Server-Side Rendering (SSR) in Next.js

Understanding SSR

Server-Side Rendering (SSR) is a method of rendering web pages on the server at the time of a request. Unlike Static Site Generation (SSG), where pages are pre-rendered at build time, SSR dynamically generates HTML pages on the server for each incoming request. The server fetches any necessary data, renders the React components to HTML, and then sends this HTML to the client.

Benefits of SSR:

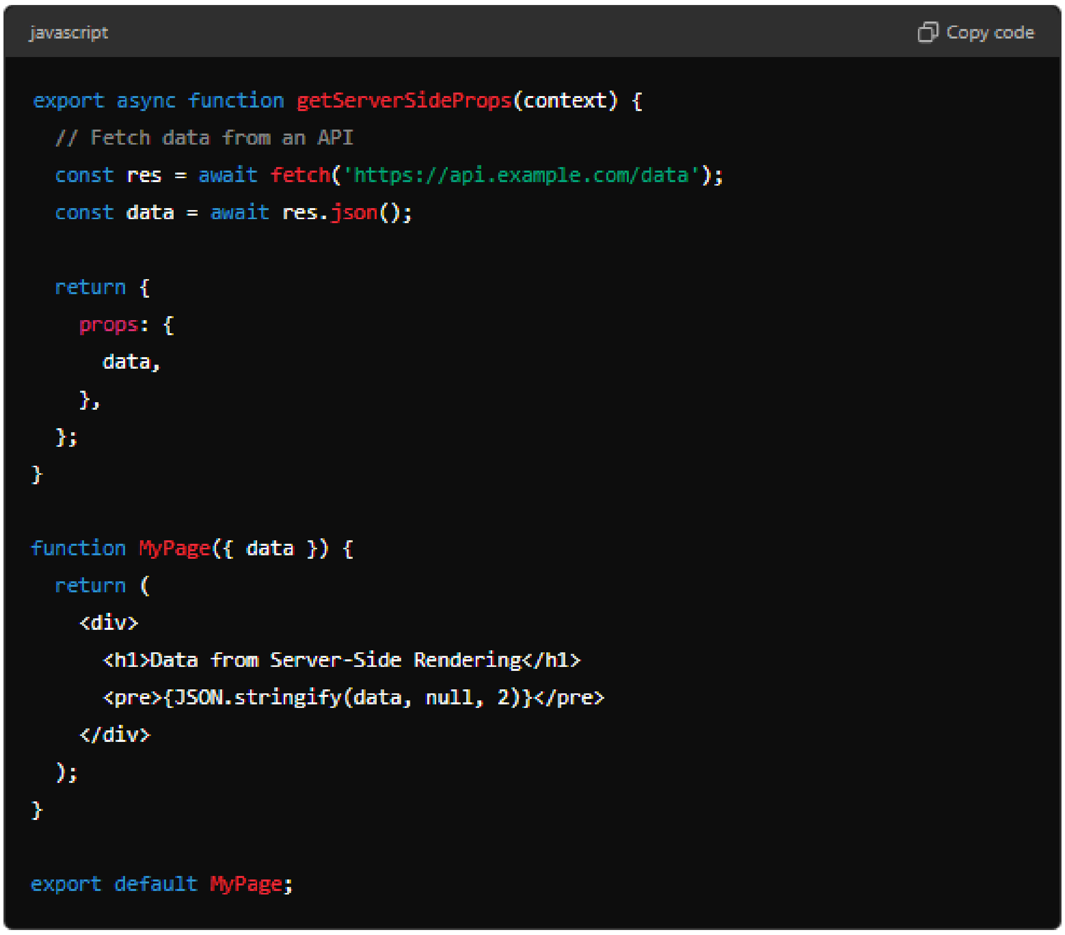
- **Dynamic Content:** Pages are always up-to-date since they are rendered at request time.
- **SEO-Friendly:** Like SSG, SSR provides pre-rendered HTML, making content easily indexable by search engines.
- **Personalization:** SSR can be used to tailor content for each user based on cookies, user authentication, or other request-specific data.

Implementing SSR in Next.js

Next.js simplifies the implementation of SSR with the getServerSideProps function. This function allows you to fetch data on the server and pass it as props to your page component at request time.

How to use getServerSideProps:

1. **Create a Page Component with SSR:**
    - To use SSR in a Next.js page, export the getServerSideProps function from your page component.
    - This function runs on the server for every request, fetching the necessary data and returning it as props.



```javascript
export async function getServerSideProps(context) {
  // Fetch data from an API
  const res = await fetch('https://api.example.com/data');
  const data = await res.json();

  return {
    props: {
      data,
    },
  };
}

function MyPage({ data }) {
  return (
    <div>
      <h1>Data from Server-Side Rendering</h1>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
}

export default MyPage;
```

□ **Advantages of Using SSR in Next.js:**

- **Fresh Data:** Always serves the latest data, making it suitable for content that changes frequently.
- **Custom Content:** Can personalize content based on user session or other request-specific information.
- **Integration with External APIs:** Fetch data from external APIs or databases and pass it directly to your component.

□ **Performance Considerations:**

- **Latency:** SSR can introduce additional latency compared to SSG since the server must render the page for each request.
- **Caching:** You can use caching strategies to mitigate performance concerns, such as caching API responses or using edge caching/CDNs to reduce load times.

□ **When to Use SSR:**

- Pages that require dynamic content based on the user's request or session.

- Situations where SEO is crucial, and you need up-to-date content for search engines.
- Content that changes frequently and needs to reflect the latest state for each visitor.

Next.js provides a flexible way to choose between SSG, SSR, and CSR (Client-Side Rendering) depending on the needs of your application, allowing you to optimize for performance, SEO, and user experience.