React Suspense & Loading Examples

Next.js 13 and React Suspense: Create a loading component

**What is React Suspense?**

React’s Suspense component was first added to React in v16.6, which was released in 2018. Suspense handles asynchronous operations like code splitting and data fetching. In simple terms, it lets you display a fallback component until the child component is fully loaded. The code below shows React Suspense’s syntax:

<Suspense fallback={<Loading />}>

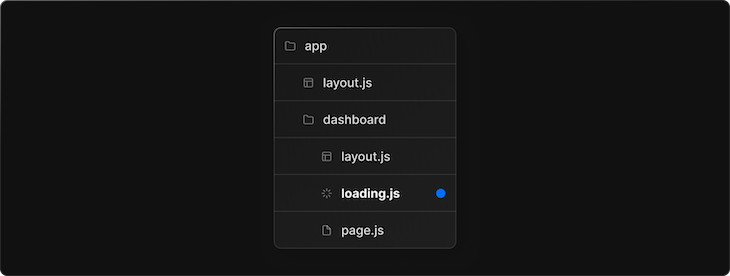
<SomeComponent />

</Suspense>

To load multiple components, you can add multiple child components within the <Suspense> component. Suspense improves a website’s performance and user experience, becoming an important part of the React ecosystem. Now, Next.js offers a new way to add Suspense to an application using its app directory, released as part of Next.js 13.

**Using Next.js and React Suspense**

The Next.js app directory has introduced a new file convention; you can now add all the files and components related to the route in a single directory. This includes both components and CSS files, so there is no need to create a separate directory for CSS files. In the route directory, you can include the loading.js file to add your loading UI for React Suspense’s fallback component:



Next.js supports server-side rendering, so the UI will take some time to load. In such cases, you can use Suspense to load the UI. The component in loading.js is defined as a functional component that can be exported as the default. The syntax is below:

export default function Loading() {

// You can add any UI inside Loading, including a Skeleton.

return <LoadingSkeleton />

}

**Setting up our Next.js 13 project**

We’ll build a web application that uses the TMDB API (<https://developer.themoviedb.org/reference/intro/getting-started> ) to fetch trending movies. We have two routes:

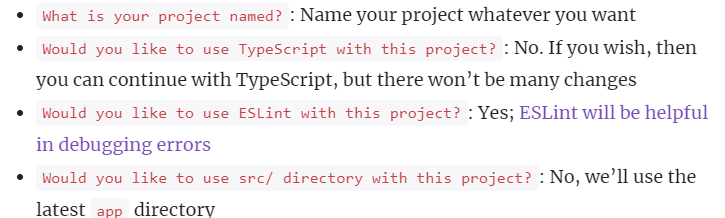
root(/): Displays a welcome screen in the application

movies(/movies): Displays the trending movies fetched from the API

**Installing Next.js 13**

You can install Next.js 13 with the command below. Keep in mind that you’ll need to have Node.js pre-installed on your machine:

npx create-next-app@latest --experimental-app



**Removing unnecessary files and folders**

There isn’t a lot of boilerplate code in Next.js, but we should clean it up anyways. Open the app directory and remove the API directory, which is for the server. Remove all the CSS code from the global.css file in the app directory. Now, open page.js and remove all the code within the return section. Then, enter the following code in page.js to display a basic page with a welcome message for the user:

async function Page() {

return (

<div>

<h3>List of trending Movies & TV</h3>

</div>

);

}

export default Page;

Now, let’s look at the layout section from the layout.js file:

import { Suspense } from "react";

import Link from "next/link";

import Loading from "./loading";

export const metadata = {

title: "Create Next App",

description: "Generated by create next app",

};

export default function RootLayout({ children }) {

return (

<html lang="en">

<body>

<h1>Trending Movies & Movies</h1>

<li>

<Link href="/">Home</Link>

</li>

<li>

<Link href="/movies">Movie</Link>

</li>

<Suspense fallback={<Loading />}>{children}</Suspense>

</body>

</html>

);

}

**Create a loading.js file**

In the root directory, create the loading.js file with the code below:

export default function Loading() {

return <p>Loading Data...</p>;

}

We’ve created a basic loading component for display, but you can add a much more sophisticated loading screen, like spinners or a skeleton loading screen. The functional component and file naming convention will remain the same.

**Creating the movies route**

In Next.js 13, creating a route is similar to its previous versions. Create a directory within the app directory named movies. Inside movies, create a file named page.js with the code below:

async function getMovies() {

let res = await fetch(

`https://api.themoviedb.org/3/trending/movie/day?api\_key=${process.env.NEXT\_PUBLIC\_TMDB\_API}`

);

await new Promise((resolve) => setTimeout(resolve, 2000));

return res.json();

}

async function Trending() {

let { results } = await getMovies();

return (

<div>

<h3>Movies</h3>

{results &&

results.map((index) => {

return <li>{index.title}</li>;

})}

</div>

);

}

export default Trending;

Above, we have a two-component file. getMovies() fetches data from the API, which is sent to the default functional component with the name Trending. You can see that this is a server-side rendering component. It has an async functional component for promise-based data fetching because Suspense will know that data fetching is happening. We’ve also implemented a delay of two seconds to see the loading component properly.

In the Trending component, we call the getMovies() function to get the fetched data. In the return, we are mapping the data to display all the trending movies in a list.

You might find it unusual that we haven’t used Suspense yet. Next.js 13 understands when there is something loading; if there is a loading.js file in the route or even the root directory, it will display its loading component when loading occurs.

We can add different loading components separately in every route with the addition of the loading.js file. Let’s check out the following gif displaying the output:

