

Web Pages

using **NEXT.js**

Outline

- Pages
- Links
- Layouts
- Data Fetching

What is Next.js?

- Next.js = React-based full stack web framework that allows creating component-based **Web pages** and **Web API**
- It provides a large set of features including:
 - File system-based routing
 - Shared layouts having UI that is shared between multiple pages
 - Caching the Web pages on the server-side to improve performance and avoid regenerating the page per request

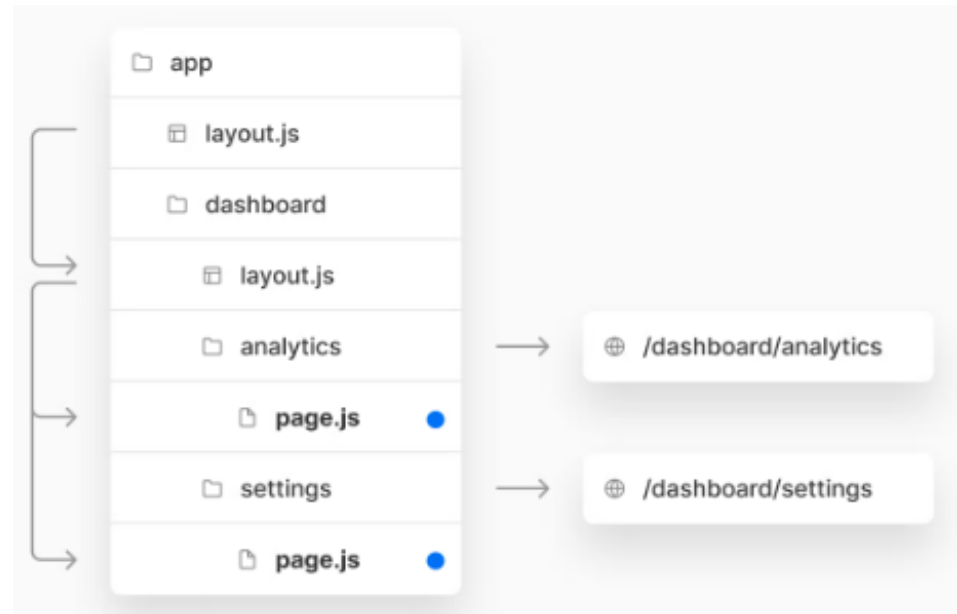
Why Server-side Web Pages?

- Improve Search Engine Optimization (SEO) to enable search engines to discover and index the app pages
- Faster initial app start by reducing client-side JavaScript that the browser has to download, parse and execute to render the result in the browser (which could take up to a few seconds for a large application)
- Access to resources that the client can't access such as direct access to a database
- Hide sensitive data from the client such as passwords and API Keys
- Allow caching the Web pages on the server-side to improve performance and avoid regenerating the page per request

Project Folder Structure

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

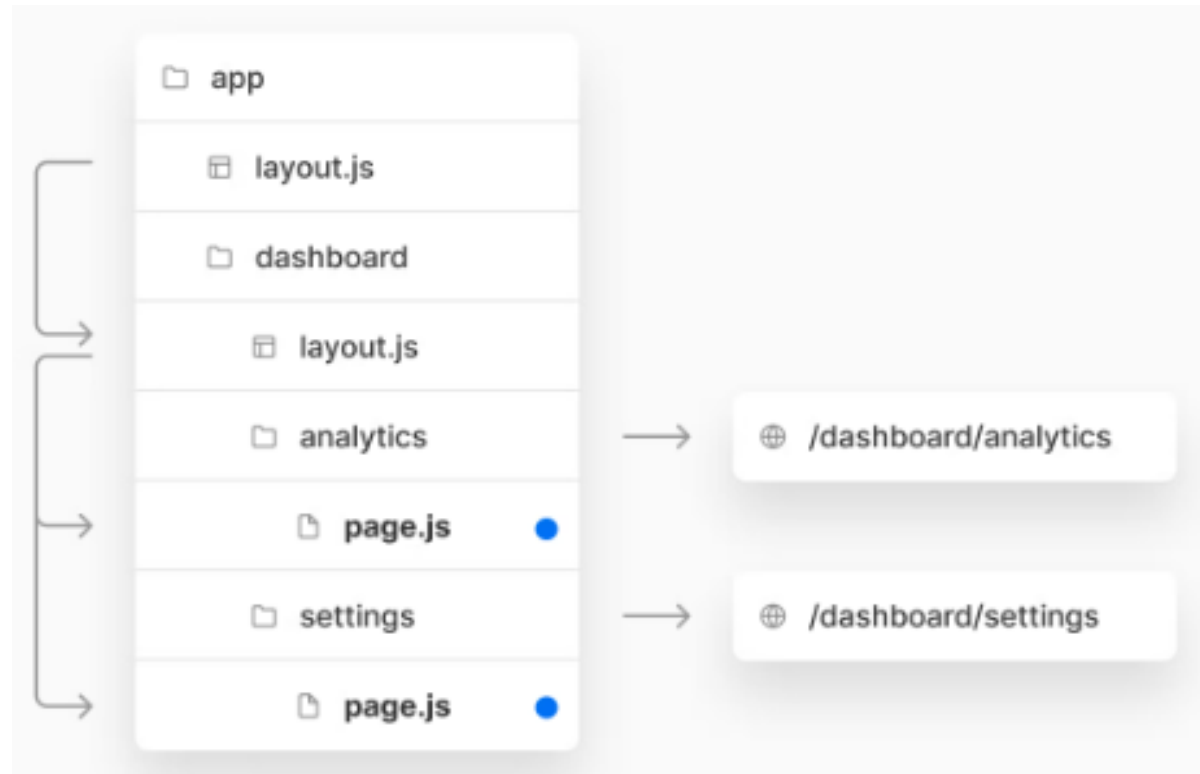
Page



UI Pages

- You can create a page by adding a **page.js** file inside a subfolder under app folder
 - Can colocate your related 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



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

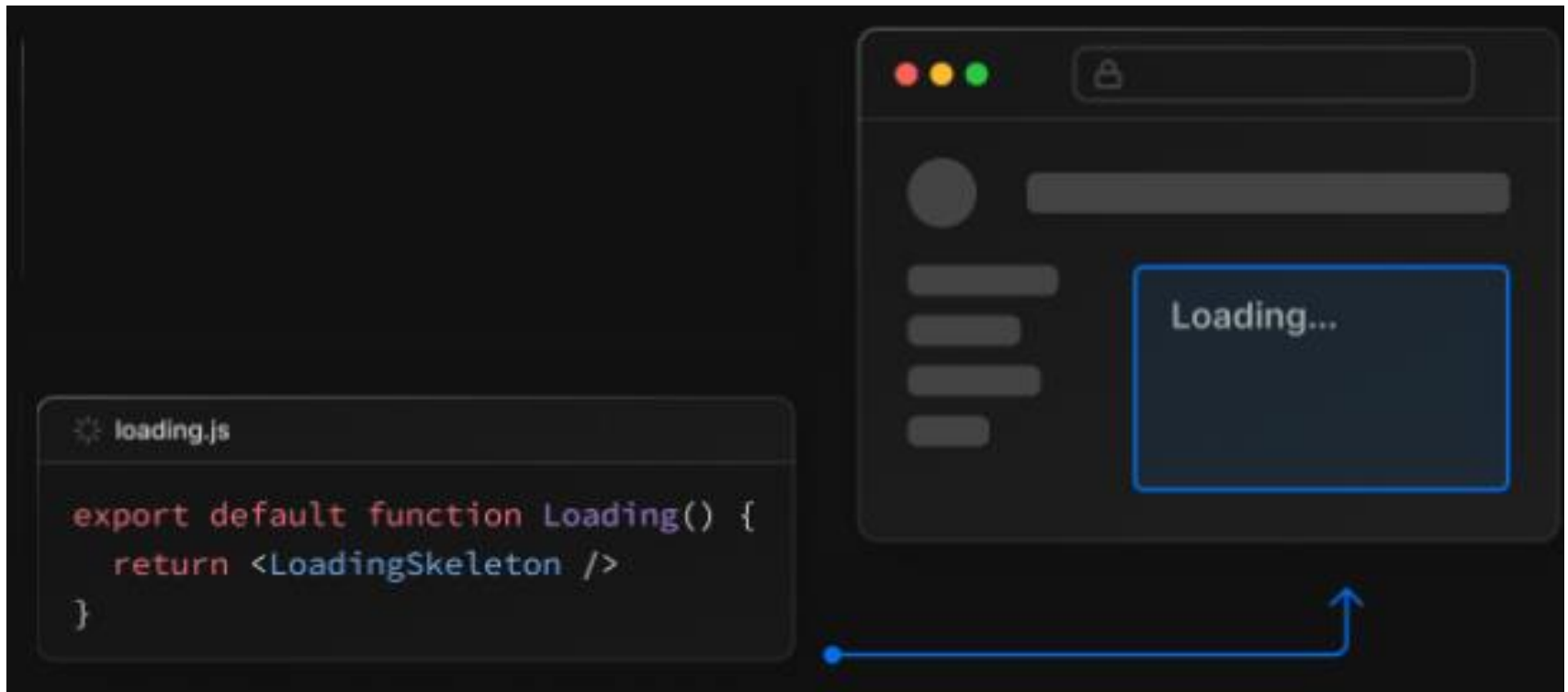
```
// app/page.js
// This file maps to the index route (/)
export default function Page() {
  return <h1>Hello, Next.js!</h1>;
}
```


UI Pages

- You can create a page by adding a **page.js** file inside a folder
- File name conventions used to define the app UI:
 - **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 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



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"
}
```

Links



Linking between pages

- The Next.js router provides a React component called **Link** to do **client-side route** transitions between pages
 - **href** specify the route associated with the link
 - Pages for any `<Link />` 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="/"> Home </Link> </li>
    <li> <Link href="/about"> About Us </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

```
import Link from 'next/link'

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

redirect()

app/team/[id]/page.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

useRouter push method

- **useRouter** hook can be used for programmatic **client-side** routing
 - E.g., use **push** method to navigate to *savings/current* accts

```
"use client"
import { useRouter } from "next/navigation"
export default function AcctTypeDropdown(){
  const router = useRouter();
  return (<>
    <select onChange={(e) => {
      router.push(`/accounts?type=${e.target.value}`)
    }}
    > <option value="Savings">Savings</option>
      <option value="Current">Current</option>
    </select>
  </>
)}
```

usePathname & useSearchParams

- **usePathname**: returns the path of the current Url
- **useSearchParams**: returns query params of the current Url

← → ↻ ⓘ http://localhost:3001/accounts?type=Savings

pathname: /accounts

searchParams - type: Savings

```
"use client"
import { usePathname, useSearchParams } from "next/navigation"
export default function AcctTypeDropdown({ onChange }) {
  const pathname = usePathname()
  const searchParams = useSearchParams()
  return (
    <>
      <p>pathname: {pathname}</p>
      <p>searchParams - type: {searchParams.get("type")}</p>
    </>
  );
}
```

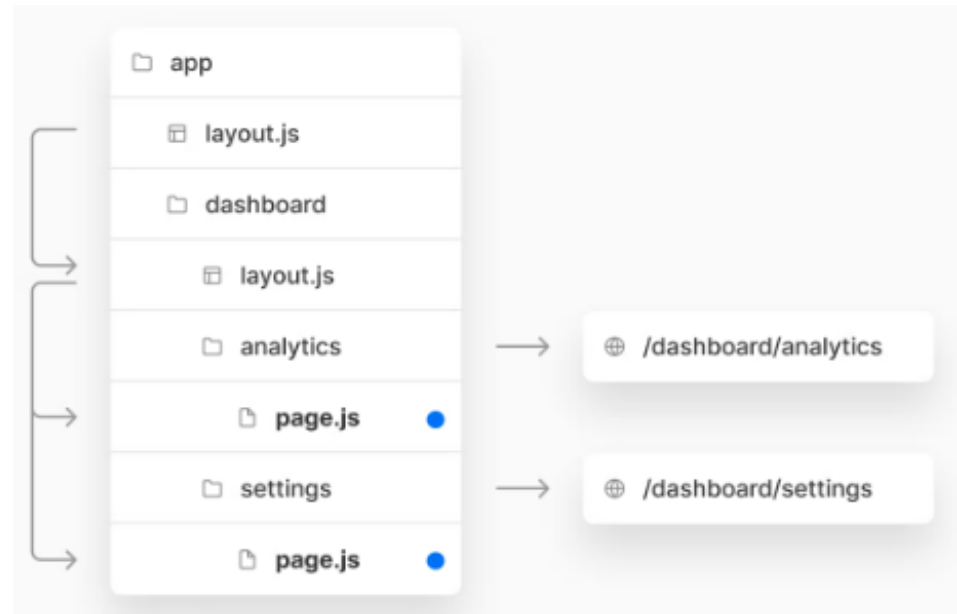
next/image

- Allows **lazy loading** of images (only loaded when they become visible) 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="leerob" src={avatar} placeholder="blur" />;
}
```

Layout

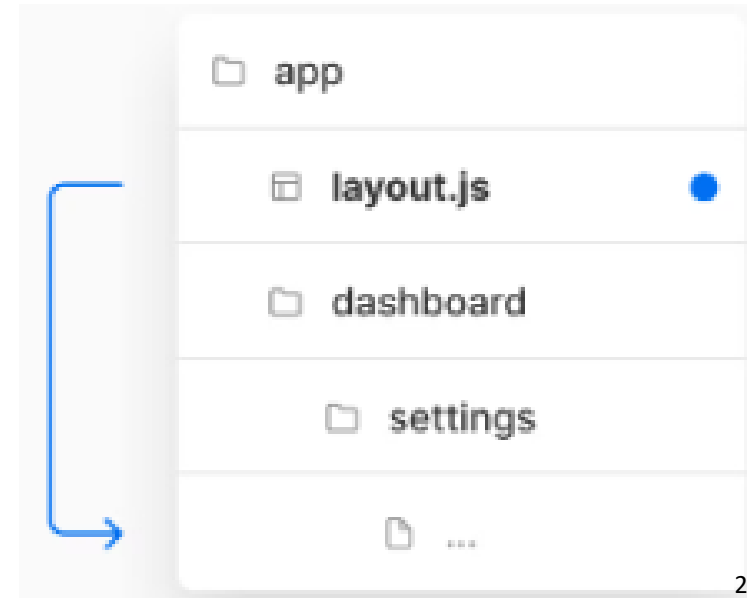


Layouts

- A layout is UI that is shared between multiple pages
 - On navigation, layouts preserve state, remain interactive, and do not re-render.
 - 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

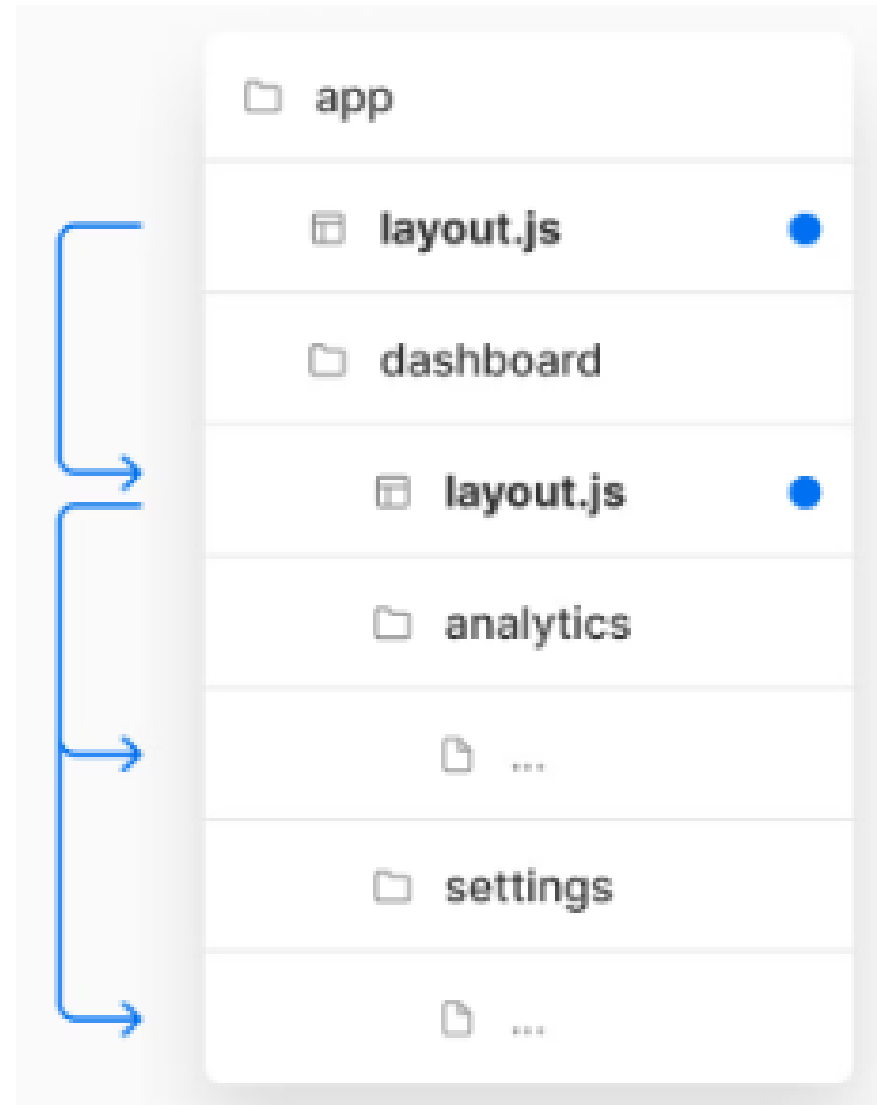
There are 2 types of layouts:

- **Root layout:** in **app** folder and applies to all routes
- **Regular layout:** inside a specific folder and applies to associated route segments



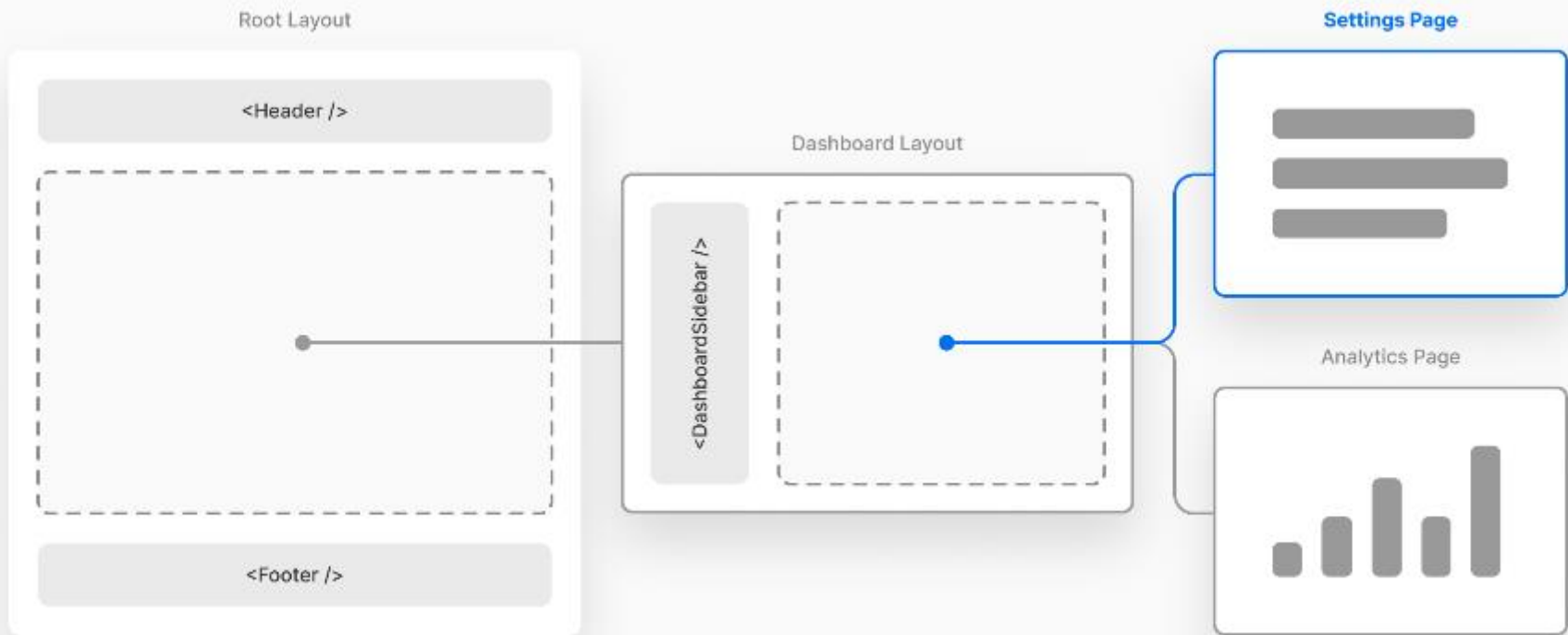
Nesting Layouts

- 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/***



Pages are Wrapped in Layouts

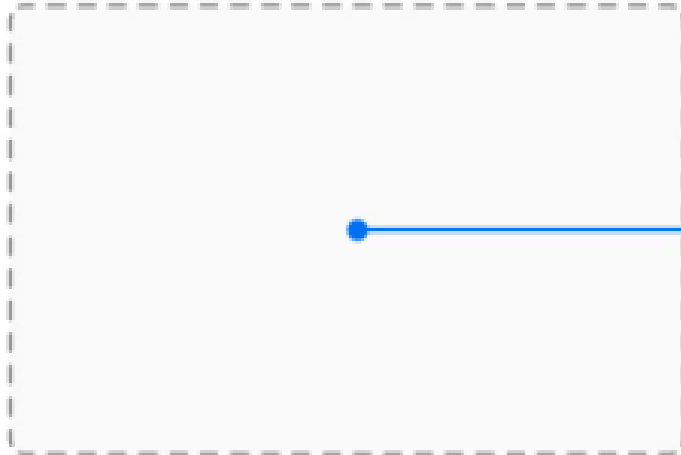
- When a user visits `/dashboard/settings` Next.js will render the `page.js` file inside the settings folder **wrapped** in any layouts that exist further up the subtree (i.e., `page.js` will be **wrapped** in the Dashboard Layout which will in turn be wrapped in the Root Layout)



AnalyticsPage will be wrapped in the Dashboard Layout which will in turn be wrapped in the Root Layout

Root Layout

<Header />



<Footer />

Dashboard Layout

<DashboardSidebar />

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

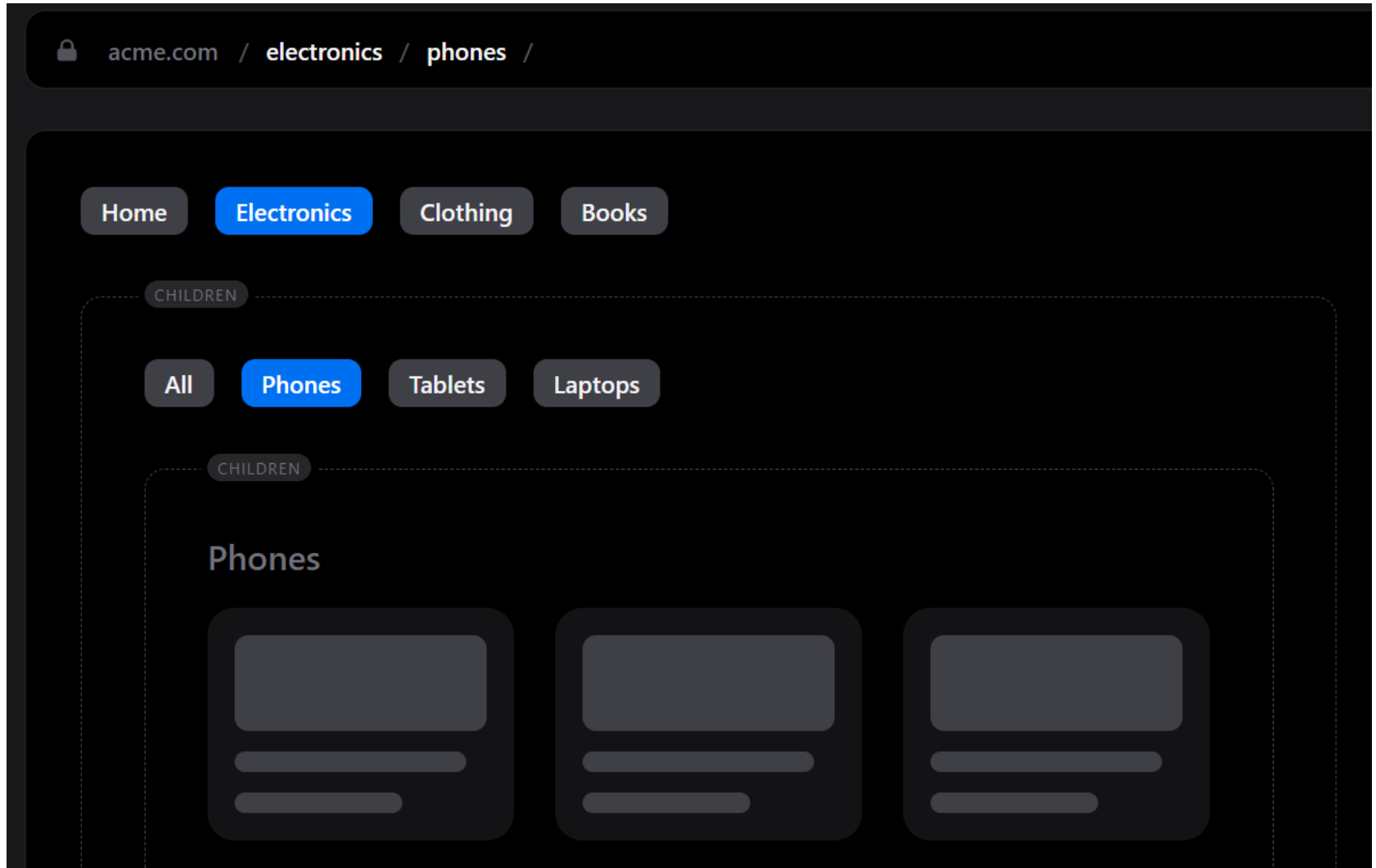
```
// Regular layout (app/dashboard/layout.js)
// - Applies to route segments in app/dashboard/*
export default function DashboardLayout({ children }) {
  return (
    <
      <DashboardSidebar />
      {children}
    >
  )
}
```

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

The above wrapping of the page in layouts would render the following component hierarchy:

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


Nested Layout Example



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

Data Fetching

Data Fetching using fetch

- **fetch()** is a Web API used to fetch remote data
- You can fetch data with `async/await` in a component, a page or a layout
 - e.g., a blog page could fetch categories which can be used to populate a dropdown

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

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.
```

```
fetch(URL, { cache: 'no-store' });
```

```
// This request should be cached with a lifetime of 10 seconds.
```

```
fetch(URL, { next: { revalidate: 10 } });
```

Summary

- Next.js = React-based full stack web framework that allows creating component-based Web pages and Web API
- Next.js has a **file-system based router**: when a subfolder is added to the **app** directory, it's automatically available as a route
 - In Next.js you can add brackets to the subfolder name to create a dynamic route
- A layout A layout is UI that is shared between multiple pages
- You can **fetch** data with `async/await` in a component, a page or a layout

Resources

- Learn Next.js

<https://nextjs.org/docs>

- Next.js Fetch API

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