20 — Ciclo de desenvolvimento

20 — Ciclo de desenvolvimento

> Do recebimento da tarefa ao merge na master: issue, branch, commits, MR, review. Processo pensado para projeto só de frontend, com o harness integrado. Este documento cobre **como o trabalho flui**; os anteriores cobrem como o código se organiza.

Referência rápida

```
tarefa recebida
    → clarificação (contexto, escopo, critérios, dependências)    → issue aberta com template + labels + assignee
    → branch criada: <id-issue>-<descricao-curta>    → ./init.sh antes de começar (baseline limpo)
    → sync com master
    → desenvolvimento com commits semânticos
    → ./init.sh FAST antes de cada commit    → ./init.sh FULL + verificação no browser antes do MR
    → sync com master antes de abrir MR
    → MR aberto com template + reviewer
```

```
    → code review → ajustes → aprovação
    → merge → issue fechada
    → fim de sessão do harness (feature_list, progress, handoff)
```

1. Recebimento da tarefa

Antes de abrir issue, você precisa conseguir responder as quatro perguntas. Se não conseguir, peça uma reunião curta (15 min) com quem trouxe a demanda.

Pergunta	O que responder
Contexto	Por que essa tarefa existe? Que problema de negócio resolve?
Escopo	O que está dentro e fora dessa entrega?
Critérios de aceite	Como saberemos que está pronto? Quem valida?
Dependências	Algo precisa estar pronto antes? Outro time envolvido?

Num projeto Curio há uma quinta pergunta, específica: **o caso de uso do backend já existe?** Se a tela depende de um caso de uso ainda não implementado, isso é dependência bloqueante — registre.

2. Issue no GitLab

Template

`.gitlab/issue_templates/default.md`:

```
## Contexto
<!-- Por que essa tarefa existe? O que motivou essa demanda? -->
## O que fazer
<!-- Descrição objetiva e clara do que precisa ser entregue. -->
## Critérios de aceite
- [ ]
- [ ]
```

```
## Notas técnicas
<!-- Id do caso de uso, nomes de request, impacto em outras telas.     Deixe em branco se não houver. -->
## Referências
<!-- Design no Figma, documentação, issue relacionada. -->
```

Título

Padrão `[Tipo] Verbo + objeto`:

```
[Feature] Adicionar tela de cadastro de fornecedor[Bug] Corrigir data de expiração enviada sem sufixo de hora
[Chore] Atualizar dependências para versão LTS[Refactor] Extrair filtro de busca para componente comum
```

Labels

Dois eixos:

Eixo	Labels
Tipo	`type::feature` `type::bug` `type::chore` `type::refactor`
Prioridade	`priority::critical` `priority::high` `priority::medium` `priority::low`

Assignee é obrigatório. Issue sem dono não entra no sprint.

Definition of Ready

Uma issue só entra no sprint quando:

- ☐ Título segue o padrão
- ☐ Contexto preenchido e compreensível
- ☐ Critérios de aceite listados e **verificáveis**
- ☐ Assignee atribuído

- [ ] Sem dependência bloqueante em aberto
- [ ] Se a tela depende de caso de uso, o id e os requests estão nas notas técnicas

3. Branch

Crie a partir da issue no GitLab — o id vem automaticamente. Sempre a partir da `master` atualizada.

```
<id-issue>-<descricao-curta>
123-cadastro-fornecedor
456-corrigir-data-expiracao
789-atualizar-dependencias
```

4. Antes de começar

Rode o Startup Workflow do harness (19):

```
git fetch origin && git merge origin/master
./init.sh
```

Se o baseline já estiver quebrado, corrija antes de começar a feature. Misturar conserto de baseline com feature nova produz um diff irrevisável.

Mantenha a branch sincronizada com a `master` ao menos uma vez por dia, e sempre antes de abrir o MR.

5. Commits

Conventional Commits:

```
<tipo>: descrição curta
```

Tipo	Quando usar

`feat`	nova funcionalidade
`fix`	correção de bug
`refactor`	refatoração sem mudança de comportamento
`docs`	documentação
`chore`	manutenção, dependências
`perf`	melhoria de performance
`test`	testes — só se o projeto adotar suíte

```
feat: adicionar tela de cadastro de fornecedorfix: corrigir data de expiração enviada sem sufixo de hora
refactor: extrair filtro de busca para componente comum
```

Princípios

- Cada commit faz uma coisa só
- O baseline passa após cada commit
- A mensagem explica o **porquê**, não o **o quê**

Verificação antes de cada commit

```
./init.sh
```

O `pre-commit` já roda lint-staged, versão de Node e checagem de flags de debug (14) — mas ele só vê os arquivos staged. O `init.sh` FAST vê o projeto inteiro.

Não suba código que quebra o baseline. Isso bloqueia o time e polui o histórico.

> Se este projeto for só frontend, ignore quaisquer instruções de build Java/Delphi herdadas de um processo combinado com backend — não se aplicam.

6. Merge Request

Template

`.gitlab/merge_request_templates/default.md`:

```
## O que foi feito
<!-- Breve descrição das mudanças. -->
## Issue relacionada
Closes #
## Como testar
1.
2.
## Screenshots (se aplicável)
<!-- Antes/depois para mudanças visuais. -->
## Checklist
- [ ] `./init.sh FULL` passando localmente- [ ] Tela aberta no browser e fluxo verificado contra o backend real
- [ ] Sem `console.log` ou código de debug (`enableLogs: true`)- [ ] Documentação atualizada (se necessário)
- [ ] Branch atualizada com a master- [ ] `harness/state/feature_list.json` e `progress.md` atualizados
```

> **Ajuste o checklist ao que o projeto realmente tem.** Um template herdado de outro processo pode pedir itens como "Testes escritos ou atualizados" e "Pipeline passando" mesmo quando o projeto não tem suíte de testes nem `.gitlab-ci.yml` (15). Checklist que pede o inexistente treina o time a marcar caixa sem ler. Se o projeto novo adotar CI, acrescente a linha; se não, não a coloque.

Configuração no GitLab

- Aprovação mínima de 1 reviewer
- Pipeline obrigatória para habilitar o merge — **se houver pipeline**

7. Code review

Para quem revisa

- Leia a descrição do MR antes do código — entenda o contexto primeiro
- Comente com intenção explícita:
  - `nit:` — sugestão menor, não bloqueia
  - `suggestion:` — sugestão de melhoria, bloqueia num primeiro momento
  - `blocking:` — precisa ser resolvido antes do merge
- Questione o **porquê**, não só o **como**
- Priorize lógica e segurança; estilo é papel do linter
- Aprove se o código está correto e legível. Não exija que seja idêntico ao que você faria

Para quem abre

- Responda todos os comentários antes de pedir re-review
- **Não faça force push depois de abrir o MR** — quebra o histórico de revisão
- Marque como resolvido ao aplicar a correção
- Atenda ao que o revisor observa, mesmo sendo pequeno — exceto em demanda urgente

O que observar

Categoria	O que verificar
Baseline	`./init.sh` passa; sem warning novo
Lógica	Edge cases não cobertos, condições incorretas
Curio	Caso de uso aberto antes do request; guarda por `useRef`; sem reabrir
Erro	Sem `try/catch` só para exibir mensagem — o global já notifica (07)
Segurança	Dado exposto, validação ausente, segredo commitado

Categoria	O que verificar
Performance	Chamada redundante, request em loop, lista sem virtualização
Legibilidade	Nome autoexplicativo, função com responsabilidade única
Convenções	`_Prefixo`, barrel atualizado, alias nos dois configs (13)

8. Merge

- Só com pipeline verde — se houver pipeline
- `Closes #123` no MR fecha a issue automaticamente

9. Fim de sessão

Depois do merge, feche o ciclo no harness (19):

1. `harness/state/feature_list.json` — status + evidência
2. Se chegou a `passing`, copiar para `harness/global_feature_list.json`
3. `harness/state/progress.md` — o que foi feito, bloqueios, próximo passo
4. `harness/state/session-handoff.md` — resultado da verificação

A próxima sessão precisa conseguir rodar `./init.sh` imediatamente.

Revision #2
Created Thu, Aug 20, 2026 5:12 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:56 PM by Geraldo Barbosa