

Explorando novas fronteiras
com **Astro**

Gabriel Napolião

Software Engineer



/whoamindx



whoamindx.github.io

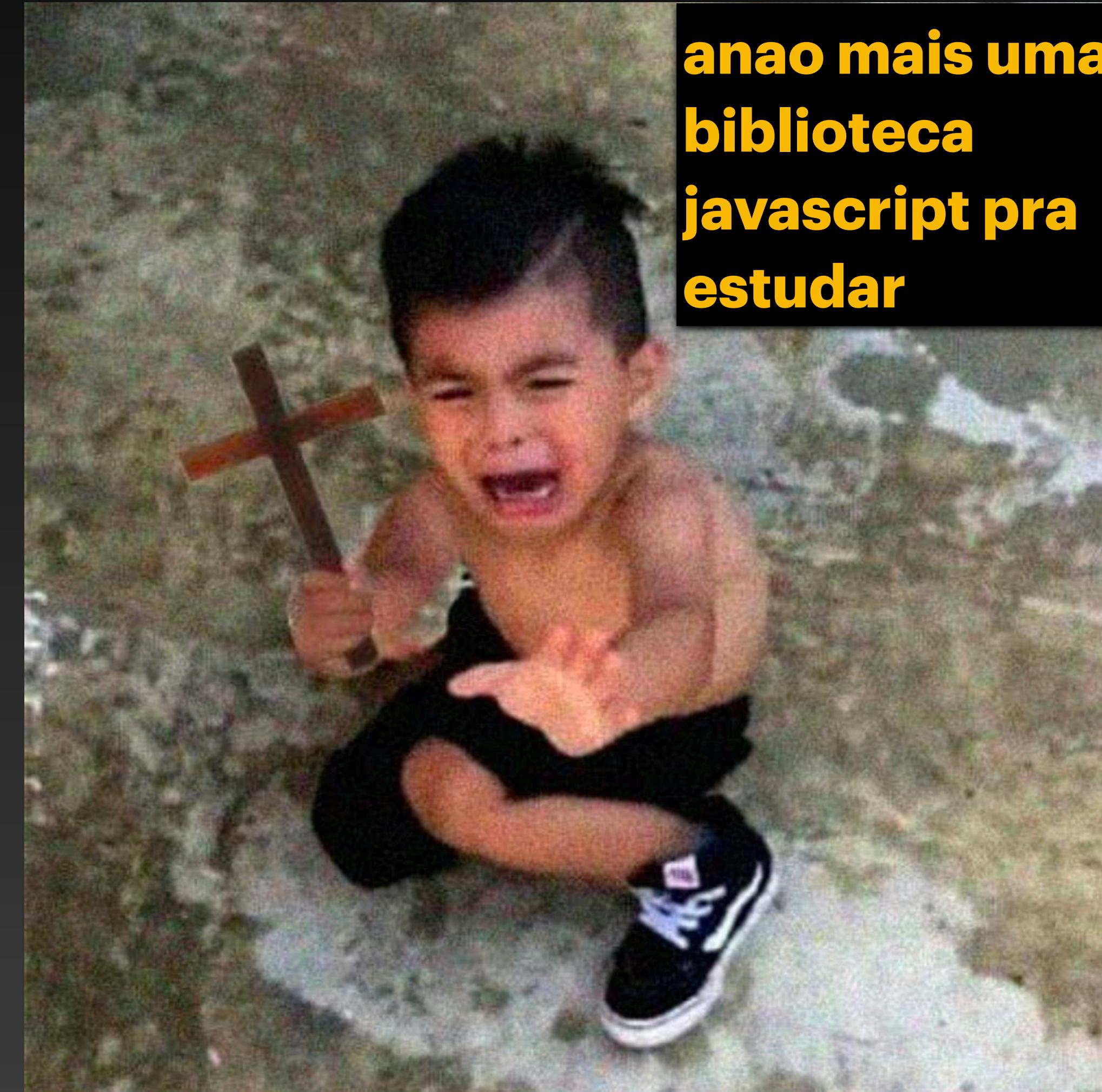


@whoamindx



*** MAIS UMA PALESTRA DE UM
FRAMEWORK JAVASCRIPT
ALEATÓRIO ACONTECE***

- VOCÊS:



**anao mais uma
biblioteca
javascript pra
estudar**



enyo

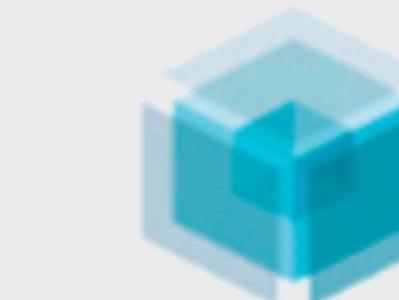
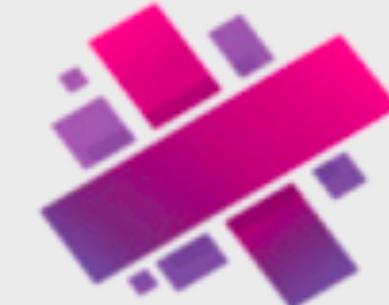


GWT



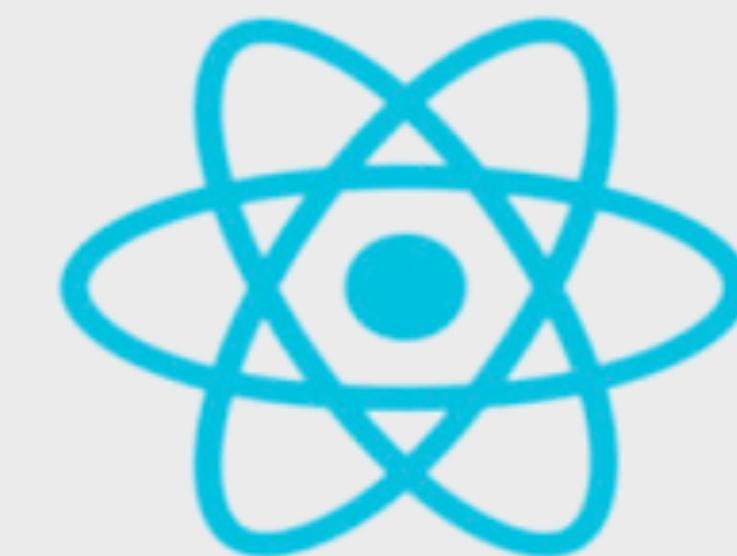
jQuery

Knockout.



ext js

Ractive.js



Sammy.js



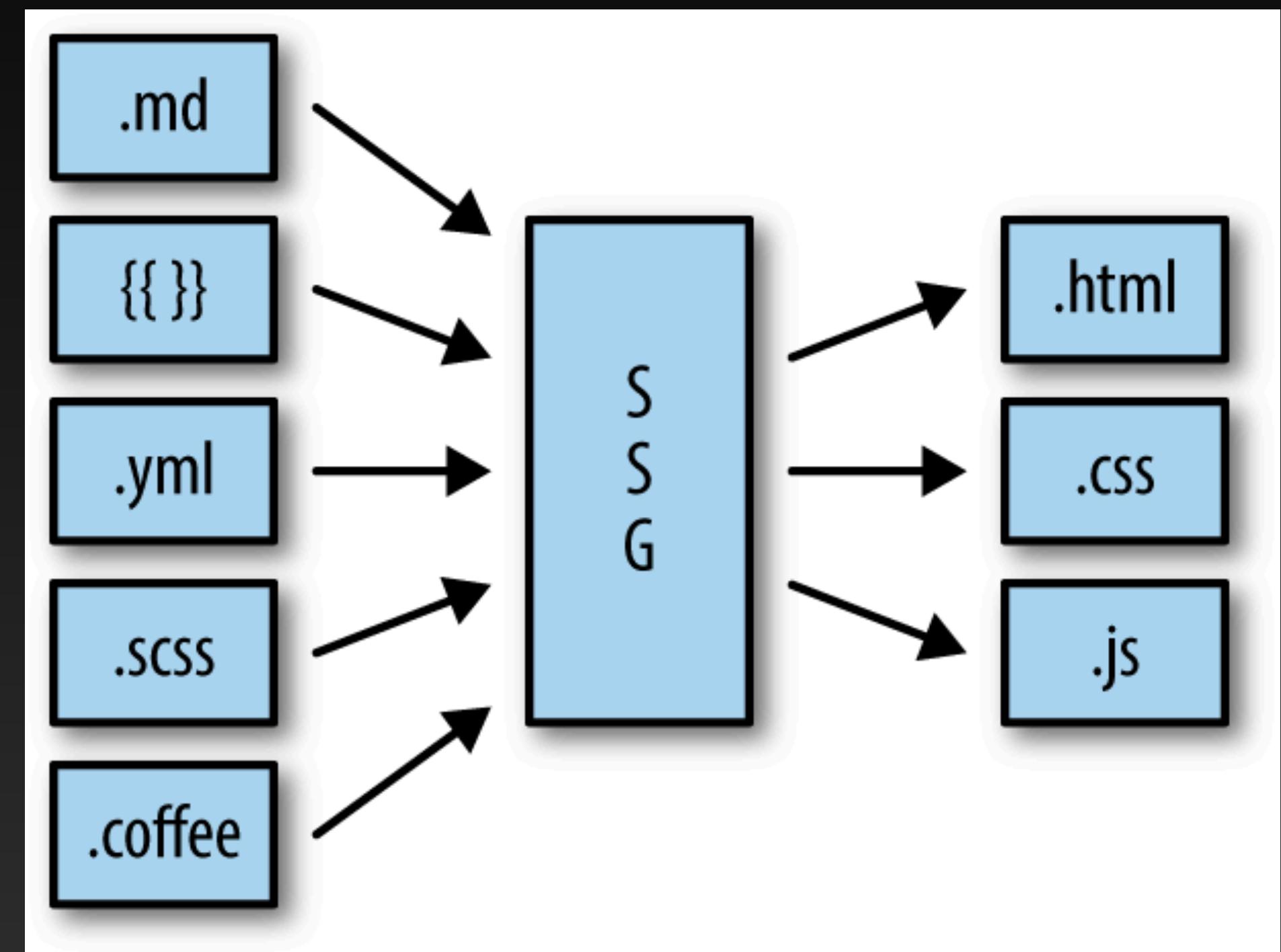
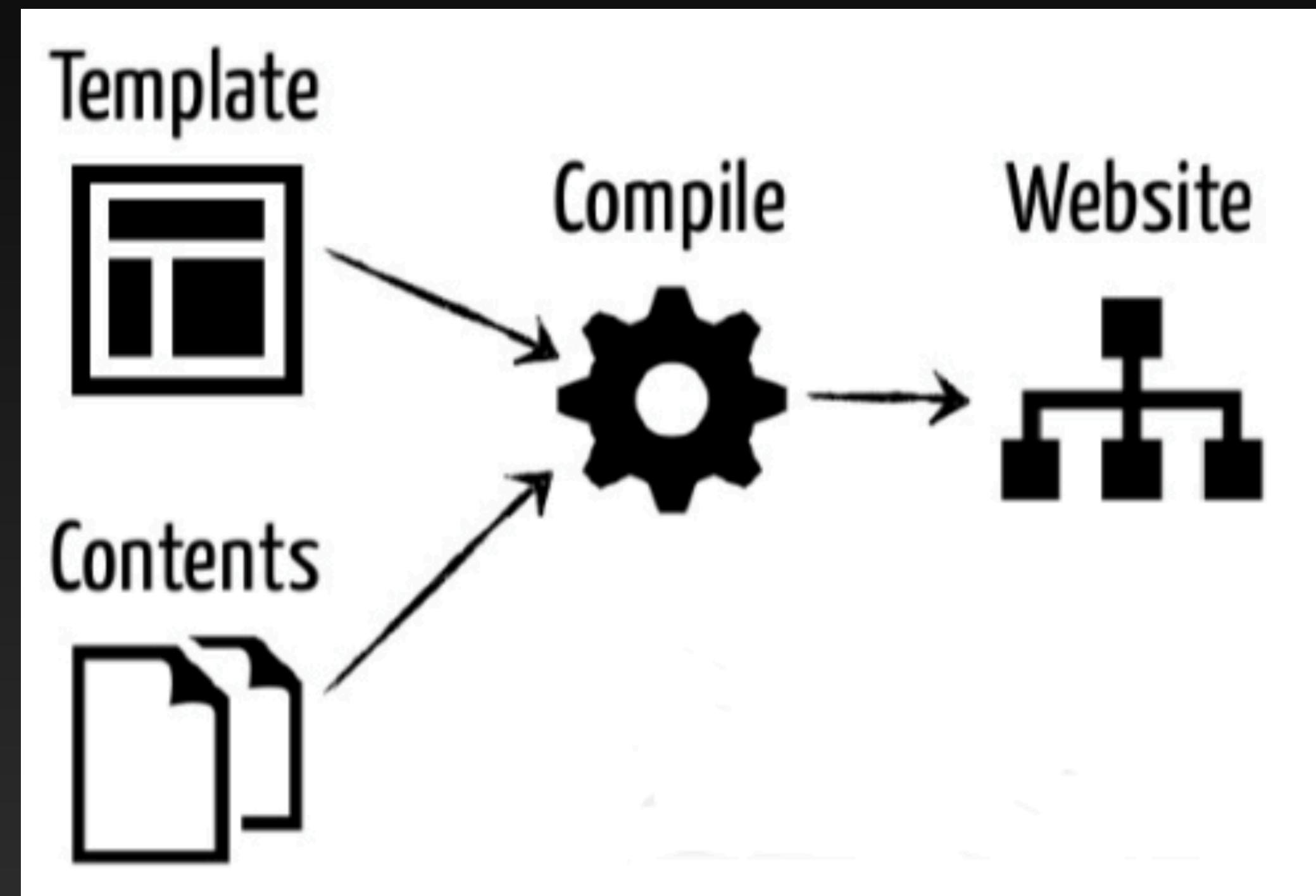
dōjō



ember



METEOR



SSG
vs
SSR

CSR

Static Site Builder

Astro is a web framework for
building **content-driven** websites like
blogs, marketing, and e-commerce.

Astro is MPA

Bundle Size

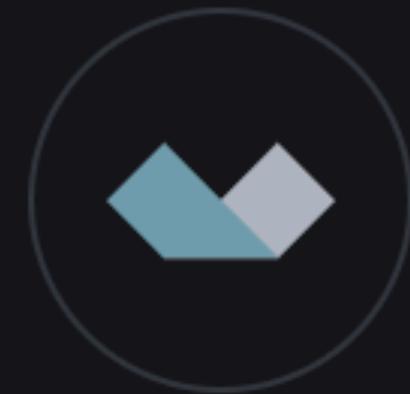
Angular ~ 500 KB

React - 150 KB (NextJS)

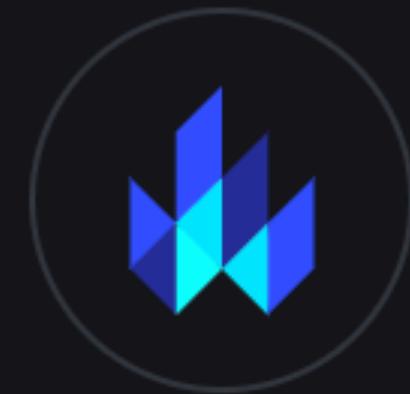
Vue - 80 KB

Official UI Framework Integrations

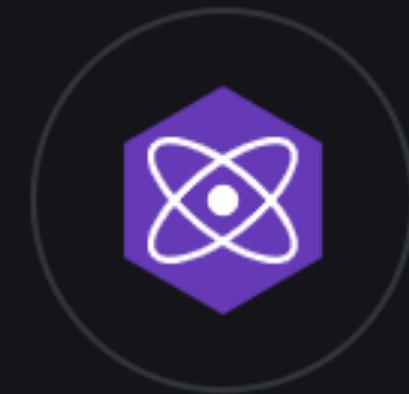
UI Frameworks



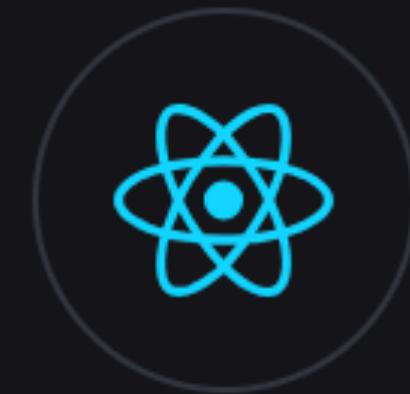
@astrojs/**alpinejs**



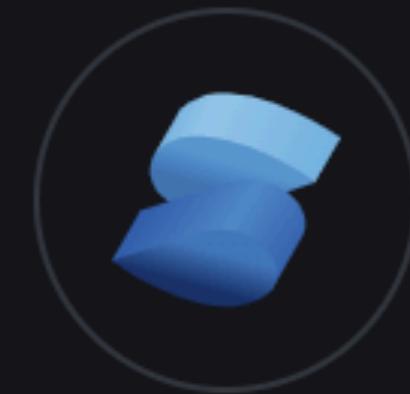
@astrojs/**lit**



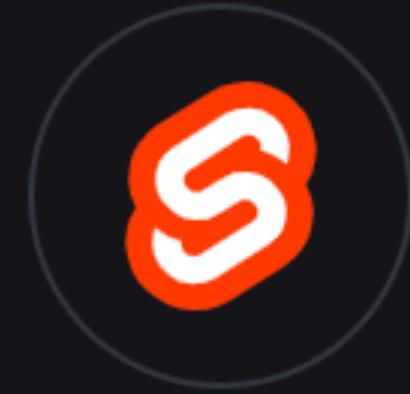
@astrojs/**preact**



@astrojs/**react**



@astrojs/**solid-js**



@astrojs/**svelte**



@astrojs/**vue**

**Astro não resolve todos os
problemas**

E ele deixa isso bem claro na sua documentação...

Dica

Se o seu projeto cai no segundo campo de “aplicações”, Astro pode não ser a escolha certa para seu projeto... **e tá tudo certo!** Veja o [Next.js](#) como uma alternativa mais focada em aplicações do que o Astro.

Astro é...

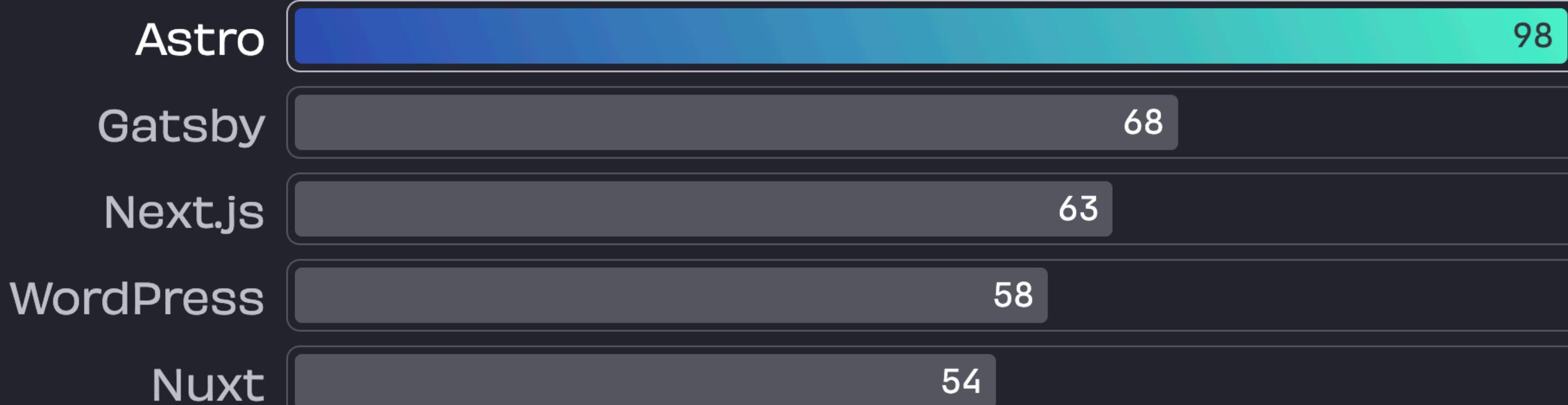
1. Focado em conteúdo: Astro foi feito para websites ricos em conteúdo.
2. Servidor em primeiro lugar: Websites são mais rápidos quando eles renderizam HTML no servidor.
3. Rápido por padrão: Deve ser impossível construir um website lento com Astro.
4. Fácil de utilizar: Você não precisa ser um especialista para construir algo com Astro.
5. Cheio de funcionalidades,_porém flexível: Mais de 100 integrações Astro para escolher.

Absurdamente rápido...

Full speed

Islands optimize your website like no other web framework can. Leverage Astro's unique page load performance to improve conversion rates, Core Web Vitals, and SEO.

Real-World Performance Comparison



Based on [HTTP Archive](#) real-world performance data (Lighthouse, P90) • [Read the full report](#)

Absurdamente rápido...

Peak Performance

Site performance has a tangible impact on the success of your business. Astro sites can be up to 40% faster and use 90% less JavaScript compared to leading static site generators.

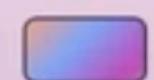
87% of websites currently fail Google's Core Web Vitals (CWV) benchmark. Astro is on a mission to change that.

“

Rebuilt my Next.js blog using Astro out of curiosity... holy shit the difference in bundle size.

Total Payload

Before: 138 kB

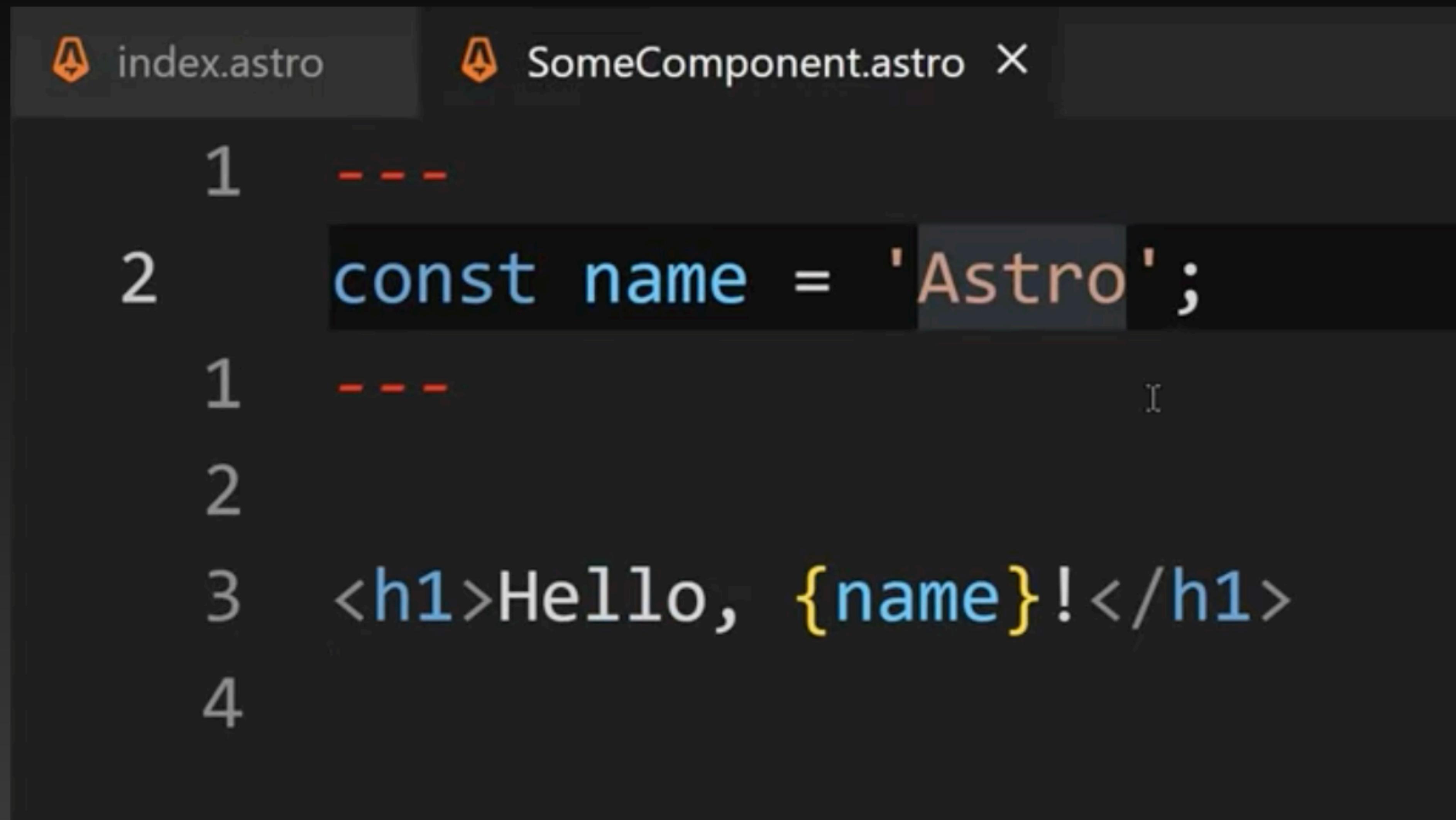


Using Astro: 7.5 kB

Real world data from [@t3dotgg](#) on Twitter

**Astro tenta compilar tudo ao máximo em
HTML e CSS tentando evitar sempre que
possível código JavaScript no lado do client**

Astro Components

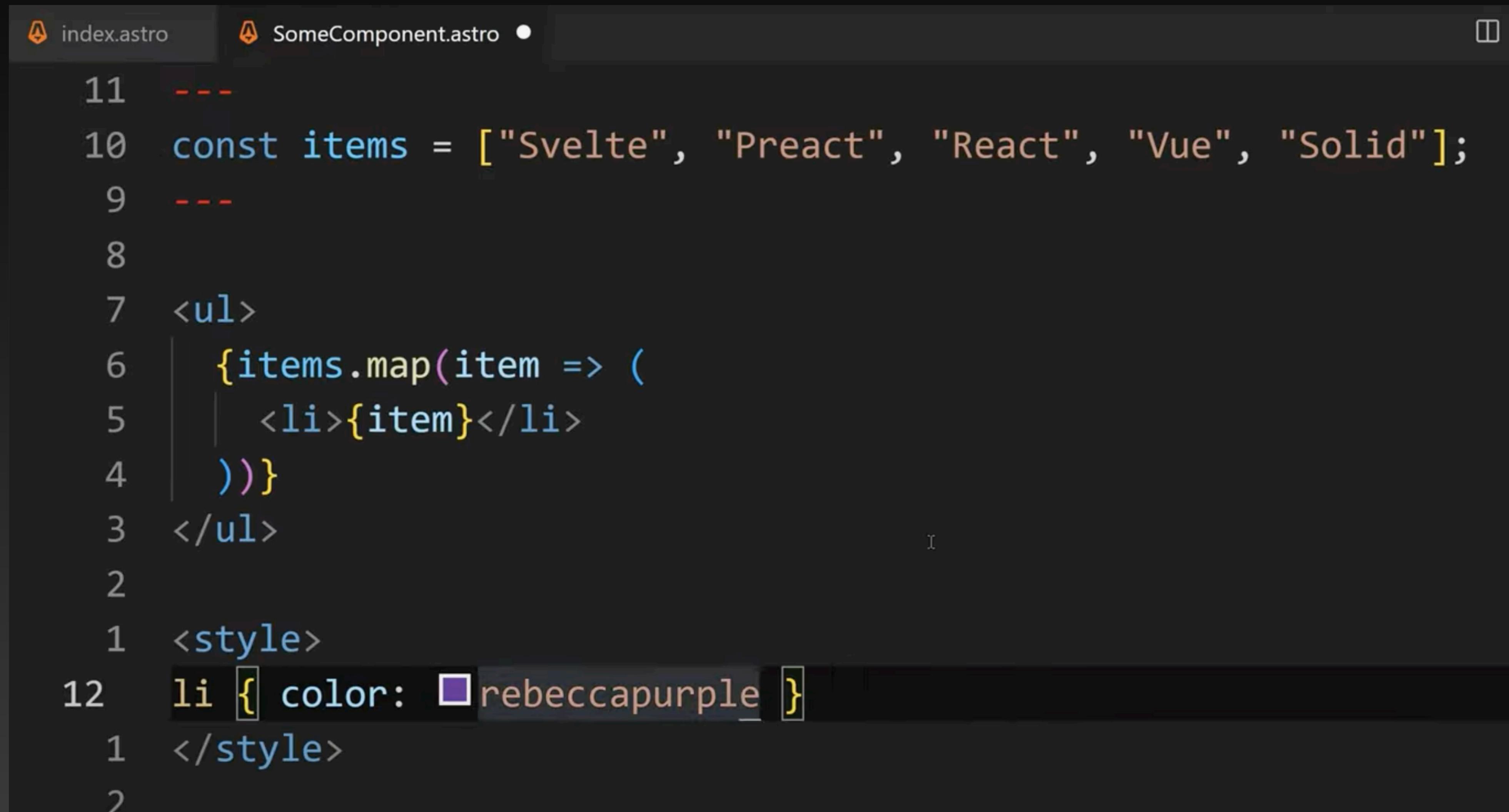


The image shows a dark-themed code editor interface with two tabs at the top: "index.astro" and "SomeComponent.astro X". The "SomeComponent.astro" tab is active, displaying the following code:

```
1 ---  
2 const name = 'Astro';  
3 ---  
4 <h1>Hello, {name}!</h1>
```

The code consists of four numbered lines. Lines 1 and 3 each contain three red dashed horizontal bars. Line 2 contains the JavaScript assignment statement `const name = 'Astro';`. Line 4 contains the Astro component template `<h1>Hello, {name}!</h1>`.

Astro Components



The screenshot shows a code editor interface with a dark theme. At the top, there are two tabs: "index.astro" (selected) and "SomeComponent.astro". The main area displays the following code:

```
11  ---
10  const items = ["Svelte", "Preact", "React", "Vue", "Solid"];
9   ---
8
7  <ul>
6  |  {items.map(item => (
5  |    |  <li>{item}</li>
4  |    ))}
3  </ul>
2
1  <style>
12  li { color: rebeccapurple }
1  </style>
2
```

The code uses Astro's syntax for components, including a map operation within a list item. A style block is also present at the bottom.

- Importação de Components
- Importação de Dados
- APIs e Banco de Dados
- Criação de Variáveis

Routing system

Astro provides both **static** and **dynamic** route generator.

.astro page components as well as **Markdown** and **MDX** Files (.md, .mdx) within the src/pages/ directory automatically become pages on your website.



Example: Static routes

src/pages/index.astro	-> mysite.com/
src/pages/about.astro	-> mysite.com/about
src/pages/about/index.astro	-> mysite.com/about
src/pages/about/me.astro	-> mysite.com/about/me
src/pages/posts/1.md	-> mysite.com/posts/1

Dynamic routes (SSG)

An Astro page file can specify dynamic route parameters to generate multiple pages.

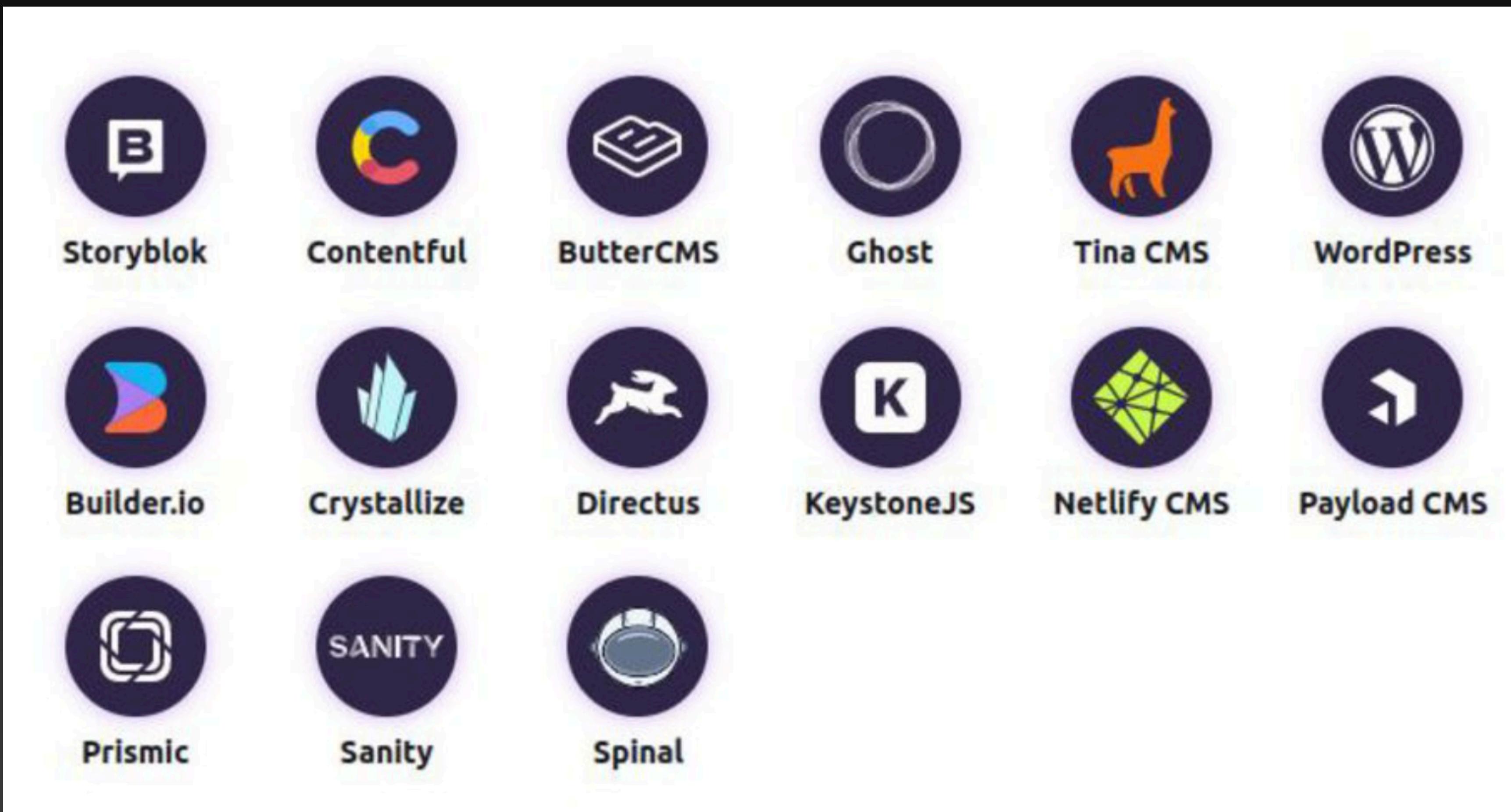
For example, `src/pages/authors/[author].astro` generates a **bio page for every author** on your blog.

This will generate three static pages:

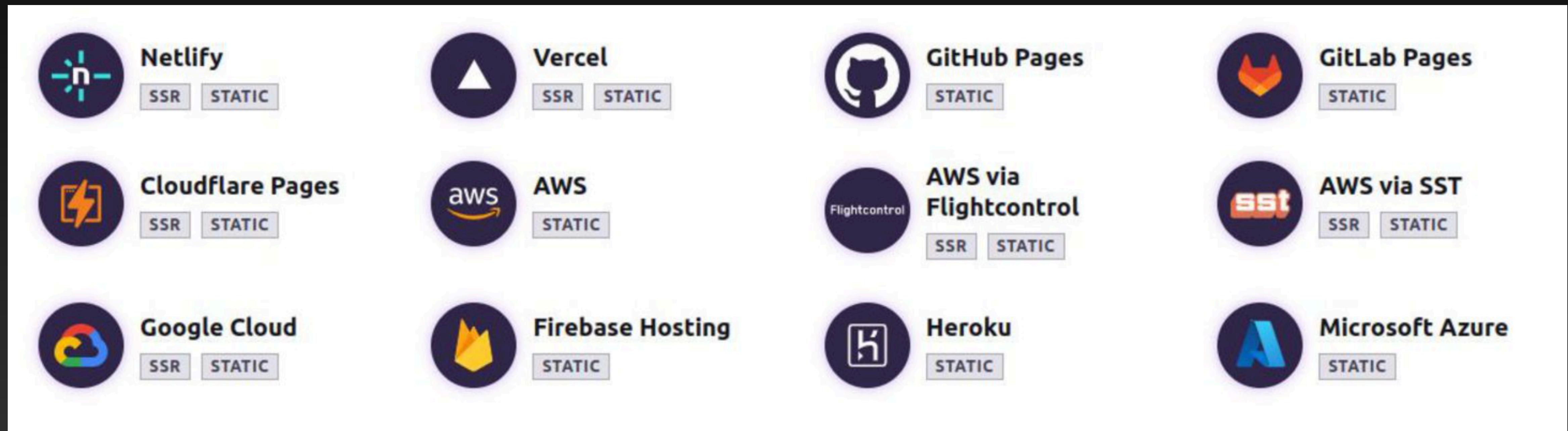
- `/authors/peter`
- `/authors/steve`
- `/authors/peggy`

```
● ● ●  
---  
export function getStaticPaths() {  
  return [  
    { params: { author: "Peter" } },  
    { params: { author: "Steve" } },  
    { params: { author: "Peggy" } },  
  ];  
}  
  
const { author } = Astro.params;  
---  
  
<div>Hello, my name is {author}!</div>
```

Conteúdo via MD, MDX ou seu CMS favorito



Deploy



Suporta TS por default

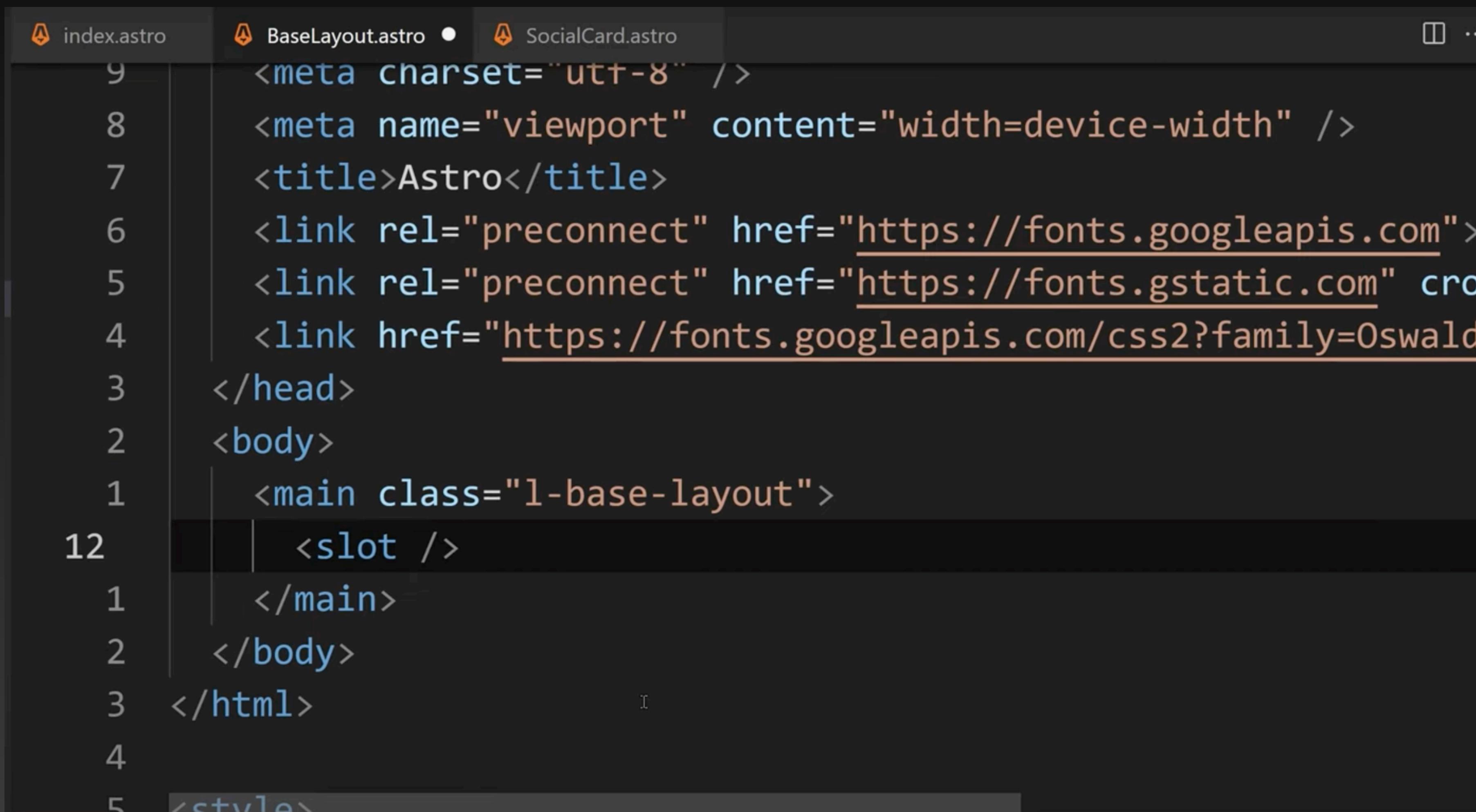
```
// SimpleComponent.astro
---
type Props = {
  title: string;
};

const { title } = Astro.props;
---

<div>
  <h3>{title}</h3>
  <section>
    <slot />
  </section>
</div>

// index.astro
<div>
  <SimpleComponent
    title="My first component">
    My first component slot render
  </SimpleComponent>
</div>
```

Layouts são Components "Especiais"



A screenshot of a code editor showing a file named `index.astro`. The code is an Astro component definition. It starts with the `<meta charset="utf-8" />` tag, followed by a `<meta name="viewport" content="width=device-width" />` tag, and a `<title>Astro</title>` tag. Then it includes several `<link rel="preconnect"` tags with hrefs pointing to Google Fonts and a CSS file. The code then moves to the `<body>` section, which contains a `<main class="l-base-layout">` element. Inside this main element, there is a `<slot />` tag at line 12. The code concludes with a closing `</main>`, `</body>`, and `</html>`. The file is part of a project with other files like `BaseLayout.astro` and `SocialCard.astro`.

```
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width" />
<title>Astro</title>
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" cro
<link href="https://fonts.googleapis.com/css2?family=Oswald
</head>
<body>
  <main class="l-base-layout">
    <slot />
  </main>
</body>
</html>
```

```
⚠ index.astro • ⚠ BaseLayout.astro ⚠ SocialCard.astro

6  ---
5 import BaseLayout from "../layouts/BaseLayout.astro";
4 import SocialCard from "../components/SocialCard.astro";
3 ---
2
1 <BaseLayout>
7   | <SocialCard />
1 </BaseLayout>
2
```

⚠️ Cuidado

Atributos HTML vão ser convertidos para strings, portanto não é possível passar funções e objetos para elementos HTML. Por exemplo, você não pode atribuir um manipulador de eventos a um elemento HTML em um componente Astro:

```
nao-faca-isso.astro
```

```
---
```

```
function tratarClique () {
  console.log("botão clicado!");
}
```

```
---
```

```
<!-- ✗ Isso não funciona! ✗ -->
<button onClick={tratarClique}>Nada vai acontecer quando você me clicar!</button>
```

Ao invés disso, use um script do lado do cliente para adicionar o manipulador de evento, como você faria no JavaScript vanilla:

```
faca-isso-em-vez-disso.astro
```

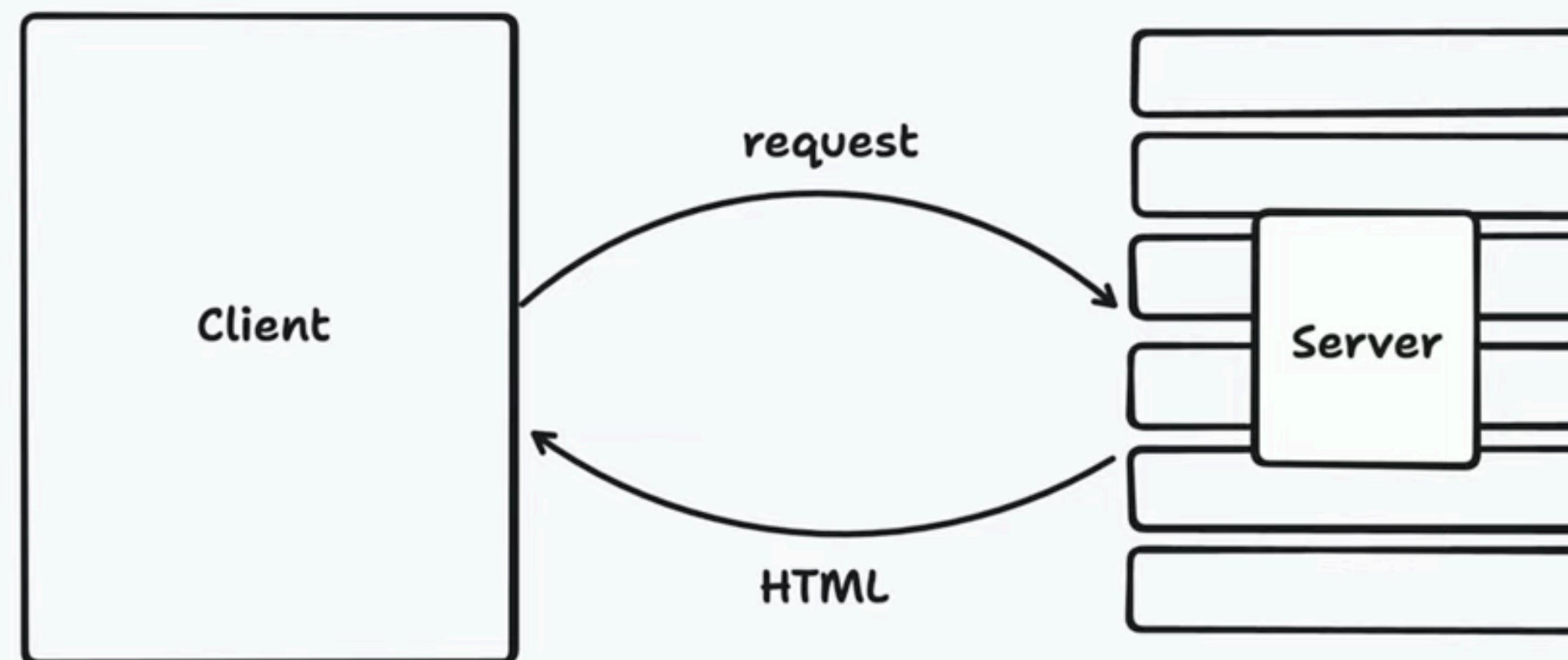
```
<button id="botao">Clique em Mim</button>
<script>
  function tratarClique () {
    console.log("botão clicado!");
  }
  document.getElementById("botao").addEventListener("click", tratarClique);
</script>
```



Cosméticos
Jequiti
Brasil no corpo e na alma

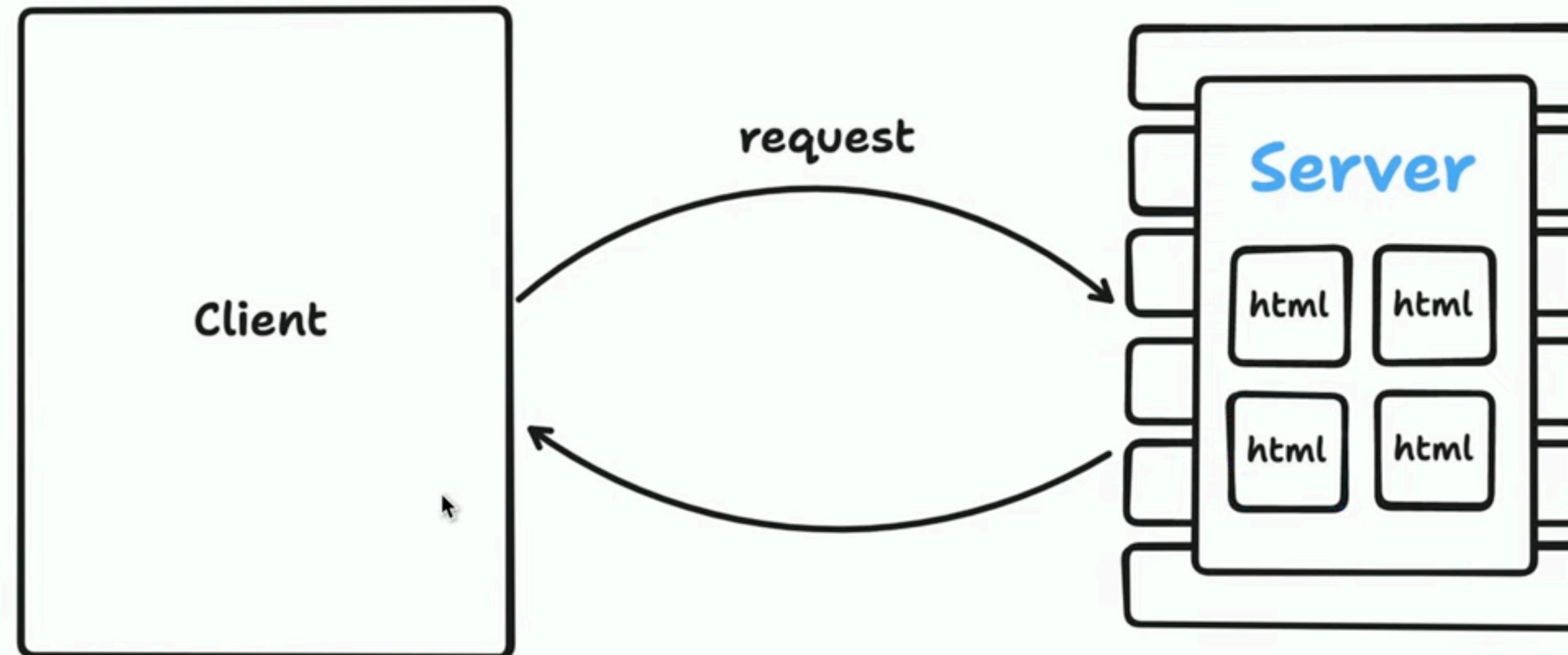


Rendering Options



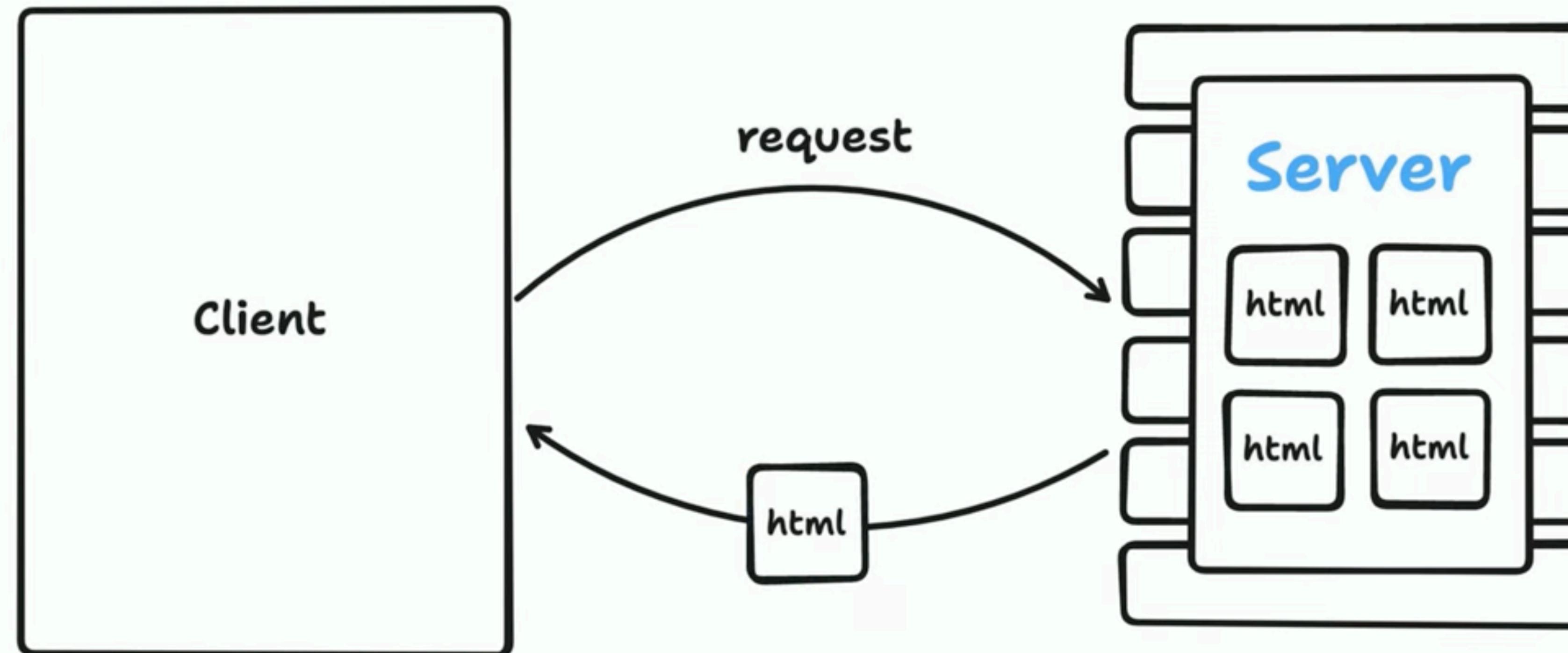
Static Site Generation

Default

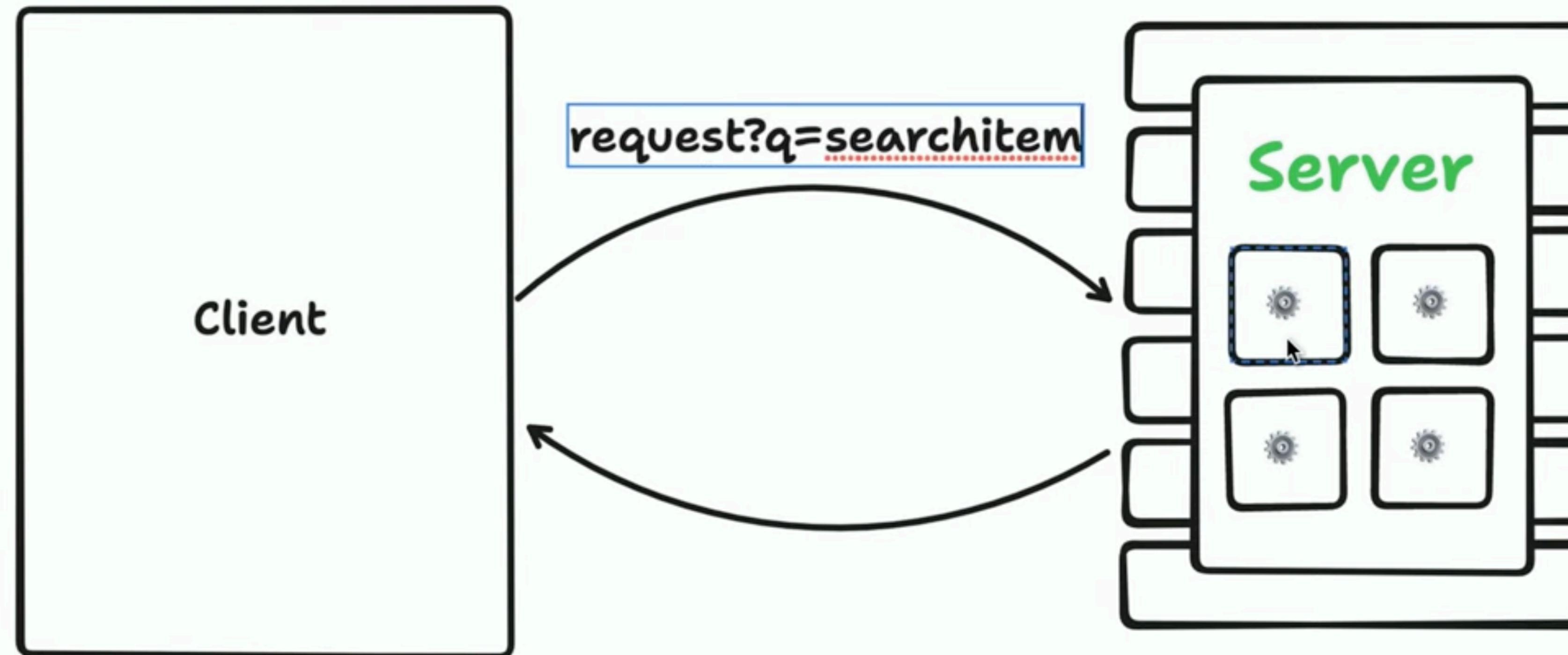


Static Site Generation

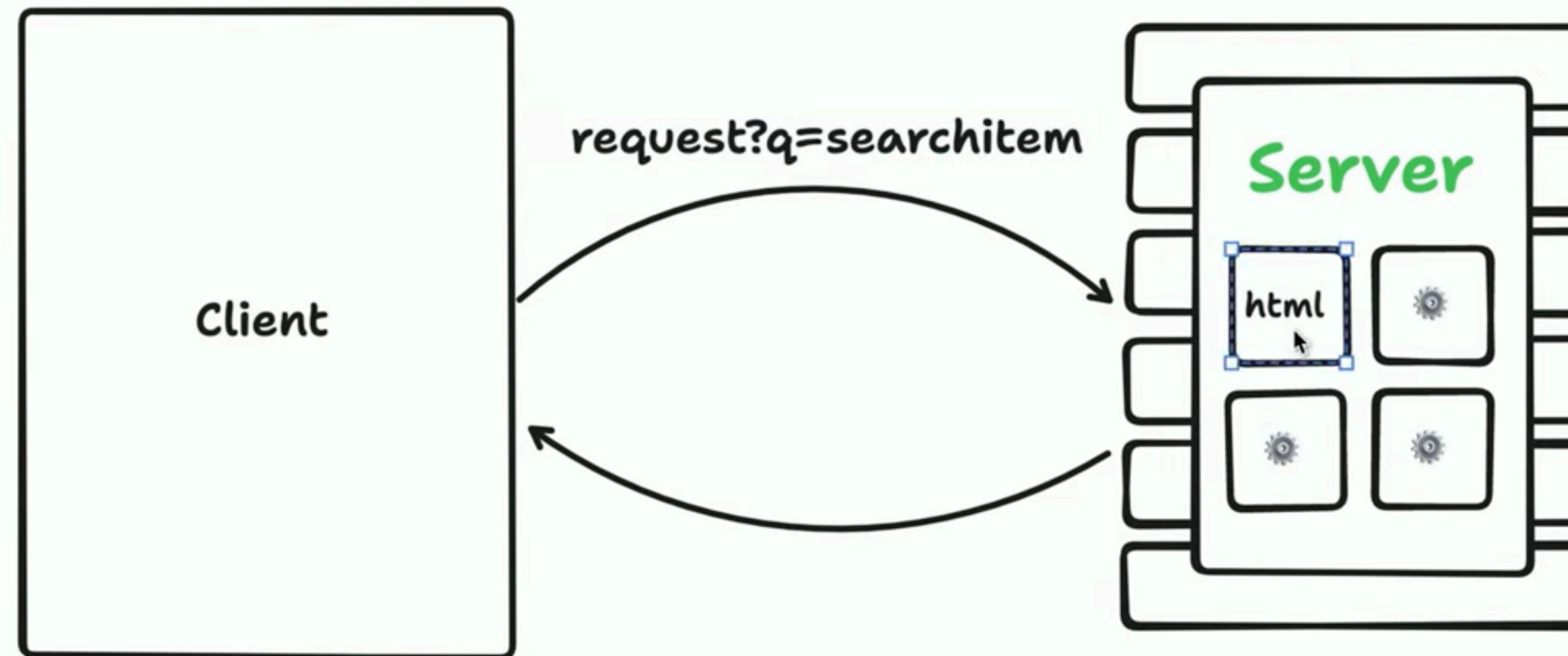
Default



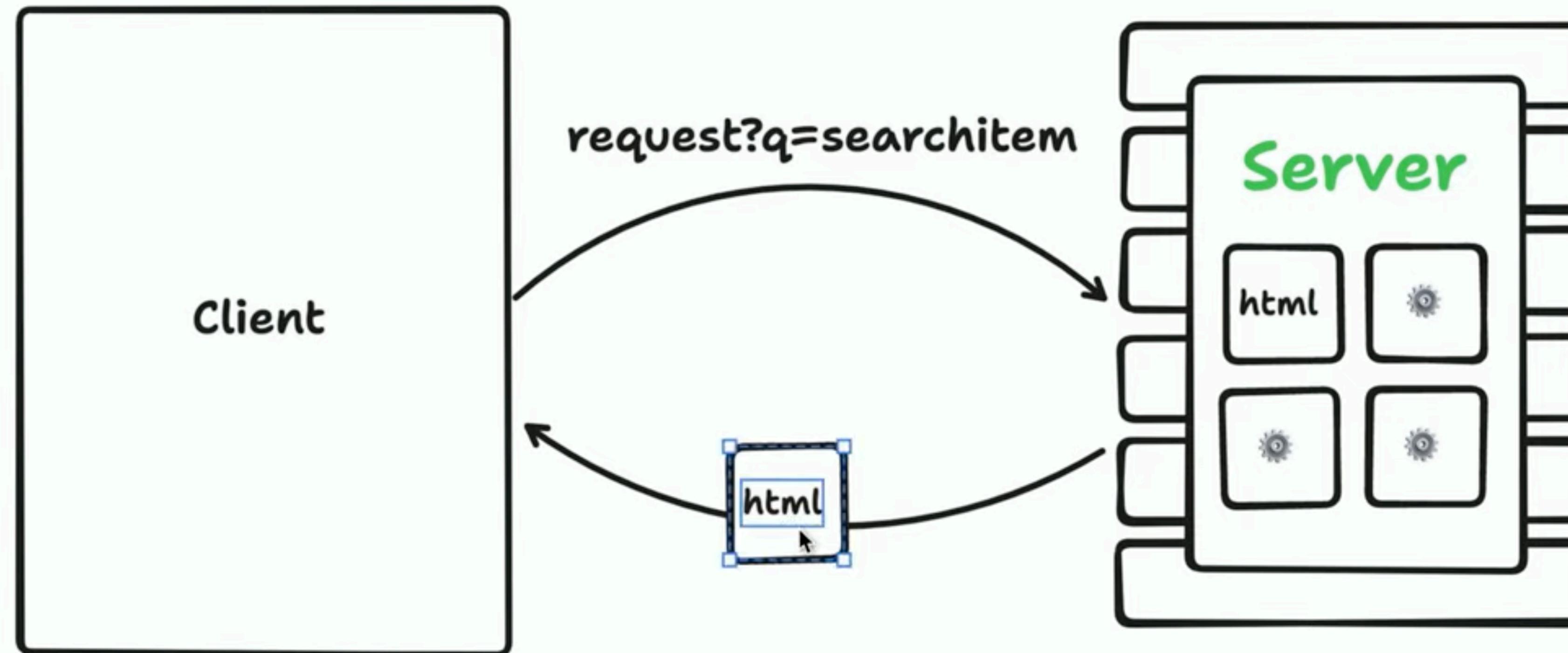
Server Side Rendered

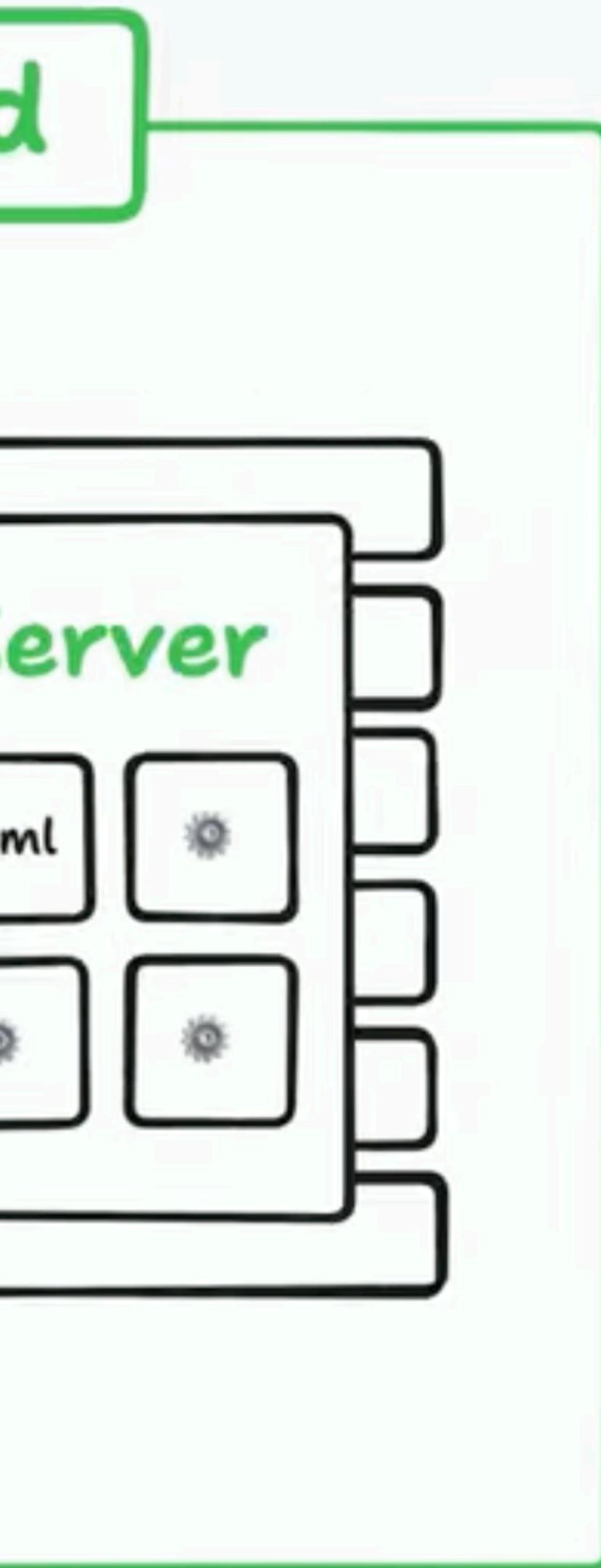


Server Side Rendered



Server Side Rendered





Defaults to SSR

```
astro.config.mjs
export default defineConfig({
  output: 'server',
  ...
});
```

```
page.astro
---
export const prerender = true;
---
```

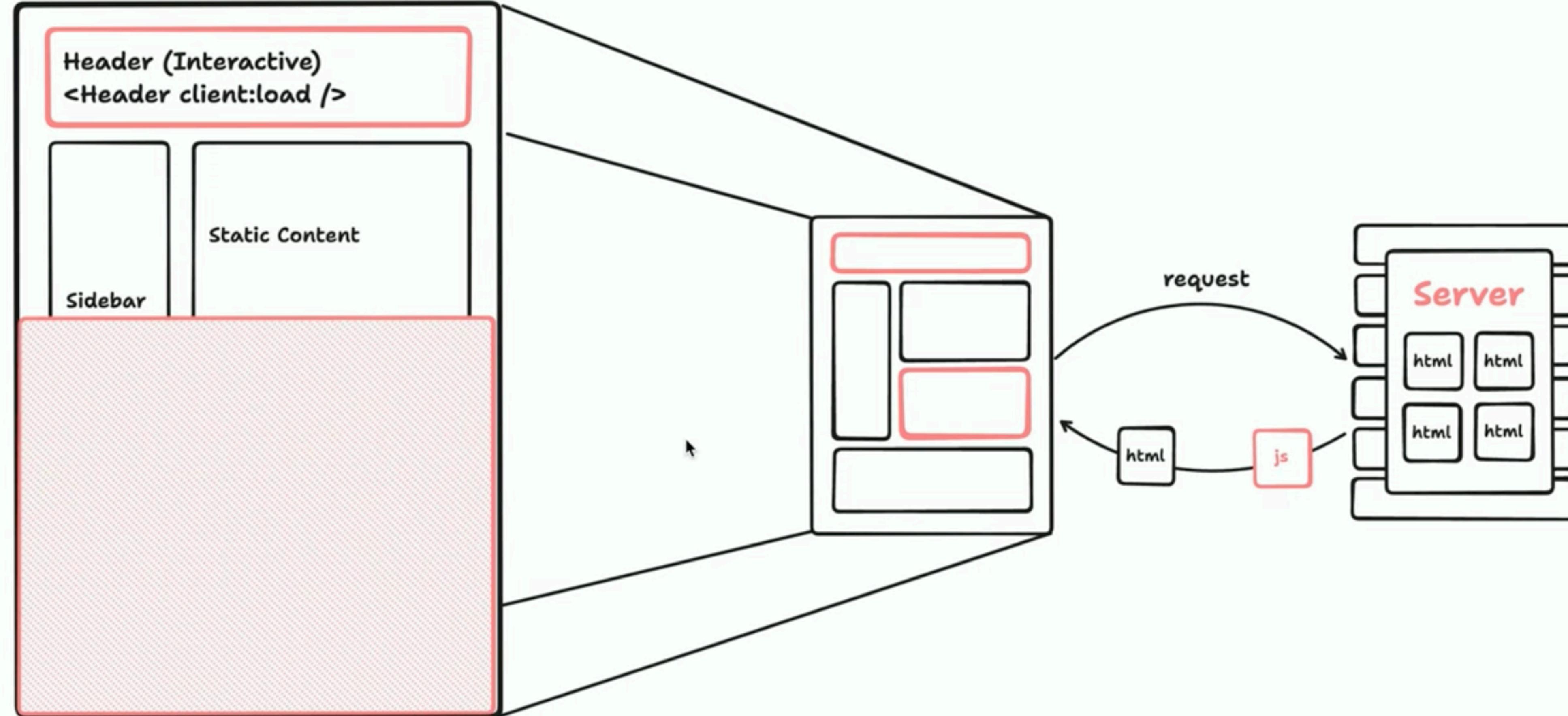
Defaults to SSG

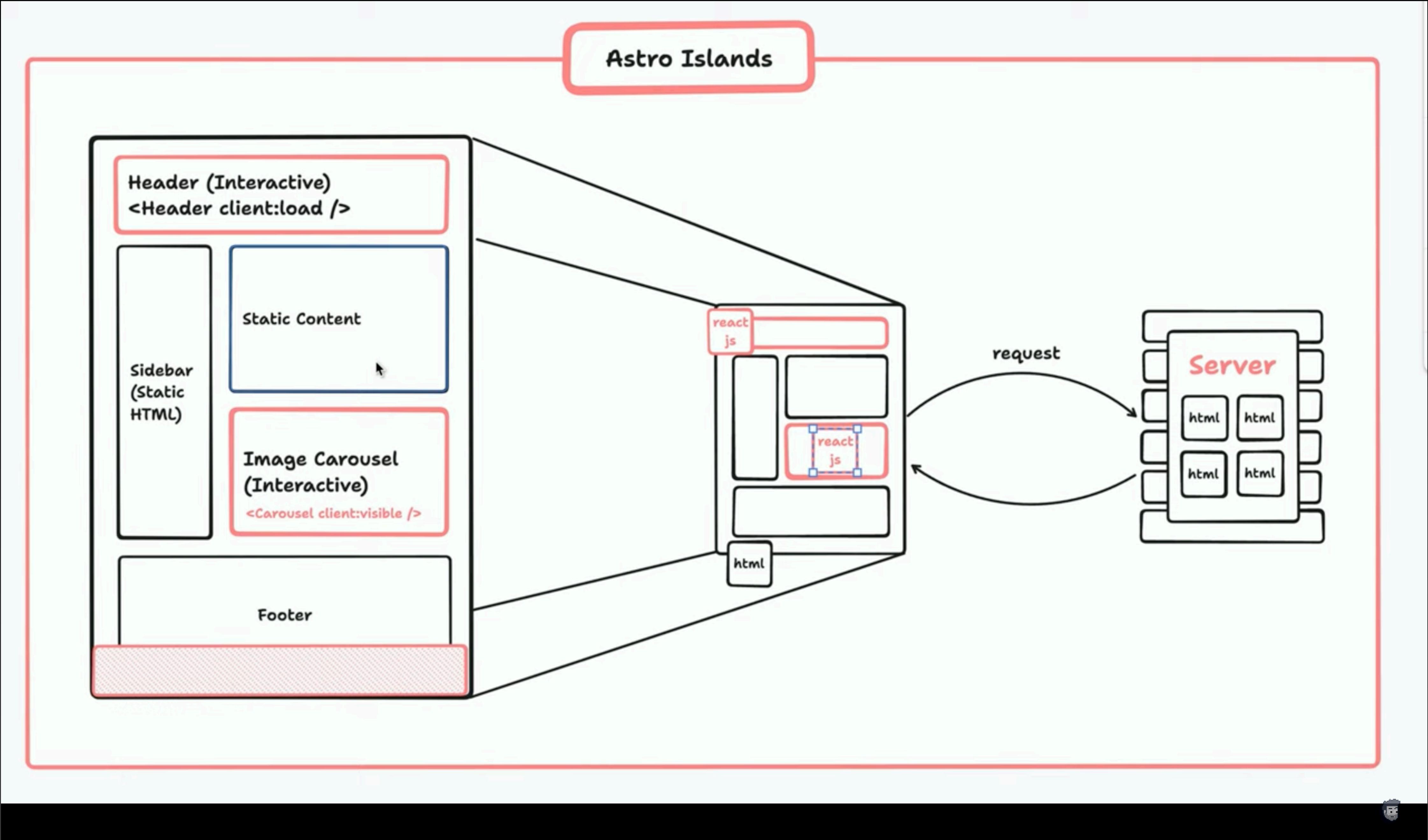
```
astro.config.mjs
export default defineConfig({
  output: 'hybrid',
  ...
});
```

```
page.astro
---
export const prerender = false;
---
```

Astro Islands

Astro Islands





Criando uma ilha

Por padrão, Astro vai renderizar automaticamente todos os componentes de UI para apenas HTML e CSS, **removendo todo o JavaScript do lado do cliente automaticamente.**

```
src/pages/index.astro
```

```
<MeuComponenteReact />
```

Isso pode soar restrito, mas esse comportamento é o que mantém websites Astro rápidos por padrão e protege desenvolvedores de acidentalmente enviar JavaScript desnecessário ou indesejado que pode tornar seu website lento.

Tornar qualquer componente de UI estático em uma ilha interativa requer apenas uma diretiva `client:*`. Astro, então, vai fazer um build automaticamente e empacotar seu JavaScript do lado do cliente para uma performance otimizada.

```
src/pages/index.astro
```

```
<!-- Este componente agora é interativo na página!
    O resto do seu website continua estático. -->
<MeuComponenteReact client:load />
```

Com ilhas, JavaScript do lado do cliente só é carregado para os componentes explicitamente interativos que você demarcar utilizando diretivas `client:*`.

E por causa das interações serem definidas a nível de componente, você pode lidar com diferentes prioridades de carregamento para cada componente baseado em seu uso. Por exemplo, `client:idle` informa um componente para carregar quando o navegador estiver inativo, e `client:visible` informa um componente para carregar apenas quando ele entrar na janela de visualização.

Obrigado pela sua atenção!

