Headers

Examples

Headers

Version History

Version	Changes
`v10.2.0`	`has` added.
`v9.5.0`	Headers added.

Headers allow you to set custom HTTP headers on the response to an incoming request on a given path.

To set custom HTTP headers you can use the `headers` key in `next.config.js`:

```
module.exports = {
  async headers() {
    return [
      {
        source: '/about',
        headers: [
          {
            key: 'x-custom-header',
            value: 'my custom header value',
          },
          {
            key: 'x-another-custom-header',
            value: 'my other custom header value',
          },
        ],
      },
    ]
  },
}
```

`headers` is an async function that expects an array to be returned holding objects with `source` and `headers` properties:

- `source` is the incoming request path pattern.
- `headers` is an array of response header objects, with `key` and `value` properties.
- `basePath`: `false` or `undefined` - if false the basePath won't be included when matching, can be used for external rewrites only.
- `locale`: `false` or `undefined` - whether the locale should not be included when matching.

- `has` is an array of has objects with the `type`, `key` and `value` properties.

Headers are checked before the filesystem which includes pages and `/public` files.

Header Overriding Behavior

If two headers match the same path and set the same header key, the last header key will override the first. Using the below headers, the path `/hello` will result in the header `x-hello` being `world` due to the last header value set being `world`.

```
module.exports = {
  async headers() {
    return [
      {
        source: '/:path*',
        headers: [
          {
            key: 'x-hello',
            value: 'there',
          },
        ],
      },
      {
        source: '/hello',
        headers: [
          {
            key: 'x-hello',
            value: 'world',
          },
        ],
      },
    ]
  },
}
```

Path Matching

Path matches are allowed, for example `/blog/:slug` will match `/blog/hello-world` (no nested paths):

```
module.exports = {
  async headers() {
    return [
      {
        source: '/blog/:slug',
        headers: [
```

```
          {
            key: 'x-slug',
            value: ':slug', // Matched parameters can be used in the value
          },
          {
            key: 'x-slug-:slug', // Matched parameters can be used in the key
            value: 'my other custom header value',
          },
        ],
      },
    ]
  },
}
```

Wildcard Path Matching

To match a wildcard path you can use `*` after a parameter, for example `/blog/:slug*` will match `/blog/a/b/c/d/hello-world`:

```
module.exports = {
  async headers() {
    return [
      {
        source: '/blog/:slug*',
        headers: [
          {
            key: 'x-slug',
            value: ':slug*', // Matched parameters can be used in the value
          },
          {
            key: 'x-slug-:slug*', // Matched parameters can be used in the key
            value: 'my other custom header value',
          },
        ],
      },
    ]
  },
}
```

Regex Path Matching

To match a regex path you can wrap the regex in parenthesis after a parameter, for example `/blog/:slug(\\d{1,})` will match `/blog/123` but not `/blog/abc`:

```
module.exports = {
  async headers() {
    return [
```

```
      {
        source: '/blog/:post(\\d{1,})',
        headers: [
          {
            key: 'x-post',
            value: ':post',
          },
        ],
      },
    ]
  },
}
```

The following characters `(`, `)`, `{`, `}`, `:`, `*`, `+`, `?` are used for regex path matching, so when used in the `source` as non-special values they must be escaped by adding `\\` before them:

```
module.exports = {
  async headers() {
    return [
      {
        // this will match `/english(default)/something` being requested
        source: '/english\\(default\\)/:slug',
        headers: [
          {
            key: 'x-header',
            value: 'value',
          },
        ],
      },
    ]
  },
}
```

Header, Cookie, and Query Matching

To only apply a header when either header, cookie, or query values also match the `has` field can be used. Both the `source` and all `has` items must match for the header to be applied.

`has` items have the following fields:

- `type`: `String` - must be either `header`, `cookie`, `host`, or `query`.
- `key`: `String` - the key from the selected type to match against.
- `value`: `String` or `undefined` - the value to check for, if undefined any value will match. A regex like string can be used to capture a specific part of the value, e.g. if the value `first-(?<paramName>.*)` is used for `first-second` then `second` will be usable in the destination with `:paramName`.

```
module.exports = {
  async headers() {
    return [
      // if the header `x-add-header` is present,
      // the `x-another-header` header will be applied
      {
        source: '/:path*',
        has: [
          {
            type: 'header',
            key: 'x-add-header',
          },
        ],
        headers: [
          {
            key: 'x-another-header',
            value: 'hello',
          },
        ],
      },
      // if the source, query, and cookie are matched,
      // the `x-authorized` header will be applied
      {
        source: '/specific/:path*',
        has: [
          {
            type: 'query',
            key: 'page',
            // the page value will not be available in the
            // header key/values since value is provided and
            // doesn't use a named capture group e.g. (?<page>home)
            value: 'home',
          },
          {
            type: 'cookie',
            key: 'authorized',
            value: 'true',
          },
        ],
        headers: [
          {
            key: 'x-authorized',
            value: ':authorized',
          },
        ],
      },
```

```
      // if the header `x-authorized` is present and
      // contains a matching value, the `x-another-header` will be applied
      {
        source: '/:path*',
        has: [
          {
            type: 'header',
            key: 'x-authorized',
            value: '(?<authorized>yes|true)',
          },
        ],
        headers: [
          {
            key: 'x-another-header',
            value: ':authorized',
          },
        ],
      },
      // if the host is `example.com`,
      // this header will be applied
      {
        source: '/:path*',
        has: [
          {
            type: 'host',
            value: 'example.com',
          },
        ],
        headers: [
          {
            key: 'x-another-header',
            value: ':authorized',
          },
        ],
      },
    ]
  },
}
```

Headers with basePath support

When leveraging `basePath` support with headers each `source` is automatically prefixed with the `basePath` unless you add `basePath: false` to the header:

```
module.exports = {
  basePath: '/docs',
```

```
  async headers() {
    return [
      {
        source: '/with-basePath', // becomes /docs/with-basePath
        headers: [
          {
            key: 'x-hello',
            value: 'world',
          },
        ],
      },
      {
        source: '/without-basePath', // is not modified since basePath: false is set
        headers: [
          {
            key: 'x-hello',
            value: 'world',
          },
        ],
        basePath: false,
      },
    ]
  },
}
```

Headers with i18n support

When leveraging `i18n` support with headers each `source` is automatically prefixed to handle the configured `locales` unless you add `locale: false` to the header. If `locale: false` is used you must prefix the `source` with a locale for it to be matched correctly.

```
module.exports = {
  i18n: {
    locales: ['en', 'fr', 'de'],
    defaultLocale: 'en',
  },

  async headers() {
    return [
      {
        source: '/with-locale', // automatically handles all locales
        headers: [
          {
            key: 'x-hello',
```

```
            value: 'world',
          },
        ],
      },
      {
        // does not handle locales automatically since locale: false is set
        source: '/nl/with-locale-manual',
        locale: false,
        headers: [
          {
            key: 'x-hello',
            value: 'world',
          },
        ],
      },
      {
        // this matches '/' since `en` is the defaultLocale
        source: '/en',
        locale: false,
        headers: [
          {
            key: 'x-hello',
            value: 'world',
          },
        ],
      },
      {
        // this gets converted to /(en|fr|de)/(.*) so will not match the top-level
        // `/` or `/fr` routes like /:path* would
        source: '/(.*)',
        headers: [
          {
            key: 'x-hello',
            value: 'world',
          },
        ],
      },
    ]
  },
}
```

Cache-Control

You can set the `Cache-Control` header in your Next.js API Routes by using the `res.setHeader` method:

```
// pages/api/user.js

export default function handler(req, res) {
  res.setHeader('Cache-Control', 's-maxage=86400')
  res.status(200).json({ name: 'John Doe' })
}
```

You cannot set `Cache-Control` headers in `next.config.js` file as these will be overwritten in production to ensure that API Routes and static assets are cached effectively.

If you need to revalidate the cache of a page that has been statically generated, you can do so by setting the `revalidate` prop in the page's `getStaticProps` function.

Related

For more information, we recommend the following sections:

Security Headers: Improve the security of your Next.js application by add HTTP response headers.