CMPS 356

Next.js 13
New Features

Dr. Abdelkarim Erradi

CSE@QU

Outline

- New app folder and Layout
- Data Fetching

New app folder and Layout

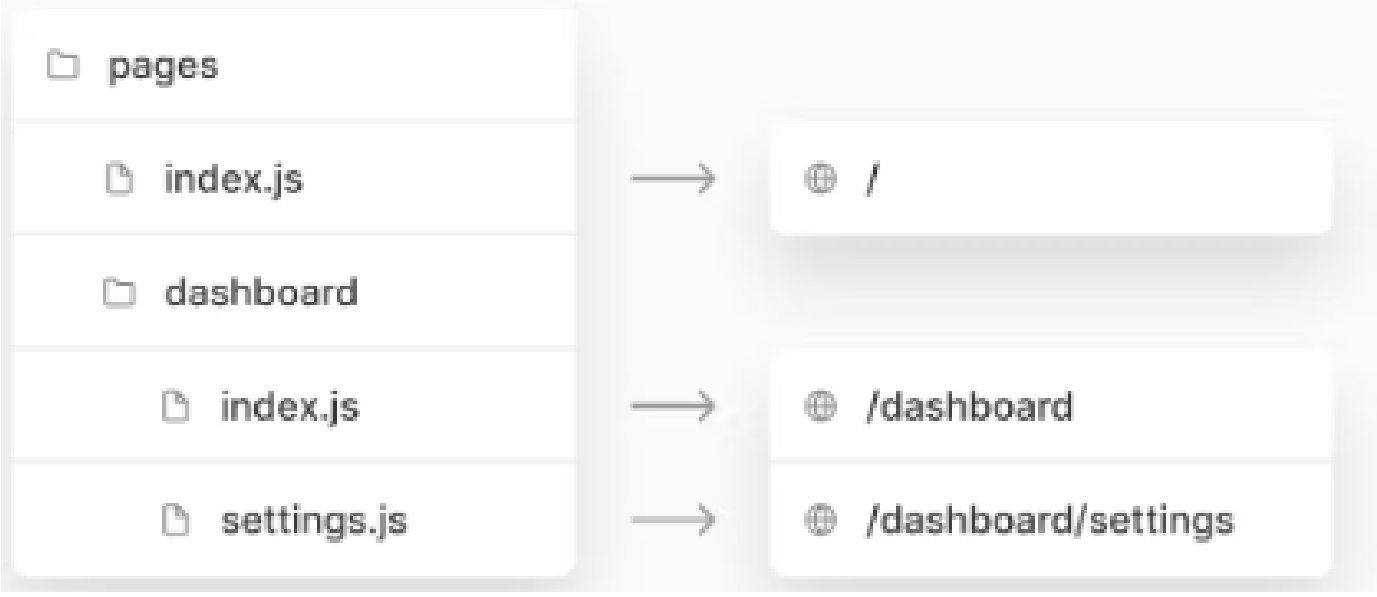
New app Directory

Next.js introduced the **app/** directory offering:

- **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
- The app/ directory can coexist with the existing pages directory for incremental adoption

Routing prior to Next.js 13

- Next.js uses the file system to map individual folders and files in the **pages** directory to routes accessible through URLs
  - Each page file exports a React Component and has an associated route based on its file name
  - Supports Dynamic Routes (including catch all variations) with the `[param].js`, `[...param].js` and `[[...param]].js` conventions

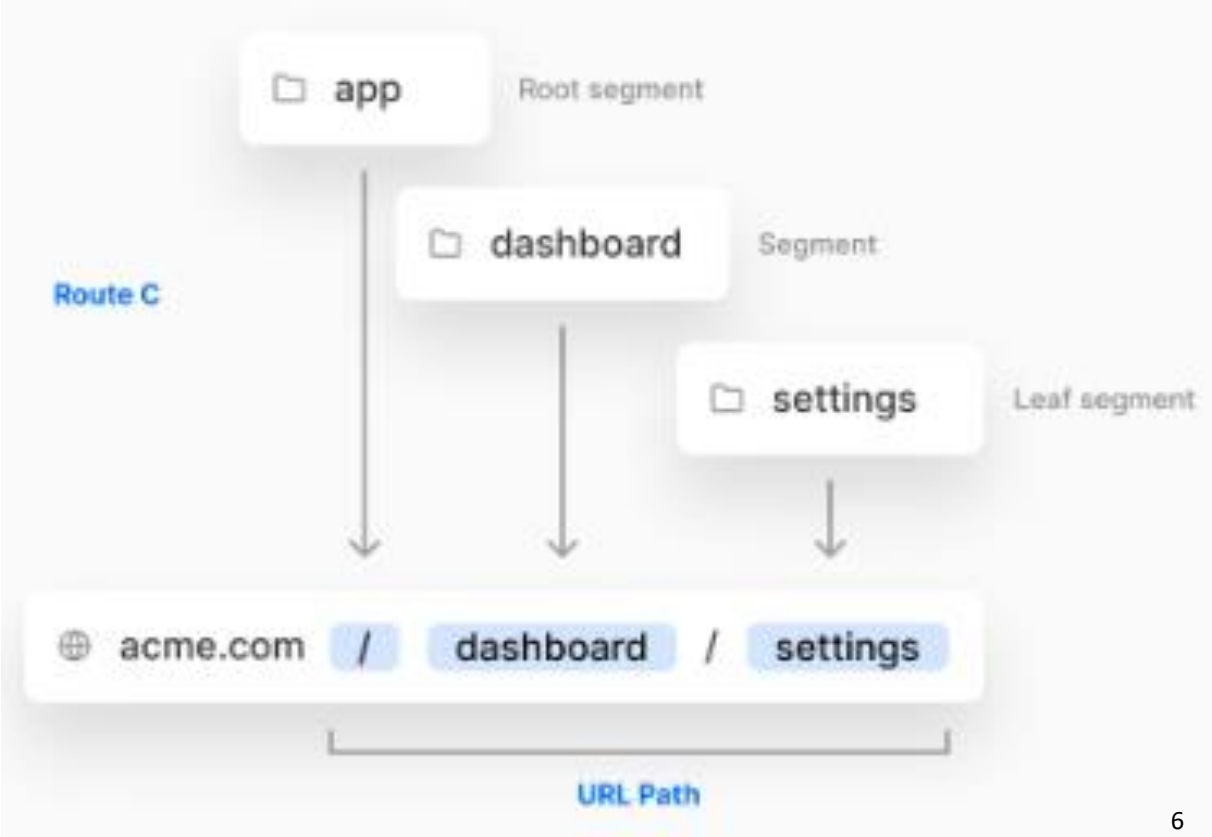


Next.js 13 Routing

- 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

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

- E.g., add a new `/dashboard/settings` route by nesting two new folders in the app directory

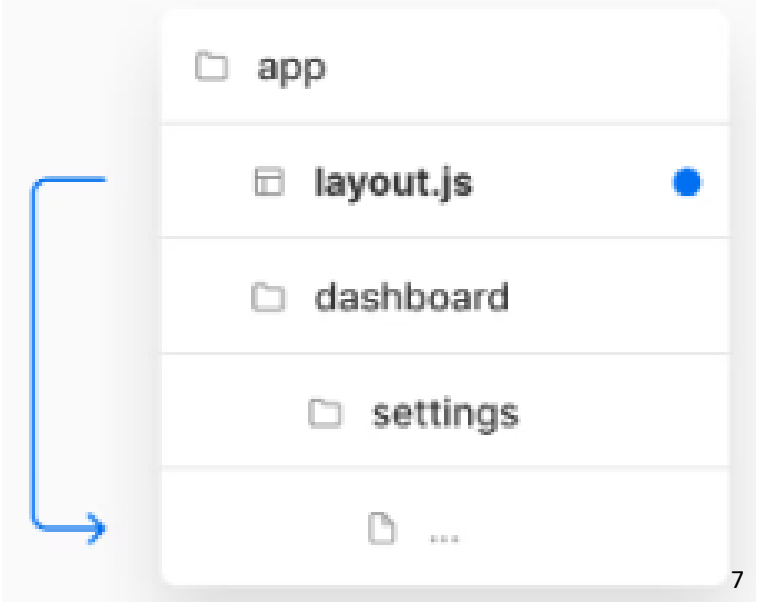


Layouts

- 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

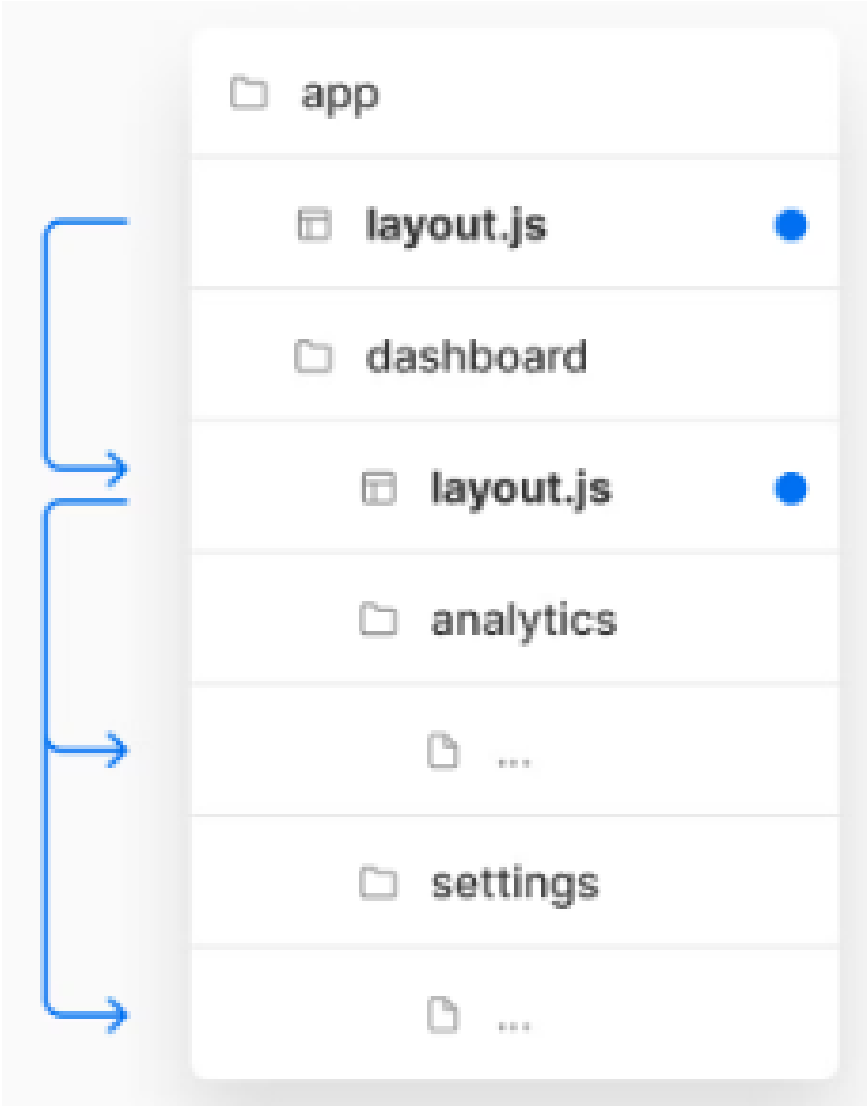
There are 2 types of layouts:

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

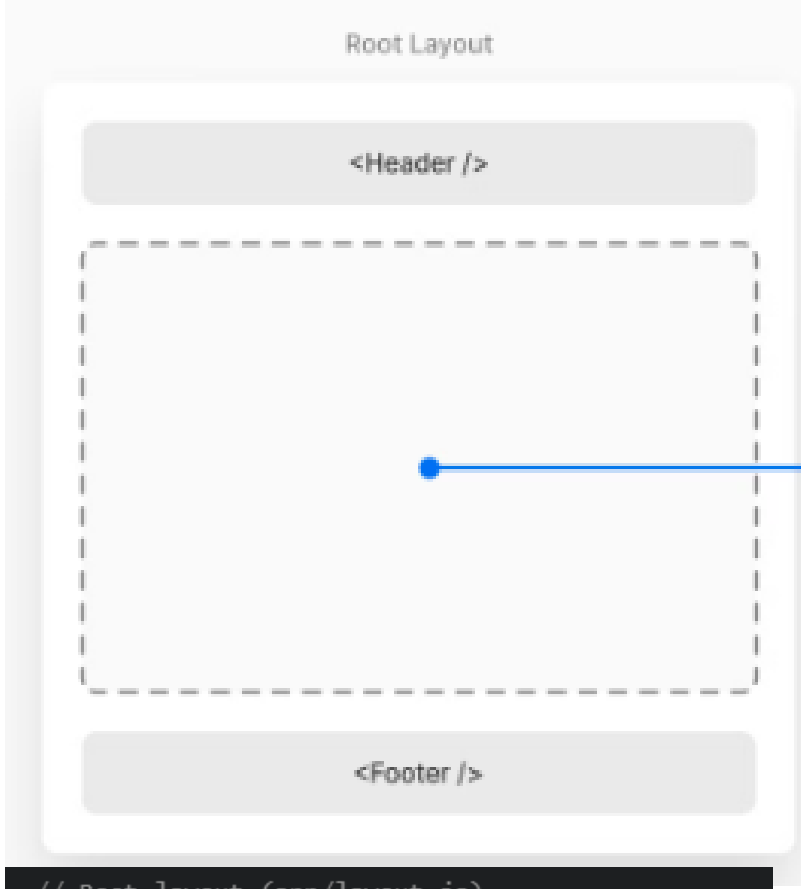


Nesting Layouts

- 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




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

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

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

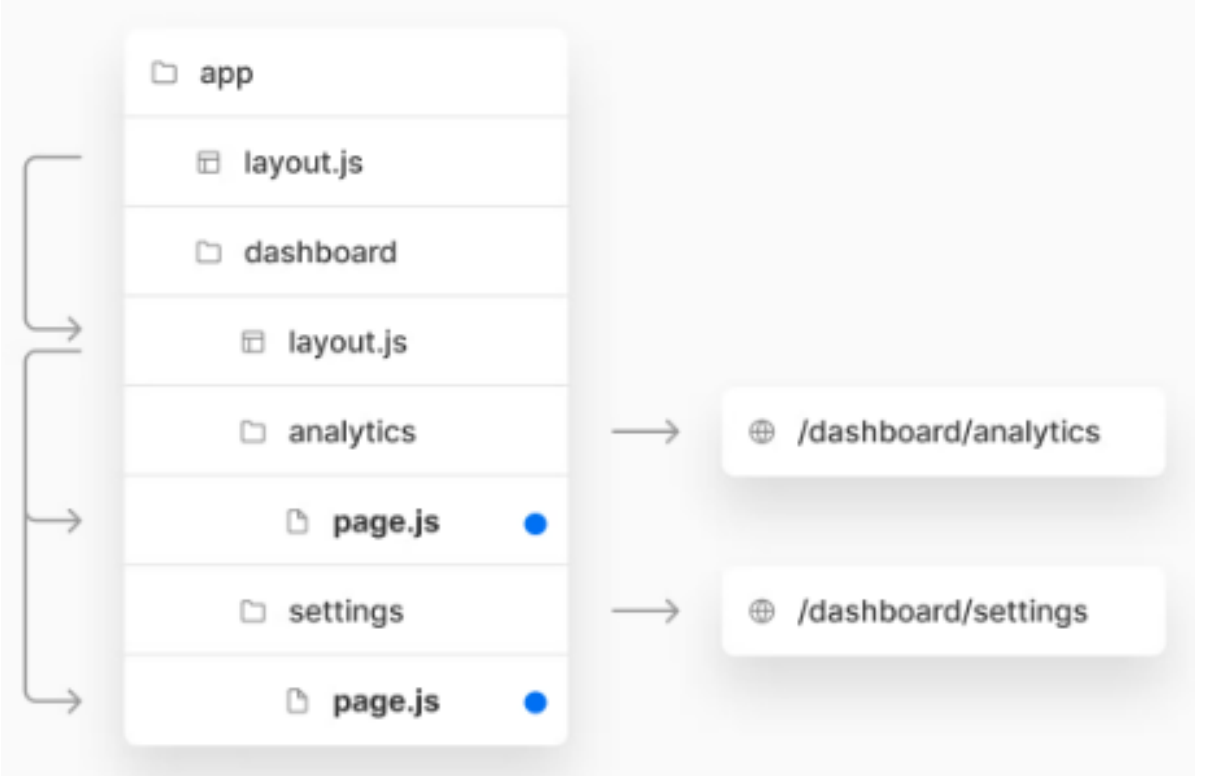
The above combination of layouts and pages would render the following component hierarchy:

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

UI Pages

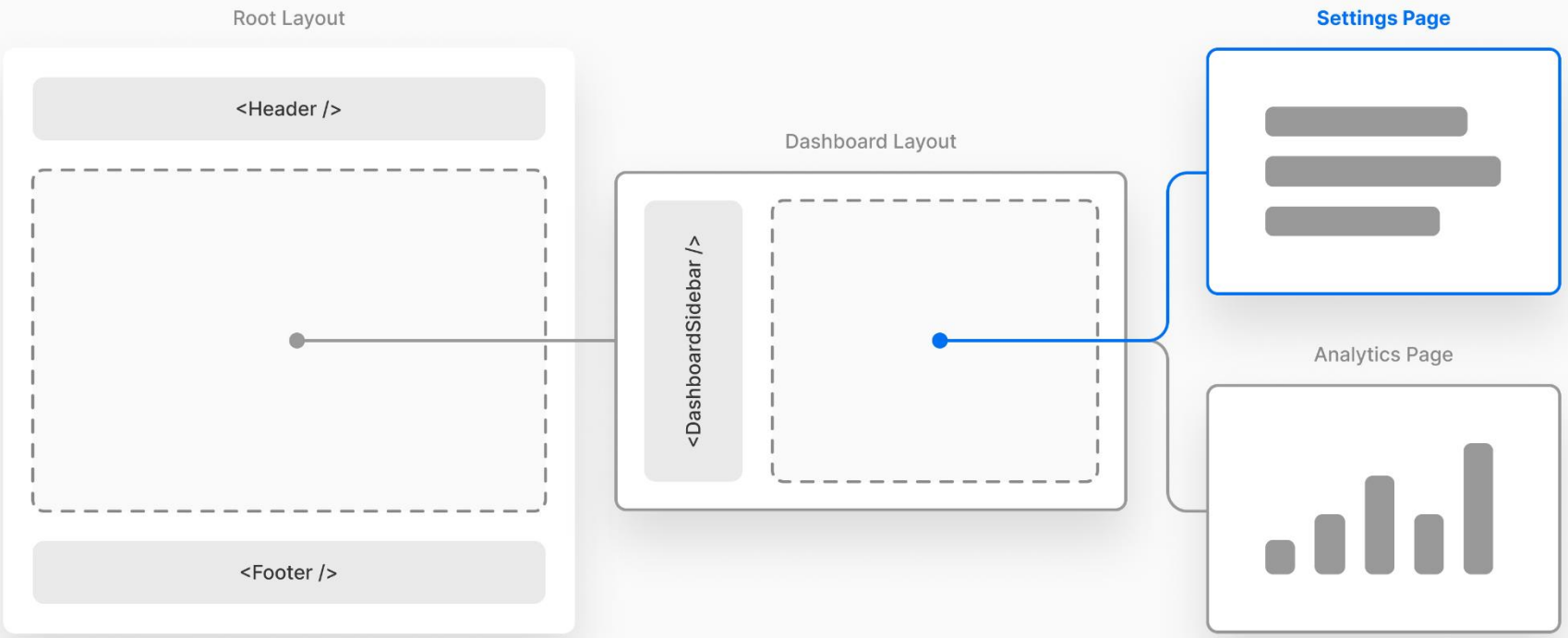
- You can create a page by adding a `page.js` file inside a folder
  - Can colocate your own project files (UI components, styles, test files, etc.) inside the app folder & subfolders

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



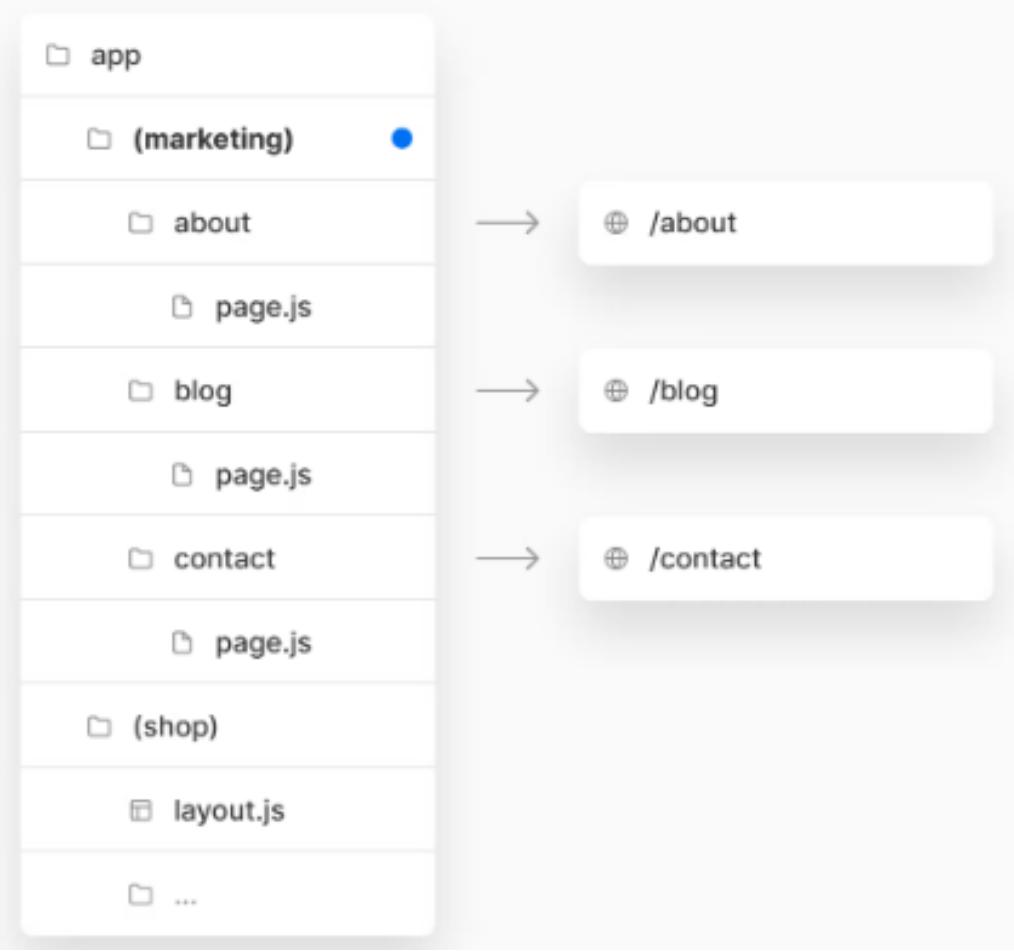
Pages are Wrapped in Layouts

- 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

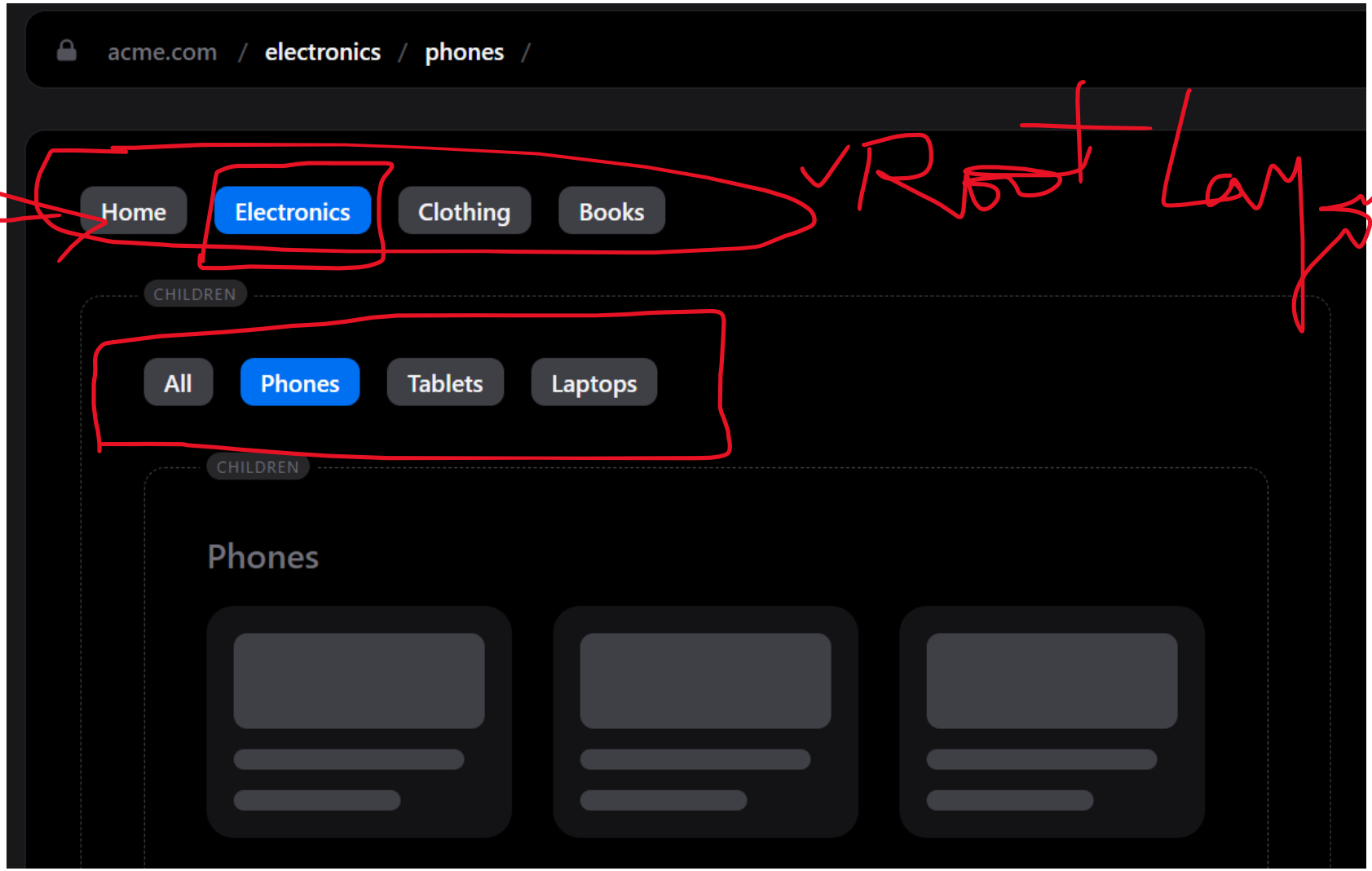


Organizing routes without affecting the URL path

- 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 will be rendered on the server as React Server Components
  - resulting in less client-side JavaScript and better performance

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:
  - page.js:
  - loading.js
  - error.js

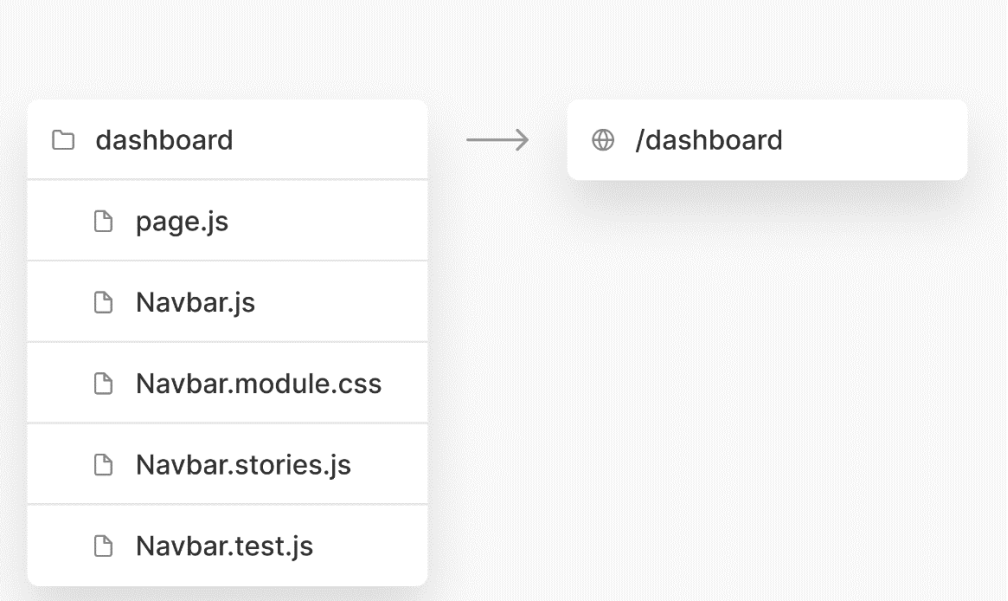
Instant Loading States

- With server-side routing, navigation happens after data fetching and rendering so it's important to show loading UI (defined in **`loading.js`**) while the data is being fetched otherwise the application will seem unresponsive

- Loading UI can be shown immediately while the content for the new segment loads
- The new content is then swapped in once rendering on the server is complete

Next 13 Layout

- **app/** directory makes it easier to create complex nested layouts
  - layouts that can be nested, shared across routes, and have their state preserved on navigation
  - avoid expensive re-renders, and enable advanced routing patterns
  - can colocate application code with the routes, like components, styles, and tests
- Folders inside **app/** are used to define routes
  - A route is a single path of nested folders from the root folder down to a final leaf folder



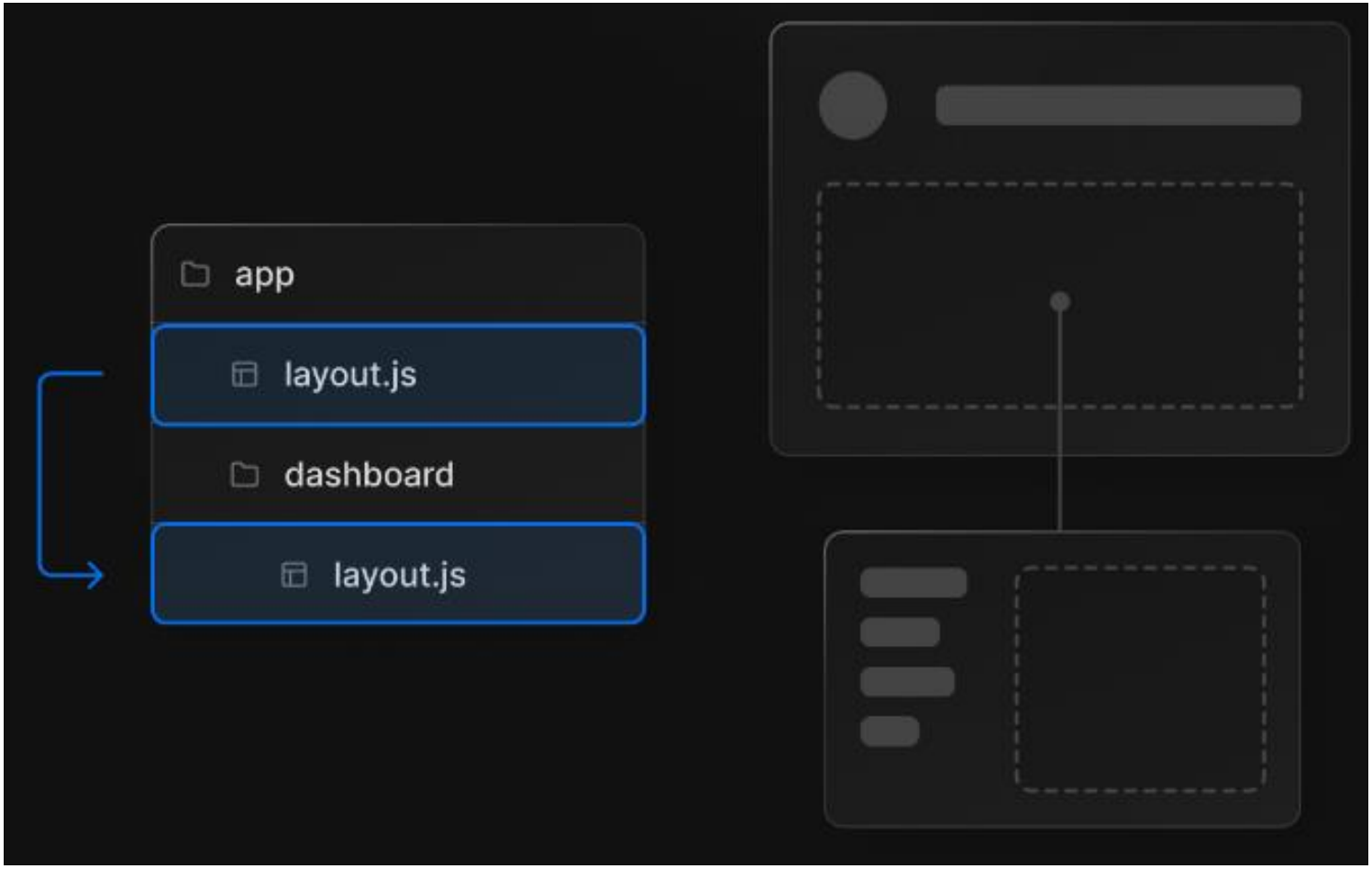
page.js

- Creating routes inside `app/` requires a single file, `page.js`

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

Nested Layouts

- The root layout (app/layout.js) would wrap the dashboard layout, which would wrap route segments inside dashboard/*



next/image

- Lazy loading and optimized files for increased performance

```
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" />;
}
```

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 otherwise it's excluded
<Link href="/about">
  <a>About</a>
</Link>

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

Data Fetching

Data Fetching prior to Next.js 13

- Next.js provides data fetching methods which can be used at the page (route) level
  - Statically Generated (getStaticProps)
  - Server-Side Rendered (getServerSideProps)
  - Incremental Static Regeneration (ISR) to create or update static pages after a site is built

Data Fetching Next.js 13

- 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

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

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

  return '...';
}
```

Static Data

- By default, fetch will automatically fetch static data (cached data)
- This is equivalent to getStaticProps() in the pages directory

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

```
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 Data Example

Dynamic Data

- To refetch data on every fetch() request, use the cache: 'no-store' option
- This is equivalent to getServerSideProps()

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

Revalidating Data

- To revalidate cached data, you can use the next.revalidate option in fetch()
- This equivalent to Incremental Static Regeneration (ISR)

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

Data fetching in Layouts

- You can fetch data in a layout.js
  - e.g., a blog layout could fetch categories which can be used to populate a sidebar component

Resources

- React Query

https://tanstack.com/query/v4/docs/overview

https://tanstack.com/query/v4/docs/videos

- Zustand

https://github.com/pmndrs/zustand

Layouts

- Layouts share UI between multiple pages. On navigation, layouts preserve state, remain interactive, and do not re-render.

```
// app/blog/layout.js
export default function BlogLayout({ children }) {
  return <section>{children}</section>;
}
```