Documentação da API de Gerenciamento de Blog

Ronald Santos

7 de Outubro de 2025

Contents

1 Introdução

Este documento fornece uma especificação abrangente para a API de Gerenciamento de Blog, um serviço RESTful construído com Flask. A API suporta operações CRUD para postagens de blog, autenticação de usuários e verificação de saúde do sistema, além de servir uma aplicação de página única (SPA) no frontend. Ela é projetada para ser robusta, com registro de logs, tratamento de erros e suporte a CORS para requisições entre origens.

2 Visão Geral

2.1 Objetivo

A API de Gerenciamento de Blog permite a criação, recuperação, atualização e exclusão de postagens de blog, além de autenticação de usuários e verificações de saúde do sistema. Ela também serve ativos estáticos do frontend e suporta roteamento do lado do cliente.

2.2 Principais Funcionalidades

- **CORS**: Habilitado para todas as origens (*), suportando métodos `GET`, `POST`, `PUT`, `DELETE` e `OPTIONS`.
- **Registro de Logs**: Nível de depuração para rastreamento de requisições e diagnóstico de erros.
- **Tratamento de Erros**: Respostas JSON padronizadas com campos `success`, `message` e `detail`.
- **URL Base**: `http://localhost:3000` (configurável).
- **Serviço de Ativos Estáticos**: Ativos do frontend servidos a partir de `../frontend/dist`, com fallback para `index.html` para roteamento SPA.

2.3 Formato de Resposta

Todas as respostas são em formato JSON:

- **Sucesso**: `{"success": true, "message": "...", ...}`
- **Erro**: `{"success": false, "detail": "..."}`

Códigos de status HTTP:

- `200`: OK
- `201`: Criado
- `400`: Requisição Inválida
- `401`: Não Autorizado
- `500`: Erro Interno do Servidor
- `503`: Serviço Indisponível

3 Endpoints da API

A tabela a seguir resume os endpoints disponíveis:

Endpoint	Método	Descrição	Corpo da Requisição	Exemplo de Resposta
/health	GET	Verifica o status da API.	Nenhum	Sucesso: {"status": "healthy", "message": "API está funcionando", "success": true}
/api/login	POST	Autentica um usuário.	{"username": "string", "password": "string"}	Sucesso: {"message": "Login bem-sucedido", "username": "user", "success": true}
/api/posts	GET	Recupera todas as postagens.	Nenhum	Sucesso: {"message": "Postagens recuperadas com sucesso", "posts": [...], "count": 1, "success": true}
/api/create_blog	POST	Cria uma nova postagem.	{"name": "string", "title": "string", "description": "string"}	Sucesso: {"message": "Postagem criada com sucesso", "post": "title", "success": true}
/api/update_post	PUT	Atualiza uma postagem pelo título antigo.	{"old_title": "string", "name": "string", "title": "string", "description": "string"}	Sucesso: {"message": "Postagem atualizada com sucesso", "post": "title", "description": "...", "success": true}
/api/delete_post	DELETE	Exclui uma postagem pelo título.	{"title": "string"}	Sucesso: {"message": "Postagem excluída com sucesso", "post": "title", "success": true}
/<path>	GET	Serve ativos do frontend ou index.html.	Nenhum	HTML/JS/CSS ou index.html

Table 1: Resumo dos Endpoints da API

3.1 Detalhes dos Endpoints

3.1.1 Verificação de Saúde (`GET /health`)

- **Objetivo**: Verifica o status da API.
- **Parâmetros**: Nenhum.
- **Códigos de Resposta**: `200` (saudável), `503` (não saudável).
- **Exemplo de Requisição**:

```
curl -X GET http://localhost:3000/health
```

3.1.2 Login (`POST /api/login`)

- **Objetivo**: Valida credenciais de usuário.
- **Corpo da Requisição**: JSON com `username` e `password`.
- **Códigos de Resposta**: `200` (sucesso), `400` (entrada inválida), `401` (não autorizado), `500` (erro do servidor).
- **Exemplo de Requisição**:

```
curl -X POST http://localhost:3000/api/login \
     -H "Content-Type: application/json" \
     -d '{"username": "admin", "password": "pass123"}'
```

3.1.3 Obter Postagens (`GET /api/posts`)

- **Objetivo**: Recupera todas as postagens de blog.
- **Parâmetros**: Nenhum.
- **Códigos de Resposta**: `200` (sucesso), `500` (erro do servidor).
- **Exemplo de Requisição**:

```
curl -X GET http://localhost:3000/api/posts
```

- **Exemplo de Resposta**:

```
{
    "message": "Postagens recuperadas com sucesso",
    "posts": [
        {
            "id": 1,
            "name": "Nome do Autor",
            "title": "Minha Primeira Postagem",
            "description": "Esta é uma descrição de exemplo."
        }
    ],
    "count": 1,
    "success": true
}
```

3.1.4 Criar Postagem (`POST /api/create_blog`)

- **Objetivo**: Adiciona uma nova postagem de blog.
- **Corpo da Requisição**: JSON com `name`, `title`, `description`.
- **Códigos de Resposta**: `201` (criado), `400` (entrada inválida), `500` (erro do servidor).
- **Exemplo de Requisição**:

```
curl -X POST http://localhost:3000/api/create_blog \
     -H "Content-Type: application/json" \
     -d '{"name": "João Silva", "title": "Novo Blog", "description": "Co
```

3.1.5 Atualizar Postagem (`PUT /api/update_post`)

- **Objetivo**: Atualiza uma postagem existente identificada por `old_title`.
- **Corpo da Requisição**: JSON com `old_title`, `name`, `title`, `description`.
- **Códigos de Resposta**: `200` (sucesso), `400` (entrada inválida), `500` (erro do servidor).
- **Exemplo de Requisição**:

```
curl -X PUT http://localhost:3000/api/update_post \
     -H "Content-Type: application/json" \
     -d '{"old_title": "Título Antigo", "name": "Maria Silva", "title":
          "description": "Novo conteúdo"}'
```

3.1.6 Excluir Postagem (`DELETE /api/delete_post`)

- **Objetivo**: Exclui uma postagem pelo `title`.
- **Corpo da Requisição**: JSON com `title`.
- **Códigos de Resposta**: `200` (sucesso), `400` (entrada inválida), `500` (erro do servidor).
- **Exemplo de Requisição**:

```
curl -X DELETE http://localhost:3000/api/delete_post \
     -H "Content-Type: application/json" \
     -d '{"title": "Título a Excluir"}'
```

3.1.7 Serviço de Arquivos Estáticos (`GET /<path>`)

- **Objetivo**: Serve ativos do frontend a partir de `../frontend/dist` ou retorna `index.html` para roteamento SPA.
- **Parâmetros**: Caminho (e.g., `/static/js/main.js` ou rotas não correspondidas).
- **Códigos de Resposta**: `200` (arquivo servido).

4 Configuração e Implantação

4.1 Pré-requisitos

- **Dependências Python**: Instale via `pip install flask flask-cors pymysql python-dotenv`.
- **Build do Frontend**: Execute `npm run build` em `../frontend` para gerar a pasta `dist/`.

4.2 Executando a Aplicação

1. Carregue as variáveis de ambiente do arquivo `.env`.
2. Construa o frontend: `cd ../frontend  npm run build`.
3. Inicie o servidor: `python app.py`.
4. Acesse em `http://0.0.0.0:3000` (modo de depuração).

Se o build do frontend estiver ausente, a API opera no modo ”somente API” com um aviso no console.

5 Considerações de Segurança

- **Autenticação**: Sem JWT ou gerenciamento de sessão; considere adicionar para controle de acesso seguro.
- **Validação de Entrada**: Verificações básicas implementadas; melhore com bibliotecas como Marshmallow.
- **CORS**: Aberto para todas as origens (`*`); restrinja em produção para maior segurança.

6 Tratamento de Erros

Erros são registrados no nível DEBUG e retornados como JSON com um campo `detail`. Dados de requisição ausentes ou inválidos retornam `400`.

7 Melhorias Futuras

- Implementar autenticação baseada em JWT.
- Adicionar sanitização e validação de entrada (e.g., Marshmallow).
- Suportar paginação para `/api/posts`.
- Gerar documentação OpenAPI/Swagger.
- Adicionar testes unitários e de integração.

8 Suporte

Para problemas, revise os logs do servidor (nível DEBUG). Contate a equipe de desenvolvimento para assistência adicional.