React 19 Cheat Sheet by Kent C. Dodds

EpicReact.dev

React Server Components

Server-rendered components that execute at build time or per request.

```
// dashboard.tsx
import * as db from './db.ts'
import { createOrg } from './org-actions.ts'
import { OrgPicker } from './org-picker.tsx'

// look! it's async!
export async function Dashboard() {
  // look! await!
  const orgs = await db.getOrgs()
  return (
    {/* this is all server-only code */}
    <div>
      <h1>Dashboard</h1>
      {/* OrgPicker is a client-side component and we can
          pass it server stuff! */}
      <OrgPicker orgs={orgs} onCreateOrg={createOrg} />
    </div>
  )
}
```

Actions

Async functions that handle form submission, error states, and optimistic updates automatically.

```
<form action={createOrg} />
```

`useActionState`

Manages form state, providing degraded experiences when JavaScript is unavailable.

```
const [error, submitAction, isPending] =
useActionState(async () => {...}, null);
```

`useFormStatus`

Access the status of a parent form without prop drilling.

```
const { pending, data, method, action } = useFormStatus();
```

`useOptimistic`

Show optimistic state while async requests are in progress.

```
const [optimisticName, setOptimisticName] = useOptimistic(name);
```

Async Script Support

Render async scripts anywhere in your component tree, with automatic deduplication.

```
<script async src="https://example.com/script.js" />
```

Improved Third-Party Script Compatibility

Unexpected tags in <head> and <body> are skipped during hydration, avoiding mismatch errors.

'use client'

Marks code that can be referenced by the server component and can use client-side React features.

```
// org-picker.tsx
'use client'

// regular component like we've been building for years
export function OrgPicker({ orgs, onCreateOrg }) {
  // The `orgs` and `onCreateOrg` props can come from a server
component!
  // This component can use useState, useActionState, useEffect,
useRef, etc etc etc...

  return <div />
}
```

'use server'

Marks server-side functions callable from client-side code.

```
// org-actions.ts
'use server'

// this function can be called by the client!
export async function createOrg(prevResult, formData) {
  // create an org with the db
  // return a handy result
}
```

Document Metadata Support

Automatically hoists `<title>`, `<meta>`, and `<link>` tags to the `<head>`.

```
<title>My Blog</title>
<meta name="author" content="Kent" />
```

Stylesheets with Precedence

Support for inserting stylesheets with precedence in concurrent rendering environments.

```
<link rel="stylesheet" href="foo.css" precedence="default" />

{/* this will get placed "higher" in the document */}
<link rel="stylesheet" href="bar.css" precedence="high" />
```

Resource Preloading APIs

Preload resources like fonts, scripts, and styles to optimize performance.

```
preload('https://example.com/font.woff', { as: 'font' })
preconnect('https://example.com')
```

Custom Element Support

React now fully supports custom elements and handles properties/attributes consistently.

```
<custom-element prop1="value" />
```

Better Error Reporting

Automatically de-duplicates errors, and introduces `onCaughtError` and `onUncaughtError` handlers for root components.

```
createRoot(container, {
  onCaughtError: (error, errorInfo) => {
    console.error(
      'Caught error',
      error,
      errorInfo.componentStack
    )
  },
  onUncaughtError: (error, errorInfo) => {
    console.error(
      'Uncaught error',
      error,
      errorInfo.componentStack
    )
  },
})
```

`use`

Reads resources like promises or context during render, allowing for conditional use.

```
const comments = use(commentsPromise)
const theme = use(ThemeContext)
```

Ref Callback Cleanup

Ref callbacks can now return a cleanup function.

```
<input ref={(ref) => () => console.log('cleanup')} />
```

Streamlined Context API

Use `<Context>` directly instead of `<Context.Provider>`.

```
<LanguageContext value="pl-PL">{children}</LanguageContext>
```

`useDeferredValue` Initial Value

The `useDeferredValue` hook now supports an initial value.

```
const deferredValue = useDeferredValue(value, 'initial');
```

Hydration Error Diffs

Improved error logging for hydration errors, providing a detailed diff when mismatches occur.

```
Uncaught Error: Hydration failed
  <App>
    <span>
+     Client
-     Server
```

`ref` as a Prop

Pass refs directly as props in function components.

```
<MyInput ref={inputRef} />
```