Gestão e Qualidade de Software

Strong Password

Enrico Aguiar Vrunski - 82210618
Thiago Ferreira Lima Gonçalves - 824156179
Matheus Tognon Siqueira - 824141731
Felipe Soares de Oliveira – 824156311
Kayky Cerquiaro Prado - 822155538

2025-1

ÍNDICE DETALHADO

1. Planejamento de Software

1.1. Cronograma de Atividades

O cronograma de atividades do projeto Strong Password foi planejado de forma a cobrir as principais etapas de testes, garantindo validação progressiva do sistema durante o desenvolvimento. As atividades foram distribuídas conforme as sprints definidas no modelo SCRUM adotado pela equipe.

- Sprint 1 (28/04/2025 a 05/05/2025): Planejamento de testes e definição de critérios.
- Sprint 2 (06/05/2025 a 13/05/2025): Implementação dos primeiros testes funcionais.
- Sprint 3 (14/05/2025 a 21/05/2025): Execução de testes de integração e validação de requisitos.
- Sprint 4 (22/05/2025 a 29/05/2025): Ajustes e reexecução de testes conforme feedback.
- Sprint 5 (30/05/2025 a 06/06/2025): Finalização e testes de usabilidade e desempenho.
- Sprint 6 (07/06/2025 a 11/06/2025): Consolidação dos resultados e documentação final.

1.2. Alocação de Recursos

Os recursos humanos e técnicos foram alocados conforme as necessidades de cada fase do projeto:

- 2 Desenvolvedores: responsáveis pela implementação das funcionalidades e correções identificadas nos testes.
- 1 Analista de Qualidade (QA): elaboração de casos de teste, execução dos testes e registro dos resultados.
- 1 Documentador(a) Técnico: Produção dos documentos técnicos do projeto
- 1 Analista de Segurança: Validação das regras de segurança aplicadas

Ferramentas utilizadas:

- Python (para desenvolvimento do sistema);
- Pytest (execução de testes automatizados);
- VSCode (IDE de desenvolvimento);
- GitHub (controle de versão e colaboração);

Hardware

- 5 Notebooks - Intel i5 ou Ryzen 5, 8GB RAM, SSD 256GB Windows

Licenças e Software

- Python (>=3.7) - Open Source (PSF) - Linguagem de programação principalPytest (execução de testes automatizados);
- VSCode / PyCharm (opcional) - Free / Community - Edição e desenvolvimento
- Git – Open Source – Controle de versão
- Getpass, string(built-in) - Python Standard Lib – Funcionalidade internas do código

Documentos necessários

- Requisitos Funcionais: Regras sobre o que a aplicação deve fazer
- Manual do Usuário (CLI): Orientações para execução via terminal
- Documento de Arquitetura: Explicação da estrutura e lógica do código
- Plano de Testes: Descrição dos testes aplicados
- Termo de Aceite (opcional): Validação do projeto pelo solicitante

Normas e frameworks

- OWASP Top 10: Evitar vulnerabilidades comuns em senhas
- PEP8 (Python Enhancement Proposal): Manter o código legível e padronizado
- Segurança da Informação (ISO/IEC 27001)
- Licença MIT (ou outra) Definição legal do uso do código (se for público)

1.3 Marcos do Projeto

Os principais marcos relacionados à fase de testes do projeto são os seguintes:

- Início da fase de testes: 28/04/2025
- Entrega do primeiro conjunto de casos de teste: 05/05/2025
- Conclusão dos testes funcionais: 13/05/2025
- Finalização dos testes de integração: 29/05/2025
- Validação final com testes de usabilidade e desempenho: 06/06/2025
- Aprovação do sistema para entrega: 11/06/2025

2 Documentos de Desenvolvimento de Software

O objetivo principal deste projeto é desenvolver um sistema capaz de avaliar a força de senhas inseridas por usuários, com base em critérios definidos de complexidade e segurança. O projeto visa educar o usuário sobre a importância de utilizar senhas fortes, ao mesmo tempo em que aplica conceitos de programação, segurança da informação e boas práticas de desenvolvimento.

2.1 Plano do projeto

2.1.1 Planejamento do projeto

O planejamento do projeto tem como objetivo organizar e definir as etapas necessárias para o desenvolvimento do software Strong Password, uma aplicação voltada para a geração, armazenamento e gerenciamento seguro de senhas robustas. A iniciativa busca atender às necessidades de usuários que desejam manter seus dados protegidos contra acessos não autorizados, por meio da criação de senhas aleatórias com alto grau de complexidade. O projeto será dividido em fases, como levantamento de requisitos, modelagem, desenvolvimento, testes e entrega final, com prazos e recursos previamente definidos para garantir a qualidade e o cumprimento dos objetivos propostos.

2.1.2 Escopo

2.1.2.1 Análise de Senha

O sistema receberá uma senha inserida pelo usuário e fará uma análise da sua força com base nos seguintes critérios:

- Tamanho da senha (mínimo de 8 caracteres)
- Uso de letras maiúsculas e minúsculas
- Inclusão de números
- Inclusão de caracteres especiais (ex: @, #, $, %, etc.)
- Detecção de padrões fracos (ex: “123456”, “senha”, “qwerty”)

2.1.2.2 Classificação da Força da Senha

O sistema classificará a senha em categorias como:

- Muito fraca
- Fraca
- Média
- Forte
- Muito forte

2.1.2.3 Interface Simples de Entrada

Uma interface de linha de comando ou web simples onde o usuário poderá digitar a senha.

2.1.2.4 Sugestões de Melhoria

Para senhas fracas ou muito fracas, o sistema fornecerá dicas para torná-las mais seguras.

2.1.3 Recursos

Para o desenvolvimento do software serão utilizados os seguintes recursos:

- Recursos Humanos

Analista de requisitos: responsável pelo mapeamento de requisitos e regras de negócios
Desenvolvedores: responsável pela construção do código
Gerente de projeto: responsável pela gerência do projeto, organização e divisão de tarefas, prazos
Testador de software: responsável pela realização dos testes do software

- Recursos Tecnológicos

Linguagem phyton: linguagem utilizada para o desenvolvimento do software

VSCode: editor de código utilizado para o desenvolvimento)

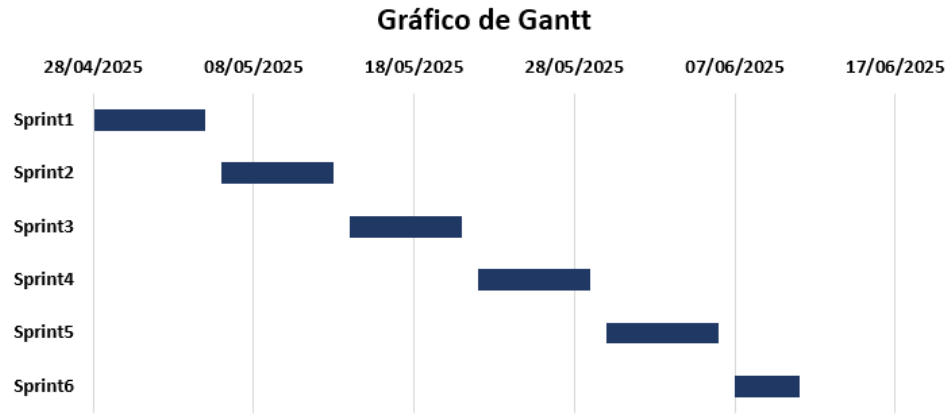
GitHub: utilizado para controle de versão e colaboração em equipe

- Recursos materiais

Computador: computadores pessoais com Python instalados para desenvolvimento do código
Internet: acesso à internet para pesquisas e colaboração em equipe
Ferramenta para Scrum: utilização da ferramenta Projet4me para controle de backlog de tarefas, sprints e gerenciamento do andamento do projeto

2.1.4 Estimativas de Projeto

O projeto será desenvolvido ao longo de um prazo dividido em Sprints com prazos aproximados durando 39 dias. Abaixo estão as estimativas do projeto:



2.2 Documento de Requisitos

Requisitos Funcionais

Os requisitos funcionais definem as funcionalidades que o sistema Strong Password deve apresentar para atender aos objetivos propostos. A seguir, são descritas as principais funcionalidades:
RF01 - O sistema deve permitir ao usuário digitar uma senha para análise.
RF02 A aplicação deve avaliar a força da senha com base em critérios como a presença de letras maiúsculas, letras minúsculas, números, espaços e caracteres especiais.
RF03 O sistema deve atribuir uma pontuação de 0 a 5 à senha analisada, de acordo com a diversidade dos caracteres utilizados.
RF04 A aplicação deve exibir recomendações de segurança ao usuário sempre que a senha for considerada fraca, com o objetivo de orientar sobre boas práticas na criação de senhas.
RF05 Opcionalmente, o sistema pode registrar os resultados da análise de senha para fins de teste, validação ou auditoria, respeitando os princípios de segurança e privacidade.

Requisitos Não-Funcionais

Os requisitos não funcionais referem-se às características técnicas e restrições que não estão diretamente relacionadas à funcionalidade, mas que impactam o desempenho, a usabilidade e a portabilidade do sistema:
RNF01 O tempo de resposta do sistema para a análise da senha deve ser inferior a dois segundos.
RNF02 O sistema deve ser desenvolvido utilizando a linguagem de programação Python.
RNF03 O código-fonte deve seguir boas práticas de desenvolvimento, incluindo indentação adequada, comentários explicativos e modularização.

RNF04 A aplicação deve ser compatível com os principais sistemas operacionais (Windows, Linux e MacOS), garantindo portabilidade e acessibilidade para diferentes usuários.

Requisitos de Qualidade

Com base no modelo de qualidade da norma ISO/IEC 25010, os seguintes requisitos de qualidade foram identificados para o projeto:
- Funcionalidade - Adequação funcional: O sistema deve cumprir corretamente as funções para as quais foi projetado, como a análise e pontuação da força de senhas.
- Usabilidade - Operacionalidade: A interface da aplicação deve ser intuitiva, clara e de fácil utilização, mesmo para usuários com pouca experiência técnica.
- Desempenho - Eficiência de tempo: O sistema deve processar a senha e fornecer os resultados em tempo quase instantâneo, com tempo de resposta inferior a dois segundos.
- Segurança - Confidencialidade: Nenhuma senha digitada deve ser armazenada, transmitida ou registrada de forma permanente, garantindo a privacidade do usuário.
- Manutenibilidade - Modularidade: O código deve ser escrito de forma modular, possibilitando atualizações e melhorias futuras de maneira simples e organizada.
- Portabilidade - Adaptabilidade: A aplicação deve ser capaz de ser executada em diferentes ambientes operacionais sem necessidade de grandes adaptações.

2.3 Planejamento de Testes

2.3.1 Plano de Testes

2.3.1.1 Introdução

Este plano de testes tem como objetivo definir as diretrizes para a verificação do correto funcionamento do sistema de verificação de força de senhas desenvolvido em Python. O planejamento abrange os critérios necessários para assegurar que o software atenda aos requisitos funcionais definidos no documento de requisitos, bem como as limitações previamente estabelecidas para este projeto..).

2.3.1.2 Escopo

O escopo dos testes contempla a validação do cumprimento dos requisitos mínimos de complexidade e comprimento das senhas inseridas, conforme o contexto de uso selecionado pelo usuário. A lógica de pontuação da força da senha, que atribui uma escala de 1 a 5 com base na diversidade de caracteres utilizados (letras minúsculas, letras maiúsculas, números, espaços e caracteres especiais), será rigorosamente verificada.

Adicionalmente, será avaliada a adequação da senha ao comprimento mínimo exigido para cada categoria de uso definida no sistema. A interface de entrada será testada quanto ao correto uso da função getpass para ocultação da senha durante a digitação, bem como ao tratamento apropriado de entradas inválidas, tais como seleção incorreta do contexto ou fornecimento de senhas muito curtas. A clareza e a completude das mensagens exibidas ao usuário ao final do processo também serão analisadas, visando garantir uma saída intuitiva e informativa.

Fora do escopo deste plano de testes estão o armazenamento seguro das senhas digitadas, uma vez que o sistema não realiza qualquer tipo de persistência de dados, bem como a integração com sistemas de autenticação externos e os testes de performance, escalabilidade ou carga.

2.3.1.3 Objetivos

Os principais objetivos deste plano de testes são garantir que:

- A lógica de verificação da senha esteja funcionando de acordo com os critérios definidos nos requisitos;

- Os diferentes contextos de uso da senha estejam corretamente associados aos respectivos comprimentos mínimos exigidos;

- A experiência do usuário seja adequada e protegida, com tratamento eficaz de erros e mensagens claras.

2.3.1.4 Requisitos a serem testados

1. Teste de Caixa Preta
Avaliação da senha *(RF02)*
Feedback baseado na avaliação *(RF08)*

2. Teste de Caixa Branca
Lógica de pontuação, regras de avaliação, e cálculos internos (RF06)

3. Testes Funcionais
Inserção de senha *(RF01)*
Exibição de pontuação *(RF04)*
Recomendações *(RF03)*

4. Testes de Interface
Campos de input e botões *(RF01, RF04)*

5. Testes de Integração
Exportação de relatórios *(RF07)*
Acesso ao gerenciamento por diferentes perfis *(RF05, RF10)*

2.3.1.5 Estratégias, tipos de testes e ferramentas a serem utilizadas

Para garantir que o sistema Strong Password funcione corretamente e ofereça avaliações e feedbacks claros e confiáveis, e cumpra os requisitos de segurança pré-estabelecidos e tenha a melhor usabilidade, serão aplicadas diferentes estratégias e tipos de testes.

Estratégias de Teste

- **Testes unitário:** Cada função do sistema será testada separadamente para garantir que todas as partes estão funcionando de maneira correta.
- **Testes incrementais:** À medida que novas funcionalidades são desenvolvidas em cada sprint, serão adicionados testes para validar e avaliar o funcionamento das mudanças implementadas.
- **Testes com variados tipos de senhas:** Senhas com diferentes níveis de complexidade, caracteres e tamanho serão testadas para garantir que o sistema avalie corretamente a força das senhas com base nos padrões de segurança pré-estabelecidos.

Requisito (ID)	Descrição do Requisito	Tipo de Teste Recomendado
RF01	O sistema deve permitir ao usuário inserir uma senha.	Teste de Interface / Unidade
RF02	O sistema deve avaliar a força da senha inserida.	Teste de Caixa Preta / Unidade
RF03	O sistema deve recomendar melhorias de segurança.	Teste Funcional / Teste de Sistema
RF04	O sistema deve exibir a pontuação da senha.	Teste de Interface / Funcional
RF05	O administrador deve poder gerenciar as recomendações.	Teste de Acesso / Teste de Sistema
RF06	O sistema deve aplicar regras para avaliar senhas.	Teste de Regras de Negócio / Unidade
RF07	O sistema deve permitir exportar a avaliação da senha.	Teste Funcional / Teste de Integração
RF08	O sistema deve fornecer feedback com base na avaliação.	Teste Funcional / Caixa Preta
RF09	O desenvolvedor deve testar o sistema.	(Organizacional - não testado pelo sistema)
RF10	O desenvolvedor deve poder gerenciar a base de regras.	Teste de Acesso / Integração

- **Testes de entrada inválida:** O sistema será testado com entradas inválidas, como senhas vazias ou contendo apenas espaços, para garantir que se comporte corretamente e não cause erros inesperados.

Tipos de Testes

- **Teste de Unidade (Caixa Branca):** Testa internamente cada função do código, analisando a construção do código e verificando todos os caminhos e condições lógicas.

- **Teste Funcional (Caixa Preta):** Testa e avalia o sistema como um todo no ponto de vista do usuário, verificando se ao digitar uma senha o sistema avalia e retorna um feedback, independentemente de como o código está implementado.

- **Teste de Integração:** Testa a interação entre os módulos do sistema, garantindo que a entrada, análise e o retorno do resultado funcionem de maneira integrada.

- **Teste de Validação de Entrada:** Testa e avalia como o sistema lida com entradas inválidas, como senhas vazias ou espaços em branco, garantindo que o sistema responda corretamente e de forma estável.

Ferramentas Utilizadas

- **Python:** Linguagem utilizada para o desenvolvimento do sistema.

- **Pytest / unittest:** Frameworks para criação e execução de testes, como os testes unitários e de integração, garantindo que o código funcione corretamente conforme esperado.

- **flake8:** Ferramenta utilizada para análise estática do código-fonte. Verificando se existe erros de sintaxe, problemas de estilo e padrões de codificação. Será usado para manter o código limpo e padronizado.

- **Git/GitHub:** Sistema para realizar o controle de versão e colaboração entre os desenvolvedores. Usado para gerenciar alterações no código e facilitar o compartilhamento do código.

2.3.1.6 Recursos a serem empregados

Recursos Humanos

Analistas de Teste: Responsáveis por planejar, projetar e executar os testes.

Desenvolvedores: Para corrigir os defeitos encontrados e às vezes apoiar na automação.

Engenheiros de Teste Automatizado: Criam e mantêm scripts de testes automatizados.

Gerente de Testes: Planeja os esforços de teste e gerencia a equipe.

Gerente de Projetos: Garante a integração dos testes no processo de desenvolvimento.

Recursos Tecnológicos

- Linguagem Python: Linguagem de programação escolhida para o desenvolvimento do sistema, devido à sua simplicidade, legibilidade e vasta biblioteca de suporte.

- Visual Studio Code (VSCode): Ambiente de desenvolvimento integrado utilizado para codificação, depuração e execução do projeto.

- GitHub: Plataforma de hospedagem de código utilizada para controle de versão, colaboração entre membros da equipe e integração contínua.

Recursos Organizacionais

Processos e Metodologias:

Metodologias Ágeis
Modelos de qualidade

Documentação

Plano de Testes
Casos de Testes
Relatórios de Defeitos e Métricas
Políticas de Qualidade e Segurança
Tempo e Orçamento Adequados

2.3.1.7 Cronograma das atividades

Sprint	Período	Atividades Principais	Entregas Esperadas
Sprint 1	28/04/2025 a 05/05/2025	• Planejamento da fase de testes• Definição dos critérios de aceitação e qualidade	Documento de planejamento e critérios de teste
Sprint 2	06/05/2025 a 13/05/2025	• Desenvolvimento dos primeiros casos de teste• Execução dos testes funcionais iniciais	Casos de teste desenvolvidos e executados
Sprint 3	14/05/2025 a 21/05/2025	• Execução de testes de integração• Validação dos	Relatório de testes de integração e validação

		requisitos implementados	
Sprint 4	22/05/2025 a 29/05/2025	• Análise de falhas encontradas• Ajustes no sistema e reexecução de testes	Versão ajustada e reavaliada do sistema
Sprint 5	30/05/2025 a 06/06/2025	• Testes finais de usabilidade e desempenho	Relatório de usabilidade e desempenho
Sprint 6	07/06/2025 a 11/06/2025	• Consolidação dos resultados de teste• Elaboração da documentação final	Documentação de testes e aprovação para entrega

2.3.1.8 Definição dos marcos do projeto

1	28/04/2025	Início da fase de testes (início da Sprint 1)
2	05/05/2025	Finalização do planejamento de testes e critérios definidos
3	13/05/2025	Conclusão da implementação dos testes funcionais
4	21/05/2025	Finalização da execução dos testes de integração e validação
5	29/05/2025	Encerramento dos ajustes e reexecução de testes
6	06/06/2025	Conclusão dos testes de usabilidade e desempenho
7	11/06/2025	Consolidação final e aprovação para entrega do sistema

2.3.2 Casos de Teste e Roteiro de Testes

Casos de Teste Caixa Branca (baseados no código interno)

Caixa Branca 1 — Senha menor que o mínimo recomendado

Entrada: senha = "abc", contexto = 1 (mínimo 8)

Esperado: Mensagem indicando que a senha tem menos caracteres que o mínimo.

Roteiro:

- Escolher opção 1 no menu (mínimo 8)
- Digitar senha "abc"
- Verificar mensagem de erro exibida sobre comprimento insuficiente

Caixa Branca 2 — Senha com todos os tipos de caracteres (mínimo 8)

Entrada: senha = "Abc123! ", contexto = 1 (mínimo 8)

Esperado: Contagem correta de: minúsculas, maiúsculas, números, espaço e caracteres especiais; força 5/5.

Roteiro:

- Escolher opção 1 (mínimo 8)
- Digitar senha "Abc123! "
- Verificar contagem correta: 2 minúsculas (b,c), 1 maiúscula (A), 3 números (1,2,3), 1 espaço, 1 especial (!).
- Força da senha exibida como 5/5 com observação de senha forte.

Caixa Branca 3 — Senha com somente números e minúsculas (mínimo 8)

Entrada: senha = "abc12345", contexto = 1 (mínimo 8)

Esperado: Contagem correta (3 minúsculas, 5 números), força 2/5 (minúsculas + números).

Roteiro:

- Escolher opção 1
- Digitar senha "abc12345"
- Confirmar contagem correta e força 2, com observação adequada.

Casos de Teste Caixa Preta (baseados em requisitos e funcionalidades sem olhar o código)

Caixa Preta 1 — Testar opção de contexto inválida

Entrada: Escolher opção 9 no menu

Esperado: Mensagem de opção inválida e novo pedido de escolha.

Roteiro:

- Iniciar programa
- Digitar 9 como escolha de contexto
- Verificar que mensagem “Opção inválida. Tente novamente.” aparece e que o menu é mostrado de novo.

Caixa Preta 2 — Testar contexto 5 (outro) com entrada inválida para mínimo

Entrada: Escolher opção 5 no menu e digitar "abc" para mínimo

Esperado: Mensagem de valor inválido e uso do padrão mínimo 8.

Roteiro:

- Iniciar programa

- Escolher opção 5 no menu
- Quando solicitar o mínimo, digitar "abc"
- Confirmar mensagem “Valor inválido, usando o padrão de 8 caracteres.”
- Continuar com verificação da senha com mínimo 8.

Caixa Preta 3 — Testar saída do programa

Entrada: Escolher opção 0 no menu

Esperado: Programa encerra com mensagem “Saindo...”

Roteiro:

- Iniciar programa
- Digitar 0 para sair
- Confirmar saída do programa e mensagem exibida.

Escolhendo o fluxo para o grafo:

Vou usar o fluxo da função verificar_forca_senha() que é o núcleo do programa e onde ocorrem as decisões importantes.

Principais decisões no código:

- Verifica se len(senha) < minimo_recomendado → desvia para mensagem de erro e termina (return)
- Para cada caractere da senha: classifica em um dos 5 tipos (mínusculas, maiusculas, numeros, espacos, especiais) — aqui temos 5 decisões em sequência dentro do loop.
- Depois, conta a força acumulando pontos se cada tipo tem pelo menos 1 caractere → 5 decisões (um if para cada tipo).
- Finalmente, um if-elif para atribuir a observação de acordo com a força (5 condições).

Cálculo da Complexidade Ciclomática (V(G))

Para calcular a complexidade ciclomática, usamos:

V(G) = E - N + 2P

- E = número de arestas
- N = número de nós
- P = número de componentes conexos (normalmente 1 para funções simples)

Mas podemos simplificar contando os pontos de decisão:

V(G) = número de decisões + 1

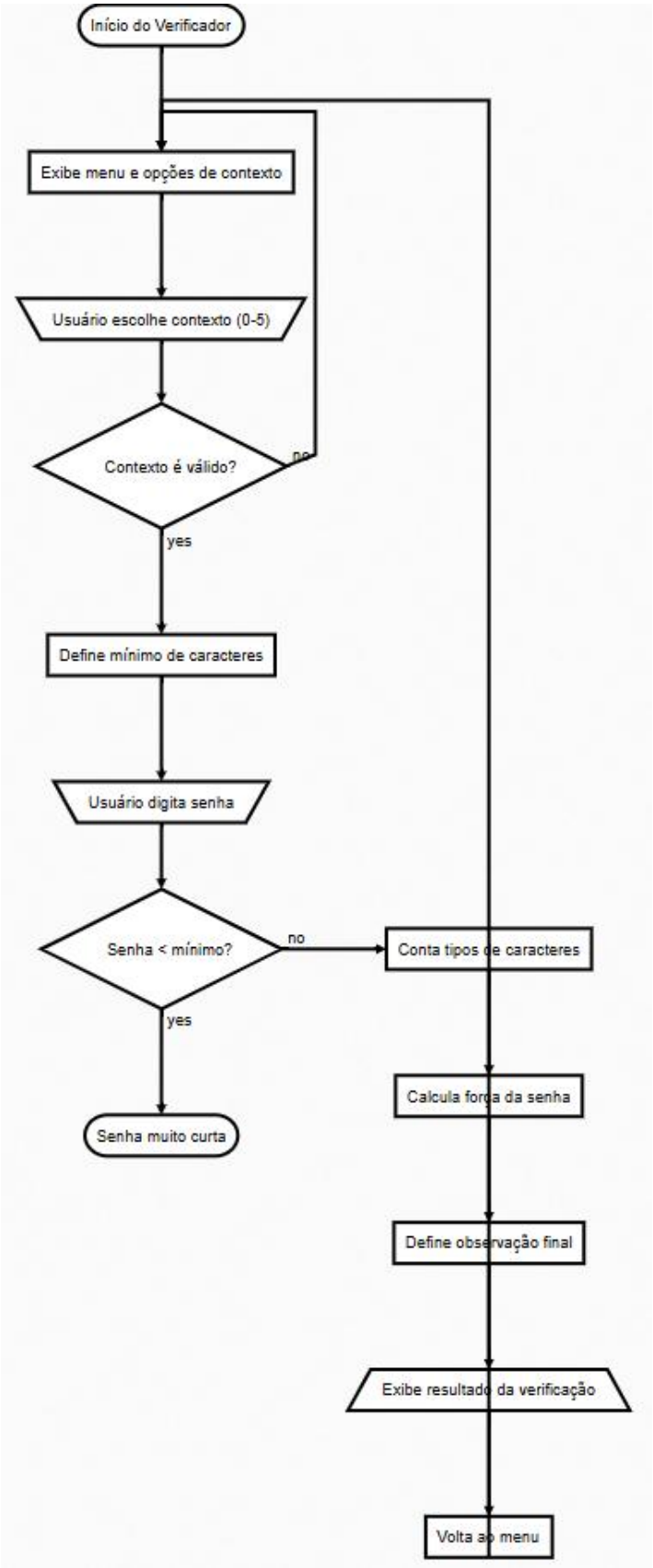
Vamos contar as decisões na função verificar_forca_senha():

- 1 decisão: if len(senha) < minimo_recomendado
- Dentro do loop: para cada caractere, temos 5 testes if-elif (mínuscula, maiúscula, número, espaço, especial). Importante: eles são sequenciais, não aninhados. Em termos de caminho, isso pode contar como 5 decisões (pois cada if decide se incrementa uma variável).
- Depois, temos 5 decisões para somar os pontos de força (if minusculas >=1, etc.)
- Por fim, temos 5 decisões para as observações (if-elif para forca de 1 a 5) — só uma decisão real (pois é if-elif), mas vamos considerar 5 para análise conservadora.

Total de decisões = 1 + 5 + 5 + 5 = 16

Assim, complexidade ciclomática = 16 + 1 = 17

Grafo de Fluxo:


Início do Verificador
Exibe menu e opções de contexto
Usuário escolhe contexto (0-5)
Contexto é válido?
no
yes
Define mínimo de caracteres
Usuário digita senha
Senha < mínimo?
no
yes
Conta tipos de caracteres
Calcula força da senha
Senha muito curta
Define observação final
Exibe resultado da verificação
Volta ao menu

3. GESTÃO DE CONFIGURAÇÃO DE SOFTWARE

A Gestão de Configuração de Software (GCS) neste projeto tem como objetivo garantir o controle e a rastreabilidade de todos os artefatos produzidos durante o desenvolvimento do verificador de força de senhas. Essa gestão foi essencial para manter a consistência do código, dos testes e da documentação ao longo das diferentes fases do projeto.

Itens de Configuração

Os principais itens de configuração definidos para este projeto foram:

Código-fonte do verificador de senha (verificador.py);

Casos de teste e roteiros de validação (caixa branca e caixa preta);

Documentação técnica (como o diagrama de fluxo, cálculo da complexidade ciclomática, etc.);

Cronograma de Sprints e marcos do projeto;

Scripts auxiliares ou bibliotecas utilizadas (por exemplo, string, getpass).

Controle de Versão

Foi utilizado o Git como sistema de controle de versão, permitindo:

Rastrear todas as alterações no código-fonte;

Criar branches separados para desenvolvimento, testes e ajustes;

Registrar o histórico de mudanças com mensagens de commit descritivas;

Restaurar versões anteriores sempre que necessário.

As versões principais foram marcadas com tags que indicam entregas parciais ou marcos relevantes (por exemplo: v1.0-teste-funcional, v2.0-usabilidade, vFinal).

Controle de Mudanças

Modificações relevantes no código ou na estrutura dos testes foram analisadas, documentadas e aplicadas de forma controlada. Cada mudança foi registrada em um arquivo de changelog com:

Motivo da mudança;

Descrição da alteração;

Responsável pela modificação;

Data de aplicação.

Rastreabilidade

A rastreabilidade foi mantida entre:

Requisitos de teste (ex: senha muito curta, senha completa, contexto inválido);

Casos de teste específicos (caixa branca e caixa preta);

Saídas esperadas do sistema;

Resultados reais de execução dos testes.

Esse controle garantiu que todas as funcionalidades fossem devidamente cobertas e validadas.

Ferramentas Utilizadas

Git para versionamento;

GitHub para hospedagem remota do repositório e controle colaborativo;

Markdown/Google Docs para documentação e roteiros;

Python 3.x como linguagem principal de desenvolvimento.

Benefícios observados no projeto

Facilidade na identificação de regressões durante alterações no código;

Melhor organização do ciclo de vida do software;

Colaboração fluida entre os envolvidos no desenvolvimento e testes;

Segurança ao aplicar mudanças em fases avançadas do projeto;

Clareza na apresentação dos marcos e das entregas parciais.

4. REPOSITÓRIO DE GESTÃO DE CONFIGURAÇÃO DE SOFTWARE

https://github.com/magictog/Strong-Password-A3

5. CONCLUSÃO

O projeto Strong Password representou uma oportunidade concreta de aplicar os conhecimentos adquiridos na UC Gestão e Qualidade de Software, reunindo práticas de engenharia de software, critérios de qualidade e técnicas de validação aplicadas a um sistema real. Por meio da definição estruturada de requisitos, do planejamento de testes e da aplicação de modelos como ISO/IEC 25010, foi possível desenvolver uma solução funcional e segura, voltada à análise da força de senhas, promovendo tanto a segurança da informação quanto a conscientização dos usuários sobre boas práticas digitais.

Durante o desenvolvimento, foram aplicadas ferramentas e estratégias de controle de versão, gestão de configuração, testes de caixa preta e caixa branca, além de uma abordagem incremental com sprints, alinhada a metodologias ágeis. O uso do GitHub como repositório e das ferramentas Python, Pytest e flake8 contribuiu diretamente para a organização, rastreabilidade e qualidade do código. Com isso, o projeto não apenas atingiu seus objetivos funcionais, mas também consolidou conceitos essenciais de qualidade de software, como confiabilidade, usabilidade, segurança e manutenibilidade — evidenciando a importância de uma gestão eficaz em todas as etapas do ciclo de vida do software.

6. Bibliografia

Biblioteca string: PYTHON SOFTWARE FOUNDATION. string — Common string operations. 2025. Disponível em: https://github.com/itsallaboutpython/Top-10-Easy-Python-Project-Ideas-For-Beginners/blob/main/password_strength_checker.py. Acesso em: 24 abr. 2025.

Biblioteca string: PYTHON SOFTWARE FOUNDATION. string — Common string operations. 2025. Disponível em: https://docs.python.org/3/library/string.html. Acesso em: 24 abr. 2025.

Biblioteca getpass: PYTHON SOFTWARE FOUNDATION. getpass — Utility to securely handle password prompts. 2025. Disponível em: https://docs.python.org/3/library/getpass.html. Acesso em: 24 abr. 2025.