In the Next.js **Pages Router**, Client-Side Rendering (CSR) works slightly differently than in plain React. While plain React starts with a completely blank HTML file, Next.js still performs **Static Optimization** by default.

This means Next.js serves a pre-rendered HTML "shell" (headers, layout, loading states) which then fetches data and becomes interactive on the client.

---

1. The Two Ways to do CSR in Pages Router

Pattern A: useEffect Data Fetching (The most common)

The page is pre-rendered at build time as a static shell. Once it hits the browser, React "hydrates" it and triggers a fetch request.

```
// pages/user-profile.js
import { useState, useEffect } from 'react'

export default function UserProfile() {
  const [data, setData] = useState(null)

  useEffect(() => {
    // This fetch only happens in the browser
    fetch('/api/user')
      .then((res) => res.json())
      .then((user) => setData(user))
  }, [])

  if (!data) return <p>Loading...</p> // This part is pre-rendered to HTML

  return <div><h1>{data.name}</h1></div>
}
```

Pattern B: Disabling SSR with next/dynamic

Sometimes you have a component that **cannot** run on a server at all (e.g., a chart library that uses window or document). You can force it to be purely CSR.

```
import dynamic from 'next/dynamic'

// This component will ONLY be rendered in the browser
const MapComponent = dynamic(() => import('../components/Map'), {
  ssr: false,
})

export default function ContactPage() {
  return (
    <div>
      <h1>Our Location</h1>
      <MapComponent />
    </div>
  )
}
```

---

2. The CSR Lifecycle in Next.js

1. **Request:** User hits the URL.
2. **Static Shell:** Next.js immediately serves the pre-rendered HTML (the "Loading..." state).
3. **Hydration:** The browser downloads the JS bundle. React "attaches" to the existing HTML.
4. **Client Fetch:** The useEffect triggers.
5. **Final UI:** The state updates, and the actual content appears.

---

3. Comparison: Plain React vs. Next.js Pages Router (CSR)

Feature	Plain React (Vite/CRA)	Next.js Pages Router (CSR)
Initial HTML	Empty <body>	Contains a "Loading" or Shell UI
SEO	Almost zero initially	Better (tags/meta can be pre-rendered)
Performance	User sees white screen until JS loads	User sees a loading state immediately
Routing	Handled by react-router-dom	Built-in file-based routing

---

4. Interview Strategy: The "Hydration Mismatch" Warning

In a system design interview, if you choose CSR in Next.js, you must mention **Hydration Mismatch**.

Because Next.js pre-renders the HTML on the server, the first render in the browser **must match** that HTML perfectly. If you do something like this:

```
// ❌ DANGEROUS: Will cause a hydration error
function Clock() {
  return <div>{new Date().toLocaleTimeString()}</div>
}
```

The server will render one time, and the client will render another time (a few milliseconds later). Next.js will throw an error because the HTML doesn't match.

The Fix: Wrap the logic in a useEffect or check if window is defined to ensure the time is only calculated on the client.

5. When to use CSR in Next.js?

- **Search/Filter Results:** When a user changes a filter, you don't want a full page reload; you fetch the new data on the client.
- **Highly Interactive Dashboards:** Pages with many "small" data updates (stock tickers, chat).
- **Private User Data:** Anything behind an auth wall where SEO is irrelevant.

Would you like to move on to Server-Side Rendering (SSR) using getServerSideProps to see how it solves the "Loading" state problem?