# Routing

Next.js has a file-system based router built on the concept of pages.

When a file is added to the `pages` directory, it's automatically available as a route.

The files inside the `pages` directory can be used to define most common patterns.

#### Index routes

The router will automatically route files named `index` to the root of the directory.

- `pages/index.js` → `/`
- `pages/blog/index.js` → `/blog`

#### Nested routes

The router supports nested files. If you create a nested folder structure, files will automatically be routed in the same way still.

- `pages/blog/first-post.js` → `/blog/first-post`
- `pages/dashboard/settings/username.js` → `/dashboard/settings/username`

#### Dynamic route segments

To match a dynamic segment, you can use the bracket syntax. This allows you to match named parameters.

- `pages/blog/[slug].js` → `/blog/:slug` ( `/blog/hello-world` )
- `pages/[username]/settings.js` → `/:username/settings` ( `/foo/settings` )
- `pages/post/[...all].js` → `/post/*` ( `/post/2020/id/title` )

> *Check out the Dynamic Routes documentation to learn more about how they work.*

## Linking between pages

The Next.js router allows you to do client-side route transitions between pages, similar to a single-page application.

A React component called `Link` is provided to do this client-side route transition.

```
import Link from 'next/link'

function Home() {
  return (
    <ul>
      <li>
        <Link href="/">
          <a>Home</a>
        </Link>
      </li>
      <li>
        <Link href="/about">
          <a>About Us</a>
        </Link>
      </li>
      <li>
```

```
        <Link href="/blog/hello-world">
          <a>Blog Post</a>
        </Link>
      </li>
    </ul>
  )
}

export default Home
```

The example above uses multiple links. Each one maps a path ( `href` ) to a known page:

- `/` → `pages/index.js`
- `/about` → `pages/about.js`
- `/blog/hello-world` → `pages/blog/[slug].js`

Any `<Link />` in the viewport (initially or through scroll) will be prefetched by default (including the corresponding data) for pages using Static Generation. The corresponding data for server-rendered routes is *not* prefetched.

### Linking to dynamic paths

You can also use interpolation to create the path, which comes in handy for dynamic route segments. For example, to show a list of posts which have been passed to the component as a prop:

```
import Link from 'next/link'

function Posts({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>
          <Link href={`/blog/${encodeURIComponent(post.slug)}`}>
            <a>{post.title}</a>
          </Link>
        </li>
      ))}
    </ul>
  )
}

export default Posts
```

> *`encodeURIComponent` is used in the example to keep the path utf-8 compatible.*

Alternatively, using a URL Object:

```
import Link from 'next/link'

function Posts({ posts }) {
  return (
    <ul>
```

```
      {posts.map((post) => (
        <li key={post.id}>
          <Link
            href={{
              pathname: '/blog/[slug]',
              query: { slug: post.slug },
            }}
          >
            <a>{post.title}</a>
          </Link>
        </li>
      ))}
    </ul>
  )
}

export default Posts
```

Now, instead of using interpolation to create the path, we use a URL object in `href` where:

- `pathname` is the name of the page in the `pages` directory. `/blog/[slug]` in this case.
- `query` is an object with the dynamic segment. `slug` in this case.

## Injecting the router

▶ **Examples**

To access the `router` object in a React component you can use `useRouter` or `withRouter`.

In general we recommend using `useRouter`.

## Learn more

The router is divided in multiple parts:

**next/link:** Handle client-side navigations.
**next/router:** Leverage the router API in your pages.