Cheatsheets / Intro to Next.js



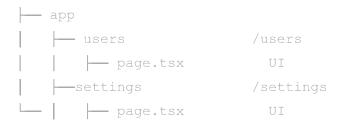
Next.js Routing

Next.js App Router

In Next.js, the App Router is used to define and structure an application's routes and their UIs. It is defined by creating a folder named app at the project's root level.

Next.js Folders and Reserved Files

In the Next.js App Router, folders define path segments, and reserved files within, like $\;page.tsx$, define that segment's UI.



Next.js Nested Routes

In Next.js, nested routes can be created by creating nested folders.





Composition of URL Paths

A URL path is composed of path segments and comes after the URL domain.

For example, in the URL

https://www.codecademy.com/catalog/language/jav ascript:

- codecademy.com is the domain.
- /catalog/language/javascript is the path.
- /catalog , /language , and /javascript are path segments.

Next.js Navigation

Next.js provides SPA-like navigation with the useRouter() hook and <Link> component. The useRouter() hook returns a router object which contains methods like push(path), back(), and forward() for navigation.

The <Link> component extends the <a> element by adding features like prefetching.



Next.js page.tsx

In Next.js, a page.tsx is a reserved file that defines a unique UI for its containing path segment and makes it accessible.

To use page.tsx , you must default export a React component.

Next.js layout.tsx and template.tsx

In Next.js, layout.tsx and template.tsx are reserved files used to create shared UIs.

Similarities include:

- Requiring a default export React component.
- Exported component receives a children prop of type ReactNode.
- Wraps other reserved files and nested routes.

Differences include:

- A template.tsx component will be reinstantiated while layout.tsx component will not.
- At least one root layout.tsx returning the https://doi.org/10.1001/j.com/ elements is required in the App Router.

```
// In root layout.tsx
export default function
RootLayout({children}: { children:
React.ReactNode}) {
  return (
    <html>
      <body>
       <h1>Root Layout</h1>
       {children}
      </body>
    </html>
// In template.tsx
export default function
Template({children}: {children:
React.ReactNode}) {
  return (
    <h2>Template Layout</h2>
    {children}
```



Next.js Dynamic Segments

In Next.js, a dynamic segment is created by wrapping a folder's name in square brackets, for example, /app/users/[userId] .

The page.tsx component exported from this folder will receive a params prop which will contain the dynamic segment data (as a string) and be referenced using the dynamic segment folder name (userId).

Next.js Catch-all Dynamic Segment

In Next.js, dynamic segments can be further modified to be made catch-all and optional. A catch-all segment is created by prefixing a dynamic segment with ellipses like /app/articles/[...articleIds] .

A catch-all segment's page.tsx component will receive a params prop containing the dynamic data as an array of string s referenced using the dynamic folder name (articleIds).

To make the catch-all dynamic segment optional, you wrap it in another pair of square brackets like: /app/articles/[[...articleIds]] .



Next.js Reserved Files

Next.js reserves special files used to define UIs by default exporting a React component. Some of the special files include:

- layout.tsx : Defines a shared UI.
- template.tsx : Defines a shared UI.
- error.tsx : Defines an ErrorBoundary fallback UI.
- loading.tsx : Defines a Suspense fallback UI.
- not-found.tsx : Defines an ErrorBoundary fallback UI for an unknown segment or nested segments.

Next.js Reserved File Component Hierarchy

Next.js defines a component hierarchy for reserve file components. The hierarchy is:

- 1. layout.tsx
- 2. template.tsx
- 3. error.tsx
- 4. loading.tsx
- 5. not-found.tsx
- 6. page.tsx

Any nested hierarchy will be placed within the hierarchy of its parent.



