CSR vs SSR vs SSG vs ISR

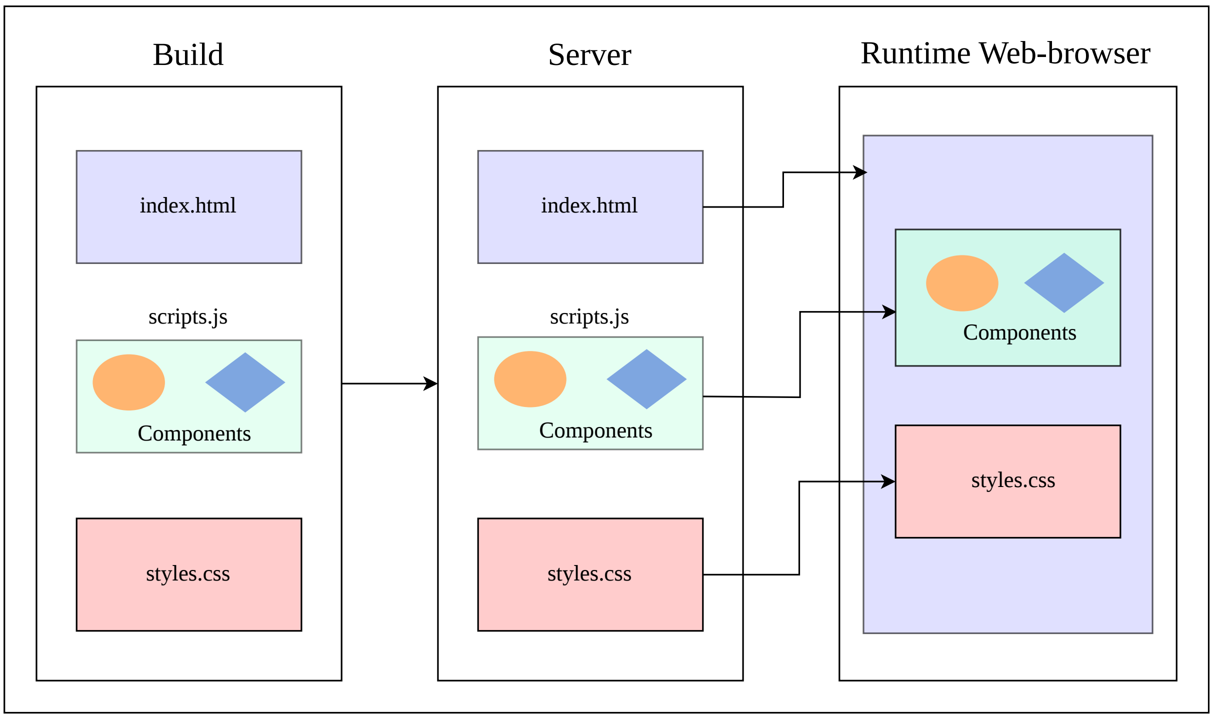
Rendering strategies are important in shaping user experiences in web development. Let’s explore four prominent rendering approaches: client-side rendering (CSR), server-side rendering (SSR), static site generation (SSG), and incremental static regeneration (ISR).

Client-side rendering (CSR)

- **Client-side rendering** involves rendering content on the client’s browser using JavaScript.
- This approach enhances interactivity and reduces server load, as the server primarily serves raw data.
- In this approach, the web browser initially loads a basic HTML file with minimal content. JavaScript and styles, which are loaded later, are responsible for rendering the full user-friendly page in the web browser.

How does CSR work?

- Upon loading a page, the client’s browser retrieves minimal HTML and JavaScript.
- The JavaScript fetches data and renders components dynamically. However, SEO (Search Engine Optimization) can be compromised due to search engines struggling to index content rendered via JavaScript.



Use cases and examples

CSR is suitable for applications requiring frequent updates and dynamic content, such as real-time chat applications or social media platforms.

Popular frameworks like React, Vue.js, and Angular enable efficient CSR implementation.

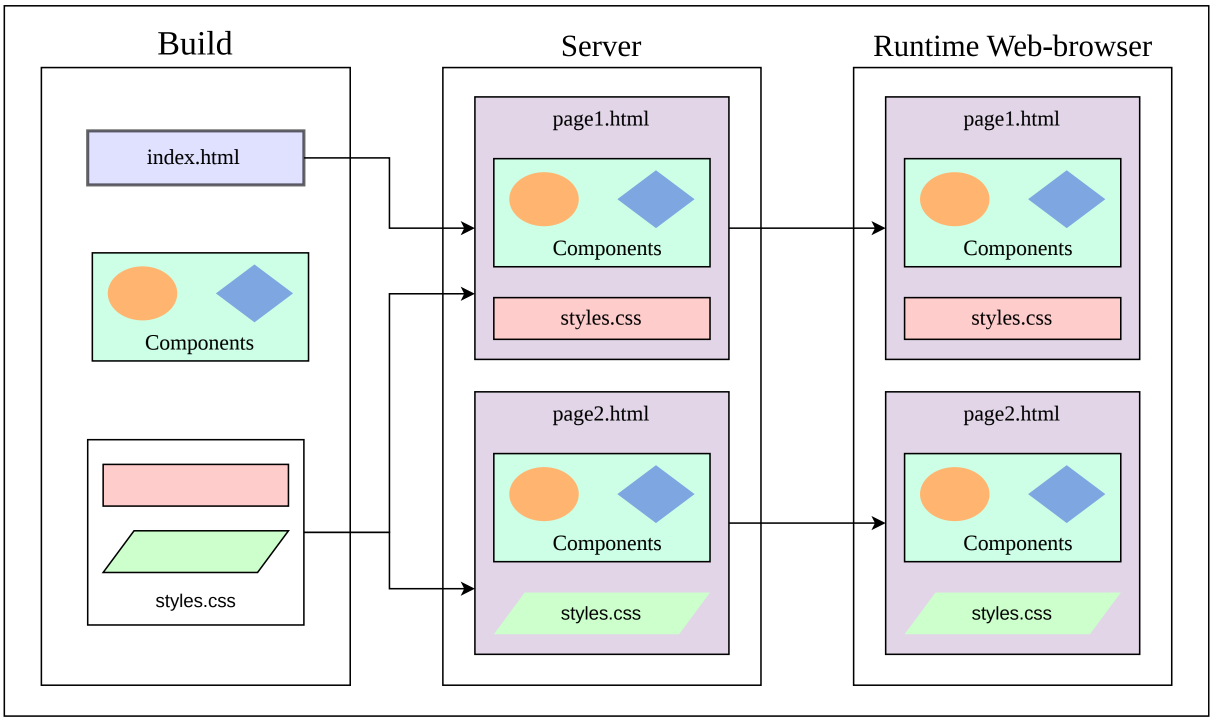
Server-side rendering (SSR)

- **Server-side rendering** involves generating HTML content on the server before delivering it to the client.
- This approach enables search engines to index content effectively, resulting in improved SEO and faster initial page loads.

How does SSR work?

When a user requests a page, the server fetches the necessary data and generates the HTML content. The fully rendered page is then sent to the client.

Despite the benefits, SSR can introduce challenges related to server load and complex data fetching.



Use cases and examples

SSR is ideal for content-heavy websites, blogs, and applications that prioritize SEO.

Frameworks like Next.js provide built-in server rendering capabilities, making it easier to implement.

Hydration:

Hydration is the process where **JavaScript attaches interactivity** to an **already-rendered HTML page** (usually from SSR or SSG). The static HTML is sent by the server, and then JavaScript "hydrates" it by:

- Attaching event listeners
- Initializing component state
- Enabling dynamic behavior

Rendering Type	Is HTML pre-rendered?	Does Hydration Happen?	When Is UI Built?
SSR + Hydration	✅ Yes (server renders HTML)	✅ Yes	Server renders HTML, client hydrates
SSG + Hydration	✅ Yes (build-time HTML)	✅ Yes	HTML is static, JS hydrates on load

CSR	❌ No	❌ No	Entire UI is built in browser via JS

In CSR:

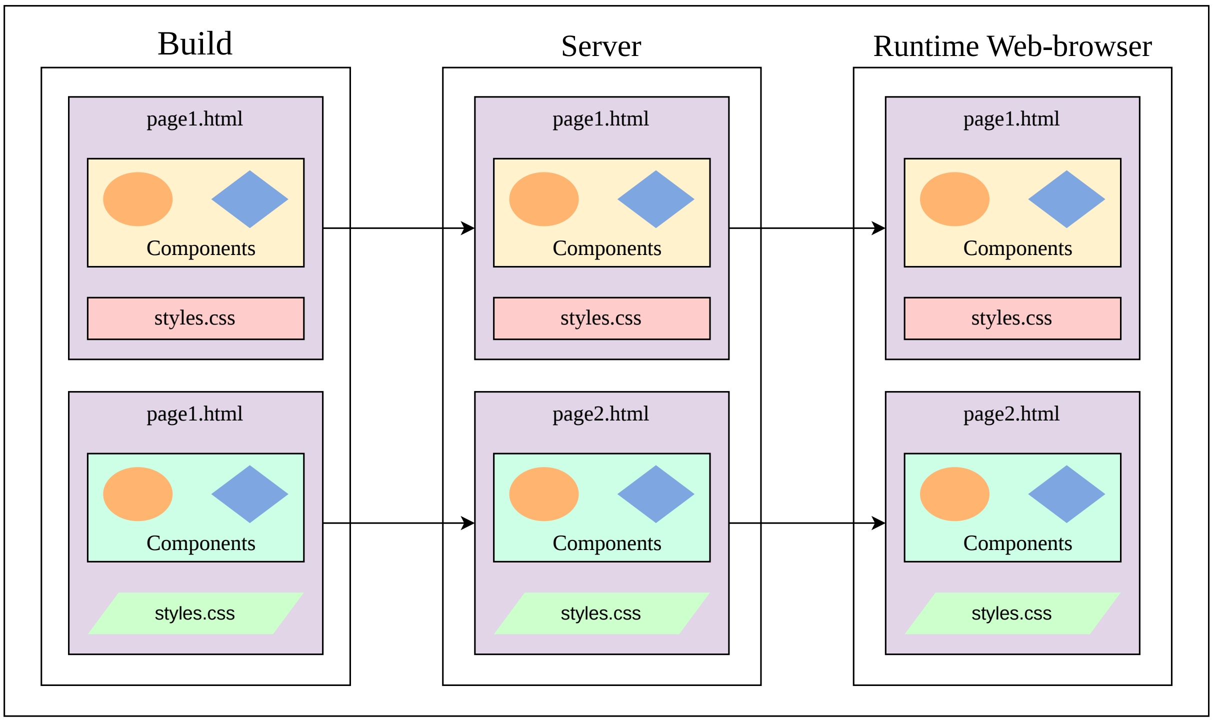
- The **server sends a blank HTML shell** (e.g., just `<div id="root"></div>`).
- JavaScript then **generates the full UI from scratch** on the client.
- Since there's no HTML to hydrate, **hydration is not needed or possible**.

Static site generation (SSG)

- **Static site generation** refers to pre-rendering all pages as static HTML files during the build process.
- To minimize rendering on every request, the files are generated during build time, allowing pages to be served instantly upon user request.
- This strategy offers lightning-fast loading times and enhanced security.

How does SSG work?

During the build phase, the application generates HTML files for each page. These static files are then served to users, minimizing the need for server-side rendering and database queries.



Use cases and examples

SSG is excellent for content-focused websites and portfolios. Tools like Gatsby and Nuxt JS make SSG implementation straightforward, ensuring optimal performance and security.

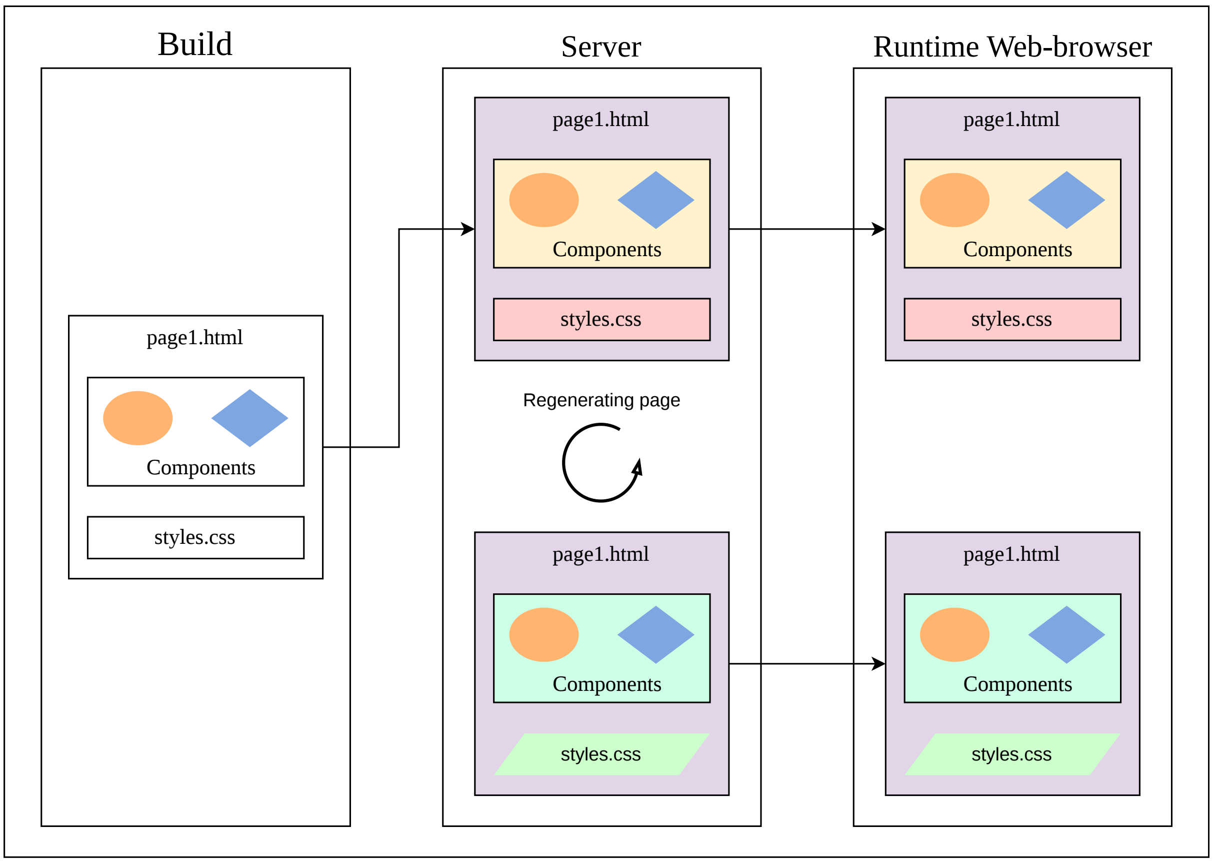
Incremental static regeneration (ISR)

- **Incremental static regeneration** combines elements of SSR and SSG.
- It allows us to pre-render static pages during build time while periodically regenerating specific pages with updated data.
- ISR represents an advancement over SSG, involving regular rebuilding and revalidating of pages to prevent content from becoming excessively outdated.

How does ISR work?

Static pages are generated at build time, and dynamic pages are regenerated incrementally based on specific triggers.

This strategy balances the benefits of static content and real-time data updates.


Build
Server
Runtime Web-browser
page1.html
Components
styles.css
page1.html
Components
styles.css
page1.html
Components
styles.css
Regenerating page
page1.html
Components
styles.css
page1.html
Components
styles.css

Use cases and examples

ISR is ideal for websites with content that requires frequent updates, such as blogs or news portals.

Frameworks like Next JS provide ISR capabilities to achieve the perfect balance between static content and dynamic data.

Comparing SSR vs. CSR vs. ISR vs. SSG

	SSR	CSR	ISR	SSG
SEO benefits	High	Compromised	High	High
Initial load time	Moderate	Fast	Moderate	Very Fast
Interactivity	Limited	High	Variable	Limited
Server load	Moderate	Low	Low	Very Low
Data fetching	Server-side	Client-side	Hybrid	Build-time
Suitable use cases	SEO optimization,content-heavy apps	Real-time updates, dynamic content	Frequent data updates, blogs	Content-focused websites, portfolios