Especificação da Camada de Segurança

1. Princípios Fundamentais

A segurança no AxonAI é um pilar fundamental da arquitetura, não uma camada adicional. Guiamo-nos pelos seguintes princípios:

- **Defesa em Profundidade:** A segurança é aplicada em múltiplas camadas. A falha de um único controle de segurança não deve comprometer o sistema como um todo.
- **Princípio do Menor Privilégio:** Cada componente e utilizador do sistema deve ter apenas as permissões estritamente necessárias para realizar as suas funções.
- **Segurança por Design (*Security by Design*):** A segurança é considerada desde o início do ciclo de vida do desenvolvimento, com padrões e práticas seguras integradas em todas as camadas da aplicação.
- ***Zero Trust* (Confiança Zero):** Nenhuma requisição, seja interna ou externa, é confiável por padrão. Toda interação deve ser autenticada e autorizada.

2. Autenticação e Gestão de Sessão

A gestão de sessão será implementada utilizando o padrão de **Refresh Token Rotation** para maximizar a segurança contra o roubo de tokens.

- **Estratégia:**
    1. **Login:** Após uma autenticação bem-sucedida, o servidor gera e retorna dois tokens:
        - **Access Token:** Um JSON Web Token (JWT) de curta duração (15 minutos), assinado com uma chave assimétrica (RS256). Contém as permissões (`claims`) do utilizador.
        - **Refresh Token:** Um token opaco, criptograficamente seguro e de longa duração (7 dias). Este token é armazenado numa tabela `refresh_tokens` no banco de dados, associado ao `userId` , e marcado como ativo.

2. **Acesso a Recursos:** O cliente envia o `Access Token` no cabeçalho `Authorization: Bearer <token>` para aceder a endpoints protegidos.
3. **Renovação de Sessão (Rotação):**
    - Quando o `Access Token` expira, o cliente envia o `Refresh Token` para um endpoint específico (`/api/v1/auth/refresh`).
    - O servidor valida o `Refresh Token` contra o banco de dados.
    - Se válido, o servidor **invalida** o `Refresh Token` antigo e gera um **novo par** de `Access Token` e `Refresh Token`, retornando-os ao cliente.
4. **Detecção de Roubo:** Se o servidor receber um `Refresh Token` que já foi invalidado (reutilizado), isso indica um potencial roubo. O sistema irá invalidar imediatamente **toda a família de tokens** para aquele utilizador, forçando um novo login em todos os dispositivos.

- **Justificativa:** A curta duração do `Access Token` limita a janela de exposição em caso de roubo. A rotação e a detecção de reutilização do `Refresh Token` fornecem um mecanismo robusto para revogar sessões e proteger contra ataques de longo prazo.

3. Autorização

A autorização é a responsabilidade da Camada de Aplicação (`application.service`).

- **Estratégia:** Antes de executar qualquer lógica de negócio, o serviço de aplicação deve:
    1. Carregar o recurso solicitado (ex: `Workspace`, `Project`) a partir da camada de persistência.
    2. Verificar se o `userId` (extraído do JWT pelo `Controller`) tem permissão para aceder ou modificar aquele recurso (ex: é membro do `Workspace`).
    3. Se a verificação falhar, o serviço deve lançar uma `ForbiddenException`, que será traduzida para uma resposta **HTTP 403 Forbidden**.

4. Segurança de Credenciais e Segredos

- **Hashing de Senha:**
    - **Padrão:** O algoritmo **Bcrypt** é mandatório para o hashing de senhas de utilizadores com autenticação local. Nenhuma senha deve ser armazenada em texto plano ou com algoritmos de hash fracos.

- **Segurança das Chaves BYOK (Bring Your Own Key):**
  - **Padrão:** Será implementado o padrão **Envelope Encryption** com um KMS (Key Management Service) dedicado (ex: AWS KMS, Google Cloud KMS).
  - **Fluxo:**
    1. Quando um utilizador submete uma chave de API, o `ApiKeyVault` solicita uma nova Chave de Encriptação de Dados (DEK) ao KMS.
    2. O KMS retorna a DEK em texto plano e a mesma DEK encriptada com uma Chave de Encriptação de Chaves (KEK) mestra.
    3. A chave de API do utilizador é encriptada com a DEK em texto plano.
    4. A chave de API encriptada e a **DEK encriptada** são armazenadas no banco de dados. A DEK em texto plano é descartada da memória.
    5. Para desencriptar, o processo é invertido: a DEK encriptada é enviada ao KMS, que a desencripta usando a KEK mestra, e a DEK em texto plano resultante é usada para desencriptar a chave de API na memória da aplicação.
  - **Justificativa:** A aplicação nunca tem acesso à chave mestra (KEK), e as chaves de dados (DEKs) são armazenadas de forma segura, oferecendo um nível de segurança de padrão empresarial para os segredos dos utilizadores.
- **Segredos da Aplicação:** Todas as credenciais da aplicação (senhas de banco de dados, chaves de assinatura de JWT, credenciais do KMS) devem ser fornecidas à aplicação através de variáveis de ambiente seguras ou um serviço de gestão de segredos da plataforma de implantação (ex: Render, AWS Secrets Manager).

5. Segurança da Aplicação (Essenciais)

- **Validação de Input:** Todas as entradas de dados provenientes de fontes não confiáveis (utilizadores, APIs externas) devem ser rigorosamente validadas na fronteira da aplicação (DTOs, Controllers) para prevenir ataques de injeção (XSS, SQL Injection).
- **Proteção contra SSRF (Server-Side Request Forgery):** As funcionalidades que interagem com URLs fornecidas pelo utilizador devem validar e

restringir rigorosamente os endereços de destino para prevenir que o servidor seja usado como um proxy para atacar sistemas internos ou externos.

- **Cabeçalhos de Segurança HTTP:** A aplicação deve configurar cabeçalhos de segurança HTTP essenciais, como `Content-Security-Policy`, `Strict-Transport-Security` e `X-Content-Type-Options`, para proteger o frontend contra ataques comuns.

Ensino e Análise Final

Esta especificação eleva a segurança do AxonAI de um modelo "suficiente para o MVP" para um modelo de "defesa em profundidade" adequado para produção. A mudança mais significativa é a adoção de padrões que limitam o "raio de explosão" de um incidente de segurança. A rotação de refresh tokens garante que um token roubado tenha uma utilidade limitada no tempo. O Envelope Encryption com KMS garante que uma violação do banco de dados não exponha imediatamente as chaves de API dos utilizadores, pois o atacante ainda precisaria comprometer o KMS. Esta abordagem em camadas é a essência da engenharia de segurança moderna: assumir que falhas podem e vão ocorrer, e construir sistemas que são resilientes a essas falhas.