20: Server-Side Rendering (SSR) in React

Last updated by | Subramanya Dixit | May 5, 2025 at 7:59 PM GMT+5:30

📘 Why SSR is Important

Server-Side Rendering (SSR) is a technique where the content of a React application is rendered on the server, rather than the client. This approach can improve the app's performance, SEO, and overall user experience by delivering a fully rendered page on the first request. SSR ensures that search engines can crawl the content of the page and index it properly.

🧠 Key Concepts

✅ What is Server-Side Rendering (SSR)?

SSR involves rendering React components on the server, sending the HTML to the client, and then hydrating the React components on the client-side. This allows the page to load faster, with the initial content already available when the user requests the page, providing an enhanced user experience.

- **Faster Initial Load**: The browser can render the page quickly, as the HTML is already generated and ready to display.
- **Improved SEO**: Search engine crawlers can index content that is rendered on the server, which is crucial for SEO.
- **Better Performance**: SSR can reduce the time to interactive (TTI) and improve perceived performance for users.

✅ How SSR Works with React

In SSR, the React components are rendered on the server using **Node.js** and then sent to the client as static HTML. Once the HTML is loaded, React "hydrates" the page, which means it attaches event listeners and makes the page interactive.

1. **Server-Side Rendering Process**:
   - The user sends a request to the server for a page.
   - The server renders the React components and sends back the fully rendered HTML.
   - The browser displays the HTML content while React "hydrates" the page, enabling interactivity.
   - React takes over and manages subsequent user interactions in the app.

✅ Setting Up SSR with React

To set up SSR with React, you'll typically use **Node.js** and a server-side rendering framework like **Next.js** or **ReactDOMServer**.

1. **Creating the React App**:
   You need to create a React app and use ReactDOMServer to render the app server-side.
2. **ReactDOMServer API**:
   The `ReactDOMServer` module is used to render a React component to a static HTML string.

```
import React from 'react';
import ReactDOMServer from 'react-dom/server';
import App from './App';

const html = ReactDOMServer.renderToString(<App />);
```

3. **Setting Up a Basic Node Server for SSR**:
   A simple Node.js server that serves the React app rendered on the server.

```
import express from 'express';
import React from 'react';
import ReactDOMServer from 'react-dom/server';
import App from './App';

const app = express();

app.get('*', (req, res) => {
const content = ReactDOMServer.renderToString(<App />);
res.send( <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>SSR with React</title>
</head> <body> <div id="root">${content}</div> </body> </html> );
});

app.listen(3000, () => {
console.log('SSR app listening on port 3000');
});
```

4. **Hydration on the Client Side**:
   On the client side, React hydrates the rendered HTML to make it interactive.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.hydrate(
<App />,
document.getElementById('root')
);
```

✅ Advantages of SSR

1. **Improved SEO**: Search engines can crawl and index the content of a page, improving visibility.
2. **Faster Initial Load**: The user receives a fully-rendered HTML page, speeding up the time to the first meaningful paint.
3. **Better User Experience**: Pages load faster, even on slower networks or devices.

✅ Challenges of SSR

1. **Server Load**: The server needs to render the components for every request, which can put additional load on the server.
2. **Complexity**: SSR introduces complexity in terms of the app's setup, requiring server-side logic in addition to client-side rendering.

3. **Potential Performance Issues**: React hydration can be expensive and may lead to performance issues if not optimized.

💡 Guidelines

- Use SSR for applications where SEO is a priority, such as blogs or e-commerce sites.
- Use frameworks like **Next.js** or **Gatsby** for easier SSR setup.
- Remember that SSR can increase server load, so it's essential to optimize server performance.
- Hydrate React on the client side to make the page interactive once the HTML is rendered.

📝 Practice Exercises

1. Set up SSR in a simple React app using **ReactDOMServer** and **Express**.
2. Implement a basic SSR app with a simple `App` component.
3. Configure your app to hydrate on the client side after being rendered by the server.
4. Implement dynamic content fetching in SSR, ensuring that server-side requests are handled properly.

❓ Quiz Questions

1. What is Server-Side Rendering (SSR)?
a) Rendering React components only on the client side
✅ **b) Rendering React components on the server and sending static HTML to the client**
c) A process that only works with Next.js
d) A method for styling React components

---

2. What is one of the primary benefits of SSR?
a) It reduces app complexity
b) It allows for faster client-side interactions
✅ **c) It improves SEO by providing fully rendered content for search engines**
d) It eliminates the need for routing

---

3. What function from ReactDOMServer is used to render React components server-side?
a) `ReactDOM.hydrate()`
b) `ReactDOM.render()`
✅ **c) `ReactDOMServer.renderToString()`**
d) `ReactDOM.createRoot()`

---

4. What is the purpose of hydration in SSR?
a) To update the server-side HTML
✅ **b) To make the server-rendered content interactive on the client side**
c) To render the React components in the background
d) To preload assets before rendering
