



# Vite

[Conhecendo o Vite](#)

[Por que Vite?](#)

[Início lento do servidor](#)

[Atualizações lentas](#)

[Por que agrupar para produção](#)

[Suporte aos navegadores](#)

[Vite Online](#)

[Criando um projeto com Vite](#)

[index.html](#) e a raiz do projeto

[Command Line Interface \(CLI\)](#)

[Variáveis de ambiente](#)

[Arquivos .env](#)

**Prioridade de carregamento dos arquivos .env**

[Substituição de Variáveis de Ambiente em HTML](#)

[Referências](#)

---

## Conhecendo o Vite

- **Vite** (palavra francesa para "rápido", pronunciada como /vit/, como "veet");
- Ferramenta de construção de projetos de código que se destina a oferecer uma experiência de desenvolvimento mais rápida;
- Ela consiste em duas partes principais:
  - Servidor de Desenvolvimento;
  - Comando de Construção.
  - Usa como empacotador o Rollup;
  - Sua documentação diz que:

***"Apesar de `esbuild` ser mais rápido, a adoção da API de plugin flexível e da infraestrutura do Rollup pela Vite contribuiu fortemente para seu sucesso no ecossistema. Por enquanto, acreditamos que o Rollup oferece uma melhor compensação entre desempenho e flexibilidade"***

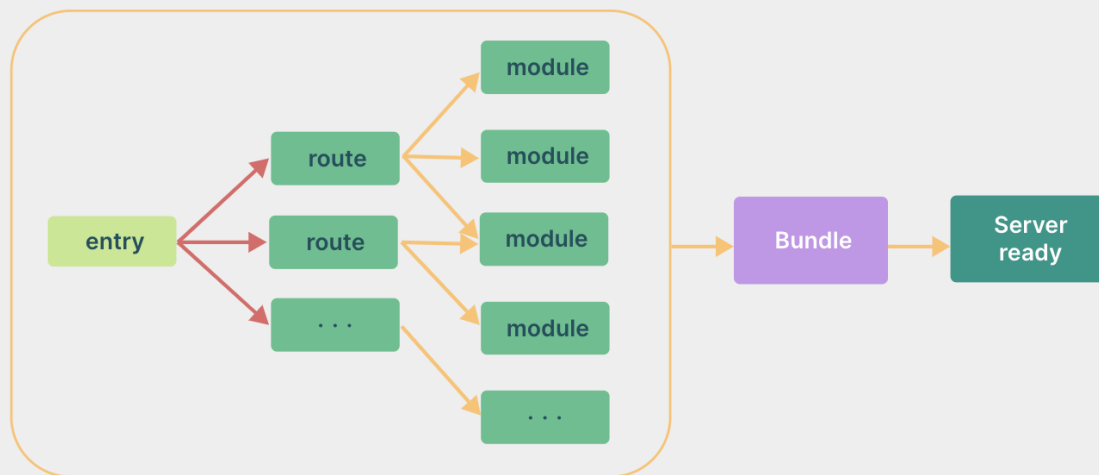
## Por que Vite?

- Antes dos módulos de ECMAScript (ESM) estarem disponíveis nos navegadores, os programadores não tinham nenhum mecanismo nativo para produção de código JavaScript que pudesse separar esses códigos em módulos;
- É por isto que estamos todos familiarizados com o conceito de "empacotamento" ou "bundle";
  - Utilizando ferramentas que rastreiam, processam e concatenam os nossos módulos de origem em arquivos que possam ser executados no navegador.
- À medida que vamos construindo aplicações mais complexas, a quantidade de JavaScript com que estamos lidando também está crescendo drasticamente;
- Problemas de desempenho para aplicações JavaScript;
  - Espera muito longa (às vezes de minutos) para rodar um servidor de desenvolvimento;
  - Mesmo com a substituição de módulos em tempo real (*Hot Module Replacement - HMR*), as edições de arquivo podem levar alguns segundos para serem refletidas no navegador;
  - Afeta significativamente a produtividade dos desenvolvedores.
- O Vite visa resolver problemas de lentidão no desenvolvimento;
- Ele faz isso aproveitando novas tecnologias, como os módulos ES nativos no navegador.

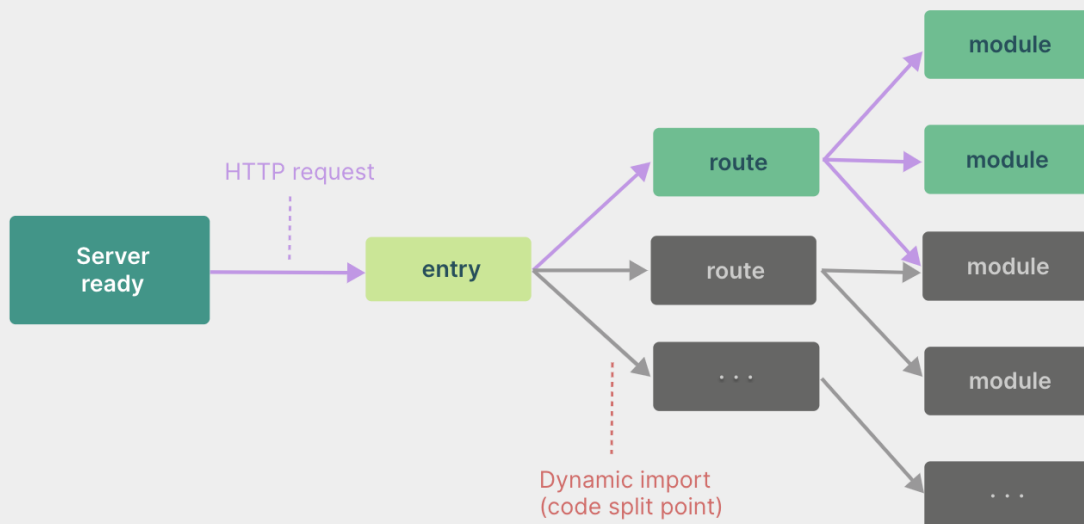
## Início lento do servidor

- Quando você começa o servidor de desenvolvimento do zero, uma configuração que usa um empacotador precisa primeiro examinar e construir todo o seu aplicativo antes de você poder começar a usá-lo;
- O Vite faz isso de uma maneira mais rápida, dividindo os diferentes pedaços do seu aplicativo em duas categorias:
  - Dependências: coisas que ele precisa de fora (como bibliotecas);
  - Código-fonte: coisas que você escreveu (seu próprio código).
- Isso ajuda a tornar o início do servidor mais rápido e ágil.
- **Source code:** O código-fonte frequentemente contém JavaScript que precisa ser transformado (por exemplo, JSX, CSS) e será editado com muita frequência;
  - Nem todo o código-fonte precisa ser carregado ao mesmo tempo (por exemplo, com divisão de código baseada em rotas);
  - O Vite serve o código-fonte através de módulos nativos ESM;
    - Permite que o navegador assuma parte do trabalho de um agrupador: o Vite só precisa transformar e servir o código-fonte sob demanda, conforme o navegador o solicita.
  - O código que está dentro de blocos de importação dinâmica condicional só será processado (ou seja, transformado e enviado ao navegador) se for realmente necessário para a tela atual que está sendo renderizada;
    - Isso significa que o Vite otimiza o processo de carregamento, enviando apenas o código necessário para a página atual, em vez de carregar todo o código do aplicativo de uma vez;
    - Isso ajuda a melhorar o desempenho geral do carregamento da aplicação.

## Bundle based dev server



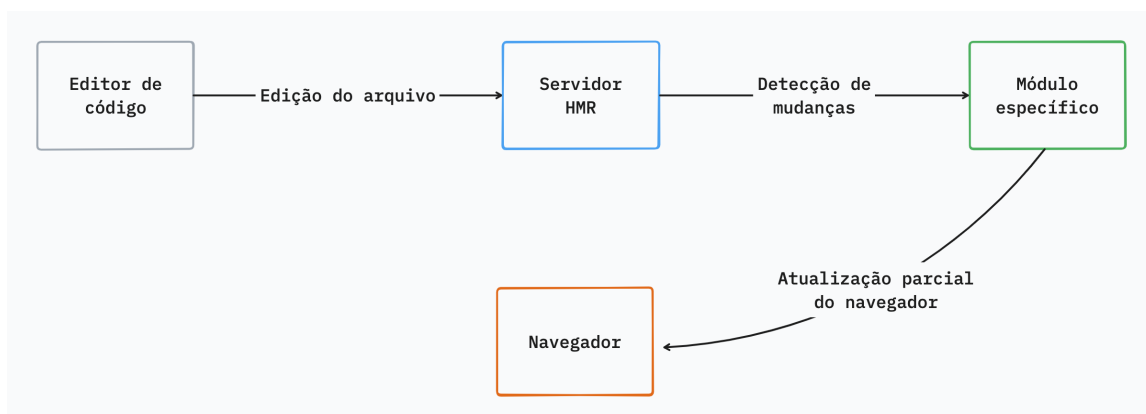
## Native ESM based dev server



## Atualizações lentas

- Em uma configuração de construção baseada em bundler, quando você edita um arquivo, reconstruir todo o pacote de aplicativos desde o início é ineficiente;

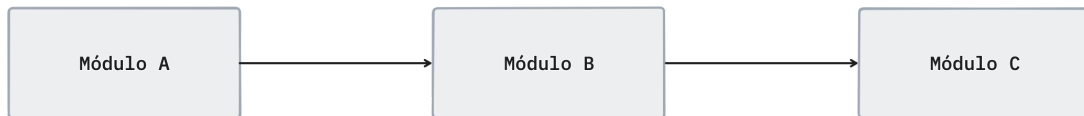
- Isso ocorre porque **quanto maior o tamanho do seu aplicativo, mais tempo levará para reconstruir o pacote inteiro.**
- Essa abordagem leva a tempos de atualização mais lentos à medida que o tamanho do aplicativo aumenta, o que não é ideal em termos de eficiência e velocidade de desenvolvimento;
- Em alguns bundlers, quando um arquivo é alterado, o servidor de desenvolvimento precisa atualizar apenas a parte relevante do seu gráfico de módulos na memória;
  - No entanto, ainda é necessário reconstruir todo o pacote de aplicativos e recarregar a página da web, o que pode ser um processo demorado e resultar na perda do estado atual da aplicação.
- Para lidar com isso, alguns bundlers oferecem suporte ao Hot Module Replacement (HMR).
  - Permite que os módulos sejam atualizados em tempo real (durante a execução) sem afetar o restante da página e sem a necessidade de recarregar a página;
    - **Edição do Arquivo:** Você edita um módulo
    - **Detecção de Mudanças:** HMR detecta a mudança
    - **Atualização Parcial:** Somente o módulo editado (e seus dependentes diretos) é atualizado



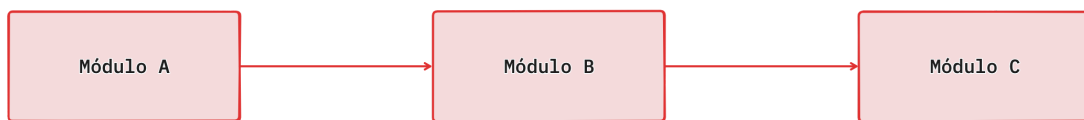
- Isso melhora significativamente a experiência do desenvolvedor;

- No entanto, conforme o tamanho da aplicação aumenta, a velocidade de atualização do HMR também pode diminuir.
- **No Vite, a substituição de módulo em tempo real (HMR) é realizada por meio de ESM nativo:**

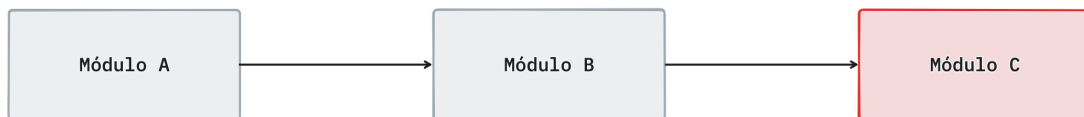
Antes da atualização do código



Depois da atualização do código sem HMR



Depois da atualização do código com HMR



- O Vite utiliza o ECMAScript Modules (ESM) nativo do JavaScript para implementar o HMR. Isso significa que ele aproveita a funcionalidade de módulos nativos do JavaScript, ao invés de usar uma solução personalizada ou uma biblioteca externa.
  - Quando você faz uma alteração no código, o Vite não recompila toda a aplicação. Em vez disso, ele apenas invalida (ou marca como desatualizado) a parte específica do código que foi alterada e suas dependências diretas até o limite mais próximo do HMR. Na maioria dos casos, isso significa apenas o módulo que foi editado, sem afetar o restante da aplicação.
    - O "limite mais próximo de HMR" se refere ao **ponto no qual o Hot Module Replacement (HMR) para de propagar a atualização;**
    - Em outras palavras, é o ponto que intercepta a atualização e decide como aplicá-la sem recarregar toda a aplicação.
- Isso resulta em atualizações rápidas do HMR, independentemente do tamanho do seu aplicativo.

- O Vite também utiliza cabeçalhos HTTP para acelerar recarregamentos completos da página (mais uma vez, permitindo que o navegador faça mais trabalho por nós):
  - Quando um navegador precisa carregar um módulo de código-fonte (JavaScript), ele faz uma solicitação ao servidor web para obtê-lo;
  - As solicitações de módulos de código-fonte são feitas de forma condicional por meio do *status code* `304 Not Modified`;
    - Isso significa que quando o navegador solicita um módulo de código-fonte, ele envia informações ao servidor sobre quais versões desse módulo ele já tem;
    - Quando o servidor recebe essa solicitação condicional e percebe que o módulo não foi alterado desde a última vez que o navegador o solicitou, ele responde com um código de status `HTTP 304 Not Modified`;
    - Esse código indica ao navegador que o módulo não foi modificado e que o navegador pode usar a versão que já possui em cache;
    - Isso reduz a quantidade de dados transferidos pela rede e economiza tempo, já que o navegador não precisa baixar o mesmo módulo novamente se ele não foi alterado.
  - As solicitações de módulos de dependência são fortemente armazenadas em cache por meio de `Cache-Control: max-age=31536000, immutable` para que não sejam solicitadas ao servidor novamente uma vez armazenadas em cache.
    - Uma vez que o navegador tenha baixado e armazenado em cache uma versão específica de um módulo de dependência, ele é fortemente armazenado em cache, o que significa que o navegador não solicitará novamente ao servidor essa mesma versão do módulo a menos que a cache expire ou seja limpo;
    - O cabeçalho Cache-Control é usado para controlar o comportamento do cache no navegador;
    - O valor `max-age=31536000` especifica que o recurso (neste caso, o módulo de dependência) pode ser armazenado em cache pelo navegador por até 31.536.000 segundos (ou seja, 1 ano) após a primeira solicitação bem-sucedida;

- O `immutable` indica que o recurso não mudará. Ou seja, o módulo de dependência é considerado "imutável", o que significa que sua versão não mudará ao longo do tempo;
  - Essa informação é útil para o navegador, pois ele pode confiar que a versão do módulo de dependência armazenada em cache não mudará, então não há necessidade de verificar com o servidor se uma versão mais recente está disponível.

## Por que agrupar para produção

### *Why bundle for production*

- Mesmo que os módulos ECMAScript nativos sejam amplamente suportados agora, enviar ESM desagrupados para produção ainda é ineficiente;
  - Viagens adicionais de rede causadas por importações aninhadas.
- Para obter o desempenho de carregamento ideal em produção, ainda é melhor agrupar seu código;
  - Carregamento lento;
  - Divisão de pedaços comuns;
  - Usando **tree-shaking**.
    - Tree-shaking é uma técnica de otimização de código utilizada em linguagens de programação como JavaScript;
    - Remover partes não utilizadas do código durante o processo de construção (build);
    - O nome "tree-shaking" vem da metáfora de um "sacudir de árvore", onde partes do código que não são referenciadas são "sacudidas" ou removidas, semelhante a sacudir uma árvore para deixar cair as folhas mortas;
    - Técnica útil em ambientes onde o código é dividido em módulos ou bibliotecas, e nem todas as partes desses módulos são necessárias para uma aplicação específica;
    - O processo de tree-shaking analisa o código e suas dependências, identificando quais partes do código não são referenciadas em nenhum lugar no aplicativo final;



- Em seguida, essas partes não utilizadas são removidas, resultando em um pacote final menor e mais eficiente em termos de desempenho;
- Isso é particularmente valioso em projetos grandes onde bibliotecas inteiras podem ser importadas, mas apenas uma fração do código é realmente usado.
- Vite é enviado com um comando de compilação pré-configurado que incorpora muitas otimizações de desempenho prontas para uso.

## Suporte aos navegadores

- **Durante o desenvolvimento** com Vite, o código é mantido o mais próximo possível do padrão ECMAScript mais recente, assumindo que um navegador moderno será utilizado, capaz de entender todas as novidades do JavaScript e CSS;
- Permite que o Vite sirva os módulos o mais próximo possível do código-fonte original;
  - Resulta em um ambiente de desenvolvimento mais eficiente e próximo do que será executado no navegador do usuário.
- **Para a compilação de produção**, o Vite tem como alvo os navegadores que suportam Módulos ES nativos, importação dinâmica nativa de ESM por padrão;
- Essas são funcionalidades mais avançadas do JavaScript que podem não ser suportadas por navegadores mais antigos;
- No entanto, navegadores antigos podem ser suportados através do plugin oficial `@vitejs/plugin-legacy`, que adiciona suporte para navegadores legados.
  - Permite que o Vite otimize o código para navegadores modernos, mas também forneça suporte para navegadores mais antigos, se necessário, durante a compilação para produção.

## Vite Online

- [StackBlitz](#);


- Ele executa a configuração de compilação baseada no Vite diretamente no navegador, tornando-a quase idêntica à configuração local, mas sem exigir a instalação de nada em sua máquina;
- Você pode navegar até `vite.new/{template}` para selecionar qual framework deseja usar.

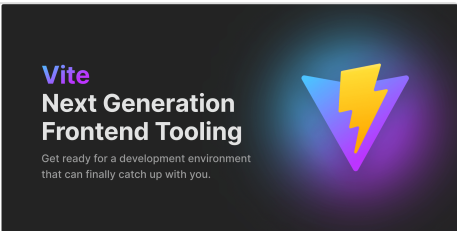
JavaScript	TypeScript
<u><a href="#">vanilla</a></u>	<u><a href="#">vanilla-ts</a></u>
<u><a href="#">vue</a></u>	<u><a href="#">vue-ts</a></u>
<u><a href="#">react</a></u>	<u><a href="#">react-ts</a></u>
<u><a href="#">preact</a></u>	<u><a href="#">preact-ts</a></u>
<u><a href="#">lit</a></u>	<u><a href="#">lit-ts</a></u>
<u><a href="#">svelte</a></u>	<u><a href="#">svelte-ts</a></u>
<u><a href="#">solid</a></u>	<u><a href="#">solid-ts</a></u>
<u><a href="#">qwik</a></u>	<u><a href="#">qwik-ts</a></u>

## Criando um projeto com Vite

Vite

Next Generation Frontend Tooling

 <https://vitejs.dev/guide/#scaffolding-your-first-vite-project>



### `index.html` e a raiz do projeto

- Em um projeto Vite, o arquivo `index.html` está centralizado;
  - Não fica escondido dentro de uma pasta `public`.
- Durante o desenvolvimento, o Vite atua como um servidor, e o `index.html` é o ponto de entrada da sua aplicação;
- O Vite trata o `index.html` como código-fonte e parte do gráfico de módulos;

- Ele resolve `<script type="module" src="...">` que referencia seu código JavaScript;
- Até mesmo `<script type="module">` embutidos e CSS referenciados via `<link href>` desfrutam de recursos específicos do Vite;
- Assim como os servidores http estáticos, o Vite possui o conceito de "diretório raiz" a partir do qual seus arquivos são servidos;
  - URLs absolutas no seu código-fonte serão resolvidas usando a raiz do projeto como base;
  - O Vite também é capaz de lidar com dependências que se encontram fora do diretório raiz, o que o torna útil mesmo em configurações de monorepositório;
- O Vite também suporta aplicativos de várias páginas com múltiplos pontos de entrada `.html`.

## Command Line Interface (CLI)

- **Usando o binário do Vite em scripts npm:** podemos configurar comandos específicos no arquivo `package.json` do seu projeto que invocam o Vite diretamente;
- **Executando o Vite diretamente com npx:** Isso significa que você pode executar comandos Vite diretamente no terminal usando `npx vite`.

```
{
  "scripts": {
    "dev": "vite", // start dev server, aliases: `vite dev`,
    "build": "vite build", // build for production
    "preview": "vite preview" // locally preview production b
  }
}
```

- Especificar opções adicionais no CLI, como `--port`;

- Para obter uma lista completa de opções da CLI, execute `npx vite --help` no projeto.

## Variáveis de ambiente

- As variáveis de ambiente guardam coisas como senhas, endereços de servidores, e outras informações importantes que o site precisa saber para funcionar;
- O Vite disponibiliza variáveis de ambiente que podem ser acessadas dentro do código do aplicativo;
- Algumas variáveis integradas estão disponíveis em todos os casos:
  - `import.meta.env.MODE` : Esta variável contém uma `string` que indica o modo em que o aplicativo está sendo executado.
    - Por exemplo, pode ser "`development`" (desenvolvimento) ou "`production`" (produção).
  - `import.meta.env.BASE_URL` : Esta variável contém uma `string` que representa o URL base de onde o aplicativo está sendo servido.
    - Isso é útil para construir URLs relativos.
  - `import.meta.env.PROD` : Esta variável é um booleano que indica se o aplicativo está sendo executado em um ambiente de produção.
    - Isso pode ser útil para realizar determinadas ações condicionalmente com base no ambiente de execução.
  - `import.meta.env.DEV` : Esta variável é um booleano que indica se o aplicativo está sendo executado em um ambiente de desenvolvimento.
    - Ela é o oposto de `import.meta.env.PROD`.
  - `import.meta.env.SSR` : Esta variável é um booleano que indica se o aplicativo está sendo executado no servidor, geralmente usado para distinguir entre a execução no servidor e no cliente em aplicativos de renderização no lado do servidor (SSR).
- Essas variáveis de ambiente são úteis para personalizar o comportamento do aplicativo com base no ambiente de execução, como ajustar

configurações, alterar comportamentos ou incluir recursos específicos para desenvolvimento ou produção.

## Arquivos `.env`

- O Vite utiliza internamente a biblioteca `dotenv` para carregar variáveis de ambiente adicionais de arquivos específicos localizados no diretório do seu ambiente de desenvolvimento;
- Esses arquivos podem conter variáveis de ambiente que você deseja disponibilizar para seu projeto durante o desenvolvimento.

```
.env                # loaded in all cases
.env.local          # loaded in all cases, ignored by git
.env.[mode]         # only loaded in specified mode
.env.[mode].local   # only loaded in specified mode, ignored
```

## Prioridade de carregamento dos arquivos `.env`

- Um arquivo de ambiente para um modo específico (por exemplo, `.env.production`) terá prioridade sobre um arquivo genérico (por exemplo, `.env`);
- Quando o Vite é executado, quaisquer variáveis de ambiente que já estejam definidas no sistema terão prioridade sobre aquelas definidas em arquivos `.env`;
  - Por exemplo, ao definir `VITE_SOME_KEY` como `123` antes de executar o comando `vite build`, o valor `123` será usado durante a compilação, mesmo que exista uma definição para `VITE_SOME_KEY` em um arquivo `.env`

```
VITE_SOME_KEY=123 vite build
```

- Os arquivos `.env` são carregados no início do Vite. É necessário reiniciar o servidor após fazer alterações nesses arquivos para que as mudanças tenham efeito.

- As variáveis de ambientes são acessíveis no código através do `import.meta.env` como strings;
- Para evitar vazamentos acidentais de variáveis de ambiente para o cliente, apenas as variáveis prefixadas com `VITE_` são expostas ao seu código processado pelo Vite:

```
VITE_SOME_KEY=123
DB_PASSWORD=foobar
```

```
console.log(import.meta.env.VITE_SOME_KEY) // "123"
console.log(import.meta.env.DB_PASSWORD) // undefined
```

- Apenas `VITE_SOME_KEY` será exposto como `import.meta.env.VITE_SOME_KEY` para o código-fonte do seu cliente;
  - Variáveis `VITE_*` não devem conter informações sensíveis por motivos de segurança.
- `DB_PASSWORD` não será exposto.



## Conversão das variáveis de ambiente

Como mostrado acima, `VITE_SOME_KEY` é um número, mas retorna uma string quando analisado. O mesmo aconteceria para variáveis de ambiente booleanas. Certifique-se de converter para o tipo desejado ao usá-lo em seu código.

```
KEY=123
NEW_KEY1=test$foo    # test
NEW_KEY2=test\ $foo  # test$foo
NEW_KEY3=test$KEY    # test123
```


## Substituição de Variáveis de Ambiente em HTML

- Quaisquer propriedades em `import.meta.env` podem ser usadas em arquivos HTML com uma sintaxe especial `%NOME_DA_ENV%`:

```
<h1>Vite is running in %MODE%</h1>
<p>Using data from %VITE_API_URL%</p>
```

Vite

Next Generation Frontend Tooling

 <https://vitejs.dev/guide/env-and-mode.html>

**Vite**  
Next Generation  
Frontend Tooling

Get ready for a development environment  
that can finally catch up with you.



## Referências

Vite

Next Generation Frontend Tooling

 <https://vitejs.dev/guide/why.html>

**Vite**  
Next Generation  
Frontend Tooling

Get ready for a development environment  
that can finally catch up with you.

