Next.js Interview Notes & Detailed Q&A;

1. What is Next.js?

Next.js is a React framework developed by Vercel that enables server-side rendering (SSR), static site generation (SSG), incremental static regeneration (ISR), and client-side rendering (CSR) within a single project. It provides file-based routing, API routes, middleware, and built-in performance optimizations such as automatic code splitting and image optimization. It is widely used for SEO-friendly, production-grade web applications.

2. Difference between Next.js and React?

React is a JavaScript library focused only on building UI components, while Next.js is a framework that extends React with powerful features:
- Server-side rendering (SSR)
- Static site generation (SSG)
- File-based routing system
- API routes
- Image optimization and performance enhancements

In short, React is the base library, and Next.js provides the structure and server-side features for production apps.

3. What are the different rendering methods in Next.js?

Next.js supports multiple rendering strategies:

1. **Server-Side Rendering (SSR):** Pages are generated on every request using `getServerSideProps`.
- Pros: Always fresh data.
- Cons: Slower response time.

2. **Static Site Generation (SSG):** Pages are generated at build time using `getStaticProps`.
- Pros: Very fast, CDN friendly.
- Cons: Content is static unless ISR is used.

3. **Incremental Static Regeneration (ISR):** Allows re-rendering of static pages after deployment using `revalidate`.

4. **Client-Side Rendering (CSR):** Data is fetched client-side after page load using React hooks like `useEffect`.

4. Explain getStaticProps with example.

`getStaticProps` is used to pre-render pages at build time. It fetches data ahead of time and generates static HTML.

Example:
```js
export async function getStaticProps() {
const res = await fetch('https://jsonplaceholder.typicode.com/posts');
const posts = await res.json();
```

```
return {
props: { posts },
revalidate: 60, // ISR: Regenerate every 60 seconds
};
}
```

This example fetches posts at build time and regenerates them every 60 seconds. Best used for blogs, product pages, or content that doesn't change frequently.

5. Explain getServerSideProps with example.

`getServerSideProps` runs at request time on the server. It is used for data that changes frequently.

Example:
```js
export async function getServerSideProps(context) {
const res = await fetch('https://api.example.com/data');
const data = await res.json();

return { props: { data } };
}
```

This fetches fresh data on every request. Best used for dashboards, user-specific data, or frequently updated content.

6. Explain getStaticPaths with example.

`getStaticPaths` is used along with `getStaticProps` for dynamic routes. It specifies which paths should be statically generated.

Example:
```js
export async function getStaticPaths() {
const res = await fetch('https://jsonplaceholder.typicode.com/posts');
const posts = await res.json();

const paths = posts.map(post => ({ params: { id: post.id.toString() } }));

return { paths, fallback: 'blocking' };
}
```

This generates static pages for all post IDs. The `fallback` option controls behavior for non-prebuilt routes.

7. What is Incremental Static Regeneration (ISR)?

ISR allows Next.js to update static pages after the initial build, without rebuilding the entire site. This provides the scalability of static generation with the freshness of server rendering.

Key points:
- Uses `revalidate` in `getStaticProps`.
- Rebuilds only the requested page, not the whole app.

- Ideal for large content-heavy sites like e-commerce, blogs, or news.

Example:
```js
export async function getStaticProps() {
const data = await fetchData();
return {
props: { data },
revalidate: 10, // Re-generate every 10 seconds
};
}
```