Atue como um Engenheiro de Software Sênior em Python e um expert em DevOps, focado em código limpo, modularidade, testes e, principalmente, em um histórico de Git impecável.

Vamos construir um projeto de avaliação técnica passo a passo, onde cada etapa representa uma funcionalidade essencial para o sistema final. O objetivo é criar um histórico de commits limpo e cronológico que demonstre a evolução do conhecimento e do desenvolvimento do sistema.

**Contexto e Escolhas Definidas:**

* **Linguagem Principal:** Python 3.10+
* **Framework API:** FastAPI (com Uvicorn)
* **Banco de Dados:** PostgreSQL (com SQLAlchemy ORM)
* **Cidade de Referência:** Florianópolis
* **Chave de API:** O nome da variável de ambiente é OPENWEATHER\_API\_KEY. O código **DEVE** ler esta chave do ambiente (os.environ ou dotenv).

**Nosso objetivo é gerar o conteúdo de cada arquivo, passo a passo.**

INSTRUÇÃO CRONOLÓGICA DE EXECUÇÃO (MANDATÓRIO):

A execução DEVE ser estritamente sequencial. A cada passo concluído, mostre o código gerado e AGUARDE A MINHA CONFIRMAÇÃO para que eu possa realizar o commit no GitHub e autorizar o prosseguimento para o próximo passo da sequência.

### FASE 1: Configuração e Modelagem de Dados

**PASSO 1: Estrutura e Dependências**

* **Commit Sugerido:** feat: Setup inicial e dependências
* **Ações:**
  1. Crie a estrutura de diretórios do projeto (ex: src/, db/, config/).
  2. Gere o arquivo requirements.txt com as dependências essenciais: fastapi, uvicorn, sqlalchemy, psycopg2-binary, requests, python-dotenv, e pydantic.
  3. Gere o arquivo .env.example com placeholders (valores vazios) para todas as variáveis de ambiente necessárias para a execução.

**-> AGUARDAR CONFIRMAÇÃO E COMMIT**

**PASSO 2: Módulo de Persistência (PostgreSQL)**

* **Commit Sugerido:** feat: Módulo de banco de dados e modelo de dados
* **Ações:**
  1. No diretório db/, crie o arquivo database.py para configurar a Engine e a Session Local do SQLAlchemy, garantindo que a URL de conexão seja lida das variáveis de ambiente.
  2. No diretório db/, crie o arquivo models.py com o Pydantic Model (WeatherRecord) e a classe de mapeamento ORM (WeatherData). A tabela deve ter campos para: id (PK, Sequence), city\_name (string), temperature (float), humidity (int), description (string), e timestamp (DateTime, com server\_default para func.now()).
  3. Crie o script initialize\_db.py (na raiz) que carregue as variáveis de ambiente e crie todas as tabelas definidas em models.py.

**-> AGUARDAR CONFIRMAÇÃO E COMMIT**

### FASE 2: Extração e Ingestão de Dados

**PASSO 3: Módulo de Extração de Dados (OpenWeather)**

* **Commit Sugerido:** feat: Cliente API OpenWeather com tratativa de dados
* **Ações:**
  1. No diretório src/, crie o arquivo weather\_client.py com uma classe OpenWeatherClient.
  2. Esta classe deve inicializar com a URL base e a chave de API (lida do ambiente).
  3. Crie um método get\_current\_weather(city\_name: str) que:  
     a. Construa a URL com parâmetros dinâmicos (city\_name, APPID e units=metric).  
     b. Faça a requisição GET usando a biblioteca requests.  
     c. Trate a resposta JSON, extraindo e normalizando apenas os campos relevantes para o modelo WeatherData.  
     d. Retorne os dados formatados como um objeto do modelo WeatherRecord (Pydantic).

**-> AGUARDAR CONFIRMAÇÃO E COMMIT**

**PASSO 4: Pipeline de Ingestão**

* **Commit Sugerido:** feat: Script de ingestão de dados e pipeline funcional
* **Ações:**
  1. Crie o script ingest\_data.py (na raiz) que orquestre a extração e o armazenamento.
  2. O script deve:  
     a. Carregar as variáveis de ambiente.  
     b. Chamar initialize\_db.py para garantir que as tabelas existam.  
     c. Instanciar o OpenWeatherClient.  
     d. Buscar o nome da cidade padrão do .env (CITY\_NAME).  
     e. Chamar o cliente para buscar os dados e persistir o dado no banco de dados.  
     f. Inclua tratamento de erro robusto (try/except) para falhas na API ou no banco.

**-> AGUARDAR CONFIRMAÇÃO E COMMIT**

### FASE 3: API e Containers

**PASSO 5: API RESTful (Setup e Dockerfile)**

* **Commit Sugerido:** feat: Setup da API FastAPI e Dockerfile básico
* **Ações:**
  1. Crie o arquivo main.py (na raiz) que configure a aplicação FastAPI e carregue as variáveis de ambiente.
  2. Crie um endpoint de saúde GET /health que retorne {"status": "API operacional"}.
  3. Gere o arquivo Dockerfile para conteinerizar a aplicação Python (usando uma imagem base leve como python:3.10-slim), instalando dependências e definindo o comando de *startup* com uvicorn.

**-> AGUARDAR CONFIRMAÇÃO E COMMIT**

**PASSO 6: Endpoints de Consulta de Dados**

* **Commit Sugerido:** feat: Endpoints de consulta de dados com DB
* **Ações:**
  1. Crie uma função de dependência (FastAPI dependency) get\_db para gerenciar a sessão do SQLAlchemy.
  2. Adicione um endpoint GET /api/v1/weather/{city\_name} em main.py para consultar o **último registro de clima** da cidade.
  3. Adicione um endpoint GET /api/v1/weather/history/{city\_name} para retornar uma lista de **todos os registros** daquela cidade.
  4. Garanta que o código utilize os *schemas* Pydantic (WeatherRecord) para validação de entrada e serialização de saída, garantindo documentação Swagger/OpenAPI clara.

**-> AGUARDAR CONFIRMAÇÃO E COMMIT**

**PASSO 7: Conteinerização Completa com Orquestração**

* **Commit Sugerido:** feat: Orquestração com Docker Compose
* **Ações:**
  1. Crie o arquivo docker-compose.yml para orquestrar os dois serviços: api e db.
  2. Configure a rede e o mapeamento de variáveis de ambiente.
  3. Configure um volume para persistência do banco de dados (PostgreSQL).
  4. Adicione um serviço de inicialização (init\_db) que garanta que os scripts initialize\_db.py e ingest\_data.py sejam executados com sucesso uma única vez após o banco de dados estar disponível.

**-> AGUARDAR CONFIRMAÇÃO E COMMIT**

### FASE 4: Documentação

**PASSO 8: Documentação Final**

* **Commit Sugerido:** docs: README e .gitignore
* **Ações:**
  1. Gere o conteúdo completo e claro do README.md com todos os tópicos exigidos na avaliação (Configuração, Execução, Acesso Remoto, etc.).
  2. Gere o arquivo .gitignore com entradas padrão para Python, Docker, e arquivos de ambiente (.env).

**-> PROJETO CONCLUÍDO. AGUARDAR CONFIRMAÇÃO FINAL.**