How Modern Static Site Generators Work 🦖

Yangshun Tay
Front End Engineer

About Me

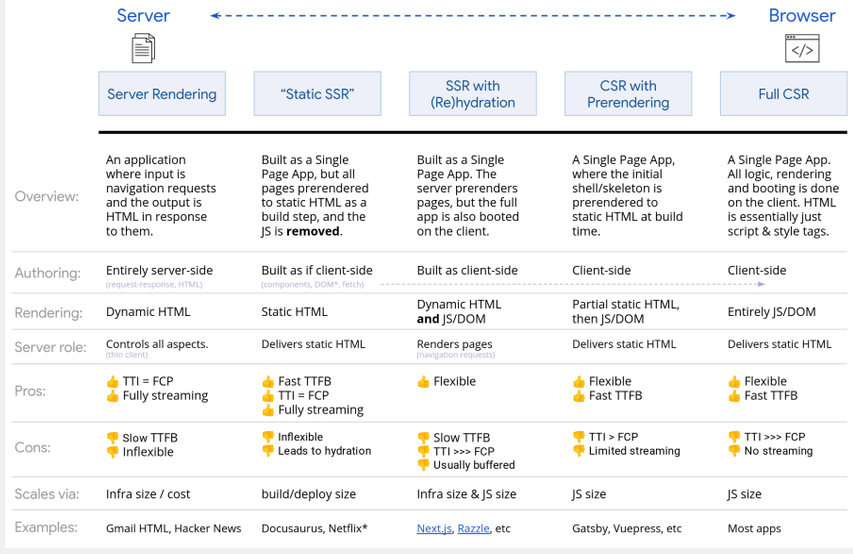
Front End Engineer @ Facebook Menlo Park

NUS School of Computing/NUSMods/NUSWhispers

Open Source - Docusaurus/Tech Interview Handbook

Static Site Generators?

- If you've learnt HTML/CSS before, one of the ways you probably would have started was to write an index.html file, wrote some HTML tags and included some CSS files inside for styling. That is an example of a simple static website.
- A static site generator is a tool that generates such static sites.

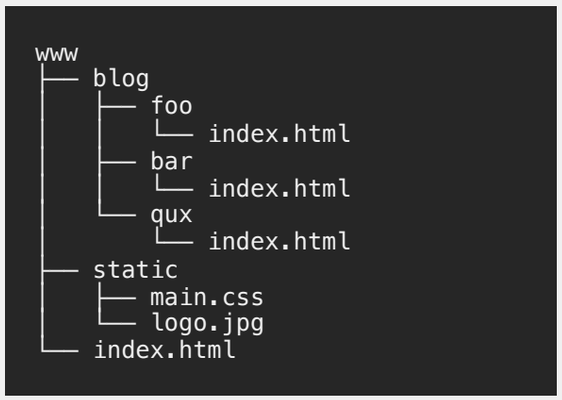


	Server Rendering	"Static SSR"	SSR with (Re)hydration	CSR with Prerendering	Full CSR
Overview:	An application where input is navigation requests and the output is HTML in response to them.	Built as a Single Page App, but all pages prerendered to static HTML as a build step, and the JS is **removed**.	Built as a Single Page App. The server prerenders pages, but the full app is also booted on the client.	A Single Page App, where the initial shell/skeleton is prerendered to static HTML at build time.	A Single Page App. All logic, rendering and booting is done on the client. HTML is essentially just script & style tags.
Authoring:	Entirely server-side (request-response, HTML)	Built as if client-side (components, DOM*, fetch)	Built as client-side	Client-side	Client-side
Rendering:	Dynamic HTML	Static HTML	Dynamic HTML **and** JS/DOM	Partial static HTML, then JS/DOM	Entirely JS/DOM
Server role:	Controls all aspects. (thin client)	Delivers static HTML	Renders pages (navigation requests)	Delivers static HTML	Delivers static HTML
Pros:	👍 TTI = FCP 👍 Fully streaming	👍 Fast TTFB 👍 TTI = FCP 👍 Fully streaming	👍 Flexible	👍 Flexible 👍 Fast TTFB	👍 Flexible 👍 Fast TTFB
Cons:	👎 Slow TTFB 👎 Inflexible	👎 Inflexible 👎 Leads to hydration	👎 Slow TTFB 👎 TTI >>> FCP 👎 Usually buffered	👎 TTI > FCP 👎 Limited streaming	👎 TTI >>> FCP 👎 No streaming
Scales via:	Infra size / cost	build/deploy size	Infra size & JS size	JS size	JS size
Examples:	Gmail HTML, Hacker News	Docusaurus, Netflix*	Next.js, Razzle, etc	Gatsby, Vuepress, etc	Most apps

Source: [Rendering on the Web]

- Over here we have the spectrum of rendering. All websites/web apps fall somewhere on the spectrum.
- Traditional SSGs are somewhere in the static SSR category
  whereas modern SSGs are closer to the client-side end are in the CSR with prerendering category.

Traditional SSGs



- Traditional static site generators are great for static websites driven by non-viewer specific content like a landing page, a blog or documentation.
- On the left you have the source files written in HTML and Markdown. And on the right, the SSG will compile the files into HTML files ready to be served by any web server.

Traditional SSGs

- Every page navigation is a full-page refresh and static assets already present on the current screen are loaded again

Traditional SSGs

- Jekyll, Hugo and Hexo are examples of traditional SSGs that have been out for a while and have been used by many

Modern SSGs

Client-side navigation

Rich interactive content

Route-based code-splitting

Asset pipeline (with the help of a module bundler)

Progressive Web Apps

- What makes modern SSGs different? Modern SSGs improve on the developer experience and performance.
- The main difference being modern SSGs use module bundlers that treat every type of resource as a module, including images.
- Hence with modern SSGs, it is easier to post-process assets thanks to the use of module bundlers.

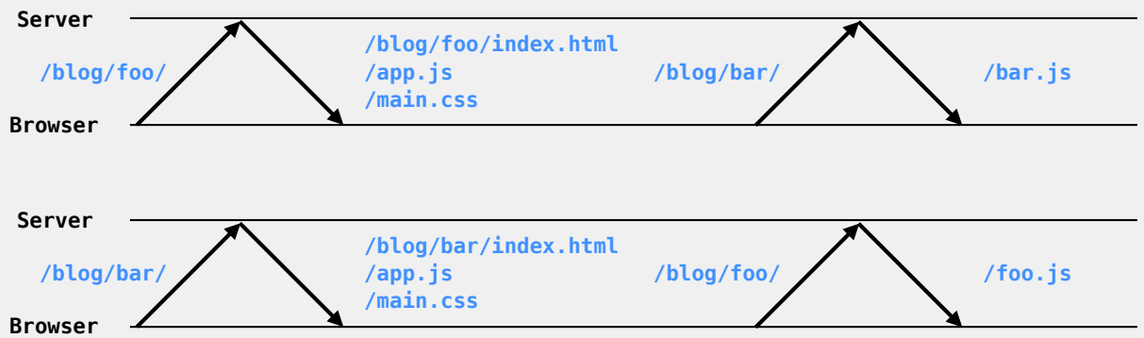
Modern SSGs

```
www
├── blog
│   ├── foo.md
│   ├── bar.md
│   └── qux.md
├── layouts
│   ├── Default.jsx
│   └── Page.jsx
├── static
│   ├── main.css
│   └── logo.jpg
└── index.html
```

```
www
├── blog
│   ├── foo
│   │   └── index.html
│   ├── bar
│   │   └── index.html
│   └── qux
│       └── index.html
├── static
│   ├── main.css
│   └── logo.jpg
├── app.js
├── foo.js
├── bar.js
├── qux.js
└── index.html
```

- The original source files of a modern SSG look really similar to a traditional SSG. But the output is pretty different.
- For each path there's both an index.html file and a corresponding .js file generated.

Modern SSGs



- In modern SSGs, after the initial load (where the HTML file is loaded, subsequent page navigations fetch the JS file for the next page and do a client-side transition, exactly like how route-splitter single-page apps work.

Modern SSGs

- Gatsby, VuePress and Docusaurus are examples of modern SSGs.

Modern SSG Stack

Bundler - Webpack/Rollup

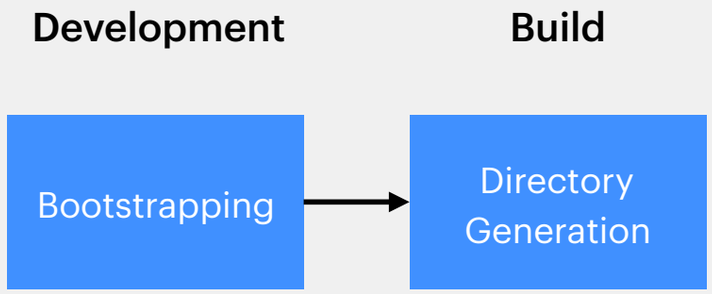
Templating/View - React/Vue

Router - React Router/Reach Router/Vue Router

Styling - Unopinionated

- Modern SSGs have similar architectures and consist of common parts such as a bundler, templating engine, router.
- Styling is however pretty unopinionated and users are free to choose their favorite solution

Lifecycle



- SSGs as they implied by the generators in their name, are actually compilers.
- During development, the bootstrap lifecycle process runs. During build, an additional step has to occur, which is to server-side render each page (prerender) as directories.

Bootstrap Lifecycle

	Blog Plugin	Sitemap Plugin	Google Analytics Plugin
`loadContent()`	Scan and load Markdown files	-	-
`contentLoaded()`	Create routes for each file	-	-
`configureWebpack()`	Add Markdown loader	-	-
`getClientModules()`	-	-	Inject analytics.js
`postBuild()`	-	Generate sitemap	-

- Every SSG's bootstrap process goes through similar phases - fetching content, allowing extensions of the bundler config, injecting client side modules and some other hooks into the lifecycle of an app.
- Plugins are a way to spread the functionality across various lifecycle methods. They're only called when a certain lifecycle method is activated.

Bootstrapping

Files + Data = Routes

```
// Generated routes.js
[
  {
    path: '/',
    component: 'HomePage.jsx',
  },
  {
    path: '/blog/foo',
    component: 'BlogPost.jsx',
  },
  {
    path: '/blog/bar',
    component: 'BlogPost.jsx',
  },
...
]
```

- The value an SSG framework provides over an SPA is that routes are based on the directory structure of your project instead of you having to explicitly define it.
- Hence it's important that the bootstrap process generates a routes file for you.

Directory Generation

Typical Single-Page App

Generate root index.html and core JS bundle

```
www
├── static
│   ├── main.css
│   └── logo.jpg
├── app.js
└── index.html
```

Modern Statically Generated Site

Generate additional index.html and JS bundle for every route

```
www
├── blog
│   ├── foo
│   │   └── index.html
│   ├── bar
│   │   └── index.html
│   └── qux
│       └── index.html
├── static
│   ├── main.css
│   └── logo.jpg
├── app.js
├── foo.js
├── bar.js
├── qux.js
└── index.html
```

- We want the website to function as a single-page app after the initial load. Hence there's an additional step of generating an index.html for every entry point available.
- Another benefit of this is that if JavaScript is disabled, the site still works as there are HTML files corresponding to each path.

THANK YOU!