Client-Side Rendering

24 min

Think of a website that offers a highly interactive, seamless, and dynamic user experience like YouTube and AirBnB. Client-side rendering (CSR) plays a significant role in making such experiences possible.

In CSR, the server's response to the client includes all the necessary files to render the components on the client's browser and enable interactivity without making additional

Preview: Docs Loading link description

requests

to the server for the render or refreshing the page. The response contains a barebones HTML page, and accompanying JavaScript assembles the rest of the page's content.

CSR is an ideal solution for components with state or many user interactions like buttons and form fields. One popular implementation of CSR is the single-page application (SPA) pattern. In this pattern, the user stays on one page, with JavaScript seamlessly updating or replacing content. This approach doesn't reload the page entirely but dynamically fetches new external data from the server as needed — think of how your Gmail inbox updates with new emails, all without refreshing the page itself.

In Next.js, client-side can be implemented explicitly through **client components**, an opt-in feature that allows developers to designate specific components to be rendered on the client.

In a later lesson, we'll dive into the details of client components, but for now, know that you can define a client component as you would a regular React component with a 'use client' directive. This directive specifies that the component and its children components should be rendered on the client side.

The following is an example of a client-side rendered component:

```
'use client'
import React, { useState } from 'react'

export default function Page() {
  const [toggle, setToggle] = useState<boolean>(false);
  return (
    <div onClick={() => setToggle(!toggle)}>
      {toggle ? 'True': 'False'}
    </div>
  )
}
```

Instructions

1. Checkpoint 1 Passed

1.

Let’s create a client-side component that reacts to a toggle value, allowing users to switch between “Day mode” and “Night mode.”

Start by including the 'use client' directive at the top of your file, indicating the component should render on the client side.

Hint

'use client' should be at the very top of your file.

2. Checkpoint 2 Passed

2.

Define a component named ThemeSwitch. The component should have a state variable to track whether the theme is day or night.

Hint

The component should go *under* the 'use client' directive.

3. Checkpoint 3 Passed

3.

In the component, return a button to let users toggle the theme. The button text should reflect the mode, such as “Night Mode” or “Day Mode” when clicked. Use the provided <BackgroundManager> component to change the app’s background color in accordance to the button.

Then, default export the component.

Hint

Define an event handler as necessary for the toggle event. The button text can be determined using a ternary operator or a separate variable tracking the boolean value of the state.

Ensure that the button has “Night” or “Day” labels for toggling.

4. Checkpoint 4 Passed

4.

While working through our Next.js lessons, you can start a local development server to see real-time results of your coding.

Enter the command npm run dev into the terminal. Once you see a console message indicating “Ready,” click the Refresh Browser button in the workspace browser bar.

http://localhost:4001/

This will allow you to see the rendered output of your code, reflecting any changes you've made. Once started, you can keep the server running for continuous, real-time updates.

To stop the server, use Ctrl+C in the terminal.

Note that for the first start, the server may require some time to fully initialize.

Hint

Enter the command inside the terminal window.

The window can be refreshed when it displays "Ready."

page.tsx

```
'use client'
import { useState } from 'react'
import BackgroundManager from '../components/BackgroundManager'


export default function ThemeSwitch() {
  const [light, setLight] = useState<boolean>(true);


  const toggleTheme = () => {
    setLight(!light);
  };


  return (
    <>
    <BackgroundManager light={light}/>
      <button onClick={toggleTheme}>
        Switch to {light ? "Night" : "Day"} Mode
      </button>
    </>
  );
}
```