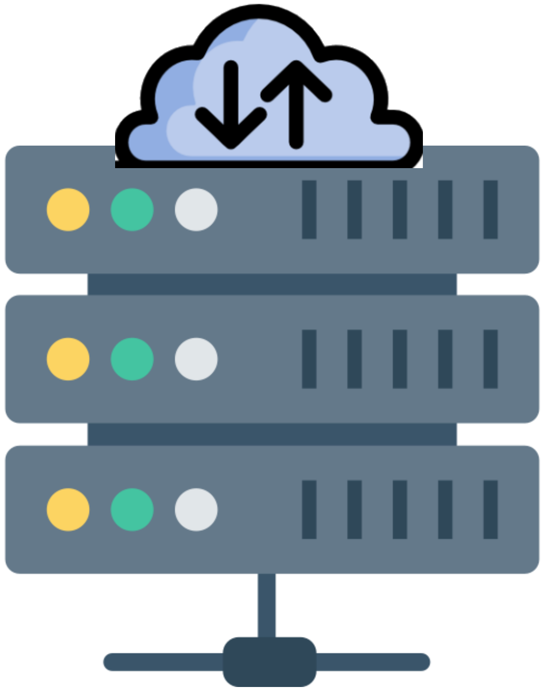
NEXT.JS

Server Actions
Data Fetching & Caching

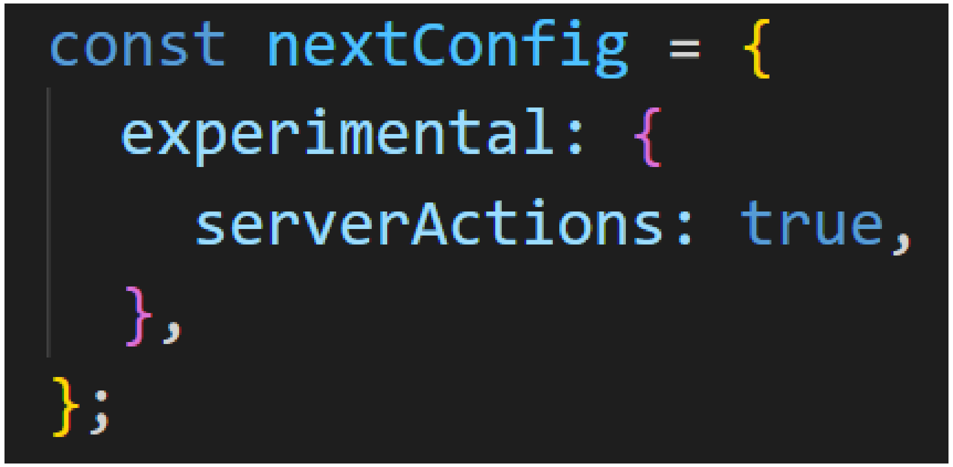
Server Actions



< Back

Server Actions

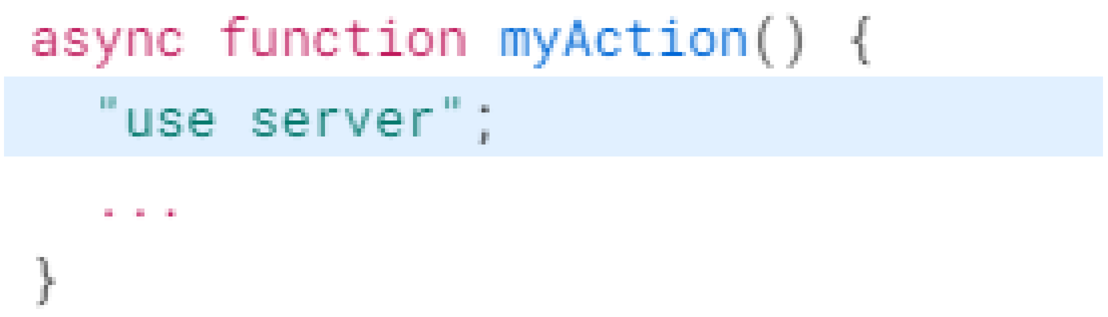
- Server Actions allow us to create functions that run on the server and can be called directly from pages/components **without needing to create an in-between Web API layer**
    - Simpler alternative to using client-side `fetch` and API routes for data mutations
    - Reduce client-side JavaScript
- Server Actions are not fully-stable yet, so you must opt-in via the **`next.config.js`** file



```
const nextConfig = {
  experimental: {
    serverActions: true,
  },
};
```

Server Actions

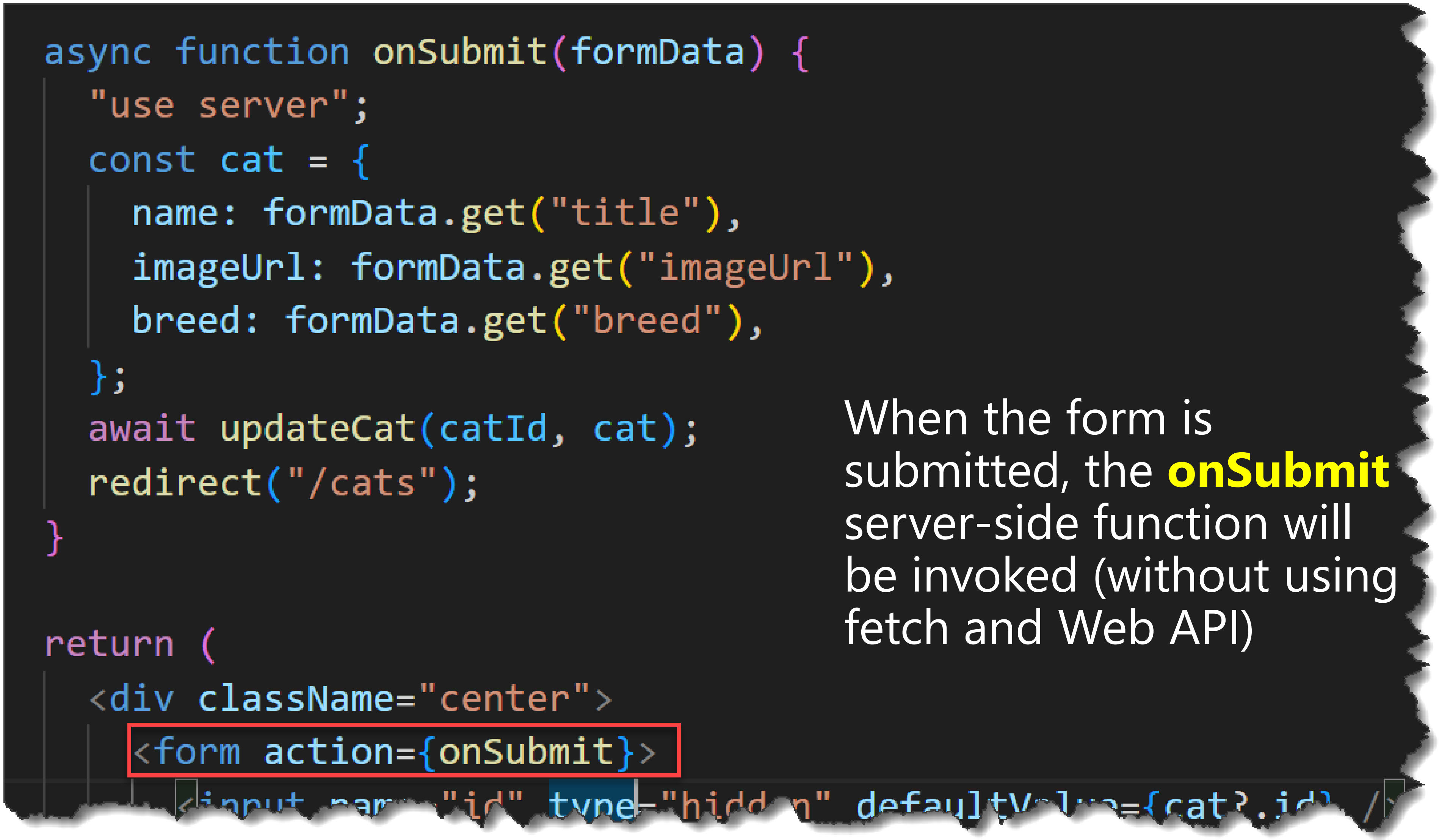
- Create a Server Action in a server-side component/page by defining an asynchronous function with the `"use server"` directive at the top of the function body



```
async function myAction() {
   "use server";
   ...
}
```

- To invoke a Server Action either:
  - Assign it to a form `action` attribute to handle the form submission
  - Pass it to a child client-side component to directly invoke it to handle an event such as button click

Example - Handle Form Submission

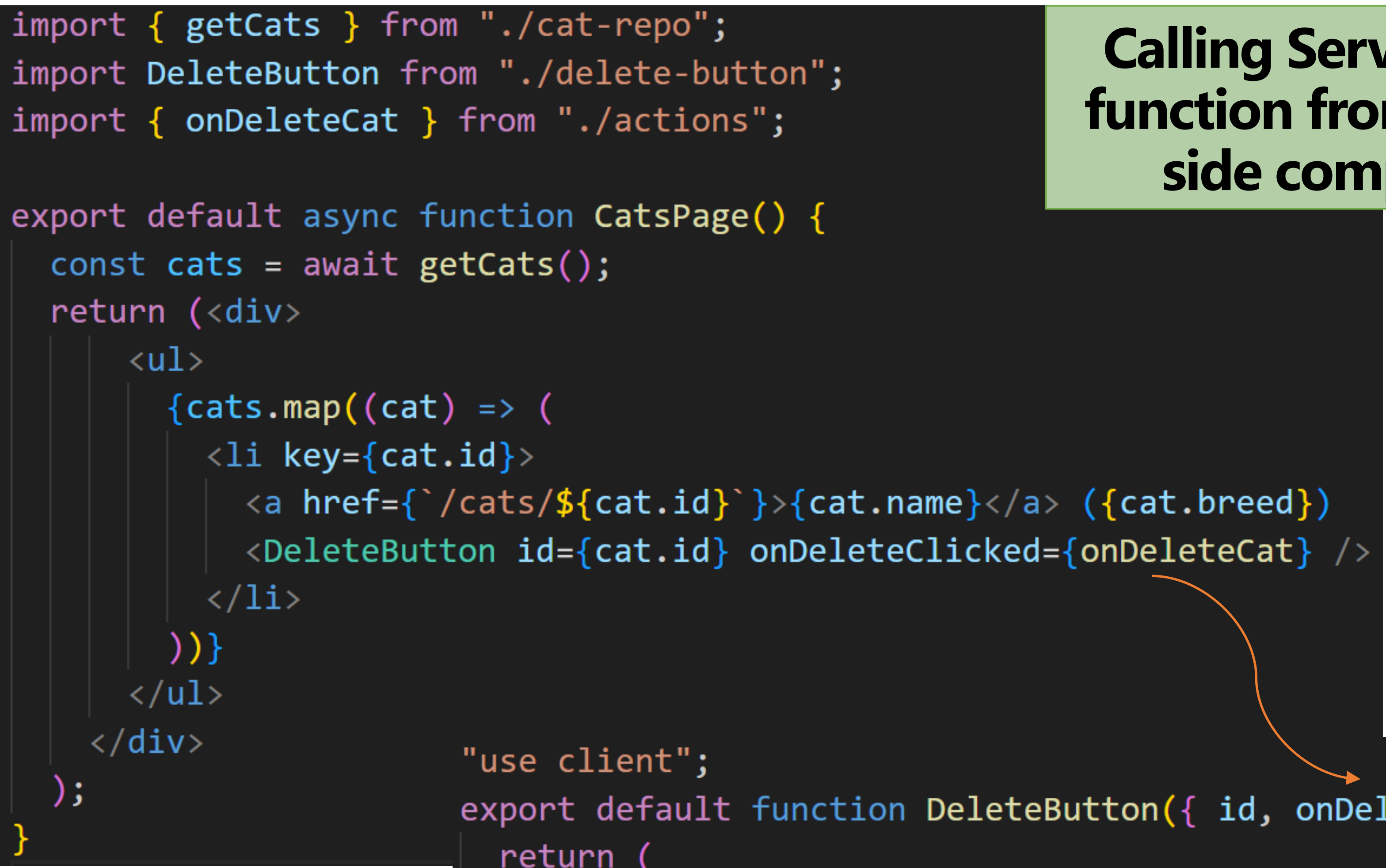


```
async function onSubmit(formData) {
  "use server";
  const cat = {
    name: formData.get("title"),
    imageUrl: formData.get("imageUrl"),
    breed: formData.get("breed"),
  };
  await updateCat(catId, cat);
  redirect("/cats");
}

return (
  <div className="center">
    <form action={onSubmit}>
      <input name="id" type="hidden" defaultValue={cat?.id} /
```

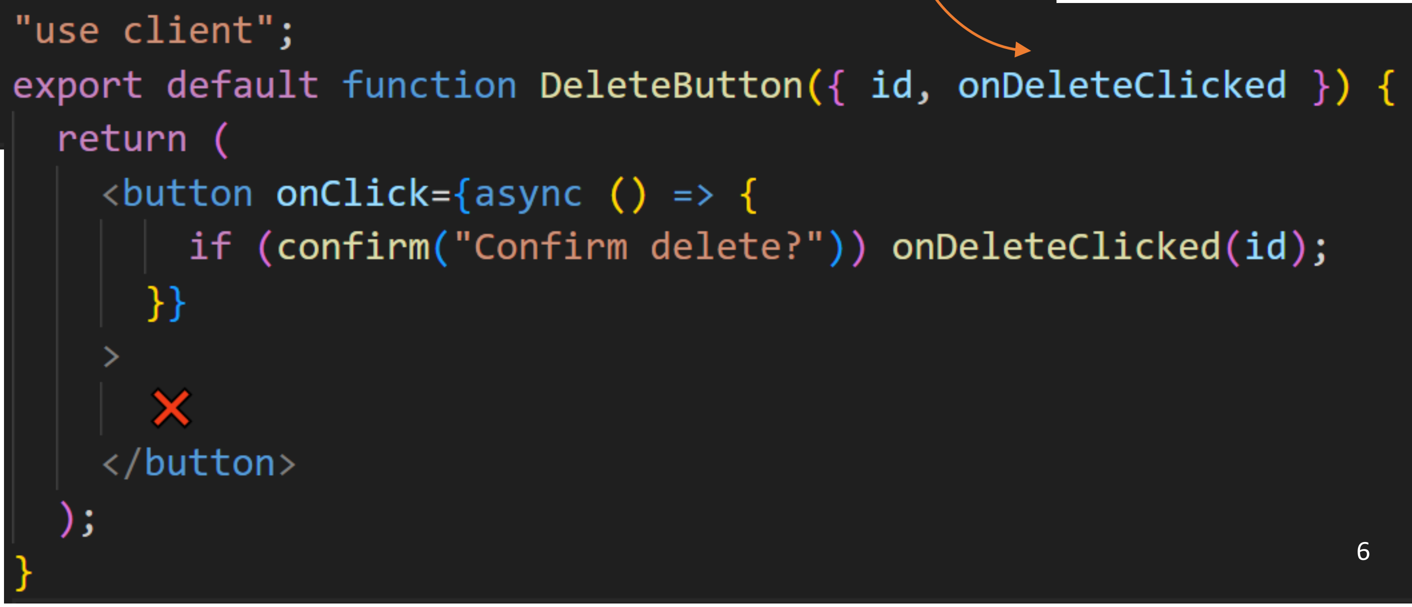
When the form is submitted, the **onSubmit** server-side function will be invoked (without using fetch and Web API)

After the update, the user is **redirected** to /cats

Calling Server Action function from a client-side component



```
import { getCats } from "./cat-repo";
import DeleteButton from "./delete-button";
import { onDeleteCat } from "./actions";

export default async function CatsPage() {
  const cats = await getCats();
  return (<div>
      <ul>
        {cats.map((cat) => (
          <li key={cat.id}>
            <a href={`/cats/${cat.id}`}>{cat.name}</a> ({cat.breed})
            <DeleteButton id={cat.id} onDeleteClicked={onDeleteCat} />
          </li>
        ))}
      </ul>
    </div>
  );
}
```

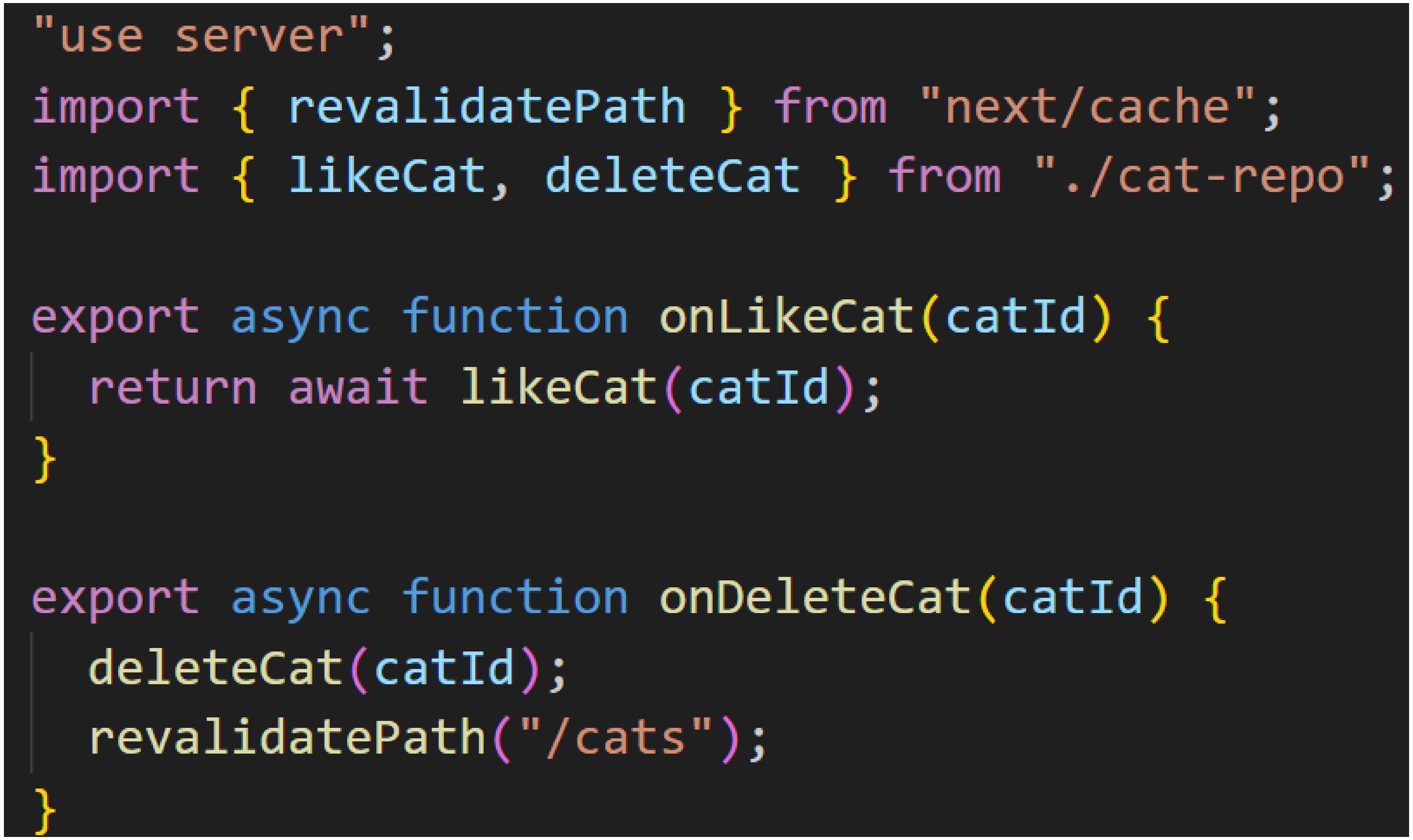


```
"use client";
export default function DeleteButton({ id, onDeleteClicked }) {
  return (
    <button onClick={async () => {
        if (confirm("Confirm delete?")) onDeleteClicked(id);
      }}
    >
      ❌
    </button>
  );
}
```

Server action function (**onDeleteCat**) is passed from CatsPage to the DeleteButton client-side component. It is called when the delete button is clicked.

Server Actions in actions.js file

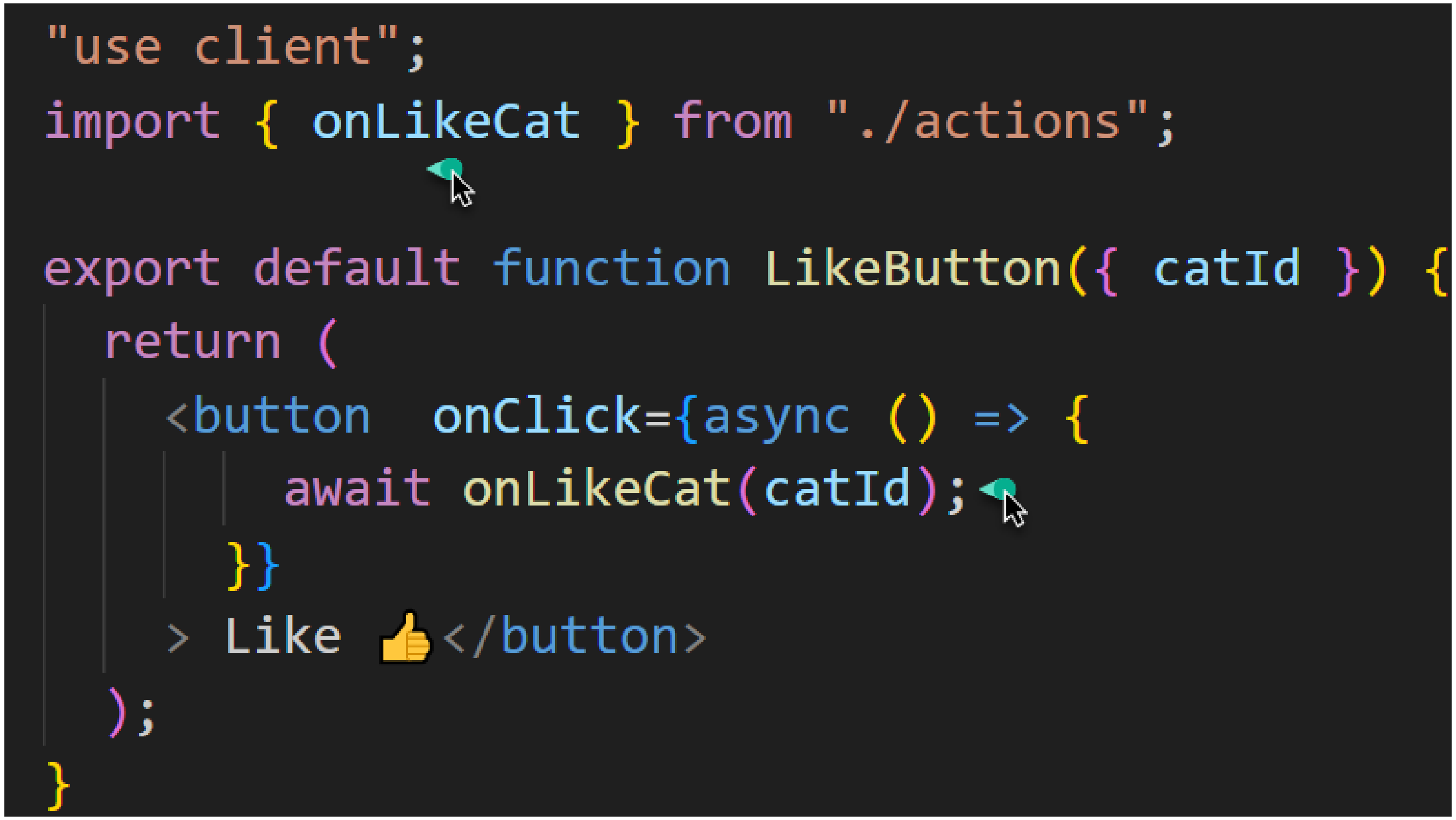
- Server Action asynchronous functions could be defined in a separate js file (such as `actions.js`) with the **`"use server"`** directive at the top of the file



```js
"use server";
import { revalidatePath } from "next/cache";
import { likeCat, deleteCat } from "./cat-repo";

export async function onLikeCat(catId) {
  return await likeCat(catId);
}

export async function onDeleteCat(catId) {
  deleteCat(catId);
  revalidatePath("/cats");
}
```

Components can import and call server action functions

- Components (including client-side ones) can **import** and **call** server action functions



```
"use client";
import { onLikeCat } from "./actions";

export default function LikeButton({ catId }) {
  return (
    <button  onClick={async () => {
        await onLikeCat(catId);
      }}
    > Like 👍</button>
  );
}
```

Re-rendering after Data Mutation

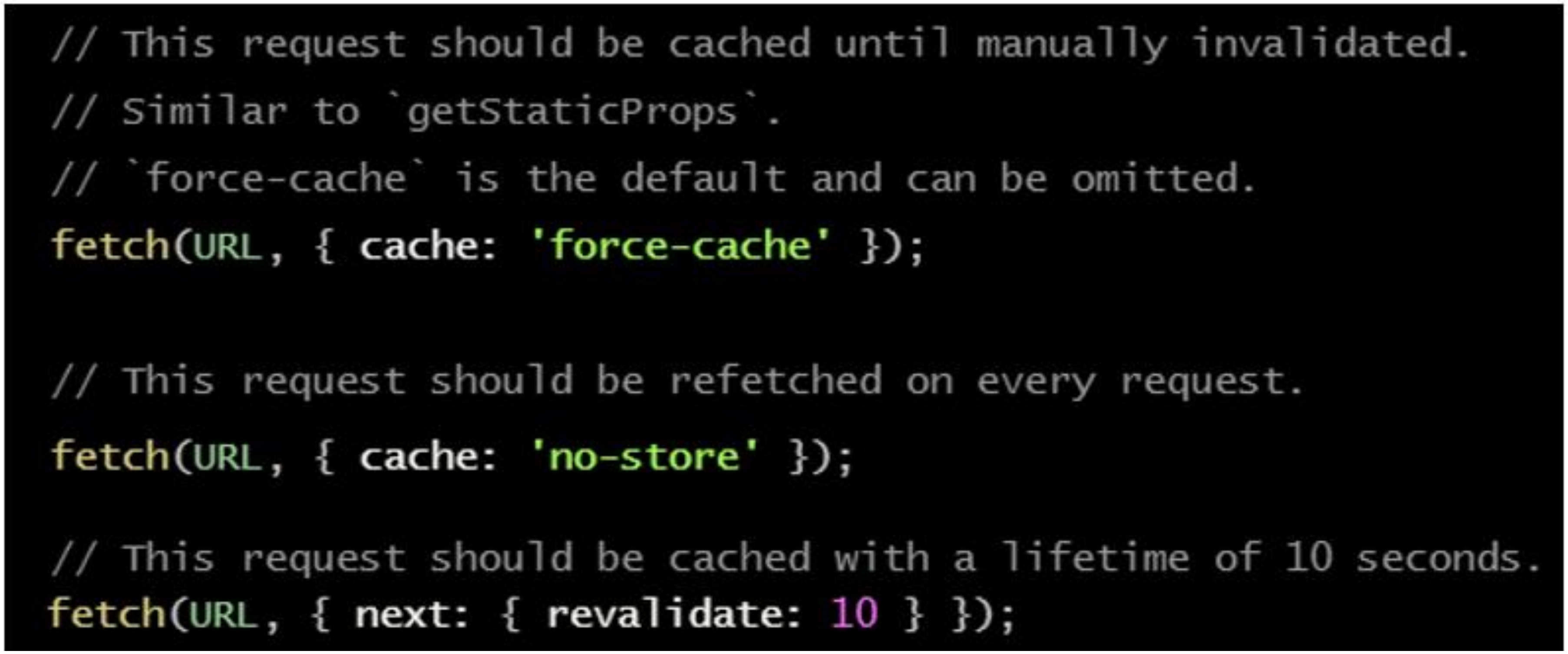
- After data mutation (e.g., handling the form submission to update a cat), you can re-render the UI to `ensure` the correct data is displayed on the client using:
  - **`revalidatePath`** function (from `"next/cache"` library) allows revalidating a Url to refresh the data
    - e.g., after deleting a cat `revalidatePath("/cats")` is called to refresh the list of cats
  - **`redirect`** function (from `"next/navigation"` library) allows redirecting to another page
    - e.g., after adding a cat `redirect("/cats")` is called to redirect to the cats page

Data Fetching & Caching

< Back

Data Fetching – Caching Options

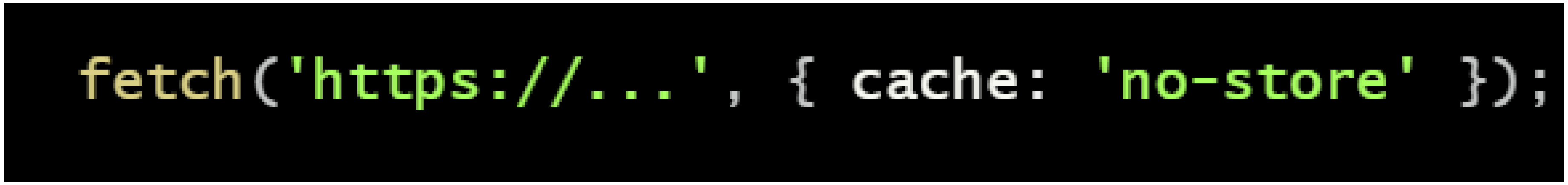
You can call fetch with async/await directly within Server Components



```
// This request should be cached until manually invalidated.
// Similar to `getStaticProps`.
// `force-cache` is the default and can be omitted.
fetch(URL, { cache: 'force-cache' });

// This request should be refetched on every request.
fetch(URL, { cache: 'no-store' });

// This request should be cached with a lifetime of 10 seconds.
fetch(URL, { next: { revalidate: 10 } });
```

Server-Side Rendering (SSR)

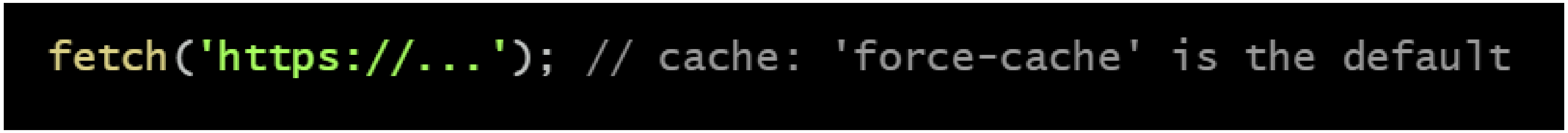
To refetch data on every fetch() request, use the `cache: 'no-store'` option



```
fetch('https://...', { cache: 'no-store' });
```

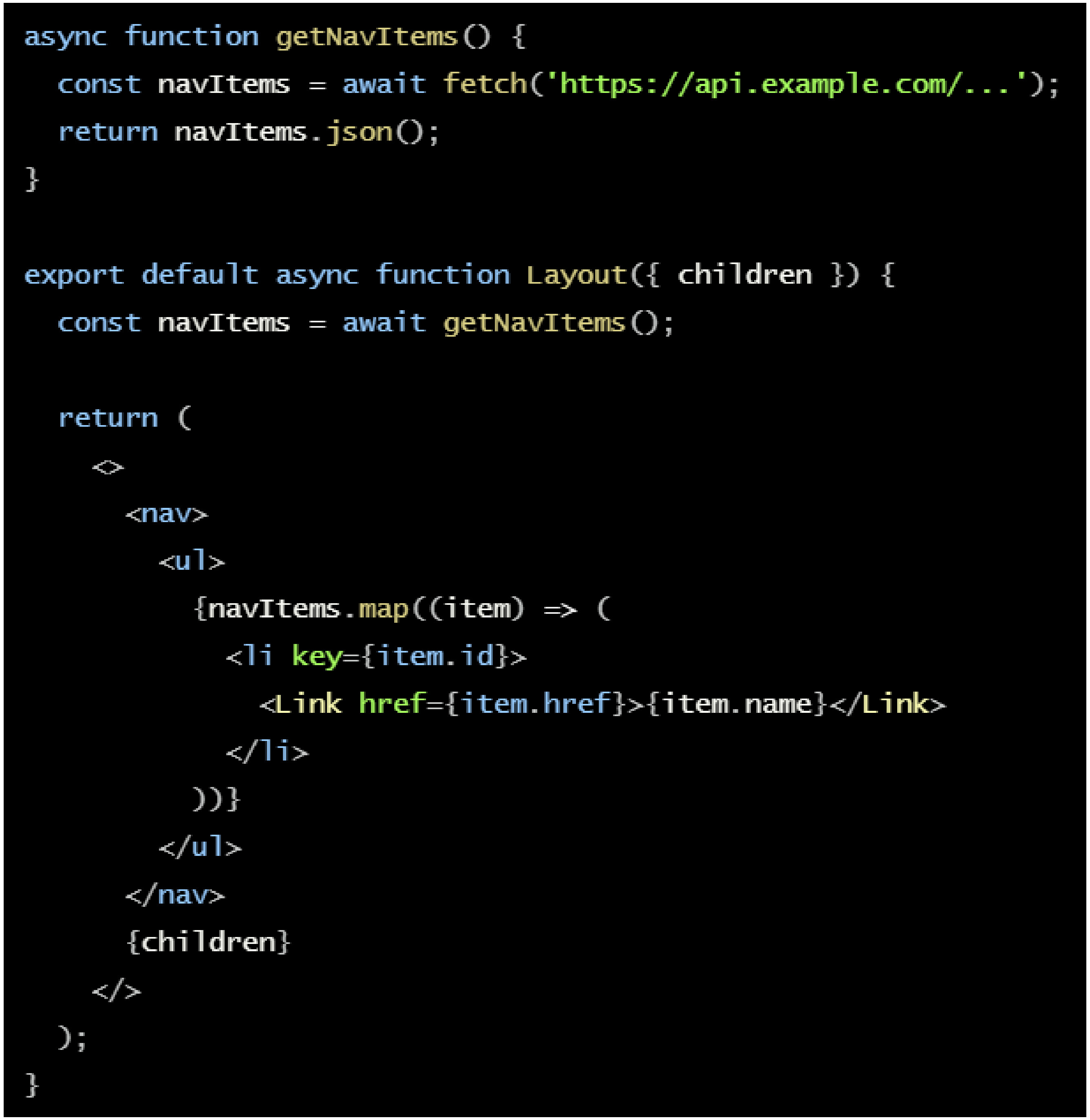
Static Site Generation (SSG)

By default, fetch will automatically fetch static data (cached data)



```
fetch('https://...'); // cache: 'force-cache' is the default
```

Static Site Generation Example

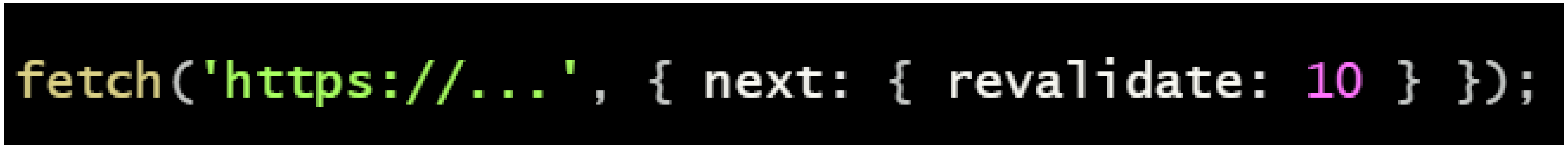


```
async function getNavItems() {
  const navItems = await fetch('https://api.example.com/...');
  return navItems.json();
}

export default async function Layout({ children }) {
  const navItems = await getNavItems();

  return (
    <>
      <nav>
        <ul>
          {navItems.map((item) => (
            <li key={item.id}>
              <Link href={item.href}>{item.name}</Link>
            </li>
          ))}
        </ul>
      </nav>
      {children}
    </>
  );
}
```

Revalidating Data

To revalidate cached data, you can use the `next.revalidate` option in fetch()

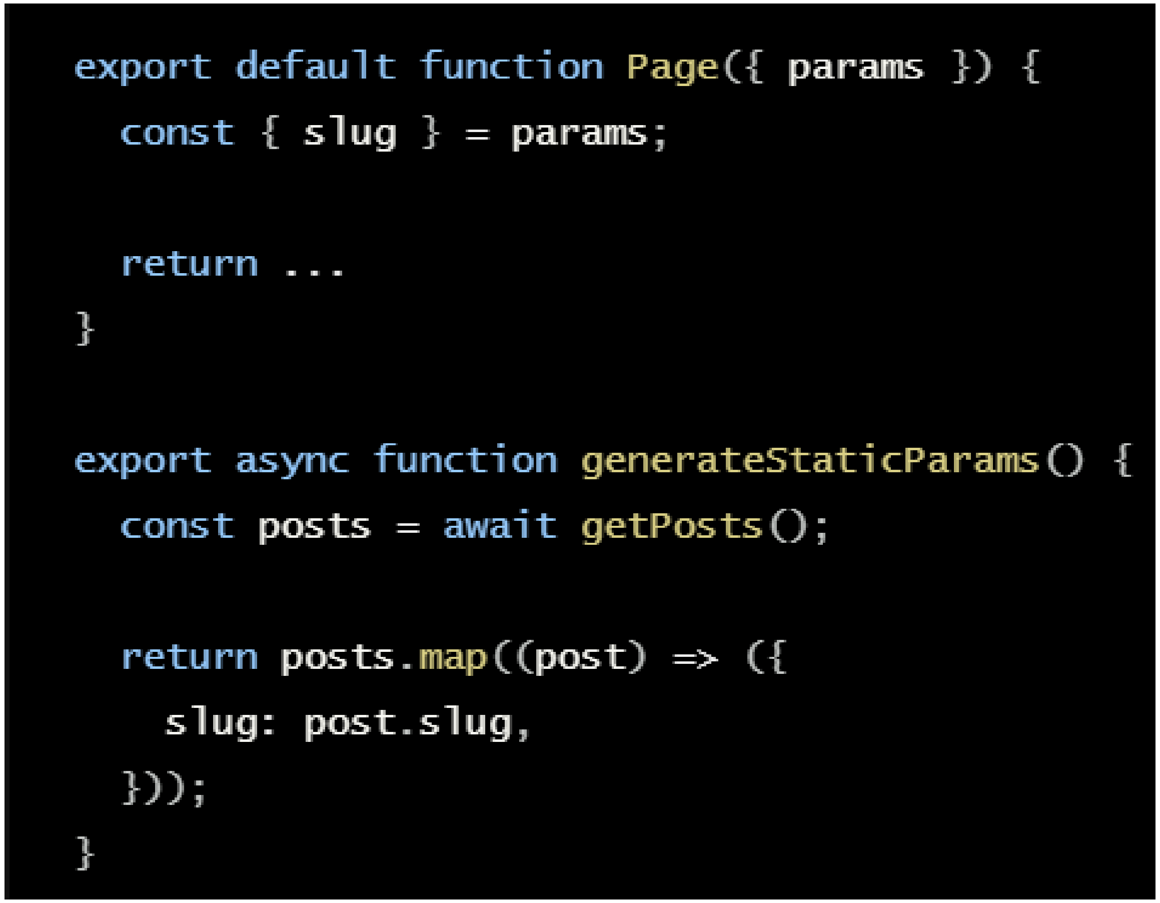
- Used for Incremental Static Regeneration (ISR)



```
fetch('https://...', { next: { revalidate: 10 } });
```

Generate Static Params

The `generateStaticParams` function can be used in combination with dynamic route segments to define the list of route segment parameters that will be statically generated at build time



```
export default function Page({ params }) {
  const { slug } = params;

  return ...
}

export async function generateStaticParams() {
  const posts = await getPosts();

  return posts.map((post) => ({
    slug: post.slug,
  }));
}
```

Summary

- Server Actions allow us to create functions that run on the server and can be called directly from pages/components **without needing to create an in-between Web API layer**

- Next.js supports different data caching strategies: Server-side rendering, Static site generation, Incremental static generation

Resources

- Server Actions

  https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions

- Data Fetching – Caching

  https://nextjs.org/docs/app/building-your-application/data-fetching