Especificação de Requisitos de Software (SRS) - Sistema Bancário Simples

1. Introdução

1.1. Propósito do Documento

Este documento define, de forma clara, completa e detalhada, os requisitos funcionais e não funcionais para o desenvolvimento do Sistema Bancário Simples. Ele serve como base para o desenvolvimento, testes, validação e futuras iterações do projeto, garantindo que todas as partes interessadas — desenvolvedores, engenheiros e stakeholders — tenham uma compreensão comum das funcionalidades, restrições e expectativas do sistema. Além disso, atua como um guia para a validação do sistema após a implementação, assegurando que o produto final atenda aos objetivos especificados.

1.2. Escopo do Sistema

O Sistema Bancário Simples é uma aplicação local com interface gráfica simples, destinada à criação e gerenciamento de contas bancárias. Desenvolvido em Python com persistência em SQLite, o sistema suporta as seguintes operações principais:

- **Criação de contas**: Contas titular (administradoras) e contas usuário (vinculadas às titulares).
- **Autenticação**: Acesso seguro com encaminhamento a menus específicos conforme o tipo de conta.
- **Operações bancárias**: Depósito, saque e consulta de extrato, com controle de limites para contas usuário.
- **Gerenciamento de usuários**: Listagem, edição, definição de limites e exclusão de contas usuário.
- **Registro e feedback**: Todas as operações são logadas, e o sistema fornece feedback claro ao usuário via interface gráfica.

O sistema é projetado para uso local, sem integração com serviços externos na versão inicial, mas com modularidade para permitir expansões futuras.

1.3. Definições, Acrônimos e Abreviações

- **Conta Titular**: Conta administrativa com permissões para gerenciar contas usuário.
- **Conta Usuário**: Conta vinculada a uma conta titular, com limites de transações definidos.
- **Transação**: Qualquer operação que altere o saldo da conta, como depósito ou saque.
- **Limite de Operação**: Restrições aplicadas a contas usuário, como número máximo de saques diários ou valor total máximo de saque por dia.
- **SRS**: Software Requirements Specification (Especificação de Requisitos de Software).
- **RBAC**: Role-Based Access Control (Controle de Acesso Baseado em Papéis).

1.4. Referências

- **IEEE Std 830-1998**: IEEE Recommended Practice for Software Requirements Specifications.
- **PEP 8**: Style Guide for Python Code. Disponível em: https://www.python.org/dev/peps/pep-0008/.
- **C4 Model**: Modelo para Arquitetura de Software. Disponível em: https://c4model.com/.

1.5. Visão Geral do Documento

Este SRS é estruturado para fornecer uma visão abrangente do Sistema Bancário Simples. Ele descreve as funcionalidades do sistema, os requisitos funcionais e não funcionais, a matriz de rastreabilidade dos requisitos, diretrizes para implementação, testes e documentação, além de sugestões de melhoria. O documento visa ser um recurso completo e didático para o desenvolvimento, testes e evolução do projeto.

2. Descrição Geral

2.1. Perspectiva do Produto

O Sistema Bancário Simples é uma aplicação standalone com interface gráfica, desenvolvida em Python 3.11 e utilizando SQLite para persistência de dados. Ele adota uma arquitetura orientada a objetos, integrando padrões de design como **Decorator** (para registro de logs) e **Iterator** (para navegação em listas de contas). O design é

modular, permitindo extensibilidade para futuras integrações, como APIs externas ou suporte a múltiplos usuários em rede.

2.2. Funções do Produto

O sistema oferece as seguintes funcionalidades principais:

- **Criação de Contas**: Geração automática de IDs únicos para contas titular (formato: xxxxxx$t$) e usuário (formato: xxxxxxu), com confirmação exibida na interface após sucesso.
- **Autenticação**: Login seguro com encaminhamento a menus distintos para contas titular e usuário.
- **Operações Bancárias**: Depósito, saque e consulta de extrato, com validação de limites e mensagens de confirmação ou erro (ex.: "Saque excedeu o limite diário").
- **Gerenciamento de Usuários**: Listagem, edição, definição de limites (ex.: número de saques diários) e exclusão de contas usuário, com feedback após cada ação.
- **Feedback e Navegação**: Validação imediata de dados na interface, mensagens de erro específicas (ex.: "Valor inválido") e botões habilitados apenas com entradas válidas.

2.3. Restrições Gerais

- O sistema opera localmente, com dados armazenados em SQLite.
- Não há integração com serviços externos ou APIs na versão inicial.
- O design deve ser modular, evitando dependências desnecessárias que dificultem expansões futuras.
- A interface gráfica deve ser simples e intuitiva, priorizando usabilidade.

3. Requisitos Específicos

3.1. Requisitos Funcionais (RF)

RF1 - Criação e Identificação de Contas

- **RF1.1 - Criação de Conta Titular**: O sistema deve gerar um ID único no formato xxxxxx$t$ (6 dígitos seguidos de t). Antes de confirmar a criação, deve verificar a unicidade do ID no banco de dados.

- **RF1.2 - Criação de Conta Usuário**: Similar ao RF1.1, mas com o formato xxxxxxu. A conta usuário deve ser vinculada a uma conta titular existente.
- **RF1.3**: Após a criação, o sistema exibe uma mensagem de sucesso, como "Conta criada com ID: xxxxxx$t$".

RF2 - Autenticação e Encaminhamento

- **RF2.1**: O sistema deve autenticar usuários com base em ID e senha.
- **RF2.2**: Após login, encaminha o usuário ao menu correspondente (titular ou usuário).
- **RF2.3**: Limita tentativas de login a 3 por conta; após 3 falhas, bloqueia a conta por 5 minutos, exibindo "Conta temporariamente bloqueada".

RF3 - Operações Bancárias

- **RF3.1**: Permite depósitos com valores positivos, atualizando o saldo e registrando a transação em log.
- **RF3.2**: Permite saques, respeitando o saldo disponível e limites diários das contas usuário.
- **RF3.3**: Fornece extrato com histórico de transações (data, tipo, valor).
- **RF3.4**: Exibe mensagens de confirmação (ex.: "Depósito de R$100 realizado") ou erro (ex.: "Saldo insuficiente").
- **RF3.5**: Garante operações atômicas com transações SQLite para evitar inconsistências em acessos simultâneos.

RF4 - Gerenciamento de Usuários

- **RF4.1**: Contas titular podem listar todas as contas usuário vinculadas.
- **RF4.2**: Permite editar dados (ex.: senha) ou excluir contas usuário.
- **RF4.3**: Define limites para contas usuário (ex.: 3 saques/dia, R$500/dia), monitorados em tempo real após cada transação.

RF5 - Validação e Feedback de Dados

- **RF5.1**: Valida campos obrigatórios (nome, senha, valores) em formulários, exibindo erros específicos, como "Nome não pode estar vazio" ou "Valor deve ser positivo".
- **RF5.2**: Botões de ação (ex.: "Confirmar") só são habilitados com dados válidos.

3.2. Requisitos Não Funcionais (RNF)

RNF1 - Plataforma e Linguagem

- Desenvolvido em Python 3.11, seguindo as diretrizes da PEP 8 para legibilidade e manutenção.

RNF2 - Desempenho

- O sistema deve processar operações (ex.: saque, depósito) em menos de 1 segundo em condições normais.

RNF3 - Persistência

- Dados são armazenados em SQLite, com esquema definido incluindo tabelas para contas, transações e logs.

RNF4 - Segurança e Privacidade

- Senhas são criptografadas com o algoritmo **bcrypt** antes do armazenamento.
- Usa **RBAC** para controle de acesso, limitando operações por tipo de conta.
- Logs registram todas as ações sensíveis (ex.: login, exclusão de conta) com data e usuário.

RNF5 - Portabilidade

- Funciona em sistemas Windows, Linux e macOS com Python 3.11 instalado.

RNF6 - Usabilidade e Interface

- Feedback é exibido via mensagens na interface (pop-ups ou textos em destaque), informando o status de cada operação.

4. Matriz de Rastreabilidade de Requisitos

ID Requisito	Descrição	Caso de Teste

RF1.1	Criação de Conta Titular	CT-Cadastro-Titular-01: Verificar ID único e mensagem de sucesso.
RF2.3	Limite de Tentativas de Login	CT-Login-01: Testar 3 falhas seguidas e bloqueio temporário.
RF3.2	Saque com Limites	CT-Saque-01: Testar saque acima do limite diário e verificar negação.
RF4.3	Monitoramento de Limites	CT-Limite-01: Verificar atualização em tempo real após transação.
RNF4	Criptografia de Senhas	CT-Segurança-01: Confirmar que senhas são armazenadas criptografadas com bcrypt.

5. Análise e Sugestões de Melhoria

- **Geração de IDs**: Use um contador incremental no SQLite ou **UUIDs** para garantir unicidade e escalabilidade.
- **Validação de Dados**: Implemente validações no cliente (interface) e servidor (backend), com testes automatizados via **unittest** para entradas inválidas.
- **Persistência**: Crie um esquema SQLite com tabelas relacionadas (ex.: `contas`, `transacoes`, `logs`), usando chaves estrangeiras para integridade.
- **Segurança**: Adote **bcrypt** para senhas e logs detalhados para auditoria.
- **Testes**: Inclua testes unitários, de integração e de aceitação para cobrir todos os requisitos.

6. Diretrizes de Implementação

- **Modularidade**: Use classes separadas para `Conta`, `Usuario`, `Transacao`, seguindo o princípio da responsabilidade única.
- **Padrões de Design**: Aplique **Singleton** para gerenciar a conexão com o SQLite e **Observer** para notificar mudanças (ex.: saldo atualizado).
- **Interface**: Utilize bibliotecas como **Tkinter** ou **PyQt** para uma GUI simples e responsiva.

7. Testes

- **Unitários**: Teste cada funcionalidade (ex.: criação de conta) com **unittest**.
- **Integração**: Verifique a interação entre módulos (ex.: banco de dados e interface).
- **Aceitação**: Valide os requisitos funcionais com cenários reais (ex.: saque com limite excedido).

8. Documentação do Código

- Inclua **docstrings** em classes e métodos (ex.: `"""Cria uma nova conta com ID único."""`).
- Comente trechos complexos para facilitar manutenção.

9. Versionamento

- Use **Git** com convenção de commits (ex.: `feat: adicionar login`, `fix: corrigir saque`).
- Mantenha um **README** atualizado no repositório com instruções de instalação e uso.

10. Considerações Finais

Este SRS deve ser revisado após cada iteração do projeto (ex.: novas funcionalidades, correções) para manter os requisitos alinhados ao sistema implementado. Ele foi projetado para ser um recurso prático e educativo, ajudando você a aprofundar seus conhecimentos em engenharia de software.

Como Aprender com Este Projeto

1. **Práticas de Engenharia**: Estude o IEEE 830-1998 para entender como estruturar SRSs robustos.
2. **Padrões de Design**: Implemente **Decorator**, **Singleton** e **Observer** para dominar arquitetura de software.
3. **Testes**: Explore **unittest** e crie cenários complexos para garantir qualidade.

4. **Segurança**: Pesquise **bcrypt** e RBAC para proteger dados sensíveis.
5. **Versionamento**: Aprofunde-se em Git e boas práticas de colaboração.