Internationalized Routing

▶ **Examples**

Next.js has built-in support for internationalized (i18n) routing since `v10.0.0`. You can provide a list of locales, the default locale, and domain-specific locales and Next.js will automatically handle the routing.

The i18n routing support is currently meant to complement existing i18n library solutions like `react-intl`, `react-i18next`, `lingui`, `rosetta`, `next-intl` and others by streamlining the routes and locale parsing.

Getting started

To get started, add the `i18n` config to your `next.config.js` file.

Locales are UTS Locale Identifiers, a standardized format for defining locales.

Generally a Locale Identifier is made up of a language, region, and script separated by a dash: `language-region-script`. The region and script are optional. An example:

- `en-US` - English as spoken in the United States
- `nl-NL` - Dutch as spoken in the Netherlands
- `nl` - Dutch, no specific region

```
// next.config.js
module.exports = {
  i18n: {
    // These are all the locales you want to support in
    // your application
    locales: ['en-US', 'fr', 'nl-NL'],
    // This is the default locale you want to be used when visiting
    // a non-locale prefixed path e.g. `/hello`
    defaultLocale: 'en-US',
    // This is a list of locale domains and the default locale they
    // should handle (these are only required when setting up domain routing)
    // Note: subdomains must be included in the domain value to be matched e.g.
"fr.example.com".
    domains: [
      {
        domain: 'example.com',
        defaultLocale: 'en-US',
      },
      {
        domain: 'example.nl',
        defaultLocale: 'nl-NL',
      },
      {
        domain: 'example.fr',
        defaultLocale: 'fr',
        // an optional http field can also be used to test
        // locale domains locally with http instead of https
        http: true,
      },
```

```
    ],
  },
}
```

Locale Strategies

There are two locale handling strategies: Sub-path Routing and Domain Routing.

Sub-path Routing

Sub-path Routing puts the locale in the url path.

```
// next.config.js
module.exports = {
  i18n: {
    locales: ['en-US', 'fr', 'nl-NL'],
    defaultLocale: 'en-US',
  },
}
```

With the above configuration `en-US`, `fr`, and `nl-NL` will be available to be routed to, and `en-US` is the default locale. If you have a `pages/blog.js` the following urls would be available:

- `/blog`
- `/fr/blog`
- `/nl-nl/blog`

The default locale does not have a prefix.

Domain Routing

By using domain routing you can configure locales to be served from different domains:

```
// next.config.js
module.exports = {
  i18n: {
    locales: ['en-US', 'fr', 'nl-NL', 'nl-BE'],
    defaultLocale: 'en-US',

    domains: [
      {
        // Note: subdomains must be included in the domain value to be matched
        // e.g. www.example.com should be used if that is the expected hostname
        domain: 'example.com',
        defaultLocale: 'en-US',
      },
      {
        domain: 'example.fr',
        defaultLocale: 'fr',
      },
      {
```

```
        domain: 'example.nl',
        defaultLocale: 'nl-NL',
        // specify other locales that should be redirected
        // to this domain
        locales: ['nl-BE'],
      },
    ],
  },
}
```

For example if you have `pages/blog.js` the following urls will be available:

- `example.com/blog`
- `www.example.com/blog`
- `example.fr/blog`
- `example.nl/blog`
- `example.nl/nl-BE/blog`

Automatic Locale Detection

When a user visits the application root (generally `/`), Next.js will try to automatically detect which locale the user prefers based on the `Accept-Language` header and the current domain.

If a locale other than the default locale is detected, the user will be redirected to either:

- **When using Sub-path Routing:** The locale prefixed path
- **When using Domain Routing:** The domain with that locale specified as the default

When using Domain Routing, if a user with the `Accept-Language` header `fr;q=0.9` visits `example.com`, they will be redirected to `example.fr` since that domain handles the `fr` locale by default.

When using Sub-path Routing, the user would be redirected to `/fr`.

Prefixing the Default Locale

With Next.js 12 and Middleware, we can add a prefix to the default locale with a workaround.

For example, here's a `next.config.js` file with support for a few languages. Note the `"default"` locale has been added intentionally.

```
// next.config.js

module.exports = {
  i18n: {
    locales: ['default', 'en', 'de', 'fr'],
    defaultLocale: 'default',
    localeDetection: false,
  },
  trailingSlash: true,
}
```

Next, we can use Middleware to add custom routing rules:

```
// pages/_middleware.ts

import { NextRequest, NextResponse } from 'next/server'

const PUBLIC_FILE = /\.(.*)$/

export function middleware(request: NextRequest) {
  const shouldHandleLocale =
    !PUBLIC_FILE.test(request.nextUrl.pathname) &&
    !request.nextUrl.pathname.includes('/api/') &&
    request.nextUrl.locale === 'default'

  if (shouldHandleLocale) {
    const url = request.nextUrl.clone()
    url.pathname = `/en${request.nextUrl.pathname}`
    return NextResponse.redirect(url)
  }

  return undefined
}
```

This Middleware skips adding the default prefix to API Routes and public files like fonts or images. If a request is made to the default locale, we redirect to our prefix `/en`.

Disabling Automatic Locale Detection

The automatic locale detection can be disabled with:

```
// next.config.js
module.exports = {
  i18n: {
    localeDetection: false,
  },
}
```

When `localeDetection` is set to `false` Next.js will no longer automatically redirect based on the user's preferred locale and will only provide locale information detected from either the locale based domain or locale path as described above.

Accessing the locale information

You can access the locale information via the Next.js router. For example, using the `useRouter()` hook the following properties are available:

- `locale` contains the currently active locale.
- `locales` contains all configured locales.
- `defaultLocale` contains the configured default locale.

When pre-rendering pages with `getStaticProps` or `getServerSideProps`, the locale information is provided in the context provided to the function.

When leveraging `getStaticPaths`, the configured locales are provided in the context parameter of the function under `locales` and the configured defaultLocale under `defaultLocale`.

Transition between locales

You can use `next/link` or `next/router` to transition between locales.

For `next/link`, a `locale` prop can be provided to transition to a different locale from the currently active one. If no `locale` prop is provided, the currently active `locale` is used during client-transitions. For example:

```
import Link from 'next/link'

export default function IndexPage(props) {
  return (
    <Link href="/another" locale="fr">
      <a>To /fr/another</a>
    </Link>
  )
}
```

When using the `next/router` methods directly, you can specify the `locale` that should be used via the transition options. For example:

```
import { useRouter } from 'next/router'

export default function IndexPage(props) {
  const router = useRouter()

  return (
    <div
      onClick={() => {
        router.push('/another', '/another', { locale: 'fr' })
      }}
    >
      to /fr/another
    </div>
  )
}
```

Note that to handle switching only the `locale` while preserving all routing information such as dynamic route query values or hidden href query values, you can provide the `href` parameter as an object:

```
import { useRouter } from 'next/router'
const router = useRouter()
const { pathname, asPath, query } = router
// change just the locale and maintain all other route information including href's
query
router.push({ pathname, query }, asPath, { locale: nextLocale })
```

See here for more information on the object structure for `router.push`.

If you have a `href` that already includes the locale you can opt-out of automatically handling the locale prefixing:

```
import Link from 'next/link'

export default function IndexPage(props) {
  return (
    <Link href="/fr/another" locale={false}>
      <a>To /fr/another</a>
    </Link>
  )
}
```

Leveraging the NEXT_LOCALE cookie

Next.js supports overriding the accept-language header with a `NEXT_LOCALE=the-locale` cookie. This cookie can be set using a language switcher and then when a user comes back to the site it will leverage the locale specified in the cookie when redirecting from `/` to the correct locale location.

For example, if a user prefers the locale `fr` in their accept-language header but a `NEXT_LOCALE=en` cookie is set the `en` locale when visiting `/` the user will be redirected to the `en` locale location until the cookie is removed or expired.

Search Engine Optimization

Since Next.js knows what language the user is visiting it will automatically add the `lang` attribute to the `<html>` tag.

Next.js doesn't know about variants of a page so it's up to you to add the `hreflang` meta tags using `next/head`. You can learn more about `hreflang` in the Google Webmasters documentation.

How does this work with Static Generation?

> *Note that Internationalized Routing does not integrate with `next export` as `next export` does not leverage the Next.js routing layer. Hybrid Next.js applications that do not use `next export` are fully supported.*

Dynamic Routes and `getStaticProps` Pages

For pages using `getStaticProps` with Dynamic Routes, all locale variants of the page desired to be prerendered need to be returned from `getStaticPaths`. Along with the `params` object returned for `paths`, you can also return a `locale` field specifying which locale you want to render. For example:

```
// pages/blog/[slug].js
export const getStaticPaths = ({ locales }) => {
  return {
    paths: [
      // if no `locale` is provided only the defaultLocale will be generated
      { params: { slug: 'post-1' }, locale: 'en-US' },
      { params: { slug: 'post-1' }, locale: 'fr' },
```

```
    ],
    fallback: true,
  }
}
```

For Automatically Statically Optimized and non-dynamic `getStaticProps` pages, **a version of the page will be generated for each locale**. This is important to consider because it can increase build times depending on how many locales are configured inside `getStaticProps`.

For example, if you have 50 locales configured with 10 non-dynamic pages using `getStaticProps`, this means `getStaticProps` will be called 500 times. 50 versions of the 10 pages will be generated during each build.

To decrease the build time of dynamic pages with `getStaticProps`, use a `fallback` mode. This allows you to return only the most popular paths and locales from `getStaticPaths` for prerendering during the build. Then, Next.js will build the remaining pages at runtime as they are requested.

Automatically Statically Optimized Pages

For pages that are automatically statically optimized, a version of the page will be generated for each locale.

Non-dynamic getStaticProps Pages

For non-dynamic `getStaticProps` pages, a version is generated for each locale like above. `getStaticProps` is called with each `locale` that is being rendered. If you would like to opt-out of a certain locale from being pre-rendered, you can return `notFound: true` from `getStaticProps` and this variant of the page will not be generated.

```
export async function getStaticProps({ locale }) {
  // Call an external API endpoint to get posts.
  // You can use any data fetching library
  const res = await fetch(`https://.../posts?locale=${locale}`)
  const posts = await res.json()

  if (posts.length === 0) {
    return {
      notFound: true,
    }
  }

  // By returning { props: posts }, the Blog component
  // will receive `posts` as a prop at build time
  return {
    props: {
      posts,
    },
  }
}
```

Limits for the i18n config

- `locales`: 100 total locales

- `domains`: 100 total locale domain items

> ***Note:*** *These limits have been added initially to prevent potential performance issues at build time. You can workaround these limits with custom routing using Middleware in Next.js 12.*