Especificação da Camada de Interação com IA

1. Objetivo Estratégico e Princípios

O objetivo estratégico desta camada é fornecer uma interface segura, resiliente, agnóstica a provedores e observável para a interação com modelos de IA (Inteligência Artificial). Os princípios que guiam esta arquitetura são:

- **Abstração e Desacoplamento:** O núcleo da aplicação deve ser completamente agnóstico a qual provedor de LLM (Large Language Model) está a ser utilizado. A troca de um provedor (ex: Google Gemini por OpenAI GPT) deve ser possível com alterações mínimas, confinadas a esta camada.
- **Segurança por Design:** A proteção das chaves de API do utilizador (BYOK - Bring Your Own Key) e o controle rigoroso sobre o uso e os custos são pilares fundamentais, não funcionalidades adicionais.
- **Resiliência e Fiabilidade:** A interação com serviços externos é inerentemente instável. A arquitetura deve incluir, por padrão, mecanismos de resiliência (Retries, Circuit Breakers) para lidar com falhas transitórias de rede ou do provedor.
- **Governança e Observabilidade:** Cada chamada à IA deve ser auditável, metrificada e registada. O sistema deve ter a capacidade de aplicar políticas complexas de uso e custo para prevenir abusos e gerir a economia do serviço.

2. Arquitetura de Pipeline de IA

A requisição de IA segue um pipeline de componentes especializados, cada um com uma responsabilidade única. Isto substitui o `AiIntegrationService` monolítico por uma arquitetura mais manutenível e extensível.

- **Componentes Principais:**
    1. `AIFacade`**:** O ponto de entrada único (façade) para a camada de aplicação. Orquestra a chamada através do pipeline.
    2. `PolicyEngine`**:** Intercetado primeiro, valida se a requisição está em conformidade com as políticas de negócio (limites de uso, permissões

de funcionalidade, controle de custos).

3. `ApiKeyVault`: Recupera e desencripta de forma segura a chave de API necessária para a requisição.
4. `ProviderRouter`: Seleciona o `Adapter` do provedor de IA apropriado com base na configuração do utilizador ou em regras de negócio.
5. `AIModelPort` **(Adapter):** A implementação concreta que traduz o `AIRequest` genérico para a chamada específica da API do provedor (ex: `GeminiAdapter`, `OpenAIAdapter`).

- **Fluxo de Execução:**
  1. Um Caso de Uso da camada de aplicação invoca o `AIFacade` com um `AIRequest`.
  2. O `AIFacade` primeiro chama o `PolicyEngine` para validar a requisição. Se a validação falhar, o processo é interrompido.
  3. O `AIFacade` invoca o `ApiKeyVault` para obter a chave de API desencriptada do utilizador.
  4. O `AIFacade` passa a requisição para o `ProviderRouter`, que retorna o `Adapter` correto.
  5. O `AIFacade` invoca o método `execute` do `Adapter` selecionado, passando a chave e a requisição.
  6. O `Adapter` lida com a lógica de resiliência (Retry/Circuit Breaker) e retorna um `AIResponse` ou lança um `AIError`.
  7. O `AIFacade` regista a utilização e os custos (através de um `UsageMeter`) antes de retornar a resposta ao Caso de Uso.

3. Contratos de Domínio (Ports)

As interfaces que definem a comunicação entre a camada de aplicação e a camada de IA são mantidas e enriquecidas.

- `AIModelPort` **(Interface):** O contrato central que define a interação com um modelo de IA (`execute` e `executeStream`). Cada provedor terá uma implementação.
- `AIRequest` **(Objeto de Comando):** Encapsula todos os detalhes de uma requisição.

    - **Campos:** `taskType` (Enum), `input` (multimodal, suportando texto e imagem), `config` (`ModelConfig`), `userContext`, e `tools` (para suportar function calling).

- `AIResponse` **e** `AIError`**:** A estrutura de resposta padronizada e a taxonomia de erros são mantidas para garantir um tratamento de erros consistente e resiliente.

4. Segurança e Governança

- **Armazenamento de Chaves (`ApiKeyVault`):**
    - **Estratégia:** Será implementado um cofre de chaves utilizando **Envelope Encryption**. As chaves de API dos utilizadores são encriptadas com uma Chave de Encriptação de Dados (DEK) única. Cada DEK, por sua vez, é encriptada com uma Chave de Encriptação de Chaves (KEK) mestra, gerida por um serviço de KMS (Key Management Service) dedicado (ex: AWS KMS, Google Cloud KMS).
    - **Justificativa:** Esta abordagem garante que os dados em repouso são seguros e que a chave mestra nunca é exposta à aplicação, oferecendo um padrão de segurança de nível empresarial.
- **Controle de Uso (`PolicyEngine`):**
    - **Estratégia:** Um motor de regras que permite a definição de políticas de governança complexas. Ele consultará o plano de subscrição do utilizador e o seu uso atual para:
        1. Aplicar limites de taxa (rate limiting).
        2. Verificar o saldo de "unidades de IA" (ver seção 4 da resolução anterior).
        3. Restringir o acesso a modelos ou funcionalidades específicas com base no plano.

5. Operações e Resiliência

- **Resiliência:** A lógica de Retry e Circuit Breaker será implementada dentro de cada `Adapter` utilizando uma biblioteca padrão como Resilience4j. As políticas serão configuráveis e acionadas com base nos `AIError.Code` retornados (ex: tentar novamente em `PROVIDER_UNAVAILABLE`, mas abrir o circuito em falhas repetidas).

- **Observabilidade:**
  - **Logs Estruturados (JSON):** Mandatório para todas as operações, contendo `traceId`, `tenantId`, `provider`, `taskType`, duração e custos. **Segredos nunca serão registados.**
  - **Métricas (Micrometer):** Exportação de métricas essenciais como latência por provedor, contagem de tokens (entrada/saída), taxas de erro e atividade do Circuit Breaker.
  - **Tracing Distribuído (OpenTelemetry):** O `traceId` será propagado desde a requisição HTTP inicial até a chamada final à API externa, permitindo uma depuração completa do ciclo de vida da requisição.