INTEGRAÇÃO CONTÍNUA EM PROJETOS DE SOFTWARE

I. INTRODUÇÃO

A Integração Contínua (*Continuous Integration* - CI) é uma prática de desenvolvimento de software na qual os desenvolvedores integram com frequência seu código em um repositório compartilhado. O objetivo é detectar erros rapidamente, promovendo um ciclo de desenvolvimento mais ágil e confiável. Segundo Martin Fowler, CI não é apenas sobre rodar um servidor de integração, mas sim sobre adotar uma disciplina de integração contínua e frequente, mantendo o sistema sempre funcional (FOWLER, 2006).

II. OBJETIVO DA CI NO PROJETO

No projeto da estação meteorológica, a CI visa garantir que todas as atualizações realizadas na "*branch develop"* sejam automaticamente verificadas quanto à qualidade, consistência e integridade. A ferramenta GitHub *Actions* foi utilizada para configurar uma pipeline automatizada, incluindo verificação de formatação com *"ruff"* e análise estática com "*mypy*".

III. FLUXO DE DESENVOLVIMENTO

Adotamos uma estratégia de *branches* para manter a organização e a qualidade do código. O fluxo compreende:

a. *Branch "main"*: versão estável em produção.

b. *Branch "develop"*: ambiente principal de desenvolvimento.

c. *Branches de "feature" e "hotfix"*, nomeadas conforme os padrões:

d. *feature*/#RF_id/#task_id-descricao_tarefa

e. *hotfix*/#RF_id/#task_id-descricao_tarefa

O fluxo inclui atribuição de tarefas, testes locais, *commits* padronizados, *pull requests*, execução de *pipelines*, revisão de código e *merge*.

IV. RASTREABILIDADE DE REQUISITOS

A rastreabilidade de requisitos conecta funcionalidades implementadas aos requisitos documentados, garantindo alinhamento e controle do ciclo de desenvolvimento. São classificados como RF (Funcionais) ou RNF (Não-Funcionais) e referenciados no *backlog* e nas tarefas.

V. PADRÃO DE COMMITS E VERSÕES

Os commits seguem padrões definidos com tipos como: *feat, fix, docs, style, refactor, test, chore e build. Releases* são feitas com *tags* na *branch main* (ex: v1.0.0) e associadas aos submódulos, quando necessário.

VI. COMUNICAÇÃO DA EQUIPE

A comunicação foi realizada via *WhatsApp* e *Discord*, com documentos de referência disponibilizados na descrição do grupo, incluindo diretrizes *DevOps* e exemplos de rotas.

VII. CONCLUSÃO

A implementação da CI e a estruturação do fluxo de desenvolvimento trouxeram inúmeros benefícios como: maior agilidade, menor risco de falhas e melhor colaboração entre os membros e otimização no trabalho em equipe. O uso de ferramentas como *GitHub Actions, ruff e mypy* contribuiu significativamente para a qualidade do código e confiabilidade do sistema.

REFERÊNCIAS

FOWLER, Martin. Continuous Integration. 2006. Disponível em: https://martinfowler.com/articles/continuousIntegration.html. Acesso em: 03. Abr. 2025.

Os commits seguem padrões definidos com tipos como: feat, fix, docs, style, refactor, test, chore e build. Releases são feitas com tags na branch main (ex: v1.0.0) e associadas aos submódulos, quando necessário.

VIII. COMUNICAÇÃO DA EQUIPE

A comunicação foi realizada via WhatsApp e Discord, com documentos de referência disponibilizados na descrição do grupo, incluindo diretrizes DevOps e exemplos de rotas.

IX. CONCLUSÃO

A implementação da CI e a estruturação do fluxo de desenvolvimento trouxeram benefícios como maior agilidade, menor risco de falhas e melhor colaboração entre os membros. O uso de ferramentas como GitHub Actions, ruff e mypy contribuiu significativamente para a qualidade do código e confiabilidade do sistema.

REFERÊNCIAS

FOWLER, Martin. Continuous Integration. 2006. Disponível