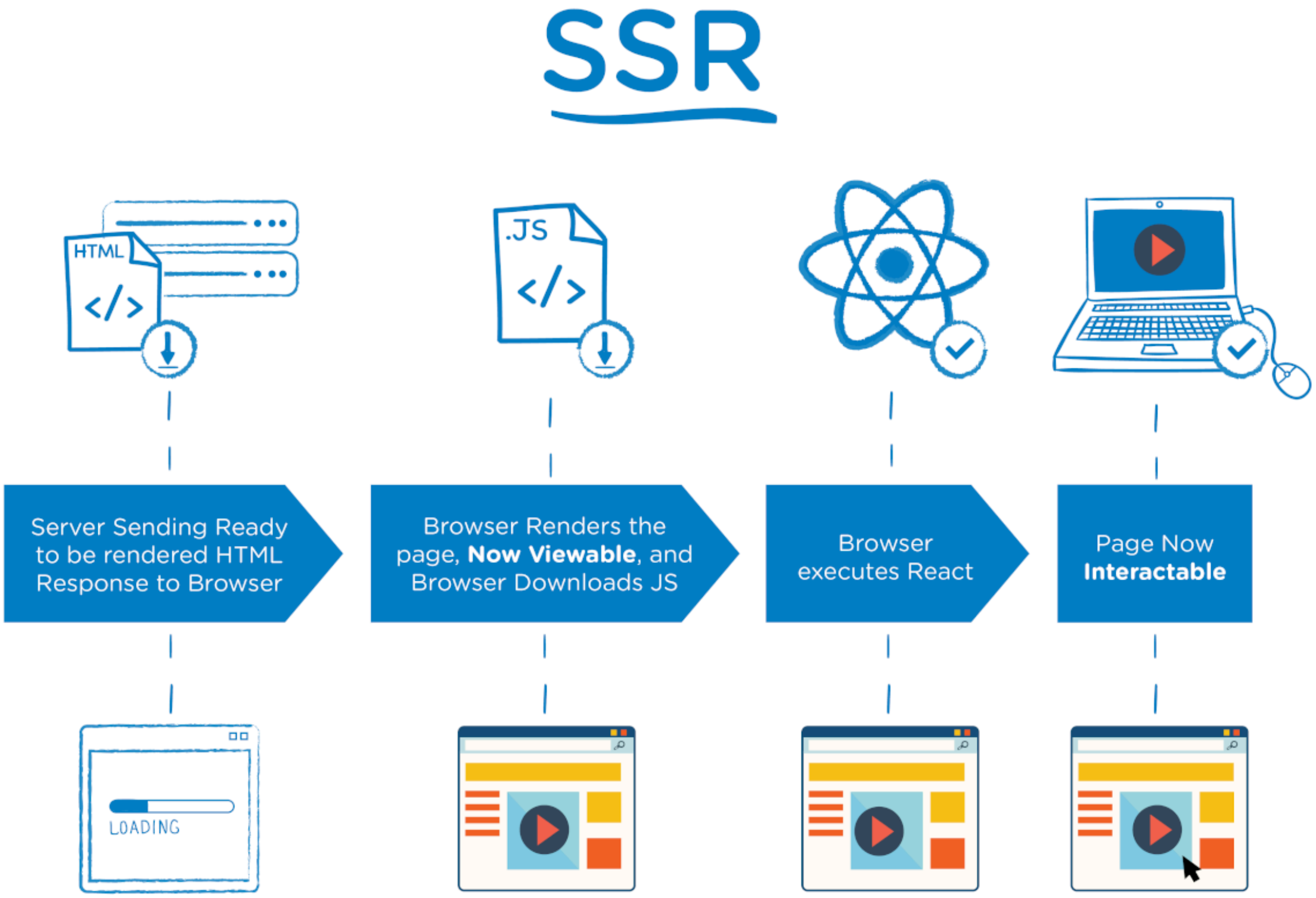
Day 23

Server-Side Rendering In React



@oluwakemi Oluwadahunsi

Introduction

Server-Side Rendering (SSR) is a technique used in React applications to generate the **initial HTML** of a page **on the server** rather than in the browser.

In a typical React application, the **initial HTML** is usually a bare-bones structure, and the content is **populated by JavaScript** after the page loads. This is called **Client-Side Rendering (CSR),** majorly what we have been doing since.

However, **in SSR**, the React **components are rendered on the server as HTML** and sent to the browser, where it is displayed immediately.

The server generates the complete HTML for the page and sends it to the client. This approach **improves the page load time** and is beneficial for **SEO** because search engines can easily crawl the fully-rendered HTML.

@oluwakemi Oluwadahunsi

How SSR Works in React

In a traditional React setup, the browser downloads a minimal HTML file, loads JavaScript, and then renders the content.

In contrast, with SSR, the following process occurs:

1. **Request:** The client (browser) makes a request for a web page.

2. **Server:** The server receives the request and runs the React code.

3. **Render:** React renders the components into HTML on the server.

4. **Response:** The server sends back a fully-rendered HTML page to the client.

5. **Hydration:** The React app on the client-side rehydrates, or attaches event listeners, to make the app interactive.

@oluwakemi Oluwadahunsi

Setting Up Server-Side Rendering in React

The easiest way to implement SSR in React is by using **Next.js**, a popular **React framework** that comes with SSR out-of-the-box. Next.js abstracts the complexities of SSR and makes it simple to use.

Let's walk through the steps of setting up SSR with Next.js.

Step 1: Setting Up a Next.js App

First, let’s create a new **Next.js** project.

In a new terminal, type:

```
npx create-next-app@latest my-nextjs-app
cd my-nextjs-app
npm run dev
```

This command creates a basic Next.js app and runs it in development mode.

@oluwakemi Oluwadahunsi

Step 2: Creating a Simple Page with SSR

Next.js automatically enables **SSR** for each page in the “**app**” directory.

Let’s create a simple page and fetch some data from an API using SSR.

The goals of this example:

We want to create a React component that:

- Renders on the server before it is sent to the browser.
- Fetches data from an API on the server.
- Displays the fetched data immediately when the user visits the page.

@oluwakemi Oluwadahunsi

```jsx
// Importing React to create our component
import React from 'react';

// This is the functional component we will render on the page
function Home({ data }) {
  return (
    <div>
      <h1>Server-Side Rendered Page</h1>
      <p>Data fetched from the server:</p>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
}

// This function runs on the server and fetches the data before rendering the page
export async function getServerSideProps() {
  // Fetch data from an external API (placeholder API in this example)
  const res = await fetch('https://jsonplaceholder.typicode.com/posts/1');
  const data = await res.json();

  // The fetched data is passed to the component as props
  return {
    props: { data }, // 'data' will be available as props inside the Home component
  };
}

// Export the Home component as the default export of the file
export default Home;
```

@oluwakemi Oluwadahunsi

1. In the “**Home**” component

```
function Home({ data }) {
  return (
    <div>
      <h1>Server-Side Rendered Page</h1>
      <p>Data fetched from the server:</p>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
}
```

- **{data}:** The data object comes as a prop, passed from the server. We will fetch this data from an API in the getServerSideProps function (explained below).
- **<h1>** and **<p>:** These are just basic HTML elements to show that the page is rendered.
- **<pre>{JSON.stringify(data, null, 2)}</pre>:** This line is responsible for displaying the fetched data as a **formatted JSON string** on the page. The JSON.stringify(data, null, 2) converts the JSON object into a readable format with indentation.

@oluwakemi Oluwadahunsi

2. **getServerSideProps** Function

```
export async function getServerSideProps() {
  // Fetch data from an external API (placeholder API in this example)
  const res = await fetch('https://jsonplaceholder.typicode.com/posts/1');
  const data = await res.json();

  // The fetched data is passed to the component as props
  return {
    props: { data }, // 'data' will be available as props inside the Home component
  };
}
```

- **What is getServerSideProps?** This function is special in Next.js because it runs **only** on the server.

Before the component renders on the page, this function executes and fetches the required data from an external source (in this case, an API). .

The server fetches the data before sending the fully rendered page to the **client (browser)**.

@oluwakemi Oluwadahunsi

- **Fetching Data:** The function makes an **API call** to the endpoint. This API returns sample data (a single blog post). Once the data is fetched using **await fetch()**, it is converted to JSON format using **await res.json()**.

- **Returning Props:** After fetching the data, it is returned inside the props object. This means the fetched data is passed as a prop to the **Home** component. In this example, the data object is passed and used inside the component to display the fetched API data.

3. **What Happens on the Server?**

- When a user requests the page, the **Next.js** server executes the **getServerSideProps** function.
- The server fetches the **data** from the API, composes the HTML by rendering the **Home** component with the fetched data, and sends the fully-rendered **HTML** to the **client**.

4. **What Happens in the Browser?**

- The **client (browser)** receives the **HTML** page with the **pre-rendered** content. This means that the **data** is already visible, and there is no need for the browser to wait for JavaScript to fetch the data.

- After the HTML is loaded, React kicks in, and the page is "**hydrated**," meaning React attaches its **event listeners** and becomes interactive.

Why Use Server-Side Rendering?

Here are the key reasons why developers opt for SSR in React apps:

- **Improved Performance:** SSR can deliver faster initial page loads because the HTML is pre-rendered on the server, and the user can see content immediately.
- **Better SEO:** Since the entire HTML content is rendered on the server, search engines can crawl and index the content more effectively, leading to better SEO performance. This is crucial for content-heavy websites like blogs, e-commerce platforms, etc.
- **Optimized for Slow Networks:** On slower network connections, SSR ensures that users receive a fully-rendered page even before the JavaScript is downloaded and executed.
- **Reduced Time-to-Interactive:** Since users see meaningful content faster, SSR decreases the time to interactive (TTI) metric, leading to a smoother user experience.

@oluwakemi Oluwadahunsi

Tabular Differences (CSR) vs. (SSR)

Features	Redux Thunk	Redux Saga
Loading Time	Initial load time is longer because the JavaScript bundle needs to be downloaded before rendering.	Faster initial load time since the page content is pre-rendered on the server.
SEO	SEO can be less effective because crawlers may not execute JavaScript properly.	Better for SEO as the content is available in HTML on the first request.
Rendering HTML	All rendering happens in the browser.	Initial rendering happens on the server, followed by rehydration on the client.
Dynanism	More dynamic, and real-time updates are efficient.	Better for static content or content that doesn't change often.

@o uwakemi Oluwadahunsi

I hope you found this material useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be helpful to someone 👌

@oluwakemi Oluwadahunsi

Hi There!

Thank you for reading through

Did you enjoy this knowledge?

💼 Follow my LinkedIn page for more work-life balancing and Coding tips.

🌐 LinkedIn: Oluwakemi Oluwadahunsi

kodemaven-portfolio.vercel.app