Fast Refresh

▼ **Examples**

- Fast Refresh Demo

Fast Refresh is a Next.js feature that gives you instantaneous feedback on edits made to your React components. Fast Refresh is enabled by default in all Next.js applications on **9.4 or newer**. With Next.js Fast Refresh enabled, most edits should be visible within a second, **without losing component state**.

How It Works

- If you edit a file that **only exports React component(s)**, Fast Refresh will update the code only for that file, and re-render your component. You can edit anything in that file, including styles, rendering logic, event handlers, or effects.
- If you edit a file with exports that *aren't* React components, Fast Refresh will re-run both that file, and the other files importing it. So if both `Button.js` and `Modal.js` import `theme.js`, editing `theme.js` will update both components.
- Finally, if you **edit a file** that's **imported by files outside of the React tree**, Fast Refresh **will fall back to doing a full reload**. You might have a file which renders a React component but also exports a value that is imported by a **non-React component**. For example, maybe your component also exports a constant, and a non-React utility file imports it. In that case, consider migrating the constant to a separate file and importing it into both files. This will re-enable Fast Refresh to work. Other cases can usually be solved in a similar way.

Error Resilience

Syntax Errors

If you make a syntax error during development, you can fix it and save the file again. The error will disappear automatically, so you won't need to reload the app. **You will not lose component state**.

Runtime Errors

If you make a mistake that leads to a runtime error inside your component, you'll be greeted with a contextual overlay. Fixing the error will automatically dismiss the overlay, without reloading the app.

Component state will be retained if the error did not occur during rendering. If the error did occur during rendering, React will remount your application using the updated code.

If you have error boundaries in your app (which is a good idea for graceful failures in production), they will retry rendering on the next edit after a rendering error. This means having an error boundary can prevent you from always getting reset to the root app state. However, keep in mind that error boundaries shouldn't be *too* granular. They are used by React in production, and should always be designed intentionally.

Limitations

Fast Refresh tries to preserve local React state in the component you're editing, but only if it's safe to do so. Here's a few reasons why you might see local state being reset on every edit to a file:

- Local state is not preserved for class components (only function components and Hooks preserve state).
- The file you're editing might have *other* exports in addition to a React component.
- Sometimes, a file would export the result of calling a higher-order component like `HOC(WrappedComponent)`. If the returned component is a class, its state will be reset.

- Anonymous arrow functions like `export default () => <div />;` cause Fast Refresh to not preserve local component state. For large codebases you can use our `name-default-component` codemod.

As more of your codebase moves to function components and Hooks, you can expect state to be preserved in more cases.

Tips

- Fast Refresh preserves React local state in function components (and Hooks) by default.
- Sometimes you might want to *force* the state to be reset, and a component to be remounted. For example, this can be handy if you're tweaking an animation that only happens on mount. To do this, you can add `// @refresh reset` anywhere in the file you're editing. This directive is local to the file, and instructs Fast Refresh to remount components defined in that file on every edit.
- You can put `console.log` or `debugger;` into the components you edit during development.

Fast Refresh and Hooks

When possible, Fast Refresh attempts to preserve the state of your component between edits. In particular, `useState` and `useRef` preserve their previous values as long as you don't change their arguments or the order of the Hook calls.

Hooks with dependencies—such as `useEffect`, `useMemo`, and `useCallback`—will *always* update during Fast Refresh. Their list of dependencies will be ignored while Fast Refresh is happening.

For example, when you edit `useMemo(() => x * 2, [x])` to `useMemo(() => x * 10, [x])`, it will re-run even though `x` (the dependency) has not changed. If React didn't do that, your edit wouldn't reflect on the screen!

Sometimes, this can lead to unexpected results. For example, even a `useEffect` with an empty array of dependencies would still re-run once during Fast Refresh.

However, writing code resilient to occasional re-running of `useEffect` is a good practice even without Fast Refresh. It will make it easier for you to introduce new dependencies to it later on and it's enforced by React Strict Mode, which we highly recommend enabling.