Plano de Gerência de Configuração de Software – SIGAUNI

# 1. Introdução

## 1.1 Objetivo do Plano

Este plano estabelece os procedimentos e responsabilidades para o controle de itens de configuração, versionamento, auditorias e mudanças no projeto SIGAUNI – Sistema Integrado de Gestão Acadêmica Universitária, como parte da avaliação da disciplina de Gerência de Configuração de Software. O objetivo é aplicar na prática os conceitos estudados, utilizando ferramentas como Git e GitHub para garantir organização, rastreabilidade e consistência ao longo do desenvolvimento do sistema.

## 1.2 Escopo do Projeto

O projeto SIGAUNI é um sistema web desenvolvido para a UniRV, com o objetivo de otimizar a gestão acadêmica de alunos e professores. O sistema inclui funcionalidades como consulta de notas e faltas, solicitação de documentos acadêmicos, lançamento de avaliações por professores e gerenciamento de processos acadêmicos pela secretaria. O sistema será acessado por três perfis de usuários: Alunos, Professores e Administradores (secretaria e coordenação).

## 1.3 Justificativa da GCS

A aplicação da Gerência de Configuração é fundamental neste projeto para garantir controle sobre a evolução do sistema, principalmente em um ambiente simulado de equipe com múltiplos papéis. A GCS permite maior organização, padronização nos processos de versionamento e rastreabilidade das alterações, além de facilitar revisões, auditorias e a entrega de versões consistentes do sistema. Também proporciona ao grupo aprendizado prático das boas práticas de engenharia de software.

# 2. Estrutura Organizacional e Papéis

## 2.1 Composição da Equipe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome | RA | GitHub | Papel no Projeto | Papel na GCS |
| Daniel Faria do Carmo | 000000000 | @danifc123 | Desenvolvedor Full Stack | Gerente de Configuração, Controlador de Mudança, Auditor |

## 2.2 Responsabilidades Específicas

Gerente de Configuração:  
- Planejar e manter atualizado o plano de GCS.  
- Controlar as versões estáveis com uso de tags e changelogs.  
- Realizar auditorias periódicas dos artefatos no repositório.  
  
Controlador de Mudança:  
- Criar e documentar solicitações de mudança via GitHub Issues.  
- Aprovar ou rejeitar mudanças com base nos critérios definidos.  
- Garantir rastreabilidade entre issues, commits e pull requests.  
  
Auditor:  
- Verificar a consistência dos itens de configuração antes de releases.  
- Validar a estrutura de diretórios e documentos obrigatórios.  
- Gerar relatórios de status com base nas atividades semanais.

# 3. Ferramentas e Ambientes

## 3.1 Principais Ferramentas Usadas

|  |  |  |
| --- | --- | --- |
| Ferramenta | Finalidade | Responsável |
| Git | Controle de versão local | Daniel Faria do Carmo |
| GitHub | Repositório remoto, gerenciamento de issues e PRs | Daniel Faria do Carmo |
| VS Code | Edição e desenvolvimento do código-fonte | Daniel Faria do Carmo |
| Postman | Testes de API | Daniel Faria do Carmo |
| PostgreSQL | Banco de dados | Daniel Faria do Carmo |
| Angular | Framework frontend | Daniel Faria do Carmo |

## 3.2 Estrutura do Repositório

A estrutura do repositório seguirá o padrão abaixo para refletir a separação entre frontend e backend, adotando boas práticas de organização para projetos fullstack:  
  
📦 sigauni  
├── frontend/ → aplicação Angular (interface do usuário)  
│ └── src/ → código-fonte do Angular  
├── backend/ → API (Node.js, .NET, etc.)  
│ └── src/ → código-fonte do backend  
├── docs/ → plano de GC, evidências, artefatos  
├── releases/ → versões entregues do sistema  
├── testes/ → scripts de testes e validação  
├── .gitignore → exclusão de arquivos desnecessários  
├── CHANGELOG.md → histórico de mudanças  
├── README.md → descrição geral do projeto

# 4. Identificação de Itens de Configuração

## 4.1 Itens Controlados

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo | Nome | Localização | Status | Responsável |
| Código | login.component.ts | /frontend/src/app/login/ | Em progresso | Daniel Faria do Carmo |
| Código | usuario.controller.ts | /backend/src/controllers/ | Planejado | Daniel Faria do Carmo |
| Código | database.config.ts | /backend/src/config/ | Concluído | Daniel Faria do Carmo |
| Documento | plano-gcs.docx | /docs/ | Em edição | Daniel Faria do Carmo |
| Documento | evidencias-semana1.png | /docs/evidencias/ | Coletando | Daniel Faria do Carmo |
| Documento | changelog.md | / | Em construção | Daniel Faria do Carmo |

## 4.2 Convenções de Nomes

- Branches:  
 - feature/nome-da-feature → para novas funcionalidades  
 - fix/nome-do-bug → para correções  
 - hotfix/urgente → para correções críticas em produção  
  
- Commits (padrão Conventional Commits):  
 - feat(login): adiciona tela de login  
 - fix(api): corrige erro de conexão com banco  
 - docs(gcs): adiciona evidências no plano  
  
- Tags:  
 - v1.0.0, v1.1.0 seguindo o padrão SemVer

# 5. Controle de Versão

## 5.1 Estratégia de Branching

Será utilizada a estratégia GitHub Flow, com os seguintes branches:  
- main: branch principal com o código estável  
- dev: branch de desenvolvimento (opcional)  
- feature/\*: funcionalidades em desenvolvimento  
- fix/\*: correções de bugs  
- hotfix/\*: correções urgentes

## 5.2 Commits e Pull Requests

Todos os commits seguirão o padrão Conventional Commits:  
- feat(login): adiciona tela de autenticação  
- fix(api): corrige erro de conexão  
  
Cada Pull Request (PR) será aberto a partir de uma branch e:  
- Referenciará uma Issue correspondente  
- Será revisado antes do merge (mesmo que por mim)  
- Será aceito apenas se cumprir os critérios definidos

# 6. Controle de Mudança

## 6.1 Processo de Mudança

Mudanças serão registradas via Issues no GitHub. O processo:  
1. Criar uma Issue descrevendo a mudança  
2. Discutir e definir critérios de aceitação  
3. Criar uma branch relacionada  
4. Implementar a mudança  
5. Criar PR referenciando a Issue  
6. Fazer merge e encerrar a Issue

## 6.2 Template de Issue

### Mudança Solicitada  
\*\*Tipo:\*\* [bug, melhoria, nova funcionalidade]  
\*\*Descrição:\*\* Descrever detalhadamente  
\*\*Impacto:\*\* Médio  
\*\*Critérios de Aceitação:\*\*  
- [ ] Item 1  
- [ ] Item 2

# 7. Auditoria e Status de Configuração

## 7.1 Auditorias

- Código: revisado a cada PR  
- Documentos: validados semanalmente  
- Estrutura: conferida antes de cada release

## 7.2 Relatório de Status

# Semana 2  
- Commits: 14  
- Issues abertas: 2  
- Releases: v0.2.0  
- Problemas: conflito de login resolvido no PR #7

# 8. Linhas de Base (Baselines) e Releases

## 8.1 Marcos e Releases

|  |  |  |  |
| --- | --- | --- | --- |
| Marco | Tag | Descrição | Data prevista |
| Estrutura inicial do projeto | v0.1.0 | Setup do repositório, estrutura de pastas, README | 16/06/2025 |
| Funcionalidade: Tela de Login | v0.2.0 | Tela de login implementada no frontend Angular | 18/06/2025 |
| Backend: Endpoint de autenticação | v0.3.0 | Criação do controller e autenticação no backend | 20/06/2025 |
| Primeira versão funcional integrada | v1.0.0 | Integração front-back com login funcionando | 23/06/2025 |

## 8.2 Critérios para Nova Baseline

- Todos os Pull Requests fechados  
- Documentação atualizada  
- Estrutura do repositório revisada  
- Tag criada seguindo o padrão SemVer

# 9. Rastreabilidade

## 9.1 Como conectar itens?

A rastreabilidade será feita conectando Issues, branches, commits e Pull Requests. Exemplo:  
- Issue #4: Criar tela de login  
- Branch: feature/login-page  
- Commit: feat(login): cria estrutura da tela de login  
- Pull Request #7: resolve a Issue #4  
- Resultado: PR mergeado e tag v0.2.0 criada

## 9.2 Matriz de Rastreabilidade

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requisito | Issue | Branch | PR | Status |
| Cadastro de Usuário | #3 | feature/signup | #6 | Concluído |
| Tela de Login (Frontend) | #4 | feature/login-page | #7 | Concluído |
| Autenticação no Backend | #5 | feature/auth-backend | #8 | Em andamento |
| Integração front-back login | #6 | feature/login-integra | - | Planejado |

# 10. Evidências

## 10.1 O que capturar?

- Print do repositório inicial (estrutura de pastas e arquivos)  
- Commits com mensagens padronizadas (Conventional Commits)  
- Screenshot de uma Issue aberta e fechada  
- Screenshot de um Pull Request com comentários e merge  
- Imagem da página de releases no GitHub  
- Print do changelog com versões publicadas  
- Print do status semanal (auditorias, progresso, conflitos)

## 10.2 Organização sugerida

📁 docs/  
└── 📁 evidencias/  
 ├── semana-01/  
 │ ├── print-repo-estruturado.png  
 │ ├── print-issue-login.png  
 ├── semana-02/  
 │ ├── print-pr-login.png  
 │ ├── print-tag-v0.2.0.png  
 ├── semana-03/  
 │ ├── print-status-projeto.png  
 │ ├── print-changelog.png

# 11. Lições Aprendidas

## 11.1 Reflexão Individual

Daniel Faria do Carmo – Gerente de Configuração (único integrante)  
Durante a realização deste projeto, aprendi a aplicar na prática os conceitos de Gerência de Configuração de Software. Compreendi melhor a importância do versionamento com Git e da rastreabilidade por meio de Issues e Pull Requests. Tive certa dificuldade inicial com organização de branches e convenções de commits, mas com o tempo consegui padronizar meu fluxo de trabalho. Também percebi como manter o controle de versões ajuda a prevenir retrabalho e melhora a clareza do desenvolvimento.

## 11.2 Conclusão Geral

Apesar de ser um projeto individual, o exercício de simular uma equipe e aplicar os processos completos de GCS foi muito valioso. A experiência reforçou a importância da disciplina e da documentação no desenvolvimento de software, mesmo em projetos menores. Para projetos futuros, manterei esse mesmo rigor desde o início.

# 12. Cronograma

|  |  |  |
| --- | --- | --- |
| Semana | Atividades realizadas | Entregável |
| 1 | Estrutura inicial do repositório, plano GC | plano-gcs.docx, docs/ |
| 2 | Implementação de funcionalidades iniciais | Branch feature/login, PR #1 |
| 3 | Integração frontend e backend | Versão v1.0.0, vídeo e slides |