivos de licença de cada repositório comprovam autoria e licenciamento. Quando uma licença não estiver declarada, não presuma seus termos. A identidade oficial do Specsfy permanece em `brand/`.