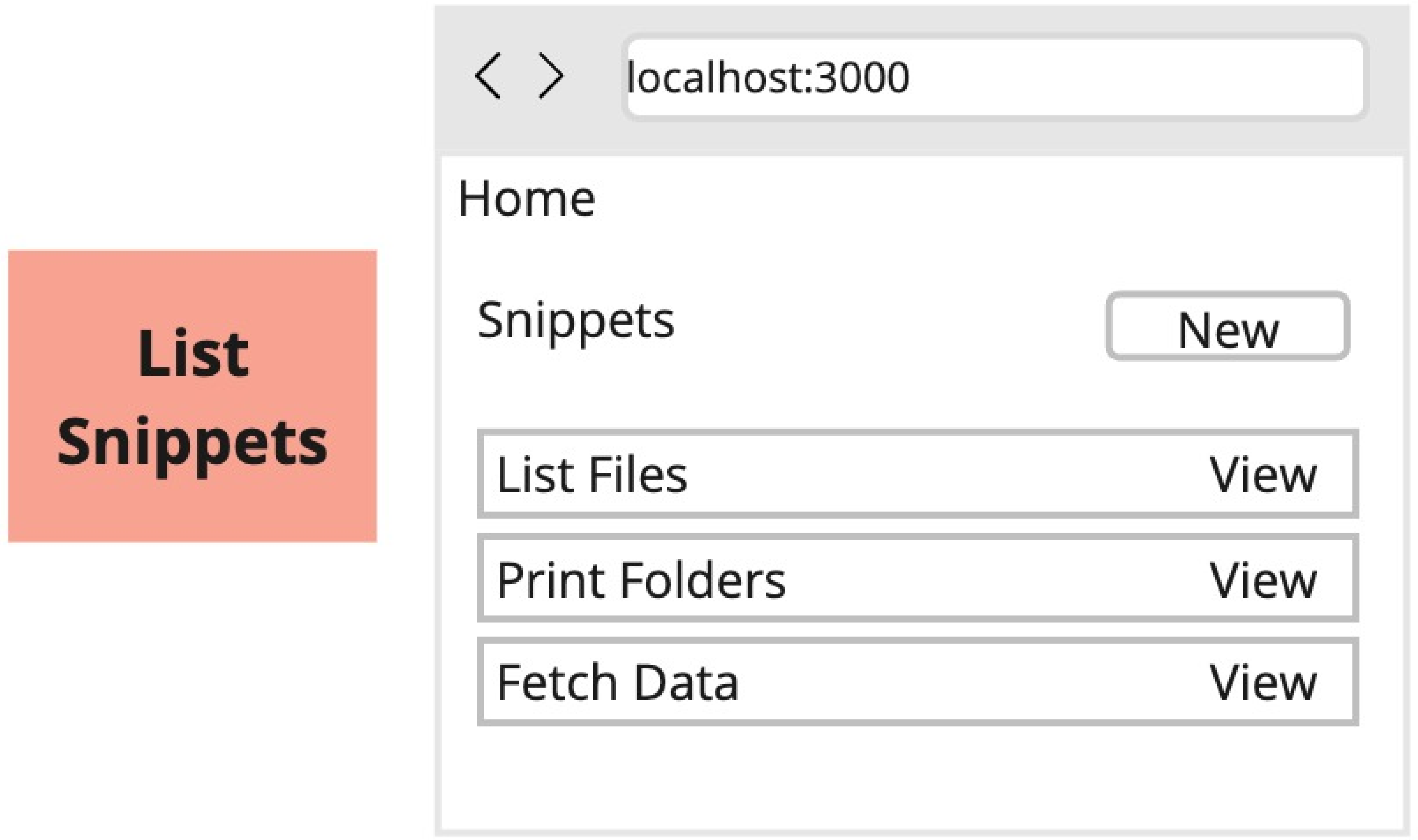
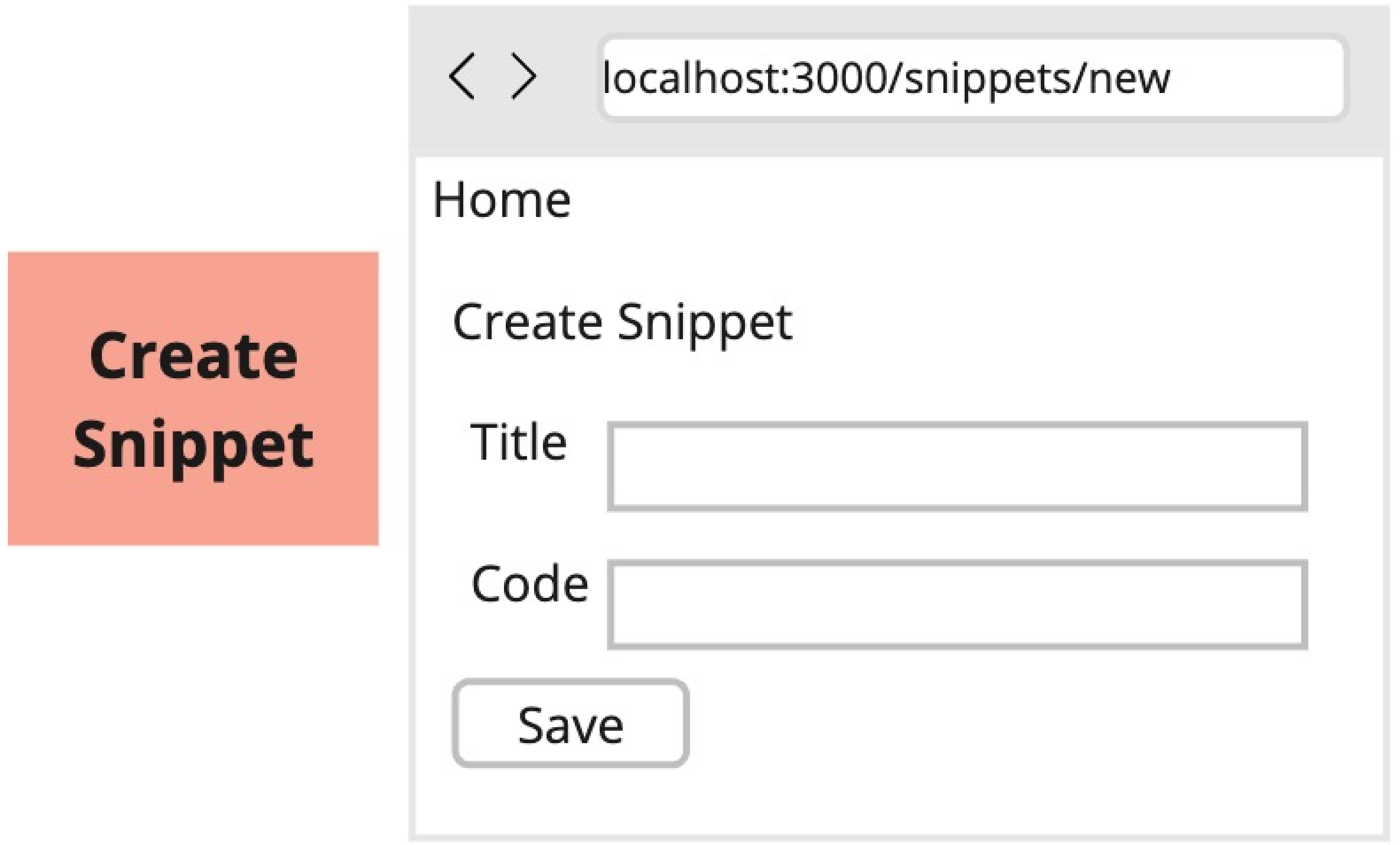
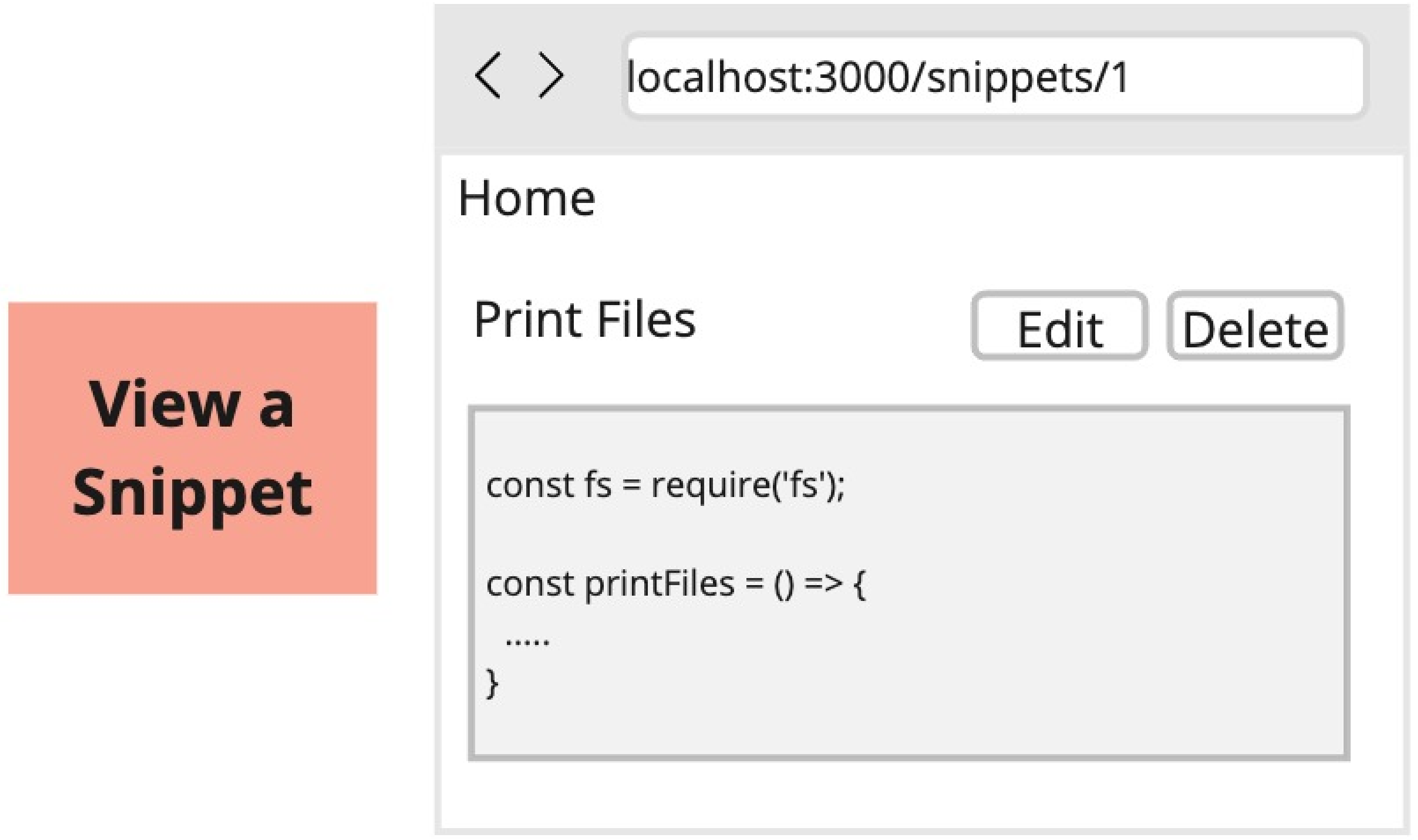
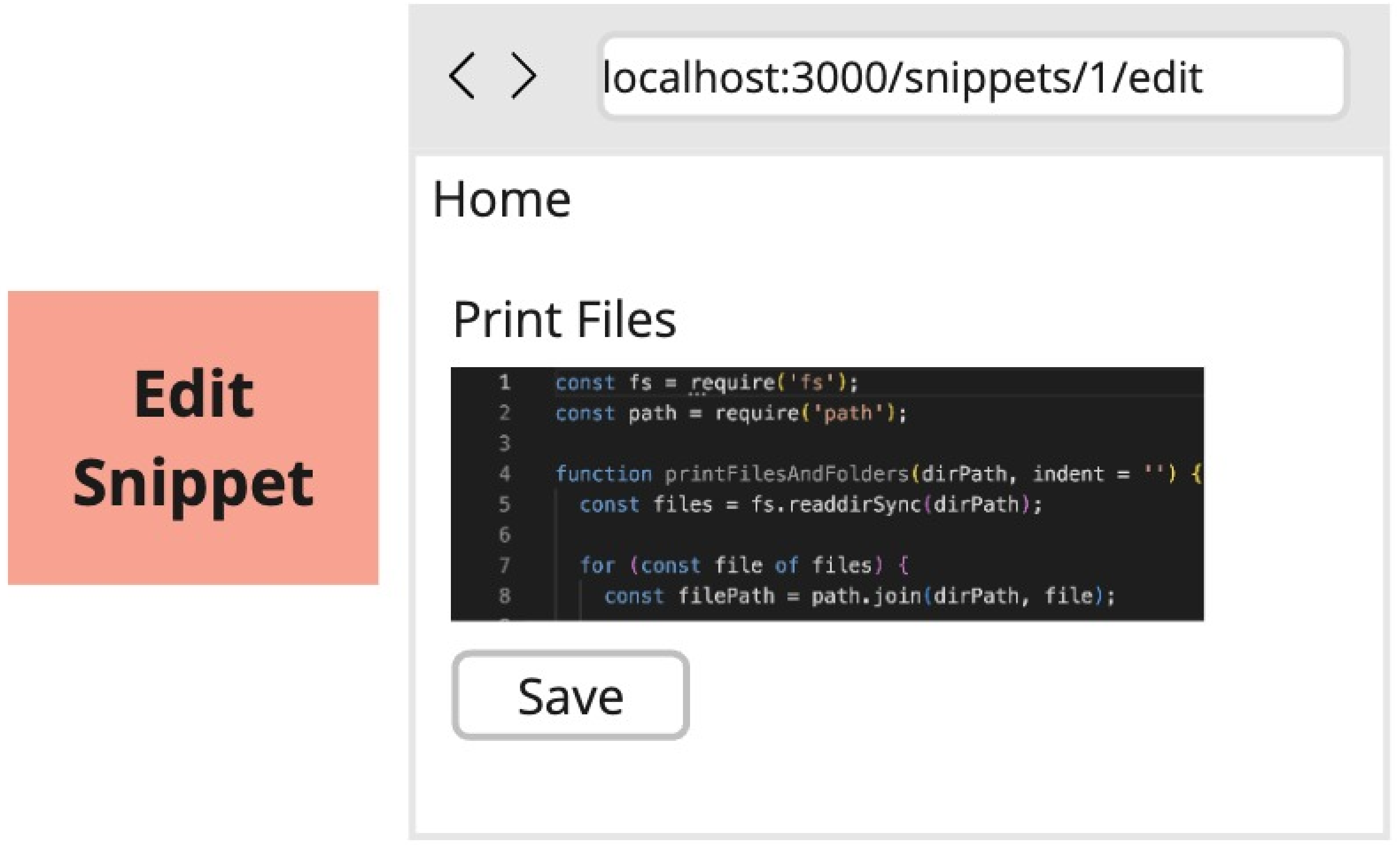






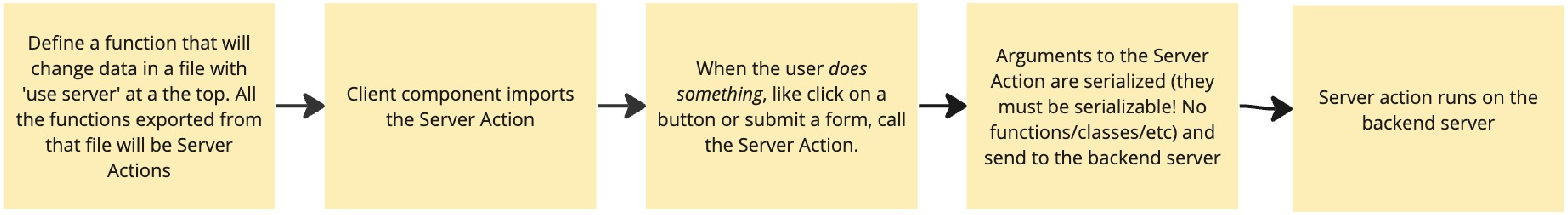
```
const fs = require('fs');

const printFiles = () => {
  .....
}
```



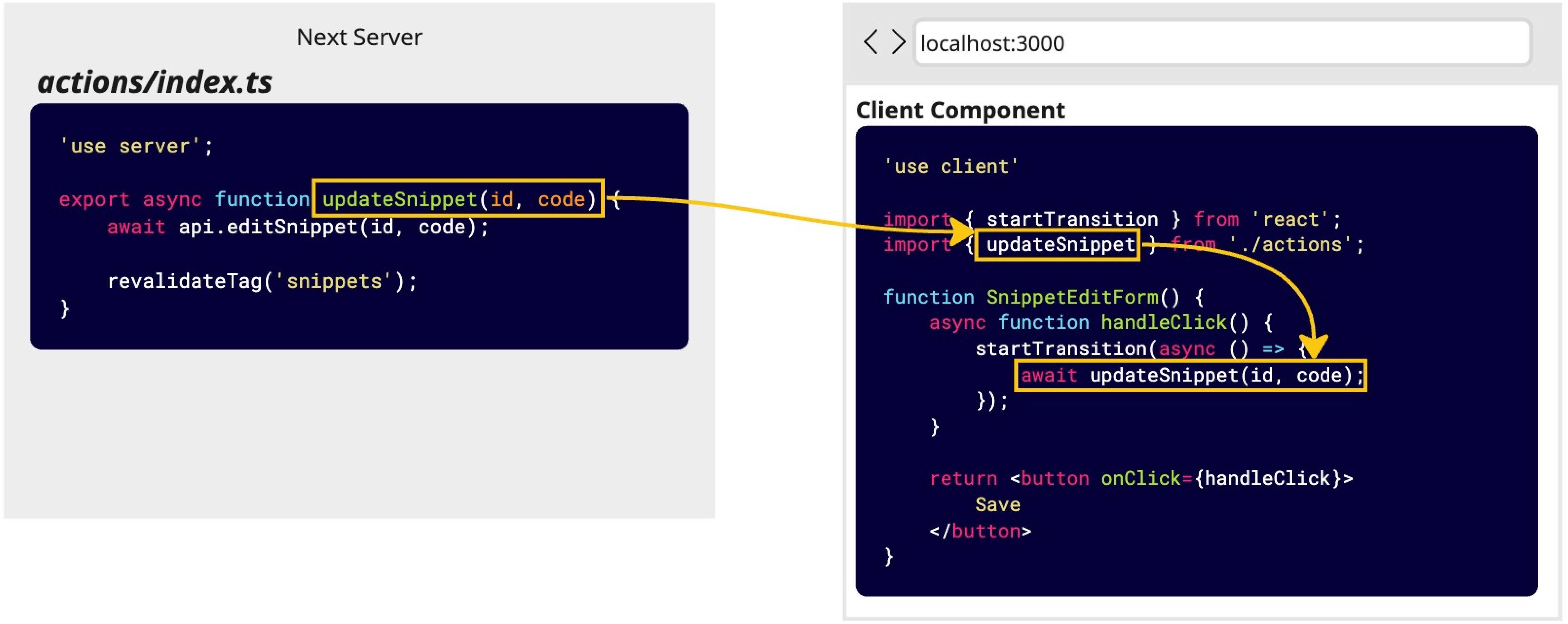
```
const fs = require('fs');
const path = require('path');

function printFilesAndFolders(dirPath, indent = '') {
  const files = fs.readdirSync(dirPath);

  for (const file of files) {
    const filePath = path.join(dirPath, file);
```

Purpose	Path	Method	Expected Body Data	Returns	Notes
List Snippets	/my/snippets	GET	-	Snippet[]	
Get snippets in a random order	/my/snippets/random	GET	-	Snippet[]	
Get a particular snippet by its ID	/my/snippets/:id	GET	-	Snippet	
Create a snippet	/my/snippets	POST	{ title: string, code: string }	Snippet	*Returns an error if you try to create a snippet with title or code that contains the string "hi there"*
Edit a snippet	/my/snippets/:id	PUT	{ code: string }	Snippet	*Returns an error if you try to create a snippet with code that contains the string "hi there"*
Delete a snippet	/my/snippets/:id	DELETE	-	Snippet	Returns an error if you try to delete the snippet with title "Can't Delete Me!"

Server Actions for Client Components



Define a function that will change data in a file with 'use server' at a the top. All the functions exported from that file will be Server Actions → Client component imports the Server Action → When the user *does something*, like click on a button or submit a form, call the Server Action. → Arguments to the Server Action are serialized (they must be serializable! No functions/classes/etc) and send to the backend server → Server action runs on the backend server



Next Server

actions/index.ts

```
'use server';

export async function updateSnippet(id, code) {
    await api.editSnippet(id, code);

    revalidateTag('snippets');
}
```

Client Component

```
'use client'

import { startTransition } from 'react';
import { updateSnippet } from './actions';

function SnippetEditForm() {
    async function handleClick() {
        startTransition(async () => {
            await updateSnippet(id, code);
        });
    }

    return <button onClick={handleClick}>
        Save
    </button>
}
```

```
export default function Page() {
    async function createSnippet() {
        'use server';

        // call api
        // revalidate
        // navigate
    }

    return (
      <div>
        .....
      </div>
    );
}
```

Alternate way to define a server action

Define your server action in a **server component**, placing 'use server' at the top of the function

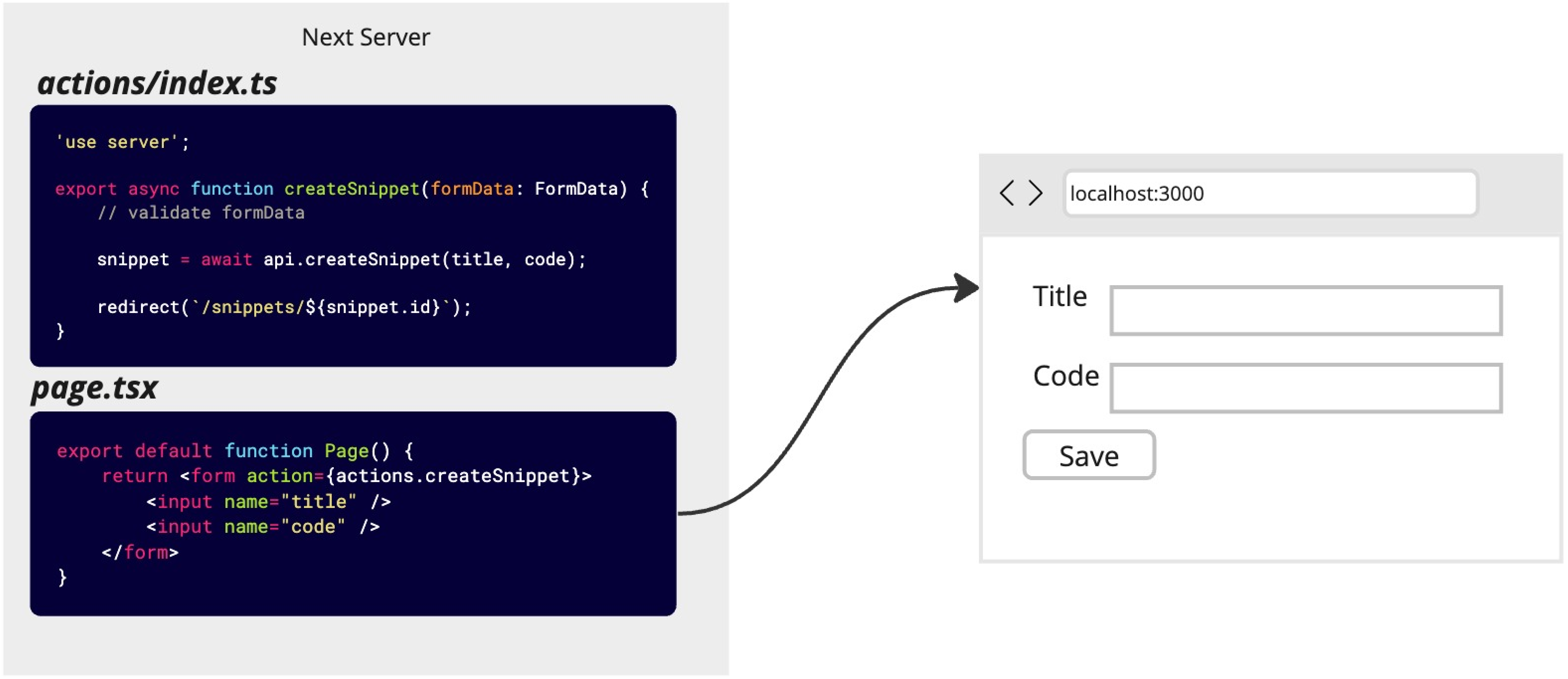
Cannot be done in a client component

The docs show this approach in a few places

I really recommend you **not** do this. Zero code reuse. Messier components.

Server Actions for Server Forms

Define a function that will change data in a file with 'use server' at a the top. All the functions exported from that file will be Server Actions → **Server component** imports the Server Action and assigns it to the **action** prop of a form. → When the user submits the form, the browser automatically collects values from 'input', 'textarea', and 'select' elements → Form values assembled into a 'FormData' object and sent to the server action → Server action runs on the backend server



actions/index.ts

```ts
'use server';

export async function createSnippet(formData: FormData) {
    // validate formData

    snippet = await api.createSnippet(title, code);

    redirect(`/snippets/${snippet.id}`);
}
```

page.tsx

```tsx
export default function Page() {
    return <form action={actions.createSnippet}>
        <input name="title" />
        <input name="code" />
    </form>
}
```

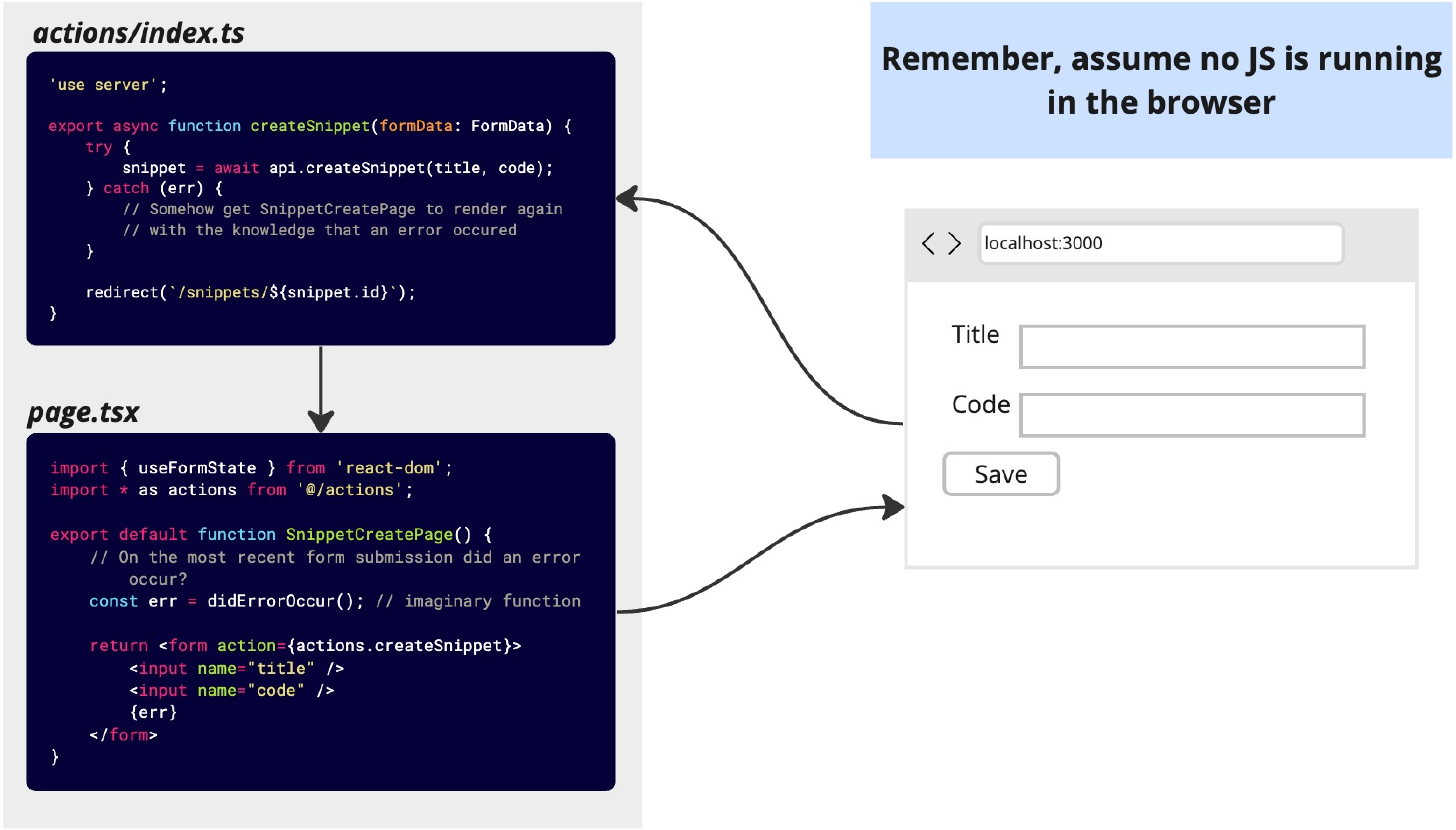
Server Actions called from **Client Components**	Server Actions called from **Server Components**
Can be invoked from a user doing just about anything - submitting a form, typing, etc	Can only be called when a user submits a form. Server Action must be assigned to a form's "action" prop
If server action returns a value, we can use it inside the component	**Much** more challenging to use values that the server action returns
Arguments to the server action are exactly what we pass into it from the Client Component	Argument to the server action is a 'FormData' object, which contains values from 'input', 'select', etc in the form
	Can be used without Javascript running on the users device

Define a schema with some validation rules

```
import { z } from 'zod';

const createSnippetSchema = z.object({
    title: z.string(),
    code: z.string()
})
```

Use the schema to validate some data. The appropriate type will be applied to the output

```
const { title, code } = createSnippetSchema.parse({
    title: formData.get('title'),
    code: formData.get('code')
});
```


actions/index.ts

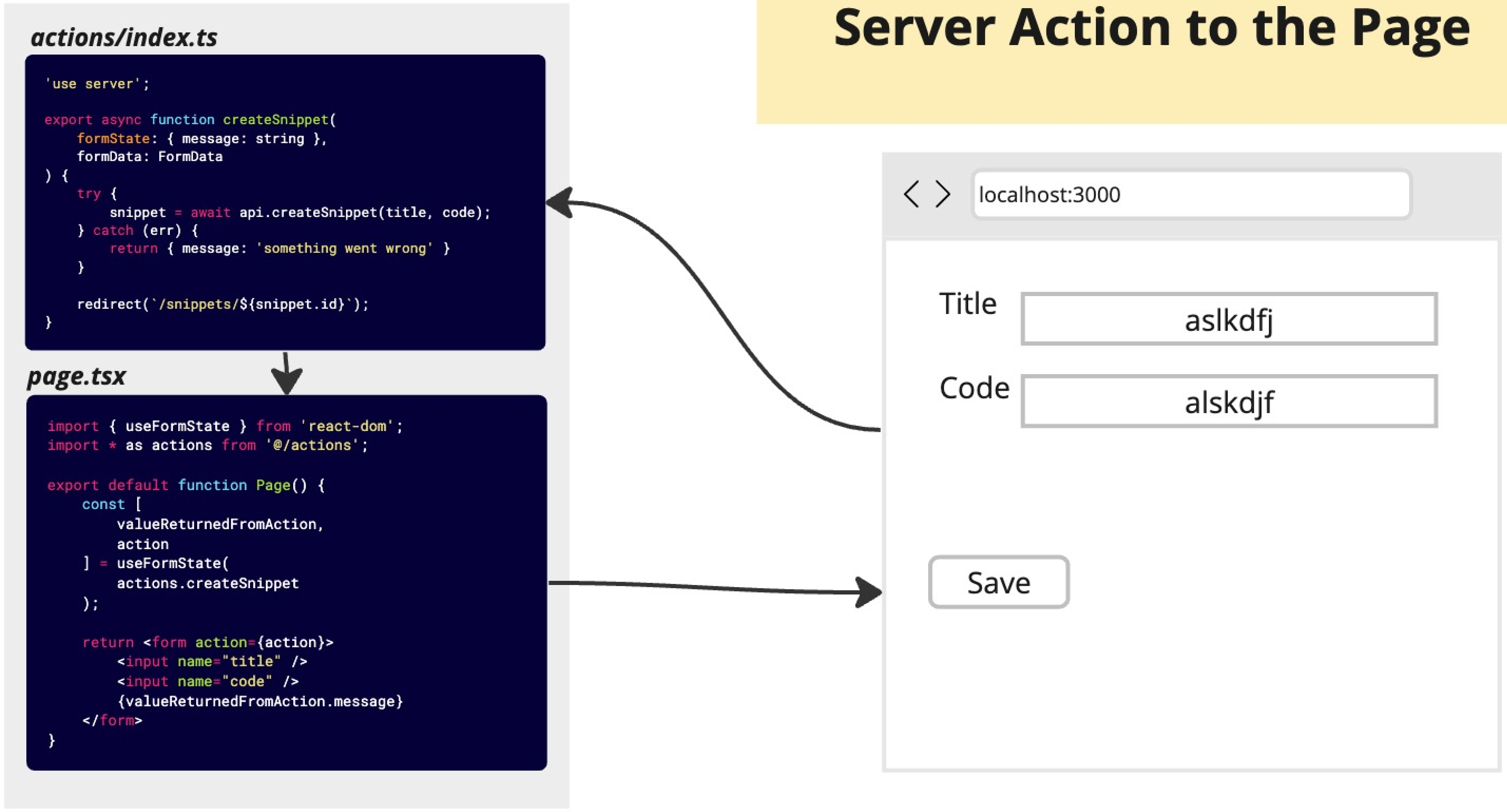
```
'use server';

export async function createSnippet(formData: FormData) {
    try {
        snippet = await api.createSnippet(title, code);
    } catch (err) {
        // Somehow get SnippetCreatePage to render again
        // with the knowledge that an error occured
    }

    redirect(`/snippets/${snippet.id}`);
}
```

page.tsx

```
import { useFormState } from 'react-dom';
import * as actions from '@/actions';

export default function SnippetCreatePage() {
    // On the most recent form submission did an error
        occur?
    const err = didErrorOccur(); // imaginary function

    return <form action={actions.createSnippet}>
        <input name="title" />
        <input name="code" />
        {err}
    </form>
}
```

Remember, assume no JS is running in the browser

Communicating from the Server Action to the Page



actions/index.ts

```
'use server';

export async function createSnippet(
    formState: { message: string },
    formData: FormData
) {
    try {
        snippet = await api.createSnippet(title, code);
    } catch (err) {
        return { message: 'something went wrong' }
    }

    redirect(`/snippets/${snippet.id}`);
}
```

page.tsx

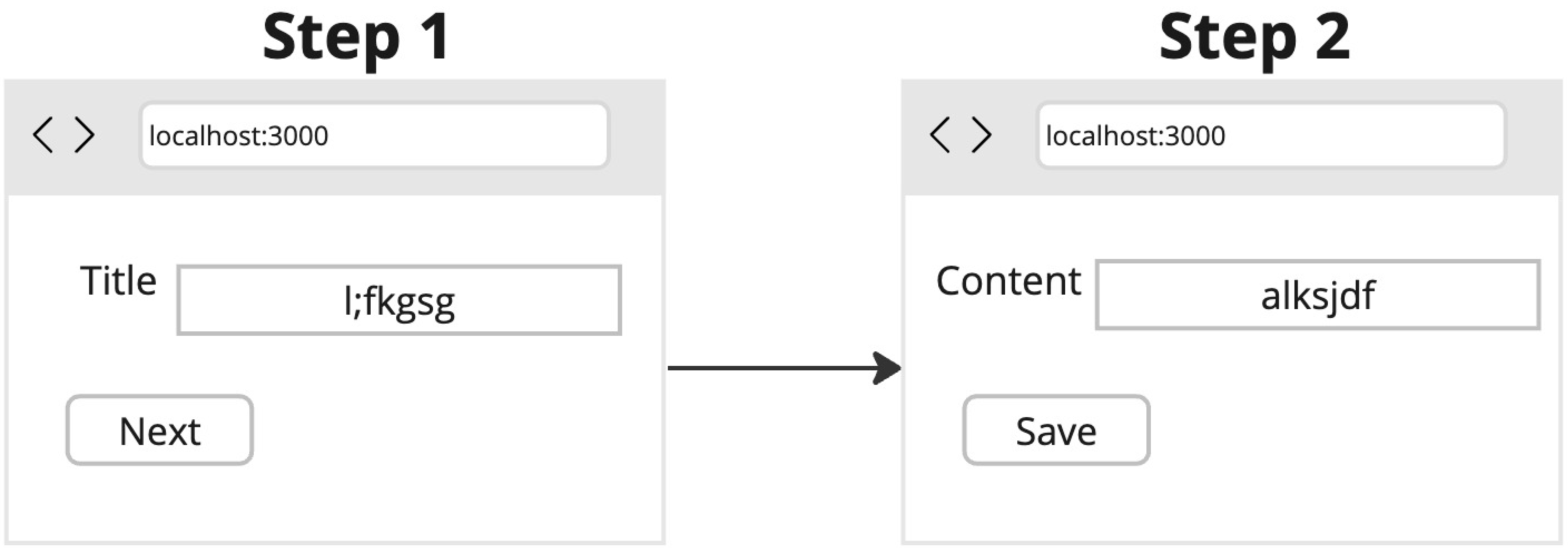
```
import { useFormState } from 'react-dom';
import * as actions from '@/actions';

export default function Page() {
    const [
        valueReturnedFromAction,
        action
    ] = useFormState(
        actions.createSnippet
    );

    return <form action={action}>
        <input name="title" />
        <input name="code" />
        {valueReturnedFromAction.message}
    </form>
}
```

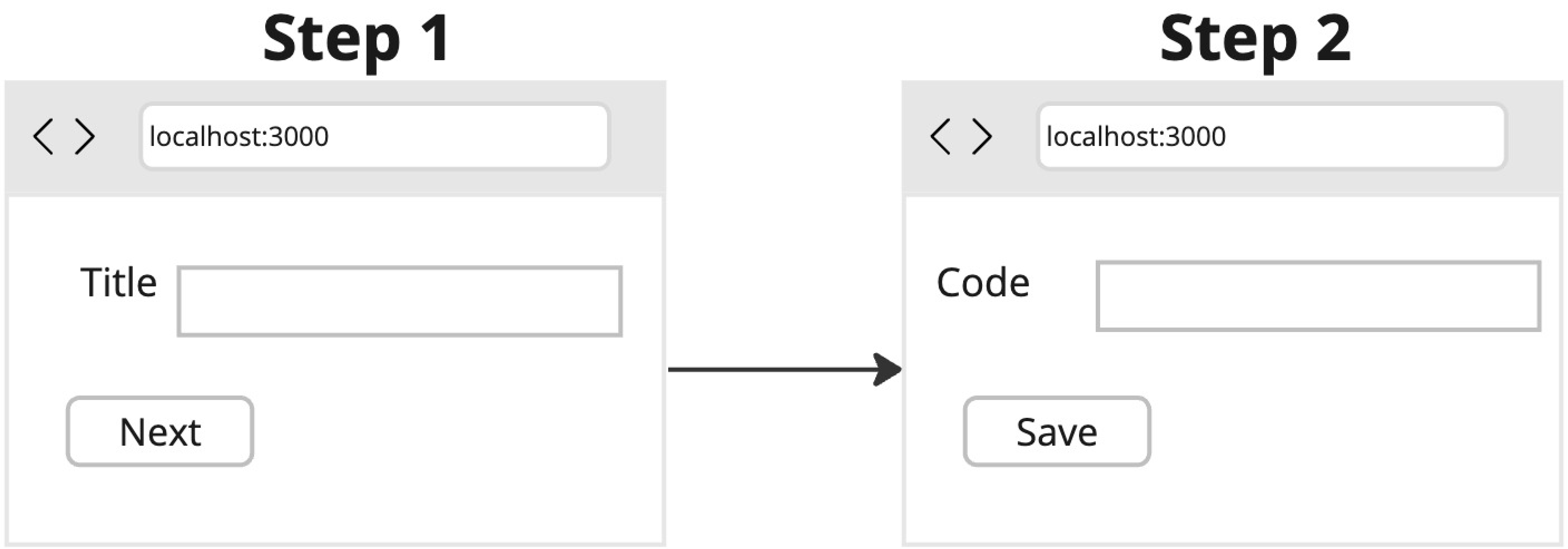
useFormState

Hook provided by 'react-dom' (not 'react')

Allows for communication between a Server Action and a client component

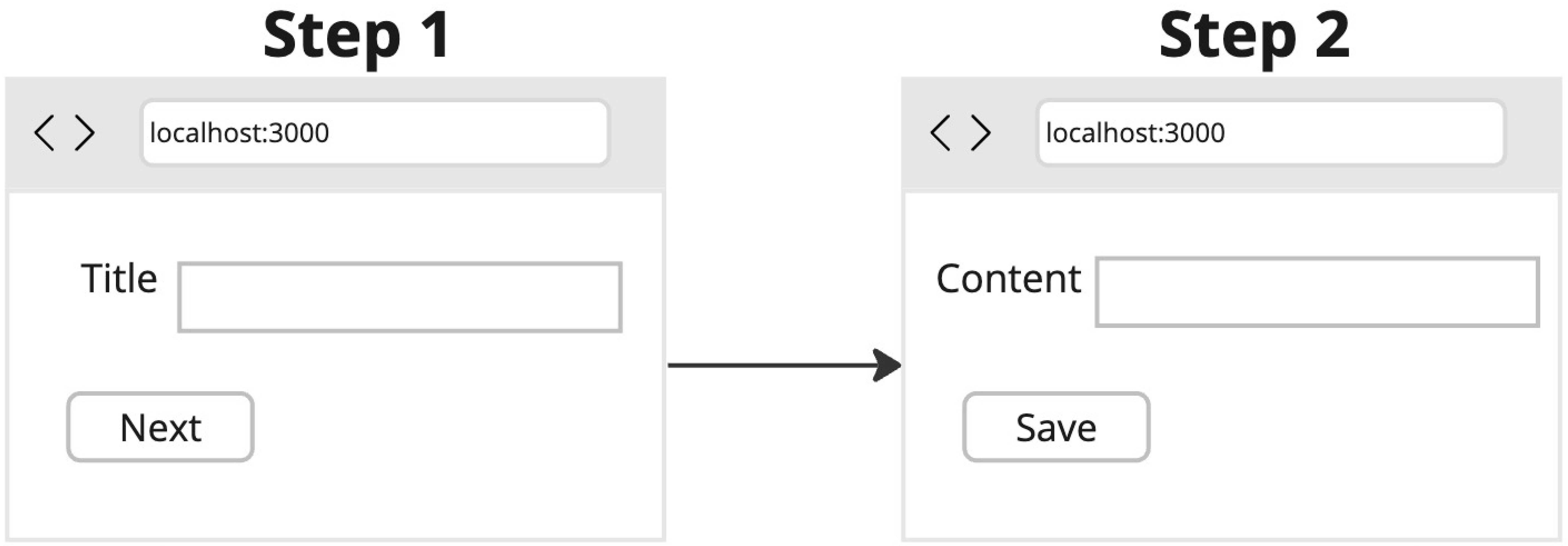
Works even if javascript is turned off!


Step 1
localhost:3000
Title
l;fkgsg
Next
Step 2
localhost:3000
Content
alksjdf
Save

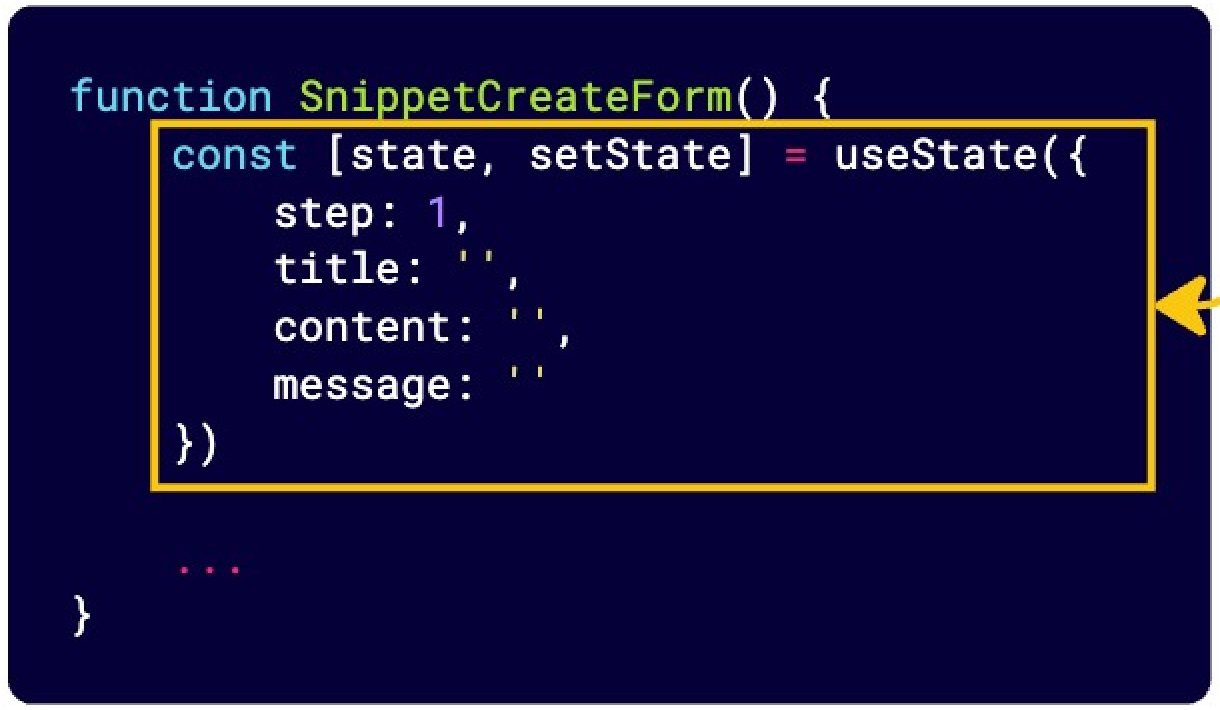


How would we implement this with the traditional 'useState' hook in a normal client component where JS is allowed?

```
function SnippetCreateForm() {
    const [state, setState] = useState({
        title: '',
        code: '',
        step: 1,
        message: ''
    });

    if (state.step === 1) {
        <div>
            <input value={state.title} onChange= />
            <button onClick={() => state.step + 1}>Next
```



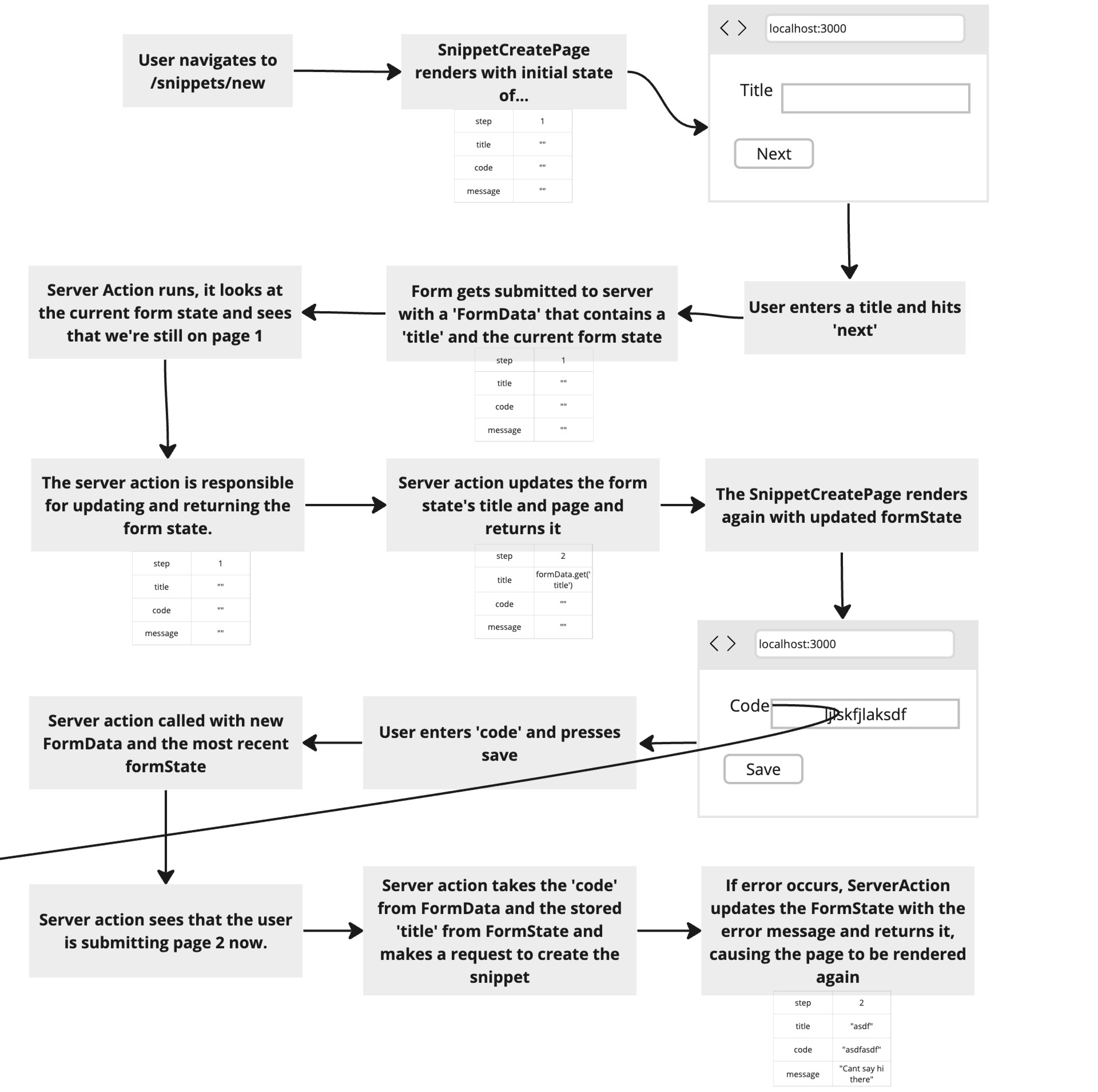
How would we implement this with the traditional 'useState' hook in a normal client component where JS is allowed?



```
function SnippetCreateForm() {
    const [state, setState] = useState({
        step: 1,
        title: '',
        content: '',
        message: ''
    })

    ...
}
```

This state has to be managed somewhere! Not in the browser because we don't want to use JS.

It will 'ping-pong' back and forth between the form and the server action through useFormState



User navigates to /snippets/new

SnippetCreatePage renders with initial state of...

step	1
title	""
code	""
message	""

User enters a title and hits 'next'

Form gets submitted to server with a 'FormData' that contains a 'title' and the current form state

step	1
title	""
code	""
message	""

Server Action runs, it looks at the current form state and sees that we're still on page 1

The server action is responsible for updating and returning the form state.

step	1
title	""
code	""
message	""

Server action updates the form state's title and page and returns it

step	2
title	formData.get('title')
code	""
message	""

The SnippetCreatePage renders again with updated formState

User enters 'code' and presses save

Server action called with new FormData and the most recent formState

Server action sees that the user is submitting page 2 now.

Server action takes the 'code' from FormData and the stored 'title' from FormState and makes a request to create the snippet

If error occurs, ServerAction updates the FormState with the error message and returns it, causing the page to be rendered again

step	2
title	"asdf"
code	"asdfasdf"
message	"Cant say hi there"