REGRAS CRÍTICAS E WORKFLOW FINMATH

Guia Completo para Todas as Sprints (S1-S3) e Histórias (H1-H24)

*Versão 2.0 Expandida | Atualizado em: 23/10/2025*

# 📑 SUMÁRIO EXECUTIVO

**Este documento estabelece as regras críticas de workflow Git, validação anti-regressão e documentação para o projeto FinMath, cobrindo:**

* ✅ 3 Sprints completas (S1, S2, S3)
* ✅ 24 Histórias de Usuário (H1-H24)
* ✅ Workflow GitHub com push apenas ao final da sprint
* ✅ Validação anti-regressão obrigatória antes de cada push
* ✅ Backup exclusivo via Git (proibido .bak/.backup/.save)
* ✅ Documentação viva durante implementação

# PARTE 1: REGRAS CRÍTICAS PRINCIPAIS

## 🎯 REGRA #1: GITHUB COMO FONTE DA VERDADE (POR SPRINT)

**OBRIGATÓRIO**: No início de cada sprint, sincronizar com o GitHub. Durante a sprint, trabalho local com Git local como backup.

### Início de Cada Sprint (obrigatório):

# INÍCIO DE CADA SPRINT (obrigatório)  
git fetch origin  
git pull origin main  
git checkout -b sprint-X # Ex: sprint-1, sprint-2, sprint-3  
  
# Durante a sprint: commits locais frequentes  
git add .  
git commit -m "feat(H9): Implementa Price PMT"  
  
# Ao final da sprint: merge e push coletivo

## 🎯 REGRA #2: VALIDAÇÃO ANTI-REGRESSÃO ANTES DE PUSH FINAL

**OBRIGATÓRIO**: Ao final da sprint, antes do push para GitHub, validação completa.

# ========================================  
# VALIDAÇÃO FINAL DE SPRINT (OBRIGATÓRIA)  
# ========================================  
echo "🔍 VALIDANDO SPRINT COMPLETA ANTES DE PUSH..."  
  
# 1. Type Check  
npm run type-check  
if [ $? -ne 0 ]; then  
 echo "❌ Type check falhou - CORRIGIR ANTES DE PUSH"  
 exit 1  
fi  
  
# 2. Linting  
npm run lint  
if [ $? -ne 0 ]; then  
 echo "❌ Lint falhou - CORRIGIR ANTES DE PUSH"  
 exit 1  
fi  
  
# 3. Testes Unitários + Propriedade  
npm run test:unit  
npm run test:property  
if [ $? -ne 0 ]; then  
 echo "❌ Testes falharam - CORRIGIR ANTES DE PUSH"  
 exit 1  
fi  
  
# 4. Testes de Integração  
npm run test:integration  
if [ $? -ne 0 ]; then  
 echo "❌ Integração falhou - CORRIGIR ANTES DE PUSH"  
 exit 1  
fi  
  
# 5. Golden Files (CRÍTICO)  
npm run golden:verify  
if [ $? -ne 0 ]; then  
 echo "❌ Golden Files falharam - CORRIGIR ANTES DE PUSH"  
 exit 1  
fi  
  
# 6. Build de Produção  
npm run build  
if [ $? -ne 0 ]; then  
 echo "❌ Build falhou - CORRIGIR ANTES DE PUSH"  
 exit 1  
fi  
  
# 7. Testes E2E (se aplicável)  
npm run test:e2e  
if [ $? -ne 0 ]; then  
 echo "❌ E2E falharam - CORRIGIR ANTES DE PUSH"  
 exit 1  
fi  
  
# 8. Verificar Swagger UI  
timeout 10s npm run dev &  
DEV\_PID=$!  
sleep 5  
curl -s http://localhost:3001/api-docs > /dev/null  
DOCS\_STATUS=$?  
kill $DEV\_PID 2>/dev/null  
if [ $DOCS\_STATUS -ne 0 ]; then  
 echo "❌ Swagger UI não responde - CORRIGIR ANTES DE PUSH"  
 exit 1  
fi  
  
echo "✅ VALIDAÇÃO COMPLETA - PODE FAZER PUSH!"

## 🎯 REGRA #3: BACKUP EXCLUSIVO VIA GIT (LOCAL)

**PROIBIDO**: Criar .bak, .backup, .save no projeto.

# LIMPEZA OBRIGATÓRIA (início de cada dia)  
find . \( -name "\*bak\*" -o -name "\*backup\*" -o -name "\*.save" \) -type f -delete  
  
# ✅ SEMPRE: Git local para backup  
git add .  
git commit -m "WIP: implementando H9"  
  
# ❌ NUNCA: Backups físicos  
cp price.ts price.ts.bak # PROIBIDO!

## 🎯 REGRA #4: DOCUMENTAÇÃO VIVA OBRIGATÓRIA

**OBRIGATÓRIO**: Toda funcionalidade documentada durante implementação.

* • JSDoc/TSDoc: Funções públicas do motor
* • OpenAPI: Endpoints da API (arquivo: openapi-3.1\_finmath-v1.0.yaml)
* • ADRs: Decisões arquiteturais importantes (docs/adr/)
* • Swagger UI: Sempre acessível em desenvolvimento

# PARTE 2: WORKFLOW DETALHADO POR SPRINT

## 📋 FASE 1: INÍCIO DA SPRINT

#!/bin/bash  
# inicio-sprint.sh  
  
echo "🚀 INICIANDO SPRINT X"  
  
# 1. Sincronizar com GitHub  
echo "🔄 Sincronizando com GitHub..."  
git fetch origin  
git pull origin main  
if [ $? -ne 0 ]; then  
 echo "❌ Falha ao sincronizar - resolver conflitos"  
 exit 1  
fi  
  
# 2. Criar branch da sprint  
SPRINT\_NUM=$1  
git checkout -b "sprint-${SPRINT\_NUM}"  
  
# 3. Limpar backups físicos  
echo "🧹 Limpando backups físicos..."  
find . \( -name "\*bak\*" -o -name "\*backup\*" -o -name "\*.save" \) -type f -delete  
  
# 4. Verificar ambiente  
echo "🔍 Verificando ambiente..."  
npm run type-check  
npm run lint  
npm run test  
npm run build  
  
if [ $? -eq 0 ]; then  
 echo "✅ Ambiente pronto para desenvolvimento!"  
 echo "📋 Histórias desta sprint: H${SPRINT\_NUM}x"  
else  
 echo "❌ Ambiente com problemas - corrigir antes de começar"  
 exit 1  
fi

## 📋 FASE 2: DURANTE A SPRINT (Trabalho Local)

# COMMITS LOCAIS FREQUENTES (a cada funcionalidade/história)  
  
# Exemplo: Implementando H9 (Price)  
git add engine/src/amortization/price.ts  
git add engine/test/unit/amortization/price.test.ts  
git add engine/test/golden/sprint2/PRICE\_001.json  
git commit -m "feat(H9): Implementa Price PMT e cronograma  
  
- Motor: calculatePMT com decimal.js  
- Testes: unit + property + golden file  
- Ajuste final: saldo\_n <= 0.01  
- DoD: 5/5 critérios atendidos"  
  
# Continuar desenvolvimento local...  
# Mais commits...  
# Não fazer push ainda!

## 📋 FASE 3: FIM DA SPRINT (Push Coletivo)

#!/bin/bash  
# finalizar-sprint.sh  
  
echo "🏁 FINALIZANDO SPRINT X"  
  
# 1. Garantir que estamos na branch correta  
CURRENT\_BRANCH=$(git branch --show-current)  
if [[ ! $CURRENT\_BRANCH =~ ^sprint-[0-9]+$ ]]; then  
 echo "❌ Não está em uma branch de sprint"  
 exit 1  
fi  
  
# 2. Limpar backups físicos (última vez)  
echo "🧹 Limpeza final de backups físicos..."  
find . \( -name "\*bak\*" -o -name "\*backup\*" -o -name "\*.save" \) -type f -delete  
  
# 3. VALIDAÇÃO ANTI-REGRESSÃO COMPLETA  
echo "🔍 VALIDAÇÃO ANTI-REGRESSÃO COMPLETA..."  
npm run type-check || { echo "❌ Type check falhou"; exit 1; }  
npm run lint || { echo "❌ Lint falhou"; exit 1; }  
npm run test:unit || { echo "❌ Testes unitários falharam"; exit 1; }  
npm run test:property || { echo "❌ Testes de propriedade falharam"; exit 1; }  
npm run test:integration || { echo "❌ Testes de integração falharam"; exit 1; }  
npm run golden:verify || { echo "❌ Golden Files falharam"; exit 1; }  
npm run build || { echo "❌ Build falhou"; exit 1; }  
npm run test:e2e || { echo "❌ E2E falharam"; exit 1; }  
  
# Verificar Swagger UI  
timeout 10s npm run dev &  
DEV\_PID=$!  
sleep 5  
curl -s http://localhost:3001/api-docs > /dev/null  
DOCS\_STATUS=$?  
kill $DEV\_PID 2>/dev/null  
if [ $DOCS\_STATUS -ne 0 ]; then  
 echo "❌ Swagger UI não responde"  
 exit 1  
fi  
  
echo "✅ VALIDAÇÃO COMPLETA - SPRINT APROVADA!"  
  
# 4. Merge na main (local)  
echo "🔀 Fazendo merge na main..."  
git checkout main  
git merge "sprint-${SPRINT\_NUM}" --no-ff -m "chore: Merge Sprint ${SPRINT\_NUM}  
  
Histórias implementadas: [Lista de histórias]  
Validação anti-regressão: ✅ PASSOU  
Golden Files: X/X verdes  
Cobertura: XX%  
motorVersion: X.X.X"  
  
# 5. PUSH FINAL PARA GITHUB  
echo "📤 PUSH FINAL PARA GITHUB..."  
git push origin main  
  
if [ $? -eq 0 ]; then  
 echo "✅ SPRINT PUBLICADA NO GITHUB COM SUCESSO!"  
else  
 echo "❌ Falha no push - verificar conectividade"  
 exit 1  
fi  
  
# 6. Deletar branch local da sprint  
git branch -d "sprint-${SPRINT\_NUM}"  
echo "🎉 SPRINT X CONCLUÍDA E PUBLICADA!"

# PARTE 3: DETALHAMENTO POR SPRINT

## 🚀 SPRINT 1: FUNDAMENTOS DO MOTOR + UI BASE

**Objetivo:** Infraestrutura de engenharia; cálculos base (Juros, Equivalência, Séries); UI dos simuladores; Explain Panel; Academy com 5 tópicos mínimos.

### Histórias da Sprint 1:

|  |  |  |
| --- | --- | --- |
| ID | Descrição | Épico |
| H1 | Repositórios e CI/CD | E4 - Infra |
| H2 | Biblioteca decimal e políticas de arredondamento | E1 - Engine |
| H3 | Observabilidade básica (logs, correlation-id, motorVersion) | E4 - Infra |
| H4 | Juros compostos (FV/PV) | E1 - Engine |
| H5 | Equivalência de taxas + Taxa real | E1 - Engine |
| H6 | Séries/Anuidades (post/ant) + inversão | E1 - Engine |
| H7 | Simuladores base (Juros/Equivalência/Séries) | E3 - UI |
| H8 | Explain Panel (fórmulas + números + exportar PDF) | E3 - UI |
| H20 | Academy — 5 tópicos (teoria + exercício guiado + "Abrir no Lab") | E5 - Academy |

### Critérios de Saída (Sprint 1 "Pronto"):

* ✅ Simuladores Juros/Equivalência/Séries com P95 ≤ 150 ms
* ✅ Explain Panel presente e exportável (snapshot PDF)
* ✅ 10 Golden Files publicados (3 juros, 3 equivalências, 4 séries)
* ✅ CI/CD estável; logs com calculationId / motorVersion
* ✅ Academy: 5 tópicos com exercícios e deep-link para o Lab

### Riscos & Mitigações (Sprint 1):

* ⚠️ Atraso em UI: usar componentes do Design System; evitar CSS ad‑hoc
* ⚠️ Precisão: validar arredondamentos com GFs e testes de borda (i→0, n=1)

## 🚀 SPRINT 2: AMORTIZAÇÕES + CET BÁSICO + EXPORTAÇÕES + AUDITORIA/VALIDAÇÃO

**Objetivo:** Price e SAC completos (day count + pró‑rata), CET básico (tarifas t0), CSV/PDF, snapshots e validador inicial.

### Histórias da Sprint 2:

|  |  |  |
| --- | --- | --- |
| ID | Descrição | Épico |
| H9 | Price — PMT, cronograma, ajuste final (saldo ≤ R$0,01) | E1 - Engine |
| H10 | Day count 30/360 e ACT/365 + pró‑rata 1ª parcela | E1 - Engine |
| H11 | SAC — cronograma e ajuste final | E1 - Engine |
| H12 | CET básico (apenas tarifas t0) — Ver: Guia CET SoT | E2 - CET |
| H13 | Exportações CSV/PDF (cronogramas + Explain) | E3 - UI |
| H21 | Snapshots com hash e motorVersion | E4 - Infra |
| H22 | Validador (/api/validate/schedule) — upload CSV e diffs por coluna | E4 - Infra |

### Critérios de Saída (Sprint 2 "Pronto"):

* ✅ Price/SAC com deltas ≤ R$0,01 vs GFs
* ✅ Day count (30/360, ACT/365) correto + pró‑rata opcional
* ✅ CET básico (IRR mensal → CET a.a., sem IOF/seguros) com erro ≤ 0,01 p.p. + Explain
* ✅ CSV/PDF consistentes; snapshots salvos ({inputs, outputs, hash, motorVersion})
* ✅ Validador usável com diffs por coluna

**Nota (CET nesta sprint):** BÁSICO = apenas tarifas t0. Definição e método em Guia CET — SoT (§2–4, §8–9).

### Riscos & Mitigações (Sprint 2):

* ⚠️ Divergência de CET entre bancos: não cobrir IOF/seguros aqui; Sprint 3 terá Perfis
* ⚠️ PDF frágil: basear no Explain e stylesheet único do Design System

## 🚀 SPRINT 3: NPV/TIR ROBUSTO + CET COMPLETO + PERFIS + COMPARADOR + XLSX + CASOS

**Objetivo:** Solver Brent (IRR), CET completo (IOF/seguros/perfis), comparador, XLSX com fórmulas e 3 casos gabaritados.

### Histórias da Sprint 3:

|  |  |  |
| --- | --- | --- |
| ID | Descrição | Épico |
| H14 | NPV (VPL) | E1 - Engine |
| H15 | IRR (TIR) com Brent + fallbacks e alerta de múltiplas raízes | E1 - Engine |
| H16 | CET completo (IOF diário + adicional + seguros) — Guia CET SoT | E2 - CET |
| H17 | Perfis de CET por instituição (versionados) | E2 - CET |
| H18 | Comparador de cenários (menor CET/total pago + justificativa) | E5 - Academy |
| H19 | XLSX com fórmulas | E3 - UI |
| H23 | Casos de Mercado gabaritados | E5 - Academy |
| H24 | Acessibilidade & E2E | E3/E4 |

### Critérios de Saída (Sprint 3 "Pronto"):

* ✅ IRR Brent com erro ≤ 0,01%, logs de método e iterações
* ✅ CET completo e Perfis versionados; 3 cenários com erro ≤ 0,01 p.p. a.a.
* ✅ Comparador destacando melhor oferta com explicação
* ✅ XLSX coerente com CSV/PDF (mesmos números)
* ✅ 3 Casos anonimizados e publicados como GFs

**Nota (CET nesta sprint):** COMPLETO = tarifas + IOF (diário + adicional) + seguros e Perfis por instituição (vigência). Ver Guia CET — SoT (§5–7).

### Riscos & Mitigações (Sprint 3):

* ⚠️ Perfis defasados: versionamento + documento de divergências obrigatório
* ⚠️ IRR instável: intervalo inicial e fallback; alerta de múltiplas raízes

# PARTE 4: CHECKLISTS OPERACIONAIS

## 📅 CHECKLIST DIÁRIO (Durante a Sprint)

* [ ] Limpei backups físicos hoje?
* [ ] Fiz commits locais das mudanças?
* [ ] npm run test passou localmente?
* [ ] Atualizei JSDoc nas funções novas?
* [ ] OpenAPI atualizado (se novo endpoint)?
* [ ] Golden Files impactados identificados?

**⚠️ Se alguma resposta for "n" → CORRIGIR ANTES DE CONTINUAR**

## 📋 CHECKLIST DE HISTÓRIA (Ao Completar H9, H10, etc.)

### Implementação:

* [ ] Motor implementado (engine/src/)
* [ ] Testes unitários (≥3 casos)
* [ ] Testes de propriedade (invariantes)
* [ ] Golden File criado/atualizado
* [ ] API implementada (se aplicável)
* [ ] Validação Zod (se aplicável)

### Documentação:

* [ ] JSDoc em funções públicas
* [ ] OpenAPI atualizado (se API)
* [ ] ADR criado (se decisão estrutural)
* [ ] Changelog da história no commit

### Qualidade:

* [ ] npm run test passa
* [ ] npm run lint passa
* [ ] npm run build passa
* [ ] Golden Files passam
* [ ] Cobertura mantida (≥80%)

### Git:

* [ ] Commit local feito
* [ ] Mensagem convencional (feat/fix/docs)
* [ ] Nenhum .bak commitado

**✅ Se todos [x] → HISTÓRIA CONCLUÍDA!**

## 🏁 CHECKLIST FINAL DE SPRINT (Antes do Push)

### Validação Anti-Regressão:

* [ ] npm run type-check: PASSOU
* [ ] npm run lint: PASSOU
* [ ] npm run test:unit: PASSOU
* [ ] npm run test:property: PASSOU
* [ ] npm run test:integration: PASSOU
* [ ] npm run golden:verify: PASSOU (10/10)
* [ ] npm run build: PASSOU
* [ ] npm run test:e2e: PASSOU
* [ ] Swagger UI: ACESSÍVEL

### Documentação:

* [ ] README atualizado
* [ ] OpenAPI sincronizado
* [ ] ADRs criados (se necessário)
* [ ] Changelog da sprint documentado

### Golden Files:

* [ ] Novos GFs revisados por especialista
* [ ] motorVersion atualizado (se necessário)
* [ ] Tolerâncias respeitadas (≤0.01)

### Limpeza:

* [ ] Nenhum arquivo .bak no projeto
* [ ] Nenhum console.log não intencional
* [ ] Nenhum TODO/FIXME crítico

### Git:

* [ ] Todos os commits na branch sprint-X
* [ ] Merge na main feito localmente
* [ ] Mensagem de merge descritiva

**✅ Se TODOS [x] → PODE FAZER PUSH PARA GITHUB!**

# PARTE 5: TEMPLATES DE COMMIT

## 📝 COMMIT DE HISTÓRIA (Durante Sprint)

git commit -m "feat(H9): Implementa Price PMT e cronograma  
  
- Motor: calculatePMT com decimal.js  
- API: POST /api/price  
- Testes: 8 unitários + 3 propriedade + 5 golden files  
- Ajuste final: saldo\_n <= 0.01  
- Cobertura: 87%  
  
DoD: 5/5 critérios atendidos  
  
Referências: Guia CET — SoT (§4), ADR-001"

## 🔧 COMMIT DE CORREÇÃO (Durante Sprint)

git commit -m "fix(H9): Corrige arredondamento em ajuste final  
  
- Problema: resíduo de 0.02 em alguns casos  
- Solução: aplicar round2 antes de comparação  
- Testes: adiciona PRICE\_006 (caso extremo)  
- Golden Files: 6/6 passam agora  
  
Closes: bug identificado em teste de propriedade"

## 📄 COMMIT DE DOCUMENTAÇÃO (Durante Sprint)

git commit -m "docs(H9): Adiciona JSDoc em calculatePMT  
  
- Documenta parâmetros e retorno  
- Adiciona exemplo de uso  
- Atualiza OpenAPI para POST /api/price  
  
Referências: Contratos de API (§1)"

## 🔀 COMMIT DE MERGE (Final da Sprint)

git commit -m "chore: Merge Sprint 2  
  
Histórias implementadas:  
- H9: Price (PMT/Cronograma/Ajuste Final)  
- H10: Day Count (30/360, ACT/365, pró-rata)  
- H11: SAC (Cronograma)  
- H12: CET Básico (tarifas t0)  
- H13: Exportações (CSV/PDF)  
- H21: Snapshots (hash + motorVersion)  
- H22: Validador (upload CSV + diffs)  
  
Validação anti-regressão: ✅ PASSOU  
- Type Check: ✅  
- Lint: ✅  
- Unit Tests: ✅ (85% cobertura)  
- Property Tests: ✅  
- Integration Tests: ✅  
- Golden Files: ✅ (10/10 verdes)  
- Build: ✅  
- E2E: ✅  
- Swagger UI: ✅  
  
Golden Files novos: PRICE\_001 a PRICE\_005, SAC\_001 a SAC\_005  
motorVersion: 0.2.0 (atualizado)  
Especialista: aprovado em 2025-10-15  
  
Próxima sprint: Sprint 3 (H14-H19, H23)"

# PARTE 6: PROIBIÇÕES ABSOLUTAS E OBRIGATORIEDADES

## ❌ NUNCA FAZER

### NUNCA criar backups físicos

cp file.ts file.ts.bak # ❌ PROIBIDO

### NUNCA fazer push durante a sprint

git push origin sprint-2 # ❌ PROIBIDO (só ao final!)

### NUNCA commit sem testes

git commit -m "fix: corrige bug" # ❌ SEM TESTES = PROIBIDO

### NUNCA push sem validação completa

git push origin main # ❌ SEM VALIDAÇÃO = PROIBIDO

### NUNCA usar float para cálculos

const result = 10000 \* 0.025; # ❌ PROIBIDO

### NUNCA duplicar lógica de CET

// Implementar CET aqui... # ❌ PROIBIDO

### NUNCA commit de Golden File sem revisão

git add PRICE\_001.json && git commit # ❌ SEM REVISÃO = PROIBIDO

## ✅ SEMPRE FAZER

### SEMPRE limpar backups físicos diariamente

find . \( -name "\*bak\*" -o -name "\*backup\*" \) -type f -delete

### SEMPRE commits locais frequentes

git add . && git commit -m "feat(H9): Implementa PMT"

### SEMPRE testar antes de commit

npm run test && git add . && git commit

### SEMPRE validação completa antes de push final

./finalizar-sprint.sh # Executa todas as validações

### SEMPRE usar decimal.js para cálculos

import { toDecimal, round2 } from '@/decimal';

### SEMPRE consultar Guia CET para mudanças de CET

# Ver: docs/cet-sot/guia-cet-v1.0.md

### SEMPRE documentar durante implementação

/\*\* @param {Decimal} pv - Valor presente \*/

# PARTE 7: CASOS ESPECIAIS

## 🚨 CASO 1: HOTFIX CRÍTICO (Fora da Sprint)

# Se houver bug crítico que exige fix imediato:  
  
# 1. Criar branch de hotfix  
git checkout main  
git pull origin main  
git checkout -b hotfix-critical-bug  
  
# 2. Implementar correção mínima  
# ... código ...  
  
# 3. Validação anti-regressão  
npm run test  
npm run golden:verify  
npm run build  
  
# 4. Push direto (exceção!)  
git add .  
git commit -m "hotfix: Corrige bug crítico de precisão em CET  
  
- Problema: arredondamento incorreto em IOF diário  
- Solução: aplicar round2 após soma de fluxos  
- Impacto: CET\_002 atualizado (delta < 0.01 restaurado)  
- Golden Files: 3/3 passam  
  
Urgência: crítico (afeta cálculos de clientes)  
Validação: ✅ completa  
Revisor: especialista Mat. Financeira"  
  
git push origin hotfix-critical-bug  
  
# 5. Abrir PR imediato para review  
# 6. Merge após aprovação

## 🧪 CASO 2: GOLDEN FILE NÃO PASSA (Durante Sprint)

# Se golden:verify falha:  
  
# 1. Identificar qual GF falhou  
npm run golden:verify  
# Output: PRICE\_001 FALHOU (esperado: 946.56, encontrado: 946.57)  
  
# 2. Investigar causa  
# - Mudança intencional de algoritmo? → Atualizar GF  
# - Bug introduzido? → Corrigir código  
  
# 3. Se mudança intencional:  
npm run golden:recalc -- --file=PRICE\_001  
# Revisar com especialista  
# Aprovar novo GF  
# Atualizar motorVersion se necessário  
  
# 4. Se bug:  
# Corrigir código  
npm run test  
npm run golden:verify # Deve passar agora  
git add . && git commit -m "fix: Corrige cálculo PMT"

## 🔀 CASO 3: CONFLITO DE MERGE (Início de Sprint)

# Se houver conflito ao fazer git pull:  
  
# 1. Abortar merge  
git merge --abort  
  
# 2. Fazer backup local  
git branch backup-local-$(date +%Y%m%d)  
  
# 3. Forçar sincronização  
git fetch origin  
git reset --hard origin/main  
  
# 4. Reaplicar mudanças locais (se necessário)  
# Manualmente, com cuidado  
  
# 5. Validar  
npm run test  
npm run golden:verify

# PARTE 8: MÉTRICAS DE PROCESSO

## 📊 KPIs DE WORKFLOW (Por Sprint)

# Ao final da sprint, gerar relatório:  
echo "📊 MÉTRICAS DE WORKFLOW - SPRINT X"  
echo ""  
echo "Commits:"  
echo " - Total: $(git log sprint-X --oneline | wc -l)"  
echo " - Por história: $(git log sprint-X --oneline | grep 'feat(' | wc -l)"  
echo " - Correções: $(git log sprint-X --oneline | grep 'fix(' | wc -l)"  
echo ""  
echo "Qualidade:"  
echo " - Cobertura: $(npm run test:coverage | grep 'All files' | awk '{print $10}')"  
echo " - Golden Files: $(npm run golden:verify | grep 'PASS' | wc -l)/10"  
echo " - Lint warnings: $(npm run lint 2>&1 | grep 'warning' | wc -l)"  
echo ""  
echo "Tempo:"  
echo " - Início: [data inicial]"  
echo " - Fim: [data final]"  
echo " - Duração: X dias"  
echo ""

## 🎯 MÉTRICAS POR TIPO DE HISTÓRIA

|  |  |  |
| --- | --- | --- |
| Tipo de História | Métrica Crítica | Tolerância |
| Cálculos (H4-H6, H9-H11, H14-H17) | Monetário / Taxas | ≤ R$0,01 / ≤ 0,01 p.p. |
| CET (H12, H16-H18) | CET a.a. | ≤ 0,01 p.p. (básico) / ≤ 0,03 p.p. (completo) |
| Performance (Todos) | P95 cálculo / Exportação | ≤ 150 ms / ≤ 2 s |
| Qualidade (Todos) | Cobertura / Builds | ≥ 80% / 100% verdes |

# PARTE 9: RESUMO E REFERÊNCIAS

## 📚 DOCUMENTOS DE REFERÊNCIA (Fonte de Verdade)

* **CET (metodologia):** Guia CET Completo — SoT (v1.0) [§2–7, §8–9]
* **Testes & QA:** Playbook de Testes & Golden Files (v1.0)
* **Design & UI:** Design System / UI Kit (v1.0)
* **Contratos de API:** Contratos de API — FinMath (v1.0) / OpenAPI 3.1
* **Decisões Arquiteturais:** ADRs (v1.0): ADR-001 (Decimal), ADR-002 (IRR/Brent), ADR-004 (Perfis), ADR-007 (Exportações)
* **Backlog Detalhado:** Backlog Detalhado — Projeto (v1.1)
* **Boards Operacionais:** Boards por Sprint (S1–S3) v1.1
* **Perfis CET:** Divergências de Perfis CET (v1.0)
* **Glossário:** Glossário & Convenções (v1.0)

## 🔗 NAVEGAÇÃO RÁPIDA (Tarefas Comuns)

* **Implementar Price/SAC:** Backlog H9/H11, Contratos de API (/price, /sac), Playbook (ajuste final), Design System (tabela)
* **Calcular CET:** Guia CET — SoT (§2–7), Backlog H12/H16/H17, Contratos (/cet/basic, /cet/full, /cet/profiles)
* **Resolver IRR:** Backlog H15, ADR-002, Contratos (/npv-irr), Playbook (matriz IRR)
* **Exportar CSV/PDF/XLSX:** Backlog H13/H19, Contratos (/reports/\*), Playbook (exportações), Design System
* **Validar cronograma externo:** Backlog H22, Contratos (/validate/schedule), Playbook (diff por coluna)
* **Atualizar perfis CET:** Guia CET — SoT (§7), ADR-004, Contratos (/cet/profiles), Divergências de Perfis

# 📋 RESUMO EXECUTIVO: COBERTURA COMPLETA

## ✅ SPRINTS COBERTAS

* Sprint 1 - Fundamentos: 9 histórias (H1-H8, H20)
* Sprint 2 - Amortizações + CET Básico: 7 histórias (H9-H13, H21-H22)
* Sprint 3 - CET Completo + Perfis: 8 histórias (H14-H19, H23-H24)

**Total: 24 Histórias de Usuário completamente documentadas**

## ✅ ÉPICOS COBERTOS

* E1 - Engine (Core): H2, H4-H6, H9-H11, H14-H15
* E2 - CET: H12, H16-H17
* E3 - UI/UX: H7-H8, H13, H19, H24
* E4 - Infraestrutura: H1, H3, H21-H22, H24
* E5 - Academy/Mercado: H18, H20, H23

## ✅ GOLDEN FILES PLANEJADOS

* Sprint 1: 10 GFs (3 juros + 3 equivalência + 4 séries)
* Sprint 2: 10 GFs (5 Price + 5 SAC)
* Sprint 3: 10 GFs (5 NPV/IRR + 5 CET)
* Total: 30 Golden Files para validação completa

## ✅ ENDPOINTS DE API PLANEJADOS

* Sprint 1: /api/interest, /api/rate, /api/series
* Sprint 2: /api/price, /api/sac, /api/cet/basic, /api/validate/schedule
* Sprint 3: /api/npv-irr, /api/cet/full, /api/cet/profiles, /reports/\*
* Total: ~15 endpoints RESTful documentados no OpenAPI

## 🎯 PRINCÍPIOS FUNDAMENTAIS DO WORKFLOW

**✅ GitHub como Fonte da Verdade:** Sincronização obrigatória no início de cada sprint. Push coletivo apenas ao final.

**✅ Validação Anti-Regressão:** 8 etapas de validação obrigatória antes de cada push (type-check, lint, tests, golden files, build, e2e, swagger).

**✅ Backup Exclusivo via Git:** Proibido criar arquivos .bak/.backup/.save. Limpeza diária automática.

**✅ Documentação Viva:** JSDoc, OpenAPI, ADRs e Changelog atualizados durante implementação, não depois.

**✅ Precisão Matemática:** Uso obrigatório de decimal.js. Tolerâncias: ≤ R$0,01 (monetário), ≤ 0,01 p.p. (taxas).

**✅ Golden Files como Contrato:** Revisão por especialista obrigatória. Freeze com motorVersion. Apenas 1 fonte de verdade.

**✅ CET: Guia SoT:** Metodologia CET exclusivamente no Guia CET Completo — SoT. Zero duplicação de lógica.

**✅ Commits Convencionais:** feat/fix/docs/chore com referências a histórias, ADRs e documentos relevantes.

# 📝 CHANGELOG

## Versão 2.0 Expandida (23/10/2025)

* ✅ Cobertura completa das 3 sprints (S1, S2, S3)
* ✅ Detalhamento de todas as 24 histórias de usuário
* ✅ Workflows específicos por fase (início, durante, fim)
* ✅ Checklists operacionais (diário, por história, final de sprint)
* ✅ Templates de commit para todos os casos
* ✅ Seção completa de proibições e obrigatoriedades
* ✅ Casos especiais documentados (hotfix, golden files, conflitos)
* ✅ Métricas de processo e KPIs por tipo de história
* ✅ Navegação rápida e referências cruzadas
* ✅ Estrutura profissional em documento Word

## Versão 1.0 (11/10/2025)

* 🚀 Criação inicial
* 📋 Adaptação para workflow "push ao final da sprint"
* ✅ Regras adaptadas: trabalho local + validação final
* 🔄 Scripts: inicio-sprint.sh + finalizar-sprint.sh
* 📝 Templates de commit específicos
* 🚨 Proibições e casos especiais documentados

# 🎓 CONCLUSÃO

Este documento estabelece as regras críticas e o workflow completo para o desenvolvimento do projeto FinMath, garantindo:

**1. Qualidade: Validação anti-regressão obrigatória impede que bugs cheguem ao GitHub.**

**2. Rastreabilidade: Commits convencionais e documentação viva garantem histórico completo.**

**3. Precisão: Golden Files revisados por especialistas asseguram correção matemática.**

**4. Consistência: Todos seguem o mesmo processo, independente da sprint ou história.**

**5. Eficiência: Workflow otimizado reduz trabalho manual e automação de validações.**

As 24 histórias de usuário estão completamente mapeadas nas 3 sprints, com critérios de aceite, riscos, mitigações e validações bem definidos.

Seguir rigorosamente estas regras garante que o projeto FinMath alcance seus objetivos de qualidade, precisão e confiabilidade.

────────────────────────────────────────────────────────────────────────────────

*📘 Documento mantido em: docs/workflow/REGRAS\_CRITICAS\_FINMATH\_v2.0.docx*

**Projeto FinMath - Academy • Lab • Mercado**