Deep dive into the Umbraco Headless Demo

Phil Whittaker - HiFi

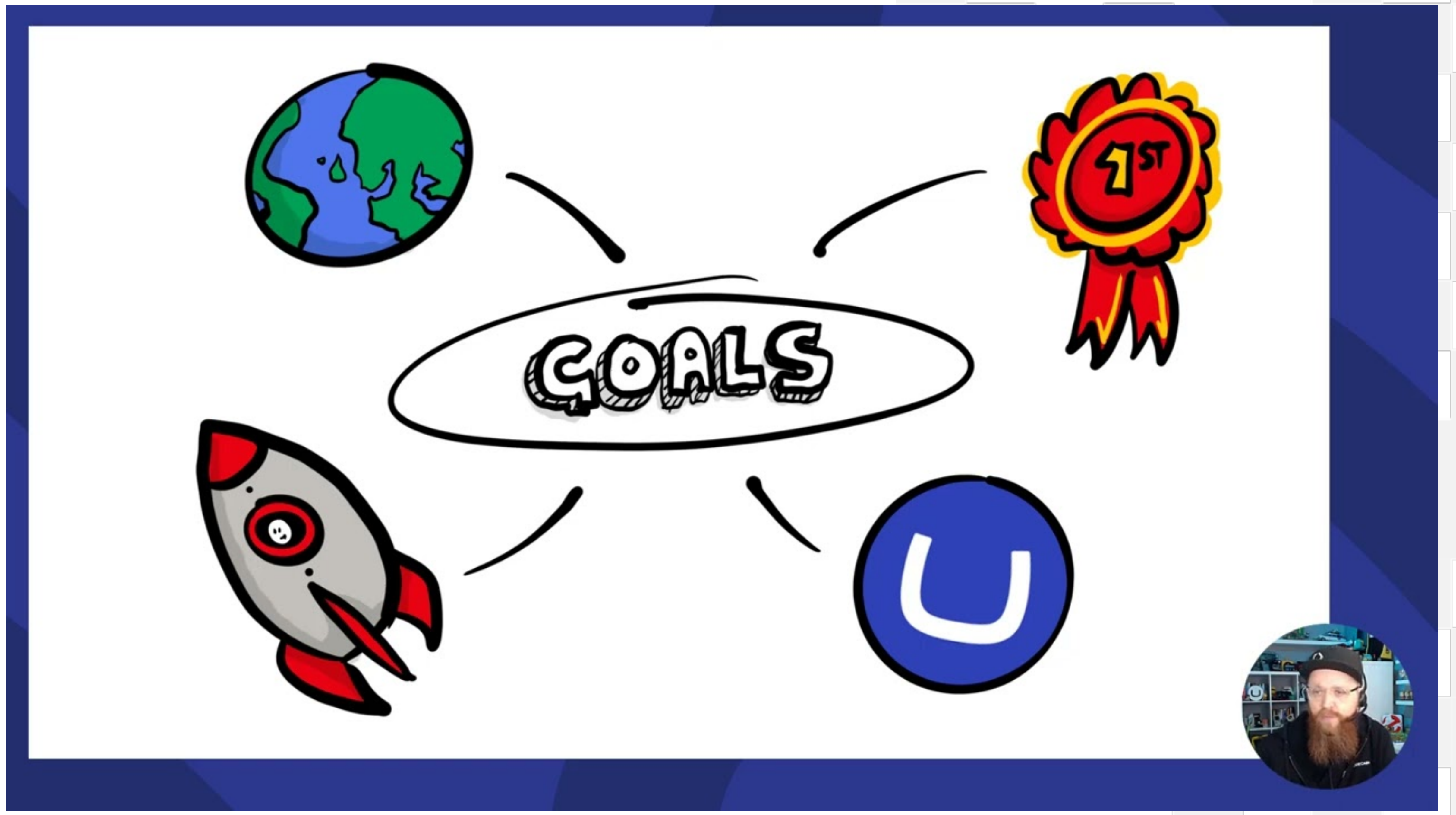
UMBRACO MVP

Umbraco Leeds

Umbraco Manchester

Why have this talk...

youtube.com/watch?v=6BYG2oOZR2I



Who is this talk for

- Umbraco developers (experience of .Net)
- With little or no understanding of NextJs
- Who are interesting in learning more
- And may have found the demo hard to understand

The aim of this talk

Through the lens of the Umbraco.Headless.Demo

- A helping hand to getting started
- Understanding the basics of NextJs
- Be aware of the common pitfalls of NextJs
- Understand the patterns and structures used in the demo
- Dispel the myth that NextJs is hard or just for frontend developers

Getting Started

Typescript is your friend

It’s makes javascript usable
(and a bit more like c#)

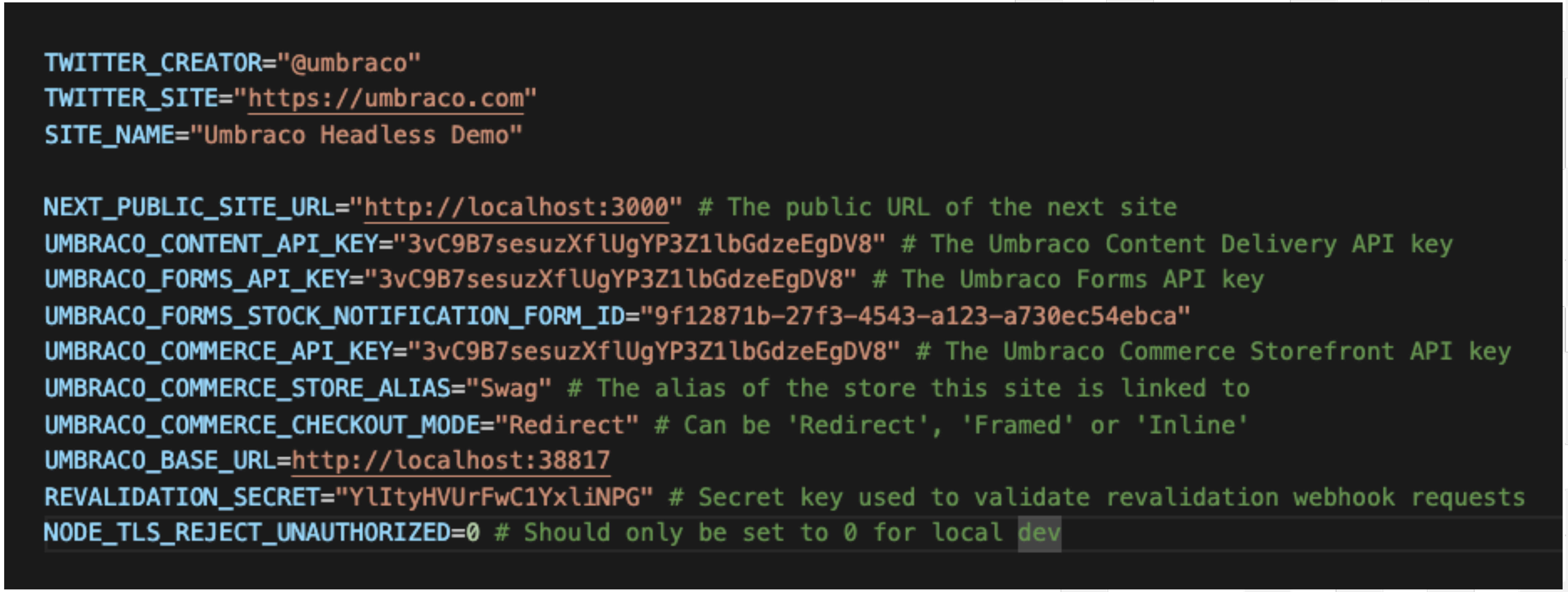
The GitHub repo

- Clone repo from github.com/umbraco/Umbraco.Headless.Demo
- Two branches
  - frontend/main
  - backend/main

Getting started - frontend

(like appSettings)

- Add an env.local file to the root



```
TWITTER_CREATOR="@umbraco"
TWITTER_SITE="https://umbraco.com"
SITE_NAME="Umbraco Headless Demo"

NEXT_PUBLIC_SITE_URL="http://localhost:3000" # The public URL of the next site
UMBRACO_CONTENT_API_KEY="3vC9B7sesuzXflUgYP3Z1lbGdzeEgDV8" # The Umbraco Content Delivery API key
UMBRACO_FORMS_API_KEY="3vC9B7sesuzXflUgYP3Z1lbGdzeEgDV8" # The Umbraco Forms API key
UMBRACO_FORMS_STOCK_NOTIFICATION_FORM_ID="9f12871b-27f3-4543-a123-a730ec54ebca"
UMBRACO_COMMERCE_API_KEY="3vC9B7sesuzXflUgYP3Z1lbGdzeEgDV8" # The Umbraco Commerce Storefront API key
UMBRACO_COMMERCE_STORE_ALIAS="Swag" # The alias of the store this site is linked to
UMBRACO_COMMERCE_CHECKOUT_MODE="Redirect" # Can be 'Redirect', 'Framed' or 'Inline'
UMBRACO_BASE_URL=http://localhost:38817
REVALIDATION_SECRET="YlItyHVUrFwC1YxliNPG" # Secret key used to validate revalidation webhook requests
NODE_TLS_REJECT_UNAUTHORIZED=0 # Should only be set to 0 for local dev
```

Getting started - running locally

- On the backend (http://localhost:38817/umbraco)

  ```
  cd src/Umbraco.Headless.Demo.Web
  dotnet run
  ```

- On the frontend (https://localhost:3000)

  ```
  npm install (first time only)
  npm run dev
  ```

Why headless

- Disconnected solution
- Easier Umbraco upgrades
- Take advantage of SSG / ISR
  state site generation
  incremental static regeneration
- Can take advantage of a mature frontend eco-system
  (Storybook, Typescript, React, Tailwind)

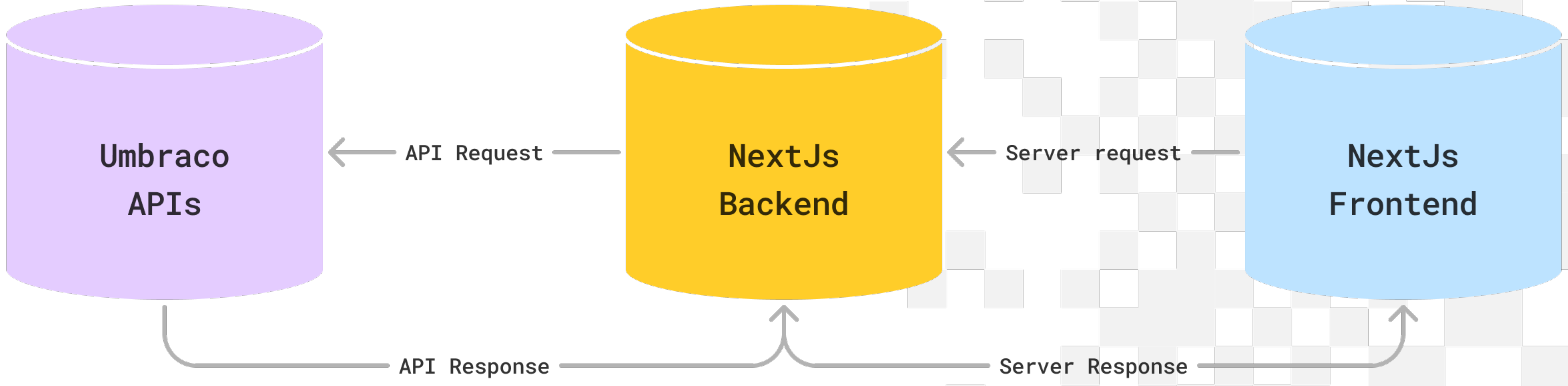
Some NextJs Basics

It helps to understand a few concepts

What is NextJs
And how does it work

- A NodeJs application, built using React
- Client Side / Server Side rendering
- Unique routing solution
- Extensive caching options (dangerous?)
- Can be deployed to Vercel hosted network
- Distributed computing by default

Umbraco + NextJs



The App Router

- Routes as separate folder
- Reserved file names for specific functions

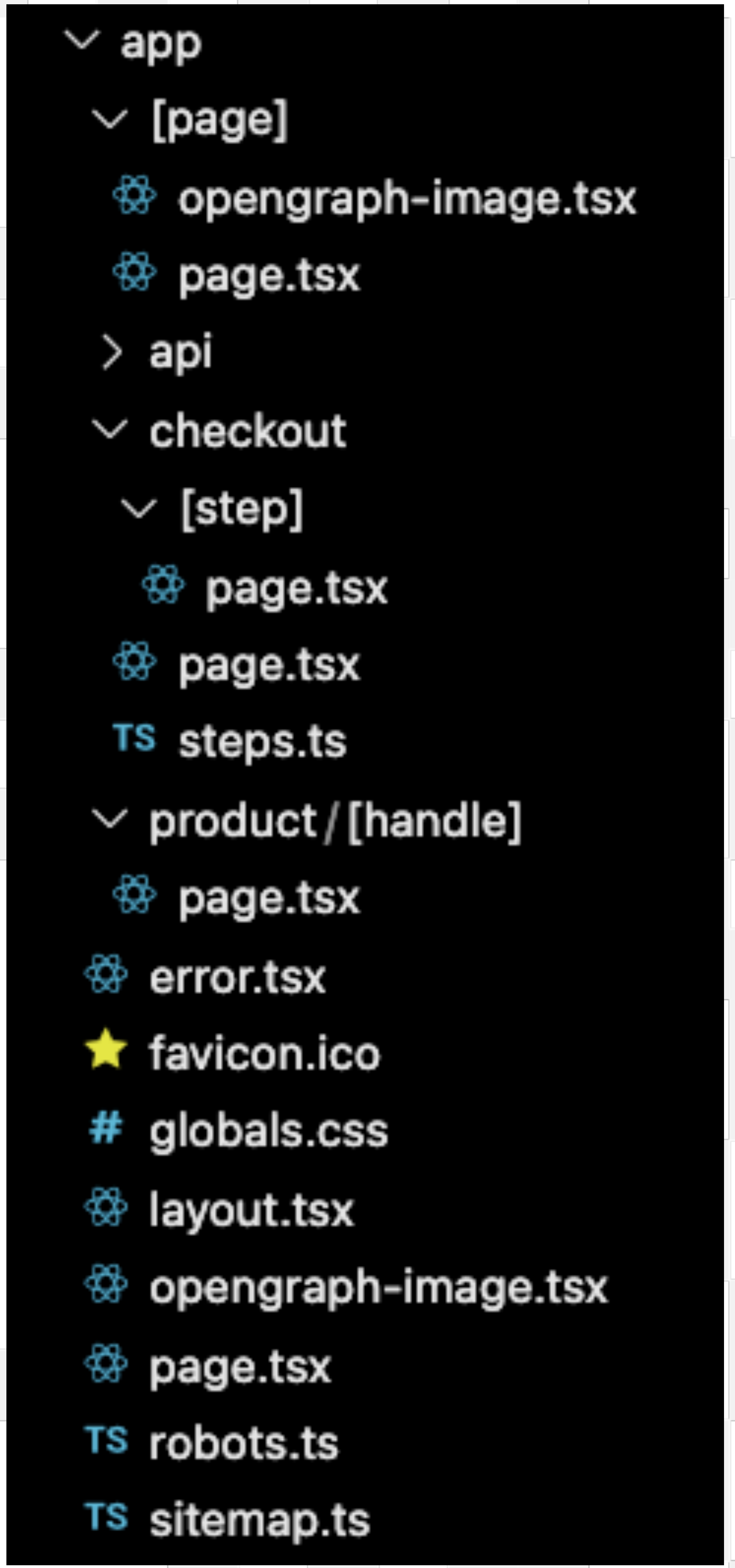
  Page, layout (template, error, loading)
  layouts are like master layouts in MVC

- Other supporting files are allowed

  (feature slicing)

- Dynamic routes, wildcards etc

  Retrieve dynamic pages from Umbraco



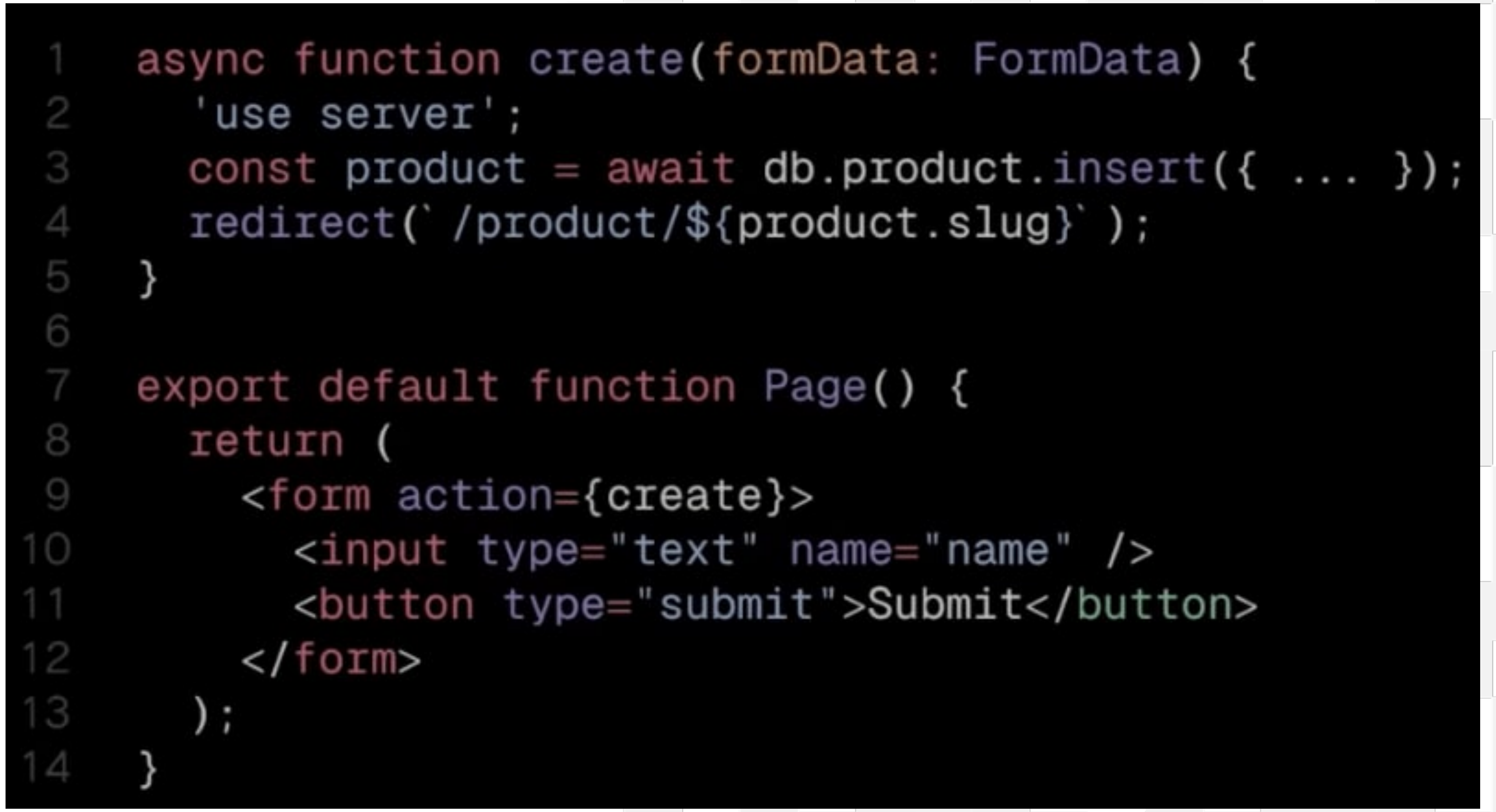
```
app
  [page]
    opengraph-image.tsx
    page.tsx
  api
  checkout
    [step]
      page.tsx
    page.tsx
    steps.ts
  product / [handle]
    page.tsx
  error.tsx
  favicon.ico
  globals.css
  layout.tsx
  opengraph-image.tsx
  page.tsx
  robots.ts
  sitemap.ts
```

Server / Client rendering

- Defaults to server side
  - Can use async functions
  - No client-side hooks
    (useState, useEffect, context providers)
- Optional client-side 'use client';
  - No async functions
  - Child components always client-side

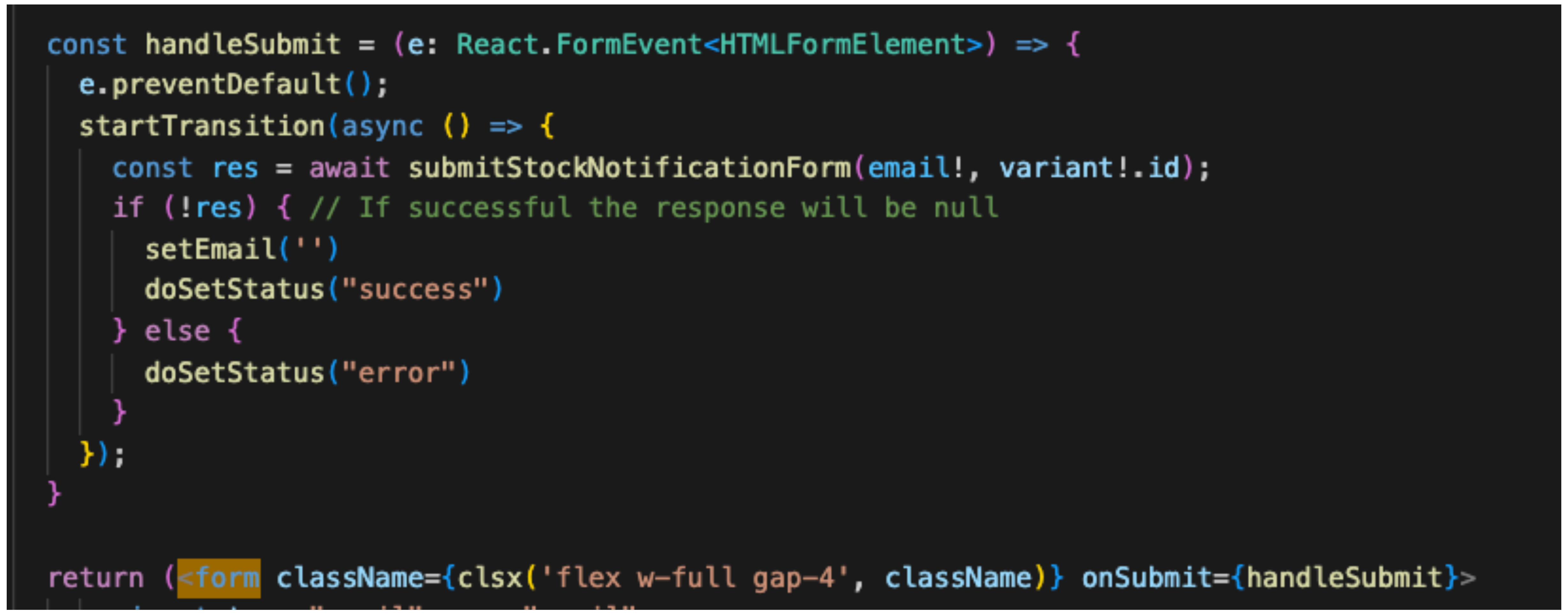
Server Actions

- Similar to surface controllers
- ‘use server’;
- Server side automatically deployed to the edge



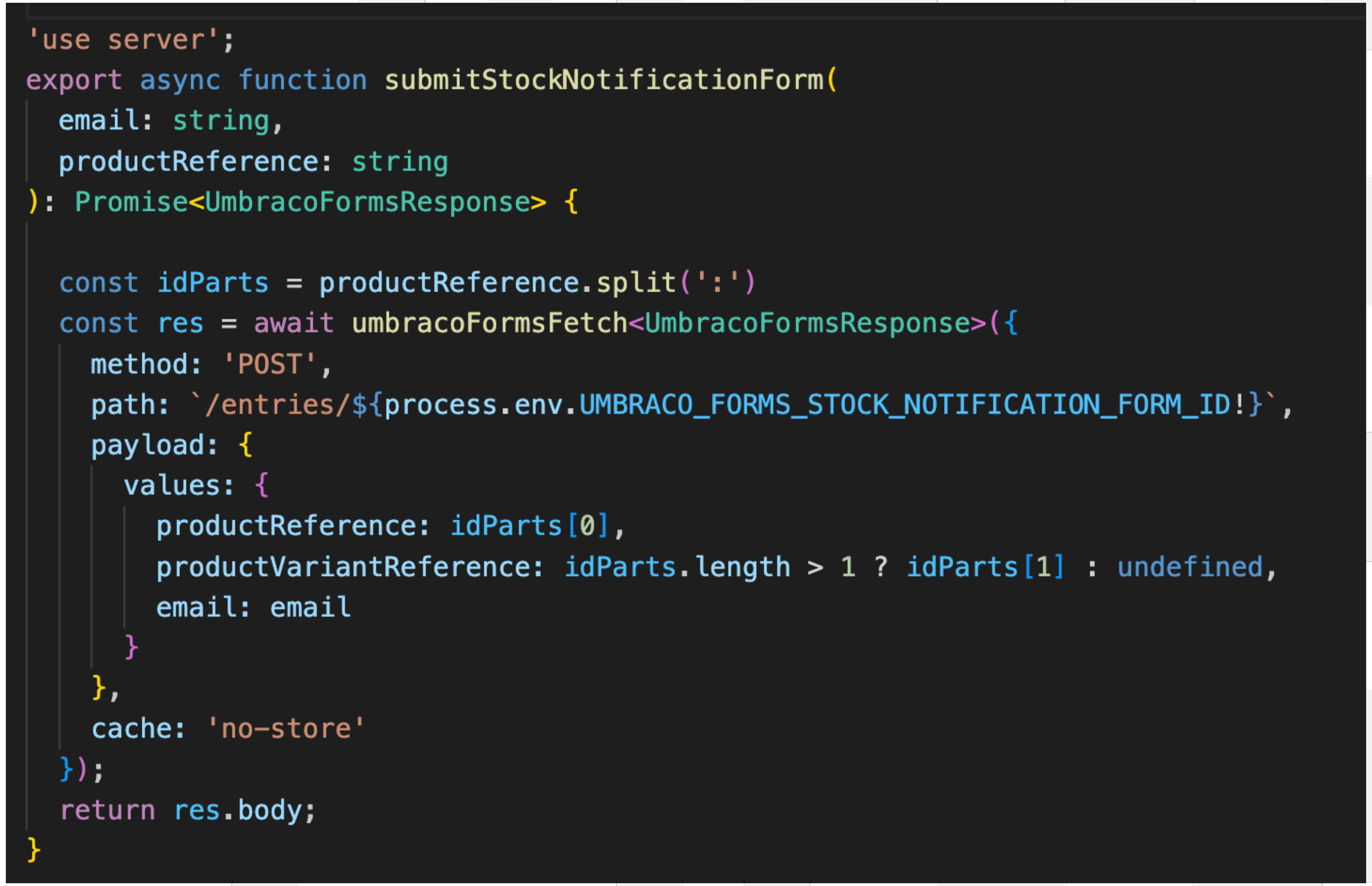
```
async function create(formData: FormData) {
  'use server';
  const product = await db.product.insert({ ... });
  redirect(` /product/${product.slug}` );
}

export default function Page() {
  return (
    <form action={create}>
      <input type="text" name="name" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Server Actions



```
const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {
  e.preventDefault();
  startTransition(async () => {
    const res = await submitStockNotificationForm(email!, variant!.id);
    if (!res) { // If successful the response will be null
      setEmail('')
      doSetStatus("success")
    } else {
      doSetStatus("error")
    }
  });
}

return (<form className={clsx('flex w-full gap-4', className)} onSubmit={handleSubmit}>
```

Client Side

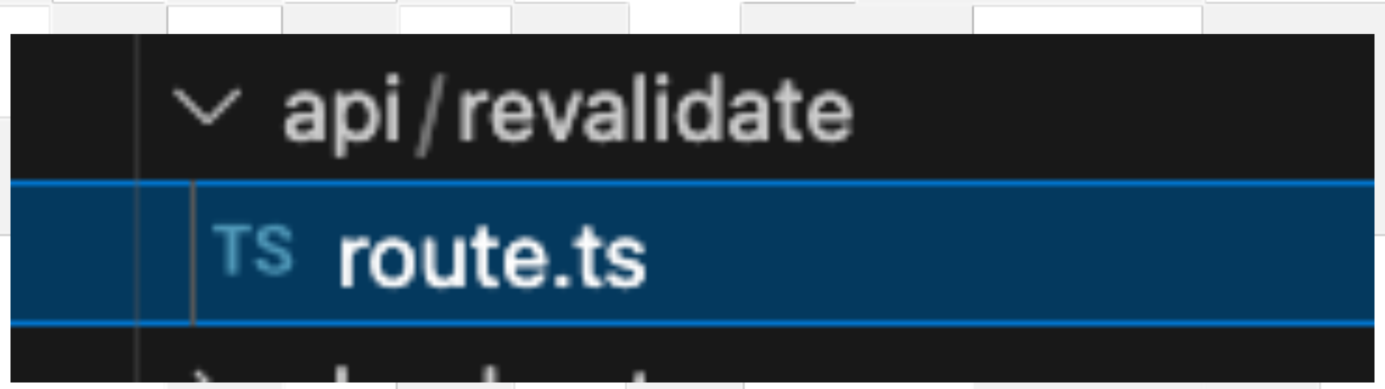


```
'use server';
export async function submitStockNotificationForm(
  email: string,
  productReference: string
): Promise<UmbracoFormsResponse> {

  const idParts = productReference.split(':')
  const res = await umbracoFormsFetch<UmbracoFormsResponse>({
    method: 'POST',
    path: `/entries/${process.env.UMBRACO_FORMS_STOCK_NOTIFICATION_FORM_ID!}`,
    payload: {
      values: {
        productReference: idParts[0],
        productVariantReference: idParts.length > 1 ? idParts[1] : undefined,
        email: email
      }
    },
    cache: 'no-store'
  });
  return res.body;
}
```

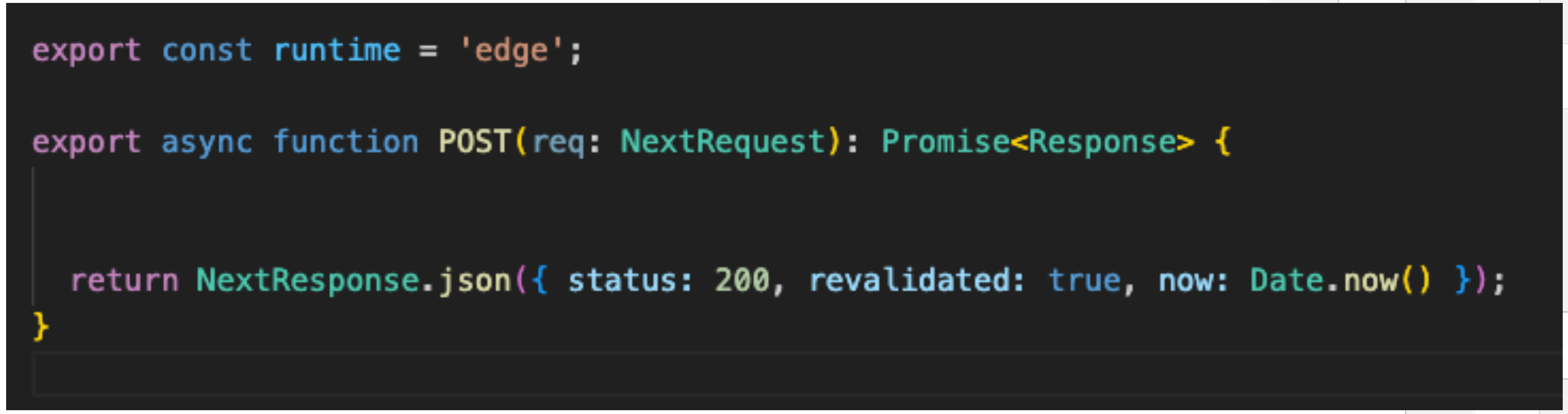
Server Side

Route Handlers

- Similar to Api controllers
- Locked down by default (CORS)
- React to HTTP request (GET, POST etc)
- Automatically deployed to the edge



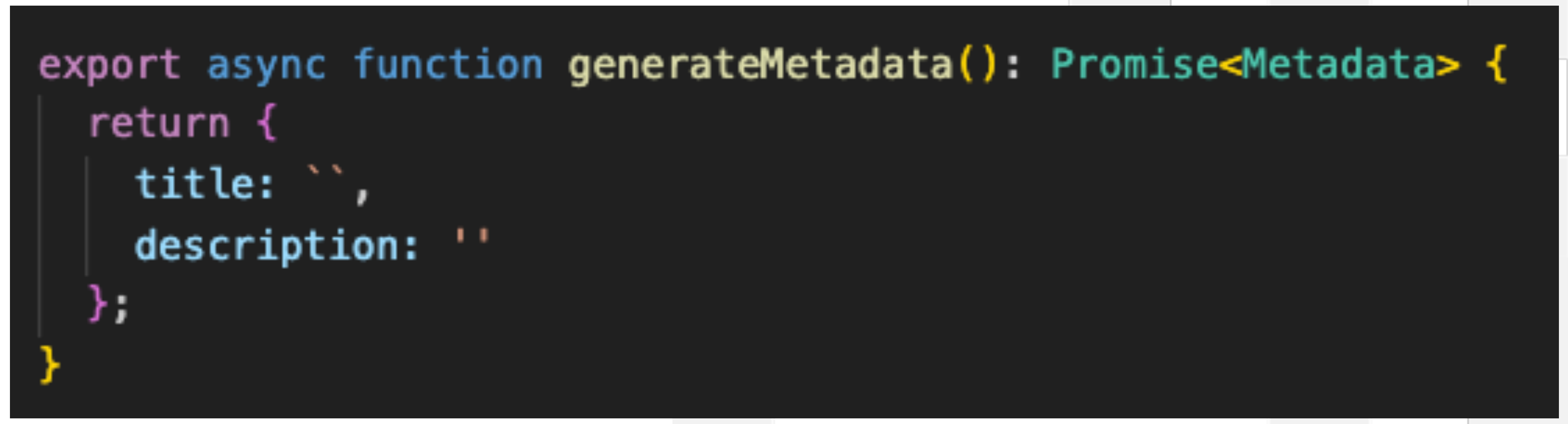
api / revalidate
TS route.ts



```
export const runtime = 'edge';

export async function POST(req: NextRequest): Promise<Response> {


  return NextResponse.json({ status: 200, revalidated: true, now: Date.now() });
}
```

Useful Misc Helpers and Automation

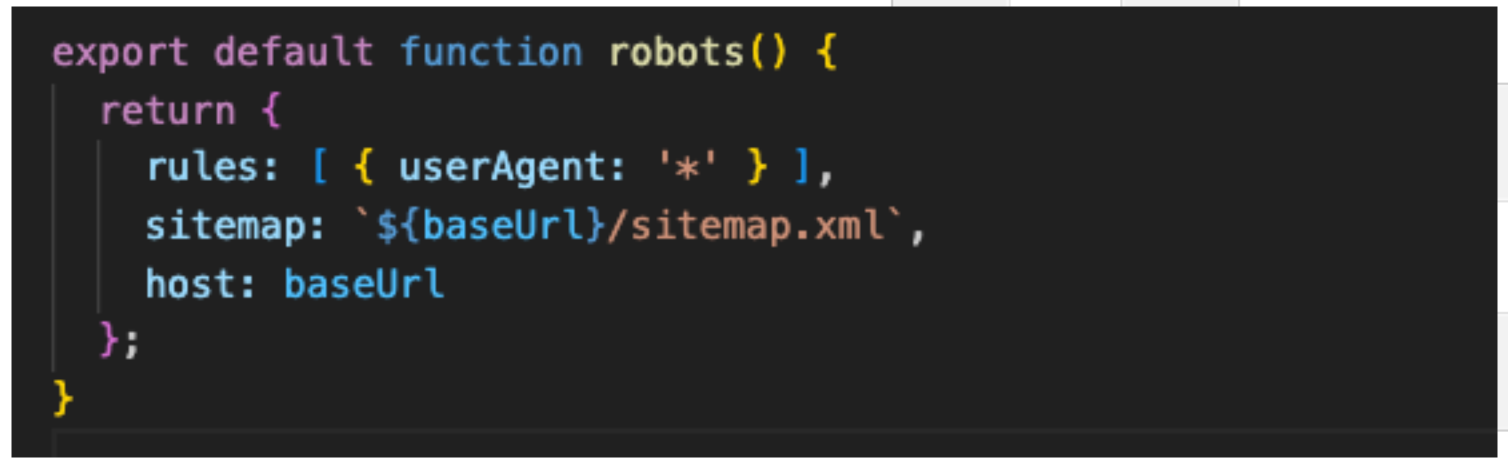
- Meta data, open graph



```
export async function generateMetadata(): Promise<Metadata> {
  return {
    title: ``,
    description: ''
  };
}
```

- Robots txt

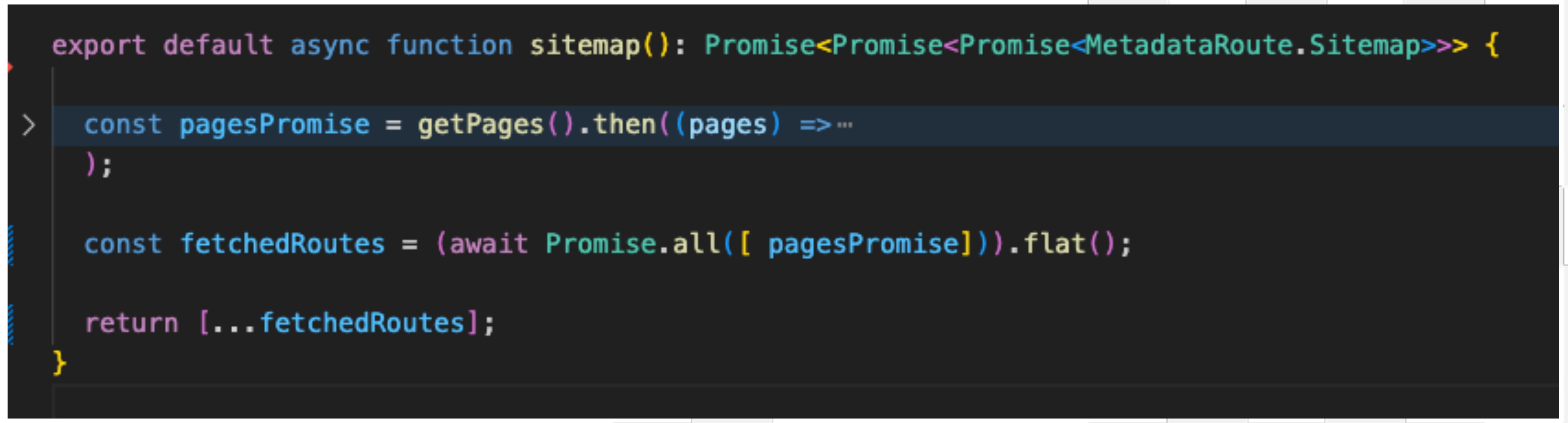
  (robots.tsx)



```
export default function robots() {
  return {
    rules: [ { userAgent: '*' } ],
    sitemap: `${baseUrl}/sitemap.xml`,
    host: baseUrl
  };
}
```

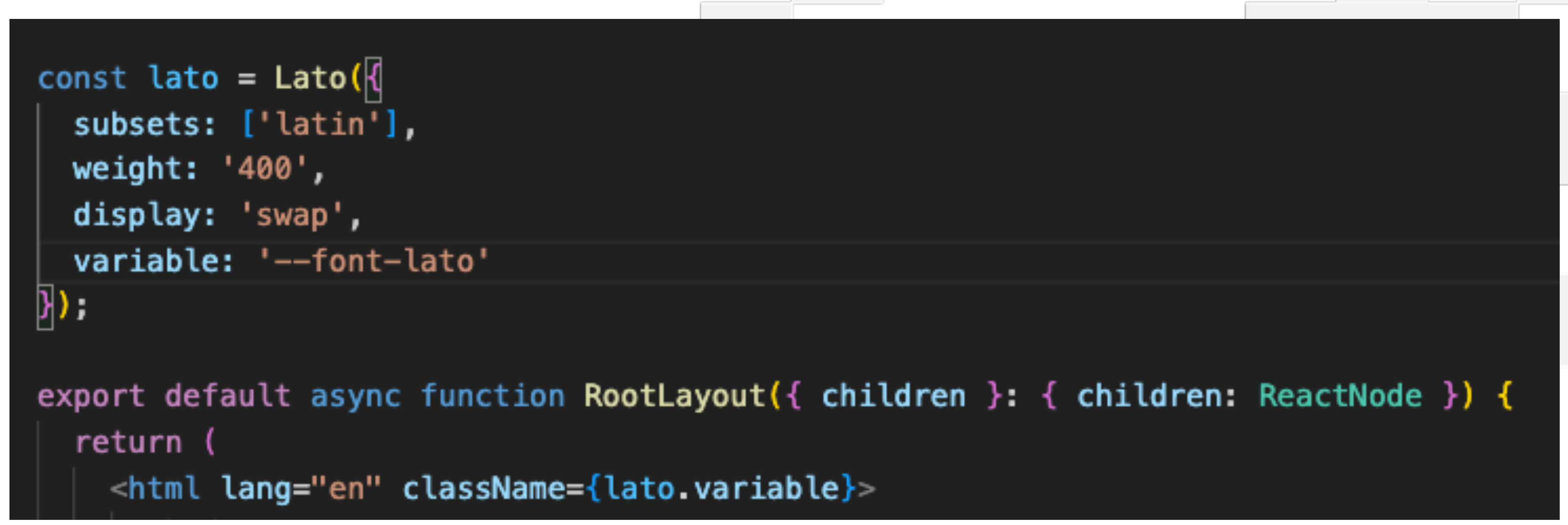
- XML Sitemaps

  (sitemaps.tsx)



```
export default async function sitemap(): Promise<Promise<Promise<MetadataRoute.Sitemap>>> {

  const pagesPromise = getPages().then((pages) =>⋯
  );

  const fetchedRoutes = (await Promise.all([ pagesPromise])).flat();

  return [...fetchedRoutes];
}
```

- Fonts



```
const lato = Lato({
  subsets: ['latin'],
  weight: '400',
  display: 'swap',
  variable: '--font-lato'
});

export default async function RootLayout({ children }: { children: ReactNode }) {
  return (
    <html lang="en" className={lato.variable}>
```

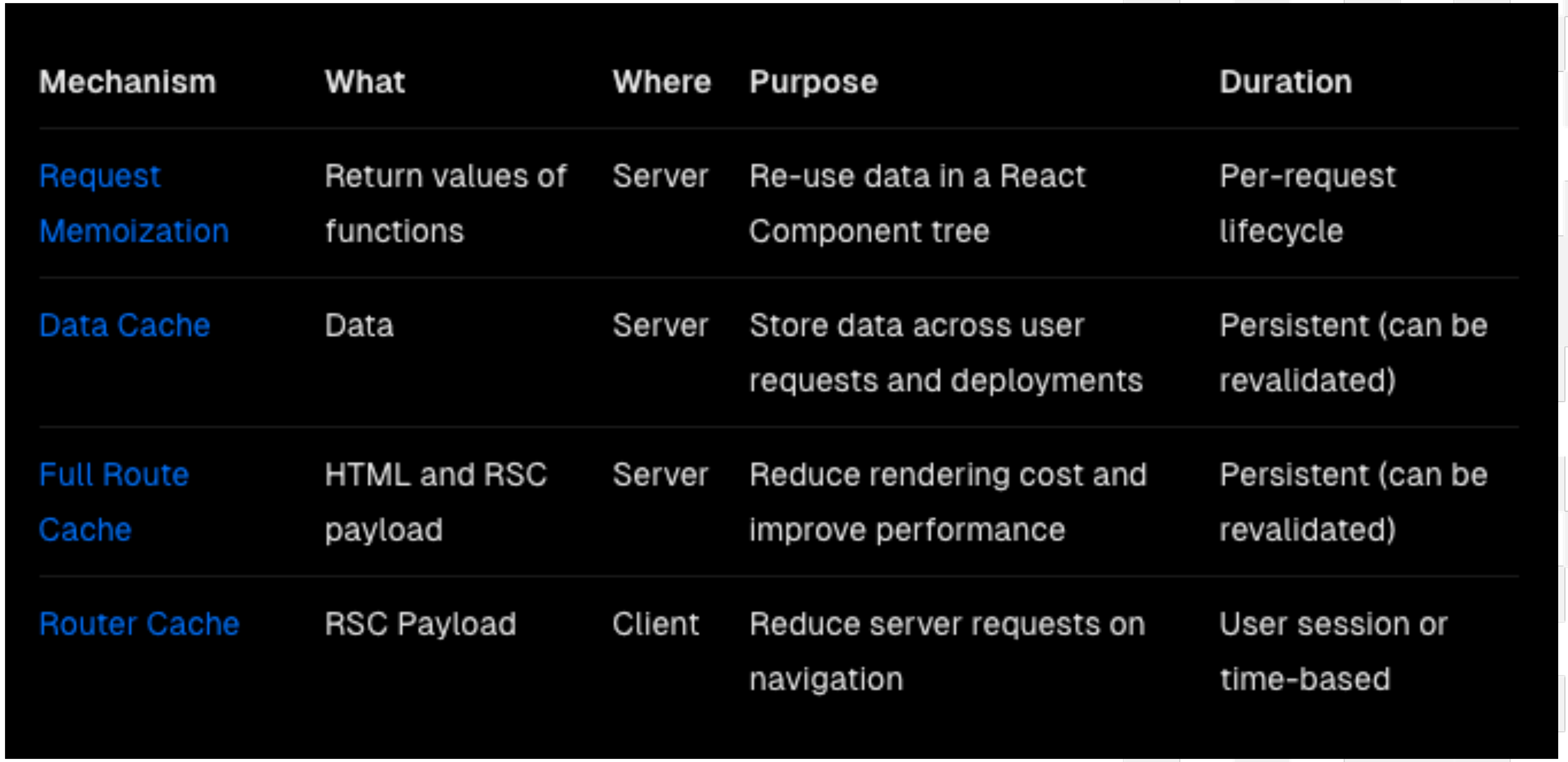
NextJs : common pitfalls

Where everyone goes wrong

Caching

Dev mode behaves very differently to published mode

Caching mechanisms in NextJs

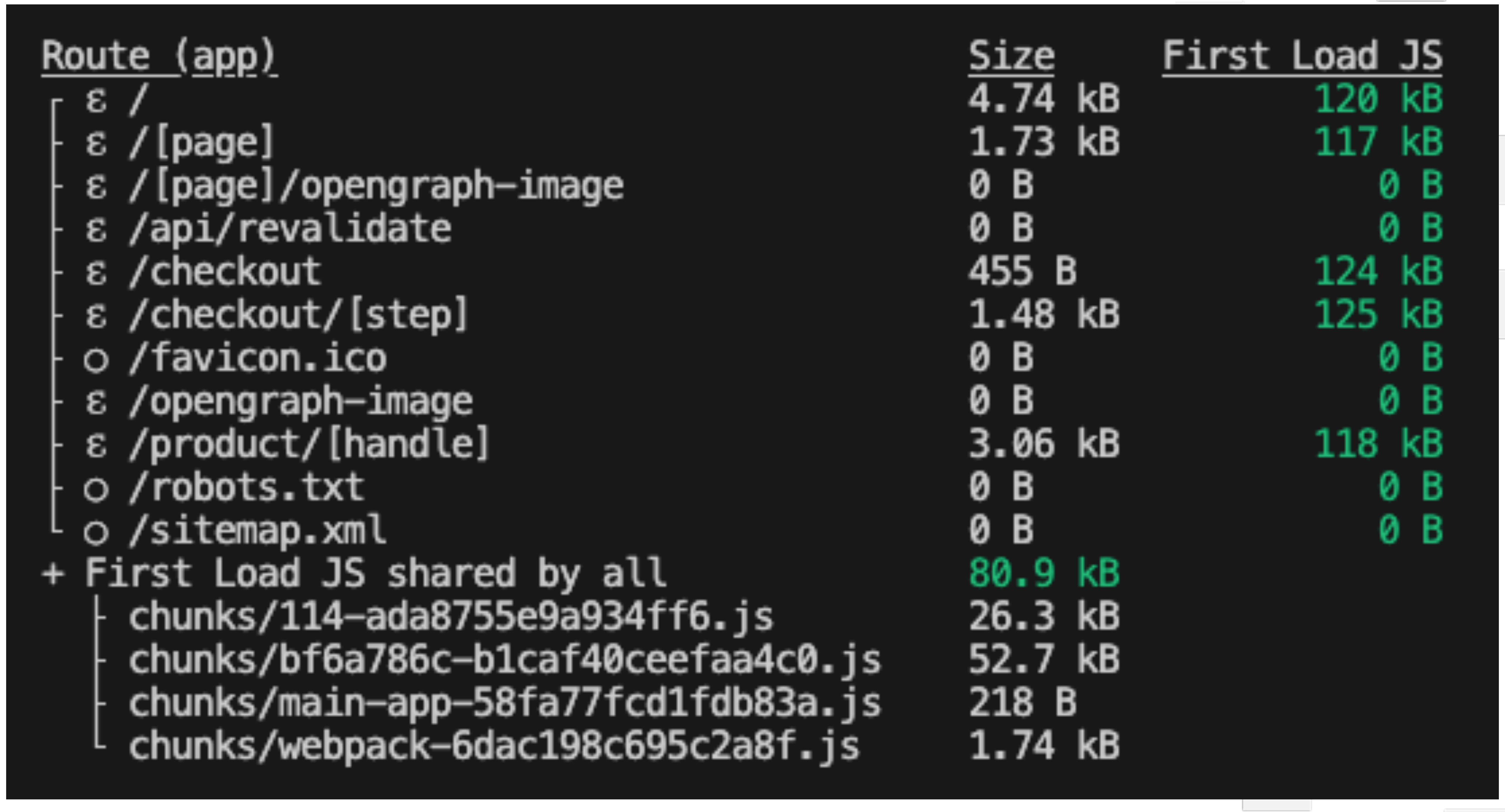


Mechanism	What	Where	Purpose	Duration
Request Memoization	Return values of functions	Server	Re-use data in a React Component tree	Per-request lifecycle
Data Cache	Data	Server	Store data across user requests and deployments	Persistent (can be revalidated)
Full Route Cache	HTML and RSC payload	Server	Reduce rendering cost and improve performance	Persistent (can be revalidated)
Router Cache	RSC Payload	Client	Reduce server requests on navigation	User session or time-based

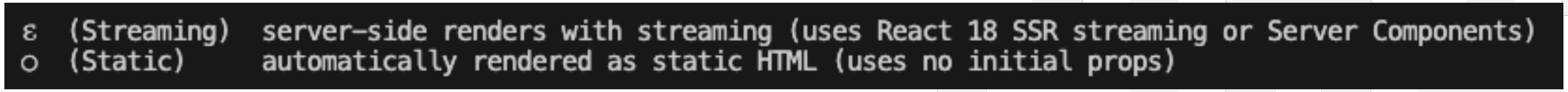
Taken from NextJs docs

Build and start

- Npm run build



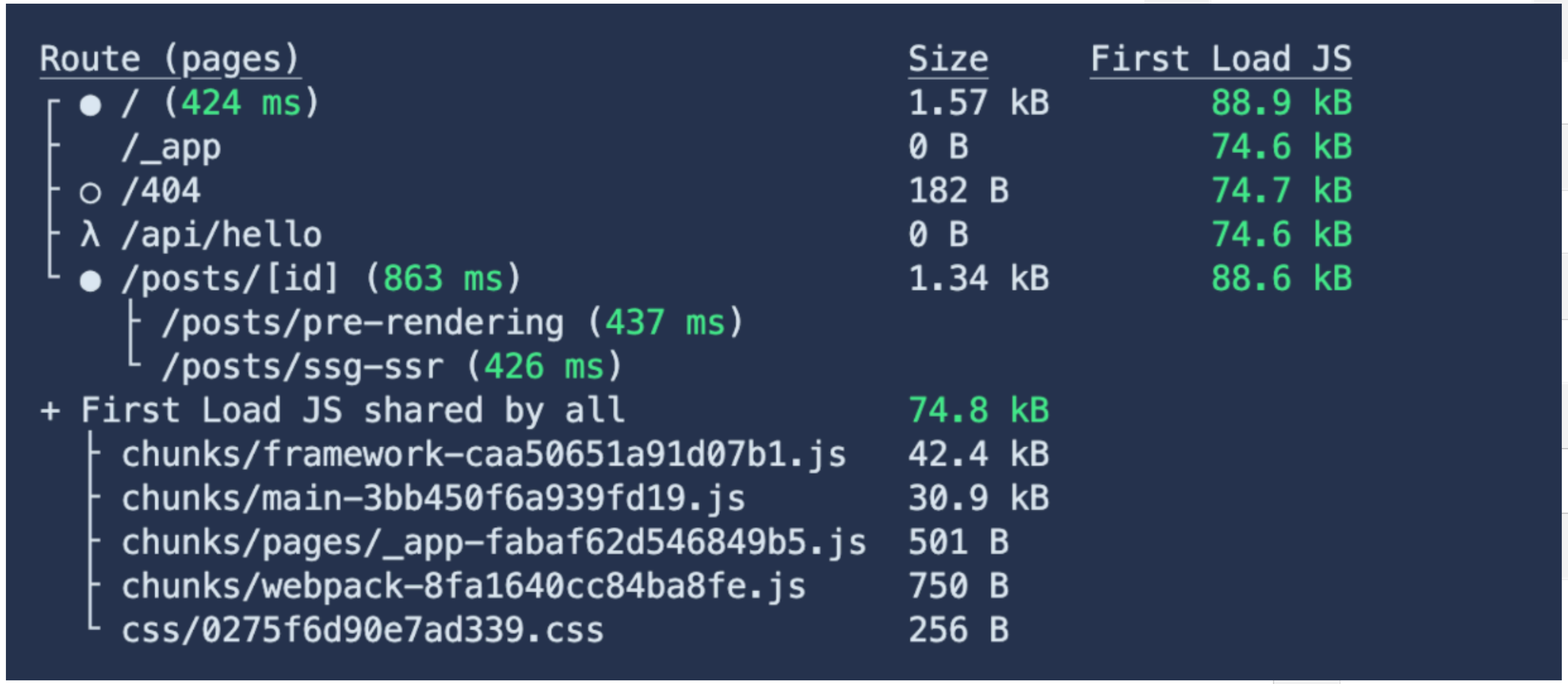
```
Route (app)                                   Size     First Load JS
┌ ε /                                         4.74 kB        120 kB
├ ε /[page]                                   1.73 kB        117 kB
├ ε /[page]/opengraph-image                   0 B              0 B
├ ε /api/revalidate                           0 B              0 B
├ ε /checkout                                 455 B          124 kB
├ ε /checkout/[step]                          1.48 kB        125 kB
├ ○ /favicon.ico                              0 B              0 B
├ ε /opengraph-image                          0 B              0 B
├ ε /product/[handle]                         3.06 kB        118 kB
├ ○ /robots.txt                               0 B              0 B
└ ○ /sitemap.xml                              0 B              0 B
+ First Load JS shared by all                 80.9 kB
  ├ chunks/114-ada8755e9a934ff6.js            26.3 kB
  ├ chunks/bf6a786c-b1caf40ceefaa4c0.js       52.7 kB
  ├ chunks/main-app-58fa77fcd1fdb83a.js       218 B
  └ chunks/webpack-6dac198c695c2a8f.js        1.74 kB
```



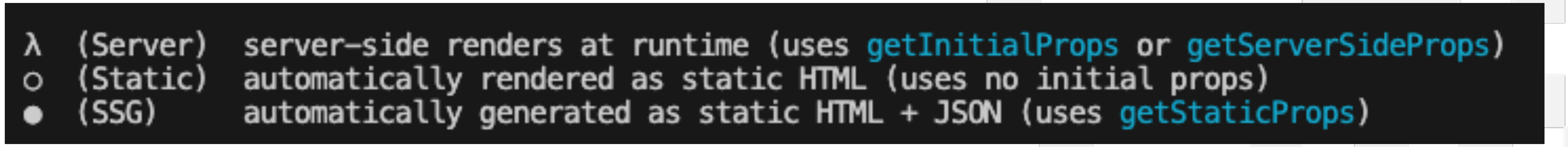
```
ε  (Streaming)  server-side renders with streaming (uses React 18 SSR streaming or Server Components)
○  (Static)     automatically rendered as static HTML (uses no initial props)
```

Build and start

- Npm run build



```
Route (pages)                              Size     First Load JS
┌ ● / (424 ms)                             1.57 kB        88.9 kB
├   /_app                                  0 B            74.6 kB
├ ○ /404                                   182 B          74.7 kB
├ λ /api/hello                             0 B            74.6 kB
└ ● /posts/[id] (863 ms)                   1.34 kB        88.6 kB
    ├ /posts/pre-rendering (437 ms)
    └ /posts/ssg-ssr (426 ms)
+ First Load JS shared by all              74.8 kB
  ├ chunks/framework-caa50651a91d07b1.js   42.4 kB
  ├ chunks/main-3bb450f6a939fd19.js        30.9 kB
  ├ chunks/pages/_app-fabaf62d546849b5.js  501 B
  ├ chunks/webpack-8fa1640cc84ba8fe.js     750 B
  └ css/0275f6d90e7ad339.css               256 B
```

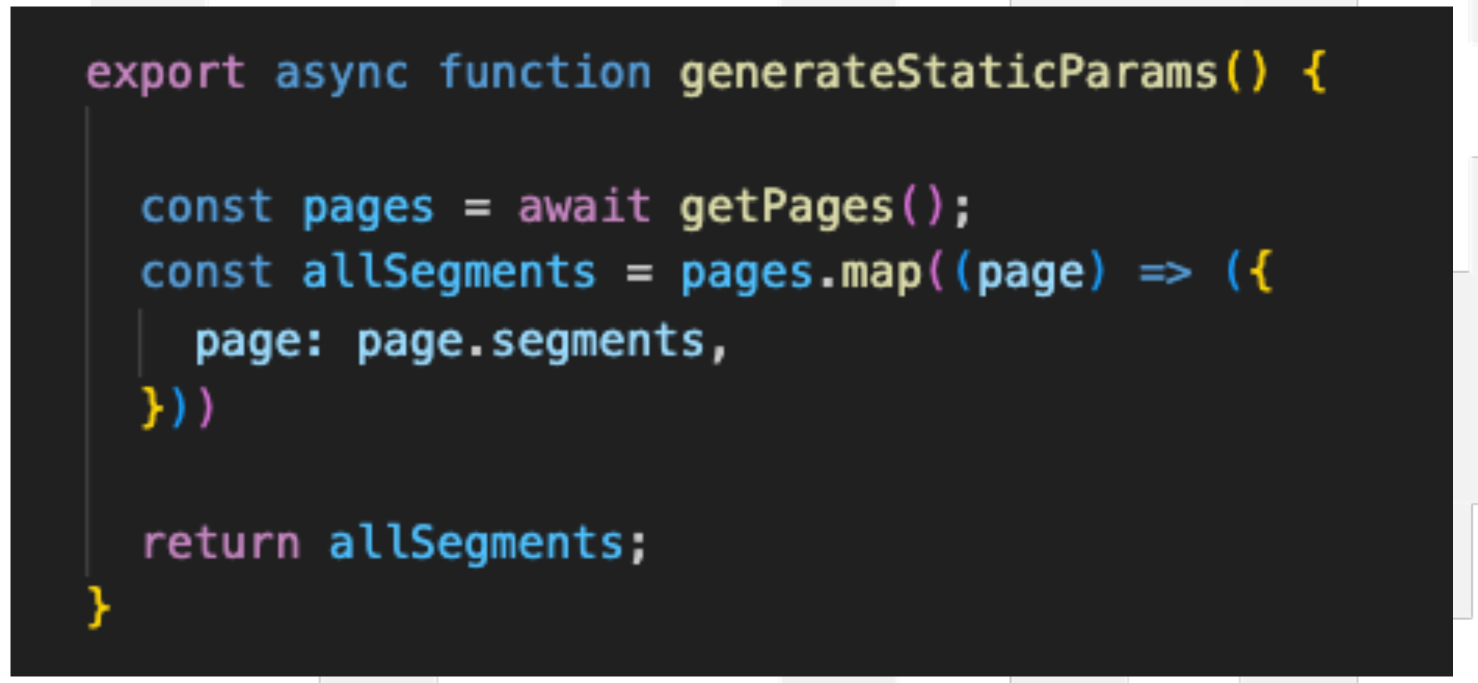


```
λ  (Server)  server-side renders at runtime (uses getInitialProps or getServerSideProps)
○  (Static)  automatically rendered as static HTML (uses no initial props)
●  (SSG)     automatically generated as static HTML + JSON (uses getStaticProps)
```

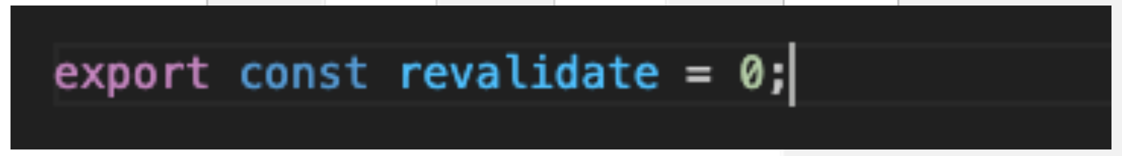
Causes of forced SSR

- Calling dynamic functions or variables on a route
  (cookies or query strings)
- Using dynamic routes
  ('some/[pages]')
- Explicitly setting caching off in a page
- Calling fetch in a page (or it's child components) with caching turned off

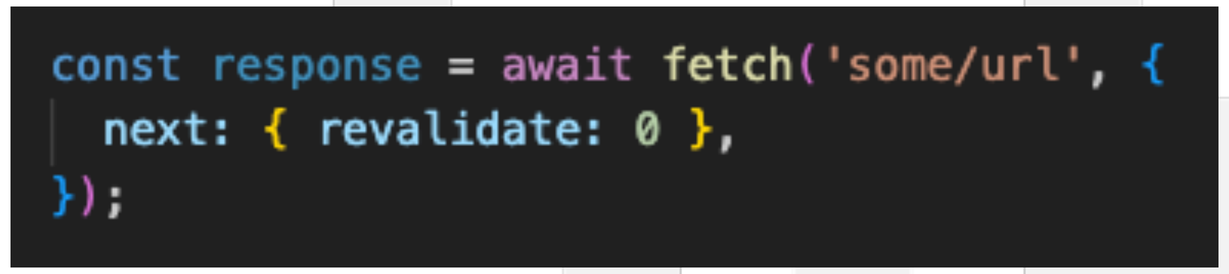
/app/products/[handle]/page.tsx



```
export async function generateStaticParams() {

  const pages = await getPages();
  const allSegments = pages.map((page) => ({
    page: page.segments,
  }))

  return allSegments;
}
```



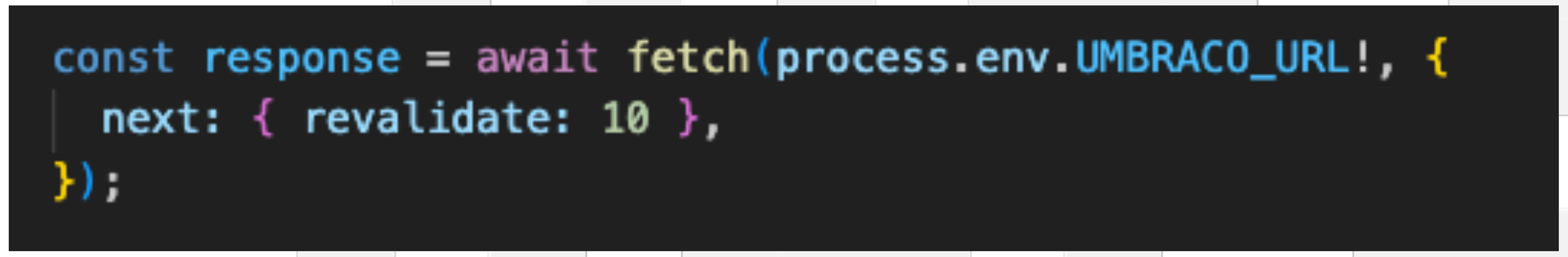
```
export const revalidate = 0;
```



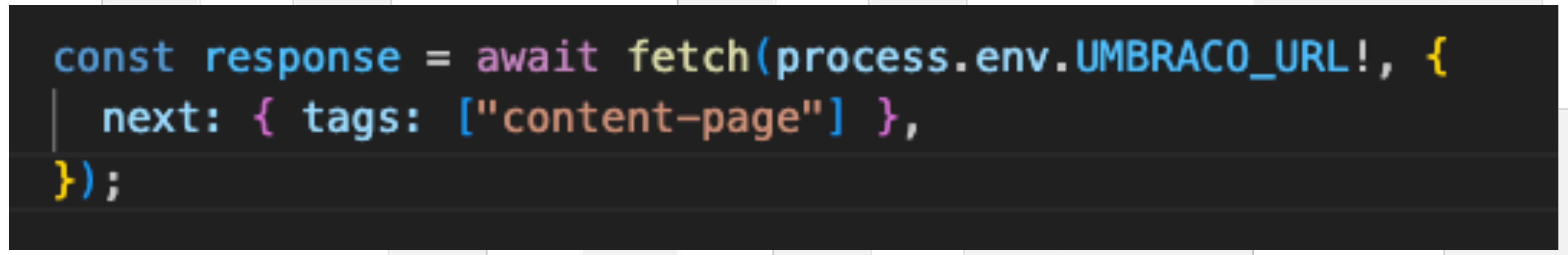
```
const response = await fetch('some/url', {
  next: { revalidate: 0 },
});
```

Revalidation of SSG

- Time based
- On Demand
  - By Tag
  - By Path



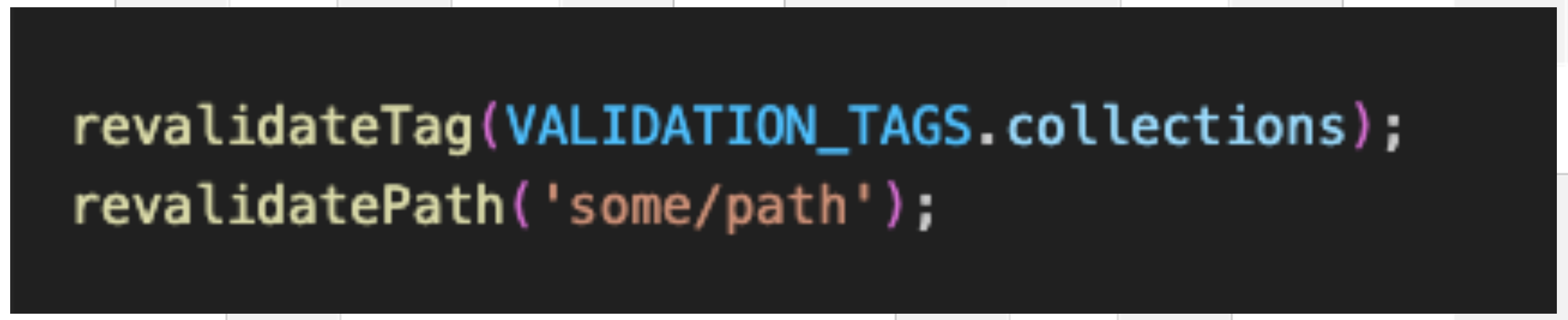
```
const response = await fetch(process.env.UMBRACO_URL!, {
  next: { revalidate: 10 },
});
```



```
const response = await fetch(process.env.UMBRACO_URL!, {
  next: { tags: ["content-page"] },
});
```

- Marking for revalidation

  can be called anywhere; server action or api call



```
revalidateTag(VALIDATION_TAGS.collections);
revalidatePath('some/path');
```

- Common mistake

  (full page caching - navigation

More info on revalidation planning
https://www.udemy.com/course/next-js-the-complete-developers-guide/

Umbraco API's

Swagger Docs

All the details are here

https://localhost:44381/umbraco/swagger/index.html

Structures and examples

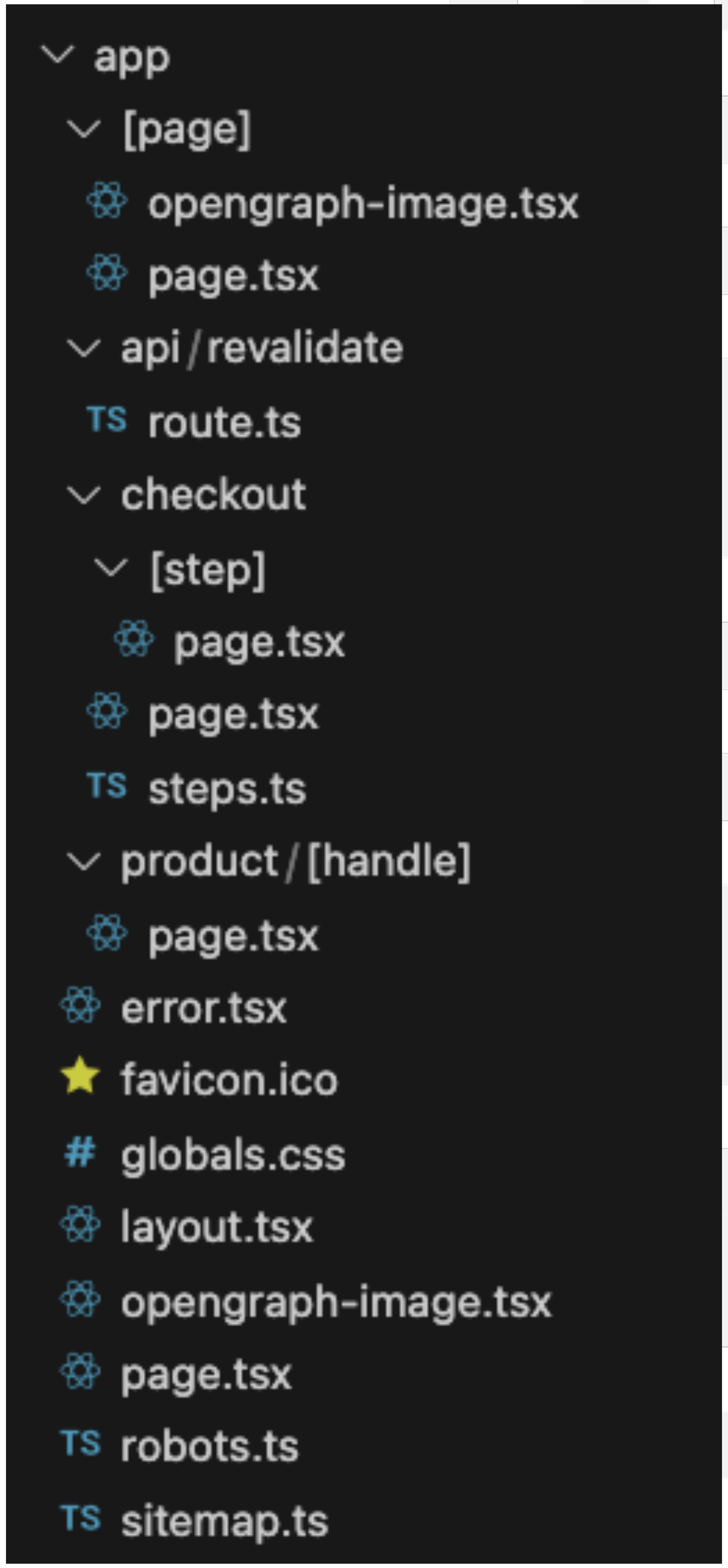
/app/

/components/

/lib/umbraco (alongside types)

App directory

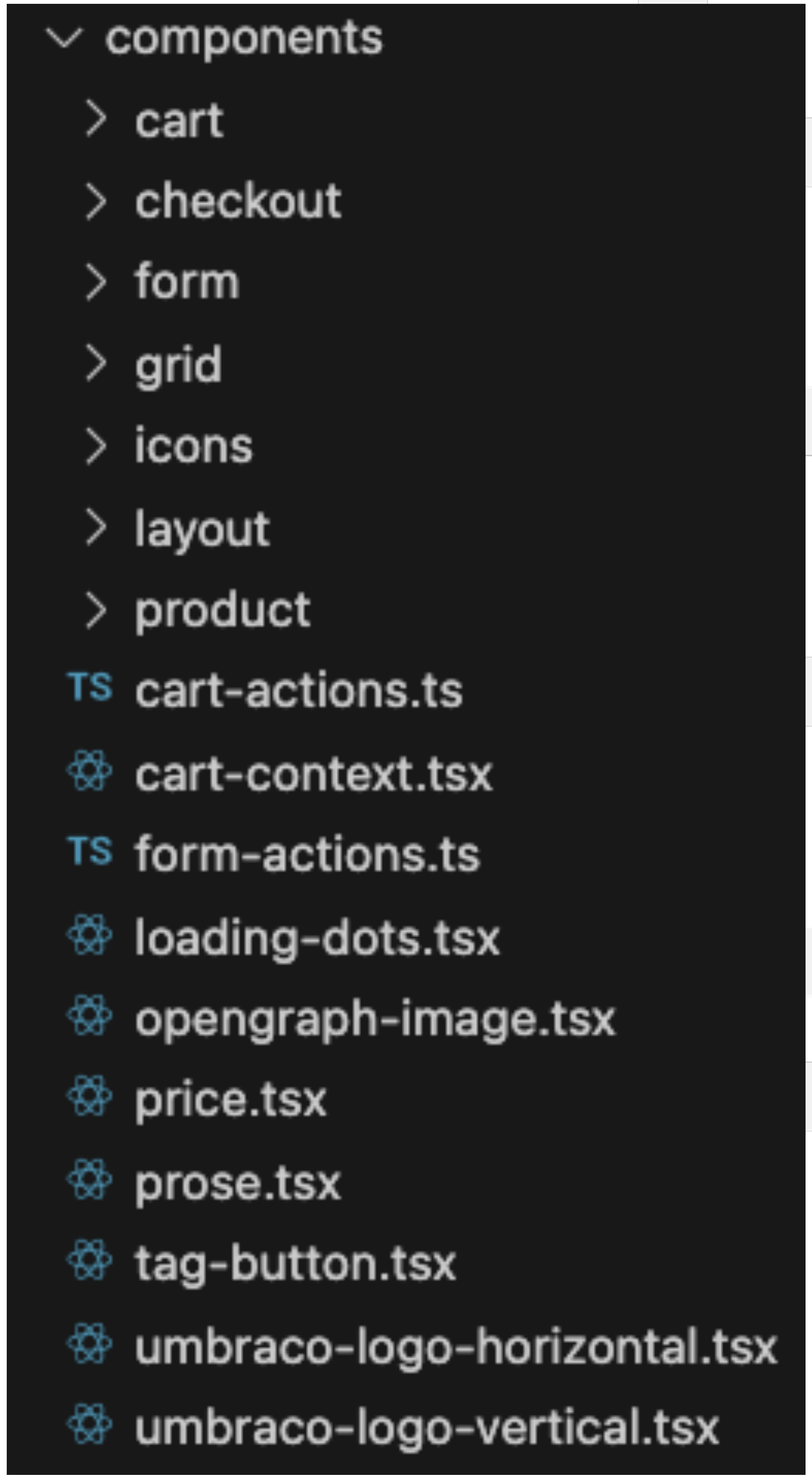
(views)



```
app
  [page]
    opengraph-image.tsx
    page.tsx
  api / revalidate
    route.ts
  checkout
    [step]
      page.tsx
    page.tsx
    steps.ts
  product / [handle]
    page.tsx
  error.tsx
  favicon.ico
  globals.css
  layout.tsx
  opengraph-image.tsx
  page.tsx
  robots.ts
  sitemap.ts
```

Components

(partials / viewcomponents)



```
components
  cart
  checkout
  form
  grid
  icons
  layout
  product
  cart-actions.ts
  cart-context.tsx
  form-actions.ts
  loading-dots.tsx
  opengraph-image.tsx
  price.tsx
  prose.tsx
  tag-button.tsx
  umbraco-logo-horizontal.tsx
  umbraco-logo-vertical.tsx
```

Defining Structure (lib/umbraco/types.ts)

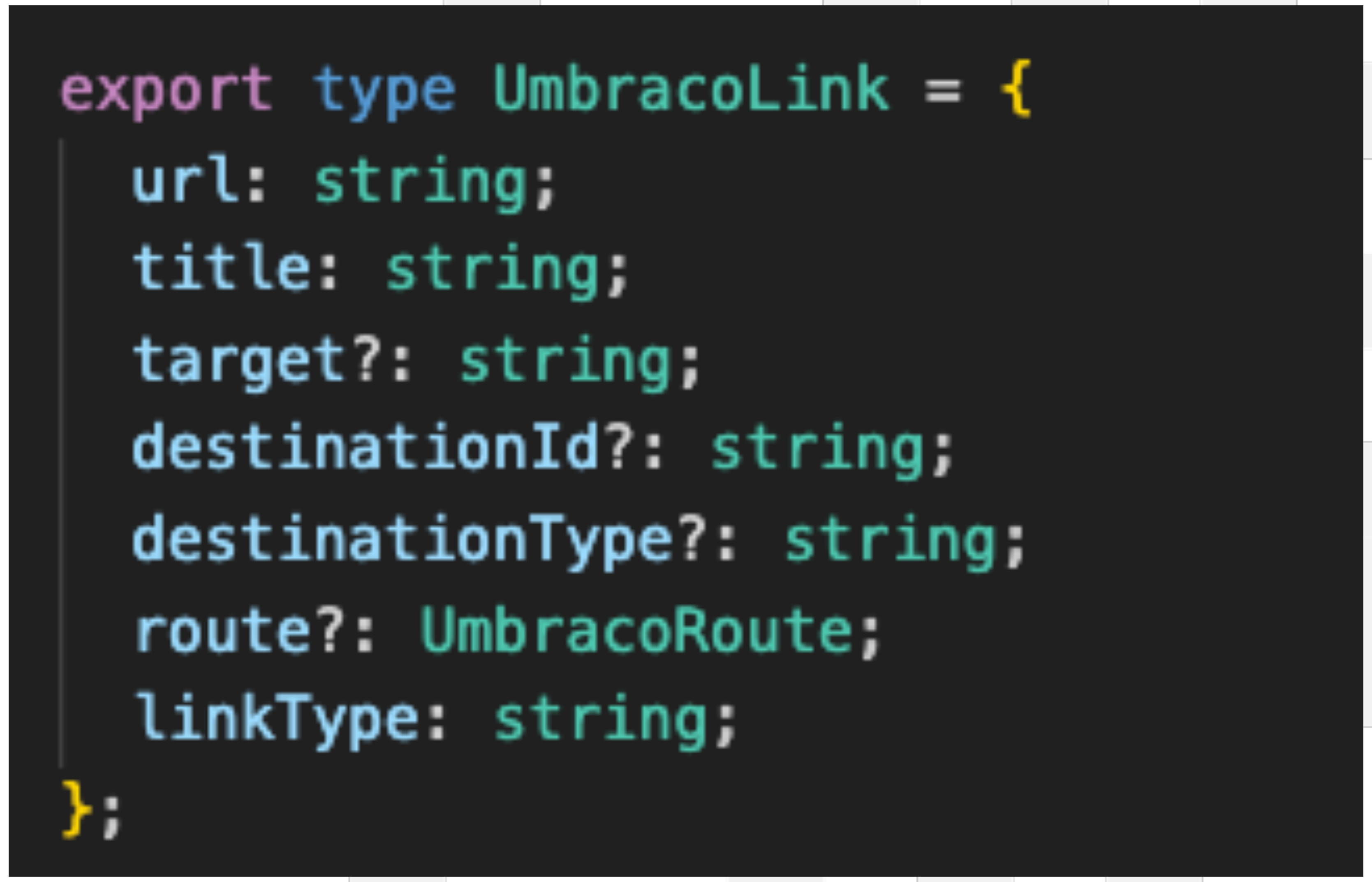
- Umbraco api model structure

  content delivery
  commerce
  forms

  used by mapping functions to create vm's

- Some internal Models

  Cart, CartItem
  Image, Manu
  etc...



```
export type UmbracoLink = {
  url: string;
  title: string;
  target?: string;
  destinationId?: string;
  destinationType?: string;
  route?: UmbracoRoute;
  linkType: string;
};
```

Could now use Delivery Api Extensions to generate model builder like swagger
Then user open api codgen to generate structures in typescript
https://marketplace.umbraco.com/package/umbraco.community.deliveryapiextensions

Calling Umbraco (lib/umbraco/index.ts)

- Base umbracoFetch

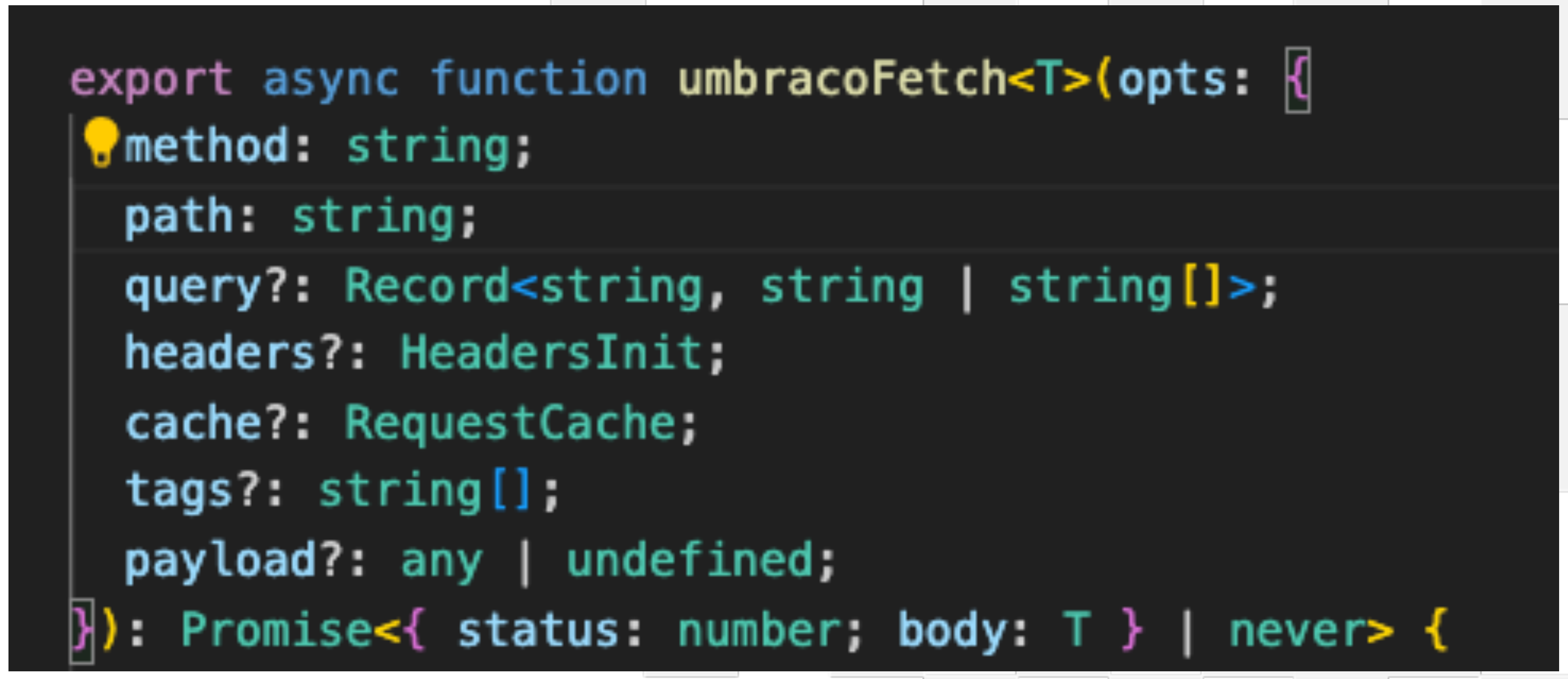
  umbracoContentFetch
  umbracoCommerceFetch
  umbracoFormsFetch

- Model to Vm mapping

  (reshaping)

- Specific API calls

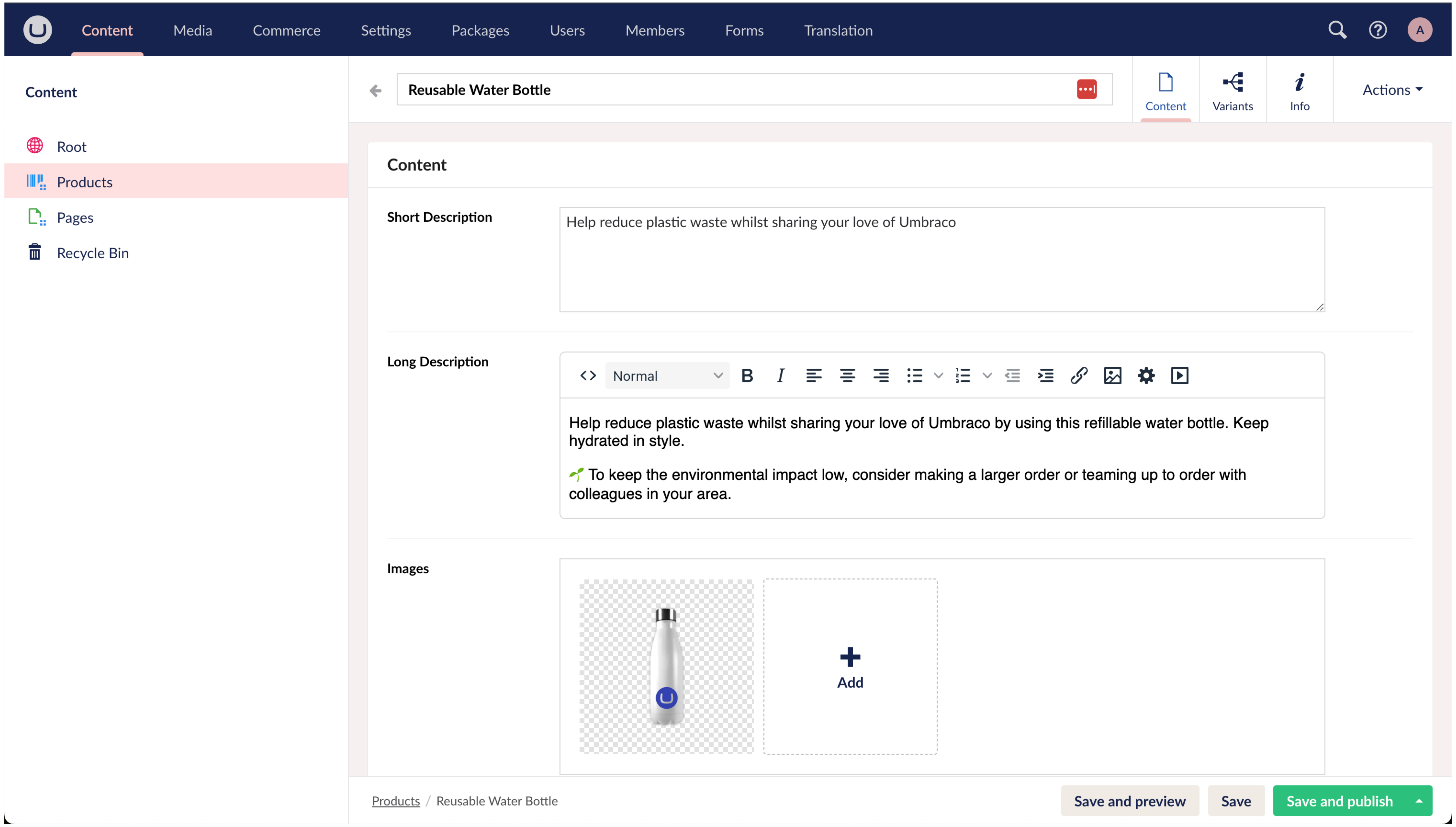
  removeFromCart, updateCartItems, updateCart, getCart
  getMenu
  getProduct(s), getPageRecommendations, getProductTags
  getPage(s)
  etc... including checkout functions & form functions



```
export async function umbracoFetch<T>(opts: {
  method: string;
  path: string;
  query?: Record<string, string | string[]>;
  headers?: HeadersInit;
  cache?: RequestCache;
  tags?: string[];
  payload?: any | undefined;
}): Promise<{ status: number; body: T } | never> {
```

Products

Product Content



Content

Short Description: Help reduce plastic waste whilst sharing your love of Umbraco

Long Description:

Help reduce plastic waste whilst sharing your love of Umbraco by using this refillable water bottle. Keep hydrated in style.

🌱 To keep the environmental impact low, consider making a larger order or teaming up to order with colleagues in your area.

Images

Products / Reusable Water Bottle

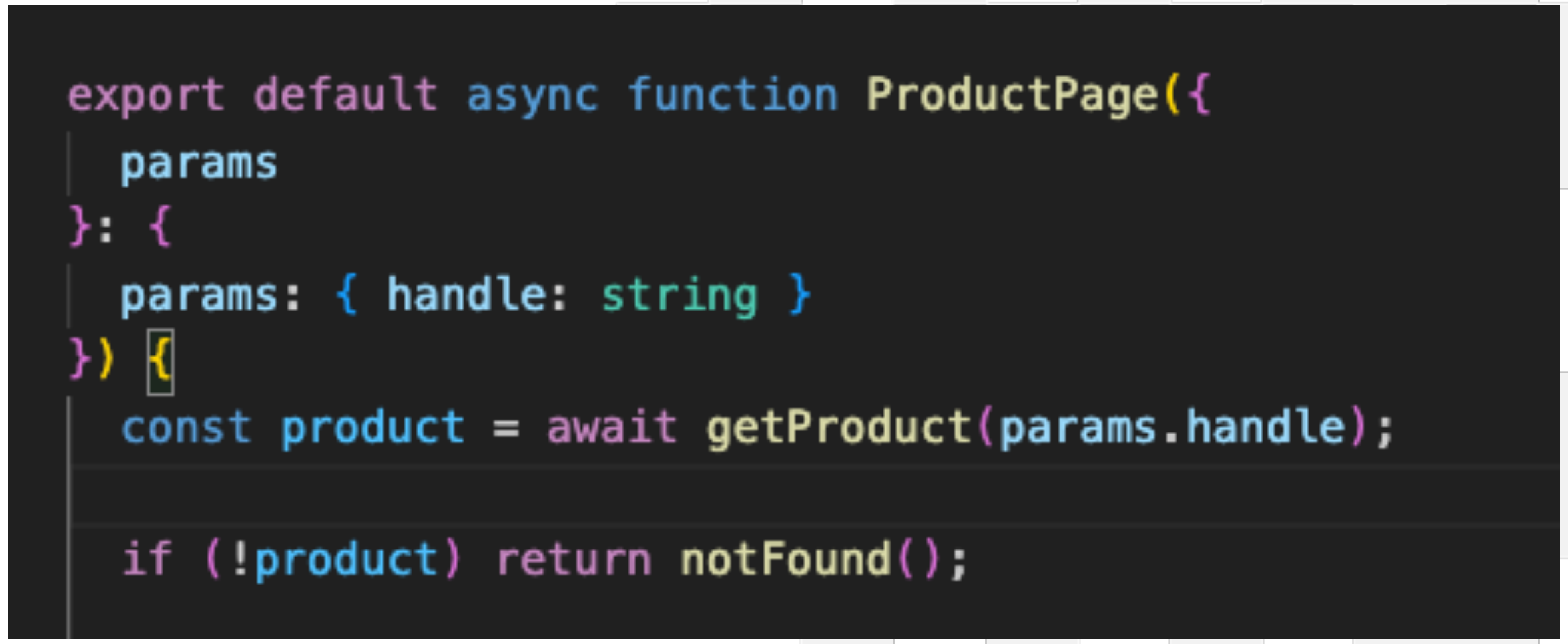
Product Content

- Retrieve product handle from segment in url

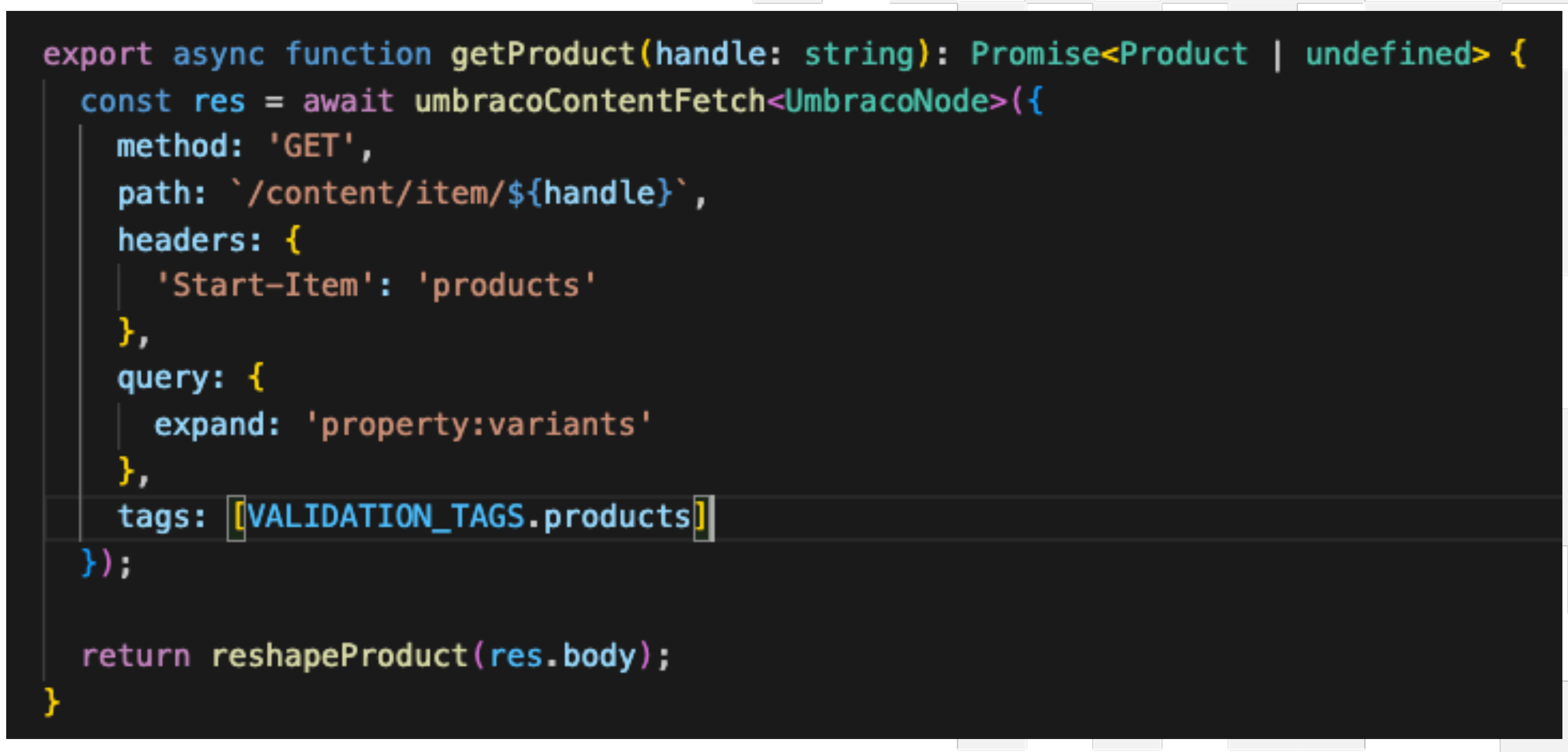
  (app/products/handle/page.tsx)

- Getting a product content from Umbraco

Server Side

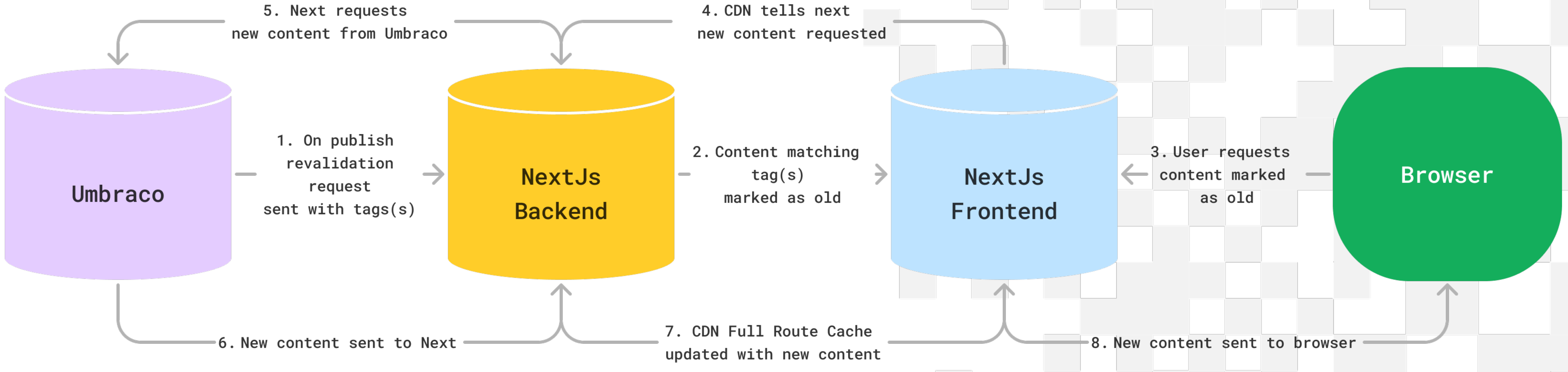


```
export default async function ProductPage({
  params
}: {
  params: { handle: string }
}) {
  const product = await getProduct(params.handle);

  if (!product) return notFound();
```

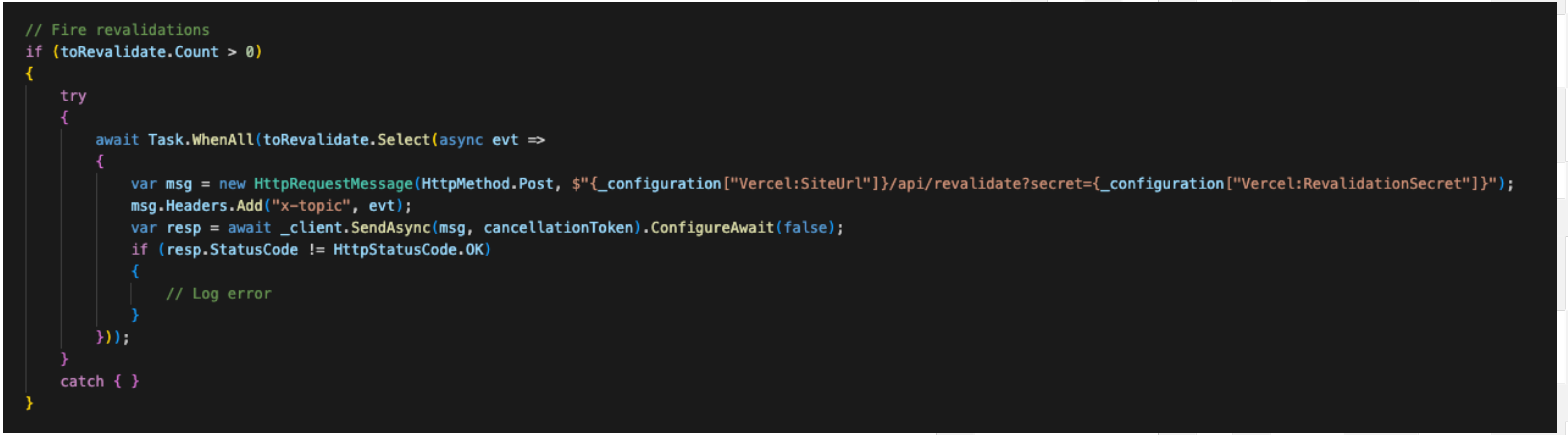


```
export async function getProduct(handle: string): Promise<Product | undefined> {
  const res = await umbracoContentFetch<UmbracoNode>({
    method: 'GET',
    path: `/content/item/${handle}`,
    headers: {
      'Start-Item': 'products'
    },
    query: {
      expand: 'property:variants'
    },
    tags: [VALIDATION_TAGS.products]
  });

  return reshapeProduct(res.body);
}
```

Product Content Revalidation

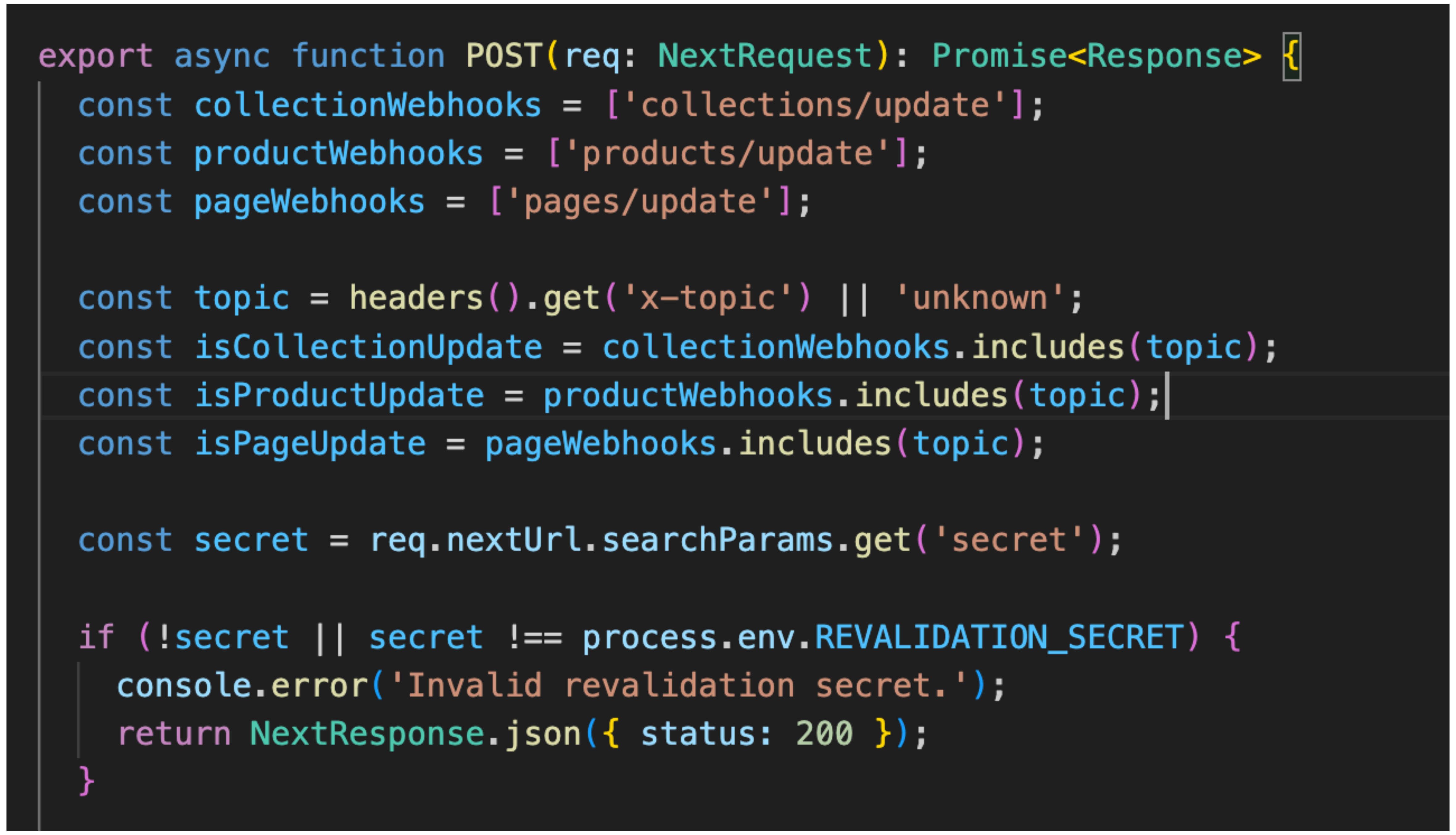

5. Next requests new content from Umbraco
4. CDN tells next new content requested
Umbraco
1. On publish revalidation request sent with tags(s)
NextJs Backend
2. Content matching tag(s) marked as old
NextJs Frontend
3. User requests content marked as old
Browser
6. New content sent to Next
7. CDN Full Route Cache updated with new content
8. New content sent to browser

Product Content Revalidation (Umbraco)

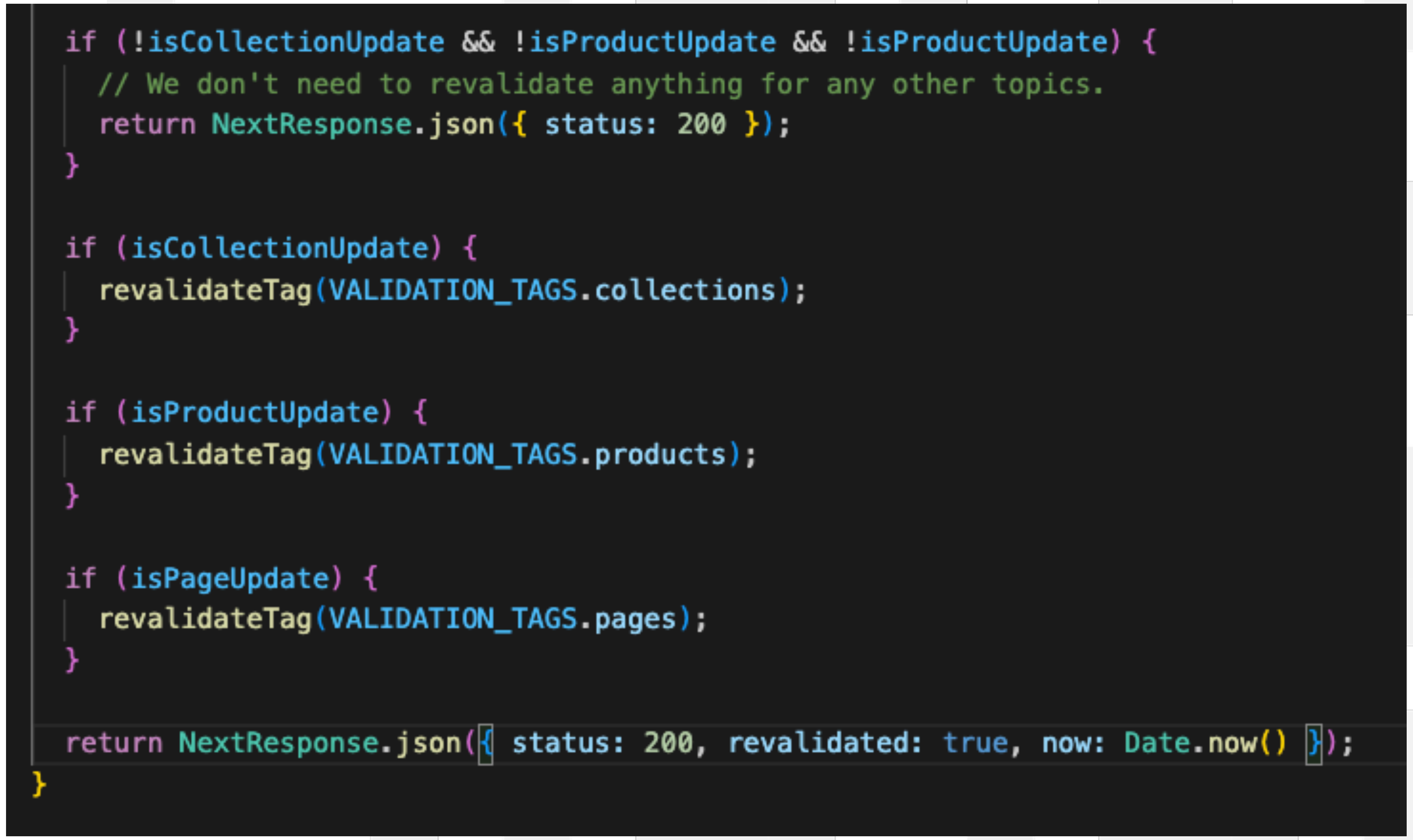


```
// Fire revalidations
if (toRevalidate.Count > 0)
{
    try
    {
        await Task.WhenAll(toRevalidate.Select(async evt =>
        {
            var msg = new HttpRequestMessage(HttpMethod.Post, $"{_configuration["Vercel:SiteUrl"]}/api/revalidate?secret={_configuration["Vercel:RevalidationSecret"]}");
            msg.Headers.Add("x-topic", evt);
            var resp = await _client.SendAsync(msg, cancellationToken).ConfigureAwait(false);
            if (resp.StatusCode != HttpStatusCode.OK)
            {
                // Log error
            }
        }));
    }
    catch { }
}
```

Product Content Revalidation (NextJs)



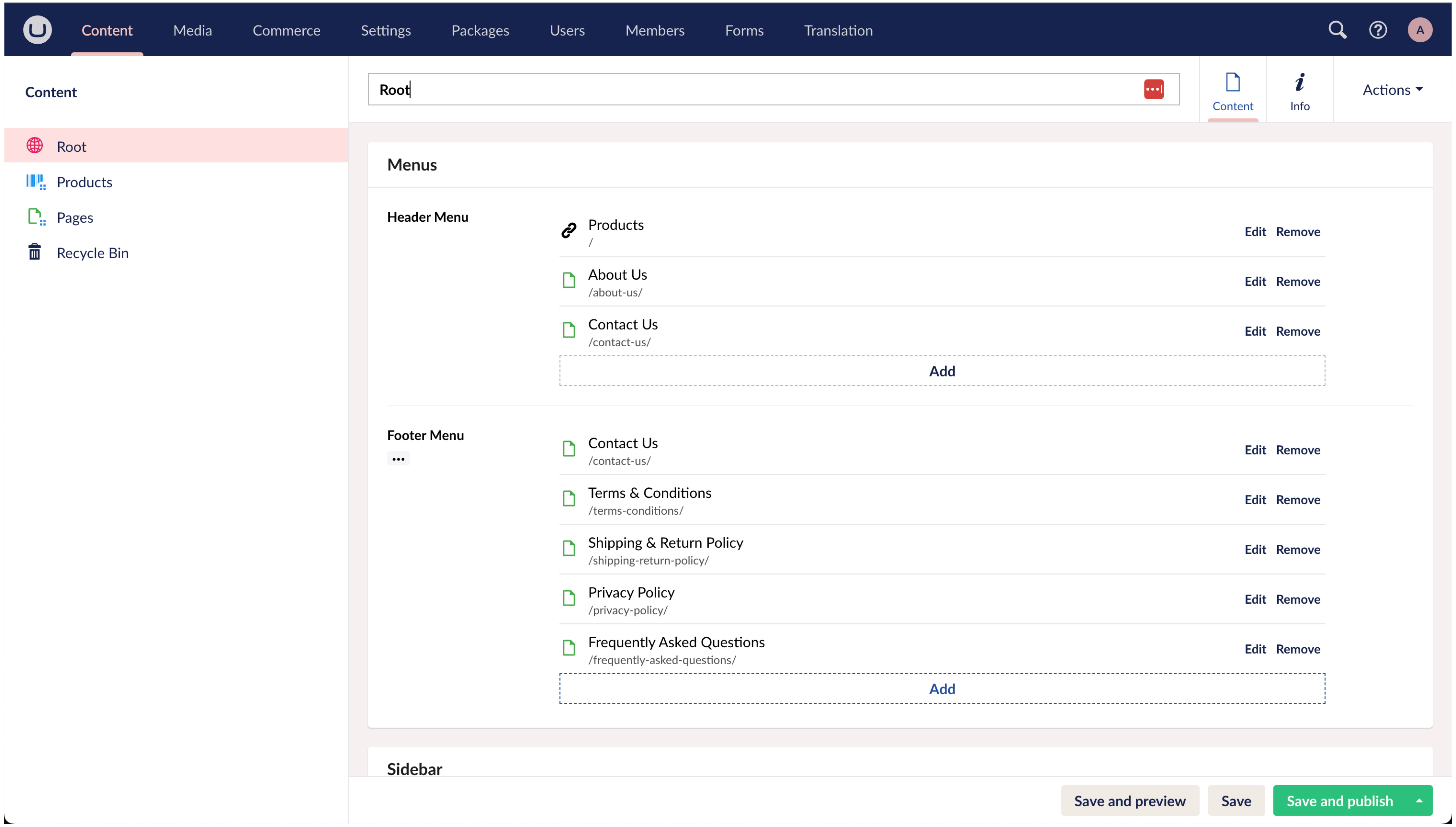
```
export async function POST(req: NextRequest): Promise<Response> {
  const collectionWebhooks = ['collections/update'];
  const productWebhooks = ['products/update'];
  const pageWebhooks = ['pages/update'];

  const topic = headers().get('x-topic') || 'unknown';
  const isCollectionUpdate = collectionWebhooks.includes(topic);
  const isProductUpdate = productWebhooks.includes(topic);
  const isPageUpdate = pageWebhooks.includes(topic);

  const secret = req.nextUrl.searchParams.get('secret');

  if (!secret || secret !== process.env.REVALIDATION_SECRET) {
    console.error('Invalid revalidation secret.');
    return NextResponse.json({ status: 200 });
  }
```



```
  if (!isCollectionUpdate && !isProductUpdate && !isProductUpdate) {
    // We don't need to revalidate anything for any other topics.
    return NextResponse.json({ status: 200 });
  }

  if (isCollectionUpdate) {
    revalidateTag(VALIDATION_TAGS.collections);
  }

  if (isProductUpdate) {
    revalidateTag(VALIDATION_TAGS.products);
  }

  if (isPageUpdate) {
    revalidateTag(VALIDATION_TAGS.pages);
  }

  return NextResponse.json({ status: 200, revalidated: true, now: Date.now() });
}
```

Navigation

Navigation



Root

Menus

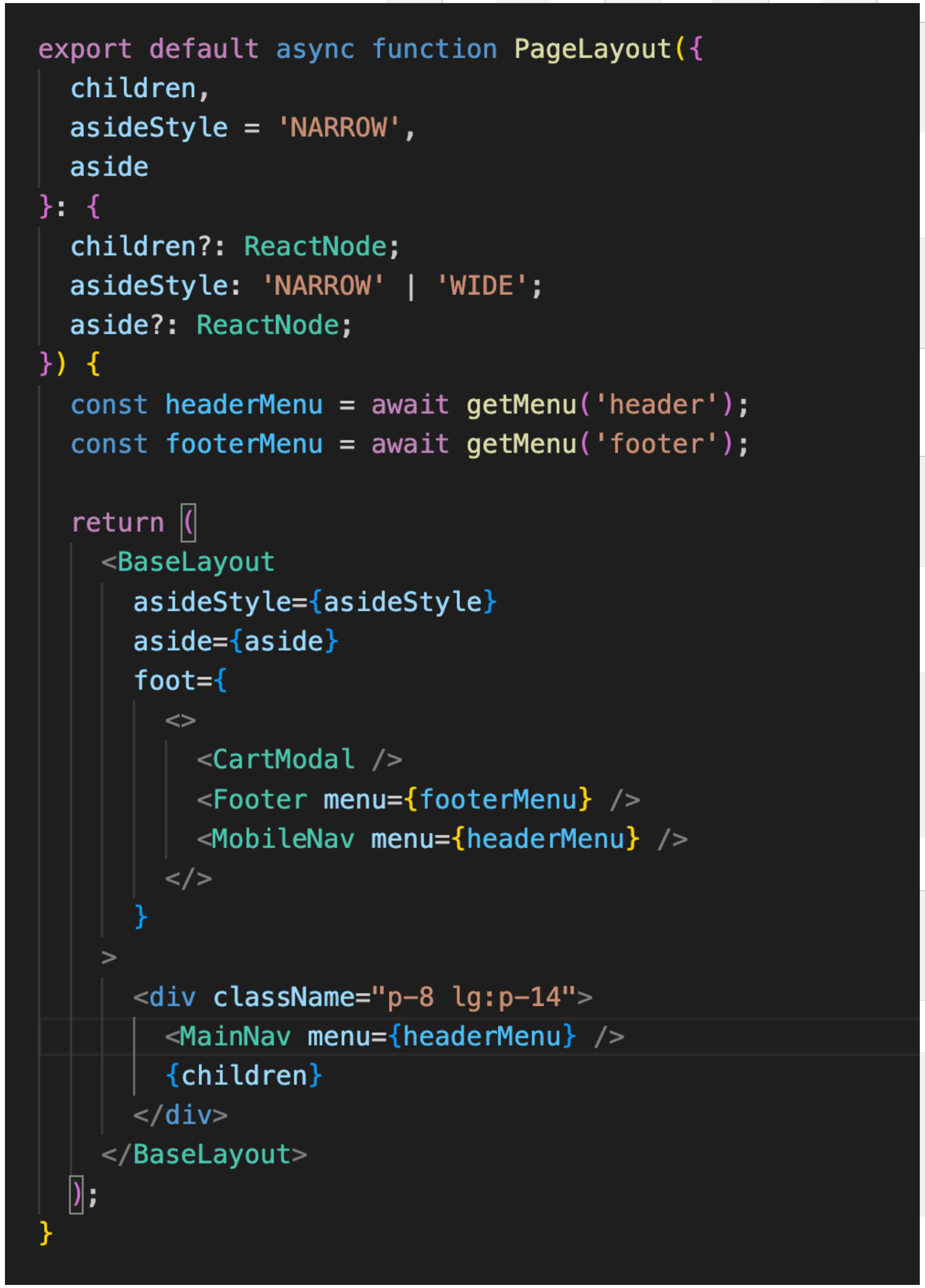
Header Menu	Products /	Edit Remove
	About Us /about-us/	Edit Remove
	Contact Us /contact-us/	Edit Remove
	Add	
Footer Menu	Contact Us /contact-us/	Edit Remove
	Terms & Conditions /terms-conditions/	Edit Remove
	Shipping & Return Policy /shipping-return-policy/	Edit Remove
	Privacy Policy /privacy-policy/	Edit Remove
	Frequently Asked Questions /frequently-asked-questions/	Edit Remove
	Add	

Sidebar

Navigation

- Page layout (server-side)

  (components/layout/page-layout.tsx)

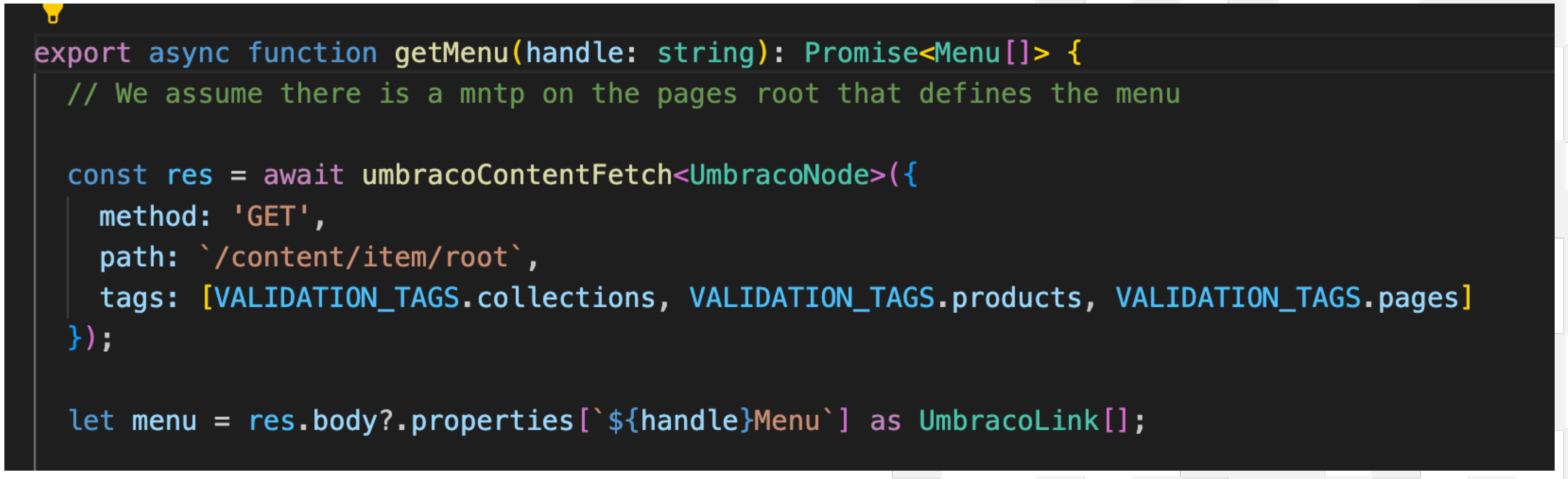


```tsx
export default async function PageLayout({
  children,
  asideStyle = 'NARROW',
  aside
}: {
  children?: ReactNode;
  asideStyle: 'NARROW' | 'WIDE';
  aside?: ReactNode;
}) {
  const headerMenu = await getMenu('header');
  const footerMenu = await getMenu('footer');

  return (
    <BaseLayout
      asideStyle={asideStyle}
      aside={aside}
      foot={
        <>
          <CartModal />
          <Footer menu={footerMenu} />
          <MobileNav menu={headerMenu} />
        </>
      }
    >
      <div className="p-8 lg:p-14">
        <MainNav menu={headerMenu} />
        {children}
      </div>
    </BaseLayout>
  );
}
```

Navigation

- Get menu

  (lib/umbraco/index.tsx)

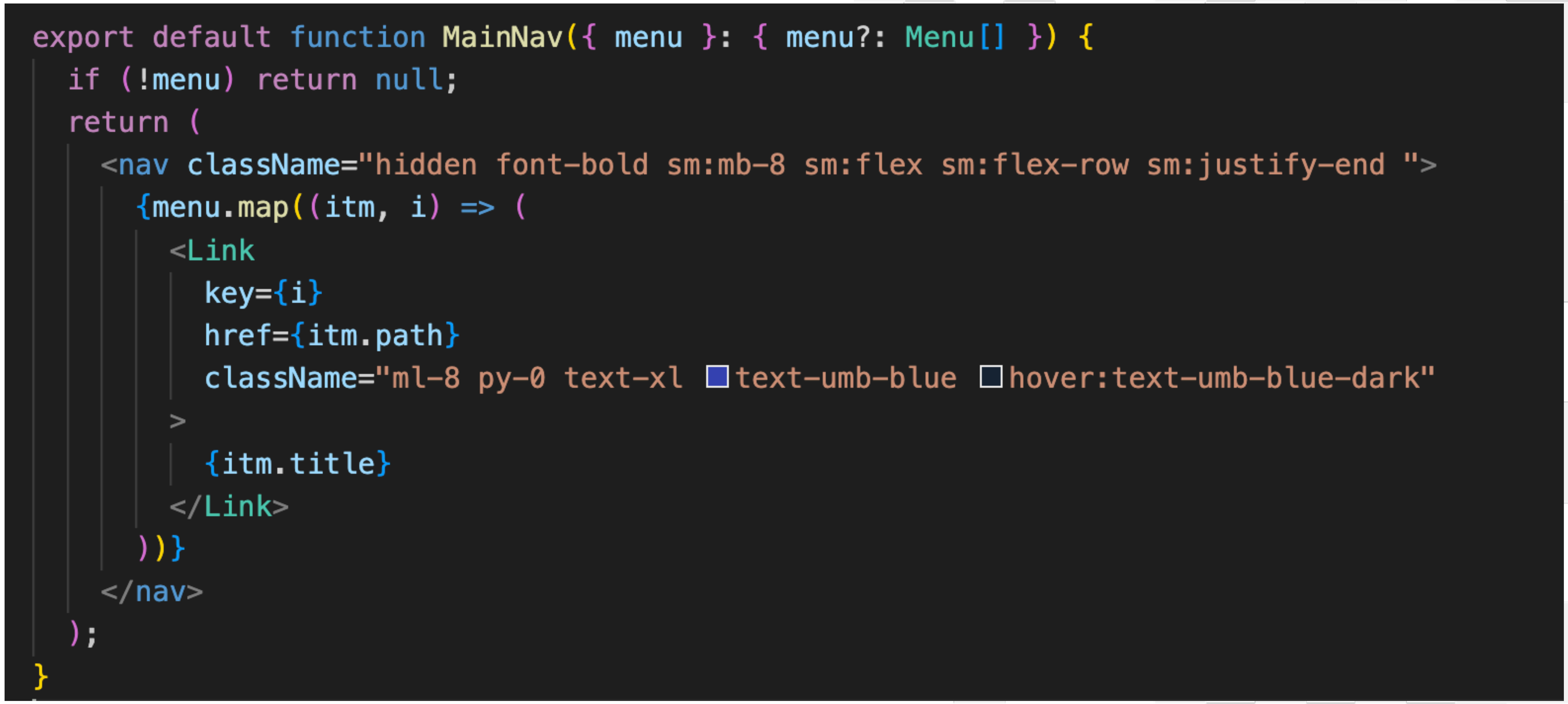


```
export async function getMenu(handle: string): Promise<Menu[]> {
  // We assume there is a mntp on the pages root that defines the menu

  const res = await umbracoContentFetch<UmbracoNode>({
    method: 'GET',
    path: `/content/item/root`,
    tags: [VALIDATION_TAGS.collections, VALIDATION_TAGS.products, VALIDATION_TAGS.pages]
  });

  let menu = res.body?.properties[`${handle}Menu`] as UmbracoLink[];
```

Navigation

- Render menu

  (components/layout/main-nav.tsx)



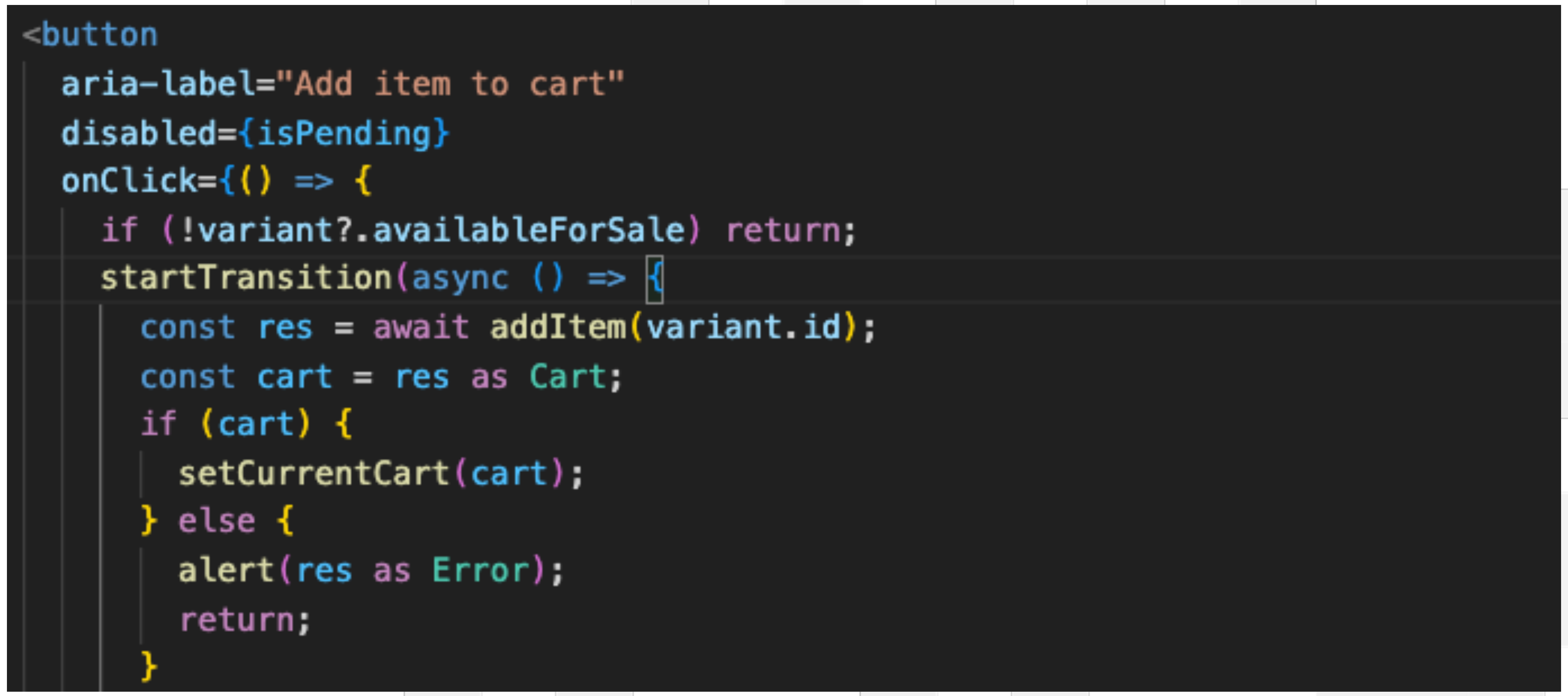
```
export default function MainNav({ menu }: { menu?: Menu[] }) {
  if (!menu) return null;
  return (
    <nav className="hidden font-bold sm:mb-8 sm:flex sm:flex-row sm:justify-end ">
      {menu.map((itm, i) => (
        <Link
          key={i}
          href={itm.path}
          className="ml-8 py-0 text-xl ■text-umb-blue □hover:text-umb-blue-dark"
        >
          {itm.title}
        </Link>
      ))}
    </nav>
  );
}
```

Cart

Adding to cart

- Button handler (client-side)

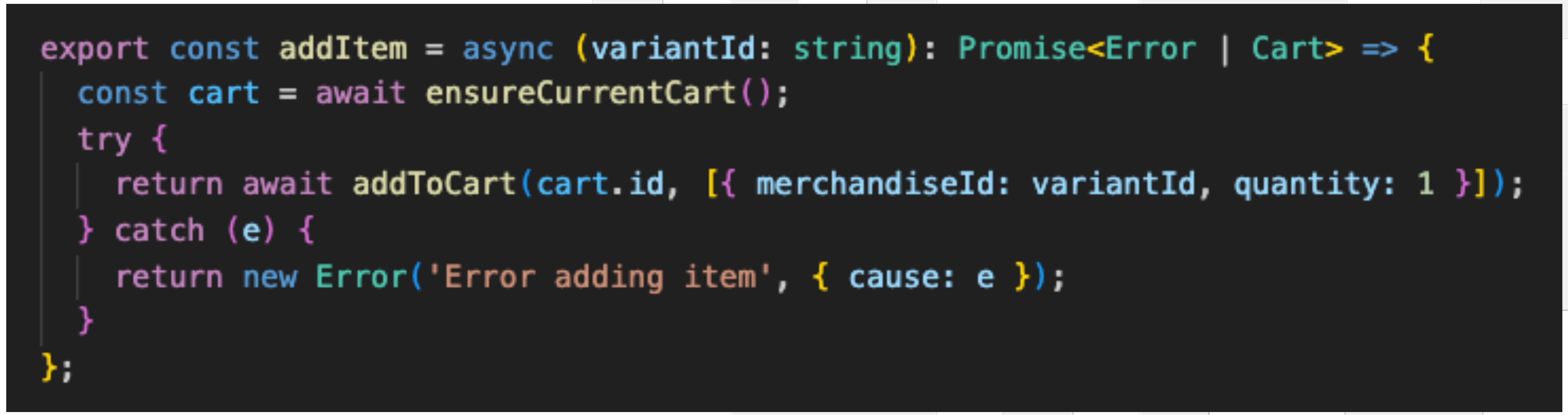
  (components/cart/add-to-cart-button.tsx)



```
<button
  aria-label="Add item to cart"
  disabled={isPending}
  onClick={() => {
    if (!variant?.availableForSale) return;
    startTransition(async () => {
      const res = await addItem(variant.id);
      const cart = res as Cart;
      if (cart) {
        setCurrentCart(cart);
      } else {
        alert(res as Error);
        return;
      }
```

- Server-side handler

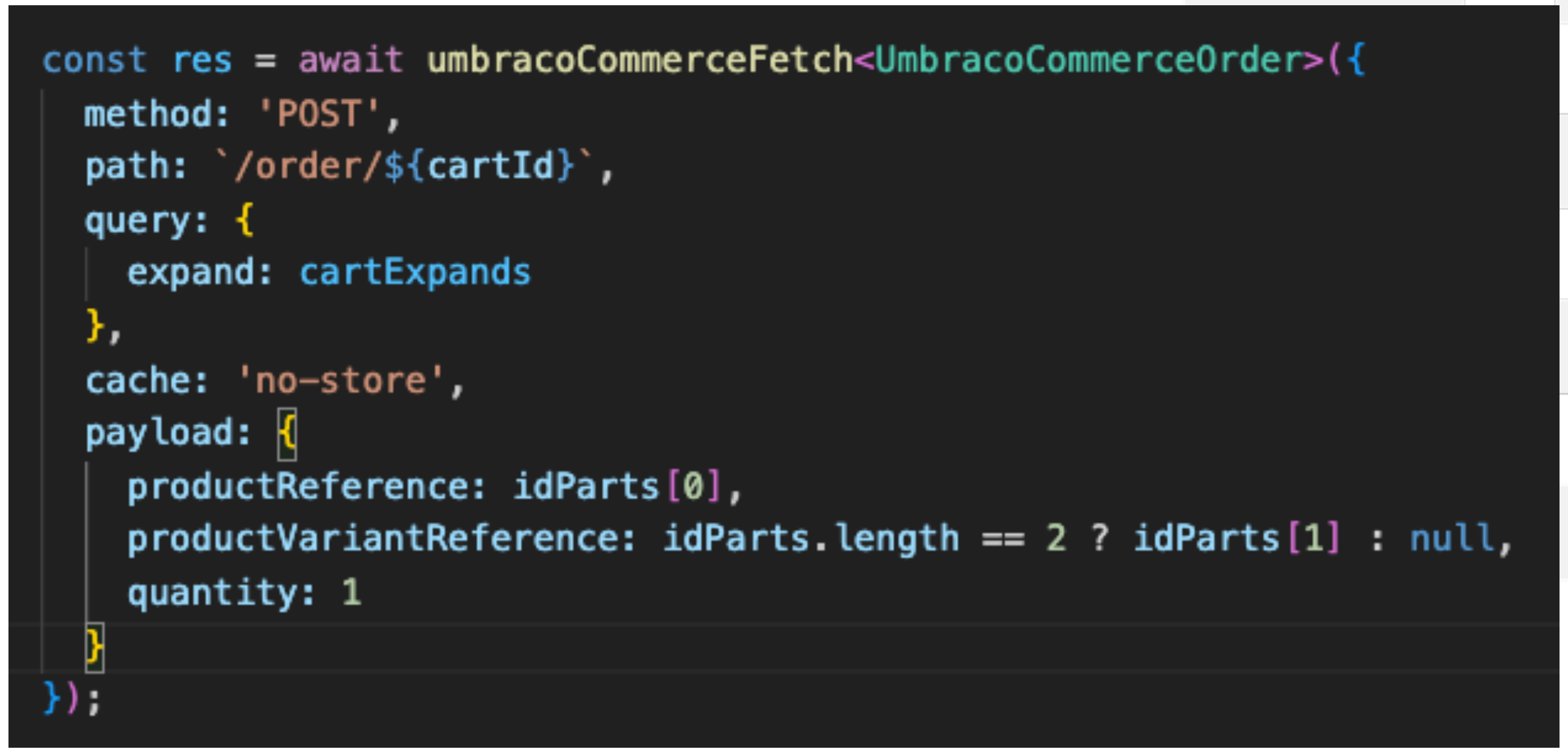
  (components/cart-actions.tsx)



```
export const addItem = async (variantId: string): Promise<Error | Cart> => {
  const cart = await ensureCurrentCart();
  try {
    return await addToCart(cart.id, [{ merchandiseId: variantId, quantity: 1 }]);
  } catch (e) {
    return new Error('Error adding item', { cause: e });
  }
};
```

- Service call

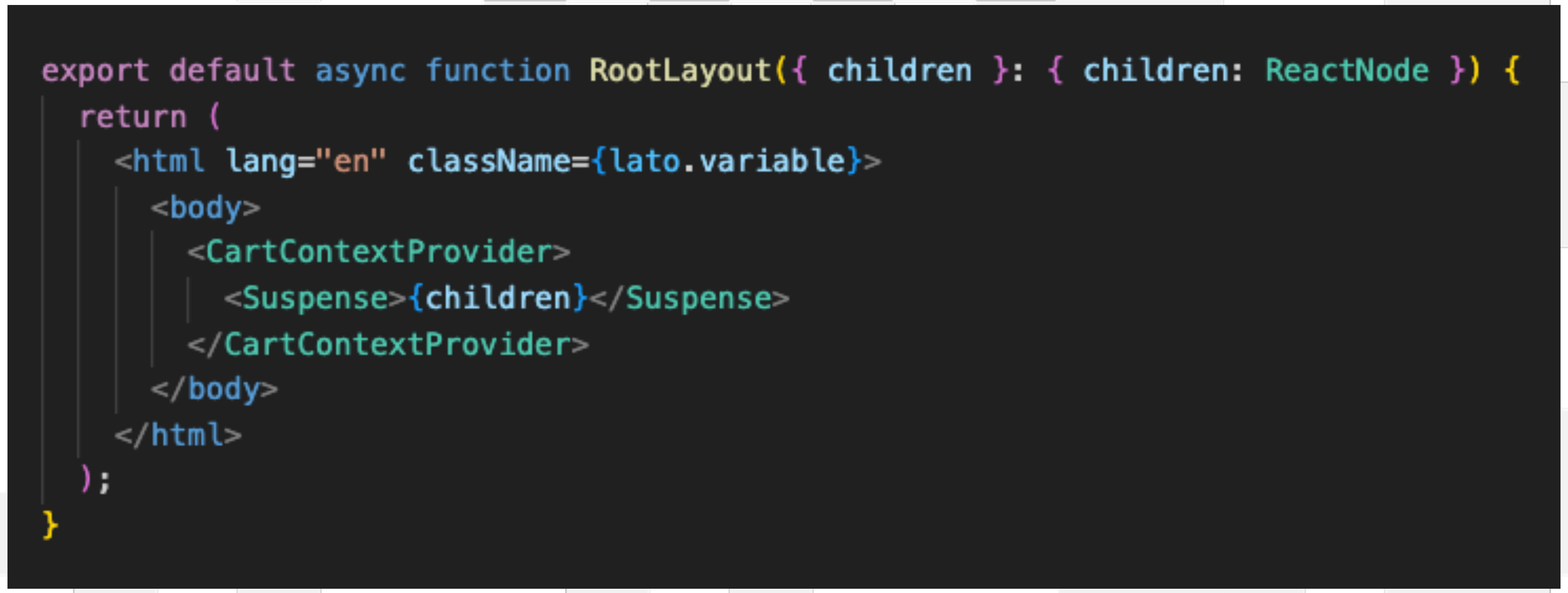
  (lib/umbraco/index.ts)



```
const res = await umbracoCommerceFetch<UmbracoCommerceOrder>({
  method: 'POST',
  path: `/order/${cartId}`,
  query: {
    expand: cartExpands
  },
  cache: 'no-store',
  payload: {
    productReference: idParts[0],
    productVariantReference: idParts.length == 2 ? idParts[1] : null,
    quantity: 1
  }
});
```

Loading Current Cart

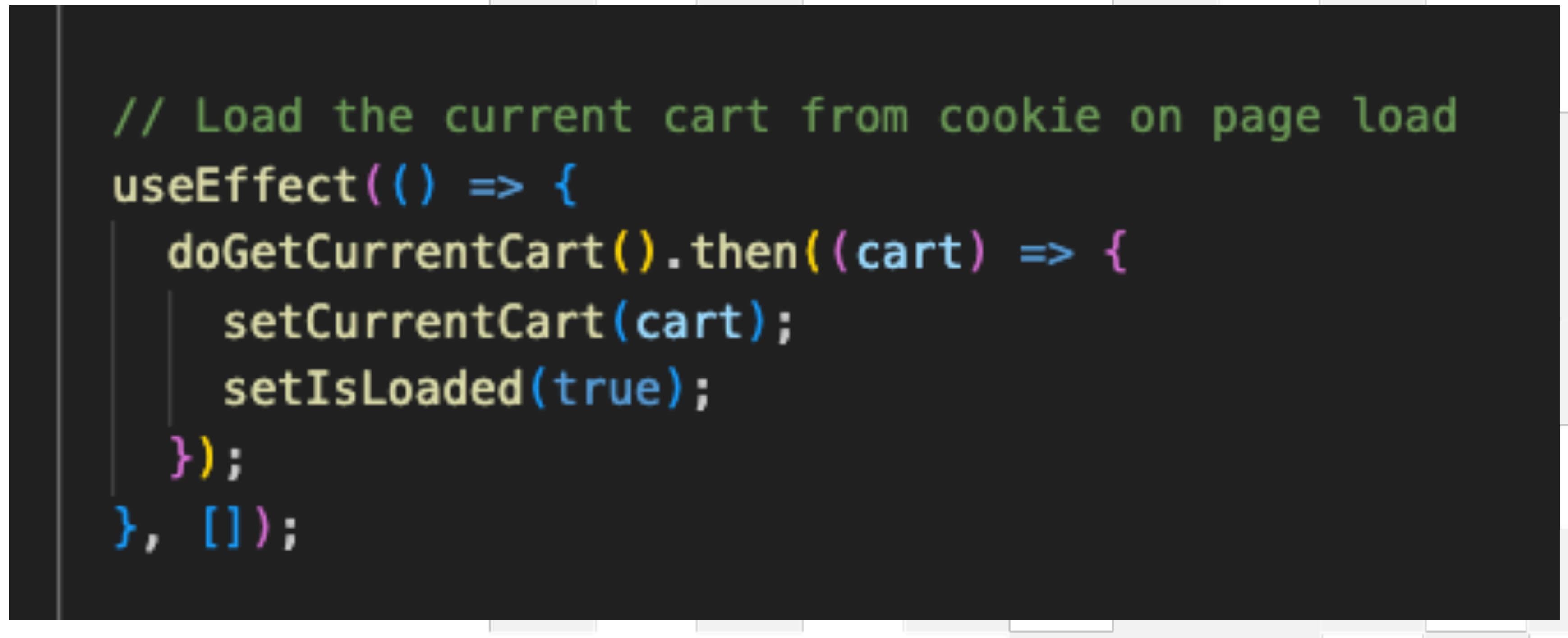
- Layout context providers



```
export default async function RootLayout({ children }: { children: ReactNode }) {
  return (
    <html lang="en" className={lato.variable}>
      <body>
        <CartContextProvider>
          <Suspense>{children}</Suspense>
        </CartContextProvider>
      </body>
    </html>
  );
}
```

- Cart Context Provider

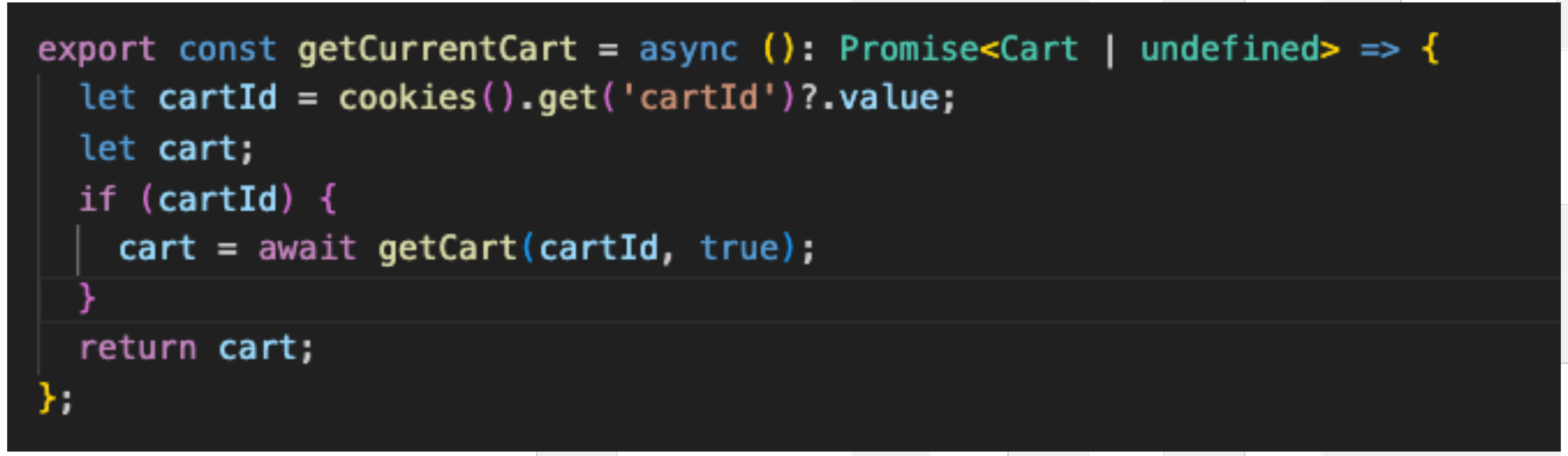
  client-side (components/cart-context.tsx)



```
// Load the current cart from cookie on page load
useEffect(() => {
  doGetCurrentCart().then((cart) => {
    setCurrentCart(cart);
    setIsLoaded(true);
  });
}, []);
```

- Server-side handler

  (components/cart-actions.tsx)



```
export const getCurrentCart = async (): Promise<Cart | undefined> => {
  let cartId = cookies().get('cartId')?.value;
  let cart;
  if (cartId) {
    cart = await getCart(cartId, true);
  }
  return cart;
};
```

In Summary

- Some parts of NextJs feel very familiar
- Be careful and build with caching in mind
- NextJs + Umbraco can be used to create rock solid and efficient sites
- NextJs is not that scary, give it a try
- Umbraco upgrades just got so much easier

Thanks for listening