

Resumo — Framework aplicado para QAs

O que é

Aplicação do `edenred-copilot-context-framework` com foco em **QA/testes** (`--qa`), em vez de desenvolvimento. QA recebe agentes especializados que leem requisitos/código como contexto e **produzem artefatos de teste** (casos de teste, bug reports, análise de gaps de cobertura) — os agentes **não geram nem alteram código de aplicação**.

Setup

```
npx --yes copilot-framework@latest setup --workspace --qa
```

- Pode combinar com uma stack (`--stack angular --qa`) para contexto adicional, ou usar `--workspace --qa` de forma agnóstica ao codebase.
- Também pode ser combinado com `--product-owner` no mesmo setup, se o workspace precisar dos dois papéis.
- Gera: `.github/agents/` , `.github/prompts/` , `.vscode/mcp.json` e o guia de uso em `docs/` .

Estrutura gerada

<code>.github/agents/</code>	
<code>qa-test-case-generation.agent.md</code>	→ requisitos/AC → casos de teste estruturados
<code>qa-bug-report.agent.md</code>	→ defeitos observados → bug report reproduzível
<code>qa-test-plan-review.agent.md</code>	→ plano/suite existente → análise de gaps
<code>.github/prompts/</code>	
<code>qa-agents.prompt.md</code>	→ roteia o pedido para o agente certo
<code>.vscode/mcp.json</code>	→ perfil MCP opcional do Azure DevOps (contexto de boards)

Os 3 agentes

1. qa-test-case-generation

- **Entrada:** user story/AC, trecho de spec, descrição de feature + prints/fluxo de UI.
- **Saída:** lista de casos de teste (ID, título, pré-condições, passos, resultado esperado, prioridade), resumo de cobertura (positivo/negativo/edge/boundary), rastreabilidade para o requisito de origem, perguntas em aberto quando o AC é ambíguo.
- **Regras:** não altera código; separa casos funcionais de não funcionais (performance, acessibilidade, segurança).

2. qa-bug-report

- **Entrada:** descrição livre de comportamento inesperado, passos já executados + esperado vs. real, logs/stack traces/erros.
- **Saída:** título, ambiente/contexto, passos para reproduzir, resultado esperado x real, severidade/prioridade com justificativa, checklist de evidências, perguntas em aberto se a reprodução estiver incompleta.
- **Regras:** não especula causa raiz no código; linguagem neutra/factual; sinaliza quando um mesmo sintoma pode mapear para múltiplos defeitos.

3. qa-test-plan-review

- **Entrada:** plano/suite de testes existente, requisitos + cobertura atual, áreas de risco ou incidentes anteriores.
- **Saída:** lista de gaps de cobertura, priorização por risco, notas de redundância/consolidação, tipos de teste adicionais recomendados (regressão, exploratório, acessibilidade, performance), perguntas em aberto.
- **Regras:** não reescreve o plano inteiro (só gap analysis); distingue gaps "bloqueiam release" de "seria bom ter"; não propõe mudanças de código.

Como usar no dia a dia

1. Abrir o repositório alvo (ou um workspace dedicado) no VS Code.
2. No Copilot Chat, selecionar o agente adequado no seletor de agentes.
3. Descrever o pedido em linguagem natural (colar a user story, ACs, logs ou plano de testes existente).
4. Revisar a saída — cada agente sinaliza suposições e perguntas em aberto em vez de "adivinhar".
5. Copiar o resultado para a ferramenta de gestão de testes/bugs do time — o agente não publica nada automaticamente.

MCP do Azure DevOps (opcional)

- **Padrão recomendado:** MCP remoto (`azure-devops` , `type: "http"` , `https://mcp.dev.azure.com/{organization}`) — hospedado pela Microsoft, sem instalação local, autenticação via Microsoft Entra ID.
- **Fallback local:** `azure-devops-local-fallback` (stdio, via `@azure-devops/mcp`) — só se o cliente não conseguir autenticar no servidor remoto. Domínios habilitados: `core` , `work-items` , `search` , `repositories` .
- É **somente leitura** — contexto para os agentes, sem acesso de escrita a boards/repos.
- `--qa` funciona sem conexão com Azure DevOps; o perfil MCP é contexto opcional.

Referência

Guia completo: [qa-workspace-usage-guide.md](#)