Redirecting

There are a few ways you can handle redirects in Next.js. This page will go through each available option, use cases, and how to manage large numbers of redirects.

API	Purpose	Where	Status Code
`redirect`	Redirect user after a mutation or event	Server Components, Server Actions, Route Handlers	307 (Temporary) or 303 (Server Action)
`permanentRedirect`	Redirect user after a mutation or event	Server Components, Server Actions, Route Handlers	308 (Permanent)
`useRouter`	Perform a client-side navigation	Event Handlers in Client Components	N/A
`redirects` in `next.config.js`	Redirect an incoming request based on a path	`next.config.js` file	307 (Temporary) or 308 (Permanent)
`NextResponse.redirect`	Redirect an incoming request based on a condition	Middleware	Any

`redirect` function

The `redirect` function allows you to redirect the user to another URL. You can call `redirect` in Server Components, Route Handlers, and Server Actions.

`redirect` is often used after a mutation or event. For example, creating a post:

app/actions.tsx — TypeScript

```tsx
'use server'

import { redirect } from 'next/navigation'
import { revalidatePath } from 'next/cache'

export async function createPost(id: string) {
  try {
    // Call database
  } catch (error) {
    // Handle errors
  }

  revalidatePath('/posts') // Update cached posts
  redirect(`/post/${id}`) // Navigate to the new post page
}
```

Good to know:

- `redirect` returns a 307 (Temporary Redirect) status code by default. When used in a Server Action, it returns a 303 (See Other), which is commonly used for redirecting to a success page as a result of a POST request.
- `redirect` internally throws an error so it should be called outside of `try/catch` blocks.
- `redirect` can be called in Client Components during the rendering process but not in event handlers. You can use the `useRouter` hook instead.
- `redirect` also accepts absolute URLs and can be used to redirect to external links.
- If you'd like to redirect before the render process, use `next.config.js` or Middleware.

See the `redirect` API reference for more information.

`permanentRedirect` function

The `permanentRedirect` function allows you to **permanently** redirect the user to another URL. You can call `permanentRedirect` in Server Components, Route Handlers, and Server Actions.

`permanentRedirect` is often used after a mutation or event that changes an entity's canonical URL, such as updating a user's profile URL after they change their username:

app/actions.ts — TypeScript

```ts
'use server'

import { permanentRedirect } from 'next/navigation'
import { revalidateTag } from 'next/cache'

export async function updateUsername(username: string, formData: FormData) {
  try {
    // Call database
  } catch (error) {
    // Handle errors
  }

  revalidateTag('username') // Update all references to the username
  permanentRedirect(`/profile/${username}`) // Navigate to the new user profile
}
```

Good to know:

- `permanentRedirect` returns a 308 (permanent redirect) status code by default.
- `permanentRedirect` also accepts absolute URLs and can be used to redirect to external links.
- If you'd like to redirect before the render process, use `next.config.js` or Middleware.

See the `permanentRedirect` API reference for more information.

`useRouter()` hook

If you need to redirect inside an event handler in a Client Component, you can use the `push` method from the `useRouter` hook. For example:

app/page.tsx TypeScript

```tsx
'use client'

import { useRouter } from 'next/navigation'

export default function Page() {
  const router = useRouter()

  return (
    <button type="button" onClick={() => router.push('/dashboard')}>
      Dashboard
    </button>
  )
}
```

Good to know:

See the `useRouter` API reference for more information.

`redirects` in `next.config.js`

The `redirects` option in the `next.config.js` file allows you to redirect an incoming request path to a different destination path. This is useful when you change the URL structure of pages or have a list of redirects that are known ahead of time.

`redirects` supports path, header, cookie, and query matching, giving you the flexibility to redirect users based on an incoming request.

To use `redirects`, add the option to your `next.config.js` file:

next.config.js

```js
module.exports = {
```

```
  async redirects() {
    return [
      // Basic redirect
      {
        source: '/about',
        destination: '/',
        permanent: true,
      },
      // Wildcard path matching
      {
        source: '/blog/:slug',
        destination: '/news/:slug',
        permanent: true,
      },
    ]
  },
}
```

See the `redirects` API reference for more information.

Good to know:

- `redirects` can return a 307 (Temporary Redirect) or 308 (Permanent Redirect) status code with the `permanent` option.
- `redirects` may have a limit on platforms. For example, on Vercel, there's a limit of 1,024 redirects. To manage a large number of redirects (1000+), consider creating a custom solution using Middleware. See managing redirects at scale for more.
- `redirects` runs **before** Middleware.

`NextResponse.redirect` in Middleware

Middleware allows you to run code before a request is completed. Then, based on the incoming request, redirect to a different URL using `NextResponse.redirect`. This is useful if you want to redirect users based on a condition (e.g. authentication, session management, etc) or have a large number of redirects.

For example, to redirect the user to a `/login` page if they are not authenticated:

middleware.ts

TypeScript

```typescript
import { NextResponse, NextRequest } from 'next/server'
import { authenticate } from 'auth-provider'

export function middleware(request: NextRequest) {
  const isAuthenticated = authenticate(request)

  // If the user is authenticated, continue as normal
  if (isAuthenticated) {
    return NextResponse.next()
  }

  // Redirect to login page if not authenticated
  return NextResponse.redirect(new URL('/login', request.url))
}

export const config = {
  matcher: '/dashboard/:path*',
}
```

Good to know:

- Middleware runs **after** `redirects` in `next.config.js` and **before** rendering.

See the Middleware documentation for more information.

Managing redirects at scale (advanced)

To manage a large number of redirects (1000+), you may consider creating a custom solution using Middleware. This allows you to handle redirects programmatically without having to redeploy your application.

To do this, you'll need to consider:

1. Creating and storing a redirect map.
2. Optimizing data lookup performance.

Next.js Example: See our Middleware with Bloom filter ↗ example for an implementation of the recommendations below.

1. Creating and storing a redirect map

A redirect map is a list of redirects that you can store in a database (usually a key-value store) or JSON file.

Consider the following data structure:

```
{
  "/old": {
    "destination": "/new",
    "permanent": true
  },
  "/blog/post-old": {
    "destination": "/blog/post-new",
    "permanent": true
  }
}
```

In Middleware, you can read from a database such as Vercel's Edge Config ↗ or Redis ↗, and redirect the user based on the incoming request:

middleware.ts — TypeScript

```
import { NextResponse, NextRequest } from 'next/server'
import { get } from '@vercel/edge-config'

type RedirectEntry = {
  destination: string
  permanent: boolean
}

export async function middleware(request: NextRequest) {
  const pathname = request.nextUrl.pathname
  const redirectData = await get(pathname)

  if (redirectData && typeof redirectData === 'string') {
    const redirectEntry: RedirectEntry = JSON.parse(redirectData)
    const statusCode = redirectEntry.permanent ? 308 : 307
```

```
    return NextResponse.redirect(redirectEntry.destination, statusCode)
  }

  // No redirect found, continue without redirecting
  return NextResponse.next()
}
```

2. Optimizing data lookup performance

Reading a large dataset for every incoming request can be slow and expensive. There are two ways you can optimize data lookup performance:

- Use a database that is optimized for fast reads, such as Vercel Edge Config ↗ or Redis ↗.
- Use a data lookup strategy such as a Bloom filter ↗ to efficiently check if a redirect exists **before** reading the larger redirects file or database.

Considering the previous example, you can import a generated bloom filter file into Middleware, then, check if the incoming request pathname exists in the bloom filter.

If it does, forward the request to a Route Handler which will check the actual file and redirect the user to the appropriate URL. This avoids importing a large redirects file into Middleware, which can slow down every incoming request.

middleware.ts TypeScript

```ts
import { NextResponse, NextRequest } from 'next/server'
import { ScalableBloomFilter } from 'bloom-filters'
import GeneratedBloomFilter from './redirects/bloom-filter.json'

type RedirectEntry = {
  destination: string
  permanent: boolean
}

// Initialize bloom filter from a generated JSON file
const bloomFilter = ScalableBloomFilter.fromJSON(GeneratedBloomFilter as any)

export async function middleware(request: NextRequest) {
  // Get the path for the incoming request
  const pathname = request.nextUrl.pathname

```

```
  // Check if the path is in the bloom filter
  if (bloomFilter.has(pathname)) {
    // Forward the pathname to the Route Handler
    const api = new URL(
      `/api/redirects?pathname=${encodeURIComponent(request.nextUrl.pathname)}`,
      request.nextUrl.origin
    )

    try {
      // Fetch redirect data from the Route Handler
      const redirectData = await fetch(api)

      if (redirectData.ok) {
        const redirectEntry: RedirectEntry | undefined =
          await redirectData.json()

        if (redirectEntry) {
          // Determine the status code
          const statusCode = redirectEntry.permanent ? 308 : 307

          // Redirect to the destination
          return NextResponse.redirect(redirectEntry.destination, statusCode)
        }
      }
    } catch (error) {
      console.error(error)
    }
  }

  // No redirect found, continue the request without redirecting
  return NextResponse.next()
}
```

Then, in the Route Handler:

app/redirects/route.ts TypeScript

```
import { NextRequest, NextResponse } from 'next/server'
import redirects from '@/app/redirects/redirects.json'

type RedirectEntry = {
  destination: string
  permanent: boolean
}

export function GET(request: NextRequest) {
```

```
  const pathname = request.nextUrl.searchParams.get('pathname')
  if (!pathname) {
    return new Response('Bad Request', { status: 400 })
  }

  // Get the redirect entry from the redirects.json file
  const redirect = (redirects as Record<string, RedirectEntry>)[pathname]

  // Account for bloom filter false positives
  if (!redirect) {
    return new Response('No redirect', { status: 400 })
  }

  // Return the redirect entry
  return NextResponse.json(redirect)
}
```

Good to know:

- To generate a bloom filter, you can use a library like `bloom-filters` ↗.
- You should validate requests made to your Route Handler to prevent malicious requests.

Next Steps

App Router > ... > Functions

redirect

API Reference for the redirect function.

App Router > ... > Functions

permanentRedirect

API Reference for the permanentRedirect function.

App Router > ... > Routing

Middleware

Learn how to use Middleware to run code before a request is completed.

App Router > ... > next.config.js Options

redirects

Add redirects to your Next.js app.

Previous
Error Handling

Next
Route Groups

Was this helpful?

Subscribe to our newsletter

Stay updated on new releases and features, guides, and case studies.

you@domain.com Subscribe