What Is the useActionState Hook, and How Does It Work?

freecodecamp.org/learn/front-end-development-libraries-v9/lecture-working-with-forms-in-react/what-is-the-useactionstate-hook-and-how-does-it-work

React 19 came with two notable new features called server components and server actions.

From that version onwards, server components became the default in frameworks like Next.js that readily support them.

Server actions on the other hand, are functions that run on the server to allow form handling right on the server without the need for API endpoints.

A server action looks like this:

```
"use server";

async function submitForm(formData) {
  const name = formData.get("name");
  return { message: `Hello, ${name}!` };
}
```

This server action extracts a `name` field from a form and returns a string greeting that name.

To simplify state management for server actions and remove the need for client-side JavaScript for simple forms, the React team introduced the `useActionState` hook in version 19.

Let's take a closer look at this hook and see how it works.

The React documentation describes the `useActionState` hook as a hook that "allows you to update state based on the result of a form action."

But this doesn't mean that you can only use the `useActionState` hook with forms. You can also use it to manage button clicks and other events, as long as you have an action in place.

And keep in mind that, since `useActionState` is a hook, you cannot use it inside a server component.

Here's the basic syntax of the `useActionState` hook:

```
const [state, action, isPending] = useActionState(actionFunction, initialState,
permalink);
```

- `state` is the current state the action returns.
- `action` is the function that triggers the server action.
- `isPending` is a boolean that indicates whether the action is currently running or not.
- The `actionFunction` parameter is the server action itself.
- `initialState` is the parameter that represents the starting point for the state before the action runs.
- `permalink` is an optional string that contains the unique page URL the form modifies.

To use the `useActionState` hook, make sure you have an action in place first. Let's use the action from the previous example for this, with a bit of a twist:

```
"use server";

export async function submitForm(_, formData) {
  const name = formData.get("name");

  const hour = new Date().getHours();
  let greeting;

  if (hour < 12) {
    greeting = "Good morning";
  } else if (hour < 18) {
    greeting = "Good afternoon";
  } else {
    greeting = "Good evening";
  }

  return { message: `${greeting}, ${name}` };
}
```

In your component, you then need to import the `useActionState` hook and call it at the top level of the component body (before the return statement) just like other hooks. You should also import the action:

```
"use client";

// Import the useActionState hook
import { useActionState } from "react";

// Import the submitForm action
import { submitForm } from "./actions/submitForm";

const Greeter = () => {

 // Initialize the hook
 const [state, submit, isPending] = useActionState(submitForm, {
   message: "",
 });

  return (
    <div className="flex flex-col items-center justify-center min-h-screen bg-gray-100
p-6">
      {/* Rest of component */}
    </div>
  );
};

export default Greeter;
```

Here's what the full code looks like with a bit of styling:

```
"use client";

import { useActionState } from "react";
import { submitForm } from "./actions/submitForm";

const Greeter = () => {
  const [state, submit, isPending] = useActionState(submitForm, {
    message: "",
  });

  return (
    <div className="flex flex-col items-center justify-center min-h-screen bg-gray-100
p-6">
      <form
        action={submit}
        className="bg-white p-6 rounded-2xl shadow-md w-full max-w-md"
      >
        <h2 className="text-2xl text-center font-semibold text-gray-700 mb-4">
          Greet Someone
        </h2>

        <input
          type="text"
          name="name"
          placeholder="Enter your name"
          required
          className="w-full p-3 border border-gray-300 rounded-lg focus:outline-none
focus:ring-2 focus:ring-green-400"
        />

        <button
          type="submit"
          disabled={isPending}
          className="w-full mt-4 p-3 bg-green-500 text-white font-semibold rounded-lg
hover:bg-green-600 disabled:bg-gray-400 transition-all"
        >
          {isPending ? "Greeting..." : "Greet"}
        </button>

        {state.message && (
          <p className="mt-4 text-green-600 text-center font-medium">
            {state.message}
          </p>
        )}
      </form>
    </div>
  );
};

export default Greeter;
```

In the browser, you would see your form button change from `Greet` to `Greeting...` while the action `isPending` - and the greeting would show `Good morning, {name}`, `Good afternoon, {name}`, or `Good evening, {name}`, depending on what time of day the form was submitted.

Remember how we mentioned that you can also use the `useActionState` hook outside of a form?

In this example, we'll fetch five users from JSONPlaceholder with a button click:

```
"use server";

export async function getUsers() {
  const res = await fetch(
    "https://jsonplaceholder.typicode.com/users?_start=0&_limit=5/"
  );
  return await res.json();
}
```

Here's the styled UI:

```
"use client";

import { useActionState } from "react";
import { getUsers } from "./actions/getUsers";

export default function FetchUsers() {
  const [users, fetchAction, isPending] = useActionState(getUsers, []);

  return (
    <div className="p-6 max-w-lg mx-auto">
      <button
        onClick={fetchAction}
        disabled={isPending}
        className="px-4 py-2 cursor-pointer bg-green-500 text-white rounded-lg
hover:bg-green-600 disabled:bg-gray-400 font-bold"
      >
        {isPending ? "Fetching Users..." : "Fetch Users"}
      </button>

      <ul className="mt-4 space-y-2">
        {users.map((user) => (
          <li key={user.id} className="p-3 bg-gray-100 rounded-lg">
            <p className="font-semibold">{user.name}</p>
            <p className="text-sm text-gray-600">{user.email}</p>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

In the browser, you would see that the button text is never updated to `Fetching Users...` after it's clicked.

This happens because React treats data fetching and rendering as a higher priority than the `isPending` state, which blocks `isPending` in the process and throws an error.

To fix this issue, you need to wrap the action in `startTransition`:

```
"use client";

// import startTransition from React
import { useActionState, startTransition } from "react";
import { getUsers } from "./actions/getUsers";

export default function FetchUsers() {
  const [users, fetchAction, isPending] = useActionState(getUsers, []);

  return (
    <div className="p-6 max-w-lg mx-auto">
      <button
        {/* wrap fetchAction in startTransition */}
        onClick={() => startTransition(() => fetchAction())}
        disabled={isPending}
        className="px-4 py-2 bg-green-500 font-bold cursor-pointer text-white rounded-
lg hover:bg-green-600 disabled:bg-gray-400"
      >
        {isPending ? 'Fetching Users...' : 'Fetch Users'}
      </button>

      <ul className="mt-4 space-y-2">
        {users.map((user) => (
          <li key={user.id} className="p-3 bg-gray-100 rounded-lg">
            <p className="font-semibold">{user.name}</p>
            <p className="text-sm text-gray-600">{user.email}</p>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

If you're wondering what `startTransition` is, it's a function that tells React that a state update is of low-priority and can be interrupted. This keeps the UI responsive while handling asynchronous updates like server actions.

That's how to use the `useActionState` hook inside and outside a form.

Questions

What is the purpose of the `useActionState` hook in React?

It manages state based on the result of an action, including form submissions, button clicks, and other event-driven updates.

It only updates state when a form is submitted and does not support other types of actions like button clicks or custom event handlers.

It handles global state management across multiple components, making it a replacement for state management libraries like Redux or Zustand.

It replaces the `useState` hook entirely by providing a built-in way to manage both local and global state without additional dependencies.

Why can't you use the `useActionState` hook inside a server component?

Because server components do not support state management, making `useActionState` incompatible.

Because `useActionState` requires a database connection, which is only available in client components.

Because `useActionState` is a React hook, and hooks can only be used inside client components.

Because `useActionState` depends on browser APIs that are not available on the server.

Which version of React introduced server components and server actions?

React 17

React 18

React 19

React 16