Como tornar-se um Desenvolvedor Nostr

Anderson Juhasc

Sumário

> "Tudo que você fizer será insignificante, mas é muito importante que você faça."
>
> — Mahatma Gandhi

Introdução

1. O que é o Nostr?

História e conceito

O Nostr (Notes and Other Stuff Transmitted by Relays) é um protocolo aberto e simples para comunicação descentralizada. Diferente de redes sociais tradicionais como Twitter, Facebook ou até mesmo plataformas descentralizadas mais complexas como Mastodon, o Nostr não depende de servidores centrais, nem de consenso entre nós — sua filosofia é radicalmente minimalista: tudo gira em torno de mensagens assinadas e retransmitidas.

A ideia por trás do Nostr surgiu como uma reação a problemas recorrentes em plataformas de comunicação: censura, banimentos arbitrários, controle centralizado, dependência de infraestrutura fechada, e dificuldade de portabilidade de identidade. O criador do protocolo, que assina como fiatjaf, buscava uma alternativa que fosse não apenas resistente à censura, mas também simples o bastante para que qualquer pessoa pudesse entender, implementar ou adaptar.

O protocolo começou a ganhar atenção em 2021, mas foi em 2022 e 2023 que o ecossistema realmente se expandiu. Com o apoio de desenvolvedores entusiastas e figuras públicas como Jack Dorsey, ex-CEO do Twitter, que doou recursos e promoveu o Nostr, surgiram diversos clientes e relays, além de propostas de extensão conhecidas como NIPs (Nostr Implementation Possibilities). Esses documentos definem padrões opcionais e ajudam a coordenar inovações dentro da rede.

O Nostr não é uma plataforma, mas uma base para criar qualquer tipo de aplicação social: redes sociais, chats privados, marketplaces, blogs, fóruns, etc. Sua estrutura é propositalmente simples: usuários criam eventos, assinam com suas chaves privadas, e enviam para relays que retransmitem para outros usuários interessados. Não há intermediários obrigatórios nem algoritmos ocultos.

A descentralização do Nostr não é uma promessa futura — ela já está presente desde a concepção do protocolo. A identidade de cada usuário é apenas sua chave pública, e sua soberania sobre seus dados é garantida pela posse da chave privada. Se um relay o censura, basta publicar em outro. Se um cliente deixa de funcionar, basta usar outro. É um protocolo que aposta na diversidade e na liberdade como elementos centrais da infraestrutura.

Assim, o Nostr representa uma ruptura com o modelo tradicional da internet social. Ele não depende de confiança em terceiros. Ele não força consenso. Ele apenas transmite notas e outras coisas — de forma aberta, resistente e direta.

Diferenças com outras plataformas descentralizadas

O Nostr se destaca por sua simplicidade e foco direto na comunicação descentralizada. Enquanto outras iniciativas buscam descentralização por meio de soluções complexas, muitas vezes ligadas a blockchains ou sistemas de governança elaborados, o Nostr segue um caminho mais minimalista e eficiente.

Uma das principais diferenças do Nostr é que ele **não precisa de blockchain para funcionar**. Embora muitos projetos descentralizados estejam vinculados a uma rede blockchain para validar identidades ou registrar interações, o Nostr se baseia apenas em **criptografia de chave pública e relays simples**. Essa abordagem o torna leve, rápido e fácil de implementar.

Ao invés de depender de consenso global ou de mineração, como acontece no Bitcoin, o Nostr adota um modelo em que **qualquer evento é válido se for corretamente assinado** por uma chave privada. Os relays simplesmente armazenam e retransmitem os eventos — eles não decidem o que é verdadeiro ou falso, apenas oferecem infraestrutura para a comunicação.

A identidade do usuário é sua chave pública, assim como no Bitcoin, onde uma chave pública representa um endereço. Essa chave é usada para assinar eventos, o que garante autenticidade sem depender de intermediários. Se um usuário tem a chave privada, ele tem controle total sobre sua conta — exatamente como o princípio "seu Bitcoin, suas chaves".

Além disso, o Nostr é modular por natureza. Qualquer um pode criar um cliente, um relay, ou até uma nova proposta de funcionalidade (NIP), sem pedir permissão a ninguém. Isso estimula um ecossistema altamente experimental e resiliente, onde inovações podem surgir de qualquer parte do mundo, sem barreiras técnicas ou políticas.

Enquanto redes tradicionais centralizadas impõem regras e algoritmos de visibilidade, o Nostr permite que cada cliente implemente a interface e a lógica de interação que desejar. O poder é transferido para as pontas — os usuários e os desenvolvedores — em vez de estar concentrado em corporações.

Essa liberdade vem com um desafio: a descentralização exige mais responsabilidade do indivíduo. Não há "esqueci minha senha", não há "denunciar e esperar moderação central". O usuário precisa cuidar de suas chaves e escolher os relays e clientes com os quais interage. Mas é justamente essa responsabilidade que garante a soberania digital que o Nostr propõe.

A filosofia por trás do Nostr

A filosofia do Nostr é profundamente enraizada em princípios de liberdade individual, resistência à censura e descentralização real. Em vez de tentar "melhorar" o modelo de redes sociais tradicionais, o Nostr parte do zero, com uma proposta simples: permitir que qualquer pessoa publique e consuma informações sem depender de intermediários confiáveis.

No coração do protocolo está a ideia de que **qualquer um deve poder se expressar**, sem medo de ser silenciado por plataformas centralizadas, governos ou corporações. O Nostr não tenta julgar ou moderar o conteúdo — ele apenas fornece o meio técnico para que mensagens sejam transmitidas entre pares. Cabe aos clientes e aos usuários decidirem o que querem ver, filtrar ou compartilhar.

Essa neutralidade técnica é um diferencial. O protocolo não tem "curadoria" embutida. Ele não tenta resolver disputas sociais, políticas ou culturais — ele apenas oferece a infraestrutura para que essas discussões possam existir livremente. Isso se alinha à ética cypherpunk e aos ideais que também influenciaram a criação do Bitcoin: **liberdade através da criptografia, e confiança minimizada em terceiros.**

Outro aspecto central da filosofia do Nostr é o **empoderamento do desenvolvedor**. Com uma base técnica acessível e bem documentada, qualquer pessoa pode criar um cliente, um relay ou uma nova ferramenta sem precisar integrar-se a uma rede complexa ou obter permissão. Isso gera um ambiente altamente criativo, onde soluções experimentais florescem e a inovação acontece nas bordas, e não no centro.

A confiança no indivíduo também se reflete na forma como o Nostr lida com identidade e controle. Sua chave privada é sua identidade. Não há recuperação de conta via e-mail. Não há login com redes sociais. Você é soberano — e, portanto, responsável — por sua presença e atividade na rede.

Em resumo, o Nostr não é apenas um protocolo técnico: é uma declaração de valores. Ele representa uma visão da internet onde os usuários não são produtos, onde os desenvolvedores não precisam de autorização para inovar, e onde a liberdade de expressão é uma característica do sistema, não uma exceção sujeita a aprovação.

2. Preparando-se para o Desenvolvimento

Ferramentas e ambientes de desenvolvimento

Você pode desenvolver aplicações Nostr com praticamente qualquer linguagem de programação. Como o protocolo utiliza WebSockets para comunicação e mensagens no formato JSON, a integração é simples e acessível, permitindo desde pequenos scripts e até aplicações robustas.

A seguir, apresento uma visão geral das ferramentas e linguagens mais comuns utilizadas pela comunidade Nostr atualmente.

Linguagens e bibliotecas

Você pode usar qualquer linguagem que ofereça suporte a WebSockets e JSON, criar suas próprias bibliotecas e SDKs, ou aproveitar as opções já disponíveis para facilitar o desenvolvimento:

- **JavaScript/TypeScript**: Ideal para clientes web e extensões de navegador. As principais bibliotecas são `nostr-tools`, e a `ndk`. Com elas, você consegue gerar chaves, assinar eventos, conectar-se a relays e muito mais.
- **Python**: Existem implementações como `python-nostr`, ótimas para scripts, bots, automações ou aplicações no lado do servidor.
- **Go**: A linguagem tem ganhado tração por sua performance e simplicidade. A biblioteca `go-nostr` é bastante completa, é usado em relays e ferramentas de comando de linha (CLI).
- **Rust**: A biblioteca `rust-nostr` é a principal opção na linguagem, ideal para projetos que buscam performance e segurança, especialmente ao desenvolver relays.
- **Dart/Flutter**: Já existem clientes móveis sendo desenvolvidos com Flutter, e a comunidade adaptou bibliotecas como `Dart NDK` e `dart-nostr` para facilitar a integração com o ecossistema.

Gerenciador de Dependências

Assim como o `apt` no Linux ou o recurso de adicionar/remover programas no Windows, cada linguagem costuma ter seu próprio gerenciador de dependências, que facilita a instalação, atualização e remoção de bibliotecas usadas no projeto.

Aqui estão alguns exemplos comuns:

- **npm ou yarn**: Gerenciadores de pacotes para projetos JavaScript/TypeScript (Node.js e web).
- **pip**: Usado em projetos Python para instalar e gerenciar bibliotecas.
- **cargo**: Ferramenta oficial do Rust para compilar, testar e gerenciar dependências.
- **go mod**: Sistema de módulos do Go para controle de versões e pacotes.
- **pub**: Gerenciador de pacotes oficial para projetos Dart e Flutter.

Relays

Para testar e publicar eventos, você pode se conectar a relays públicos ou rodar o seu próprio.

Alguns relays públicos populares incluem:

- `wss://relay.damus.io`
- `wss://nos.lol`
- `wss://nostr.wine`
- `wss://relay.nostr.band`

Rodar um relay local é útil para testes, especialmente se você quiser entender o que acontece nos bastidores.

Exemplos de software de relay:

- **nostream**: Um relay Nostr escrito em TypeScript e utiliza o banco de dados PostgreSQL.
- **nostr-rs-relay**: Um relay minimalista escrito em Rust que armazena dados usando SQLite.
- **go-nostr-relay**: Relay escrito em Go baseado na biblioteca relayer com suporte a SQLite3, PostgreSQL e MySQL.

Signers e gerenciadores

Como a identidade no Nostr é construída a partir de chaves criptográficas, é essencial utilizar ferramentas para gerá-las, gerenciá-las e usá-las na assinatura de eventos.

Algumas opções:

- **Extensões para Navegador**: Como *nos2x*, *Nostrame* e *Alby* permitem gerenciar identidades e assinar eventos diretamente no navegador.
- **Aplicativos móveis**: O *Amber* é um assinador de eventos Nostr que mantém a chave privada isolada em um app dedicado. Já o *Nowser* permite gerenciar e assinar chaves Nostr com segurança, oferecendo suporte a múltiplos métodos de autenticação.
- **Aplicaticos para Web**: O *Nsec App* é um gerenciador de identidades Nostr simples e seguro, focado em gerar, importar e usar chaves para assinar eventos diretamente no navegador.
- **Linha de comando ou bibliotecas**: Você pode gerar e usar chaves com ferramentas de CLI como `nak` ou até mesmo usando `openssl`, além de bibliotecas como `nostr-tools` e `go-nostr`.

Ferramentas Úteis

Aqui estão algumas ferramentas que podem facilitar o desenvolvimento e os testes com Nostr, seja para interagir com relays, assinar e enviar eventos, ou até integrar pagamentos via Lightning.

- **Postman ou Insomnia**: Para testar conexões WebSocket e enviar eventos manualmente.
- **Wallets Lightning (opcional)**: Caso deseje testar pagamentos lightning via Nostr (zaps), você pode usar carteiras como Phoenix, Breez ou Wallet of Satoshi.

Versionamento de Código

Controlar o histórico do seu código é essencial para qualquer projeto. Utilize o **Git** para rastrear

mudanças e colaborar com outras pessoas. Sempre que possível, hospede seu repositório em plataformas como o **GitHub** ou em alternativas mais descentralizadas, como o **Codeberg**.

Editores e ambiente local

Para começar rápido:

- Um editor de texto moderno como **VS Code**, **NeoVim** ou **Zed**.
- Uma linguagem de servidor instalada localmente com suporte a WebSockets (Node.js, Python, Go, etc.).
- Um navegador moderno que permita conexão com WebSocket `wss://`.

A vantagem do Nostr é que você pode criar aplicações completas com um simples HTML + JavaScript, sem necessidade de um servidor.

Clientes e interfaces de referência

Você pode estudar o funcionamento do protocolo observando como os principais clientes se comportam:

- **noStrudel** (web)
- **Iris.to** (web)
- **Snort.social** (web)
- **Coracle.social** (web)
- **Damus** (iOS/macOS)
- **Amethyst** (Android)
- **Amethys** (Android)
- **Nostr Console** (CLI)

A maioria desses projetos é open-source, e você pode ler o código, clonar e testar localmente.

Testando com redes reais

No Nostr, não é necessário um ambiente de testes separado, como em blockchains. Tudo o que você faz é real. Por isso, crie identidades de teste, conecte-se a um relay, seja ele próprio ou público, e comece a publicar notas. No entanto, é importante ter cuidado para não expor dados privados inadvertidamente.

Com essas ferramentas em mãos, você já pode começar a criar seu primeiro projeto.

O que você precisa saber antes de começar

Antes de mergulhar no código, é importante entender alguns conceitos fundamentais do Nostr que influenciam diretamente a forma como você vai projetar suas aplicações. O protocolo é simples, mas essa simplicidade esconde uma mudança profunda de paradigma em relação ao desenvolvimento tradicional para a web.

Identidade = Chave Pública

No Nostr, não existe e-mail, login ou senha. A identidade de um usuário é sua **chave pública**, e ele assina todas as suas ações com uma **chave privada**. Isso significa que:

- Não há recuperação de conta tradicional.
- Se você perde sua chave privada, perde o acesso à identidade.
- Se alguém obtém sua chave privada, assume completamente o seu perfil.

Essa estrutura é semelhante ao que acontece com carteiras de Bitcoin. É fundamental adotar práticas seguras desde o início, como armazenar suas chaves privadas de forma segura, usar hardware wallets, apps como *Amber* ou extensões como *nos2x*, *Nostrame* e *Alby*, e evitar jamais expor a chave privada em código ou navegadores.

Tudo é um Evento

No Nostr, tudo que acontece é modelado como um **evento JSON** assinado. Não há banco de dados relacional, nem APIs REST. Você cria um objeto JSON com campos como `kind`, `content`, `tags`, `created_at`, etc., assina esse objeto com sua chave privada, e envia para um ou mais relays via WebSocket.

Essa estrutura exige que você pense em termos de **eventos imutáveis**. Não há "editar um post" da maneira convencional — você publica um novo evento que substitui o anterior, ou marca o anterior como "apagado" (com um novo evento do tipo delete).

A Infraestrutura é Voluntária

Diferente de um sistema com back-end centralizado, onde seu servidor define o que está disponível, no Nostr os relays são voluntários e independentes. Isso significa que:

- Nem todos os relays guardarão seus eventos.
- Você pode publicar para múltiplos relays ao mesmo tempo.
- Os clientes decidem de quais relays querem ler, definidos pelo usuário ou padrão do cliente..

É importante projetar seu cliente para funcionar com redundância: publique em mais de um relay, leia de vários, e trate casos de ausência de resposta e excesso de dados. Isso ajuda a garantir a resiliência e a disponibilidade do seu aplicativo Nostr, mesmo que algumas relays estejam indisponíveis ou com problemas, ou que você receba um volume de dados maior do que o esperado.

Não há "API oficial"

Não há um servidor central com um endpoint `/api/posts` ou `/login`. Tudo é feito por meio de **conexões WebSocket** diretas com os relays, usando uma estrutura de mensagens muito simples:

- `REQ`: solicita eventos.
- `EVENT`: envia um evento.
- `NOTICE`, `EOSE`, `CLOSED`: mensagens de controle da conexão.

Você deve estar confortável com WebSockets e programação assíncrona para interagir eficientemente com o protocolo.

O ecossistema está em constante evolução

O Nostr é jovem e altamente dinâmico. Novos NIPs (propostas de implementação) surgem com frequência, e alguns se tornam rapidamente adotados por toda a comunidade. É importante acompanhar os repositórios e canais de discussão, como:

- https://github.com/nostr-protocol/nips
- Grupos no Nostr, Telegram, e outros canais

Estar por dentro do que está em desenvolvimento ajuda a tomar decisões técnicas mais sustentáveis.

Escolhendo seu primeiro projeto com Nostr

Se você está começando agora no desenvolvimento com o protocolo Nostr, a melhor forma de aprender é construindo algo real. Mas, para evitar frustrações iniciais, é importante escolher um projeto com escopo bem definido, que te permita entender os fundamentos enquanto se diverte no processo.

Abaixo estão algumas sugestões de projetos ideais para iniciantes — cada um abordando diferentes aspectos do protocolo:

Cliente Leitor de Eventos Públicos

Objetivo: Criar uma aplicação web ou de linha de comando que se conecte a um ou mais relays e exiba eventos do tipo `kind: 1` (posts simples).

Você vai aprender a:

- Estabelecer conexões WebSocket com um ou mais relays.
- Enviar mensagens `REQ` para buscar eventos.
- Ler e interpretar o conteúdo dos eventos.
- Renderizar conteúdo filtrado (por autor, palavras-chave ou data).

Tecnologias recomendadas: JavaScript/TypeScript com HTML básico. Frameworks como React ou Vue (opcional).

Gerador de Identidade Nostr

Objetivo: Criar uma ferramenta simples que gere pares de chave pública/privada e mostre o perfil correspondente.

Você vai aprender a:

- Trabalhar com bibliotecas como `nostr-tools`.
- Entender o formato das chaves no Nostr.
- Armazenar chaves de forma segura (pelo menos localmente).

Importante: Nunca exponha chaves privadas sem saber dos riscos.

Bot de Anúncios Automáticos

Objetivo: Criar um bot que publique eventos em intervalos regulares com mensagens automáticas (ex: citações, status, anúncios).

Você vai aprender a:

- Assinar e publicar eventos.
- Programar tarefas recorrentes.

- Lidar com erros de publicação e falhas de conexão.

Pode ser feito com: Node.js, Python ou Go.

Relay Privado Local

Objetivo: Rodar seu próprio relay em sua máquina para testes e aprendizado.

Você vai aprender a:

- Instalar e configurar um relay (como `nostr-rs-relay`).
- Ver como os eventos são armazenados e servidos.
- Entender como funciona a persistência, moderação e filtragem.

Esse tipo de projeto te dá visão de "nos bastidores" e é útil mesmo que seu foco principal seja no front-end.

Perfil Público com NIP-05

Objetivo: Criar um site estático que ofereça um endpoint `.well-known/nostr.json` para mapear sua chave pública para um nome legível como `@seunome@seudominio.com`.

Você vai aprender a:

- Usar o NIP-05 para validação de identidade.
- Configurar um domínio e servidor web básico ou hospedar o site no github pages.
- Entender a interoperabilidade entre identidades e clientes.

Dica Final

Comece pequeno e pense iterativamente. O protocolo Nostr valoriza a simplicidade — você não precisa de um sistema complexo com autenticação, back-end e banco de dados para lançar seu primeiro experimento. Com apenas uma chave, um JSON assinado e uma conexão WebSocket, você já pode fazer parte da rede global.

Parte 1: Fundamentos do Nostr

3. Arquitetura e Componentes do Nostr

Relays, chaves públicas e privadas

Para compreender como o Nostr funciona de verdade, é essencial entender seus componentes fundamentais. A arquitetura do protocolo é deliberadamente simples, mas poderosa. Ela é composta por apenas três elementos essenciais:

1. **Usuários**, que criam e assinam eventos com chaves criptográficas.
2. **Relays**, que armazenam e retransmitem os eventos.
3. **Eventos**, que são a unidade de informação no Nostr.

Vamos focar, neste capítulo, em dois pilares centrais: os **relays** e o **sistema de chaves públicas e privadas**.

Relays

Os relays são servidores simples que se comunicam via WebSocket. Eles não processam lógica complexa — seu único trabalho é receber eventos assinados dos usuários, armazená-los (opcionalmente) e retransmiti-los para outros usuários conectados que fizerem uma requisição compatível.

Cada relay é independente: ele escolhe o que aceita, o que armazena, por quanto tempo, e para quem envia. Não há uma cadeia de confiança, nem um mecanismo de consenso entre os relays. Um usuário pode se conectar a quantos relays quiser, e pode publicar em todos eles simultaneamente.

Essa arquitetura distribui o poder: se um relay aplicar censura, basta publicar em outro. Se um relay sair do ar, os eventos ainda estão disponíveis em outros. Isso garante resiliência e resistência à censura.

Chaves públicas e privadas

Toda identidade no Nostr é representada por um par de chaves:

- **Chave pública** (`npub⋯`): usada para identificar o usuário. É como o seu “nome” dentro da rede.
- **Chave privada** (`nsec⋯`): usada para assinar eventos e provar autoria. Deve ser mantida em segredo absoluto.

Essa arquitetura criptográfica tem raízes nos mesmos princípios utilizados no Bitcoin: a chave privada garante soberania, autenticidade e controle direto sobre a identidade.

Não há servidores de autenticação, senhas ou cookies. Ao assinar um evento com sua chave privada, você está afirmando: “isso foi escrito por mim”. Os relays, por sua vez, podem validar a assinatura com sua chave pública — e saber que é válida sem precisar confiar em ninguém.

Essa abordagem tem diversas implicações:

- **Autonomia total**: ninguém pode te banir de "existir" no Nostr.

- **Portabilidade de identidade**: você pode usar sua chave em qualquer cliente.
- **Risco de perda total**: se perder sua chave privada, não há recuperação.

Por isso, desde o início, é importante adotar práticas de segurança robustas: guardar backup de suas chaves privadas com criptografia, usar carteiras Nostr seguras como *nos2x*, *Nostrame*, *Amber* e *Alby*, ou até soluções como hardware wallets em aplicativos mais sensíveis.

Essa arquitetura minimalista — usuários com chaves e relays simples — torna o Nostr extremamente flexível. A partir disso, é possível construir redes sociais, chats, fóruns, sistemas de reputação, mensageria privada e muito mais, tudo em cima dos mesmos princípios.

Na próxima seção, veremos como os eventos são estruturados e como tudo isso ganha vida em um fluxo contínuo de mensagens assíncronas.

Estrutura de eventos (NIP-01)

O que são eventos?

No Nostr, toda ação do usuário — postar uma nota, enviar uma mensagem, atualizar seu perfil, reagir a algo — é representada por um **evento**. Um evento é uma estrutura de dados assinada com a chave privada do usuário, e é a menor unidade de comunicação no protocolo.

Eventos são imutáveis: uma vez publicados, não podem ser modificados. Se algo precisar ser alterado, um novo evento deve ser emitido, referenciando o anterior se necessário. Isso garante auditabilidade e evita manipulações invisíveis.

Estrutura básica de um evento

Um evento no Nostr é um objeto JSON com os seguintes campos obrigatórios:

```
{
  "id": "<hash sha256 do evento sem o ID e a assinatura>",
  "pubkey": "<chave publica do autor>",
  "created_at": <unix timestamp>,
  "kind": <número inteiro do tipo do evento>,
  "tags": [["<tipo>", "<valor>", ...]],
  "content": "<texto ou dados do evento>",
  "sig": "<assinatura da estrutura acima>"
}
```

Vamos entender o que cada campo significa:

- `id`: hash SHA-256 gerado a partir do conteúdo do evento (sem a assinatura).
- `pubkey`: chave pública do autor do evento.
- `created_at`: unix timestamp indicando data e hora que o evento foi criado.
- `kind`: tipo do evento. Define seu propósito (ex: para nota pública, para metadados de perfil, mensagem privada, etc.).
- `tags`: lista de tags para relacionar eventos entre si ou adicionar contexto.
- `content`: conteúdo textual do evento.
- `sig`: assinatura do evento com a chave privada do autor.

Tipos de eventos (`kind`)

O campo `kind` é usado para identificar o tipo de evento. Alguns exemplos comuns:

- `0`: metadados do usuário (nome, imagem, descrição, etc.)
- `1`: nota de texto (posts ou mensagens públicas)
- `3`: lista de contatos

- `4`: mensagem privada (conteúdo criptografado NIP-4)
- `5`: solicitação de exclusão de evento
- `6`: repostar nota (como um “compartilhar”)
- `7`: reação (como um “like”)

Além disso, o ecossistema define novos tipos de evento por meio das NIPs (propostas de extensão). Isso permite que o protocolo evolua sem precisar de atualizações centralizadas.

Assinatura e verificação

Antes de um evento ser transmitido, ele é assinado com a chave privada do autor. Os relays e clientes verificam a validade da assinatura usando a chave pública (`pubkey`). Isso garante que:

- O autor do evento é quem diz ser.
- O conteúdo não foi alterado depois da assinatura.

Esse processo de assinatura e verificação é fundamental para a confiança no Nostr — e dispensa qualquer servidor de autenticação.

O papel das tags

O campo `tags` é uma lista de listas. Cada tag adiciona metadados ao evento. Por exemplo:

```
"tags": [
  ["e", "id de outro evento", "relay_url"],
  ["p", "chave pública de alguem"],
  ["d", "identificador customizado"],
  ["r", "https://example.com/products?category=books&id=123"],
  ["t", "nostr"]
]
```

Tags comuns:

- `["e", "⋯"]`: referência a outro evento (por exemplo, para respostas ou threads).
- `["p", "⋯"]`: menção a outro usuário (por exemplo, em mensagens diretas).
- `["d", "⋯"]`: identificador customizado de referência.
- `["r", "⋯"]`: referência a um recurso externo (como uma URL).
- `["t", "⋯"]`: tópicos ou hashtags.

As tags permitem que eventos sejam relacionados e organizados de forma eficiente, sem depender de esquemas de dados rígidos.

A estrutura de eventos no Nostr é poderosa em sua simplicidade. Com poucos campos e uma assinatura criptográfica, ela permite a construção de funcionalidades complexas e seguras — sempre mantendo a transparência, a portabilidade e a soberania do usuário como princípios centrais.

Publicando seu primeiro evento

Uma das melhores formas de aprender como o Nostr funciona é publicar um evento manualmente. Nesta seção, vamos criar uma par de chaves e um evento simples (uma nota de texto) e enviá-lo para um relay.

Dependências necessárias

Antes de começar, você precisa ter o **Node.js** e o **npm** instalados no seu sistema. Eles são essenciais para executar os scripts JavaScript apresentados aqui.

1. Instalando o Node.js

Você pode baixar e instalar o Node.js a partir do site oficial:

https://nodejs.org

A instalação do Node.js já inclui o **npm** (Node Package Manager), que é usado para instalar bibliotecas JavaScript.

Para verificar se tudo está instalado corretamente, execute os seguintes comandos no terminal:

```
node -v
npm -v
```

Eles devem exibir as versões instaladas do Node.js e do npm, respectivamente.

2. Instalando as bibliotecas necessárias

Com o ambiente configurado, crie uma nova pasta para o projeto e inicialize o `package.json`:

```
mkdir meu-primeiro-evento-nostr
cd meu-primeiro-evento-nostr
npm init -y
npm pkg set type="module"
```

Agora instale as bibliotecas usadas nos scripts:

```
npm install nostr-tools
```

3. O que estamos instalando?

- `nostr-tools`: Esta biblioteca nos ajudará a gerar chaves, assinar eventos, se conectar a relays via WebSocket e outras coisas mais.

Com essa biblioteca instalada, você estará pronto para publicar seu primeiro evento em um relay Nostr.

No trecho de código abaixo incluimos a funções da biblioteca `nostr-tools` para trabalhar com o protocolo Nostr de forma segura e funcional. A função `generateSecretKey` cria uma chave privada, enquanto `getPublicKey` deriva a chave pública correspondente. Já `finalizeEvent` assina eventos com a chave privada, garantindo autenticidade.

```
import { generateSecretKey, getPublicKey, finalizeEvent } from 'nostr-tools/pure'
import { Relay } from 'nostr-tools/relay'
import { bytesToHex, hexToBytes } from '@noble/hashes/utils'
```

A classe `Relay` permite conectar a servidores Nostr (relays) para enviar e receber eventos. A função `bytesToHex` e `hexToBytes`, da biblioteca `@noble/hashes/utils` (a biblioteca @noble vem juntamente instalada com nostr-tools), facilitam a conversão entre formatos binário e hexadecimal, comuns em operações criptográficas. Com essas ferramentas, é possível criar e publicar eventos de forma descentralizada e confiável.

Gerando suas chaves

Para começar, você precisa de um par de chaves: uma chave privada e uma chave pública.

Você pode gerar essas chaves com bibliotecas como `nostr-tools` (JavaScript), `rust-nostr` (Rust), ou até mesmo com ferramentas online — embora, para fins de segurança, seja altamente recomendável gerar suas chaves localmente.

```
import { bytesToHex } from '@noble/hashes/utils'
import { generateSecretKey, getPublicKey } from 'nostr-tools/pure'

// gerar as chaves aleatoriamente
const privateKey = generateSecretKey()
const publicKey = getPublicKey(privateKey)

console.log('Chave privada:', bytesToHex(privateKey))
console.log('Chave pública:', publicKey)
```

Criando um evento

Agora que você entendeu como gerar suas chaves, podemos montar o corpo de um evento `kind: 1` e assiná-lo com a chave privada.

```
import { finalizeEvent } from 'nostr-tools/pure'
import { generateSecretKey } from 'nostr-tools/pure'

const privateKey = generateSecretKey()

// finalizeEvent faz a maior parte do trabalho incluindo a chave pública, id e a
assinatura
const event = finalizeEvent({
  kind: 1,
```

```
  created_at: Math.floor(Date.now() / 1000),
  tags: [],
  content: "Olá, mundo Nostr!",
}, privateKey)

console.log('Evento:', event)
```

Esse objeto `event` agora está pronto para ser enviado. Ele contém o tipo de evento, o unix timestamp, o conteúdo, a chave pública e uma assinatura válida com a chave privada.

Conectando-se a um relay

Relays usam WebSockets. Você pode se conectar e publicar seu evento assim:

```
import { Relay } from 'nostr-tools/relay'
import { generateSecretKey, finalizeEvent } from 'nostr-tools/pure'

// conecta ao relay
const relay = new Relay('wss://relay.damus.io')
await relay.connect()

const privateKey = generateSecretKey()

const event = finalizeEvent({
  kind: 1,
  created_at: Math.floor(Date.now() / 1000),
  tags: [],
  content: "Olá, mundo Nostr!",
}, privateKey)

// publica o evento
await relay.publish(event)

console.log('Evento publicado!', event)

// fecha a conexão para terminar o script
relay.close()
```

Você verá seu evento sendo transmitido com sucesso se tudo estiver correto. Pode também receber respostas de confirmação, dependendo do relay.

Código completo

Utilize o código a seguir para criar o arquivo `publishingEvent.js`

```
import { generateSecretKey, getPublicKey, finalizeEvent } from 'nostr-tools/pure'
import { Relay } from 'nostr-tools/relay'
import { bytesToHex, hexToBytes } from '@noble/hashes/utils'
```

```
const privateKey = generateSecretKey()
const publicKey = getPublicKey(privateKey)

console.log('Chave privada:', bytesToHex(privateKey))
console.log('Chave pública:', publicKey)

const event = finalizeEvent({
  kind: 1,
  created_at: Math.floor(Date.now() / 1000),
  tags: [],
  content: "Olá, mundo Nostr!",
}, privateKey)

const relayUrl = 'wss://relay.damus.io'
const relay = new Relay(relayUrl)

await relay.connect()
await relay.publish(event)

console.log('Evento publicado!', event)
relay.close()
```

Rode o arquivo usando Node.js:

```
node publishingEvent.js
```

Veja a saída será algo similar:

```
Chave privada: daff342084ab433b55590402361dae273fc629a38bdd2eb2ff4a3389abd8cb65
Chave pública: 189173df59bcf9a34902ebdf75a1f42205c27d37d5b6f295689a3a3f9794a55e

Evento publicado! {
  kind: 1,
  created_at: 1747180986,
  tags: [],
  content: 'Olá, mundo Nostr!',
  pubkey: 'fcd06203f65d0fbd9d2f24dce3a66325d6c9e9165b9f7716790eaf3425826c8e',
  id: 'f97dad9d5382c7c80cad911ec9ae420d3a5235a0bbb03ffd7bc44bb3816ae3f9',
  sig:
'1185e75350167ce7d4107a4f6bbf7322ccb799e84b814e3ef39f0b2c1a5c0b1946603100fb0c3e0900deb
031a101711088ce8e7b77441b2ebd8fc2fd953adb6e',
  [Symbol(verified)]: true
}
```

Observando o resultado

Você pode verificar se o evento foi publicado com sucesso usando uma ferramenta ou cliente conectado ao mesmo relay, como:

- https://njump.me/

Basta colar seu `id` ou `pubkey` a frente do link do site njump para ver se seu evento apareceu.

Ou você também pode passar sua `id` para pegar o evento publicado usando o código abaixo:

```
// Utilizando a classe SimplePool podemos nos conectar em mais de um relay
import { SimplePool } from 'nostr-tools'

const relays = ['wss://relay.damus.io']

const pool = new SimplePool()

const eventId = 'f97dad9d5382c7c80cad911ec9ae420d3a5235a0bbb03ffd7bc44bb3816ae3f9'

const event = await pool.get(relays, { ids: [eventId] })

if (event) {
  console.log('Evento encontrado:', event)
} else {
  console.log('Evento não encontrado nos relays fornecidos.')
}

// encerra a conexão com o relay manualmente
process.exit(0)
```

Com isso, você publicou seu primeiro evento no Nostr de forma programática. Essa é a base de todo o funcionamento do protocolo — e a partir disso, você pode construir clientes, bots, aplicativos móveis ou o que imaginar.