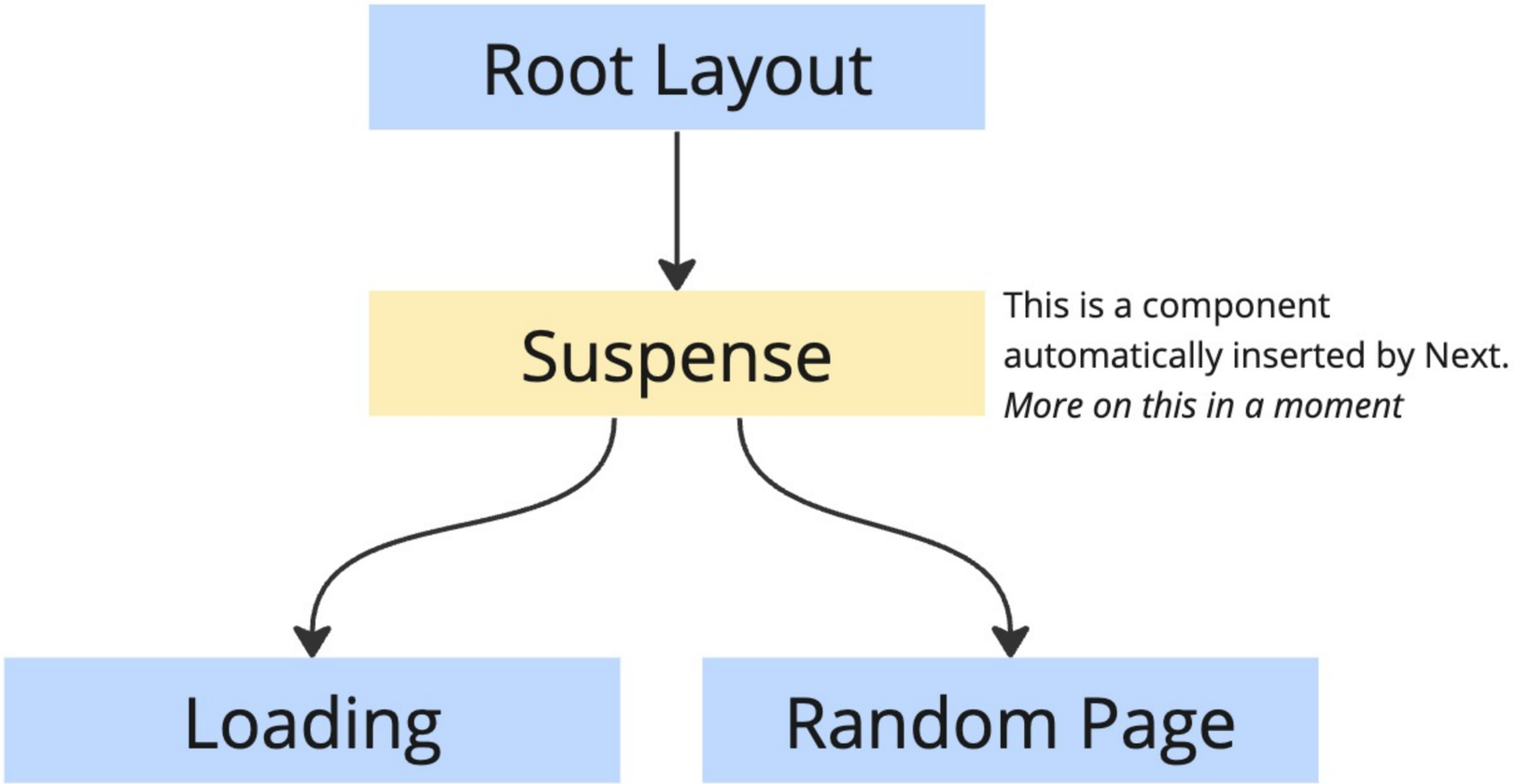
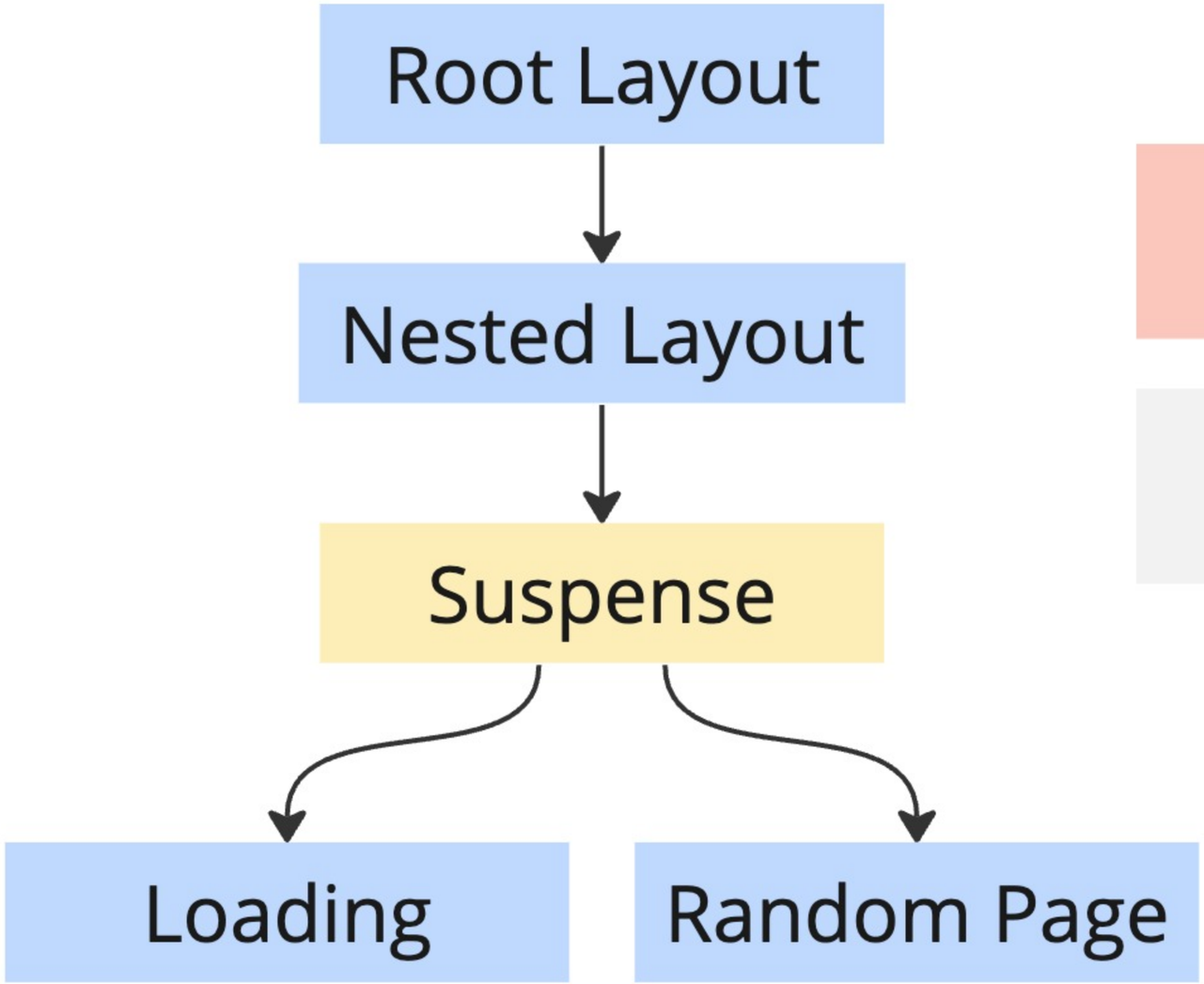
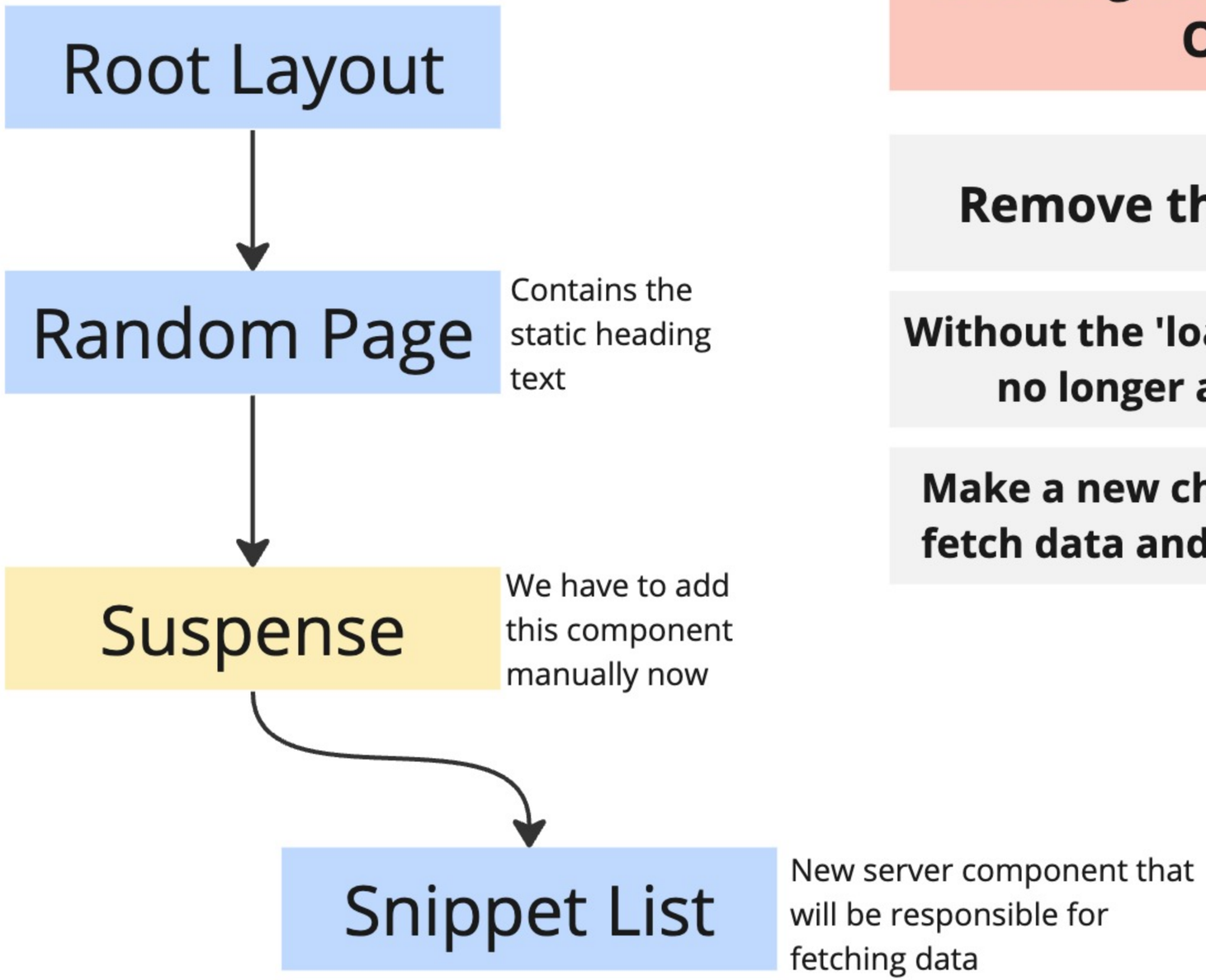

Root Layout
Suspense
This is a component automatically inserted by Next.
More on this in a moment
Loading
Random Page



Showing content during loading
Option #1

Add a new *nested layout* with some content appropriate for the page



Showing content during loading **Option #2**

Remove the 'loading.tsx' file

Without the 'loading.tsx' file, 'Suspense' no longer added automatically

Make a new child component that will fetch data and wrap it with 'Suspense'

List Snippets

localhost:3000

Home Random Projects

Snippets New

List Files View

Print Folders View

Fetch Data View

Create Snippet

localhost:3000/snippets/new

Home Random Projects

Create Snippet

Title

Code

Save

View a Snippet

localhost:3000/snippets/1

Home Random Projects

Print Files Edit Delete

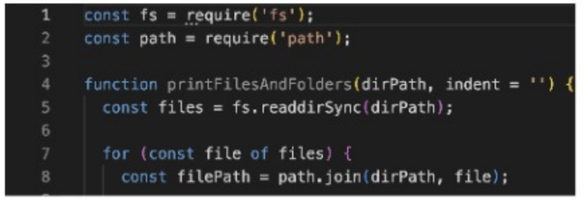
```
const fs = require('fs');

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

Edit Snippet

localhost:3000/snippets/1/edit

Home Random Projects

Print Files



Save

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!"

https://api-next.fly.dev/my/snippets**?key=12345&delay=2000&error=70**

Add on a random '**key**' query string param. This ensures you'll use different data from anyone else in this course.

'delay' query string param adds a delay to the response to simulate lag

'error' query string param adds chance for the request to return an error. Example: "error=30" means there's a 30% chance the request will error

Data Fetching Recommendations

Do make a wrapper function around fetch	→ Error handling, headers, 'body' creation with fetch is tedious. Make a helper function to automate some of these steps
Do make helper functions to access your api in a single file. Call these instead of calling 'fetch' directly in a server component	→ Rewriting the same 'fetch' calls in your different server components is error-prone and frequently makes revalidation more confusing.
Do use 'tags' for revalidation rather than paths.	→ Remembering the different 'paths' that have to be revalidated is **super** error-prone - paths change all the time!
Don't ignore error handling. Stuff blows up all the time!	→ Next doc's official recommendation on error handling is **extremely misleading for production environments.** Error handling behaves differently in dev vs prod!

Page Files

```
export default function ListSnippetsPage() {
    const snippets = await api.listSnippets();

    return <div>{snippets.map(() => ...)}</div>
}
```

```
export default function ShowSnippetsPage() {
    const snippet = await api.getSnippet(id);

    return <div>{snippet.title}</div>
}
```

```
export default function RandomSnippetsPage() {
    const snippets = await api.listRandomSnippets
        ();

    return <div>{snippets.map(() => {})}</div>
}
```

API file

```
export function listSnippets() {
    return request('/my/snippets');
}
```

```
export function getSnippet(id) {
    return request(`/my/snippets/${id}`);
}
```

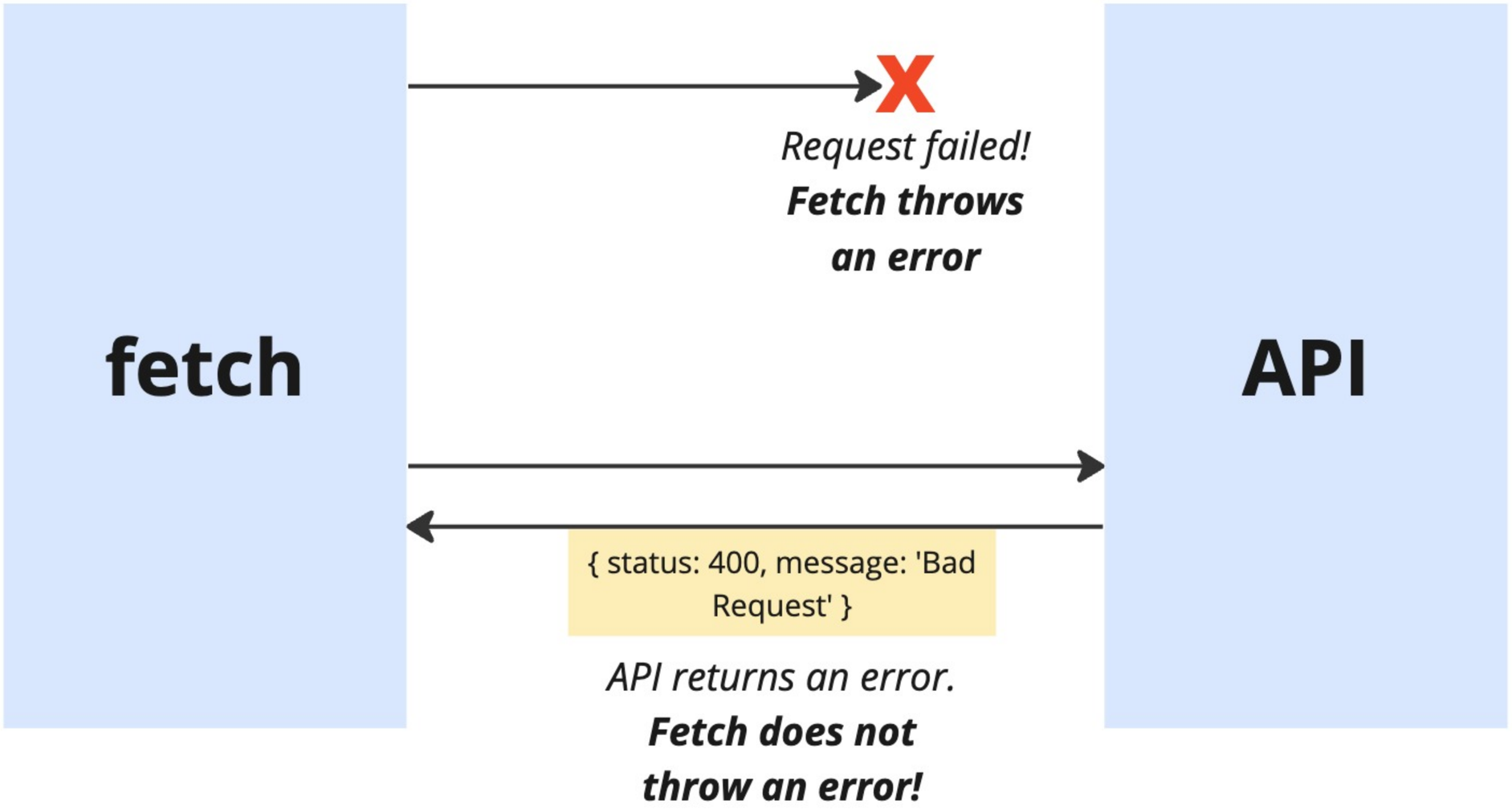
```
export function deleteSnippet(id) {
    return request(`/my/snippets/${id}`, {
        method: 'DELETE'
    });
}
```

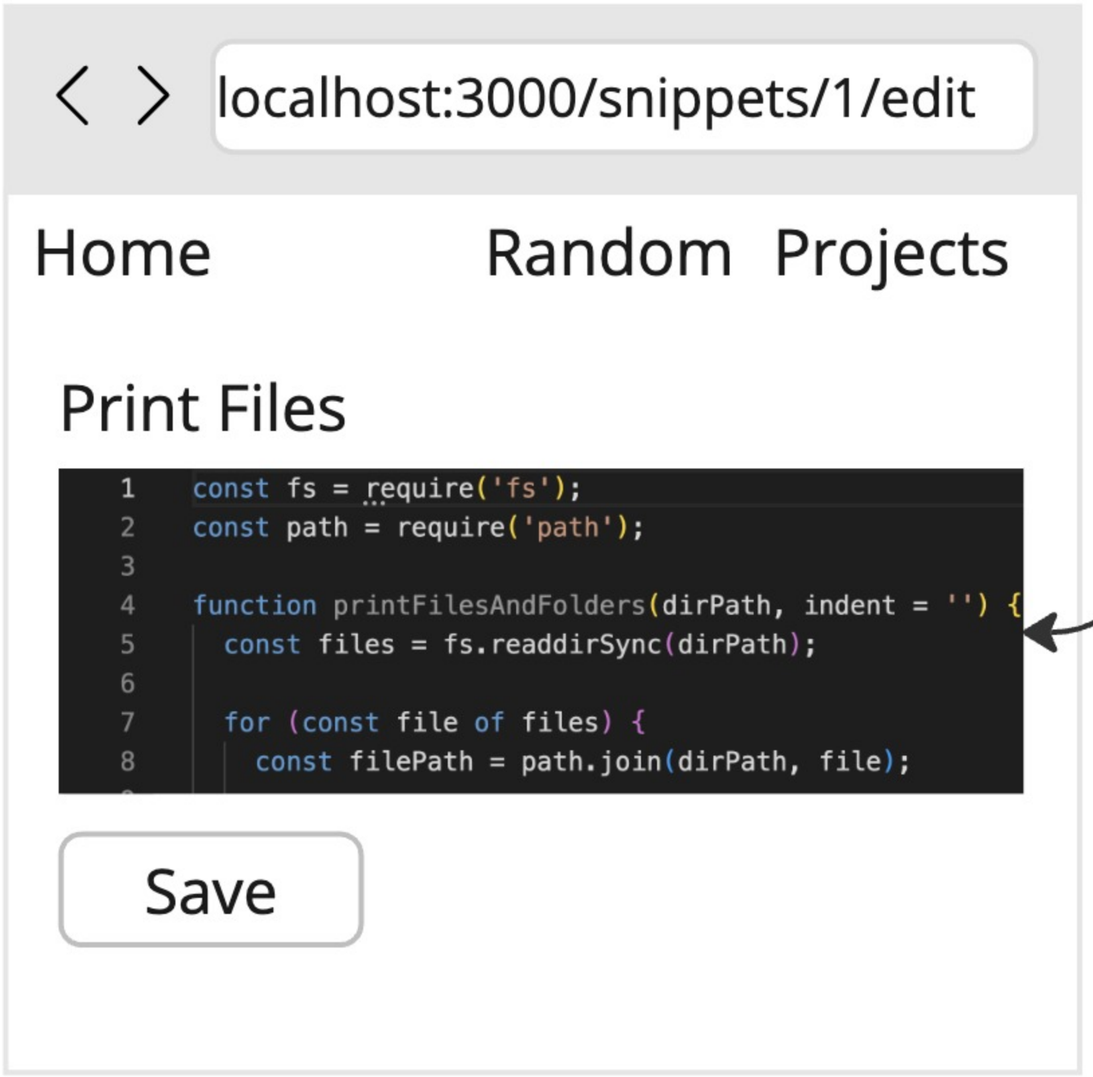
Super simple functions that make a request to the API

Fetch Wrapper

```
export async function request(path, opts) {
    const res = await fetch(path, {
        //...various options
    });
    const data = await res.json();

    // error handling

    // return data
}
```

Calls the 'fetch' function and handles all the tedious parts


fetch
API
Request failed!
Fetch throws
an error
{ status: 400, message: 'Bad Request' }
API returns an error.
Fetch does not
throw an error!

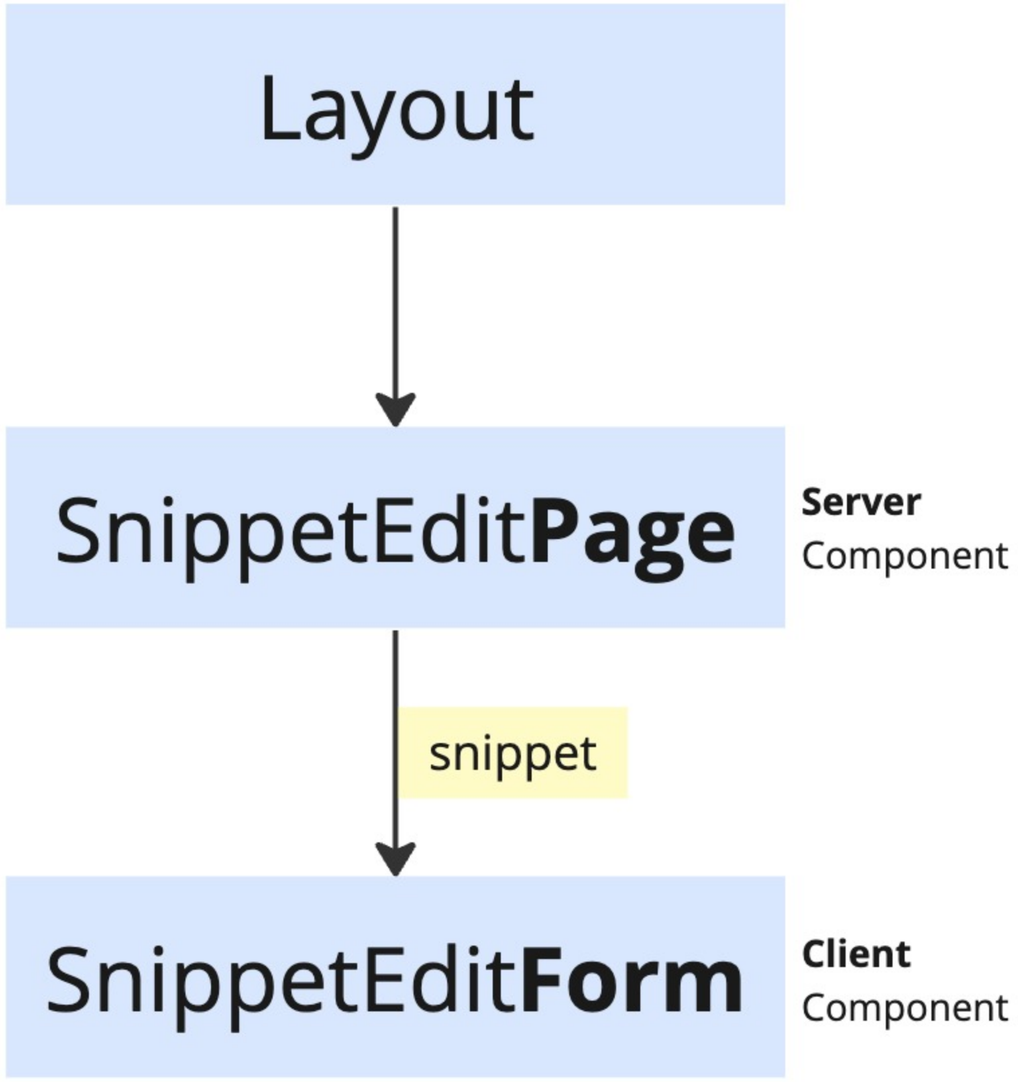


React Monaco Editor

Requires hooks + event handlers to work correctly

We need a **Client Component**

But we still need to fetch data...still need a **Server Component** somewhere!



Server components can display **Client** components

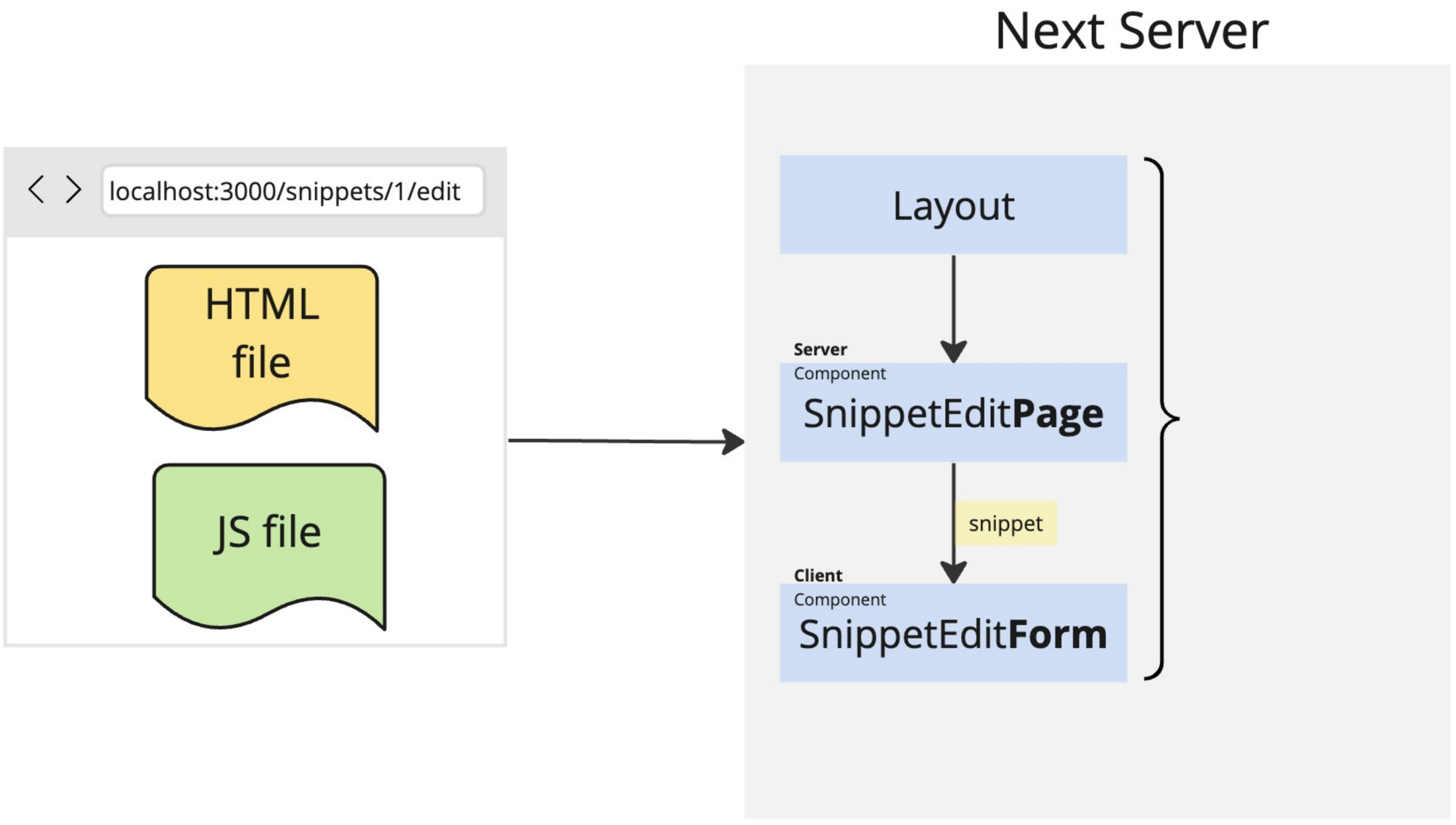
Server components can pass ***serializable*** values to client components

Serializable === can be turned into JSON

Arrays, strings, numbers, plain objects
OK

Classes, instances of classes, functions
NOT OK

There is one exception for passing down functions - we'll see this soon.


Next Server
localhost:3000/snippets/1/edit
HTML file
JS file
Layout
Server
Component
SnippetEditPage
snippet
Client
Component
SnippetEditForm

Multiple ways to **define** them

Option #1

Write 'use server' at the top of a file. All functions exported will be server actions.

```
'use server';

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

    revalidate('...')
}


export async function deleteSnippet(id) {
    await api.editSnippet(id);

    revalidate('...')
}
```

Need to create, delete, modify data?

Server Actions!

Functions that get called from the browser, and run on the server

Two ways to define Server Actions

Many different ways to call them

For right now, I'm going to show you the way that I think you'll most often define + call them

We will look at the other ways of using them for creating + deleting snippets

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');
}
```

localhost:3000

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>
}
```

1 Update some data then revalidate some part of the cache

```
'use server';

export async function updateSnippet(id, code) {
    // Update some data
    await api.editSnippet(id, code);

    // Revalidate to clear cache
    revalidateTag('snippets');
}
```

2 Update some data, revalidate, then navigate the user to another page

```
'use server';

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

    revalidateTag('snippets');
    redirect('/snippets/${id}');
}
```

3 Fetch some data

```
'use server';

export async function fetchSnippet(id) {
    const snippet = await api.getSnippet(id);

    return snippet;
}
```

API file

```ts
import { request } from './request';

export interface Snippet {
  id: number;
  title: string;
  code: string;
}

export function listSnippets() {
  return request<Snippet[]>('/my/snippets');
}

export function listRandomSnippets() {
  return request<Snippet[]>('/my/snippets
      /random');
}

export function getSnippet(id: number) {
  return request<Snippet>(`/my/snippets/${id}`
      );
}

export function editSnippet(id, code) {
    return request('...')
}
```

Server Actions

```ts
'use server';

import * as api from '@/api/snippets';

export async function editSnippet(id, code) {
    const snippet = await api.editSnippet(id,code);

    revalidateTag('snippets');
    redirect(`/snippets/${id}`)
}
```

The API file and Server Actions files kind of look similar - they both modify data!

Should we turn the API file functions into Server Actions?

Probably not.

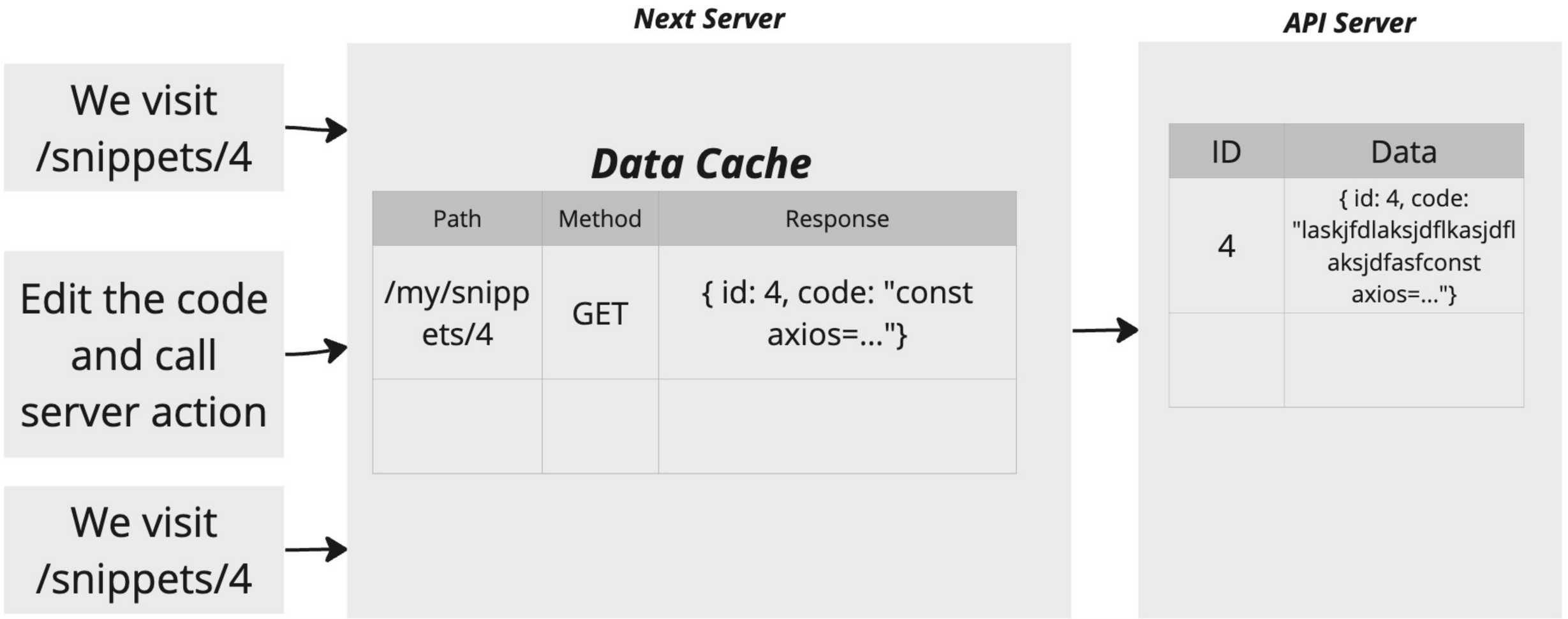
The **API** file contains functions that **access the api**. **Server actions** need to **access the api + manage cache + redirect**

Caching

Next implements caching in several locations.

Can lead to unexpected behavior

Data Cache →	Responses from requests made with '**fetch**' are stored and used across requests.
Request Memoization →	Make two or more requests with 'fetch' during a user's request to your server? Only one is actually executed.
Router Cache →	'Soft' navigation between routes are cached in the browser and reused when a user revisits a page.
Full Route Cache →	At build time, Next decides if your route is ***static*** or ***dynamic***. If it is static, the page is rendered and the result is stored. In production, users are given this pre-rendered result.


We visit /snippets/4
Edit the code and call server action
We visit /snippets/4
Next Server
Data Cache
Path
Method
Response
/my/snippets/4
GET
{ id: 4, code: "const axios=..."}
API Server
ID
Data
4
{ id: 4, code: "laskjfdlaksjdflkasjdflaksjdfasfconst axios=..."}

Help! My next route is rendering with out-of-date data!

There are several ways to control caching

Time-Based	→	Every 60 seconds, ignore the cached response and fetch new data
On-Demand	→	Forcibly purge a cached response
Disable Caching	→	Don't do any caching at all

On-Demand → Forcibly purge a cached response

Option #1

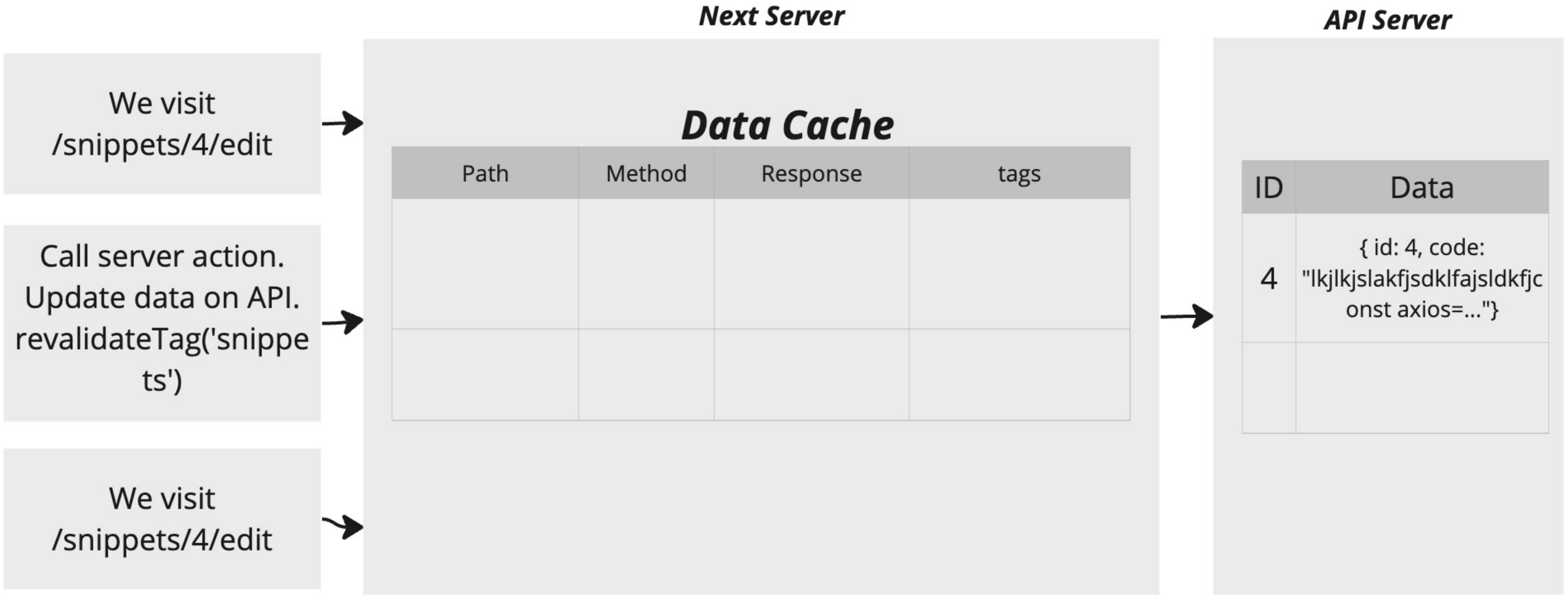
Dump cache for everything in a page

```
import { revalidatePath } from "next/cache";

// When we think data that the '/random'
// route uses has changed...
revalidatePath('/snippets/:id/change');
```

Option #2

Dump cache for all data with a certain tag

```
import { revalidateTag } from "next/cache";

// When we think data with tag 'snippets'
// has changed...
revalidateTag('snippets');


export default async function Page() {
    await fetch('/snippets/random', {
        next: { tags: ['snippets'] }
    })
}
```


Next Server
API Server
We visit /snippets/4/edit
Call server action. Update data on API. revalidateTag('snippets')
We visit /snippets/4/edit
Data Cache
Path
Method
Response
tags
ID
Data
4
{ id: 4, code: "lkjlkjslakfjsdklfajsldkfjc onst axios=..."}