instituto Federal do Norte de Minas Gerais Campus - Januária
***Curso Bacharelado em Sistemas de Informação***
**Disciplina:** Sistemas Distribuídos
Prof. Adriano Antunes Prates

RAFAELA RAMOS FARIAS E GEOVANA DE SOUSA RODRIGUES

# NEWSFLOW – SISTEMA DE GERENCIAMENTO DE CONTEÚDO (CMS) DISTRIBUÍDO

Januária - MG
2025

RAFAELA RAMOS FARIAS

GEOVANA DE SOUSA RODRIGUES

# NEWSFLOW – SISTEMA DE GERENCIAMENTO DE CONTEÚDO (CMS) DISTRIBUÍDO

Trabalho apresentado à disciplina de Sistemas Distribuídos, do curso de Sistemas de Informação do Instituto Federal do Norte de Minas Gerais, como requisito parcial para obtenção de nota.

Prof. Adriano Antunes Prates

**SUMÁRIO**

# 1. Proposta Geral da Aplicação (Cenário de Uso)

O projeto consiste no desenvolvimento do *backend* de um **Portal de Notícias (CMS) Distribuído**, projetado para suportar alta concorrência de acessos e grande volume de dados.

No cenário atual da web, portais de conteúdo frequentemente enfrentam picos repentinos de tráfego (efeito "viral"), em que milhares de usuários tentam acessar uma mesma notícia simultaneamente. Arquiteturas monolíticas tradicionais tendem a falhar ou degradar a performance nessas situações devido a gargalos no banco de dados centralizado ou ao bloqueio de requisições.

O **NewsFlow** resolve este problema simulando um ambiente de produção real onde a leitura de dados é priorizada e distribuída. O sistema permitirá:

1. A publicação de artigos (título, corpo, autor, categoria) através de uma API REST.
2. A consulta massiva desses artigos, garantindo baixa latência.
3. A fragmentação automática dos dados baseada em categorias (ex: Esportes, Política, Tecnologia), garantindo que nenhuma máquina (nó) do banco de dados seja sobrecarregada individualmente.

O foco não está na interface gráfica (frontend), mas na robustez da arquitetura distribuída, provando conceitos de disponibilidade e escalabilidade horizontal.

**2. Tecnologias Adotadas**

Para atender aos requisitos de desempenho e distribuição, a seguinte stack tecnológica foi selecionada:

- **Linguagem de Programação: Python 3.11+**. Escolhida pela sua vasta biblioteca de suporte a redes e facilidade de prototipagem rápida.
- **Framework Web: FastAPI**. Um framework moderno e de alta performance para construção de APIs.
    - *Diferencial:* Utiliza o padrão ASGI (Asynchronous Server Gateway Interface), permitindo processamento assíncrono real. Isso significa que o servidor pode lidar com múltiplas requisições de entrada sem bloquear a thread principal enquanto aguarda respostas do banco de dados.
- **Banco de Dados: MongoDB (NoSQL)**.

    - *Justificativa:* Como um portal de notícias lida com dados semi-estruturados (artigos variam em tamanho e metadados), um banco orientado a documentos é ideal.
- **Tecnologia de Distribuição: MongoDB Sharding**.
    - Utilizaremos um cluster nativo do MongoDB para fragmentar os dados horizontalmente. O cluster será composto por:
        - **Config Servers:** Armazenam metadados do cluster.
        - **Shard Servers:** Armazenam os subconjuntos de dados reais.
        - **Mongos (Query Router):** Roteia as consultas da aplicação para o shard correto.
- **Virtualização e Orquestração: Docker e Docker Compose**.
    - Essenciais para simular o sistema distribuído em um único ambiente de desenvolvimento. O Docker Compose irá subir múltiplos containers, simulando diferentes servidores físicos na rede.
- **Driver de Conexão: Motor (Async MongoDB driver)**.
    - Biblioteca Python que permite a comunicação assíncrona entre o FastAPI e o MongoDB, garantindo que o gargalo de I/O do banco não trave a API.

**3. Justificativa: Relação com Sistemas Distribuídos**

Este projeto aplica diretamente os conceitos fundamentais da disciplina de Sistemas Distribuídos, com ênfase em **Escalabilidade** e **Transparência de Acesso**.

**3.1. Escalabilidade Horizontal (Sharding)** Diferente da escalabilidade vertical (adicionar mais memória a um único servidor), este projeto implementa a escalabilidade horizontal. Através do *Sharding* do MongoDB, demonstraremos como o sistema pode crescer indefinidamente apenas adicionando novos nós ao cluster. Escolheremos uma *Shard Key* estratégica (ex: categoria do artigo) para provar como o sistema distribui a carga de escrita e leitura entre diferentes servidores automaticamente.

**3.2. Processamento Assíncrono e Concorrência** A utilização do FastAPI introduz o conceito de *Non-blocking I/O*. Em sistemas distribuídos, a latência de rede é inevitável. O uso de async/await demonstra como maximizar o uso de recursos computacionais (CPU)

enquanto se aguarda respostas de serviços remotos (o banco de dados distribuído), permitindo um *throughput* (vazão) muito superior a frameworks síncronos tradicionais.

**3.3. Transparência de Distribuição** A arquitetura proposta garante transparência ao cliente final. O usuário (ou o frontend) faz uma requisição à API sem saber se o dado está armazenado no Servidor A, B ou C. O componente mongos atua como um *middleware* transparente, abstraindo a complexidade da localização física dos dados.

## 4. Equipe e Responsabilidades Técnicas Individualizadas

A equipe dividirá as tarefas focando na especialização das tecnologias propostas (FastAPI e Sharding), convergindo para a integração final via Docker.

**Membro 1: Geovana de Sousa Rodrigues**

- **Responsabilidade Principal:** Engenharia de Backend e API (Application Layer).
- **Tarefas Detalhadas:**
    1. Desenvolvimento da estrutura do projeto em **FastAPI**.
    2. Implementação dos modelos de dados (Pydantic models) para validação de entrada e saída.
    3. Criação dos *endpoints* RESTful (GET, POST, PUT, DELETE) para Artigos.
    4. Implementação da lógica de conexão assíncrona com o banco de dados usando **Motor**.
    5. Documentação automática da API via Swagger UI.

**Membro 2: Rafaela Ramos Farias**

- **Responsabilidade Principal:** Engenharia de Dados e Infraestrutura Distribuída (Data & Network Layer).
- **Tarefas Detalhadas:**
    1. Projeto e configuração do ambiente **Docker Compose** para orquestrar os múltiplos containers.
    2. Configuração do Cluster **MongoDB**: setup dos Config Servers e Inicialização dos Shards.

3. Definição e implementação da estratégia de **Sharding** (escolha da *Shard Key* e criação dos índices).
4. Testes de carga e verificação da distribuição dos dados (comprovar que os dados estão sendo, de fato, divididos entre os servidores).
5. Scripts de *seed* (população) do banco de dados para testes.

**5. Link do Repositório**

Todo o código, arquivos de configuração Docker e documentação técnica serão versionados e disponibilizados publicamente no seguinte repositório:

- **URL:** *https://github.com/murphiie/Projeto_Sistemas_Distribuidos*
-
-