explanin ssr, ssg, csr, hydration error in depth with example

SSR (Server-Side Rendering), SSG (Static Site Generation), CSR (Client-Side Rendering), and **hydration errors** are foundational Next.js rendering concepts. Here's a detailed explanation of each, with real-world examples and causes of hydration errors.[1] [2] [3] [4]

SSR (Server-Side Rendering)

In **SSR**, the page HTML is generated on the server for each request. This is best for dynamic pages that change per user or visit (e.g., user dashboards, frequently updated content), and is great for SEO and fast initial loading.[5] [2]

Example: SSR in Next.js

```
// pages/index.js
export default function Home({ data }) {
  return <div>Data from server: {data.value}</div>;
}

export async function getServerSideProps() {
  // Fetch data for each request (API, DB, etc.)
  const res = await fetch('https://api.example.com/data');
  const data = await res.json();
  return { props: { data } };
}
```

This ensures each visitor gets the most up-to-date content, as the HTML sent to the browser is freshly generated with every page load.[6] [2]

SSG (Static Site Generation)

With **SSG**, HTML is generated at build time and served statically for every request. Use SSG for content that rarely changes and is the same for all users, such as blog posts or documentation pages. This offers the best performance as pages are prebuilt.[7] [2] [8]

Example: SSG in Next.js

```
// pages/posts.js
export default function Posts({ posts }) {
  return <ul>{posts.map(post => <li key={post.id}>{post.title}</li>)}</ul>;
}
```

```
export async function getStaticProps() {
  const res = await fetch('https://api.example.com/posts');
  const posts = await res.json();
  return { props: { posts } };
}
```

The page never queries the backend at request time – it displays the same static HTML until the site is rebuilt (for example, after updating content).[2] [7]

CSR (Client-Side Rendering)

CSR means the initial HTML is minimal and the browser fetches the rest of the data via JavaScript after loading. Ideal for dashboard-type apps, interactive features, or content that's user-specific and not SEO-critical.[3] [1]

Example: CSR in Next.js

```
import { useEffect, useState } from "react";
export default function Page() {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch("https://api.example.com/data")
      .then(res => res.json())
      .then(setData);
  }, []);
  return <div>{data ? data.value : "Loading..."}</div>;
}
```

Nothing is fetched until the browser runs the JavaScript, so the user may see a blank or loading state at first.[3]

Hydration Error

Hydration is the process where React enables interactivity on a page that was server-rendered. If the server-rendered HTML doesn't match what the React client expects, a **hydration error** occurs. Causes often include dynamic values (dates, random numbers), data differences, or incorrect HTML structure.[9] [4]

Example of Hydration Error

```
// Causes error due to different random numbers
export default function Home() {
  return <div>Random number: {Math.random()}</div>;
}
```

- On the server, React renders a specific value (e.g., 0.135).
- On the client, React re-renders and gets a different value (e.g., 0.857), causing a mismatch.

Correcting Hydration Errors
Move dynamic JS-only values to the client using hooks:

```
import { useEffect, useState } from "react";
function RandomComponent() {
  const [random, setRandom] = useState(null);
  useEffect(() => {
    setRandom(Math.random());
  }, []);
  return <div>Random: {random}</div>;
}
```

Other strategies include disabling SSR for specific components, or using the `suppressHydrationWarning` prop.[4] [9]

Summary Table

Rendering Type	When to Use	Example Code Function	SEO	Performance	Fresh Data	Example Use
SSR	Dynamic, per-user content	`getServerSideProps`	Good	Medium	Always	Dashboards, news feeds
SSG	Static, rarely-changing content	`getStaticProps`	Best	Best	At build	Blogs, docs, landing
CSR	Interactive, client-specific	`useEffect` for fetching data	Weak	Slower	Always	User settings, widgets

All these rendering modes can be mixed in a Next.js site, choosing the technique per page or component as needed.[8] [1] [2] [4] [3]

⁂

1. https://dev.to/smartterss/client-side-vs-server-side-rendering-in-nextjs-explained-46n
2. https://strapi.io/blog/ssr-vs-ssg-in-nextjs-differences-advantages-and-use-cases
3. https://nextjs.org/docs/pages/building-your-application/rendering/client-side-rendering
4. https://www.geeksforgeeks.org/reactjs/how-to-fix-hydration-errors-in-server-rendered-components-in-next-js/
5. https://www.geeksforgeeks.org/nextjs/what-is-ssr-in-nextjs/
6. https://www.freecodecamp.org/news/server-side-rendering-in-next-js-for-improved-seo/
7. https://theanshuman.dev/articles/what-the-heck-is-ssg-static-site-generation-explained-with-nextjs-5cja
8. https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation
9. https://dev.to/georgemeka/hydration-error-4n0k
10. https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering
11. https://blog.appsignal.com/2023/12/13/server-side-rendering-with-nextjs-react-and-typescript.html
12. https://www.youtube.com/watch?v=WAMqFdCFotY

13. https://nextjs.org/docs/messages/react-hydration-error
14. https://stackoverflow.com/questions/73162551/how-to-solve-react-hydration-error-in-nextjs
15. https://sentry.io/answers/hydration-error-nextjs/
16. https://www.youtube.com/watch?v=A6TlNMclAbA
17. https://swhabitation.com/blogs/how-to-fix-hydration-errors-nextjs
18. https://www.linkedin.com/pulse/avoiding-hydration-errors-nextjs-technical-guide-ali-hamza
19. https://www.reddit.com/r/nextjs/comments/1gabiqn/hydration_error_when_installing_nextjs_15/