Next.js Cheat Sheet

A concise cheat sheet covering essential Next.js concepts, commands, and best practices for building efficient and scalable React applications.

Getting Started & Basic Concepts

Project Setup

Create a new Next.js app:

```
npx create-next-app@latest my-nextjs-app
cd my-nextjs-app
```

Start the development server:

```
npm run dev
# or
yarn dev
# or
pnpm dev
# or
bun dev
```

Build for production:

```
npm run build
# or
yarn build
# or
pnpm build
# or
bun build
```

Start the production server:

```
npm run start
# or
yarn start
# or
pnpm start
# or
bun start
```

Key Concepts

Pages	Files in the `pages` directory become routes based on their filename. For example, `pages/about.js` becomes `/about`.
Components	Reusable pieces of UI. Can be functional components or class components.
Layouts	Components that wrap pages to provide a consistent UI structure across different routes. Often implemented using a `_app.js` file or layout components.
API Routes	Serverless functions defined in `pages/api` for handling backend logic directly within your Next.js application.

File Structure

`pages/` - Contains React components that are automatically converted into routes.
`public/` - For static assets like images, fonts, etc.
`components/` - (Optional) A common place to store React components.
`styles/` - For CSS modules, global stylesheets, etc.
`_app.js` - Custom app component for initializing pages. Can be used for layouts, global styles, and more.
`_document.js` - Custom document for controlling the `<html>` tag. Advanced usage.

Routing & Navigation

Basic Routing

Files in the `pages` directory automatically become routes.

`pages/index.js` -> `/` (the homepage)
`pages/about.js` -> `/about`
`pages/blog/index.js` -> `/blog`
`pages/blog/[id].js` -> `/blog/:id` (dynamic route)

Link Component

Import	`import Link from 'next/link';`
Usage	`<Link href="/about">` `<a>About Us</a>` `</Link>`
Prefetching	The `Link` component automatically prefetches pages in the background for faster navigation. It makes the page faster when you click the link.

Dynamic Routes

Use bracket syntax `[]` to create dynamic routes. For example, `pages/posts/[id].js` will handle routes like `/posts/1`, `/posts/2`, etc.

Access the route parameters using the `useRouter` hook:

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

function Post() {
  const router = useRouter();
  const { id } = router.query;

  return <p>Post: {id}</p>;
}
```

useRouter Hook

Import	`import { useRouter } from 'next/router';`
Properties	`router.pathname`: The path of the current page. `router.query`: An object containing the query parameters. `router.asPath`: The path in the browser (including the query parameters). `router.push(url, as, options)`: Programmatically navigate to a new page. `router.replace(url, as, options)`: Programmatically replace the current route in the history stack.

Data Fetching

Data Fetching Methods

Next.js provides several data fetching methods for different use cases:

- `getStaticProps`: Fetch data at build time.
- `getServerSideProps`: Fetch data on each request.
- `getStaticPaths`: Specify dynamic routes to pre-render based on data.

getStaticProps

Description	Fetches data at build time. Ideal for content that doesn't change frequently (e.g., blog posts, marketing pages).
Usage	`export async function getStaticProps(context) { const data = await fetchData(); return { props: { data }, // will be passed to the page component as props revalidate: 10, // Optional: Refetch data every 10 seconds }; }`
When to Use	Use when you can pre-render the page at build time based on the data.

```
export async function getStaticProps(context) {
  const data = await fetchData();

  return {
    props: { data }, // will be passed to the page component as props
    revalidate: 10, // Optional: Refetch data every 10 seconds
  };
}
```

getServerSideProps

Description	Fetches data on each request. Use for data that changes frequently or requires authentication.
Usage	`export async function getServerSideProps(context) { const data = await fetchData(context.req, context.res); return { props: { data }, // will be passed to the page component as props }; }`
When to Use	Use when you need to fetch data on every request, such as when you have user-specific data or data that updates very frequently.

```
export async function getServerSideProps(context) {
  const data = await fetchData(context.req, context.res);

  return {
    props: { data }, // will be passed to the page component as props
  };
}
```

getStaticPaths

Description	Specifies which dynamic routes to pre-render at build time. Required for dynamic routes when using `getStaticProps`.
Usage	`export async function getStaticPaths() { const paths = await getAllPostIds(); return { paths, fallback: false, // or 'blocking' or true }; }`
Fallback Options	`fallback: false`: Any paths not returned by `getStaticPaths` will result in a 404 page. `fallback: true`: Next.js will serve a static page with a loading indicator. After the page is generated, it will be cached and served for future requests. `fallback: 'blocking'`: The user will wait for the page to be generated; Next.js will server the complete page for future requests.

```
export async function getStaticPaths() {
  const paths = await getAllPostIds();
  return {
    paths,
    fallback: false, // or 'blocking' or true
  };
}
```

API Routes and Middleware

API Route Basics

Create API endpoints by creating files in the `pages/api` directory.

```
pages/api/hello.js
```

API routes are server-side only and won't increase your client-side bundle size.

API Route Handler

Example	`export default function handler(req, res) { res.status(200).json({ name: 'John Doe' }); }`
Request Object (`req`)	Contains information about the incoming request, such as headers, query parameters, and body.
Response Object (`res`)	Used to send a response back to the client. Includes methods like `res.status()`, `res.json()`, `res.send()`, etc.

```
export default function handler(req, res) {
  res.status(200).json({ name: 'John Doe' });
}
```

Middleware

Next.js 13+ introduced Middleware to run code before a request is completed. You can rewrite, redirect, add headers, or even block requests based on the incoming request.

Create a `middleware.js` or `middleware.ts` file in the root directory.

Middleware Example

Example	`import { NextResponse } from 'next/server' import type { NextRequest } from 'next/server' export function middleware(request: NextRequest) { if (request.nextUrl.pathname.startsWith('/admin')) { return NextResponse.rewrite(new URL('/login', request.url)) } } export const config = { matcher: ['/about/:path*', '/dashboard/:path*'], }`
Matcher	The `matcher` config defines on which paths the middleware should run.

```
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

export function middleware(request: NextRequest) {
  if (request.nextUrl.pathname.startsWith('/admin')) {
    return NextResponse.rewrite(new URL('/login', request.url))
  }
}

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