1. What is Next.js?

Next.js is a **React framework** that enables developers to build **server-side rendered (SSR), statically generated, and hybrid web applications** with React easily.

It enhances React by providing **routing, performance optimizations, and backend integration** out-of-the-box.

Key Features of Next.js

1. **Server-Side Rendering (SSR)**
    - Pages are rendered on the server and sent as HTML
    - Improves **SEO** and **initial load performance**
2. **Static Site Generation (SSG)**
    - Pre-renders pages at **build time**
    - Fast and efficient for content that doesn't change frequently
3. **API Routes**
    - Build **backend APIs** within the same project
4. **File-Based Routing**
    - Pages are created automatically from **files in the pages/ directory**
5. **Fast Refresh & Hot Reloading**
    - Instant feedback during development
6. **Image & Script Optimization**
    - Built-in components like <Image> and <Script> for better performance

---

Example

Creating a page in Next.js (pages/index.js):

```
export default function Home() {
  return (
    <div>
      <h1>Welcome to Next.js!</h1>
      <p>This is a server-rendered React app.</p>
    </div>
  );
}
```

Creating an API route (pages/api/hello.js):

```
export default function handler(req, res) {
  res.status(200).json({ message: "Hello from Next.js API!" });
}
```

Real-Life Example

Think of **Next.js like a full-service restaurant** 🍽️:

- React → Ingredients (UI components)
- Next.js → Chefs, kitchen, and serving staff (handles routing, rendering, optimization)
- Result → A **ready-to-serve meal (web page)**, fast and optimized

---

Hindi Explanation

Next.js ek React framework hai jo server-side rendering, static site generation aur file-based routing provide karta hai, jisse React apps fast, SEO-friendly aur scalable ban jati hain.

Short Interview Answer

Next.js is a React framework that enables server-side rendering, static site generation, and API routes, providing optimized performance, SEO benefits, and easy routing for modern web applications. 🚀

2. What are the advantages of Next.js over React?

Advantages of Next.js Over React

Next.js is **built on top of React**, but it provides several **additional features and optimizations** that React alone does not offer:

Feature	React	Next.js	Advantage
Rendering	Client-Side Rendering (CSR) by default	Supports SSR, SSG, ISR & CSR	Better **SEO** and **faster initial load**
Routing	Requires **React Router**	**File-based routing** built-in	No extra setup; pages auto-generated
API Integration	Separate backend needed	Built-in **API routes**	Can create backend endpoints inside the same project
Performance	Manual optimization	**Automatic code splitting, image & script optimization**	Faster page load & better UX
SEO	CSR pages are harder to crawl	SSR & SSG pages are SEO-friendly	Better visibility in search engines
Build Time	Only client-side bundle	Pre-render pages (SSG)	Reduces server load and improves speed
Developer Experience	Needs setup for SSR/SSG	**Zero-config SSR/SSG**	Simplifies development, faster setup

Real-Life Example

Think of **React like a standard car** 🚙:

- You can drive anywhere, but you need to **install extra features** for comfort, GPS, or safety.

Next.js is like a **premium car** 🚙:

- Comes with **GPS, cruise control, advanced safety**, and features ready to use out-of-the-box.

Hindi Explanation

Next.js React ka framework hai jo SSR, SSG, file-based routing, API routes aur performance optimizations provide karta hai, jisse React apps fast, SEO-friendly aur scalable ho jati hain.

Short Interview Answer

Next.js provides advantages over React by offering server-side rendering, static site generation, built-in routing, API routes, automatic optimizations, and better SEO, making web apps faster, scalable, and SEO-friendly. 🚀

3. What is server-side rendering (SSR)?

Server-Side Rendering (SSR) is a technique where **web pages are generated on the server** for each request, instead of rendering them in the browser like traditional React apps (CSR).

- The **HTML is pre-built on the server** and sent to the browser
- The browser receives a **fully-rendered page**, which can be displayed immediately
- After that, **React hydrates** the page to make it interactive

---

How SSR Works

1. User requests a page → GET /page
2. Server generates the **HTML content dynamically**
3. Server sends the **rendered HTML** to the browser
4. Browser displays content instantly → then React takes over for interactivity

---

Example in Next.js

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

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

export default function Page({ data }) {
  return (
    <div>
      <h1>Server-Side Rendered Page</h1>
      <p>Data: {data.message}</p>
    </div>
```

```
  );
}
```

- getServerSideProps fetches data **on the server** for every request
- The page is **pre-rendered** before reaching the browser

Real-Life Example

Think of SSR like **a restaurant delivering pre-made meals** 🍲:

- The chef (server) prepares the meal before the customer (browser) receives it
- Customer can eat immediately → no need to cook themselves

In contrast, CSR is like **delivering raw ingredients**, and the customer has to cook (render) at home.

Hindi Explanation

Server-Side Rendering me web page server par render hota hai aur fully-built HTML browser ko bheja jata hai, jisse page fast load hota hai aur SEO-friendly hota hai.

Short Interview Answer

Server-Side Rendering (SSR) is when web pages are generated on the server for each request, sending fully-rendered HTML to the browser for faster loading, better SEO, and immediate content display. 🚀

4. What is static site generation (SSG)?

Static Site Generation (SSG) is a technique where **web pages are pre-rendered at build time**, rather than on each request.

- HTML pages are **generated once during the build** and stored on the server
- When a user requests a page, the **pre-built HTML** is served instantly
- Great for pages with **content that doesn't change often**

How SSG Works

1. During **build time**, the server fetches all necessary data
2. Generates **HTML files** for each page
3. Stores them on the server or CDN
4. On request, server serves **pre-rendered HTML → fast load time**

Example in Next.js

```
export async function getStaticProps() {
  const res = await fetch("https://api.example.com/posts");
  const posts = await res.json();

  return {
    props: { posts }, // passed to the page component
  };
}

export default function Blog({ posts }) {
  return (
    <div>
      <h1>Blog Posts</h1>
      {posts.map((post) => (
        <div key={post.id}>{post.title}</div>
      ))}
    </div>
  );
}
```

- getStaticProps runs **at build time**, not on every request
- Page is **pre-rendered** and served **instantly**

Real-Life Example

Think of SSG like **pre-packaged meals** 🍱:

- Meals are prepared and packaged in advance
- When a customer orders, they get it **immediately** → no waiting for cooking

In contrast, SSR is like **cooking meals on order**.

Hindi Explanation

Static Site Generation me pages build time par pre-render hote hain, aur user ko request par instantly serve kiye jate hain, jo pages ko fast aur SEO-friendly banata hai.

Short Interview Answer

Static Site Generation (SSG) pre-renders pages at build time, serving ready-made HTML for faster load, better SEO, and optimal performance, especially for pages with static content. 🚀

5. What is client-side rendering?

Client-Side Rendering (CSR) is a technique where **web pages are rendered in the browser** using JavaScript, instead of being pre-rendered on the server.

- The server sends a **bare-bones HTML file** and JavaScript
- Browser downloads the JS, executes it, and **builds the HTML dynamically**
- Common in **React, Angular, Vue** apps

How CSR Works

1. User requests a page → GET /page
2. Server sends **HTML shell + JS files**
3. Browser downloads and runs JS → generates the **full HTML content dynamically**
4. Page becomes **interactive after JS loads**

Example

React App (CSR)

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

- HTML initially has only
- JS runs in browser → renders full page

Real-Life Example

Think of CSR like **receiving raw ingredients** 🍅🥬:

- The server gives you ingredients (HTML shell + JS)
- You (browser) cook the meal (render the page) yourself
- Takes longer to **see the complete page**, but interaction is dynamic

SSR/SSG → ready-made meal 🍲
CSR → cook yourself at home 🏠

Hindi Explanation

Client-Side Rendering me page browser me JavaScript ke through dynamically render hota hai. Server sirf basic HTML aur JS bhejta hai, aur browser content generate karta hai.

Short Interview Answer

Client-Side Rendering (CSR) is when the browser renders the full web page using JavaScript after receiving a minimal HTML shell from the server, making pages interactive but potentially slower for initial load. 🚀

6. What is file-based routing in Next.js?

File-based routing is a **feature in Next.js** where the **file structure inside the pages/ directory automatically defines the routes of the application**.

- No need to manually configure routes (like React Router)
- Each file in pages/ becomes a **URL path**
- Supports **dynamic routes** using square brackets [param]

How It Works

Basic Routes

```
pages/
  index.js       → '/'
  about.js       → '/about'
  contact.js     → '/contact'
```

- Visiting /about automatically loads pages/about.js

Nested Routes

```
pages/
  blog/
    index.js     → '/blog'
    [id].js      → '/blog/:id'
```

- Dynamic route [id].js → /blog/123 will match id = 123

Catch-All Routes

```
pages/
  docs/
    [...slug].js → '/docs/*'
```

- Handles /docs/intro/setup, /docs/api/auth, etc.

Example

pages/blog/[id].js

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

export default function BlogPost() {
  const router = useRouter();
  const { id } = router.query;

  return <h1>Blog Post ID: {id}</h1>;
}
```

- Accessing /blog/42 → renders **Blog Post ID: 42**

Real-Life Example

Think of file-based routing like **folders and files in a filing cabinet** 🗂️:

- File name → URL path
- Folder → Nested routes
- Easy to find and organize content without writing extra code

---

Hindi Explanation

Next.js me file-based routing ka matlab hai ki pages/ folder ke andar files aur folders automatically routes create kar dete hain, bina manually route configure kiye.

Short Interview Answer

File-based routing in Next.js automatically maps files in the pages/ directory to URL paths, supporting nested and dynamic routes without additional configuration. 🚀

7. What is getServerSideProps?

getServerSideProps is a **special function in Next.js** used for **Server-Side Rendering (SSR)**.

- Runs **on the server for every request**
- Fetches data before rendering the page
- Returns the data as **props** to the page component

This ensures that the page is **pre-rendered with dynamic data** on each request, which is great for **SEO** and **fast initial load**.

How It Works

1. User requests a page
2. Next.js calls getServerSideProps on the server
3. Fetches data (from API, database, etc.)
4. Returns an object with props
5. Page component receives these props and renders the content

---

Example

```
export async function getServerSideProps(context) {
  const res = await fetch("https://api.example.com/posts");
  const posts = await res.json();

  return {
    props: { posts }, // Passed to the page component
  };
}

export default function BlogPage({ posts }) {
```

```
 return (
  <div>
   <h1>Blog Posts</h1>
   {posts.map((post) => (
    <p key={post.id}>{post.title}</p>
   ))}
  </div>
 );
}
```

- Each time a user visits /blog, **server fetches fresh data**
- Page is **rendered with updated posts**

Real-Life Example

Think of it like **a restaurant preparing a meal when you order** 🍲:

- Chef (server) cooks **fresh food on request**
- Customer (browser) gets **ready-to-eat meal immediately**
- Always up-to-date, unlike pre-made meals

Hindi Explanation

getServerSideProps Next.js me ek function hai jo server par run hota hai aur har request par data fetch karke page ko render karta hai, jisse SEO-friendly aur updated content milta hai.

Short Interview Answer

getServerSideProps in Next.js fetches data on the server for each request, providing dynamic, server-rendered props to a page for SEO-friendly and fast-loading content. 🚀

8. What is getStaticProps?

getStaticProps is a special function in Next.js used for **Static Site Generation (SSG)**.

- Runs **at build time**, **not on every request**
- Fetches data and pre-renders the page into **static HTML**
- Returns the data as **props** to the page component

This makes pages **fast, SEO-friendly, and ideal for content that doesn't change frequently**.

How It Works

1. During **build time**, Next.js runs getStaticProps
2. Fetches data (from API, database, etc.)
3. Generates **HTML pages with the fetched data**

4. Stores the pre-rendered pages
5. On user request, **static HTML is served instantly**

Example

```
export async function getStaticProps() {
 const res = await fetch("https://api.example.com/posts");
 const posts = await res.json();

 return {
  props: { posts }, // Passed to the page component
 };
}

export default function BlogPage({ posts }) {
 return (
  <div>
   <h1>Blog Posts</h1>
   {posts.map((post) => (
    <p key={post.id}>{post.title}</p>
   ))}
  </div>
 );
}
```

- Fetches **all posts at build time**
- Page is **pre-rendered** and served as static HTML
- Very **fast** for end users

Real-Life Example

Think of it like **pre-packaged meals** 🍱:

- Meals are prepared **in advance** during production
- Customers get them **immediately**, no waiting
- Content is **static**, suitable for menus, blogs, or landing pages

In contrast, getServerSideProps is like **cooking meals on order**.

Hindi Explanation

getStaticProps Next.js me ek function hai jo build time par run hota hai aur page ke liye static HTML pre-render karta hai, jisse page fast aur SEO-friendly ban jata hai.

Short Interview Answer

getStaticProps in Next.js fetches data at build time to pre-render static HTML pages, providing fast, SEO-friendly content ideal for pages with mostly static data. 🚀

9. What is getStaticPaths?

getStaticPaths is a special function in Next.js used **along with getStaticProps** for **dynamic routes** in **Static Site Generation (SSG)**.

- Defines **which dynamic routes should be pre-rendered at build time**
- Returns an array of **paths** and a **fallback option**
- Ensures dynamic pages like /blog/[id] are **generated as static HTML**

How It Works

1. You have a **dynamic route**: pages/blog/[id].js
2. Next.js needs to know **which id values to pre-render**
3. getStaticPaths returns a **list of paths**
4. getStaticProps fetches data for each path and pre-renders the page

---

Example

pages/blog/[id].js

```
export async function getStaticPaths() {
  const res = await fetch("https://api.example.com/posts");
  const posts = await res.json();

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

  return { paths, fallback: false };
}

export async function getStaticProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
  const post = await res.json();

  return { props: { post } };
}

export default function BlogPost({ post }) {
  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.content}</p>
    </div>
  );
}
```

- getStaticPaths tells Next.js **which ids to pre-render**
- fallback: false → Only pre-rendered paths exist; others 404

- Each page is **statically generated at build time**

Real-Life Example

Think of it like **preparing tickets 🎫 for a concert**:

- You know the **list of attendees (paths)** in advance
- Tickets are printed (pages pre-rendered)
- Only those on the list get tickets, others cannot enter (fallback: false)

Hindi Explanation

getStaticPaths dynamic routes ke liye Next.js me use hota hai, jisme build time par kaunse paths pre-render honge ye define kiya jata hai.

Short Interview Answer

getStaticPaths in Next.js defines which dynamic route paths should be statically generated at build time, working with getStaticProps to pre-render pages for dynamic content. 🚀

10. What is API routing in Next.js?

API Routing in Next.js allows you to **create backend API endpoints inside the Next.js project** without setting up a separate server like Express.js.

- API routes are **server-side functions**
- Placed in the pages/api/ folder
- Can handle **HTTP requests** (GET, POST, PUT, DELETE)
- Useful for fetching data, authentication, or server-side logic

---

How It Works

1. Each file inside pages/api/ becomes an **API endpoint**
2. Request to that endpoint runs the **server-side function**
3. Response is sent as **JSON or any other format**

Example

pages/api/hello.js

```
export default function handler(req, res) {
  if (req.method === "GET") {
    res.status(200).json({ message: "Hello from Next.js API!" });
  } else {
    res.status(405).json({ error: "Method not allowed" });
```

```
  }
}
```

- Request: GET /api/hello → Response: { message: "Hello from Next.js API!" }
- Supports dynamic APIs with files like [id].js → /api/users/123

Real-Life Example

Think of API routes like **waiters in a restaurant** 🍽️:

- You (frontend) place an **order (HTTP request)**
- Waiter (API route) brings **food (data) from the kitchen (server)**
- Frontend receives the data and displays it to the user

Hindi Explanation

Next.js me API routing ka matlab hai ki hum pages/api/ folder me backend endpoints bana sakte hain jo server-side requests handle karte hain, jaise data fetch karna ya authentication.

Short Interview Answer

API routing in Next.js allows creating server-side endpoints inside the pages/api/ folder to handle HTTP requests, enabling backend logic and data fetching without a separate server. 🚀

11. What is image optimization in Next.js?

Image Optimization in Next.js is a built-in feature that automatically **optimizes images for performance and faster loading**.

- Uses the **next/image** component instead of standard <img>
- Automatically **resizes, compresses, and serves images in modern formats** like WebP
- Supports **lazy loading**, **responsive images**, and **placeholder blur**

Key Features

1. **Automatic resizing** – Serve images at the size required for each device
2. **Lazy loading** – Loads images only when they appear in the viewport
3. **Modern formats** – Converts images to WebP if the browser supports it
4. **Placeholder blur** – Shows a low-quality image while the main image loads
5. **CDN-ready** – Works well with remote image hosting

Example

```
import Image from "next/image";
```

```
export default function Home() {
 return (
  <div>
   <h1>Optimized Image</h1>
   <Image
    src="/images/photo.jpg"
    alt="Beautiful Landscape"
    width={800}
    height={500}
    placeholder="blur"
    blurDataURL="/images/photo-blur.jpg"
   />
  </div>
 );
}
```

- width & height → Next.js optimizes image to these dimensions
- placeholder="blur" → Shows a blurred version while loading
- Automatic **lazy loading** → Improves page speed

Real-Life Example

Think of image optimization like **packing a suitcase efficiently** 🧳 :

- Instead of carrying **full-size heavy images**, Next.js **resizes and compresses them** for faster delivery
- Loads them **only when needed** → saves bandwidth and improves user experience

---

Hindi Explanation

Next.js me image optimization ka matlab hai ki next/image component ke through images automatically resize, compress aur modern formats me serve hoti hain, jisse pages fast load hote hain aur bandwidth bacha hai.

Short Interview Answer

Image optimization in Next.js automatically resizes, compresses, lazy-loads, and serves images in modern formats using the next/image component, improving performance and user experience. 🚀

12. What is middleware in Next.js?

Middleware in Next.js is a **function that runs before a request is completed** and allows you to **intercept, modify, or redirect requests**.

- Runs **on the Edge** (closer to the user) for **fast execution**
- Can be used for:
    - Authentication / authorization
    - Redirects
    - Logging requests
    - Rewriting URLs

- Middleware is placed in the **middleware.js** file at the **root** of your Next.js project or inside specific folders for scoped routes

How It Works

1. A request is made to your Next.js app
2. Middleware runs **before the request reaches the page or API route**
3. You can **modify request/response**, redirect, or block the request

---

Example

middleware.js

```
import { NextResponse } from "next/server";

export function middleware(request) {
  const { pathname } = request.nextUrl;

  // Redirect users from /admin if not logged in
  const isLoggedIn = false; // Example check
  if (pathname.startsWith("/admin") && !isLoggedIn) {
    return NextResponse.redirect(new URL("/", request.url));
  }

  return NextResponse.next(); // Allow the request
}
```

- Requests to /admin → redirected to / if not logged in
- Other requests → continue normally

Real-Life Example

Think of middleware like a **security guard at a gate** 🚪 :

- Every visitor (request) passes the guard first
- Guard checks **access permission** → allows, redirects, or blocks
- Only authorized visitors reach the inner area (pages or API routes)

Hindi Explanation

Next.js me middleware ek function hai jo request complete hone se pehle run hota hai aur request ko intercept, modify ya redirect karne ke liye use hota hai, jaise authentication ya logging.

Short Interview Answer

Middleware in Next.js runs before a request reaches a page or API route, allowing you to intercept, modify, redirect, or block requests, commonly used for authentication, logging, and URL rewrites. 🚀

13. What is dynamic routing?

Dynamic Routing in Next.js allows you to create **routes with variable parameters** instead of fixed paths.

- Useful when you have pages that depend on **data or IDs** (e.g., blog posts, products, profiles)
- Achieved using **square brackets [param]** in file names inside the pages/ folder

How It Works

1. Create a **dynamic route file**: pages/blog/[id].js
2. [id] acts as a **placeholder** for any value
3. Use **useRouter** or params from getStaticProps / getServerSideProps to access the value

---

Example

pages/blog/[id].js

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

export default function BlogPost({ post }) {
  const router = useRouter();
  const { id } = router.query;

  return (
    <div>
      <h1>Blog Post ID: {id}</h1>
      <p>{post.content}</p>
    </div>
  );
}

// For SSG
export async function getStaticProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
  const post = await res.json();

  return { props: { post } };
}

// For SSG: define paths to pre-render
export async function getStaticPaths() {
  const res = await fetch("https://api.example.com/posts");
  const posts = await res.json();

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

  return { paths, fallback: false };
}
```

- Access /blog/1 → id = 1
- Access /blog/42 → id = 42
- Same page component renders **different content based on the route**

Real-Life Example

Think of dynamic routing like **hotel rooms** 🏨:

- /rooms/101 → Room 101 details
- /rooms/102 → Room 102 details
- Same template, but content changes based on **room number (dynamic parameter)**

Hindi Explanation

Next.js me dynamic routing ka matlab hai ki routes variable parameters ke saath create kiye ja sakte hain, jaise /blog/[id], jahan id ke hisaab se page content change hota hai.

Short Interview Answer

Dynamic routing in Next.js allows creating routes with variable parameters using [param] in file names, enabling the same page component to render different content based on the URL. 🚀

14. What is incremental static regeneration (ISR)?

Incremental Static Regeneration (ISR) is a Next.js feature that allows **static pages to be updated after build time**, without rebuilding the entire site.

- Combines the **speed of static generation** with **fresh, dynamic content**
- Pages are **pre-rendered at build time**, then **regenerated on-demand** based on a **revalidation interval**
- Great for pages that mostly stay static but occasionally need updates (e.g., blogs, product listings)

How It Works

1. Page is **statically generated at build time**
2. revalidate property defines **how often the page should update**
3. When a user requests the page after the interval:
    - Next.js serves the **existing page immediately**
    - Regenerates the page **in the background**
    - Next request gets the **updated page**

Example

pages/blog/[id].js

```
export async function getStaticProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
```

```
  const post = await res.json();

  return {
    props: { post },
    revalidate: 60, // Regenerate page at most once every 60 seconds
  };
}

export async function getStaticPaths() {
  const res = await fetch("https://api.example.com/posts");
  const posts = await res.json();

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

  return { paths, fallback: "blocking" };
}

export default function BlogPost({ post }) {
  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.content}</p>
    </div>
  );
}
```

- Page is **statically generated at build time**
- After 60 seconds, **Next.js regenerates it in the background**
- Users always get a **fast static page**, but content stays fresh

---

Real-Life Example

Think of ISR like **pre-printed newspapers** 📰:

- Newspapers are printed (pre-rendered) in the morning
- Some sections (news updates) can be refreshed during the day (regenerated)
- Readers get a **ready paper quickly**, but news stays up-to-date

---

Hindi Explanation

Incremental Static Regeneration me Next.js pages build time par pre-render hote hain, aur revalidate interval ke hisaab se background me regenerate hote hain, jisse pages fast aur updated rehte hain.

Short Interview Answer

Incremental Static Regeneration (ISR) in Next.js allows static pages to be updated after build time by regenerating them in the background at a set interval, combining fast load times with fresh content. 🚀

15. What is the Link component?

The **Link component** in Next.js is used for **navigating between pages within a Next.js application**. Unlike a regular HTML <a> tag, it enables **client-side navigation**, which means pages load faster **without a full browser refresh**.

- Used for **internal links** (links between pages of your app)
- Supports **prefetching** pages in the background for faster navigation
- Can wrap elements like <a> or other tags

Example

```
import Link from "next/link";

export default function Home() {
  return (
    <div>
      <h1>Welcome to My Website</h1>
      <Link href="/about">
        <a>Go to About Page</a>
      </Link>
    </div>
  );
}
```

- Clicking the link navigates to /about **without reloading the page**
- Improves **user experience and performance**

Real-Life Example

Think of it like **turning pages in an e-book** 📖:

- You move from chapter to chapter instantly
- No need to fetch the whole book again

Hindi Explanation

Next.js me Link component client-side navigation ke liye use hota hai, jisse pages fast load hote hain aur browser refresh nahi hota.

Short Interview Answer

The Link component in Next.js enables client-side navigation between internal pages, allowing faster, seamless transitions and optional prefetching of linked pages. 🚀