Next.js introduction overview

⚡ PHASE 1: Next.js Basics (App Router)

Learn the fundamentals of Next.js (v14 or newer)

🔹 1. Introduction

What is Next.js? Why use it over React?

Let's start with **Phase 1: Next.js Basics (App Router)** — specifically, **"Introduction"** 👇

🔹 1. What is Next.js?

Next.js is a **React framework** built on top of React that helps you create **full-stack web applications** easily.
It extends React with **powerful features** like:

- **File-based routing**
- **Server-Side Rendering (SSR)**
- **Static Site Generation (SSG)**
- **API routes (backend inside frontend)**
- **Optimized performance & SEO**
- **Built-in image and font optimization**

In short:

> 🧠 *Next.js = React + Routing + Rendering + Optimization + Backend capabilities.*

🔹 2. Why use Next.js over React?

Feature	React	Next.js
Routing	You need to install React Router manually	Built-in **file-based routing** (automatic)
Rendering	Client-side only (CSR)	Supports **SSR**, **SSG**, **ISR**, and CSR
SEO	Not ideal for SEO (content loads after JS)	Excellent SEO (pre-rendered pages)
Backend	Needs separate backend setup (like Express)	Has **API routes** (you can create backend endpoints directly)
Performance	Depends on setup	**Optimized by default** with caching, compression, image optimization

Feature	React	Next.js
Deployment	Needs configuration	Seamless with **Vercel** (the creators of Next.js)

💡 Example:

In React:

```
jsx

import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from "./Home";
import About from "./About";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
```

In Next.js (App Router):

```
java

app/
 ├─ page.js          →  Home page ("/")
 ├─ about/
 │   └─ page.js      →  About page ("/about")
```

✅ No manual routing setup needed — it's **automatic**.

🧭 In summary:

- **React** → UI library (frontend only)
- **Next.js** → Full-stack framework (frontend + backend + optimization)

Would you like me to explain **how the App Router works** (the new routing system introduced in Next.js 13+) next?

> ↪
>
> Would you like me to explain **how the App Router works** (the new routing system introduced in Next.js 13+) next?
> Ok

Perfect 👌 Let's dive into the **App Router** — one of the most important and modern parts of **Next.js (v13 and above)**.

⚡ PHASE 1.2: App Router in Next.js

🔹 What is the App Router?

The **App Router** is a new way of building routes in Next.js (introduced in v13).
It's based on the **file system** — meaning the folder structure inside your `app/` directory **defines your routes automatically**.

It replaces the older **Pages Router** (`pages/` directory).

🔹 Folder structure example

```vbnet
app/
├── page.js               → Renders at "/"
├── about/
│   └── page.js           → Renders at "/about"
├── blog/
│   ├── page.js           → Renders at "/blog"
│   └── [id]/
│       └── page.js       → Dynamic route like "/blog/1", "/blog/2"
├── layout.js             → Shared layout (navbar, footer, etc.)
├── loading.js            → Loading UI (for suspense)
├── error.js              → Error UI (for route-level errors)
└── not-found.js          → Custom 404 page
```

🔹 File roles in the App Router

File	Purpose
`page.js`	Defines a **page component** for a specific route
`layout.js`	Defines a **shared layout** (e.g., header, footer) for that route and its children
`loading.js`	Displays a **loading state** while data is being fetched
`error.js`	Handles **errors** specific to that route
`not-found.js`	Custom 404 page for missing content
`route.js`	Defines an **API endpoint** (acts like a backend file)

🔹 Example: A simple project

```markdown
app/
├── layout.js
├── page.js
├── about/
│   └── page.js
└── contact/
    └── page.js
```

`app/layout.js`

```jsx
export default function RootLayout({ children }) {
  return (
    <html>
      <body>
        <header>🌐 My Website</header>
        {children}
        <footer>© 2025 Soumalya Jana</footer>
      </body>
    </html>
  );
}
```

`app/page.js`

```jsx
export default function HomePage() {
  return <h1>Welcome to My Next.js App 🚀</h1>;
}
```

`app/about/page.js`

```jsx
export default function AboutPage() {
  return <h1>About Us</h1>;
}
```

✅ Visiting `/` → shows HomePage
✅ Visiting **`/about`** → shows AboutPage

Layout stays the same (header/footer), content changes dynamically.

🔹 Why App Router is powerful

- **Server Components** (default) → better performance
- **Automatic code-splitting** → smaller page bundles
- **Colocation** → route-specific files (loading, error, layout) live together
- **Built-in data fetching** using async/await directly in components
- **Better SEO** with server rendering

Would you like me to explain **“Rendering Modes in Next.js”** next — i.e., CSR, SSR, SSG, ISR, and how Next.js handles them automatically?

3. Routing System (App Router)

File-based routing: /app/page.tsx → /

Nested routes: /app/blog/page.tsx → /blog

Excellent 👍 You're now on **PHASE 1.3: Routing System (App Router)** —
Let's understand **how routing works in Next.js (v13+)** step by step 👇

⚡ 3. Routing System (App Router)

In **Next.js App Router**, your **folder structure = route structure**.
You don't need `react-router-dom` or manual route setup — Next.js automatically handles it for you.

🔹 1. File-based Routing

Each folder inside the `app/` directory represents a **route segment**,
and every route must contain a **`page.js` (or `page.tsx`)** file.

Example:

```arduino
app/
 ├─ page.tsx          →  "/"
 ├─ about/
 │   └─ page.tsx      →  "/about"
 ├─ contact/
 │   └─ page.tsx      →  "/contact"
```

✅ Routes created:

- `/` → **`app/page.tsx`**
- `/about` → **`app/about/page.tsx`**
- `/contact` → **`app/contact/page.tsx`**

🔹 2. Nested Routes

To create **nested routes**, you use **nested folders**.

Example:

```arduino
app/
 ├─ blog/
 │   ├─ page.tsx          → "/blog"
 │   ├─ tech/
 │   │   └─ page.tsx      → "/blog/tech"
 │   └─ travel/
 │       └─ page.tsx      → "/blog/travel"
```

✅ Routes created:

- `/blog`
- `/blog/tech`
- `/blog/travel`

Each folder can also have its **own layout** (we'll cover that later).

🔹 3. Dynamic Routes

Use **square brackets `[ ]`** to create **dynamic routes** (like parameters).

Example:

```bash
app/
├─ blog/
│   ├─ page.tsx           → "/blog"
│   └─ [id]/
│       └─ page.tsx       → "/blog/:id"
```

`app/blog/[id]/page.tsx`

```tsx
export default function BlogDetails({ params }) {
  return <h1>Blog ID: {params.id}</h1>;
}
```

✅ Visiting `/blog/5` → displays `Blog ID: 5`

🔹 4. Layout Nesting

Each folder can include a `layout.tsx` file that wraps its pages.
This allows **different sections** of your app to have **different layouts**.

Example:

```css
app/
├─ layout.tsx           → Root layout (header, footer)
├─ page.tsx
├─ blog/
│   ├─ layout.tsx       → Blog-specific layout
│   └─ page.tsx
```

✅ Blog pages will use both layouts:
Root layout → Blog layout → Blog page.

🔹 5. Route Groups (Optional)

You can group routes **without affecting the URL** using **parentheses** `( )`.

Example:

```arduino
app/
├─ (marketing)/
│  ├─ about/
│  │  └─ page.tsx
│  └─ contact/
│     └─ page.tsx
├─ (dashboard)/
│  ├─ home/
│  │  └─ page.tsx
│  └─ settings/
│     └─ page.tsx
```

✅ Routes:

- `/about`
- `/contact`
- `/home`
- `/settings`

Even though grouped, they share separate layouts or logic.

🔹 Summary Table

Type	Example Path	URL
Static	`app/about/page.tsx`	`/about`
Nested	`app/blog/tech/page.tsx`	`/blog/tech`
Dynamic	`app/blog/[id]/page.tsx`	`/blog/:id`
Grouped	`app/(dashboard)/home/page.tsx`	`/home`

Would you like me to continue to the next topic —
"Rendering Modes in Next.js (CSR, SSR, SSG, ISR)" — so you can understand how data and pages are rendered?

Dynamic routes: /app/blog/[slug]/page.tsx

Catch-all routes: [...slug].tsx

Perfect ⚡ You're moving into one of the **most important topics** in Next.js routing —
Dynamic & Catch-all Routes 🔥

Let’s break them down clearly 👇

⚡ 4. Dynamic & Catch-all Routes (App Router)

🔹 1. Dynamic Routes

Dynamic routes let you create **pages with variable path segments** — for example, **`/blog/first-post`**, **`/blog/second-post`**, etc.

To do this, you use **square brackets `[ ]`** in your folder or file name.

Example:

```
arduino

app/
 ├─ blog/
 │   ├─ page.tsx          →  "/blog"
 │   └─ [slug]/
 │       └─ page.tsx      →  "/blog/:slug"
```

✅ URLs generated:

- **`/blog/hello-nextjs`**
- **`/blog/learn-react`**
- **`/blog/nextjs-routing`**

📘 Example Code — `app/blog/[slug]/page.tsx`

```
tsx

export default function BlogPost({ params }) {
  return <h1>Blog Slug: {params.slug}</h1>;
}
```

🧠 Note:

- Next.js automatically passes **`params`** to the component.
- **`{ params.slug }`** gives you the **dynamic value** from the URL.

✅ **`/blog/hello-nextjs`** → **`params.slug = "hello-nextjs"`**

🔹 2. Catch-all Routes (`[...slug]`)

Catch-all routes are used when you don’t know **how many segments** might appear after a path.

Example:

```
css

app/
 └─ docs/
     └─ [...slug]/
         └─ page.tsx
```

✅ URLs matched:

- `/docs`
- `/docs/getting-started`
- `/docs/guides/nextjs/installation`

All of them map to this **one file**.

📘 Example Code — `app/docs/[...slug]/page.tsx`

```
tsx

export default function DocsPage({ params }) {
  return (
    <div>
      <h1>Docs Page</h1>
      <p>Path segments: {JSON.stringify(params.slug)}</p>
    </div>
  );
}
```

✅ `/docs/guides/nextjs` → `params.slug = ["guides", "nextjs"]`

🔹 3. Optional Catch-all Routes (`[[...slug]]`)

If you want the route to also match the **base path** (like `/docs`),
you can make it **optional** using **double brackets** `[[...slug]]`.

Example:

```
lua

app/
 └─ docs/
     └─ [[...slug]]/
         └─ page.tsx
```

✅ URLs matched:

- `/docs`
- `/docs/setup`
- `/docs/setup/installation`

If no slug is provided, `params.slug` will be `undefined`.

🧭 Summary

Route Type	Folder Name	Example URL	`params.slug`
Dynamic	`[slug]`	`/blog/nextjs`	`"nextjs"`
Catch-all	`[...slug]`	`/docs/setup/installation`	`["setup","installation"]`
Optional Catch-all	`[[...slug]]`	`/docs` or `/docs/setup`	`undefined` or `["setup"]`

✅ When to use what:

- `[slug]` → One dynamic segment (like `/blog/:slug`)
- `[...slug]` → Multiple unknown segments
- `[[...slug]]` → Multiple unknown segments, *plus the base path itself*

Would you like me to continue next with
👉 **"Rendering Modes in Next.js (CSR, SSR, SSG, ISR)"** — how these routes get rendered and how data is fetched?

📘 Learn difference between:

pages/ (legacy routing)

app/ (new app router with React Server Components)

Perfect! Let's clearly compare **`pages/` (legacy) vs `app/` (new App Router)** in Next.js ⚡

📘 `pages/` vs `app/` in Next.js

Feature	`pages/` (Legacy)	`app/` (New App Router)
Routing	File-based routing with **`pages/*.js`**	File-based routing with **`app/*/page.js`**
Rendering	Pages are **Client Components by default**	Pages are **Server Components by default** (faster, SEO-friendly)
Data Fetching	**`getStaticProps`**, **`getServerSideProps`**, **`getStaticPaths`**	Direct **async/await** inside components; no need for special functions
Layouts	No built-in layout nesting; shared layout must be added manually	**Nested layouts** using **`layout.js/ts`** for route-specific layouts
Dynamic Routes	**`[slug].js`** and **`[...slug].js`**	**`[slug]/page.js`** and **`[...slug]/page.js`** (similar, but with params object)
Server Components	Not supported natively	Fully supported (fetch data on server, render HTML, send to client)
Error/Loading States	No built-in error/loading per route; must handle manually	**`error.js`**, **`loading.js`**, **`not-found.js`** per route

Feature	pages/ (Legacy)	app/ (New App Router)
Colocation	Page, data fetching, and layout are separate	**All route-specific files live together** (page, layout, loading, error)
API Routes	`pages/api/*`	`app/*/route.js` (can colocate API endpoints per route)
Bundle Splitting	Automatic per page	Automatic per component + route segment

Key Differences Explained

1. **Server Components by default**
    - `app/` renders components on the server first → smaller JS sent to browser → faster and SEO-friendly.
2. **Nested Layouts**
    - `app/` allows layouts to wrap **only specific routes** or **nested routes**, no extra logic needed.
3. **Better Data Fetching**
    - No need for `getServerSideProps` or `getStaticProps`; just use `async` components.
4. **Colocated files**
    - In `app/`, a route folder can have its `page.js`, `layout.js`, `loading.js`, and `error.js` together.
    - Makes the project **more organized**.
5. **API routes**
    - In `pages/`, all API routes live under `pages/api`.
    - In `app/`, API routes can live alongside the route folder (`route.js`) → easier to maintain.

Example: Blog Route Comparison

Pages Router (`pages/blog/[slug].js`)

```
jsx

export async function getServerSideProps({ params }) {
  const post = await fetchPost(params.slug);
  return { props: { post } };
}

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

App Router (`app/blog/[slug]/page.js`)

```
jsx

async function getPost(slug) {
  const res = await fetch(`https://api.com/posts/${slug}`);
  return res.json();
}

export default async function BlogPost({ params }) {
  const post = await getPost(params.slug);
  return <div>{post.title}</div>;
}
```

✅ Notice: **no `getServerSideProps` needed** in App Router. Just plain async code.

🔹 TL;DR

- `pages/` = legacy, client-first, manual layouts, separate data fetching
- `app/` = modern, server-first, nested layouts, async/await data fetching, colocated route files

---

If you want, I can create a **diagram showing `pages/` vs `app/` folder structure** for a visual comparison — it really makes it click.

Do you want me to do that?