# Next.Js

Interview Questions

## Q: What is Next.js?

A: Next.js is a React framework developed by Vercel that provides infrastructure for server-rendered React applications. It includes features such as server-side rendering (SSR), static site generation (SSG), file-system based routing, and API routes. It's designed to be a comprehensive solution for building production-ready, highly scalable, and SEO-friendly React applications.

## Q: What are the key features of `Next.js`?

A: Key features of `Next.js` include: `server-side` rendering (`SSR`), static site generation (`SSG`), automatic code splitting, `file-system` based `routing`, `API routes`, hot code reloading, `built-in` CSS and `Sass` support, `image` optimization, automatic `TypeScript` support, and a `development` environment with hot module replacement.

## Q: What is the difference between `Next.js` and `React.js`?

A: `React.js` is a `JavaScript` library for building user interfaces, mainly for `single-page` applications. It allows for building reusable UI `components`. `Next.js` is a framework built on top of `React` that provides additional functionality like `server-side` rendering, static site generation, `routing`, and `API routes` that aren't available in `React` out of the box. `Next.js` essentially provides a more comprehensive solution while `React` focuses on UI `components`.

## Q: What is the difference between `Next.js` and Create `React` App?

A: Create `React` App is a `client-side` rendering app generator, meaning all rendering takes place in the client's browser. `Next.js`, on the other hand, is a more complete framework offering `server-side` rendering, static site generation, and `file-system` based `routing`. While Create `React` App is focused on simplifying the setup of `React` applications, `Next.js` provides additional `production-ready` features with better `SEO` capabilities.

## Q: How is `Next.js` different from Gatsby?

A: `Next.js` is a framework for `server-rendered` `React` applications with both `SSR` and `SSG` capabilities, while Gatsby is primarily a static site generator. Gatsby is great for static websites with content from various sources, while `Next.js` offers more flexibility for dynamic applications and `server-rendered` content. `Next.js` can better handle frequently changing data and `larger-scale` applications.

## Q: What is server-side rendering (SSR) in Next.js?

A: Server-side rendering is a technique where the server renders the web page and sends the fully rendered HTML to the client. Next.js provides optimized server-side rendering, allowing the initial page load to have all the necessary HTML content available immediately. This improves performance and SEO as search engines can easily index the content, and users don't have to wait for JavaScript to load and execute to see the page content.

## Q: What is static site generation (SSG) in Next.js?

A: Static Site Generation is the pre-rendering of pages at build time. The pre-rendered HTML pages are then reused on each request. Next.js supports this with the getStaticProps and getStaticPaths functions. SSG is useful when the content doesn't need to be updated in real-time or very frequently, as it provides excellent performance and can be cached by CDNs.

## Q: What is Incremental Static Regeneration (ISR) in Next.js?

A: Incremental Static Regeneration allows you to update existing statically generated pages by re-rendering them in the background as traffic comes in. With ISR, you can create static pages with getStaticProps and set a revalidate property to specify how frequently the page should be regenerated. This gives you the benefits of static generation while keeping content fresh without rebuilding the entire site.

## Q: How does `file-based` `routing` work in `Next.js`?

A: `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 path to the file becomes the URL path. For example, if you create a file at `pages/about`.js, it will be accessible at `yoursite.com`/about. This eliminates the need for a separate `routing` configuration.

## Q: What are dynamic routes in `Next.js` and how do you create them?

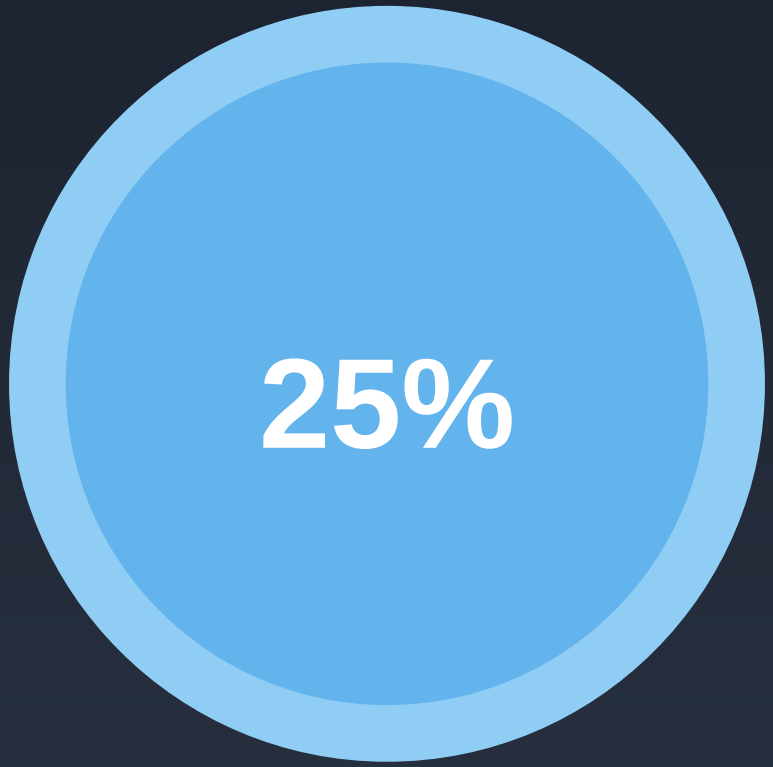
A: Dynamic routes allow you to create pages with paths that depend on external data. You can create dynamic routes by adding brackets [] around a segment of the filename in the pages directory. For example, `pages/posts/[id].js` would match /`posts/1`, /`posts/2`, etc. The dynamic segments are available in the `query` property of the page `components`, or as parameters in `getStaticPaths` and `getServerSideProps`.

## Q: What is the purpose of the `_app.js` file in `Next.js`?

A: `_app.js` is a custom App `component` used to initialize pages. You can override it to control page initialization. This allows you to do things like persist layout between page changes, keep `state` when navigating pages, add global CSS, handle errors, and inject additional data into pages.

## Q: What is the purpose of the `_document.js` file in `Next.js`?

A: `_document.js` is used to augment your application's `<html>` and `<body>` tags. This is necessary because `Next.js` pages skip the definition of the surrounding document's markup. This file is only rendered on the server, so event handlers like onClick cannot be used in _document.




You're 25% through! Keep going!

Success is built one step at a time.

## Q: What are API Routes in Next.js?

A: API Routes allow you to create API endpoints inside a Next.js application by creating files in the pages/api directory. These are serverless functions that run on the server-side and don't increase the client-side bundle size. They're great for handling form submissions, database operations, authentication, and other server-side operations.

## Q: What is `getInitialProps` in `Next.js`?

A: `getInitialProps` is a legacy `async` static method in `Next.js` that fetches data on the server and passes it as `props` to the page `component`. It enables `server-side` rendering but runs on both server and client when navigating between pages. For newer applications, it's recommended to use `getServerSideProps` or `getStaticProps` instead.

## Q: What is `getStaticProps` in `Next.js`?

A: `getStaticProps` is a `Next.js` data fetching method that runs at build time in `production` and allows your page to `fetch` data ahead of time. The fetched data is then used to `pre-render` the page as static HTML, which can be cached and reused for each `request`, improving performance.

## Q: What is `getServerSideProps` in `Next.js`?

A: `getServerSideProps` is a `Next.js` data fetching method that runs on every `request` on the `server-side`. It fetches data for a page and passes it as `props` to the `component`. This is useful for pages where data changes frequently and you need the latest data on each `request`.

## Q: What is the difference between `getStaticProps` and `getServerSideProps`?

A: `getStaticProps` generates static HTML at build time and reuses it on each `request`, making it faster but potentially stale. `getServerSideProps` generates `server-side` rendered HTML on each `request`, providing always fresh data but potentially slower performance. Choose based on how frequently your data changes and your performance requirements.

## Q: How do you handle CSS in `Next.js`?

A: `Next.js` has `built-in` support for CSS and `Sass` which allows you to `import` .css and .scss files directly in your `components`. It supports CSS Modules which automatically scope CSS to `components`, global CSS `import` in `_app.js`, and `CSS-in-JS` solutions like `styled-jsx`, `styled-components`, and emotion.

## Q: What is the `Image component` in `Next.js` and why is it useful?

A: The `Image component` (`next/image`) is a extension of the HTML img element that provides automatic `image` optimization. It automatically serves images in modern formats (WebP), resizes images to avoid shipping large images to small viewports, lazy loads images by default (loading them only when they enter the viewport), and applies best practices for Core Web Vitals and performance.

## Q: How does `Next.js` handle error pages?

A: `Next.js` has `built-in` error handling with default error pages. You can customize these by creating a `_error.js` file (for general errors) or a `404.js` file (for not found errors) in the pages directory. These files are `React components` that will be displayed when the corresponding error occurs.

## Q: How to deploy a `Next.js` application?

A: You can deploy a `Next.js` application on any hosting provider that supports `Node.js`. The easiest way is to use Vercel (created by the same team behind `Next.js`), which offers optimized `deployment` with features like automatic HTTPS, global CDN, and `analytics`. Alternatives include Netlify, `AWS` Amplify, or traditional hosting where you build the app (npm run build) and then `start` the server (npm `start`).

## Q: How can you optimize performance in `Next.js`?

A: `Next.js` automatically optimizes applications through code splitting, `image` optimization, and automatic static optimization. You can further improve performance by using appropriate data fetching methods (`SSG` over `SSR` when possible), implementing `ISR` for `semi-dynamic` content, using the `Image component`, leveraging the `Link component` for prefetching, minimizing `JavaScript` with `production` builds, and using the `Next.js` Analyzer to identify large bundles.

## Q: What is SWR in `Next.js`?

A: SWR (`stale-while-revalidate`) is a `React Hooks` library for data fetching created by the Vercel team. It returns cached data first (stale), then sends the `fetch request` (revalidate), and `finally` updates the data with the latest values. SWR features include automatic revalidation, focus tracking, error retries, and support for `real-time` data. It's commonly used with `Next.js` for `client-side` data fetching.

## Q: How do you add environment variables in `Next.js`?

A: You can add environment variables in `Next.js` by creating a `.env.local` file in the root of your project. `Next.js` automatically loads environment variables from this file. By default, environment variables are only available in `Node.js` environment. To expose variables to the browser, prefix them with `NEXT_PUBLIC_`. You can also use `.env.development` and `.env.production` for `environment-specific` variables.


Halfway there! Every expert was once a beginner.

## Q: How do you `enable` `TypeScript` in `Next.js`?

A: To `enable` `TypeScript` in `Next.js`, you need to install `TypeScript` and then create a `tsconfig.json` file in your project root. `Next.js` will automatically populate this file with default values. After that, you can create pages with `.tsx` extension and use `TypeScript` in your `components`. `Next.js` provides `built-in` `TypeScript` types for its APIs and `components`.

## Q: How do you `enable` a custom webpack configuration in `Next.js`?

A: `Next.js` allows for custom webpack configuration by creating a `next.config`.js file in the root directory. This file should `export` an `object` with a webpack property that is a `function`. This `function` receives the default webpack configuration which you can modify and `return`. This allows you to add loaders, plugins, or modify any webpack behavior.

## Q: What is the 'Link' component in Next.js?

A: The Link component from 'next/link' is a React component for client-side navigation between routes in Next.js applications. It wraps anchor tags and provides functionality like prefetching pages in the background for faster navigation. Using Link prevents full page refreshes, maintaining application state and providing a smoother user experience compared to regular anchor tags.

## Q: Can you use `Next.js` with a `backend` other than `Node.js`?

A: Yes, you can use `Next.js` with any `backend` or no `backend` at all. While `Next.js` itself runs on `Node.js`, you can make `API` calls to any `backend service` (`REST`, `GraphQL`, etc.). You can either use `Next.js API routes` as a proxy to your `backend` or connect directly from the `client-side`. `Next.js` is `backend-agnostic` and works well with services written in any language.

## Q: Can you use `GraphQL` with `Next.js`?

A: Yes, you can use `GraphQL` with `Next.js`. You can add any `GraphQL` client library like Apollo Client or urql to communicate with a `GraphQL` server. You could also create a `GraphQL` server inside `Next.js` using `API routes` with libraries like Apollo Server. `GraphQL` can be used with any of `Next.js`'s data fetching methods like `getStaticProps` or `getServerSideProps`.

## Q: How does Next.js handle SEO?

A: Next.js is excellent for SEO because it supports server-side rendering and static site generation, allowing search engines to easily crawl and index content. You can add meta tags using the Head component from next/head to customize page titles, descriptions, and other metadata. Next.js also supports generating sitemaps and robots.txt files through plugins or custom API routes.

## Q: What is Automatic Static Optimization in `Next.js`?

A: Automatic Static Optimization is a feature in `Next.js` where it automatically determines which pages can be `pre-rendered` as static HTML at build time. If a page doesn't have blocking data requirements (no `getServerSideProps` or `getInitialProps`), `Next.js` will automatically statically optimize it, generating HTML files that can be cached by CDNs for better performance.

## Q: How do you prefetch pages in `Next.js`?

A: `Next.js` automatically prefetches pages linked by the `Link component` when they appear in the viewport. This happens in `production` only and makes navigating to linked pages almost instant. You can also manually control prefetching behavior using the prefetch prop on the `Link component`, or use the `router.prefetch` method from `useRouter` hook for programmatic prefetching.

## Q: How can you handle forms in `Next.js`?

A: Handling forms in `Next.js` is similar to regular `React` applications. You can use controlled `components` with `state` `hooks` like `useState` for form data, handle form submission with onSubmit events, and process the data either `client-side` or by sending it to an `API` route. For more complex forms, you can use libraries like Formik, `react-hook-form`, or `react-final-form`, which integrate well with `Next.js`.

## Q: Can `Next.js` be used for building mobile applications?

A: `Next.js` is primarily designed for web applications, not native mobile apps. While you can create responsive web apps that work well on mobile browsers using `Next.js`, it's not a framework for building native mobile applications. For native mobile app `development` with `React`, `React` Native would be more appropriate. Some developers use `Next.js` for a web version and `React` Native for native mobile versions of the same application.

## Q: How does `Next.js` handle static file serving?

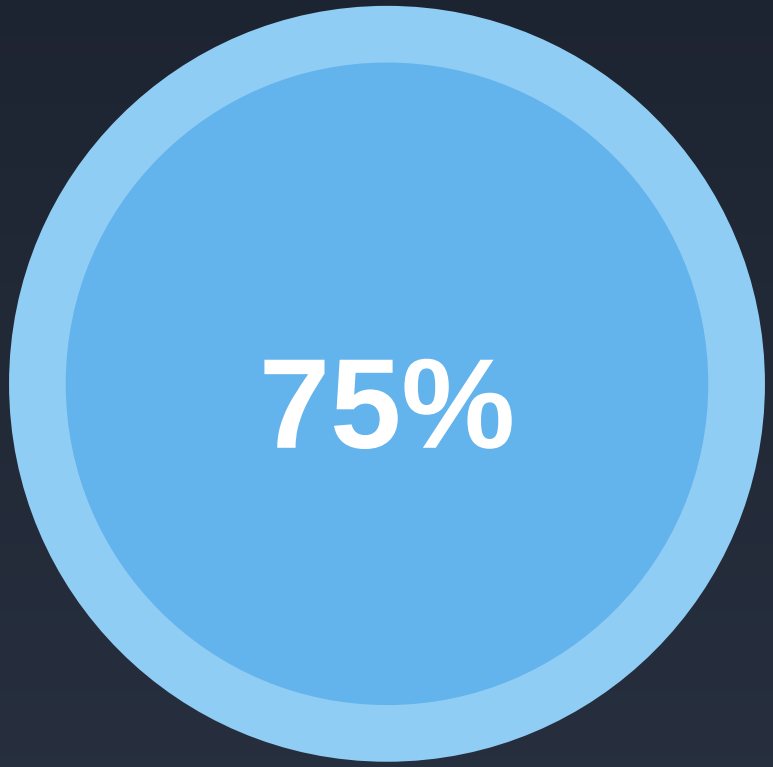
A: `Next.js` serves static files like images, fonts, and other media from the public directory at the root of the project. Files inside public can be referenced from the root of the application. For example, `public/logo.png` can be accessed as `/logo.png` in your application. This directory is ideal for `robots.txt`, `favicon.ico`, and other static assets that don't require processing.

## Q: What is `getStaticPaths` in `Next.js`?

A: `getStaticPaths` is a `function` used with dynamic routes and `getStaticProps` to specify which paths should be `pre-rendered` at build time. It returns an `array` of path objects with params that match the dynamic parts of your route. This `function` is necessary when using `getStaticProps` with dynamic routes, as `Next.js` needs to know which pages to generate at build time.

## Q: What is the `useRouter` hook in `Next.js`?

A: `useRouter` is a hook from the `next/router` package that gives you access to the router `object` inside your `components`. It provides information about the current route, `query` parameters, and methods for navigating between pages programmatically. You can use it to access route parameters, detect route changes, perform redirects, and handle `client-side` navigation.




You're at 75%! Almost done, push through and finish strong!

## Q: How do you implement internationalization (i18n) in Next.js?

A: Next.js has built-in support for internationalization through its i18n routing feature. You can configure supported locales in next.config.js, define which locale is the default, and specify whether to use sub-path routing (/en/blog) or domain routing (en.example.com/blog). Next.js automatically handles locale detection, redirects, and provides locale information to your pages through the useRouter hook.

## Q: What is `Next.js` `Middleware` and how is it used?

A: `Next.js` `Middleware` is code that runs on the edge before a `request` is completed, allowing you to modify responses based on incoming requests. You can create a `middleware.js` file at the root of your project or in the pages directory. `Middleware` can be used for `authentication`, bot protection, redirects, rewriting URLs, adding headers, and handling localization. It runs before rendering and can control access to pages and `API routes`.

## Q: What are `Next.js` Preview Mode and Draft Mode?

A: Preview Mode (renamed to Draft Mode in `Next.js` 13) is a feature that allows you to temporarily bypass static generation for specific pages. It's useful for content management systems where you want to preview draft content before publishing. You can `enable` it through `API routes` and then use cookies to tell `Next.js` to `fetch` the latest data instead of using cached static content.

## Q: What is the '`next/script`' `component` and how does it help with script loading?

A: The Script `component` from `next/script` is used to optimize loading of `third-party` scripts. It offers various loading strategies like beforeInteractive (for critical scripts), afterInteractive (default, loads after page becomes interactive), lazyOnload (loads during idle time), and worker (loads in a web worker). This `component` helps improve performance by controlling when and how external scripts are loaded without blocking page rendering.

## Q: What is the App Router in Next.js 13 and how does it differ from the Pages Router?

A: The App Router, introduced in Next.js 13, is a new routing system built on React Server Components. Unlike the Pages Router which uses the pages directory, the App Router uses the app directory and implements nested routing with layouts, loading states, and error handling. It offers features like server components, streaming, parallel routes, and intercepting routes. The App Router is more powerful but has a different mental model than the Pages Router.

## Q: What are React Server Components in Next.js?

A: React Server Components (RSC) are a new type of component available in Next.js that run exclusively on the server. They can access server resources directly (like databases), reduce the client-side JavaScript bundle, and improve performance. Server Components can't use hooks, browser APIs, or event handlers, but they can be interspersed with Client Components which have full interactivity. In the App Router, components are Server Components by default.

## Q: How do you implement data mutations in Next.js applications?

A: Data mutations in Next.js can be implemented through several approaches: 1) Using API routes to create REST endpoints that handle data changes, 2) Implementing form submissions that post to these API routes, 3) Using client-side libraries like SWR or React Query with their mutation methods, 4) In Next.js 13+ App Router, you can use Server Actions which allow server-side functions to be called directly from client components. The approach depends on your architecture and data requirements.

## Q: What is `next build` and what does it generate?

A: The `next build` command creates an optimized `production` build of your `Next.js` application. It generates .next directory containing: 1) HTML files for statically generated pages, 2) `JavaScript` bundles for `client-side` rendering, 3) `Server-side` code for `API routes` and `server-rendered` pages, 4) JSON files with page data, and 5) Various optimization artifacts. The command also outputs build statistics, showing page sizes, rendering methods used, and route information.

## Q: How can you implement `authentication` in `Next.js`?

A: `Next.js` supports multiple `authentication` methods: 1) Using `NextAuth.js`/`Auth.js`, a popular library with `built-in` providers for `OAuth`, `email/password`, etc., 2) Implementing custom `authentication` with `API routes` for `login/logout`/session management, 3) Using a `third-party service` like `Firebase`, Auth0, or Clerk. With the App Router, you can implement `authentication` using `middleware`, server `components`, and route handlers. Session `state` can be managed with cookies, `JWT` tokens, or external providers.

## Q: What is the `Next.js` Compiler (`Rust-based`) and what benefits does it provide?

A: The `Next.js` Compiler (also known as SWC or Speedy Web Compiler) is a `Rust-based` compiler introduced to replace Babel and Terser in the `Next.js` build process. It provides significantly faster builds (up to 17x faster) and refreshes (up to 5x faster) compared to the previous `JavaScript-based` tooling. The compiler handles `TypeScript`, `JSX` transformation, minification, and various optimizations while maintaining high compatibility with existing code.

## Q: What are Edge Runtime and `Node.js` Runtime in `Next.js`?

A: `Next.js` offers two runtime environments: 1) `Node.js` Runtime, which provides full access to `Node.js` APIs and npm packages but may have slower cold starts, and 2) Edge Runtime, which is a lightweight `JavaScript` environment that runs closer to users with faster startup times but limited `API` access. You can specify which runtime to use for `API routes`, Server `Components`, and `middleware` using the `export const` runtime = 'edge' or 'nodejs' directive.

## Q: How does `Next.js` handle `caching` and what types of cache does it use?

A: `Next.js` `implements` multiple `caching` mechanisms: 1) Full Route Cache for rendered UI at build time or during revalidation, 2) Router Cache for temporarily storing UI on the client while navigating, 3) `Request` Memoization for `caching` `fetch` requests and responses during a single `render` pass, 4) Data Cache for persisting `fetch` responses across multiple renders, and 5) `Client-side` cache with SWR or `React Query`. `Caching` can be configured using `fetch` options, Segment Config options, or route handlers.

## Q: What is the `<Suspense>` `component` in `Next.js` and how is it used?

A: The Suspense `component` in `Next.js` allows you to display a fallback UI while `components` that require data are loading. It works with `React`'s concurrent features and is especially powerful with `React` Server `Components` in the App Router. By wrapping `components` in Suspense, you can create streaming UI patterns where different parts of the page load as their data becomes available, improving perceived performance and user experience while preventing blocking rendering.

# Thank You!

You've completed all the questions.

GM SUNSHINE