App Directory: The App Router works in a new directory named app .It is a replacement for the pages directory. Pages in the app directory are server components by default, which means that they are rendered on the server and then sent to the client as static HTML. The app directory supports colocation, which means that you can put multiple components in the same file. This can make your code more organized and easier to maintain. app directory also supports component-level fetching and dynamic routing.

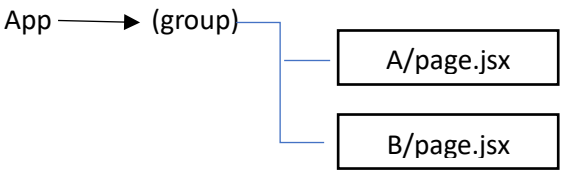
loading.tsx: loading.tsx is an optional file that we can create within any directory inside of the app folder. It automatically wraps the page inside of a react suspense boundary. The component will be shown immediately on the first load as well as when we're navigating between the sibling routes.

error.tsx : error.tsx is an optional file that isolates the error to the smallest possible subsection of the app. Creating the error.tsx file automatically wraps the page inside of a React error boundary. Whenever any error occurs inside the folder where this file is placed, the component will be replaced with the contents of this component.

layout.tsx : we can use the layout.tsx file to define a UI that is shared across multiple places. A layout can render another layout or a page inside of it. Whenever a route changes to any component that is within the layout, its state is preserved because the layout component is not unmounted.

Route groups

Every folder inside the app directory contributes to the URL path. But, it is possible to opt-out of it by wrapping the folder name inside of parentheses. All of the files and folders inside of this special folder are said to be a part of that route group:



Page.tsx : The Pages Router has a file-system based router built on concepts of pages. When a file is added to the `pages` directory it's automatically available as a route.
Src Directory: The src directory is an optional directory that can be used to store your application's source code. It is a good practice to use the src directory if you have a large or complex application, as it can help to keep your code organized and easier to maintain

The src directory can contain any of the following files and folders:

- Components: This directory can contain your application's React components.
- Pages: This directory can contain your application's pages.
- Utils: This directory can contain your application's utility functions and classes.
- Styles: This directory can contain your application's CSS styles.
- Other files and folders: You can also add any other files and folders to the src directory that your application needs.

not-found.js : The not-found.js file is used to handle 404 errors. When a user visits a URL that does not exist in your application, Next.js will look for a file called not-found.js in the current directory. If it finds the file, it will render the contents of the file as the 404 page.

Nested Routes

Nested routes allow to create a hierarchy of pages within an application. This can be useful for organizing y content and providing a more user-friendly experience for visitors.

```
/blog
  /posts
    /my-first-post
    /my-second-post
  /categories
    /tech
    /news
```

This would create a blog application with a root page for the blog, a subpage for posts, and two subpages for categories. When a user visits the `/blog` route, they would see a list of all the posts in the blog. If they click on the `/posts` route, they would see a list of all the posts in the blog, organized by category. And if they click on the `/categories` route, they would see a list of all the categories in the blog, with links to the posts in each category.

Dynamic Routes:

Dynamic routes in Next.js 13 allow you to create pages that are dynamically generated based on the URL parameters. This means that you can create pages that are unique to each user or each request.

`/ Dynamic / [slug]`

Globals.css: Global CSS is CSS that is applied to all pages in application. It is typically used for styles that need to be applied globally, such as font sizes, colors, and margins.

Layout.js explain:

`import './globals.css'` →This line imports the `globals.css` file, which contains global styles for the application.

`import { Inter } from 'next/font/google'` →This line imports the `Inter` font from the `next/font/google` package. This font will be used for the application's text.

`const inter = Inter({ subsets: ['latin'] })` →This line creates a new instance of the `Inter` font, specifying that the `latin` subset should be used. This will ensure that the font is properly displayed for all languages that use the Latin alphabet.

```
export const metadata = {
  title: 'Create Next App',
  description: 'Generated by create next app',
}
```

This line exports a constant called `metadata`. This constant contains the application's metadata, such as its title and description. This metadata will be used by Next.js 13 to generate the application's SEO meta tags.

```
export default function RootLayout({ children }) {
  return (
    <html lang="en">
```

```
      <body className={inter.className}>{children}</body>
    </html>
  )
}
```

This line exports a function called `RootLayout`. This function is the default layout for the application. It renders an `html` element with the `lang` attribute set to `en`. It also renders a `body` element with the `className` attribute set to the `className` property of the `inter` font. The `children` prop of the `RootLayout` function is passed to the `children` prop of the `body` element. This allows the `RootLayout` function to be used to render any content within the application.