In a plain React setup (created via **Vite** or **Create React App**), Client-Side Rendering (CSR) is the default and only behavior. There is no server-side logic involved in generating the UI; the "server" in this case is just a file server (like an S3 bucket or Nginx) that hands over static files.

---

1. The Anatomy of a CSR App

When a user visits a CSR site, the process follows a specific sequence. The browser starts with nothing and builds everything locally.

Step 1: The "Empty Shell" HTML

The server sends back a minimal HTML file. If you inspect the source code of a CSR React app, you will see almost nothing inside the <body>.

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="root"></div> <script type="module" src="/src/main.jsx"></script> </body>
</html>
```

Step 2: The JS Bundle Download

The browser sees the <script> tag and begins downloading the entire React library, your component code, and any third-party dependencies.

Step 3: Mounting and Data Fetching

Once the JS is parsed, React "mounts" to the #root div. Usually, this is when your useEffect hooks trigger to fetch data from an API.

---

2. Code Example: Plain React CSR

Here is how a typical data-driven component looks in a plain React application. Note that the data fetching happens **after** the component has already appeared on the screen.

```jsx
import { useState, useEffect } from 'react';

function UserProfile() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // This runs only in the browser AFTER the initial empty render
    fetch('https://api.example.com/user/1')
      .then((res) => res.json())
      .then((data) => {
        setUser(data);
        setLoading(false);
      });
  }, []);

  if (loading) return <div>Loading...</div>; // User sees this first

  return (
    <div>
      <h1>{user.name}</h1>
      <p>{user.bio}</p>
    </div>
  );
}
```

---

3. Trade-offs in System Design

The Performance Gap

In CSR, the **FCP (First Contentful Paint)** and **TTI (Time to Interactive)** are often identical.

- **The Problem:** The user sees a white screen while the JS downloads, and then a "Loading..." spinner while the data fetches.
- **The "Double Jump":** First the UI shell loads, then the data "pops" in. This can cause layout shifts if not handled with skeletons.

SEO Impact

Since the initial HTML is empty, search engine crawlers (like Googlebot) have to execute the JavaScript to see the content.

- **The Risk:** While Google is good at this, it's not perfect. Other crawlers (Twitter/OpenGraph for link previews) often fail to see anything, leading to "empty" social media cards.

Hosting & Cost

- **The Win:** CSR is incredibly cheap to host. Since it's just static files (index.html, app.js, style.css), you can host it on a **CDN** (Content Delivery Network) like Cloudflare Pages or Netlify for nearly $0. There is no "running server" to maintain.

---

4. When to choose Plain React (CSR) over Next.js?

In an interview, you might be asked: *"If Next.js is so good, why would you ever use plain React?"*

- **Admin Dashboards/SaaS Platforms:** SEO doesn't matter behind a login wall. The rich interactivity of a SPA (Single Page Application) is more important.
- **Internal Tools:** Speed of development and low hosting complexity are the priorities.
- **Offline Support:** CSR apps are much easier to turn into **PWAs (Progressive Web Apps)** that work offline.

Summary for Interviews

Feature	CSR (Plain React)
Initial HTML	Empty
Data Fetching	Happens in the browser (useEffect)
Server Load	Zero (just serves static files)
Best For	Logged-in dashboards, private tools, PWAs