Dynamic Import

▼ **Examples**

- Dynamic Import

Next.js supports ES2020 dynamic `import()` for JavaScript. With it you can import JavaScript modules dynamically and work with them. They also work with SSR.

In the following example, we implement fuzzy search using `fuse.js` and only load the module dynamically in the browser after the user types in the search input:

```
import { useState } from 'react'

const names = ['Tim', 'Joe', 'Bel', 'Max', 'Lee']

export default function Page() {
  const [results, setResults] = useState()

  return (
    <div>
      <input
        type="text"
        placeholder="Search"
        onChange={async (e) => {
          const { value } = e.currentTarget
          // Dynamically load fuse.js
          const Fuse = (await import('fuse.js')).default
          const fuse = new Fuse(names)

          setResults(fuse.search(value))
        }}
      />
      <pre>Results: {JSON.stringify(results, null, 2)}</pre>
    </div>
  )
}
```

You can think of dynamic imports as another way to split your code into manageable chunks.

React components can also be imported using dynamic imports, but in this case we use it in conjunction with `next/dynamic` to make sure it works like any other React Component. Check out the sections below for more details on how it works.

Basic usage

In the following example, the module `../components/hello` will be dynamically loaded by the page:

```
import dynamic from 'next/dynamic'

const DynamicComponent = dynamic(() => import('../components/hello'))
```

```
function Home() {
  return (
    <div>
      <Header />
      <DynamicComponent />
      <p>HOME PAGE is here!</p>
    </div>
  )
}

export default Home
```

`DynamicComponent` will be the default component returned by `../components/hello`. It works like a regular React Component, and you can pass props to it as you normally would.

> ***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

If the dynamic component is not the default export, you can use a named export too. Consider the module `../components/hello.js` which has a named export `Hello`:

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

To dynamically import the `Hello` component, you can return it from the Promise returned by `import()`, like so:

```
import dynamic from 'next/dynamic'

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

function Home() {
  return (
    <div>
      <Header />
      <DynamicComponent />
      <p>HOME PAGE is here!</p>
    </div>
  )
}

export default Home
```

With custom loading component

An optional `loading` component can be added to render a loading state while the dynamic component is being loaded. For example:

```
import dynamic from 'next/dynamic'

const DynamicComponentWithCustomLoading = dynamic(
  () => import('../components/hello'),
  { loading: () => <p>...</p> }
)

function Home() {
  return (
    <div>
      <Header />
      <DynamicComponentWithCustomLoading />
      <p>HOME PAGE is here!</p>
    </div>
  )
}

export default Home
```

With no SSR

You may not always want to include a module on server-side. For example, when the module includes a library that only works in the browser.

Take a look at the following example:

```
import dynamic from 'next/dynamic'

const DynamicComponentWithNoSSR = dynamic(
  () => import('../components/hello3'),
  { ssr: false }
)

function Home() {
  return (
    <div>
      <Header />
      <DynamicComponentWithNoSSR />
      <p>HOME PAGE is here!</p>
    </div>
  )
}

export default Home
```

With suspense

Option `suspense` allows you to lazy-load a component, similar to `React.lazy` and `<Suspense>` with React 18. Note that it only works on client-side or server-side with `fallback`. Full SSR support in concurrent mode is still a work-in-progress.

```
import dynamic from 'next/dynamic'

const DynamicLazyComponent = dynamic(() => import('../components/hello4'), {
  suspense: true,
})

function Home() {
  return (
    <div>
      <Suspense fallback={`loading`}>
        <DynamicLazyComponent />
      </Suspense>
    </div>
  )
}
```