Lazy Loading

Lazy loading in Next.js helps improve the initial loading performance of an application by decreasing the amount of JavaScript needed to render a route.

It allows you to defer loading of **Client Components** and imported libraries, and only include them in the client bundle when they're needed. For example, you might want to defer loading a modal until a user clicks to open it.

There are two ways you can implement lazy loading in Next.js:

1. Using Dynamic Imports with `next/dynamic`
2. Using `React.lazy()` with Suspense

By default, Server Components are automatically code split, and you can use streaming to progressively send pieces of UI from the server to the client. Lazy loading applies to Client Components.

`next/dynamic`

`next/dynamic` is a composite of `React.lazy()` and Suspense. It behaves the same way in the `app` and `pages` directories to allow for incremental migration.

Examples

By using `next/dynamic`, the header component will not be included in the page's initial JavaScript bundle. The page will render the Suspense `fallback` first, followed by the `Header` component when the `Suspense` boundary is resolved.

```
import dynamic from 'next/dynamic';

const DynamicHeader = dynamic(() => import('../components/header'), {
  loading: () => <p>Loading...</p>,
});

export default function Home() {
  return <DynamicHeader />;
}
```

Note: In `import('path/to/component')`, the path must be explicitly written. It can't be a template string nor a variable. Furthermore the `import()` has to be inside the `dynamic()` call for Next.js to be able to match webpack bundles / module ids to the specific `dynamic()` call and preload them before rendering. `dynamic()` can't be used inside of React rendering as it needs to be marked in the top level of the module for preloading to work, similar to `React.lazy`.

With named exports

To dynamically import a named export, you can return it from the Promise↗ returned by `import()`↗:

components/hello.js

```
export function Hello() {
  return <p>Hello!</p>;
}

// pages/index.js
import dynamic from 'next/dynamic';

const DynamicComponent = dynamic(() =>
  import('../components/hello').then((mod) => mod.Hello),
);
```

With no SSR

To dynamically load a component on the client side, you can use the `ssr` option to disable server-rendering. This is useful if an external dependency or component relies on browser APIs like `window`.

```
import dynamic from 'next/dynamic';
```

```
2
3  const DynamicHeader = dynamic(() => import('../components/header'), {
4    ssr: false,
5  });
```

With external libraries

This example uses the external library `fuse.js` for fuzzy search. The module is only loaded in the browser after the user types in the search input.

```
1  import { useState } from 'react';
2
3  const names = ['Tim', 'Joe', 'Bel', 'Lee'];
4
5  export default function Page() {
6    const [results, setResults] = useState();
7
8    return (
9      <div>
10       <input
11         type="text"
12         placeholder="Search"
13         onChange={async (e) => {
14           const { value } = e.currentTarget;
15           // Dynamically load fuse.js
16           const Fuse = (await import('fuse.js')).default;
17           const fuse = new Fuse(names);
18
19           setResults(fuse.search(value));
20         }}
21       />
22       <pre>Results: {JSON.stringify(results, null, 2)}</pre>
23     </div>
24   );
25 }
```