



# Client Side rendering

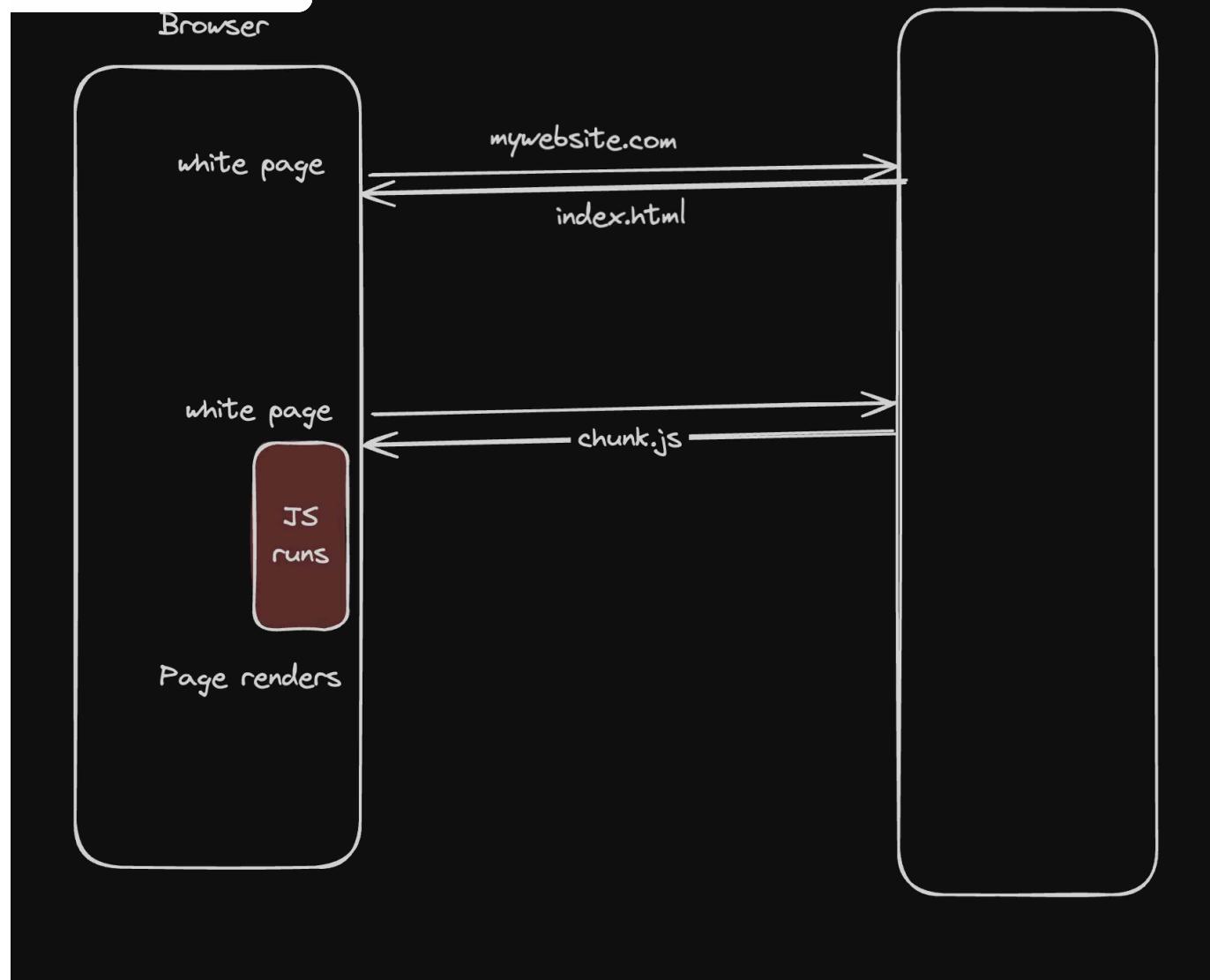
Client-side rendering (CSR) is a modern technique used in web development where the rendering of a webpage is performed in the browser using JavaScript. Instead of the server sending a fully rendered HTML page to the client

Good example of CSR – React

[Back to home](#)[Jump To ↗](#)[⟨ Prev](#)[Next ⟩](#)[Go to Top ↑](#)



## CSR vs SSR vs SSG 1 of 3



- Initialise a react project



CSR vs SSR vs SSG 1 of 3

›@latest

- Add dependencies

npm i

- Start the project

npm run build

- Serve the project

cd dist/  
serve

Open the network tab and notice how the initial HTML file doesn't have any content



## CSR vs SSR vs SSG 1 of 3

The screenshot shows the Network tab of the Vite DevTools. The 'Overview' section displays a timeline from 50 ms to 650 ms. A request for 'localhost' is selected in the list, showing its details. The 'Response' tab is selected, displaying the HTML content of the page. A red box highlights the opening `<html>` tag and the `<div id="root"></div>` div where the application content will be rendered.

Name	Headers	Preview	Response	Initiator	Timing	Cookies
localhost			<pre>7 &lt;title&gt;Vite + React&lt;/title&gt; 8 &lt;script type="module" crossorigin src="/assets/index-BPgokE0a.js" 9 &lt;link rel="stylesheet" crossorigin href="/assets/index-DiwrqTda.css" 10 &lt;/head&gt; 11 &lt;body&gt; 12 &lt;div id="root"&gt;&lt;/div&gt; 13 &lt;/body&gt; 14 &lt;/html&gt; 15</pre>			
index-BPgokE0a.js						
index-DiwrqTda.css						
injected.js						
inpage.js						
inapp.js						

This means that the JS runs and actually **populates / renders** the contents on the page

React (or CSR) makes your life as a developer easy. You write components, JS renders them to the DOM.

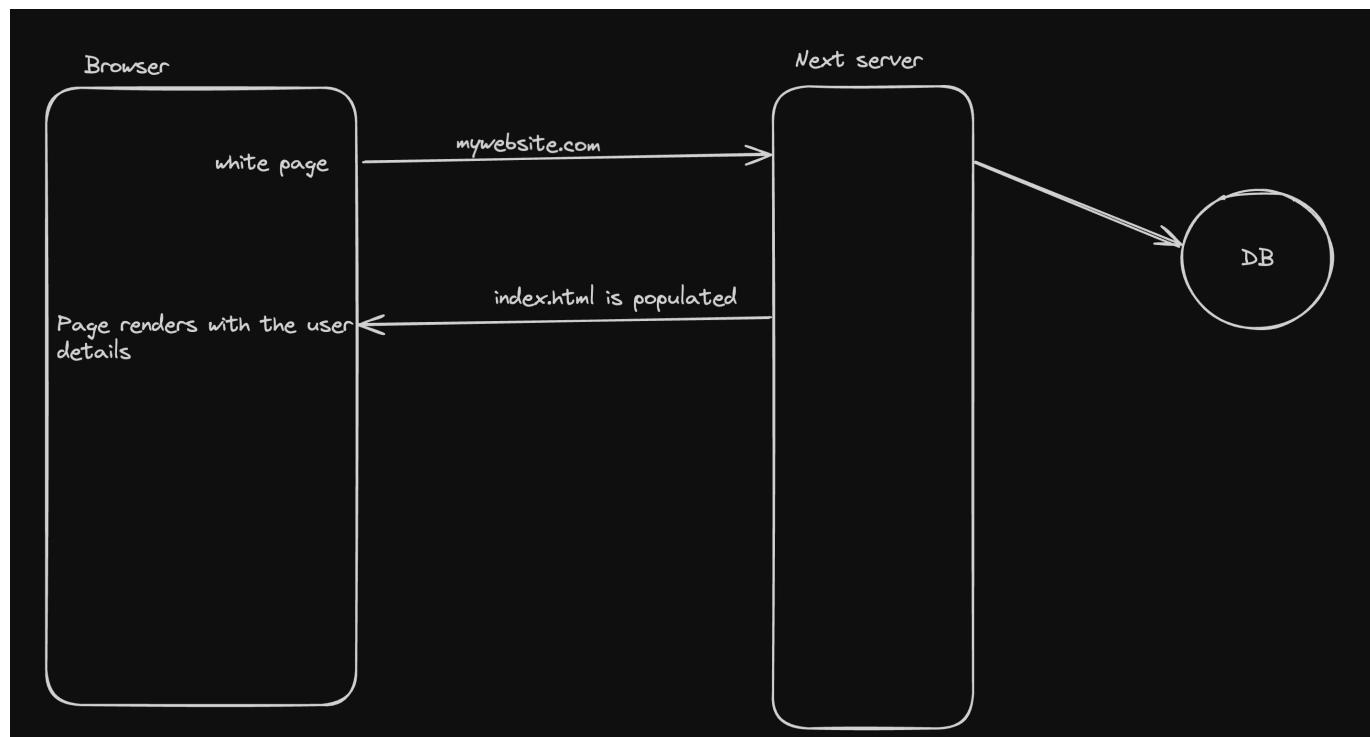
## Downsides?

1. Not SEO optimised
2. User sees a **flash** before the page renders
3. Waterfalling problem



# Server side rendering

When the `rendering` process (converting JS components to HTML) happens on the server, it's called SSR.





## 1. SEO Optimisations

CSR vs SSR vs SSG 1 of 3 Waterfalling problem

3. No white flash before you see content

Try creating a NextJS app and notice the HTML file you receive is populated

- Create next app `npx create-next-app`
- Build the project

`npm run build`

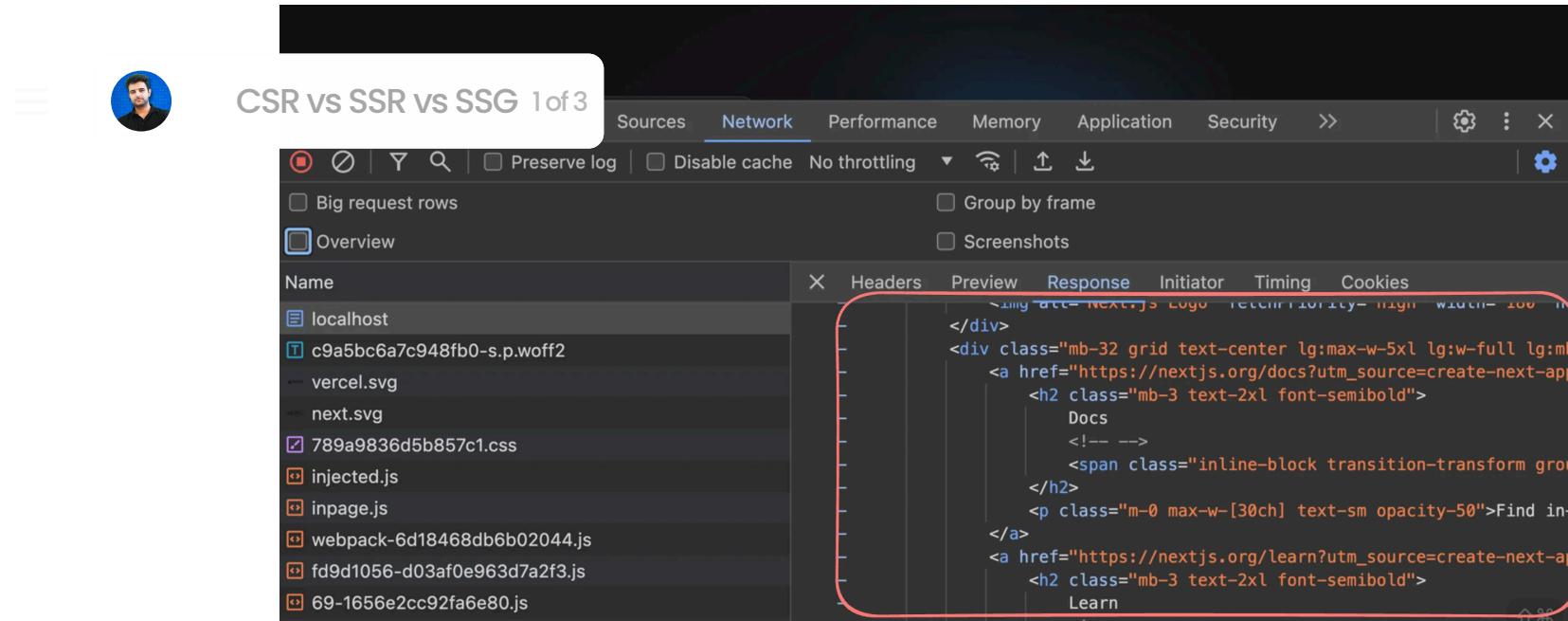


- Start the NEXT Server

`npm run start`



Notice the initial HTML page is populated



The screenshot shows the Chrome DevTools Network tab for a Next.js application. The left sidebar lists resources: 'localhost', 'c9a5bc6a7c948fb0-s.p.woff2', 'vercel.svg', 'next.svg', '789a9836d5b857c1.css', 'injected.js', 'inpage.js', 'webpack-6d18468db6b02044.js', 'fd9d1056-d03af0e963d7a2f3.js', and '69-1656e2cc92fa6e80.js'. The main pane displays the response for 'localhost', which is a static HTML page. A red box highlights the HTML content, showing the structure of the Next.js landing page with its logo, navigation links for 'Docs' and 'Learn', and a search bar.

# Static site generation

Ref <https://nextjs.org/docs/app/building-your-application/data-fetching/fetching-caching-and-revalidating>

If a page uses **Static Generation**, the page HTML is generated at **build time**. That means in production, the page HTML is generated when you run `next build`. The resulting files are then served by a web server, such as Nginx or Apache, which handles requests from users.



## CSR vs SSR vs SSG 1 of 3

If you use static site generation, you can defer the **expensive** operation of rendering a page to the **build time** so it only happens once.

### How?

Let's say you have an endpoint that gives you all the **global** todos of an app.

By **global todos** we mean that they are the same for all users, and hence this page can be statically generated.

<https://sum-server.100xdevs.com/todos>

- Create a fresh next project
- Create **todos/page.tsx**

```
export default async function Blog() {
  const res = await fetch('https://sum-server.100xdevs.com/todos')

  const data = await res.json();
  const todos = data.todos;

  console.log(todos)

  return <div>
```



## CSR vs SSR vs SSG 1 of 3

```
{todos.map((todo: any) => <div key={todo.id}>
    ...
    </div>)}
</div>
```

}

- Try updating the `fetch` requests

## Clear cache every 10 seconds

```
const res = await fetch('https://sum-server.100xdevs.com/todos', {
  next: { revalidate: 10 }
});
```



## Clear cache in a next action

```
import { revalidateTag } from 'next/cache'

const res = await fetch('https://sum-server.100xdevs.com/todos', { next: { tag: 'use server' } })
```



'use server'





CSR vs SSR vs SSG 1 of 3

```
  isync function revalidate() {  
    ('todos')  
  }
```