Implementando Data aNd Application Tracking (DNAT)

A partir do artigo *On the Tracking of Sensitive Data and Confidential Executions*

Davi Guimarães

28 de novembro de 2025

Resumo da apresentação

- Contexto: explosão de IA e coleta massiva de dados
- Problema: como rastrear uso de dados sensíveis em ambientes não confiáveis
- Proposta do artigo: DNAT (Data aNd Application Tracking)
- Minha contribuição: implementação prática do sistema DNAT

Sumário

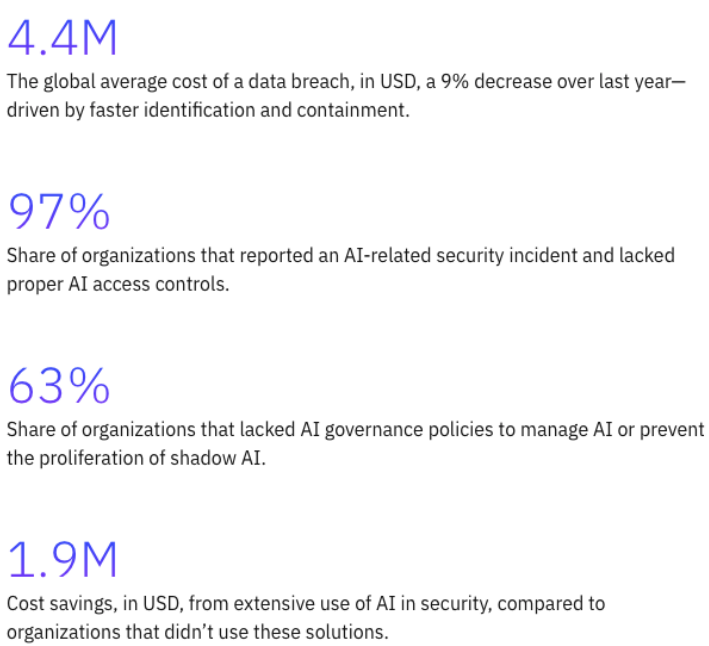
Por que falar de DNAT?

- Sistemas de **IA** e **aprendizado de máquina** dependem de grandes volumes de dados.
- Aplicativos, serviços on-line e dispositivos IoT coletam dados: localização, hábitos de consumo, saúde, preferências, etc.
- Muitas bases de dados parecem "anônimas", mas podem ser **re-identificadas** por correlação:
  - Ex.: idade, sexo, cidade, doença
  - + nome, cidade, idade, profissão
- Isso aumenta a preocupação com **confidencialidade**, **privacidade** e **uso responsável** desses dados.

Referências: Nascimento Jr. et al., *On the Tracking of Sensitive Data and Confidential Executions*, DEBS'20.

Exemplo real: vazamento de dados da Uber

- Em 2016, a Uber sofreu um incidente que expôs dados de ∼57 milhões de usuários e motoristas.
- Informações incluíam nomes, e-mails, números de telefone e dados de carteira de motorista de centenas de milhares de motoristas.
- A empresa ainda foi acusada de tentar **ocultar o vazamento**, pagando para que os atacantes apagassem os dados.
- Esse tipo de incidente mostra:
  - o risco de concentrar grandes volumes de dados sensíveis;
  - a importância de **auditar quem acessa o quê e quando**.



Fonte: IBM, *Cost of a Data Breach Report*.

Referências: Uber Technologies Inc., "2016 Data Security Incident".

Regulação e direito ao rastreio de uso

- Incidentes como Uber e Cambridge Analytica pressionaram pela criação/fortalecimento de leis de proteção de dados: **GDPR** (UE) e **LGPD** (Brasil).
- A GDPR garante aos titulares direitos como:
  - direito de ser informado sobre o tratamento;
  - direito de acesso aos dados;
  - direito de retificação e *right to be forgotten*;
  - direito de saber **quem** está usando os dados e **para quê**.
- Na prática, isso exige:
  - **rastrear execuções** de aplicações sobre dados sensíveis;
  - manter um histórico auditável, mesmo em nuvens públicas.

Referências: GDPR (Cap. 3 — direitos do titular); Comissão Europeia, "Data protection explained"; casos Facebook/Cambridge Analytica.

Proposta do DNAT em uma frase

Ideia central

DNAT (Data aNd Application Tracking) é uma plataforma para rastrear execuções de aplicações sobre dados sensíveis em ambientes não confiáveis, combinando blockchain e execução confidencial.

- **Blockchain (Ethereum):**
  - Uso de *smart contracts* para registrar: *registro de ativos*, *aquisição de direitos de acesso*, *execuções* e *revogações*.
  - Log imutável que permite auditar quem executou o quê, quando e sobre quais dados.
- **Execução confidencial:**
  - Aplicações e dados sensíveis só são executados em ambiente confiável (enclave), previamente atestado.
  - Detalhes de SGX, SCONE e CAS serão explicados na seção de **Background**.

Referências: Nascimento Jr. et al., DEBS'20 (arquitetura DNAT e uso de Ethereum + execução confidencial).

Cenários cobertos e avaliação do DNAT

- DNAT foi pensado para cenários em que:
  - os **dados são confidenciais** e a aplicação é pública; ou
  - a **aplicação é confidencial** e os dados são públicos.
- A ideia geral é permitir que:
  - quem possui dados possa controlar quem executa aplicações sobre eles;
  - quem desenvolve aplicações possa rodá-las em dados de terceiros sem precisar enxergar o conteúdo bruto.
- No trabalho original, os autores analisam de forma geral:
  - o custo adicional de registrar operações na blockchain;
  - o impacto de rodar a aplicação em um ambiente protegido.
- Nesta apresentação, o foco é:
  - mostrar como essas ideias foram traduzidas em uma **implementação prática**;
  - comentar o que funcionou bem e o que ainda pode melhorar.

Referências: Nascimento Jr. et al., DEBS'20 (arquitetura DNAT e uso de Ethereum + execução confidencial).

Blockchain: visão geral

- **Livro-razão distribuído** que registra transações em blocos encadeados.
- Cada bloco contém:
  - um conjunto de transações;
  - um cabeçalho com hash do bloco anterior, timestamp, etc.
- A cadeia de blocos torna muito difícil alterar o histórico: para mudar um bloco antigo, seria necessário refazer todos os posteriores.
- Em redes públicas (Bitcoin, Ethereum), qualquer nó pode verificar a consistência do histórico.

Referências: Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.; Buterin, *Ethereum White Paper*, 2013.

Transações e operações

- **Transação** é a unidade básica de operação:
  - transferir valor entre endereços;
  - chamar funções de um *smart contract*.
- Cada transação inclui:
  - endereço de origem e destino;
  - valor;
  - dados opcionais (parâmetros de função);
  - assinatura digital do remetente.
- Transações são agrupadas em blocos e, depois de confirmadas, passam a fazer parte do histórico imutável da cadeia.
- No DNAT, operações como registrar ativo ou conceder acesso são implementadas como transações em um contrato Ethereum.

Referências: Buterin, *Ethereum White Paper*, 2013.; Wood, *Ethereum: A Secure Decentralised Generalised Transaction Ledger*, 2014.

Gas e custo de execução

- Na Ethereum, cada operação de contrato consome **gas**: uma medida do custo computacional.
- Ao enviar uma transação, o usuário define:
  - um limite de gas (*gas limit*);
  - quanto está disposto a pagar por unidade de gas.
- O custo final é aproximadamente:

$$\text{custo} \approx \text{gas usado} \times \text{preço do gas}.$$

- Isso evita abusos (transações maliciosas, loops infinitos) e faz com que quem usa a rede pague pelo recurso consumido.
- Para DNAT, cada registro ou atualização no contrato tem um custo, o que influencia o desenho da solução.

Referências: Wood, *Ethereum: A Secure Decentralised Generalised Transaction Ledger*, 2014.

Consenso em blockchain

- Em uma rede distribuída, os nós precisam concordar sobre:
  - qual é o próximo bloco válido;
  - qual é o histórico "oficial" de transações.
- Mecanismos de consenso:
  - **Proof of Work (PoW)**: mineradores competem resolvendo problemas criptográficos;
  - **Proof of Stake (PoS)**: validadores travam moedas e são escolhidos para propor/validar blocos.
- Hoje, a Ethereum usa PoS, enquanto o Bitcoin continua em PoW.
- O consenso é o que garante que o log onde o DNAT registra as operações não possa ser adulterado por um único ator.

Referências: Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.; Buterin, *Ethereum White Paper*, 2013.

Filtros de Bloom

- Estrutura de dados probabilística usada para testar se um elemento pertence a um conjunto.
- Propriedades:
  - se o filtro diz que **não** está, realmente não está;
  - se diz que está, pode ser um **falso positivo**.
- Implementação:
  - vetor de bits + várias funções de hash.
  - inserções e consultas são rápidas e usam pouca memória.
- Na Ethereum, filtros de Bloom nos cabeçalhos de bloco ajudam a localizar rapidamente eventos (logs) de contratos.
- Em DNAT, isso é útil para filtrar blocos que podem conter eventos de interesse (registro, acesso, revogação).

Referências: Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors", 1970.; Broder & Mitzenmacher, "Network Applications of Bloom Filters", 2004.

Smart contracts

- Programas que rodam em cima da blockchain, com:
  - código imutável após o *deploy*;
  - estado armazenado na própria blockchain.
- Expostos via funções que:
  - podem alterar o estado (transações);
  - ou apenas ler (chamadas de leitura, sem custo on-chain).
- Na Ethereum, são escritos tipicamente em Solidity e executados na EVM.
- No DNAT, o contrato é responsável por:
  - registrar ativos;
  - controlar quem tem direito de acesso;
  - registrar execuções e revogações.

Referências: Szabo, "Smart Contracts", 1994.; Buterin, *Ethereum White Paper*, 2013.

Bitcoin e Ethereum

- **Bitcoin**:
  - primeira criptomoeda amplamente adotada;
  - foco em pagamentos P2P;
  - script limitado, sem plataforma geral de contratos.
- **Ethereum**:
  - plataforma de blockchain programável;
  - suporta smart contracts Turing-completos;
  - hoje usa Proof of Stake para consenso.
- Para o DNAT:
  - Bitcoin é referência histórica de blockchain;
  - Ethereum é a base prática para implementar o contrato que rastreia ativos e execuções.

Referências: Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.; Buterin, *Ethereum White Paper*, 2013.

Execução confidencial e enclaves SGX

- **Objetivo:** proteger código e dados em execução mesmo em máquinas potencialmente maliciosas.
- **Intel SGX:**
  - extensão de hardware que permite criar enclaves;
  - enclaves são regiões de memória isoladas e criptografadas;
  - mesmo o sistema operacional e o hypervisor não devem conseguir ler ou alterar o conteúdo em claro.
- No DNAT, o Executor e o CAS rodam dentro de enclaves SGX.
- Dados e aplicações são armazenados criptografados e só são descriptografados dentro do enclave.

Medição e atestação remota

- Cada enclave tem um **measurement** (hash) que identifica o código e o estado inicial (*MRENCLAVE*).
- Na **atestação remota**:
  - o enclave gera um *quote* com seu measurement;
  - esse quote é verificado por um serviço de atestação;
  - um serviço externo (como o CAS) decide se confia naquele measurement.
- Se a atestação for bem-sucedida, o CAS pode liberar segredos (chaves, credenciais, configurações) para aquele enclave.
- Assim, o DNAT garante que apenas código autorizado recebe as chaves para acessar dados e aplicações.

Referências: Tutoriais de SGX sobre atestação remota; exemplos com SCONE.

SCONE, CAS e LAS

- **SCONE**:
  - plataforma para rodar aplicações em containers dentro de enclaves SGX;
  - facilita integração com Docker e Kubernetes.
- **CAS (Configuration and Attestation Service)**:
  - serviço SGX que guarda chaves e configurações sensíveis;
  - só libera segredos para enclaves que passam pela atestação;
  - pode rodar na nuvem ou na infraestrutura do dono dos dados.
- **LAS (Local Attestation Service)**:
  - roda em cada host;
  - gera os *quotes* e intermedia a comunicação com o CAS.
- No DNAT, Executor e CAS interagem via SCONE/LAS para que as chaves só sejam entregues a código íntegro e atestado.

Referências: Documentação do SCONE (CAS/LAS); trabalhos que usam SCONE.

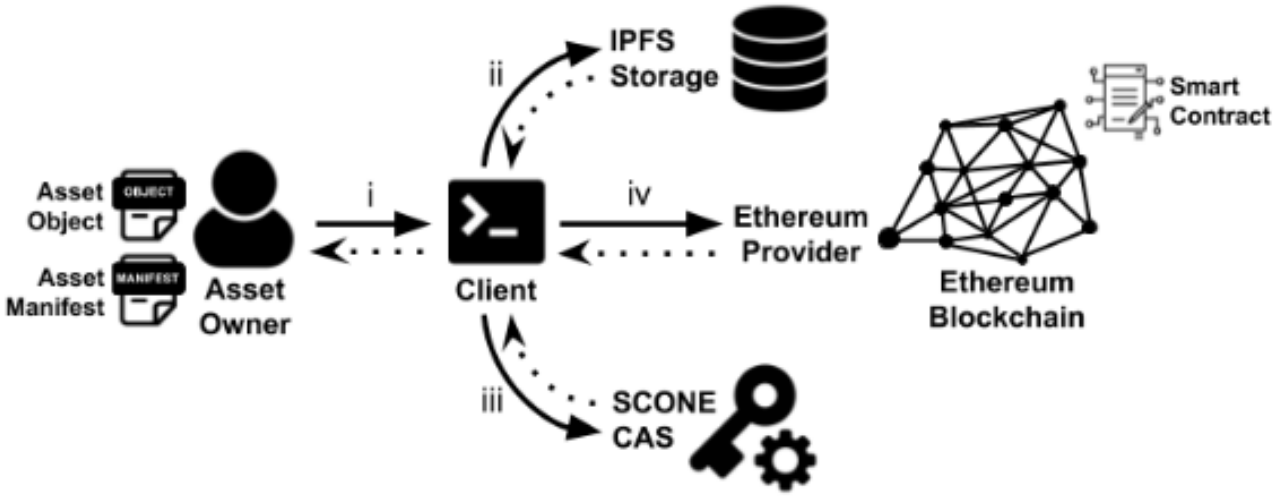
Componentes principais do DNAT

- **User**:
  - registra dados e aplicações como ativos na plataforma;
  - consome ativos executando aplicações sobre dados.
- **Notação dos ativos**:
  - δ: um **dataset** (conjunto de dados sensíveis);
  - π: uma **application** (código que será executado);
  - κ: **chave simétrica** usada para criptografar δ ou π.
- **Client**: programa que o usuário usa para registrar δ e π, gerar κ, enviar dados criptografados e interagir com a blockchain.
- **Ethereum Provider**: nó/serviço Ethereum usado pelo Client para enviar transações e ler o contrato DNAT.

Executor, CAS e fluxo geral

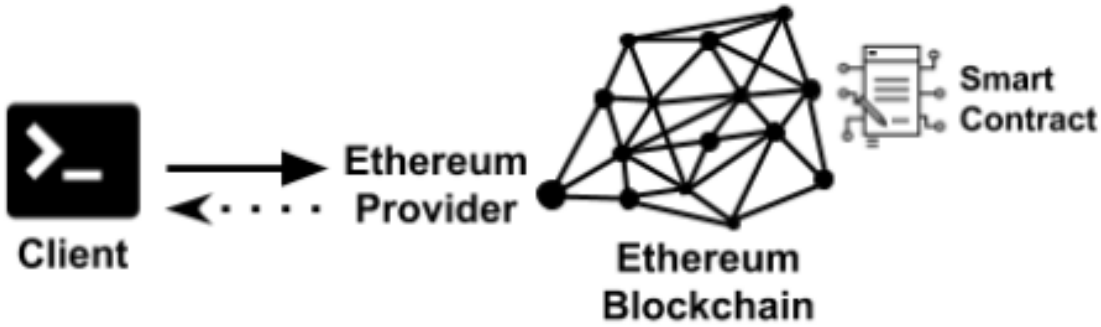
- **Executor**:
  - roda em máquina com Intel SGX;
  - verifica na blockchain se o User tem direito de executar π sobre δ;
  - baixa δ e π criptografados e executa tudo dentro do enclave.
- **CAS**:
  - armazena chaves κ associadas a cada ativo;
  - só libera κ para executores atestados (código correto em SGX).
- Visão rápida do fluxo:
  - User, via Client, registra δ e π e envia κ para o CAS;
  - compra o direito de executar π sobre δ no contrato DNAT;
  - Executor consulta o contrato, fala com o CAS, obtém κ e roda a execução de forma confidencial.

1. Registro de ativo



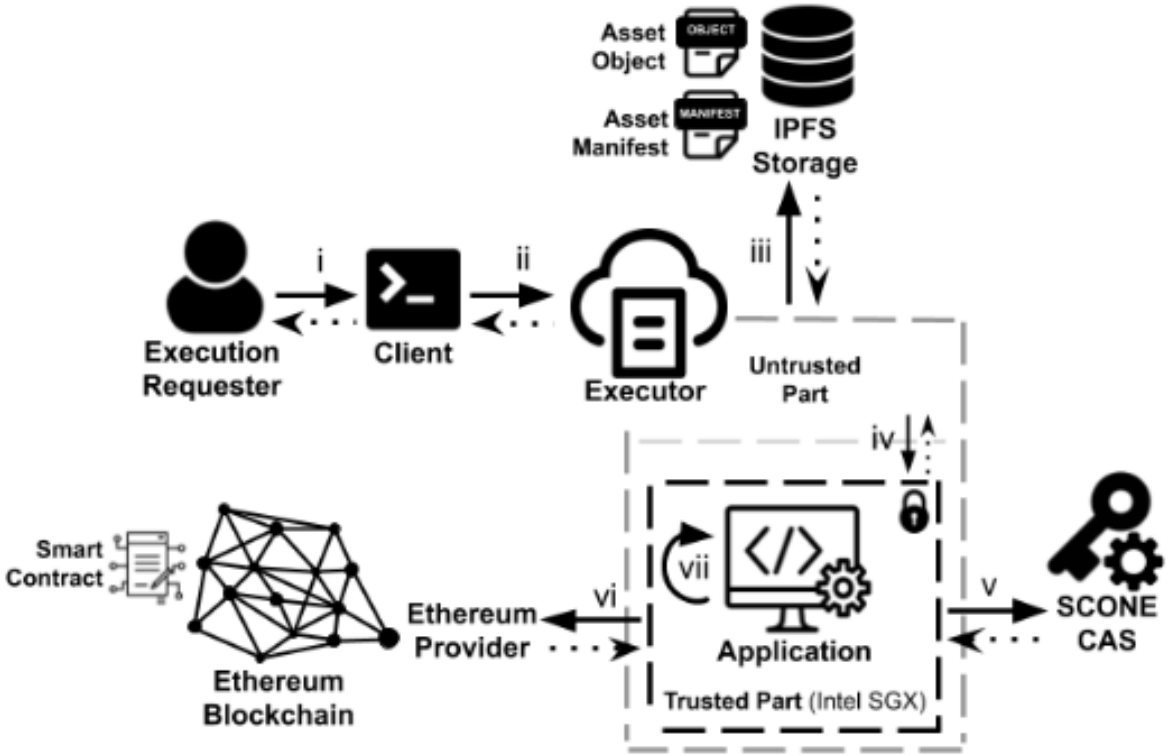
- O dono escolhe um ativo α:
  - $\alpha = \delta$ (dataset) ou $\alpha = \pi$ (aplicação).
- O **Client**:
  - gera uma chave κ para α;
  - criptografa α e envia para o IPFS;
  - cria um manifesto com metadados (descrição, preço, etc.);
  - registra tudo em um contrato DNAT na Ethereum.

2. Aquisição de direito de acesso



- Um usuário quer executar π sobre um dataset δ.
- Via **Client**, ele:
  - escolhe δ e π no catálogo;
  - envia uma transação para o contrato DNAT pagando em Wei;
  - a transação registra que ele tem direito de executar π sobre δ.
- A partir daí, qualquer Executor da plataforma pode usar esse direito para rodar a execução de forma confidencial.

3. Execução da aplicação



- O Executor:
  - verifica no contrato se o usuário tem direito $\pi \rightarrow \delta$;
  - baixa δ e π criptografados do IPFS;
  - faz atestação com o CAS e recebe as chaves κ;
  - descriptografa e executa π sobre δ dentro do enclave SGX.
- O resultado sai do enclave.

4. Catálogo, rastreamento e revogação

- **Catálogo**:
  - lista datasets δ e aplicações π registrados;
  - mostra manifestos (descrição, preço, condições).
- **Rastreamento**:
  - o contrato guarda eventos de registro, aquisição e execução;
  - é possível ver quem executou π sobre δ e quando.
- **Revogação**:
  - o dono pode despublicar um ativo;
  - novos direitos deixam de ser concedidos, mas o histórico permanece.

Minha implementação: visão geral

- Adaptei a ideia do DNAT para um ambiente de testes:
  - rede Ethereum local com **Hardhat**;
  - **VM bare metal** na Gcore para o Executor SGX;
  - **Client** como aplicação web;
  - **IPFS local** para armazenar os ativos criptografados.
- Foco:
  - reproduzir os principais fluxos: registro, acesso e execução;
  - visualizar como os componentes conversam na prática.

Infraestrutura: Hardhat, VM bare metal e IPFS local

- **Hardhat** (simulador Ethereum):
  - rede local para fazer *deploy* do contrato DNAT;
  - blocos rápidos, contas de teste e inspeção de transações.
- **VM bare metal** na Google Cloud:
  - máquina com SGX habilitado para rodar o Executor;
  - conectada à rede Hardhat e ao serviço de CAS.
- **IPFS local**:
  - nó IPFS rodando na minha infraestrutura;
  - armazena δ e π já criptografados com κ;
  - o contrato guarda apenas os hashes (conteúdo fica fora da blockchain).

Client web

Client como aplicação web

- Interface para:
  - registrar datasets δ e aplicações π;
  - listar ativos e ver manifestos;
  - adquirir direito de executar π sobre δ;
  - disparar a execução no Executor.
- A aplicação web fala com:
  - a rede Hardhat (contrato DNAT);
  - o backend que aciona o Executor na VM bare metal.

Demo

Limitações da minha implementação

- Não existe código oficial do DNAT disponível no artigo.
- Precisei assumir alguns detalhes de arquitetura e de fluxo.
- Algumas partes da integração com SCONE e CAS foram adaptadas em relação ao que é descrito teoricamente.

Limitações: SCONE, CAS e MRENCLAVE

- O SCONE, na prática, permite apenas uma sessão por aplicação.
- No meu protótipo:
  - registrei no CAS um **segredo único** para o app;
  - não consegui amarrar esse segredo a um **MRENCLAVE** específico;
  - a aplicação roda apenas uma rodada por execução.
- Isso é diferente da visão do artigo, que assume uma associação mais rígida entre MRENCLAVE, sessões e chaves.

Limitações: documentação do SCONE

- A documentação pública do SCONE é razoável, mas:
  - parte do material está desatualizado;
  - algumas funcionalidades mais avançadas ficaram privadas;
  - precisei complementar com exemplos, issues e tentativa e erro.
- Isso impactou:
  - a forma como configurei sessões;
  - a forma como integrei CAS, LAS e Executor.

Funcionalidades não implementadas

- **Whitelist de aplicações**:
  - o artigo propõe whitelists para permitir execuções “de graça”;
  - no meu protótipo, essa lógica não foi implementada.
- **Revogação**:
  - não implementei a operação de revogar publicação de ativos;
  - uma vez registrado, o ativo só pode ser “ignorado” pelo cliente, mas não há revogação on-chain.

Limitações: rede real

- Todos os testes foram feitos em:
  - rede local com Hardhat;
  - ambiente de nuvem com VM bare metal, mas sem uma testnet pública.
- Não foi feito:
  - deploy do contrato em uma rede Ethereum pública (testnet/mainnet);
  - medições reais de custo de gas em um ambiente de produção.

Como usei IA neste projeto

- **Explicação do artigo**:
  - usei IA para destrinchar partes mais densas do texto;
  - resumir seções longas e checar se eu tinha entendido os fluxos corretamente.
- **Busca por referências**:
  - apoio para encontrar trabalhos relacionados (SGX, SCONE, blockchain);
  - ajuda para lembrar nomes de autores, conferências e termos.
- **Vibecoding**:
  - usei IA como "par de programação" para rascunhar código e refatorar trechos;
  - adaptei tudo para a minha realidade e revisei manualmente antes de usar.

Ferramentas de IA utilizadas

- **Cursor (Sonnet 4.5)**:
  - integrado ao editor de código;
  - usado para sugerir trechos de implementação, testes e pequenos ajustes.
- **GPT 5 Pro**:
  - apoio na organização das ideias da apresentação;
  - ajuda para estruturar os slides, revisar texto e explicar conceitos teóricos.

Perguntas & Respostas

Espaço para perguntas.