Layout vs Template in Next.js

In Next.js 15, differences between layout and template files, and how to implement them effectively in your Next.js applications.

In Next.js 15, the introduction of `layout.js` and `template.js` files offers developers enhanced control over component rendering and state management during navigation. Understanding the distinctions between these two is crucial for optimizing your application's performance and user experience.

🧱 Layout vs. Template in Next.js 15

📌 Layout (`layout.js`)

- **State Persistence**: Maintains state across route transitions.
- **Reusability**: Ideal for consistent UI elements like headers, footers, and navigation bars.
- **Rendering Behavior**: Does not re-render when navigating between sibling routes.
- **Use Cases**:
  - Persistent UI elements across multiple pages.
  - Components that should retain state, such as sidebars or authentication wrappers.
  - Performance optimization by avoiding unnecessary re-renders.

🧩 Template (`template.js`)

- **State Reset**: Resets state on every route transition.
- **Rendering Behavior**: Re-renders every time the route changes.

- **Use Cases**:
  - Pages where UI should reset on navigation, such as interactive forms or animated views.
  - Ensuring fresh UI state for each page load.
  - Triggering lifecycle methods like `useEffect` on every navigation.

🔍 Key Differences

Feature	Layout (`layout.js`)	Template (`template.js`)
State Persistence	Yes	No
Re-rendering	No (persists across sibling routes)	Yes (re-renders on every navigation)
Ideal for	Persistent UI elements	Isolated UI components requiring fresh state
Use Cases	Navigation bars, sidebars, authentication wrappers	Forms, animations, page-specific content

---

🛠️ Implementation Example

Layout Example

```tsx
// app/layout.js
export default function Layout({ children }) {
  return (
    <div>
      <header>Header Content</header>
      <nav>Navigation Links</nav>
      <main>{children}</main>
    </div>
  );
}
```

Template Example

```tsx
// app/template.js
export default function Template({ children }) {
  return (
    <div>
      <header>Page Header</header>
      <main>{children}</main>
      <footer>Page Footer</footer>
    </div>
  );
}
```

✅ When to Use Each

- **Use `layout.js`:**
  - When you need a consistent structure across multiple pages.
  - To maintain state and avoid unnecessary re-renders.
  - For components like navigation bars or authentication wrappers.
- **Use `template.js`:**
  - When each page should load with a fresh UI state.
  - For pages requiring isolated state or lifecycle methods on navigation.
  - To implement animations or transitions that require re-rendering.

By strategically utilizing `layout.js` and `template.js`, you can optimize your Next.js 15 application for both performance and user experience. Ensure that persistent UI elements are placed in layouts, while components requiring fresh state or re-initialization are handled within templates.

Sources