Aplicação Web: Sistema de Compartilhamento de Arquivos

1. **Visão Geral**

O Sistema de Compartilhamento de Arquivos é uma aplicação web que tem como objetivo permitir aos usuários a realização de operações relacionadas ao gerenciamento de arquivos, como cadastramento de usuários, upload, download e compartilhamento de arquivos por meio de links gerados pelo sistema. A aplicação é desenvolvida com uma arquitetura frontend e backend integrada, visando oferecer uma experiência otimizada e segura para os usuários.

1.1. Funcionalidades Principais

a. Cadastro de novos usuários.
b. Login e autenticação de usuários.
c. Upload de arquivos.
d. Compartilhamento de arquivos por links únicos.
e. Download e gerenciamento de arquivos enviados.

Requisitos Funcionais

1.2.1 Cadastro de Usuários

a. O sistema deve permitir o cadastro de novos usuários com e-mail, nome de usuário e senha.
b. O sistema deve validar o e-mail para garantir que seja único no banco de dados.
c. A senha do usuário deve ser criptografada antes de ser armazenada no banco de dados.

1.2.2 Login e Autenticação de Usuários

a. O sistema deve permitir que usuários façam login com e-mail e senha.
b. A autenticação deve ser feita utilizando JSON Web Tokens (JWT).
c. O sistema deve retornar mensagens de erro adequadas caso o login falhe (e.g., "E-mail ou senha incorretos").

1.2.3 Upload de Arquivos

a. O usuário deve poder fazer upload de arquivos.
b. O sistema deve suportar múltiplos tipos de arquivos (com extensões e tamanhos permitidos previamente definidos).
c. O sistema deve renomear arquivos duplicados de forma automática, adicionando um número incremental.

d. O arquivo deve ser armazenado na pasta específica do usuário.

1.2.4 Download de Arquivos

a. O sistema deve permitir que o usuário baixe seus próprios arquivos.

b. Links únicos devem ser gerados para que arquivos possam ser baixados por terceiros (somente para arquivos compartilhados).

1.2.5 Compartilhamento de Arquivos

a. O sistema deve permitir que o usuário compartilhe arquivos através de um link único gerado pelo sistema.

b. Esses links devem ser únicos e armazenados no banco de dados junto com a data de compartilhamento.

1.2.6 Gerenciamento de Conta

a. O sistema deve permitir que o usuário atualize suas informações pessoais (nome, e-mail, senha).

b. O sistema deve solicitar a senha atual antes de permitir alterações de senha.

1.2.7 Gerenciamento de Arquivos

a. O usuário deve poder visualizar uma lista de seus arquivos enviados, incluindo nome, tamanho, data de envio e links de compartilhamento.

b. O usuário deve poder excluir arquivos.

1.2.8 Logout

a. O sistema deve permitir que o usuário faça logout, invalidando o token JWT.

Requisitos Não Funcionais

1.2.9 Segurança

a. As senhas dos usuários devem ser armazenadas de forma segura utilizando hashing (e.g., bcrypt ou hashlib).

b. Todas as rotas que lidam com dados sensíveis (cadastro, login, upload, download) devem usar HTTPS para garantir a criptografia dos dados.

c. Tokens JWT devem ser utilizados para autenticar usuários, e esses tokens devem expirar após um período de inatividade.

d. O sistema deve evitar vulnerabilidades comuns como SQL Injection, XSS e CSRF.

1.2.10 Desempenho

a. O sistema deve ser capaz de processar uploads e downloads de arquivos de até 100 MB sem degradação significativa de performance.

b. As respostas às requisições de API devem ser retornadas em menos de 1 segundo para operações simples (como login ou consulta de arquivos).

c. O sistema deve ser escalável, permitindo adição de servidores ou armazenamento conforme o número de usuários crescer.

1.2.11 Usabilidade

a. A interface deve ser amigável e fácil de usar, com layouts responsivos que funcionem em dispositivos móveis e desktops.

b. O sistema deve oferecer feedback imediato ao usuário em ações como upload de arquivos, compartilhamento e login (e.g., usando pop-ups de sucesso ou erro).

c. O processo de navegação entre telas e funcionalidades deve ser fluido, sem carregamentos excessivos.

1.2.12 Manutenibilidade

a. O código do sistema deve ser modular, permitindo fácil atualização e correção de bugs.

b. O sistema deve utilizar boas práticas de desenvolvimento (e.g., separação clara entre frontend e backend, uso de ORM como SQLAlchemy).

c. Devem existir logs para auditoria e solução de problemas, registrando operações como login, upload e download de arquivos.

1.2.13 Escalabilidade

a. O sistema deve ser projetado para suportar centenas de usuários simultâneos, com a capacidade de expansão para milhares de usuários.

b. O armazenamento de arquivos deve ser flexível e escalável, usando estrutura de diretórios organizados por usuário.

1.2.14 Disponibilidade

a. O sistema deve ser altamente disponível, com um tempo de inatividade (downtime) mínimo durante manutenções ou atualizações.

b. Devem ser realizados backups automáticos periódicos do banco de dados e arquivos.

1.2.15 Compatibilidade

a. O sistema deve funcionar corretamente nos principais navegadores modernos (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).

b. O layout e a responsividade devem ser compatíveis com telas de diferentes tamanhos, incluindo dispositivos móveis e tablets.

1.2.16 Experiência do Usuário (UX)

a. A aplicação deve fornecer uma experiência de usuário rápida e sem complicações, com ações que requerem poucos cliques.

b. Animações leves e transições suaves devem ser usadas para melhorar a interação com o sistema, sem comprometer o desempenho.

2. Tecnologias Utilizadas

2.1. Frontend

HTML: Estrutura da página, com cabeçalhos, conteúdos principais e rodapés.

CSS: Estilização e layout responsivo, utilizando Flexbox para organização.

Javascript: Funcionalidades dinâmicas, como manipulação de DOM e carrossel de seções.

2.2. Backend

Flask: Framework para o desenvolvimento de aplicações web utilizando a linguagem Python.

MySQL: Banco de dados relacional para armazenamento de informações dos usuários e arquivos.

Hashlib: Biblioteca utilizada para hashing de senhas, garantindo segurança dos dados dos usuários.

PyJWT: Sistema de autenticação baseado em JSON Web Tokens (JWT) para gerenciar as sessões dos usuários.

2.3. Ferramentas e Bibliotecas Auxiliares

Google Fonts: Para estilização tipográfica.

Flask-SQLAlchemy: Mapeamento objeto-relacional (ORM) para o MySQL.

Flask-Uploads: Gerenciamento e manipulação de arquivos enviados pelos usuários.

Flexbox: Utilizado para o desenvolvimento de layouts flexíveis e responsivos.

Keyframes: Animações aplicadas a botões para melhorar a experiência do usuário.

3. Funcionalidades de Backend

3.1. Banco de Dados MySQL

O sistema utiliza um banco de dados MySQL para armazenar informações relacionadas aos usuários e arquivos. Abaixo estão as tabelas principais e seus respectivos campos:

3.1.1. Tabela de Usuários (Users)

- **id**: Identificador único do usuário (INT, AUTO_INCREMENT, PRIMARY KEY).
- **username**: Nome de usuário único (VARCHAR(150), UNIQUE, NOT NULL).
- **email**: Endereço de e-mail único do usuário (VARCHAR(150), UNIQUE, NOT NULL).
- **password**: Senha criptografada do usuário (VARCHAR(150), NOT NULL).

3.1.2. Tabela de Arquivos (Files)

- **id**: Identificador único do arquivo (INT, AUTO_INCREMENT, PRIMARY KEY).
- **filename**: Nome do arquivo (VARCHAR(255), NOT NULL).
- **extension**: Extensão do arquivo (VARCHAR(10), NOT NULL).
- **size**: Tamanho do arquivo em bytes (INT, NOT NULL).
- **upload_date**: Data e hora do envio do arquivo (DATETIME, DEFAULT CURRENT_TIMESTAMP).
- **user_id**: Referência ao id do usuário que fez o upload (INT, FOREIGN KEY).
- **link_token**: Token único gerado para compartilhamento do arquivo (VARCHAR(255), UNIQUE, NULL).

3.2.1. Cadastro de Usuários

A rota **/cadastro** permite que novos usuários se registrem no sistema. A senha é criptografada usando hashlib e armazenada no banco de dados. Caso o e-mail informado já esteja cadastrado, uma mensagem de erro é retornada ao usuário.

3.2.2. Login de Usuários

A rota **/login** permite a autenticação de usuários existentes. Após a validação, um JWT é gerado e enviado ao cliente, que o utilizará para acessar as funcionalidades protegidas do sistema.

3.3. Gerenciamento de Arquivos

3.3.1. Upload de Arquivos

Os usuários podem fazer upload de arquivos por meio da rota **/upload**. Utiliza-se a biblioteca Flask-Uploads para o gerenciamento e armazenamento dos arquivos no

servidor. O nome do arquivo é validado e, caso exista um arquivo com o mesmo nome, o sistema adiciona um número incremental ao nome.

3.3.2. Download de Arquivos

A rota **/download/<int:arquivo_id>** permite aos usuários baixarem arquivos específicos que foram previamente enviados e armazenados no servidor.

3.3.3. Compartilhamento de Arquivos

A rota **/api/gerar-link/<int:arquivo_id>** gera um link único para que o usuário possa compartilhar um arquivo com terceiros. Esse link é armazenado no banco de dados, juntamente com a data de compartilhamento.

3.4. Outras Funcionalidades

3.4.1. Atualização de Dados do Usuário

Atualização de Nome: A rota **/atualizar-nome** permite que o usuário atualize seu nome de perfil.

Atualização de E-mail: A rota **/atualizar-email** permite a alteração do e-mail.

Atualização de Senha: A rota **/atualizar-senha** possibilita a alteração da senha, desde que o usuário forneça a senha atual corretamente.

3.4.2. Logout

A rota **/logout** finaliza a sessão do usuário, removendo o JWT associado.

4. Considerações Finais

O sistema de compartilhamento de arquivos foi projetado com foco em segurança e usabilidade. O backend é responsável por garantir que as operações de upload, download e compartilhamento sejam realizadas de forma segura, enquanto o frontend proporciona uma experiência agradável e responsiva aos usuários. O sistema segue as melhores práticas de desenvolvimento web, utilizando hashing de senhas, autenticação JWT e rotas seguras para manipulação de dados.

Alunos

Cristiano Lopes

Elvis Gabriel

Eduardo Henrique Tresmann

Eduardo Henrique Marques

José Victor

Kauan Roger