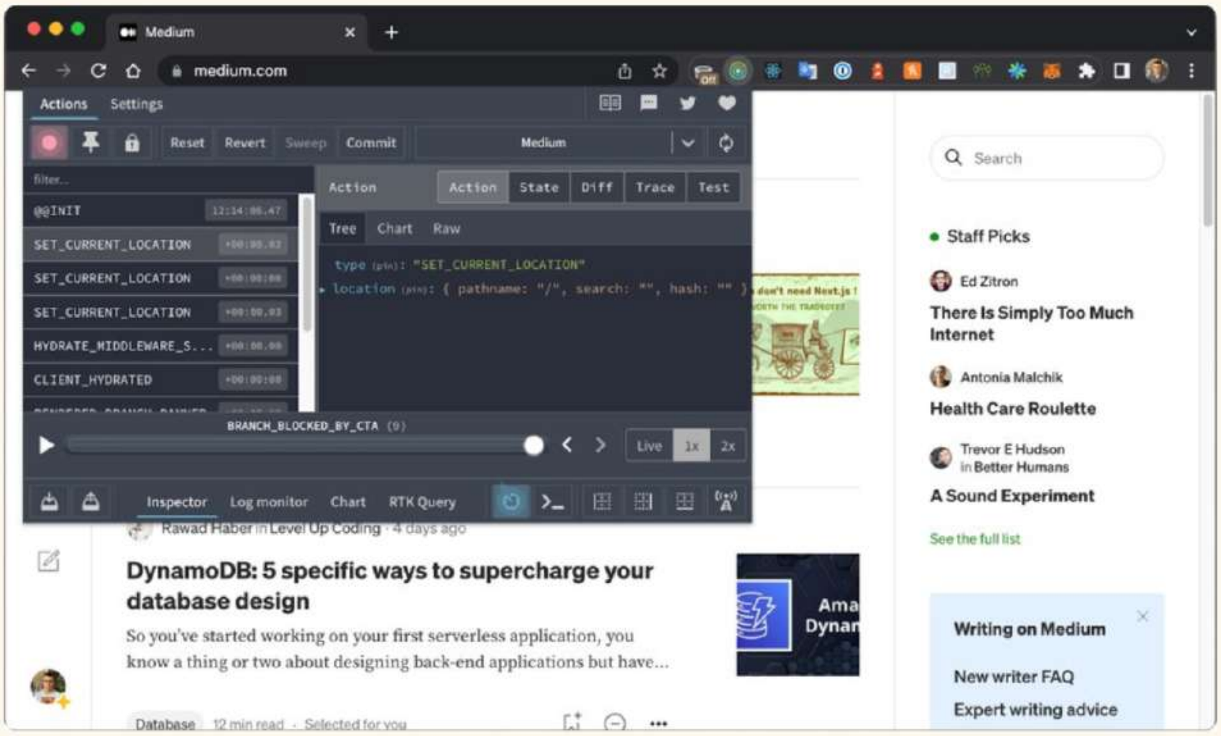
25.3.3 Source Maps

Source maps help map compiled code back to the original source code, making debugging easier in production.

Configuration:

- Adddevtool: "source-map"inWebpackforsourcemaps.

Illustration: Image showing the React Developer Tools interface highlighting state inspection on a medium page.



```
@@INIT                      12:14:06.47
SET_CURRENT_LOCATION
SET_CURRENT_LOCATION
SET_CURRENT_LOCATION
HYDRATE_MIDDLEWARE_S...
CLIENT_HYDRATED
BRANCH_BLOCKED_BY_CTA (9)
```

```
type (pin): "SET_CURRENT_LOCATION"
▸ location (pin): { pathname: "/", search: "", hash: "" }
```

Ed Zitron
There Is Simply Too Much Internet

Antonia Malchik
Health Care Roulette

Trevor E Hudson in Better Humans
A Sound Experiment

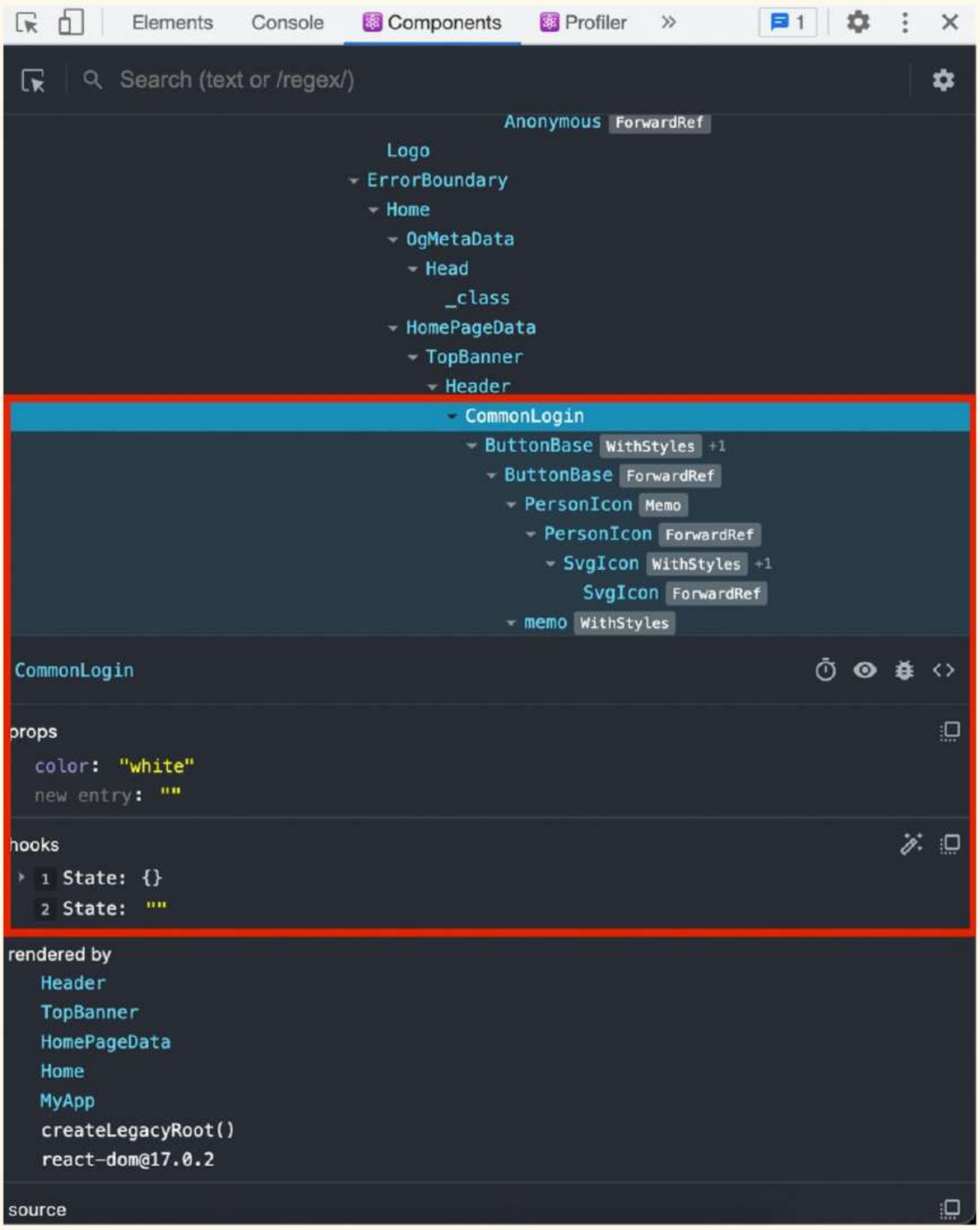
Rawad Haber in Level Up Coding · 4 days ago

DynamoDB: 5 specific ways to supercharge your database design

So you've started working on your first serverless application, you know a thing or two about designing back-end applications but have...

Database · 12 min read · Selected for you

Illustration: React DevTools interface with component tree, state, and props highlighted.



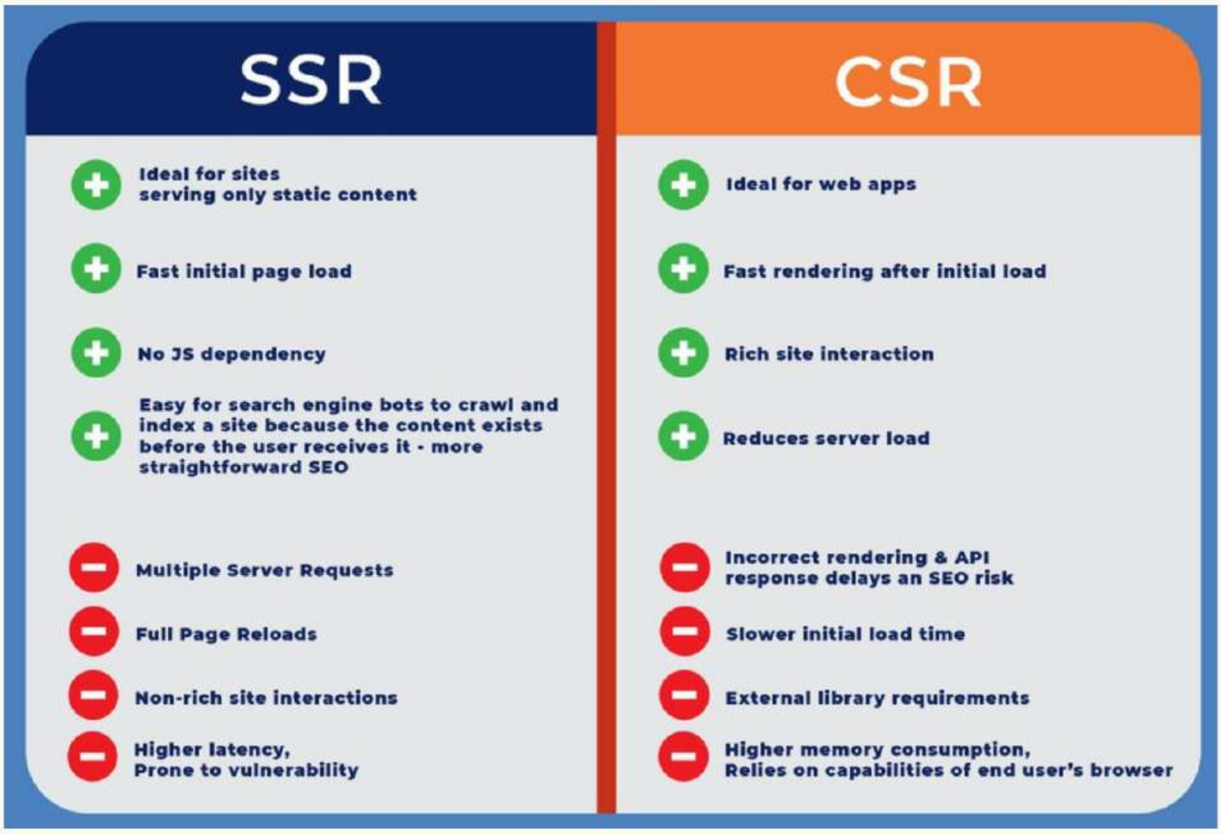
```
Anonymous ForwardRef
Logo
▾ ErrorBoundary
  ▾ Home
    ▾ OgMetaData
      ▾ Head
          _class
    ▾ HomePageData
      ▾ TopBanner
        ▾ Header
          ▾ CommonLogin
            ▾ ButtonBase WithStyles +1
              ▾ ButtonBase ForwardRef
                ▾ PersonIcon Memo
                  ▾ PersonIcon ForwardRef
                    ▾ SvgIcon WithStyles +1
                        SvgIcon ForwardRef
                ▾ memo WithStyles
```

```
CommonLogin

props
  color: "white"
  new entry: ""

hooks
 ▸ 1 State: {}
   2 State: ""

rendered by
  Header
  TopBanner
  HomePageData
  Home
  MyApp
  createLegacyRoot()
  react-dom@17.0.2

source
```

14.2 What is Server-Side Rendering?

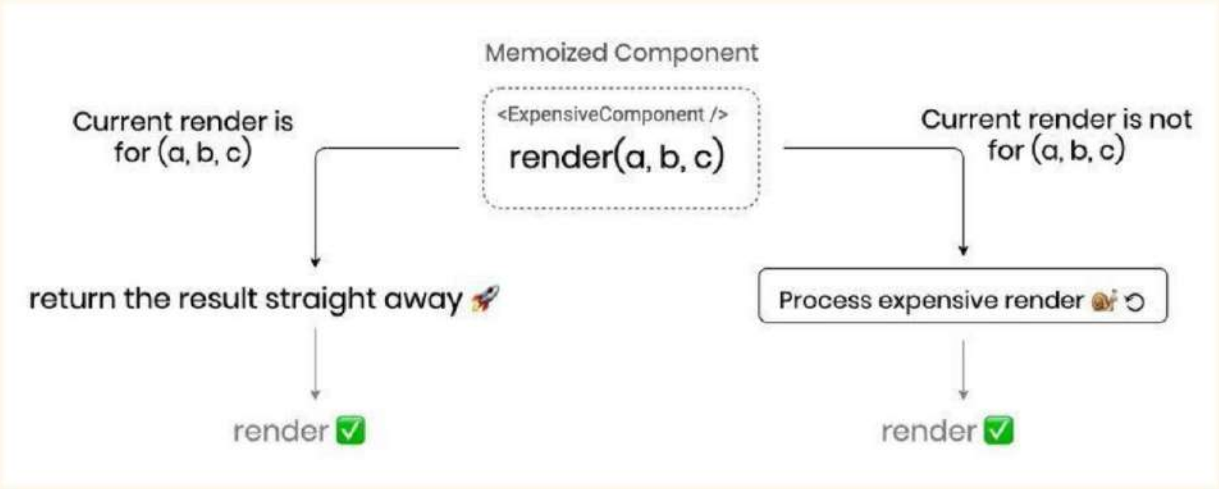
SSR involves rendering HTML on the server and sending it to the browser. Once delivered, JavaScript hydrates the page, enabling React's interactivity.

Illustration: Comparison of SSR vs. CSR (Client-Side Rendering) showing workflow differences.



SSR	CSR
+ Ideal for sites serving only static content	+ Ideal for web apps
+ Fast initial page load	+ Fast rendering after initial load
+ No JS dependency	+ Rich site interaction
+ Easy for search engine bots to crawl and index a site because the content exists before the user receives it - more straightforward SEO	+ Reduces server load
− Multiple Server Requests	− Incorrect rendering & API response delays an SEO risk
− Full Page Reloads	− Slower initial load time
− Non-rich site interactions	− External library requirements
− Higher latency, Prone to vulnerability	− Higher memory consumption, Relies on capabilities of end user's browser

Illustration: React performance optimization flowchart showing rendering, state management, and memoization.


Memoized Component
<ExpensiveComponent />
render(a, b, c)
Current render is for (a, b, c)
Current render is not for (a, b, c)
return the result straight away 🚀
Process expensive render 🐌 ↻
render ✅
render ✅

13.3 Measuring Performance

React provides tools and methods for profiling and measuring performance:

Example: Using React Profiler

jsx

Copy code

```
import React, { Profiler } from "react";


function App() {
  const onRenderCallback = (id, phase, actualDuration) => {
    console.log(`Component: ${id}, Phase: ${phase}, Time:
${actualDuration}ms`);
```

12.8 Real-Life Scenarios and Case Studies

Case Study: E-Commerce Routing

1. **Scenario**: An e-commerce app needs routes for:
   - /:Home
   - /product/:id:Productdetails
   - /cart:Shoppingcart
2. **Implementation**:
   - UseuseParamsfordynamicproductIDs.
   - Usenestedroutesforcartoperations.

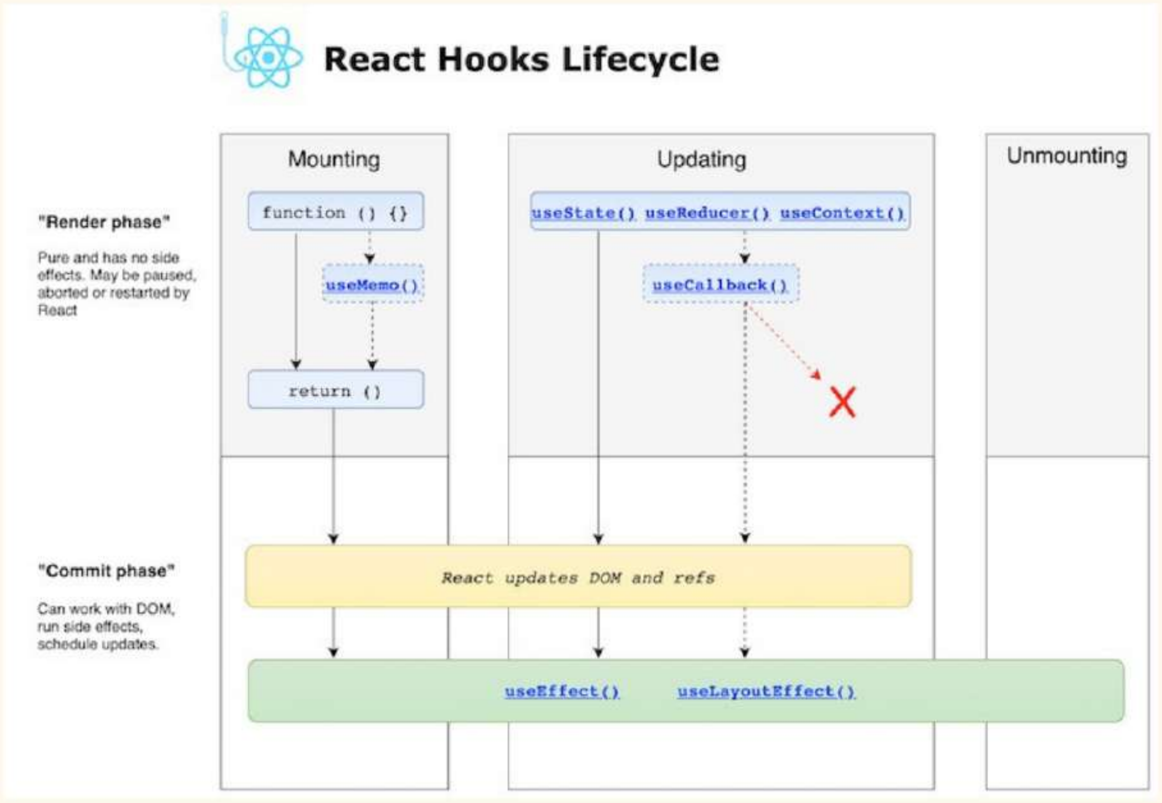
12.9 Cheat Sheet

Feature	Description	Example
`<BrowserRouter>`	Wraps the app for routing support.	`<BrowserRouter>`
`<Routes>`	Contains route definitions.	`<Routes>`
`<Route>`	Maps path to component.	`<Route path="/" element={<Home/>}/>`
`useNavigate`	Navigate programmatically.	`navigate("/path")`
`useParams`	Access route parameters.	`const { id } = useParams();`

```
      <p>Count: {count}</p>

      <button onClick={() => setCount(count + 1)}>Increment</button>

    </div>

  );

}


export default App;
```

Output: Logs messages during mounting, updating, and unmounting.

Illustration: React Hooks lifecycle


React Hooks Lifecycle
Mounting
Updating
Unmounting
"Render phase"
Pure and has no side effects. May be paused, aborted or restarted by React
function () {}
useMemo()
return ()
useState() useReducer() useContext()
useCallback()
"Commit phase"
Can work with DOM, run side effects, schedule updates.
React updates DOM and refs
useEffect()
useLayoutEffect()

```
    <CartContext.Provider value={{ cart, addToCart }}>

      <ProductList />

      <CartSummary />

    </CartContext.Provider>

  );

}


function ProductList() {

  const { addToCart } = useContext(CartContext);


  return (

    <div>

      <button onClick={() => addToCart("Product A")}>Add Product
A</button>

      <button onClick={() => addToCart("Product B")}>Add Product
B</button>
    </div>


  );

}


function CartSummary() {

  const { cart } = useContext(CartContext);
```