# Documentação Técnica do MVP: Agregador de Notícias Insight Finder AI

## 1. Objetivo do MVP

O objetivo principal deste MVP (Produto Mínimo Viável) é construir a base funcional e robusta da plataforma: um agregador de notícias simples, mas completo. Ele irá coletar dados de feeds RSS, armazená-los de forma eficiente em um banco de dados e exibi-los em uma interface web funcional.

O foco é na **simplicidade** e **funcionalidade**. A arquitetura de filas com Redis e os serviços de IA serão implementados em iterações futuras, após a validação desta primeira etapa.

## 2. Visão Geral da Arquitetura

O MVP será dividido em dois projetos principais, gerenciados por um monorepo com pnpm:

* **apps/backend**: Uma API construída com **NestJS** para gerenciar a coleta de dados, a persistência no banco de dados e a comunicação com o frontend.
* **apps/frontend**: Uma aplicação web construída com **Next.js** para consumir os dados do backend e exibi-los em uma interface de usuário responsiva.

### Tecnologias

* **Backend**: Node.js, TypeScript, NestJS, Prisma, PostgreSQL, Docker.
* **Frontend**: Next.js, React, TypeScript, Tailwind CSS.
* **Ferramentas de Desenvolvimento**: pnpm, Biome.js.

## 3. Estrutura e Passo a Passo do Desenvolvimento

### Backend (NestJS, Prisma, PostgreSQL)

1. **Configuração Inicial**:
   * Criar o projeto NestJS (pnpm create nest-app apps/backend).
   * Configurar o TypeScript e integrar o Biome.js para padronização de código.
2. **Banco de Dados (PostgreSQL e Prisma)**:
   * Modelagem de dados:
     + Tabela Source: id, name (nome da fonte), rss\_url (URL do feed RSS), logo\_url (URL do logo da fonte).
     + Tabela Article: id, source\_id (chave estrangeira para Source), title, link (URL do artigo, para ser a chave de identificação única), content, publication\_date.
   * Usar o **Prisma** para gerenciar o esquema do banco de dados e as migrações.
   * Criar o arquivo schema.prisma com as entidades Source e Article e suas relações.
3. **Dockerização**:
   * Criar o arquivo docker-compose.yml para orquestrar os contêineres do backend (Node.js) e do banco de dados (PostgreSQL).
   * A imagem do PostgreSQL deve ter volumes para persistir os dados.
   * O contêiner do backend deve ser configurado para o ambiente de desenvolvimento, incluindo *hot-reload*.
4. **Desenvolvimento da Lógica de Coleta**:
   * Criar um **módulo Collector** no NestJS.
   * Usar o módulo Schedule do NestJS para agendar um trabalho de coleta em intervalos regulares (e.g., a cada 10 minutos).
   * O serviço de coleta irá:
     + Buscar todas as URLs RSS da tabela Source.
     + Fazer uma requisição HTTP para cada URL.
     + Parsear o XML do feed RSS para um objeto JavaScript.
     + Iterar sobre os artigos do feed e, para cada um, verificar se o link já existe na tabela Article para evitar duplicatas.
     + Se o link for novo, inserir o artigo na tabela Article.
5. **Exposição da API**:
   * Criar um **módulo Article** com um Controller e um Service.
   * Expor um endpoint GET /articles que busca os artigos mais recentes no banco de dados e os retorna em formato JSON, ordenados pela publication\_date em ordem decrescente.

### Frontend (Next.js)

1. **Configuração Inicial**:
   * Criar o projeto Next.js (pnpm create next-app apps/frontend).
   * Configurar o Biome.js para o frontend.
   * Instalar e configurar o **Tailwind CSS** para estilização.
2. **Design e Interface**:
   * A página inicial será a index.tsx.
   * Ela fará uma requisição para o endpoint GET /articles do backend.
   * Criar um layout responsivo para exibir uma lista de cards. Cada card representará uma notícia.
   * O design de cada card deve incluir:
     + O logo da fonte de notícias (usando a logo\_url da tabela Source).
     + O título da notícia.
     + A data e hora de publicação.
     + Um link para o artigo original.
3. **Comunicação com o Backend**:
   * Usar fetch ou uma biblioteca como axios no Next.js para fazer a chamada HTTP para o backend.
   * Tratar o estado de carregamento e possíveis erros na requisição.

## 4. Próximas Etapas e Avaliação

Após a conclusão e validação do MVP, as seguintes melhorias podem ser consideradas para as próximas iterações:

* **Paginação**: Implementar paginação na API do backend para evitar sobrecarga de dados.
* **Busca**: Adicionar funcionalidade de busca de notícias por palavra-chave.
* **Melhoria na Sincronização**: Mudar o agendamento para um sistema de filas com Redis e BullMQ para maior robustez e escalabilidade.
* **Módulo de IA**: Adicionar a sumarização de notícias e outras funcionalidades baseadas em IA.