UNIVERSIDADE FEDERAL DO OESTE DO PARÁ

EVERTON HIAN DOS SANTOS PINHEIRO

DOCUMENTO DE VISÃO DO PROJETO SNAPGRAM

Santarém/Pa

2025

1. Introdução

1.1 Objetivo

O SnapGram é uma plataforma de rede social moderna e intuitiva, projetada para conectar pessoas e permitir o compartilhamento de momentos, fotos e interações por meio de postagens, curtidas e comentários. Nosso objetivo principal é oferecer uma experiência de usuário leve, segura e escalável, com foco em desempenho, usabilidade e transparência.

1.2 Escopo

1.2.1 Frontend: Uma interface gráfica web responsiva e interativa, desenvolvida com tecnologias padrão da web, como HTML, CSS e JavaScript. O frontend é projetado para ser leve e acessível, garantindo uma experiência de usuário fluida em dispositivos desktops.

1.2.2 Backend (API RESTful): Um conjunto de endpoints que respondem às requisições do frontend e possibilitam a integração com outras aplicações externas. O backend é construído com Flask e utiliza SQLAlchemy para gerenciar o banco de dados, garantindo escalabilidade e segurança.

1.2.2.1 Além disso, o backend inclui:

- Um sistema de autenticação seguro baseado em JWT (JSON Web Tokens)
- Funcionalidades pra receber os upload do usuário e gerenciar os arquivos (fotos de perfil, postagens, etc...).
- Armazena curtidas e comentários, para promover engajamento entre os usuários.

2. Descrição Geral

2.1 Problema

As redes sociais existentes no mercado frequentemente apresentam desafios significativos, como:

Problema	Descrição
Interfaces pesadas e pouco responsivas	Muitas plataformas possuem interfaces complexas e carregadas, que consomem muitos recursos e dificultam a experiência em dispositivos móveis ou com hardware limitado.
Falta de transparência no uso de dados	Usuários estão cada vez mais preocupados com a privacidade e o uso indevido de seus dados pessoais, algo comum em redes sociais tradicionais.
Dificuldade de uso em dispositivos com recursos limitados	Plataformas existentes não são otimizadas para dispositivos de baixo custo ou com conexões de internet lentas, excluindo uma parcela significativa de usuários.

2.2 Solução

O SnapGram foi desenvolvido para resolver esses problemas, oferecendo uma plataforma que combina simplicidade, segurança e desempenho. Nossa solução inclui:

Solução	Descrição
Interface leve e acessível	Um frontend otimizado para desktops, com carregamento rápido e navegação intuitiva, garantindo uma experiência agradável mesmo em conexões lentas.
Tecnologias modernas e escaláveis	Utilizamos Flask para o backend, garantindo flexibilidade e desempenho, e SQLAlchemy para gerenciamento eficiente do banco de dados. Além disso, a API RESTful permite integração com outras aplicações e futuras expansões.
Segurança e privacidade	Implementamos um sistema de autenticação seguro baseado em JWT, com criptografia de dados sensíveis e proteção contra ataques comuns. O SnapGram prioriza a transparência no uso de dados, garantindo que os usuários tenham controle sobre suas informações.
Engajamento social	Funcionalidades como curtidas, comentários e upload de fotos promovem a interação entre os usuários, criando uma comunidade ativa e engajada.

3. Atores

Ator	Interação
Usuários Finais	Pessoas que utilizam o frontend pra se comunicar com a api, com o objetivo de criar contas, fazer postagens e interagir.
Administrador	Responsável por gerenciar usuários, monitorar atividades e moderar conteúdos.
API Consumers	Aplicações externas de terceiros que também queiram consumir a API para aproveitar funcionalidades

4. Requisitos

4.1 Requisitos Funcionais

Id	Requisito Funcional
RF01	O usuário deve poder criar uma conta com e-mail e senha.
RF02	O sistema deve autenticar os usuários por meio do login.
RF03	O usuário deve poder criar postagens.
RF04	O usuário deve poder excluir postagens.
RF05	O usuário deve poder enviar imagens junto às postagens.
RF06	Os usuários devem poder curtir postagens.
RF07	Os usuários devem poder comentar nas postagens.
RF08	O usuário deve poder editar sua foto de perfil

RF09	O usuário deve poder sair de sua conta

4.2 Requisitos Não Funcionais

Id	Requisito Não Funcional
RNF01	O tempo de resposta da API deve ser inferior a 300ms para solicitações comuns (ex.: listagem de posts, obtenção de perfil de usuário).
RNF02	O sistema deve suportar até 1.000 usuários simultâneos sem degradação significativa no desempenho.
RNF03	Todas as comunicações entre o frontend e o backend devem ser realizadas através de JSON.
RNF04	A API deve seguir o padrão RESTful para garantir consistência e facilidade de integração.
RNF05	Senhas devem ser armazenadas de forma segura, utilizando hashing com bcrypt.
RNF06	A autenticação deve ser feita através de JWT (JSON Web Tokens) para garantir segurança e escalabilidade.
RNF07	O sistema deve ser protegido contra SQL Injection, XSS (Cross-Site Scripting) e CSRF (Cross-Site Request Forgery).
RNF08	Tokens JWT inválidos ou expirados devem ser armazenados em uma blacklist para evitar reutilização.
RNF09	O banco de dados deve ser projetado para suportar até 100.000 registros de usuários, posts, comentários e curtidas.
RNF10	A arquitetura do sistema deve permitir a escalabilidade horizontal (ex.: adição de mais servidores para distribuir a carga).
RNF11	O sistema deve ter uma disponibilidade de 99,9% (uptime).
RNF12	Deve ser implementado um sistema de backup automático do banco de dados para garantir a recuperação de dados em caso de falha.
RNF13	A interface do usuário deve ser responsiva e funcionar corretamente em dispositivos móveis e desktops.
RNF14	O tempo de carregamento da página inicial deve ser inferior a 2 segundos.
RNF15	O código deve seguir boas práticas de clean code e ser documentado para facilitar a manutenção.
RNF16	Deve ser utilizado um sistema de controle de versão (Git) para gerenciar alterações no código.
RNF17	O frontend deve ser compatível com os principais navegadores (Chrome, Firefox, Safari, Edge).
RNF18	A API deve suportar requisições de diferentes clientes (web, mobile, etc.).
RNF19	O sistema deve incluir testes automatizados para garantir a qualidade do código.
RNF20	Testes de unidade e integração devem cobrir pelo menos 80% do código.
RNF21	O sistema deve permitir o upload de fotos de perfil e posts, com tamanho máximo de 5MB por arquivo.

RNF22	As imagens enviadas devem ser armazenadas em um diretório seguro e servidas através de rotas protegidas.
RNF23	O sistema deve gerar logs de todas as operações críticas (ex.: autenticação, upload de arquivos, exclusão de posts).
RNF24	Deve ser implementado um sistema de monitoramento para acompanhar o desempenho e a saúde da aplicação.

5. Tecnologias Envolvidas

5.1 Backend

- **Linguagem de Programação:** O backend foi desenvolvido em **Python**, uma linguagem simples.
- **Framework:** Utilizamos o **Flask**, um microframework leve e flexível para Python, ele é bom para construir APIs RESTful e aplicações web escaláveis
- **Banco de Dados: SQLite** foi escolhido para o desenvolvimento inicial devido à sua simplicidade e facilidade de configuração, mas planejamos migrar para o PostgreSQL em produção, devido à sua robustez, suporte a grandes volumes de dados e recursos avançados.
- **Autenticação:** A autenticação é gerenciada pelo **Flask-JWT-Extended**, que permite a implementação segura de tokens JWT (JSON Web Tokens) para autenticação e autorização.
- **API:** A API foi construída com **Flask-RESTful**, uma extensão do Flask que facilita a criação de APIs RESTful de forma organizada e eficiente.

Outras Bibliotecas:

- **Flask-SQLAlchemy:** Utilizado como ORM (Object-Relational Mapping) para simplificar a interação com o banco de dados.
- **Flask-Bcrypt:** Responsável pela criptografia segura de senhas, garantindo a proteção dos dados dos usuários.
- **Flask-Migrate:** Facilita o gerenciamento de migrações do banco de dados, permitindo alterações no schema de forma controlada.
- **Flask-WTF:** Utilizado para validação de formulários e proteção contra ataques CSRF (Cross-Site Request Forgery).

5.2 Frontend

- **HTML/CSS:** A estrutura e a estilização básica da interface foram desenvolvidas com HTML e CSS, garantindo uma base sólida para a apresentação do conteúdo.
- **JavaScript:** Utilizamos JavaScript para adicionar funcionalidades dinâmicas à interface, como interações com a API, validações em tempo real e atualizações de conteúdo sem recarregar a página.

6. Modelos de Casos de Uso

6.1 Caso de Uso: Criar Postagem

Usuário:

Fluxo principal do usuário:

1. O usuário acessa sua conta.
2. Clica em "Nova Postagem".
3. Escreve o conteúdo e faz upload de uma imagem.
4. Salva a postagem.
5. A postagem é exibida no feed.

6.2 Caso de Uso: Curtir Postagem

Fluxo principal do usuário:

1. O usuário acessa sua conta e navega até o feed.
2. Clica no botão de "curtir" em uma postagem.
3. O número de curtidas é incrementado.

6.3 Caso de Uso: Comentar Postagem

Fluxo principal do usuário:

4. O usuário acessa sua conta e navega até o feed.
5. Clica no botão de "comentar" em uma postagem.
6. Escreve o conteúdo.
7. Clica no botão enviar.

7. Estrutura do Banco de Dados