


Estratégia de Controle de Versão	
Nome do Projeto: Rootin	

Controle de Versões			
Versão	Data	Autor	Notas da Revisão
1.0	10/06	Ludmila Oliva	Preenchimento Completo do Documento

1 Objetivos deste documento

Este documento descreve a estratégia de controle de versão a ser adotada para o desenvolvimento do Rootin. Uma estratégia de controle de versão eficaz é fundamental para gerenciar as mudanças no código-fonte, colaborar entre a equipe de desenvolvimento e garantir a rastreabilidade das alterações.

2 Ferramenta de Controle de Versão


Para o projeto Rootin, será utilizado o **Git** como sistema de controle de versão distribuído. O Git é amplamente reconhecido por sua flexibilidade, desempenho e capacidade de suportar fluxos de trabalho complexos de desenvolvimento.

O repositório principal será hospedado em uma plataforma como Github ou GitLab, que oferece recursos adicionais como gerenciamento de issues, revisões de código (pull requests/merge requests) e integração contínua.

3 Organização do Projeto e Matriz de Responsabilidade

Será adotado o fluxo de trabalho **Gitflow**, que é adequado para projetos com ciclos de lançamento bem definidos e que exigem suporte a múltiplas versões em paralelo. O Gitflow define um conjunto de branches com propósitos específicos:

- **main (ou master):** Contém o código de produção, sempre estável e pronto para ser lançado. Apenas merges de release ou hotfix branches são permitidos.


Estratégia de Controle de Versão	
Nome do Projeto: Rootin	

- **develop** : Branch principal de desenvolvimento. Todas as novas funcionalidades são integradas aqui. É a base para as feature branches. (front-end), o sistema de back-end e o módulo de inteligência artificial, podendo causar atrasos ou falhas de funcionalidade.
- **feature branches:** Criadas a partir de develop para desenvolver novas funcionalidades. Cada funcionalidade deve ter sua própria branch. Após a conclusão e revisão, são mergeadas de volta para develop .
- **Convenção de Nomenclatura:** feature/<nome-da-funcionalidade>
(ex: feature/autenticacao-usuario)
- **release branches:** Criadas a partir de develop quando uma nova versão está pronta para ser lançada. Usadas para correções de bugs finais, testes e preparação para o lançamento. Após a estabilização, são mergeadas para main e develop .
Convenção de Nomenclatura: release/<numero-da-versao>
(ex:release/1.0.0)
- **hotfix branches:** Criadas a partir de main para corrigir bugs críticos em produção. Após a correção, são mergeadas de volta para main e develop .
Convenção de Nomenclatura: hotfix/<descricao-do-bug>
(ex:hotfix/erro-login-producao)

4 Boas Práticas de Commit

Para garantir um histórico de commits limpo e significativo, as seguintes boas práticas serão seguidas:

- **Mensagens de Commit Claras:** As mensagens de commit devem ser concisas, descritivas e explicar o propósito da alteração. Utilizar o formato Tipo: Descrição (ex: feat: Adiciona funcionalidade de registro de gastos).

Estratégia de Controle de Versão	
Nome do Projeto: Rootin	


- **Tipos Comuns:** feat (nova funcionalidade), fix (correção de bug), docs (documentação), style (formatação, sem mudança de código), refactor (refatoração de código), test (testes), chore (tarefas de manutenção).
- **Commits Atômicos:** Cada commit deve conter uma única alteração lógica. Evitar commits grandes que misturam diferentes funcionalidades ou correções.
- **Frequência de Commits:** Realizar commits pequenos e frequentes para facilitar a revisão e o rastreamento de problemas.

5 Processo de Revisão de Código (Pull Requests/ Merge Requests)

Todas as alterações significativas no código-fonte, especialmente aquelas que serão mergeadas para develop ou main , deverão passar por um processo de revisão de código. Isso será feito através de Pull Requests (no GitHub) ou Merge Requests (no GitLab).

O processo incluirá:

- **Criação do PR/MR:** O desenvolvedor cria um PR/MR da sua feature branch para a develop branch (ou hotfix para main).
- **Revisão:** Outros membros da equipe revisam o código, verificam a qualidade, a conformidade com os padrões e a lógica.
- **Testes:** Testes automatizados (se houver) são executados. Testes manuais podem ser solicitados.
- **Aprovação e Merge:** Após a aprovação de um número mínimo de revisores e a resolução de quaisquer comentários, o PR/MR é mergeado.

Estratégia de Controle de Versão	
Nome do Projeto: Rootin	

6 Versionamento Semântico

Será adotado o Versionamento Semântico (SemVer) para numerar as versões do aplicativo. O formato MAJOR.MINOR.PATCH será utilizado:

- **MAJOR:** Incrementado para mudanças incompatíveis de API.
- **MINOR:** Incrementado para novas funcionalidades que são compatíveis com versões anteriores.
- **PATCH:** Incrementado para correções de bugs compatíveis com versões anteriores.

7 Tags e Lançamentos

Tags serão utilizadas no Git para marcar pontos específicos no histórico do repositório, geralmente para indicar lançamentos de versões. Cada lançamento oficial (produção) será marcado com uma tag no branch main correspondente à versão SemVer (ex: v1.0.0).

Aprovações		
Participante	Assinatura	Data
Patrocinador do Projeto		
Gerente do Projeto	Luís Gustavo Pereira de Sá	14/06/2025