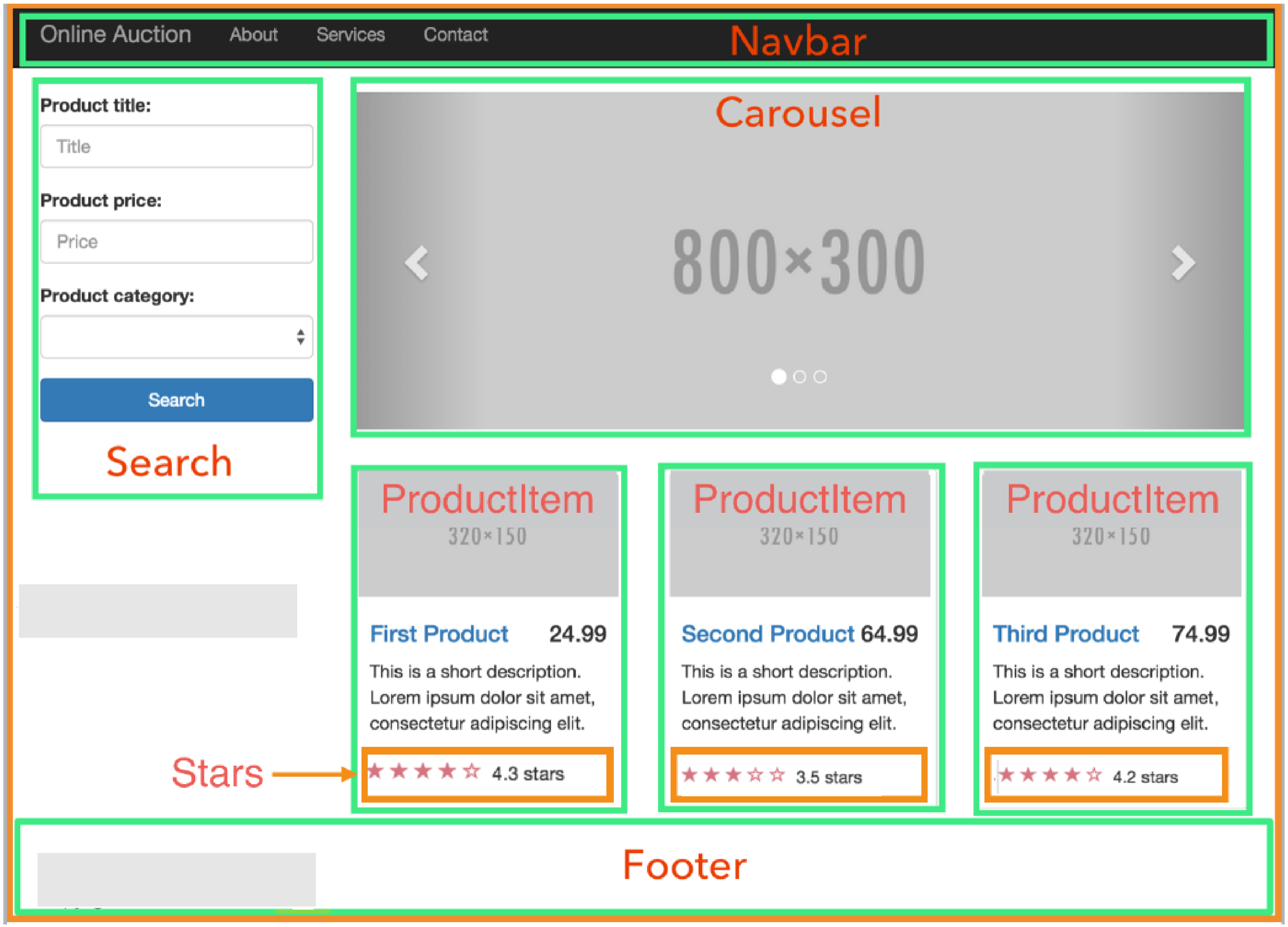
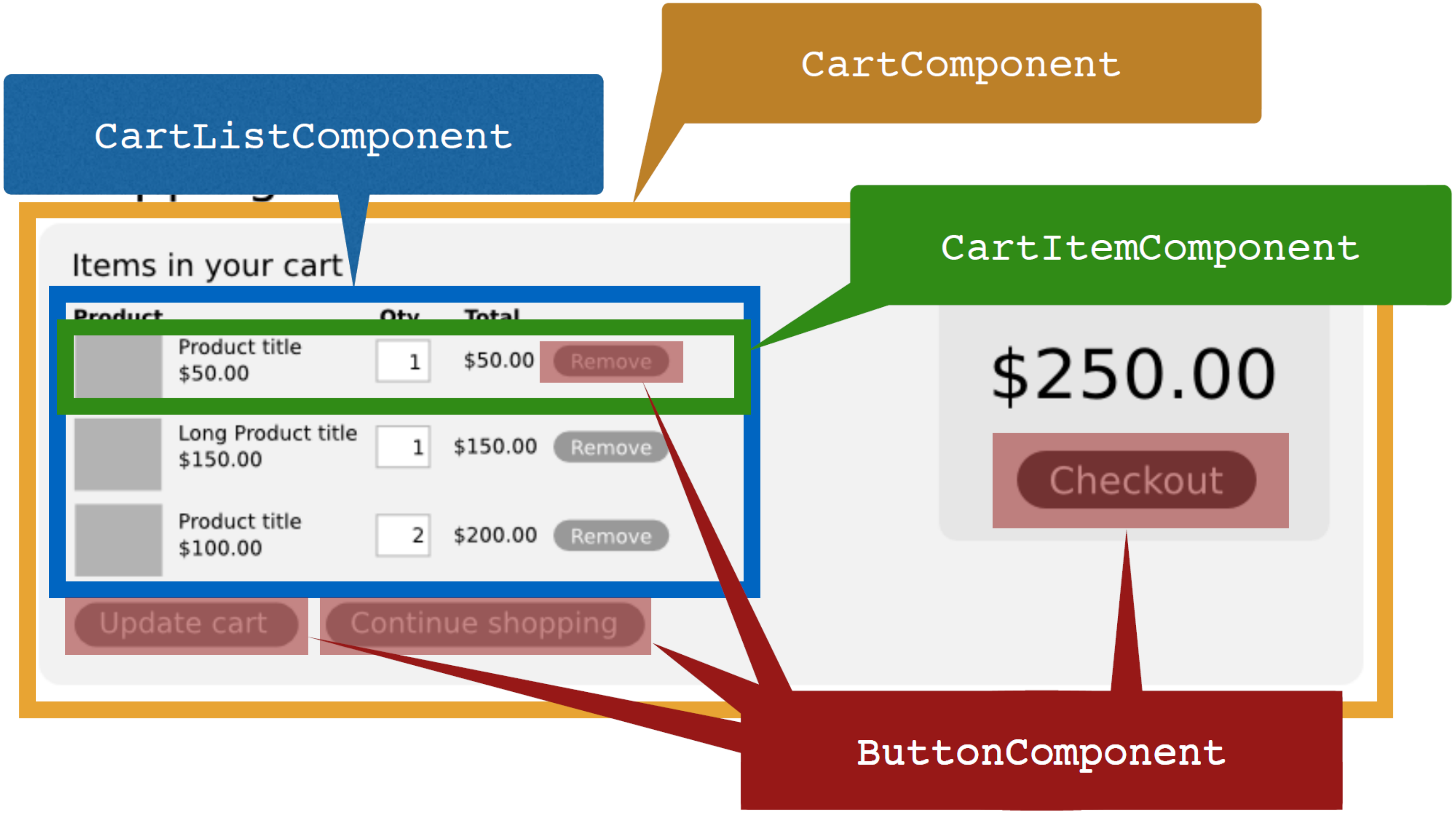
NEXT.JS Basics

Outline

An app = a composition of components



An app = a tree of components



Next.js vs React

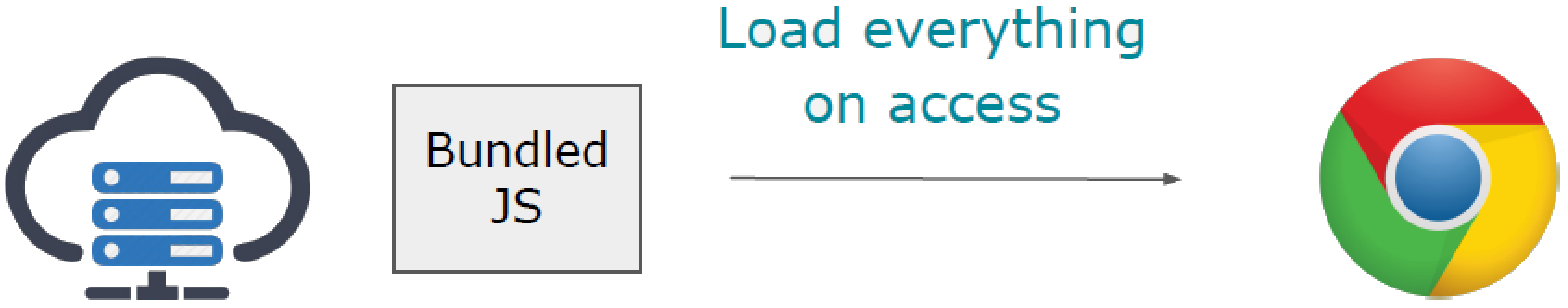
- React is just a **client-side JavaScript** library, Next.js is a framework for building rich and complete Web App **both on the client and server sides**
- React runs on the client side
    - Could negatively affect Search Engine Optimization (SEO) and
    - Slow initial load performance: To display the complete web app, the browser had to download the entire application bundle, parse its content, then execute it and render the result in the browser
        - which could take up to a few seconds for a large application

What is Next.js?

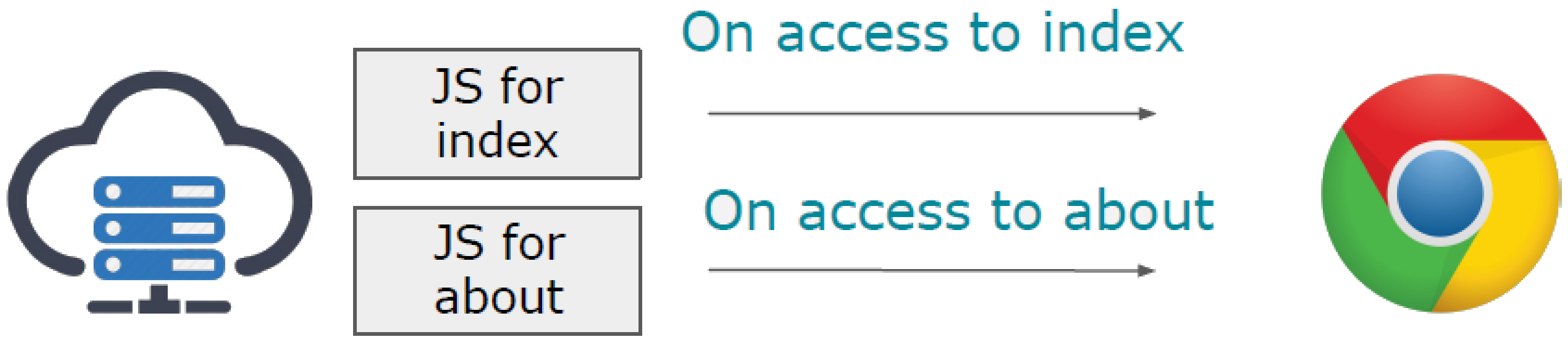
- Next.js = React-based full stack web framework that allows creating user interfaces, static pages, server-side rendered pages, and Web API
- It provides a large set of features out of the box, such as:
  - Automatic code-splitting
  - File system-based routing systems
  - Route prefetching
  - API Routes
  - Automatic image optimization
  - Different rendering strategies: Server-side rendering, Static site generation, Incremental static generation
  - Support for internationalization
  - Fast refresh on the development environment

Code splitting

- In SPA, a large bundled file will be loaded as default



- With Next.js , code will be split on per page base as default



Getting started

- Install latest **Node.js** https://nodejs.org/en/
- Download **VS Code** https://code.visualstudio.com/
- Create an empty folder (with no space in the name use **dash -** instead)
- Create a react app

```
npx create-next-app .
```

- Run the app in dev mode: `npm run dev`
- Build the app: `npm run build`
- Run the optimized build: `npm run start`

Project Folder Structure

- Next.js uses **`pages/`** folder for routing, every JavaScript file inside it will be a page
    - the pages/ directory is a container for the app pages
- The **`public/`** folder contains all the public and static assets such as images, fonts, etc.
- public/ and pages/ are mandatory and reserved directories so make sure not to delete or use them for different purposes
- **`styles/`** optional folder for organizing stylesheets

Routing

app Directory

Next.js **app/** offers:

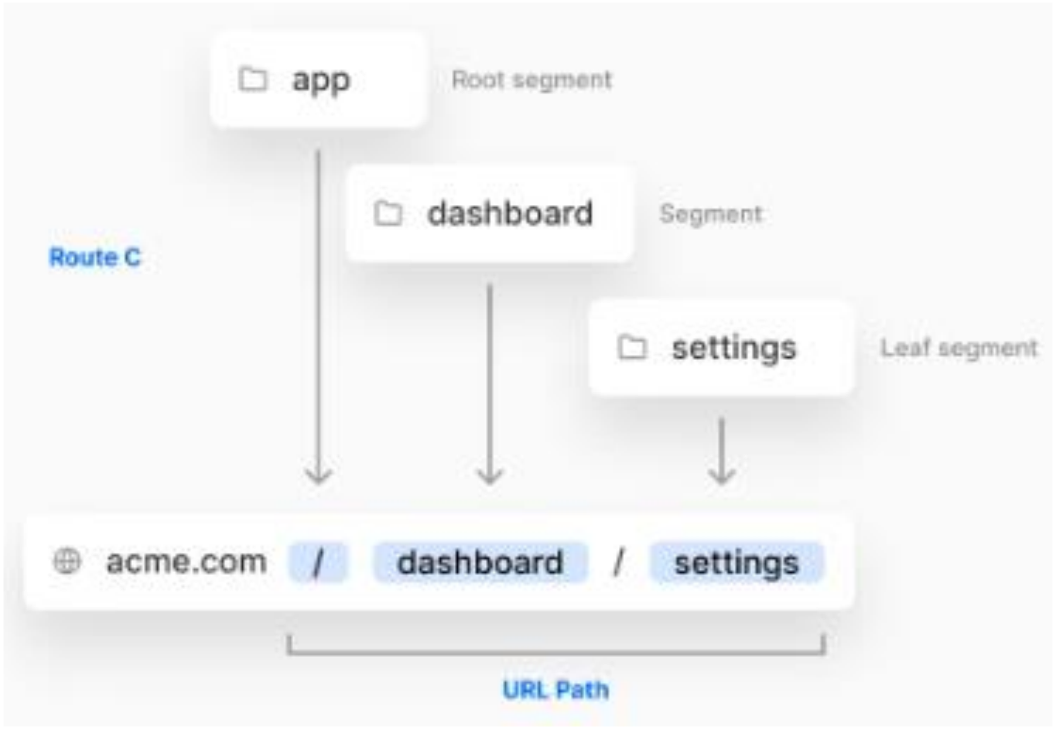
- **Layouts**: Easily share UI while preserving state and avoiding re-renders
- Server Components: Making server-first the default to reduce client-side JS
- Streaming: Display instant loading states and stream in updates
- Suspense for Data Fetching: async/await support and the **use** hook for component-level fetching

Routing

- Use folder hierarchy inside the **app** folder to define routes, and files to define UI
  - A route is a single path of nested folders, from the root folder down to a leaf folder
  - Use a special **page.js** file to make a route segment publicly accessible

- Each folder in the subtree represents a route segment in a URL path

- E.g., create `/dashboard/settings` route by nesting two subfolders in the app directory



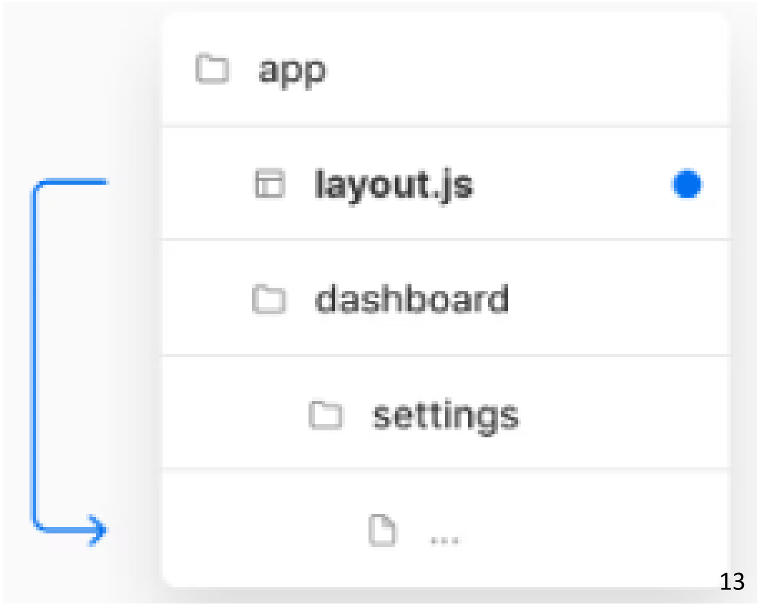
Layouts

- A layout is UI that is shared between route segments
  - Do not re-render (React state is preserved) when a user navigates between sibling segments
  - Navigating between routes only fetches and renders the segments that change
- A layout can be defined by exporting a React component from a `layout.js` file
  - The component should accept a `children` prop which will be populated with the segments the layout is wrapping

There are 2 types of layouts:

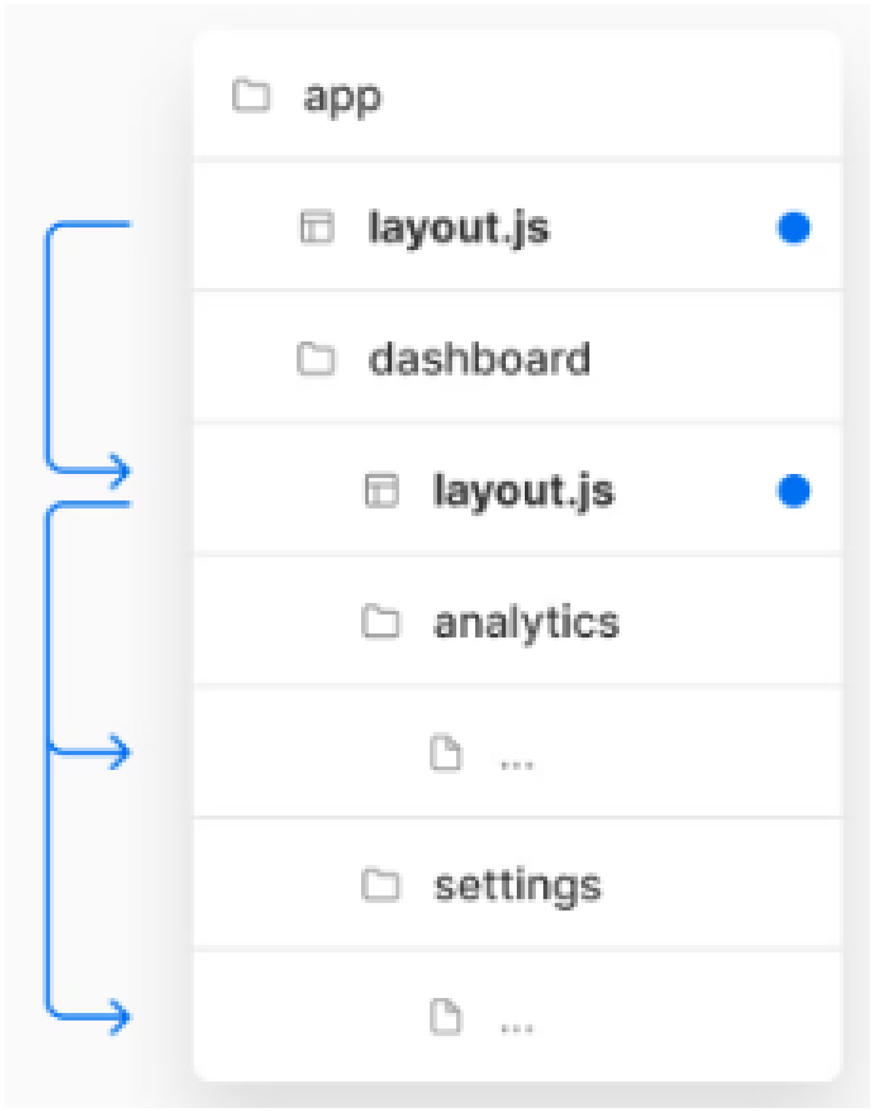
- **Root layout**: in **app** folder and applies to all routes

- **Regular layout**: inside a specific folder and applies to associated route segments

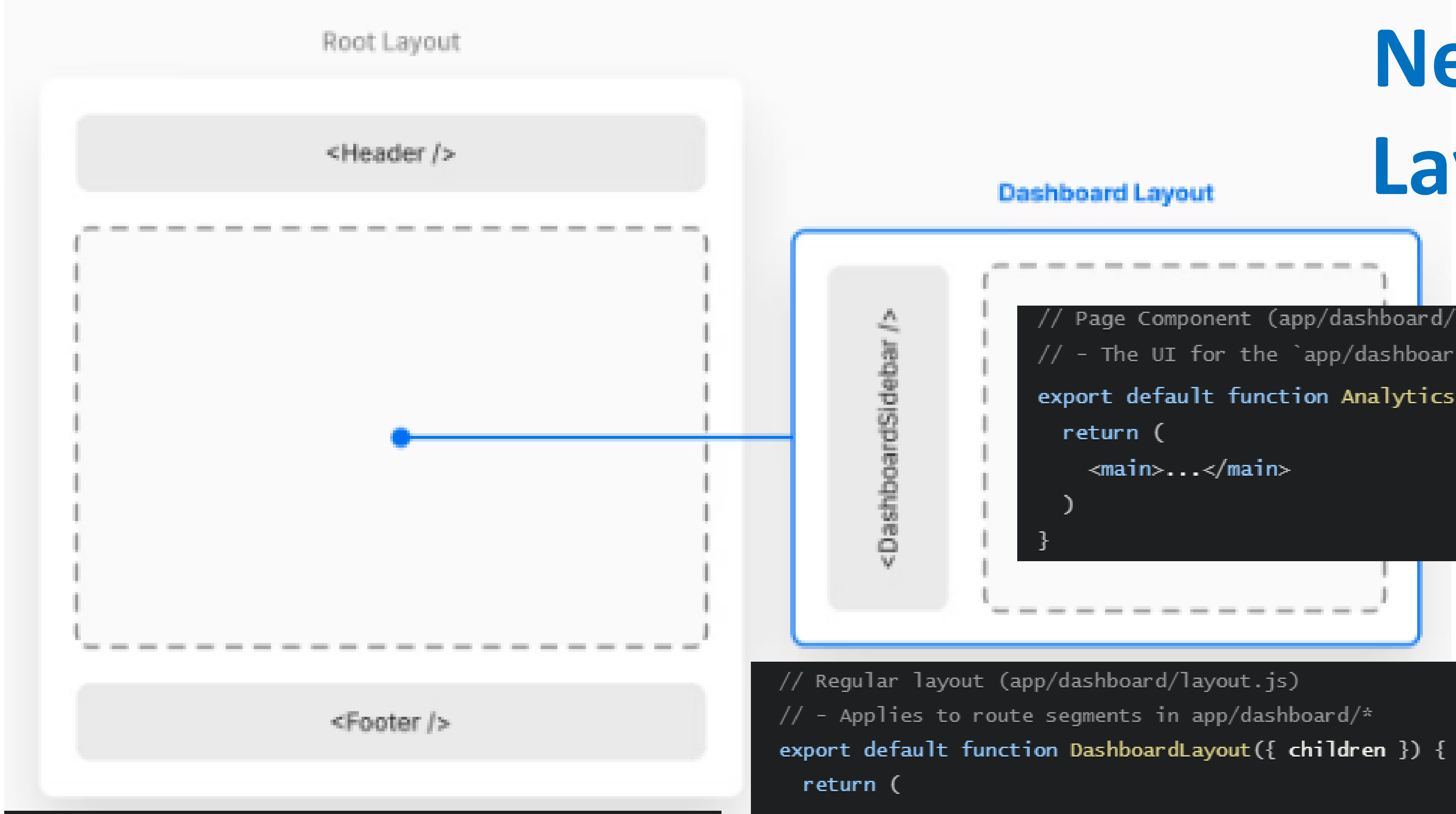


Nesting Layouts

- Layouts that can be nested and shared across routes
- E.g., the root layout (`app/layout.js`) would be applied to the dashboard layout, which would also apply to all route segments inside `dashboard/*`



Nesting Layouts



```
// Page Component (app/dashboard/analytics/page.js)
// - The UI for the `app/dashboard/analytics` segment
export default function AnalyticsPage() {
  return (
    <main>...</main>
  )
}
```

```
// Regular layout (app/dashboard/layout.js)
// - Applies to route segments in app/dashboard/*
export default function DashboardLayout({ children }) {
  return (
    <>
      <DashboardSidebar />
      {children}
    </>
  )
}
```

```
// Root layout (app/layout.js)
// - Applies to all routes
export default function RootLayout({ children })
  return (
    <html>
      <body>
        <Header />
        {children}
        <Footer />
      </body>
    </html>
  )
}
```

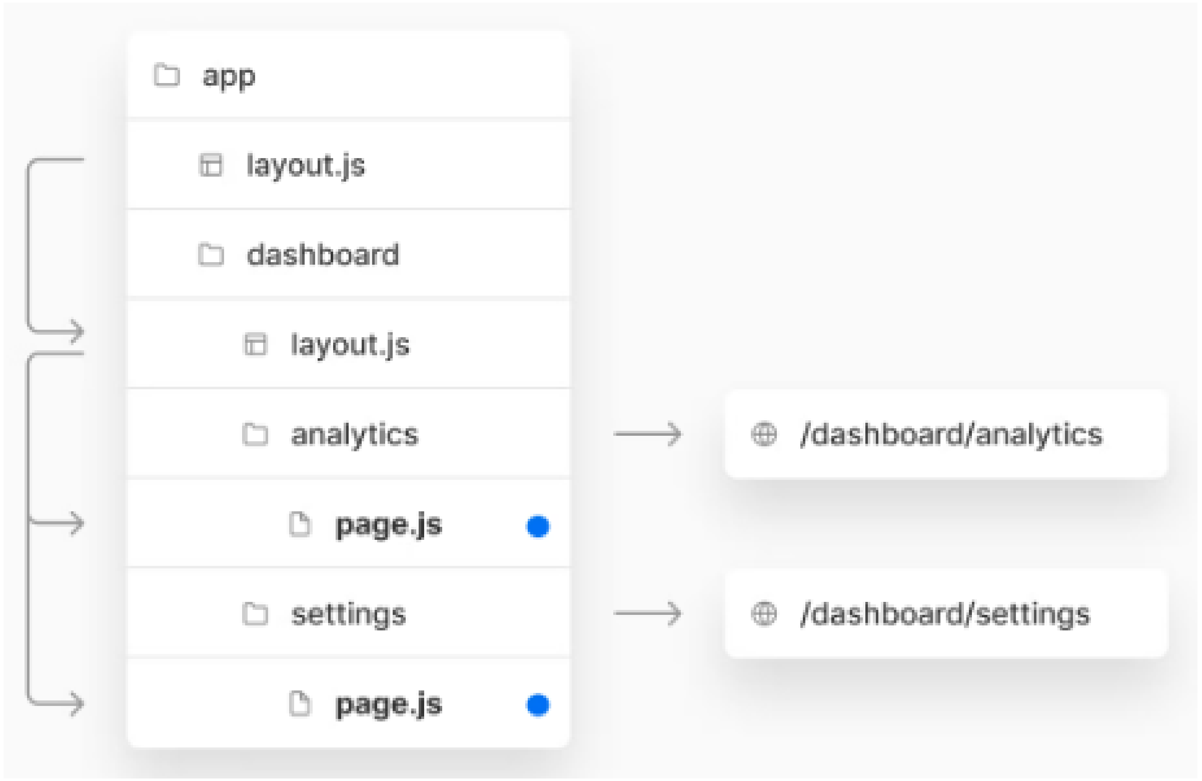
The above combination of layouts and pages would render the following component hierarchy:

```
<RootLayout>
  <Header />
  <DashboardLayout>
    <DashboardSidebar />
    <AnalyticsPage>
      <main>...</main>
    </AnalyticsPage>
  </DashboardLayout>
  <Footer />
</RootLayout>
```

UI Pages

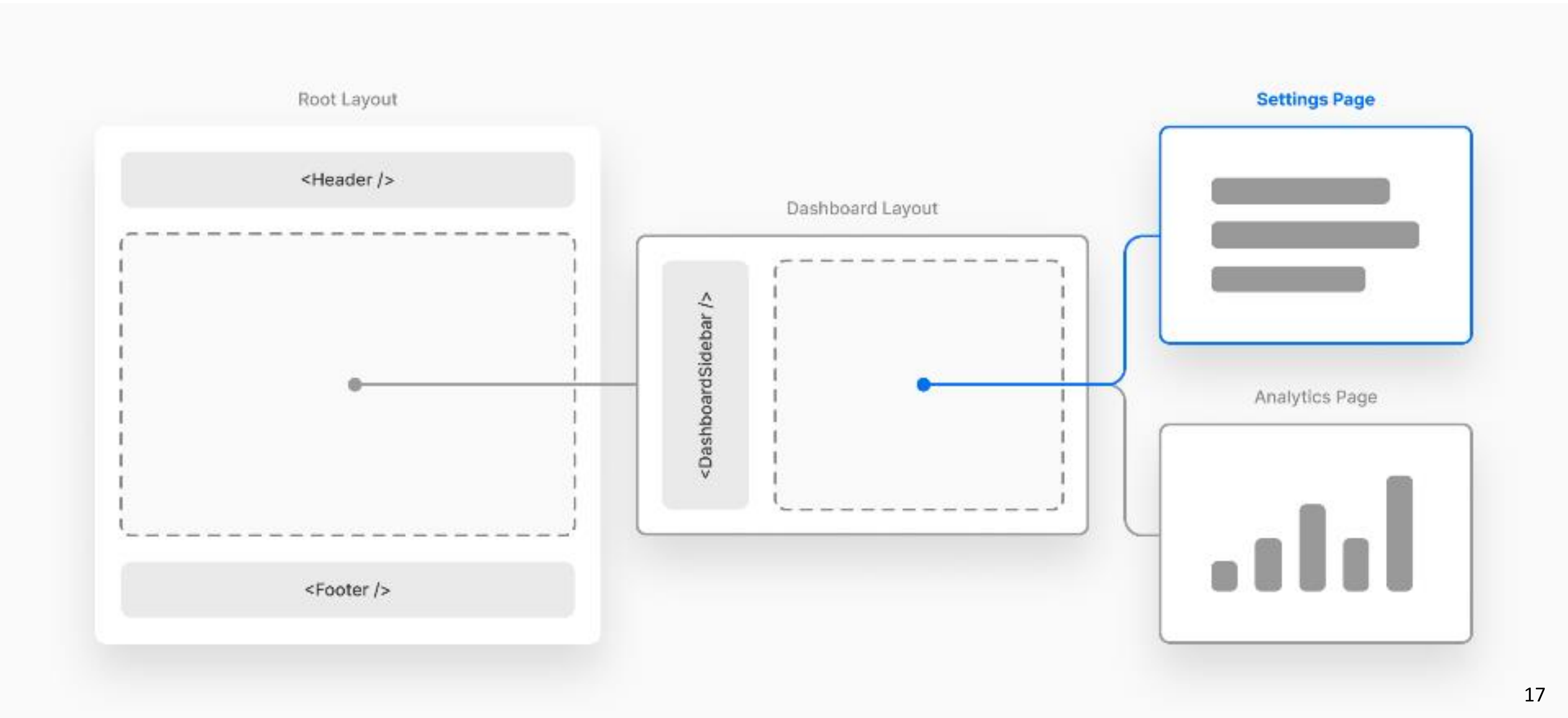
- You can create a page by adding a `page.js` file inside a folder
  - Can colocate your own project files (UI components, styles, images, test files, etc.) inside the app folder & subfolders

When a user visits `/dashboard/settings` Next.js will render the `page.js` file inside the settings folder



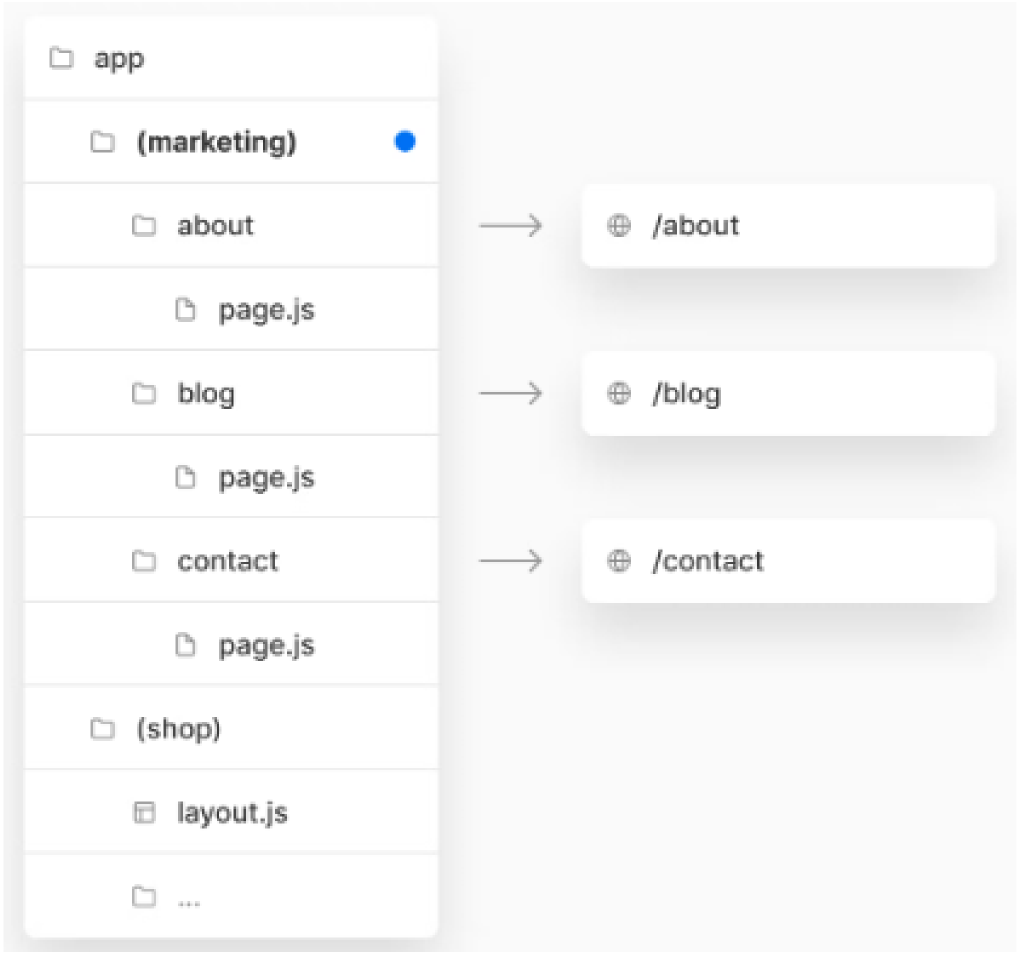
Pages are Wrapped in Layouts

- When a user visits `/dashboard/settings` Next.js will render the `page.js` file inside the settings folder wrapped in any layouts that exist further up the subtree

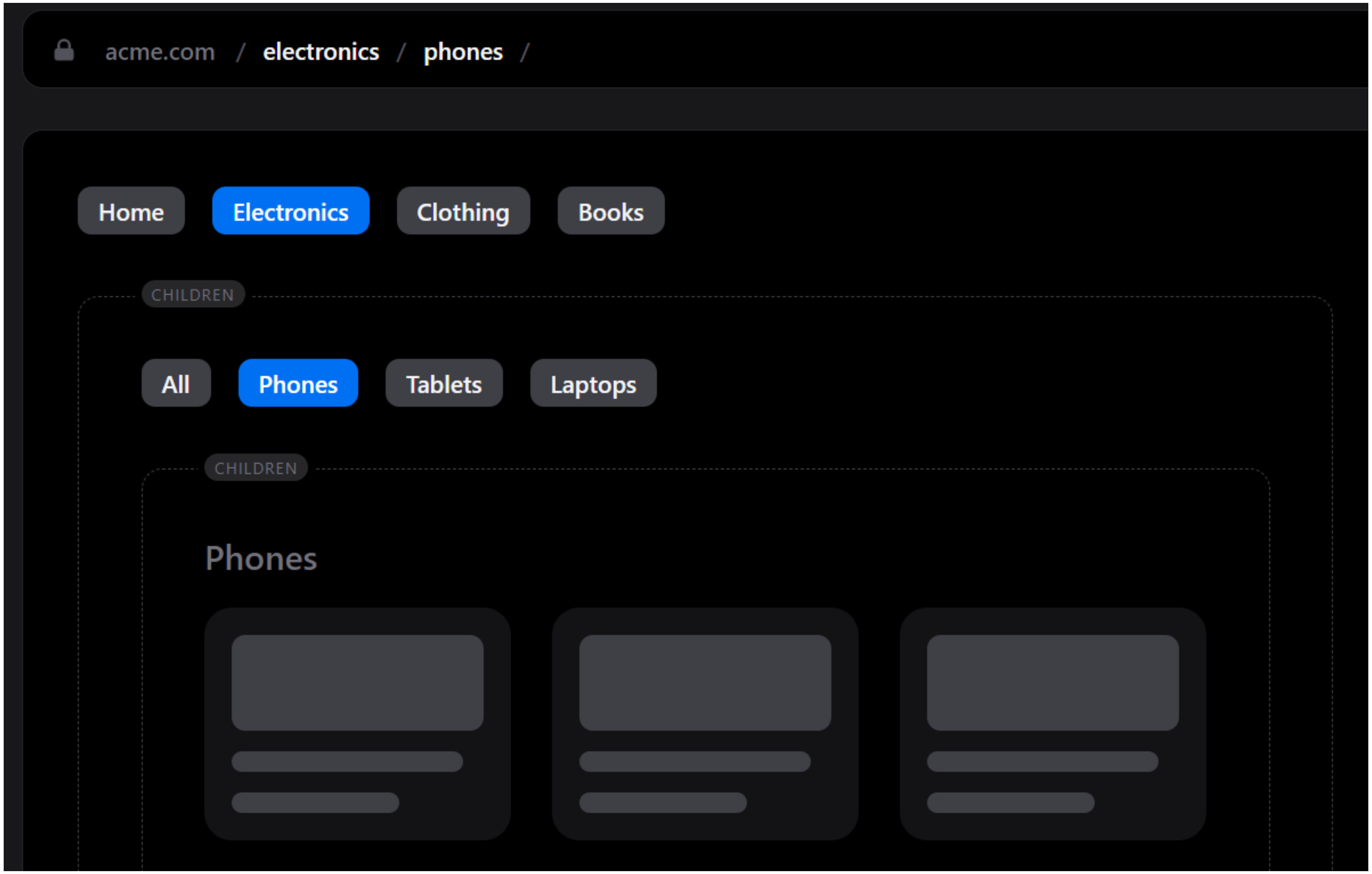


Organizing routes without affecting the URL path

- To organize routes, create a group to keep related routes together. The folders in parenthesis will be omitted from the URL (e.g. (marketing) or (shop))



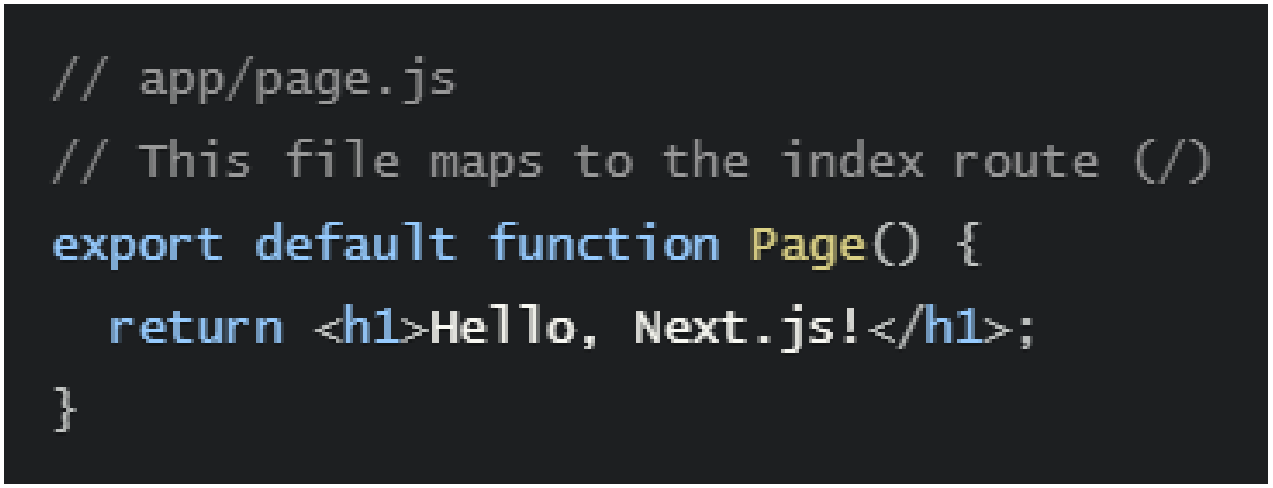
Nested Layout Example



https://app-dir.vercel.app/layouts/electronics/phones

React Server Components

- By default, files inside **app** folder and its subfolders will be rendered on the server as **React Server Components**
  - resulting in less client-side JavaScript and better performance
- Making the route accessible requires adding **`page.js`** file



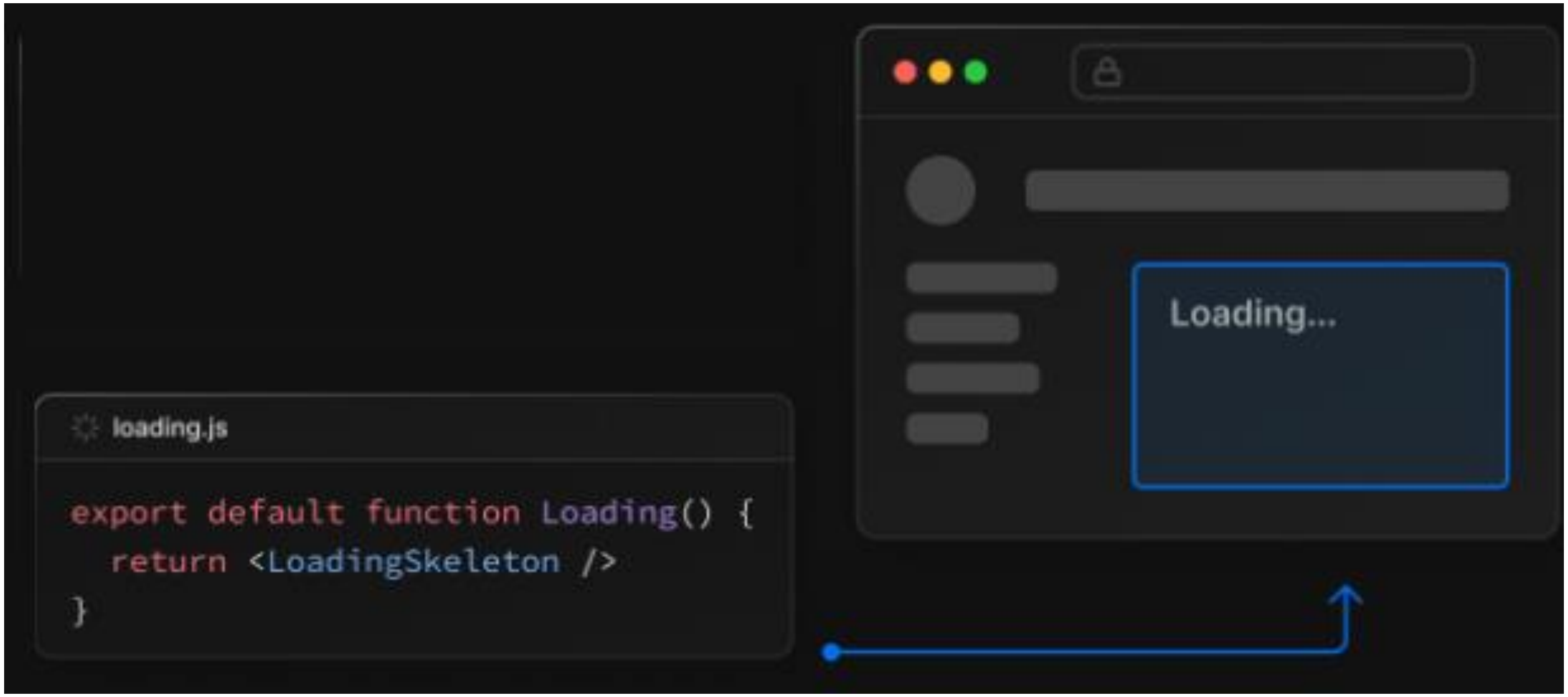
```
// app/page.js
// This file maps to the index route (/)
export default function Page() {
  return <h1>Hello, Next.js!</h1>;
}
```

UI Pages

- You can create a page by adding a **`page.js`** file inside a folder
- Files are used to define UI with new file conventions such as:
    - **layout.js**: define UI that is shared across multiple routes
    - **page.js**: define UI unique to a route
    - **loading.js**: show a loading indicator such as a spinner
    - **error.js**: show specific error information
    - **not-found.js**: render UI when the notFound function is thrown within a route segment

Loading UI

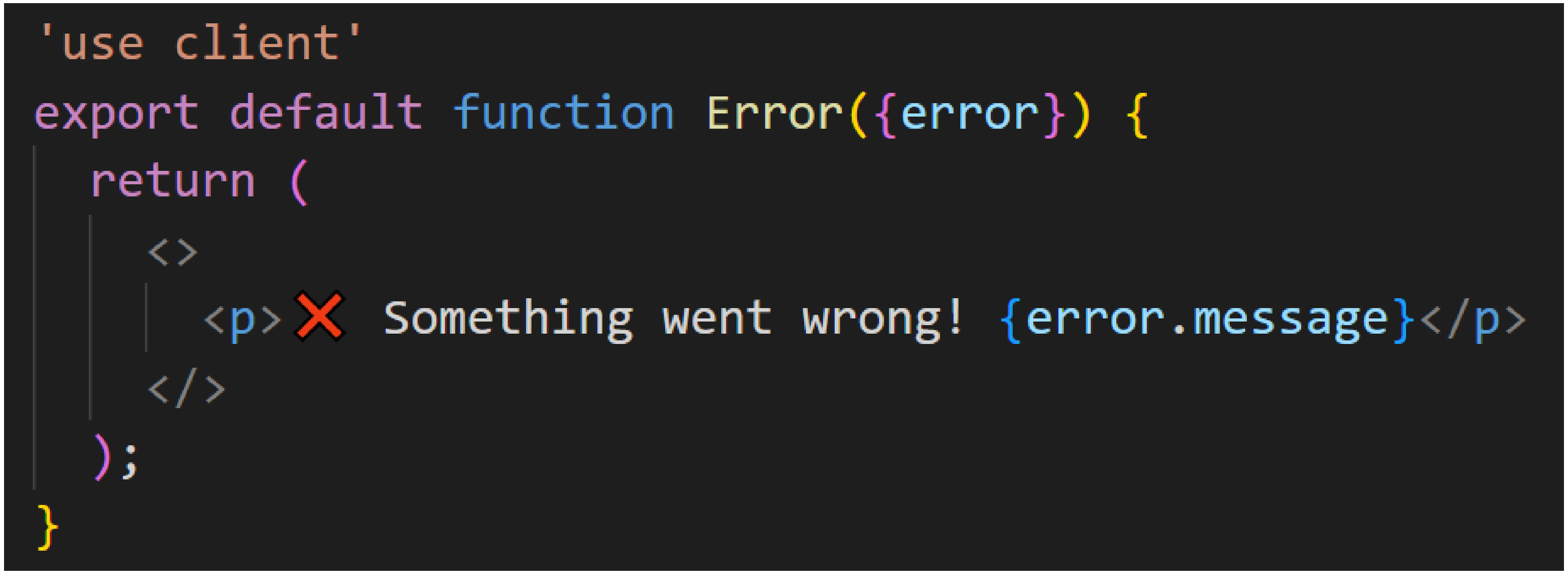
- **loading.js** return a loading indicator such as a spinner while the content of the route segment loads. The new content is automatically swapped in once rendering on the server is complete
  - This provides a better user experience by indicating that the app is responding


loading.js

```
export default function Loading() {
  return <LoadingSkeleton />
}
```

error.js

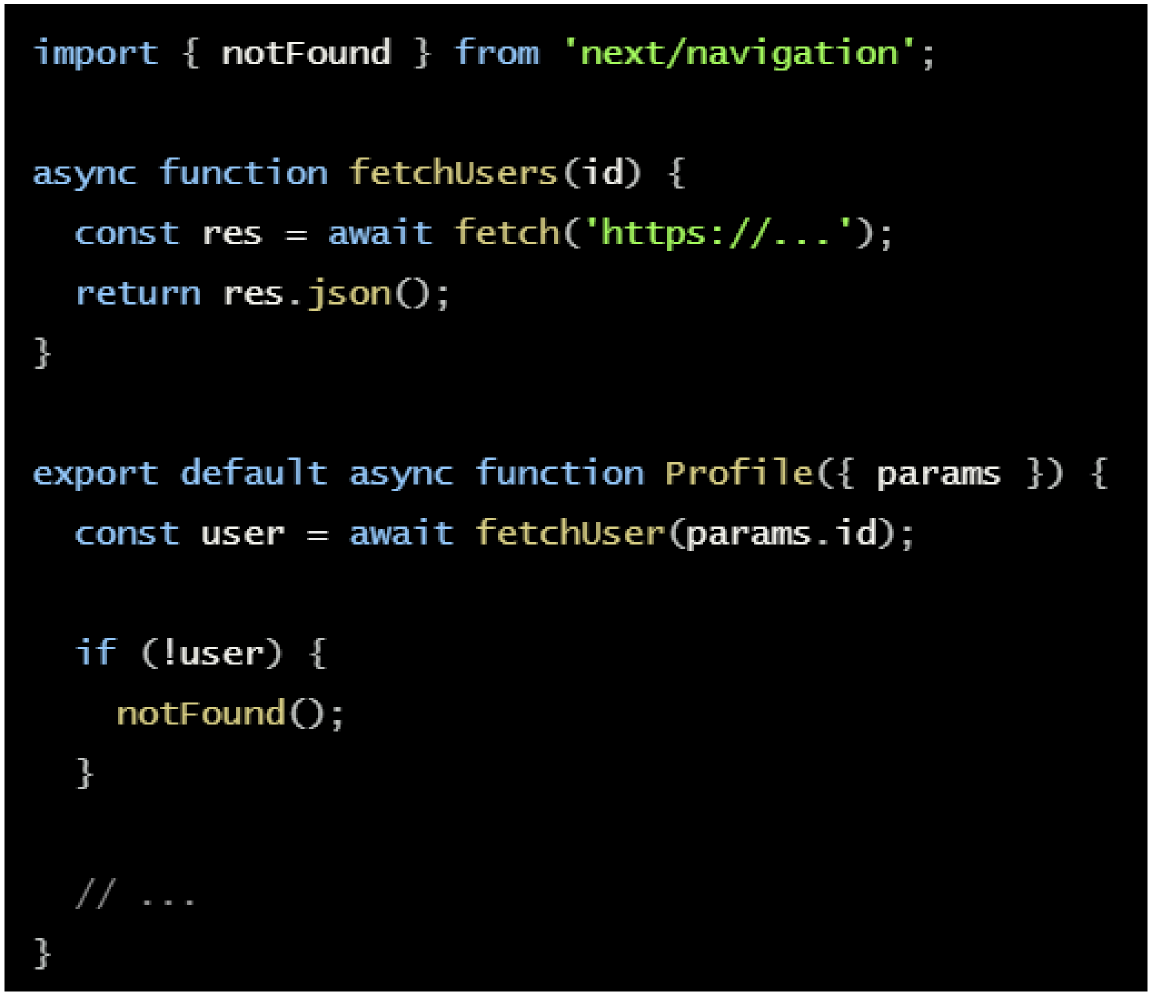
- `error.js` defines the error boundary for a route segment and the children below it. It can be used to show specific error information, and functionality to attempt to recover from the error
  - Should return a client-side component



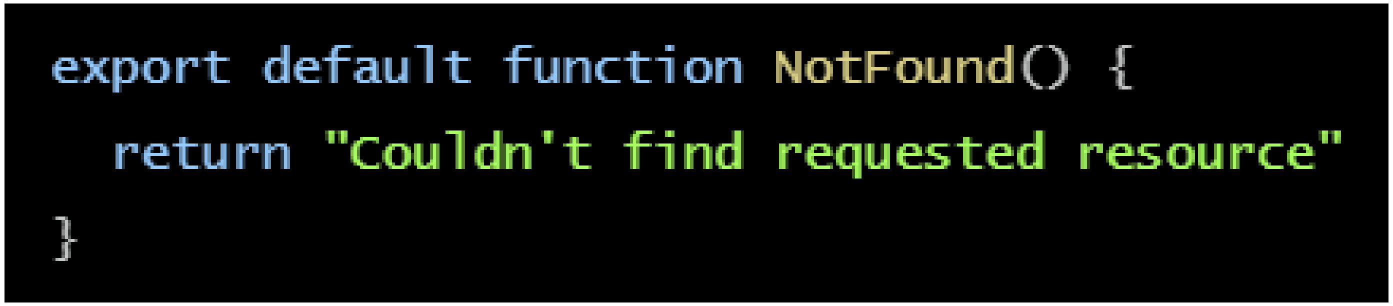
```
'use client'
export default function Error({error}) {
  return (
    <>
      <p>❌ Something went wrong! {error.message}</p>
    </>
  );
}
```

not-found.js

- **not-found.js**: is used to render UI when the notFound function is thrown within a route segment

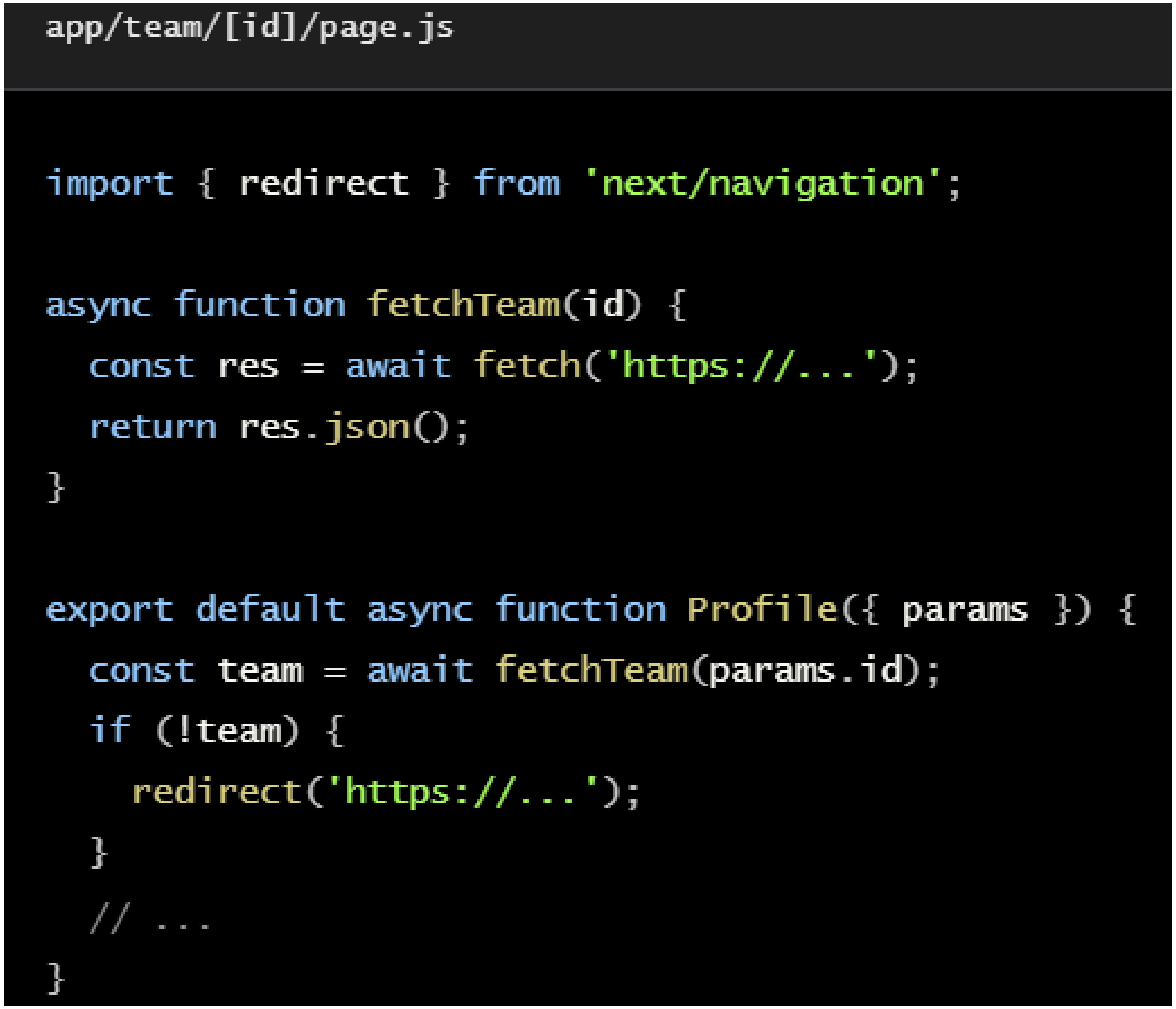


```
import { notFound } from 'next/navigation';

async function fetchUsers(id) {
  const res = await fetch('https://...');
  return res.json();
}

export default async function Profile({ params }) {
  const user = await fetchUser(params.id);

  if (!user) {
    notFound();
  }

  // ...
}
```



```
export default function NotFound() {
  return "Couldn't find requested resource"
}
```

redirect()



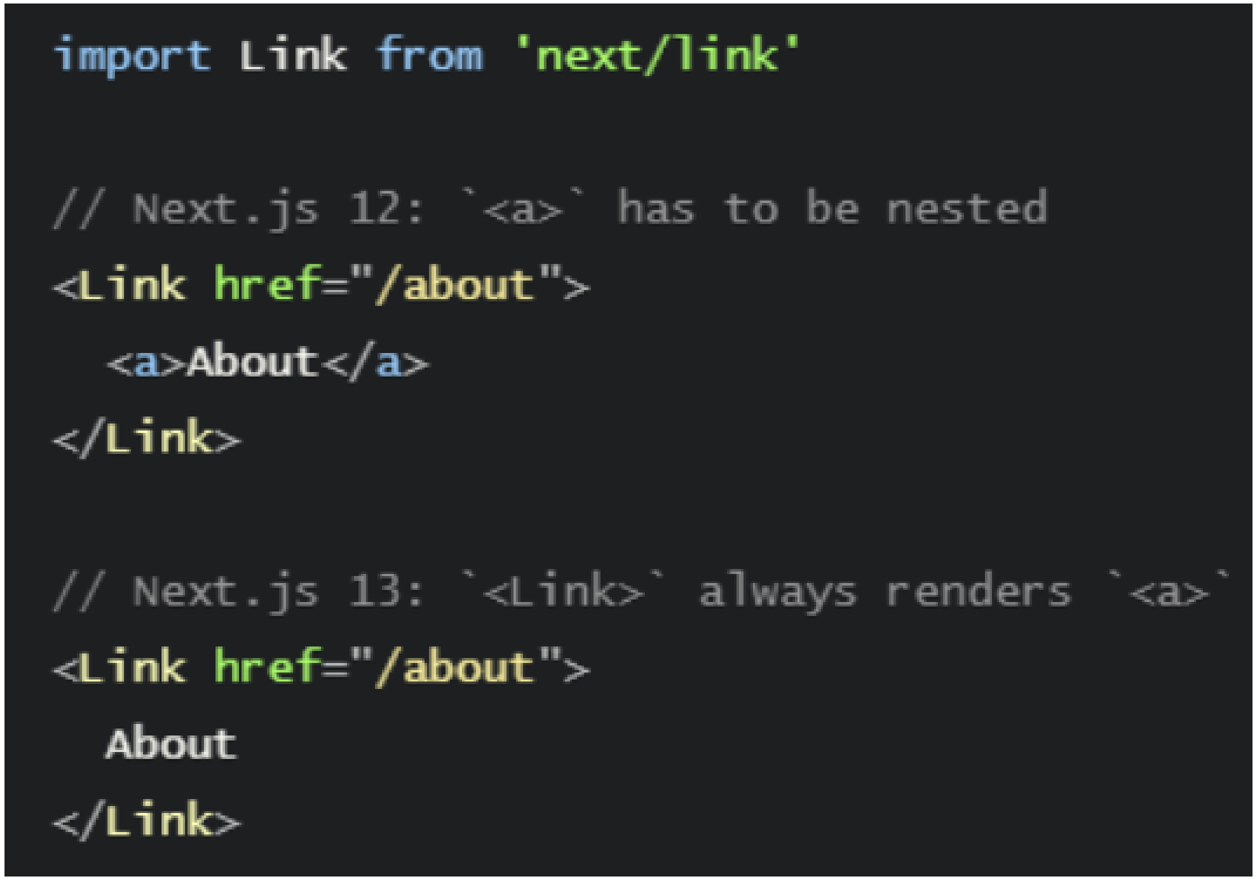
```
app/team/[id]/page.js
```

```js
import { redirect } from 'next/navigation';

async function fetchTeam(id) {
  const res = await fetch('https://...');
  return res.json();
}

export default async function Profile({ params }) {
  const team = await fetchTeam(params.id);
  if (!team) {
    redirect('https://...');
  }
  // ...
}
```

The redirect function allows you to redirect the user to another URL

next/link

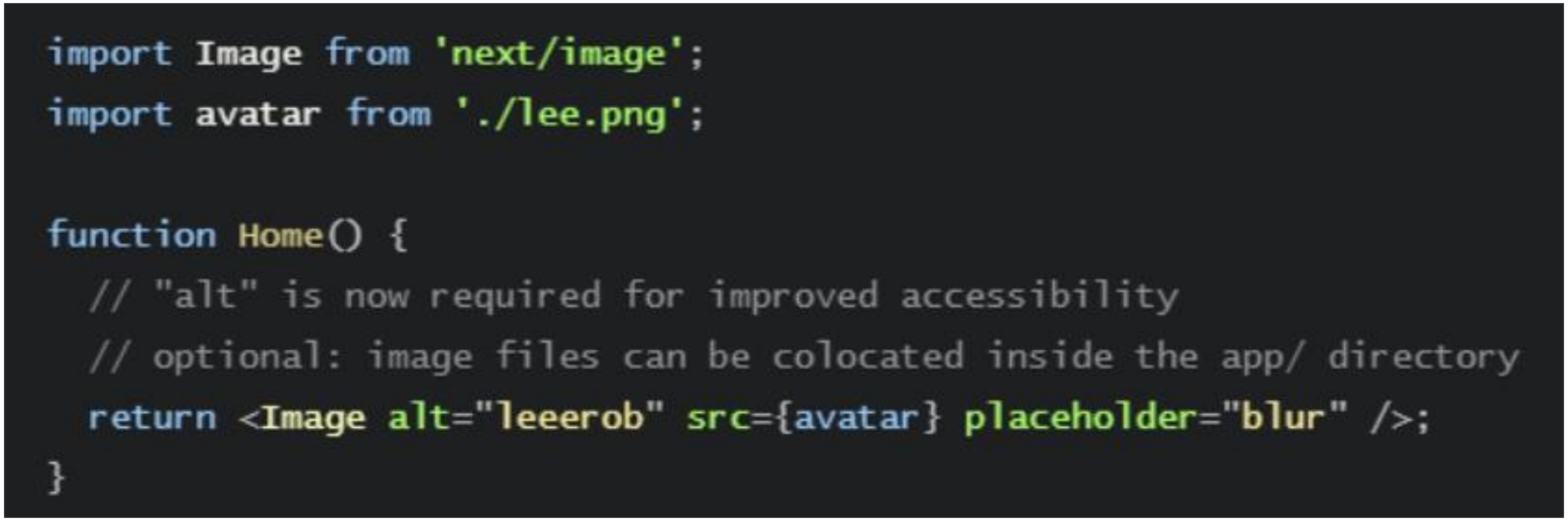
- next/link component no longer requires manually adding <a> tag as a child



```
import Link from 'next/link'

// Next.js 12: `<a>` has to be nested
<Link href="/about">
  <a>About</a>
</Link>

// Next.js 13: `<Link>` always renders `<a>`
<Link href="/about">
  About
</Link>
```

next/image

- Lazy loading and optimized files for increased performance with less client-side JavaScript



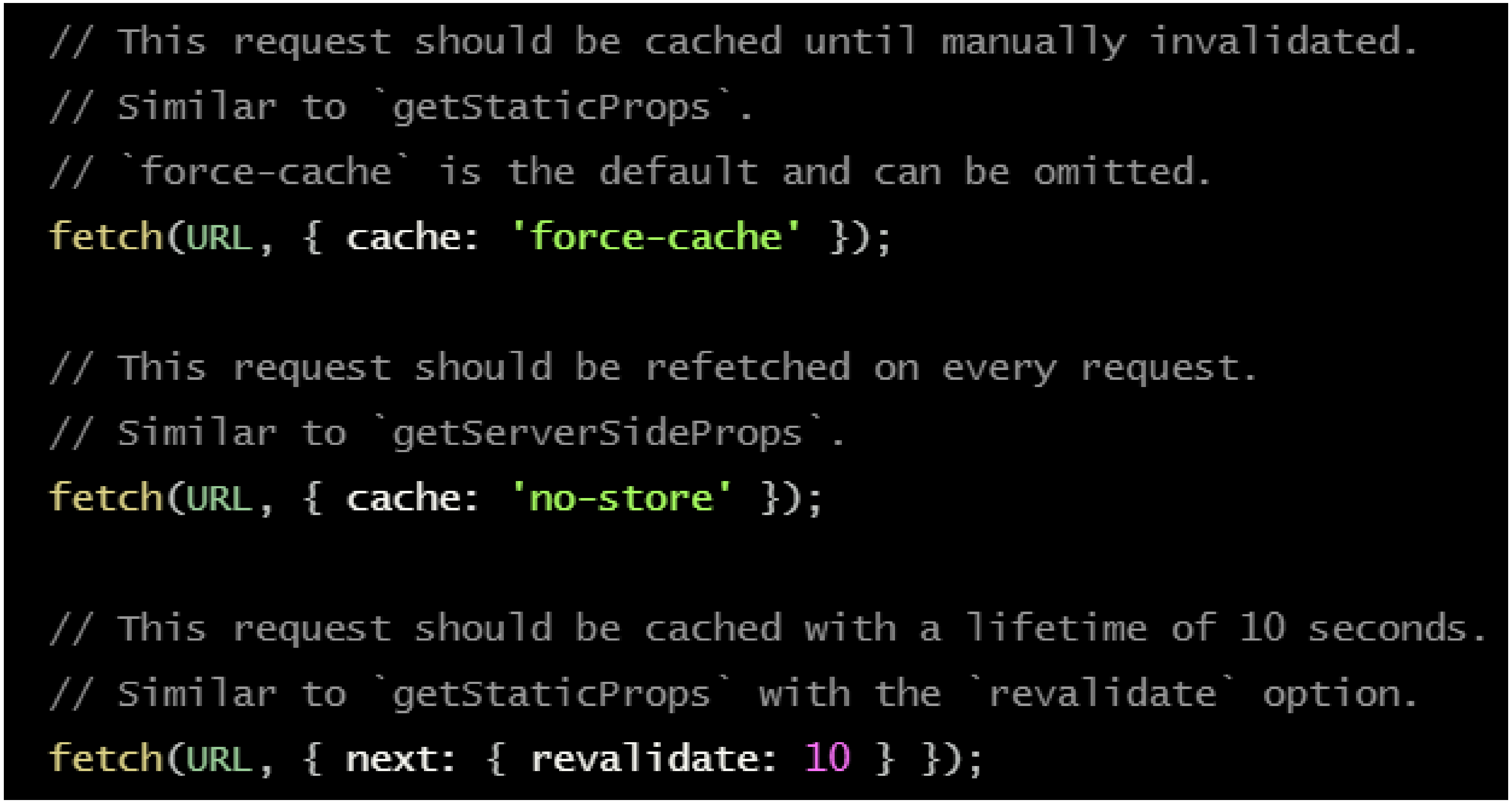
```
import Image from 'next/image';
import avatar from './lee.png';

function Home() {
  // "alt" is now required for improved accessibility
  // optional: image files can be colocated inside the app/ directory
  return <Image alt="leeerob" src={avatar} placeholder="blur" />;
}
```

Data Fetching

Back

Data Fetching

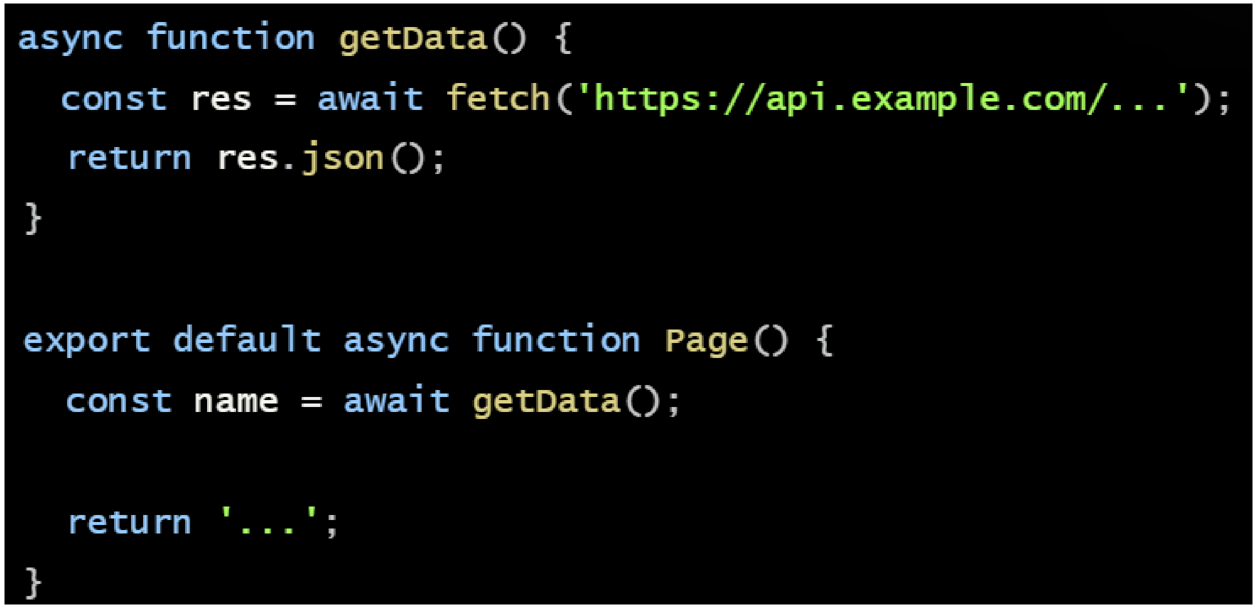
- You can call fetch with async/await directly within Server Components



```
// 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.
// Similar to `getServerSideProps`.
fetch(URL, { cache: 'no-store' });

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

Data Fetching

- fetch() is a Web API used to fetch remote resources and returns a promise
- Next.js extends the fetch options object to allow each request to set its own caching and revalidating
- You can fetch data in a component, a page or a layout
  - e.g., a blog layout could fetch categories which can be used to populate a sidebar component



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

export default async function Page() {
  const name = await getData();

  return '...';
}
```

Server-Side Rendering (SSR)

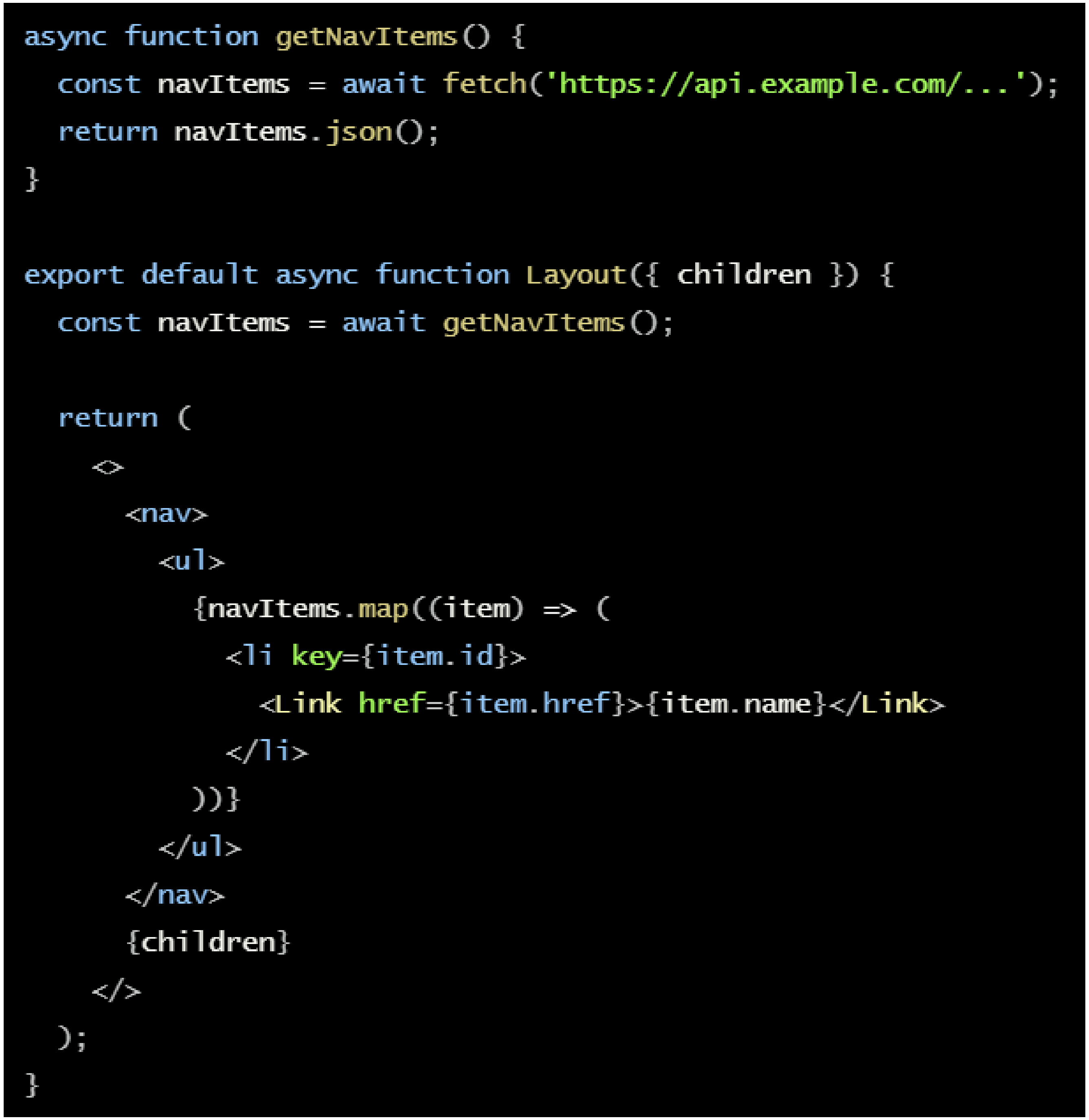
- Next.js extends the Web fetch() API to allow configuring caching and revalidation

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

- `cache: 'no-store'` is the default, Next.js fetches the resource from the remote server on every request

- To catch fetch response use

  `cache:'force-store'`

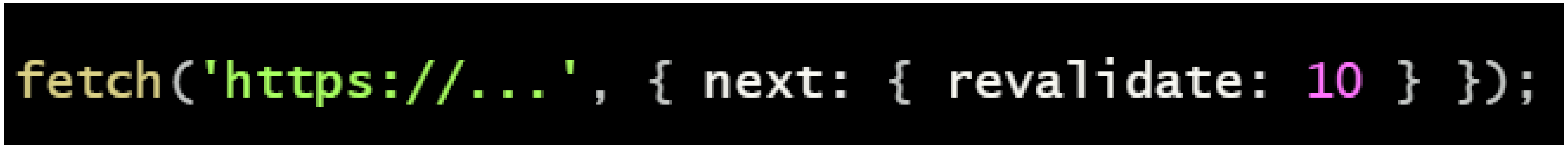


```
async function getNavItems() {
  const navItems = await fetch('https://api.example.com/...');
  return navItems.json();
}

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}
    </>
  );
}
```

Static Site Generation Example

Revalidating Data

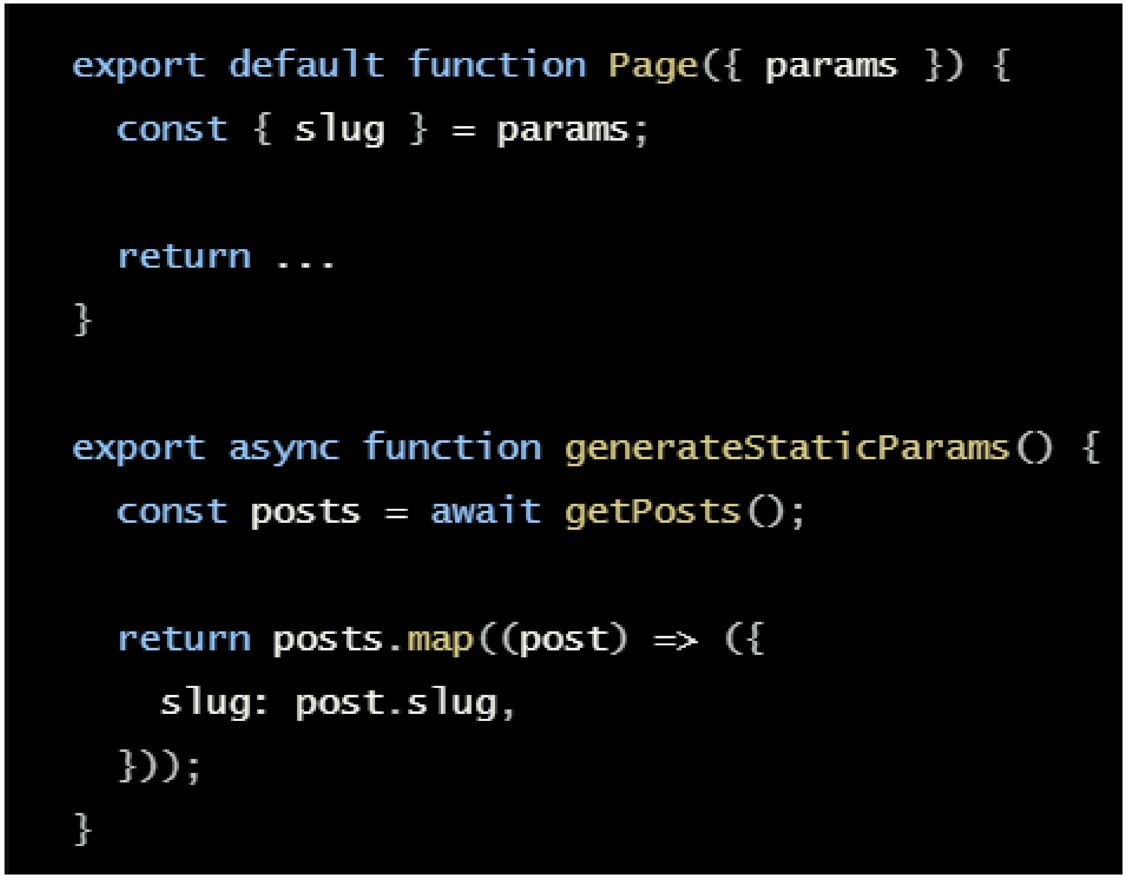
- To revalidate cached data, you can use the `next.revalidate` option in fetch()
  - => Incremental Static Regeneration (ISR)



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

Generate Static Params

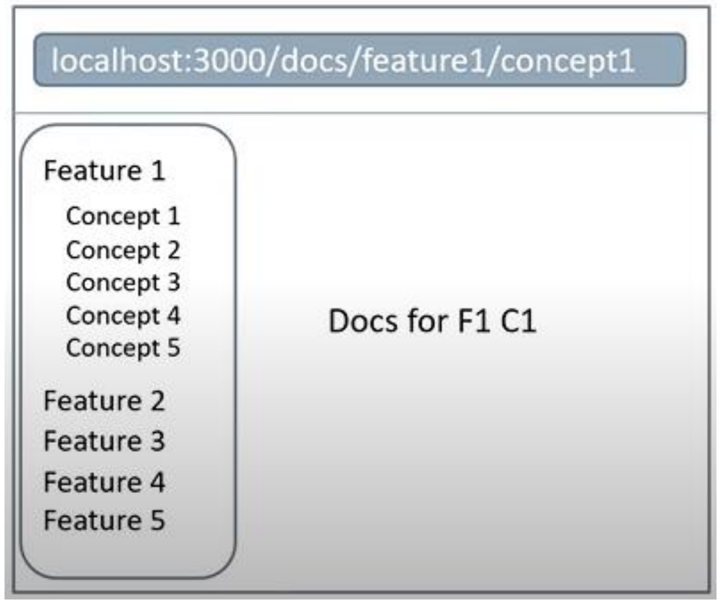
- The `generateStaticParams` function can be used in combination with dynamic route segments to define the list of route segment parameters that will be statically generated at build time



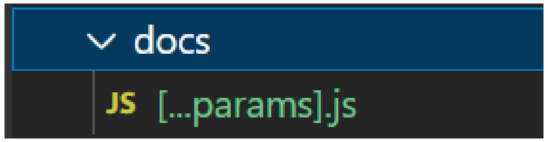
```
export default function Page({ params }) {
  const { slug } = params;

  return ...
}

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

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

Catch-All Route Example

See posted example



```
const router = useRouter()
const { params  } = router.query
console.log(params)

if (params.length === 2) {
  return (
    <h1>
      Viewing docs for feature {params[0]} and concept {params[1]}
    </h1>
  )
} else if (params.length === 1) {
  return <h1>Viewing docs for feature {params[0]}</h1>
}
```



Linking between pages

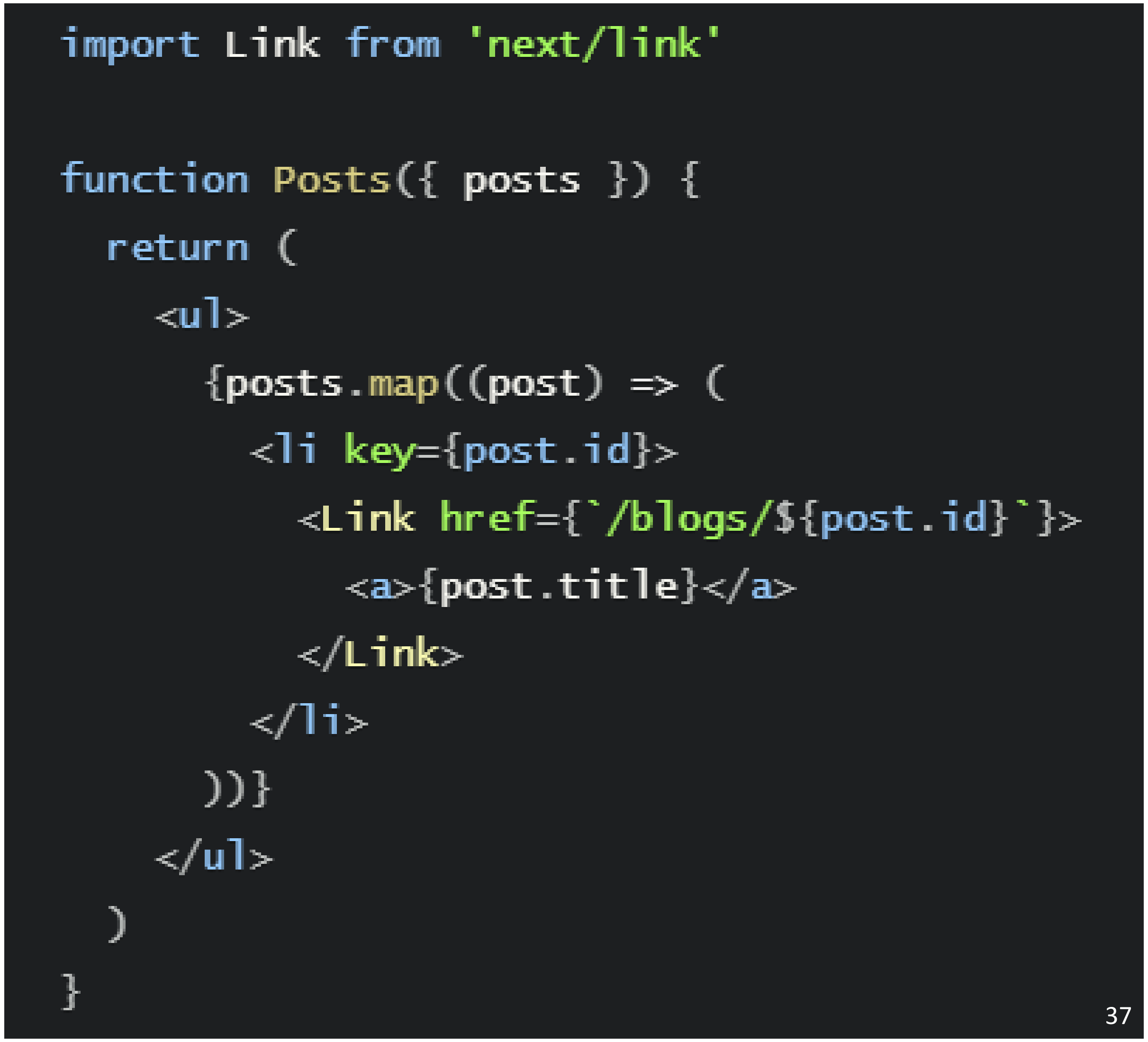
- The Next.js router provides a React component called **Link** to do **client-side route** transitions between pages, similar to a single-page application
    - **href** specify the route associated with the link
    - Pages for any <Link /> in the viewport (visible to the user) will be prefetched by default (including the corresponding data) for pages using Static Generation. The corresponding data for server-rendered routes is not prefetched.

```
import Link from 'next/link'
export default function Home() {
return ( <ul>
    <li> <Link href="/"> <a>Home</a> </Link> </li>
    <li> <Link href="/about"> <a>About Us</a> </Link> </li>
   </ul>)
}
```

Linking to dynamic paths

- Links can be created for dynamic paths

E.g., creating links to access posts for a list which have been passed to the component as a prop



```
import Link from 'next/link'

function Posts({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>
          <Link href={`/blogs/${post.id}`}>
            <a>{post.title}</a>
          </Link>
        </li>
      ))}
    </ul>
  )
}
```

useRouter

- **useRouter** hook to access the router object inside any app component
- Router properties include:
  - **query**: returns the query string parsed to an object, including dynamic route parameters
  - **asPath**: returns the path as shown in the browser including the query params

```
import { useRouter } from 'next/router'
const Post = () => {
      const router = useRouter()
      const { pid } = router.query
      return <p>Post: {pid}
        Path: router.asPath </p>
} export default Post
```

For /posts/1

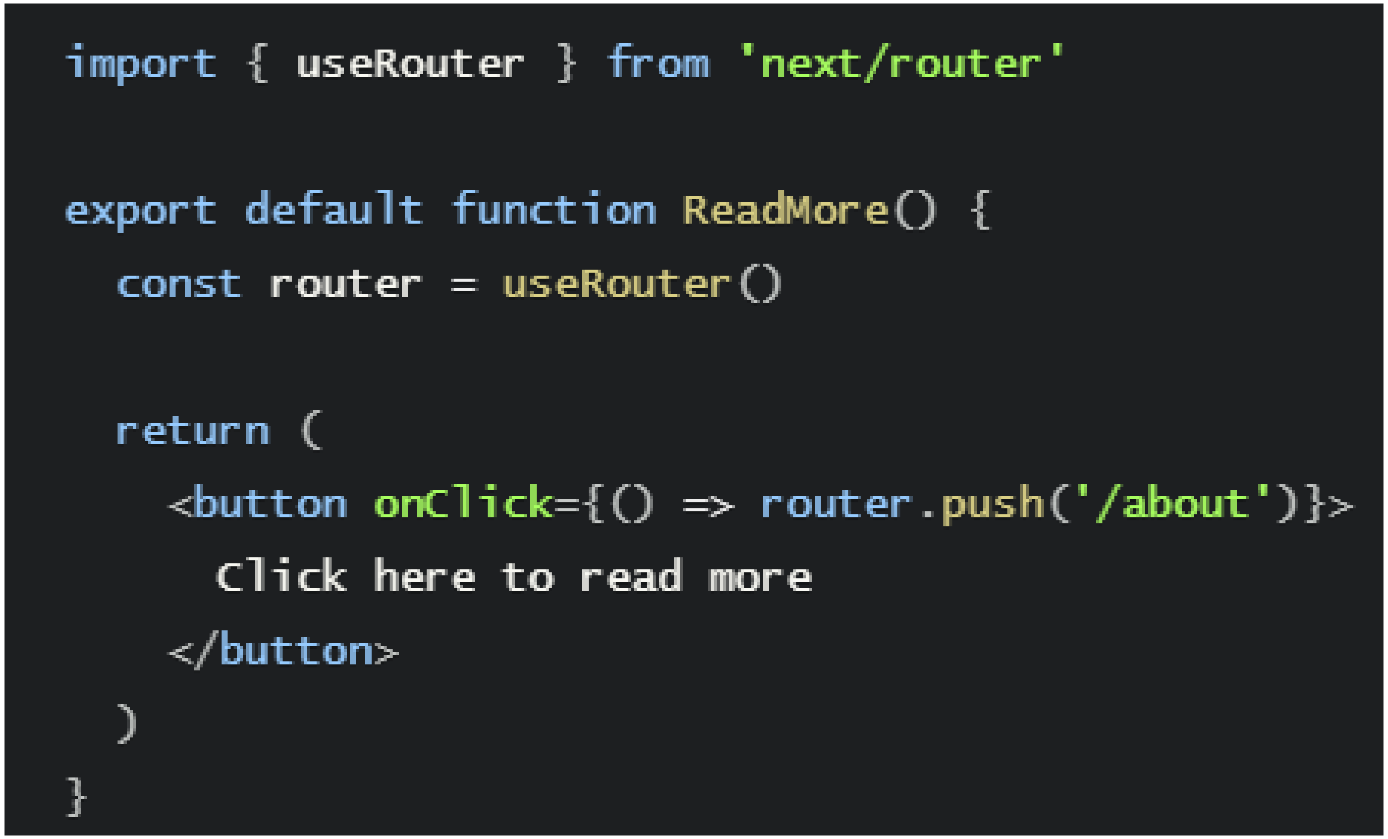
pid will be **1**

Router.asPath will return /posts/1

Router push method

- Router **push** method can be used for programmatic client-side routing

E.g., navigating to *pages/about.js*



```
import { useRouter } from 'next/router'

export default function ReadMore() {
  const router = useRouter()

  return (
    <button onClick={() => router.push('/about')}>
      Click here to read more
    </button>
  )
}
```

Summary

- Next.js = React-based full stack web framework that allows creating user interfaces, static pages, server-side rendered pages, and Web API
- Next.js has a **file-system based router**: when a file is added to the **app** directory, it's automatically available as a route
  - In Next.js you can add brackets to the file name of a page to create a dynamic route
- To create API Route simply add a handler function to a route.js file under app folder

Resources

- Learn Next.js

http://nextjs.org/learn

- E-commerce Demo

https://nextjs.org/commerce

- Useful list of resources

https://github.com/unicodeveloper/awesome-nextjs

Resources

Next.js 15 Documentation

https://nextjs.org/

Next.js Fetch API

https://nextjs.org/docs/api-reference/fetch