- Next JS static site generation is *just one thing* that Next.js is great at.

- Next JS is an **automatic static optimization** framework, which is opinionated about embracing React at its core.

- Next.js uses an opinionated **file-system based routing**

- In the early 90s you would make a request to a specific Web server using a URL. The request would ultimately get delivered to a Web server, which would then respond back with an HTML file.

- **server-side rendering**
    - The server-side rendering approach enabled tons of great Web page *but at the cost of page load time.*

- **JavaScript and AJAX**

- **Client-side rendering and SPAs**

Ditched PHP, and rendered the *entire* page using only JavaScript, running right in the Web browser.

Around this time, the concept of a "single page application" (**SPA**) took shape, in which a Web page only sent one request to the server to fetch the initial HTML, and then every update to the page afterwards was done with JavaScript and AJAX, instead of completely reloading the page.

- **Problems with client-side rendering**

  Seo, loading time

With client-side rendering, the browser receives a mostly blank HTML page, then starts downloading JavaScript, then parses, compiles, and executes the JavaScript, which then generates HTML, and then the page is finally rendered.

This all takes time.

- **The return of server-side rendering**

- Fortunately, **Node.js** was created — a technology which allows writing a Web server in JavaScript

- Libraries like ReactDOMServer made it even easier to render the app on a Node.js server.

- **hydration** - **progressively enhance**
- When combined with techniques like lazy module loading, code splitting, and bundling, this new world can result in a page that loads much faster than a fully client-rendered page.

Introducing Next.js

Problems until now:

A huge fraction of Web pages can just be static HTML files which did not require dynamic rendering.

Solution - We can use static file with ajax

So, with static HTML served from a CDN, we get the initial HTML to the user *super* quickly

The initial page load sequence can be optimized using *tons* of ingenious techniques like prefetching, preloading, preconnecting, SWR, font and stylesheet inlining, service worker caching, and much more.

New strategy:

1. If a particular page on our site is 100% static (contains no dynamic content whatsoever), then we should make sure it is available on a CDN as a static HTML file.

2. If a Web page *does* have dynamic functionality, then we should try to make as much of the page static as possible, so that the static parts can load almost instantly, and *then* we should enhance the page by fetching the data that we need from our server.

How do we do it?

Next.js to the rescue: automatic static optimization!

- It makes as much of the site static as possible and produces CDN-friendly static output for pages that are 100% static.

- If getServerSideProps is used, then the page is marked “dynamic.” Otherwise, it’s marked “static.”

- Next.js performs **automatic code splitting** when it builds each page, so that the code for each page is only loaded when you request that page, and no sooner.

Conclusion

- Next.js is a powerful React framework optimized for super-fast page load times that can handle **both static and dynamic sites** extremely well.
- To fully leverage the power of Next.js, **avoid using `getServerSideProps` and `getInitialProps` unless it is necessary**, and make sure that you write your dynamic pages by fetching data on the client side using a library like SWR.
- Even for pages that accept dynamic URL parameters like `/products/1234`, you can **use Next.js's incremental static generation feature** to generate static pages at runtime.

https://medium.com/swlh/the-hitchhikers-guide-to-next-js-fd7aa14ae8d0