INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO GRANDE DO NORTE

INFORMÁTICA PARA INTERNET

EMILY MEDEIROS DOS SANTOS, MARIA RITA LUCENA SANTOS, MICKAELLE KARINE SOUZA SILVA

DOCUMENTO DE REQUISITOS
Projeto de programação orientada a serviços

Caicó/RN
2025

1. Introdução

Este documento detalha os requisitos funcionais e não funcionais para a **API de Resumo de Artigos**. O objetivo principal da API é fornecer um serviço que receba um texto em formato **Markdown** e retorne um resumo conciso, acompanhado de **palavras-chave** relevantes, de forma eficiente e confiável.

2. Visão Geral do Sistema

A **API de Resumo de Artigos** será uma ferramenta essencial para usuários que precisam de resumos rápidos e precisos de documentos extensos. Ela utilizará modelos de **Inteligência Artificial** para processar o texto e extrair as informações mais importantes. A comunicação com a API será feita via **HTTP**, utilizando requisições **POST** e respostas no formato **XML**.

3. Requisitos Funcionais

3.1. Endpoint de Resumo

- **Identificador**: FR-001
- **Descrição**: A API deve expor um endpoint para o envio de textos a serem resumidos.
- **Método HTTP**: `POST`
- **Caminho**: `/resumir`
- **Parâmetros de Requisição**:
    - `texto`: String (obrigatório). Conteúdo completo do artigo em formato **Markdown**.
- **Exemplo de Requisição**:

  HTTP
  ```
  POST /resumir HTTP/1.1
  Host: api.exemplo.com
  Content-Type: application/json
  ```

```
{
  "texto": "# Título do Artigo\nEste artigo discute..."
}
```

3.2. Formato de Resposta

- **Identificador**: FR-002
- **Descrição**: A API deve retornar o resumo e as palavras-chave no formato **XML**.
- **Estrutura da Resposta**:

  XML

```
<?xml version="1.0" encoding="UTF-8"?>
<resumo>
  <texto_resumido>Este artigo discute...</texto_resumido>
  <palavras_chave>
    <palavra>artigo</palavra>
    <palavra>discussão</palavra>
  </palavras_chave>
</resumo>
```

3.3. Geração de Resumo

- **Identificador**: FR-003
- **Descrição**: A API deve utilizar um **modelo de IA pré-treinado** (por exemplo, **BERT** ou **GPT**) para gerar um resumo conciso do texto fornecido.
- **Critério de Aceitação**: O resumo gerado deve ser **coerente**, **gramaticalmente correto** e **capturar os pontos principais** do texto original.

3.4. Extração de Palavras-Chave

- **Identificador**: FR-004
- **Descrição**: A API deve aplicar técnicas de **extração de palavras-chave** (por exemplo, **TF-IDF** ou algoritmos de aprendizado de máquina) para identificar as palavras mais relevantes do texto.
- **Critério de Aceitação**: As palavras-chave extraídas devem ser **relevantes e representativas** do conteúdo do texto, fornecendo uma visão rápida dos temas abordados.

3.5. Persistência em Banco de Dados

- **ID:** FR-005
- **Descrição:** Todos os resumos devem ser salvos no banco de dados SQLite.
- **Tecnologia:** SQLModel + SQLite (sumarize.db)
- **Critérios de Aceitação:**
    - Dados persistidos corretamente.
    - Operações CRUD completas.

3.6. Atualização de Resumos

- **ID:** FR-006
- **Descrição:** Atualizar um resumo existente por ID.
- **Método:** PUT
- **Rota:** /resumos/{id}
- **Corpo:**

```
{
  "texto": "Novo texto para atualizar..."
}
```

3.7. Remoção de Resumos

- **ID:** FR-007
- **Descrição:** Excluir resumo do banco por ID.
- **Método:** DELETE
- **Rota:** /resumos/{id}

3.8. Listar Todos os Resumos

- **ID:** FR-008
- **Descrição:** Retornar lista de todos os resumos cadastrados.
- **Método:** GET
- **Rota:** /resumos/

3.9. Mini Aplicação Cliente (CLI)

- **ID:** FR-009
- **Descrição:** Criar script interativo cliente.py para acessar a API via terminal.
- **Funcionalidades:**
    - Criar resumo (POST /resumir)
    - Listar todos (GET /resumos)
    - Atualizar resumo (PUT /resumos/{id})

    - Deletar resumo (DELETE /resumos/{id})
    - Extrair palavras-chave (POST /palavras-chaves/)
    - Obter resposta completa XML (POST /processar/)
- **Critérios de Aceitação:**
    - Interface clara via print() e input()
    - Mensagens de erro e sucesso bem definidas

4. Código Requisitos Não Funcionais

4.1. Performance

- **Identificador**: NFR-001
- **Descrição**: O tempo de resposta da API para o processamento de textos deve ser otimizado para garantir uma experiência de usuário eficiente.
- **Métricas**: A API deve responder em no máximo **3 segundos** para textos de até **5.000 palavras**. Para textos de até **20.000 palavras**, o tempo de resposta não deve exceder **10 segundos**.
- **Considerações**: A utilização de **GPUs** em servidores de produção é mandatória para otimizar o desempenho do modelo de **IA** e garantir os tempos de resposta especificados.

4.2. Confiabilidade

- **Identificador**: NFR-002
- **Descrição**: A API deve estar disponível e funcional para os usuários na maior parte do tempo.
- **Métricas**: A disponibilidade do serviço deve ser de, no mínimo, **99,5%** ao longo de um mês.
- **Considerações**: Implementação de mecanismos de monitoramento, logs de erro detalhados e planos de recuperação de desastres para minimizar o tempo de inatividade.

4.3. Escalabilidade

- **Identificador**: NFR-003
- **Descrição**: A arquitetura da API deve permitir a escalabilidade horizontal e vertical para lidar com um volume crescente de requisições e cargas de trabalho variadas.

- **Considerações**: A escolha de frameworks como **FastAPI** e servidores **ASGI** como recursos de autoescalonamento (ex: **AWS Auto Scaling**, **Kubernetes**) será fundamental.

4.4. Segurança

- **Identificador**: NFR-004
- **Descrição**: A API deve proteger os dados do usuário e o próprio serviço contra acessos não autorizados e ataques.
- **Métricas**: Implementação de autenticação via **Chave de API (API Key)** ou **OAuth 2.0** para controle de acesso.
- **Considerações**: Limitação de taxa de requisições (rate limiting) para prevenir ataques de negação de serviço (DoS). Validação de entrada robusta (já coberta pelo NFR-005) para mitigar vulnerabilidades de injeção de código.

4.5. Manutenibilidade

- **Identificador**: NFR-005
- **Descrição**: O código da API deve ser fácil de entender, modificar e estender.
- **Considerações**: Adoção de boas práticas de codificação (Clean Code), documentação interna do código, testes automatizados (unitários e de integração) e uso de sistemas de controle de versão (Git).

4.6. Validação de Entrada

- **Identificador**: NFR-006
- **Descrição**: A API deve validar o texto recebido para garantir que esteja no formato **Markdown** e não esteja vazio, além de impor limites de tamanho.
- **Métricas**:
    - **Texto vazio**: Retornar **código 400 (Bad Request)**.
    - **Formato inválido**: Retornar **código 400 (Bad Request)** com uma mensagem específica.
    - **Tamanho máximo do texto**: Limitar o texto de entrada a **20.000 palavras**. Textos excedentes devem resultar em **código 413.**
- **Tratamento de Erros**:

- **Requisições Inválidas**: Retornar **código 400 (Bad Request)** com uma mensagem de erro clara e específica (ex: "O campo 'texto' é obrigatório", "Formato Markdown inválido").

4.7. Tratamento de Erros

- **Identificador**: NFR-007
- **Descrição**: A API deve fornecer respostas de erro claras, informativas e consistentes para diferentes cenários.
- **Tipos de Erro**:
  - **Erros Internos do Servidor**: Retornar **código 500 (Internal Server Error)** com uma mensagem genérica de erro interno e um ID de transação para rastreamento em logs.
  - **Erros de Processamento da IA**: Retornar **código 500 (Internal Server Error)** com uma mensagem indicando falha no processamento do resumo, se aplicável, sem expor detalhes internos do modelo.

5. Tecnologias Utilizadas

- **Linguagem de Programação**: **Python**
- **Framework Web**: **FastAPI**
- **Servidor ASGI**: **Uvicorn**
- **Validação de Dados**: **Pydantic**
- **Banco de Dados: SQLite com SQLModel**
- **Modelos de IA**: Modelos pré-treinados para **processamento de linguagem natural** (ex: **BERT**, **GPT**, ou alternativas otimizadas para inferência).
- **Extração de Palavras-Chave**: Algoritmos como **TF-IDF**, **RAKE** (Rapid Automatic Keyword Extraction) ou modelos de aprendizado de máquina dedicados.
  - comentado e testado