

# NEXT.JS

*What is Next.js?*

*Next.js is a React framework for building fast, scalable, and SEO-friendly web applications with features like server-side rendering (SSR) and static site generation (SSG).*

*Why learn Next.js?*

*Next.js simplifies the process of building a web application for production by providing features like:*

- 1. Routing*
- 2. API routes*
- 3. Rendering — client-side and server-side*
- 4. Data fetching*
- 5. Styling*
- 6. Optimization — Images, forms, and scripts*
- 7. Dev and Prod build system*

*Key Features*

- 1. File-based Routing — Pages are created based on the pages/ directory.*
- 2. Optimized Rendering — Supports SSG, SSR, ISR, and CSR for better performance.*
- 3. Data Fetching — Uses `getStaticProps()`, `getServerSideProps()`, and `useEffect()`.*
- 4. API Routes — Backend logic inside `pages/api/` without a separate server.*
- 5. Automatic Code Splitting — Loads only required JavaScript per page.*
- 6. Image Optimization — Uses `next/image` for better performance.*

## Create first Next.js App

To create a new Next.js project, you can use the following command, replacing with the name of your project:

```
npx create-next-app@latest <app-name>
```

The create-next-app command will prompt you to configure your project. You can choose whether to use TypeScript, ESLint, or Tailwind CSS. Follow the prompts and select the options that suit your project requirements.

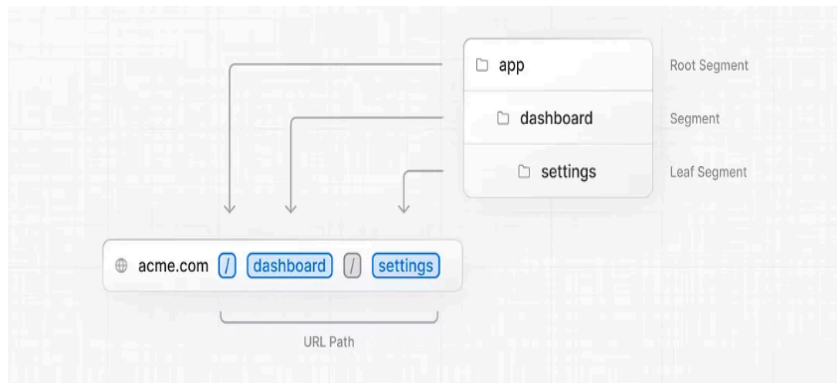
```
Need to install the following packages:
create-next-app@14.0.3
√ Would you like to use TypeScript? ... No / Yes
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like to use `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to customize the default import alias (@/*)? ... No / Yes
√ What import alias would you like configured? ... @/*
Creating a new Next.js app in C:\Users\rusha\OneDrive\Desktop\Next\demo.
```

To start the development server and see your new Next.js application in action, run the following command:

```
npm run dev
```

## Routing

Next.js introduced a new App Router built on React Server Components. It has a file-system-based routing mechanism. Files and folders in your codebase define URL paths the user can access in the browser. A special `page.js` file is used to make route segments publicly accessible.



layout	Shared UI for a segment and its children
page	Unique UI of a route and make routes publicly accessible
loading	Loading UI for a segment and its children
not-found	Not found UI for a segment and its children
error	Error UI for a segment and its children
global-error	Global Error UI
route	Server-side API endpoint
template	Specialized re-rendered Layout UI
default	Fallback UI for <b>Parallel Routes</b>

Now we will take some scenarios to learn more about routing in Next.js

Scenario 1: When a user visits the root directory i.e., `localhost:3000`, show him the home page.

Create a `page.tsx` file in the `src/app` directory.

```
//src/app/page.tsx

export default function Home() {
  return <h1>Home Page</h1>
}
```

Scenario 2: When a user visits localhost:3000/about, show him the about page.

Create a page.tsx file in src/app/about directory.

```
//src/app/about/page.tsx

export default function About() {
  return <h1>About Page</h1>
}
```

Scenario 3: Dynamic Routes-

It refers to the capability of generating pages dynamically based on the parameters provided in the URL. Instead of creating separate static pages for each possible route, you can use dynamic routing to handle patterns or parameters in the URL, making your application more flexible and scalable.

```
// src/app/products/[productId]/page.tsx

export default function ProductDetails({ params } : {
  params: { productId: string };
}) {
  return(
    <>
      <h1>Details about product {params.productId}</h1>
    </>
  )
}
```

← → ↺ Ⓢ localhost:3000/products/2

## Details about product 2

## Scenario 4: Nested Dynamic Routes-

It involves combining the concepts of nested routes and dynamic routes to create a more complex and flexible routing structure.

```
//src/app/products/[productId]/reviews/[reviewId]/page.tsx

import React from "react";

export default function ReviewDetail({
  params,
}): {
  params: {
    productId: string;
    reviewId: string;
  };
} {
  return (
    <>
      <h1>
        Review {params.reviewId} for product {params.productId}
      </h1>
    </>
  );
}
```



```
app
├── products
│   ├── [productId]
│   │   ├── reviews
│   │   │   ├── [reviewId]
│   │   │   │   └── page.tsx
│   │   └── page.tsx
└── page.tsx
layout.tsx
page.tsx
```



localhost:3000/products/2/reviews/2

## Review 2 for product 2

404 page-

The not-found file is used to render UI when the notFound function is thrown within a route segment.

← → ↻ ⓘ localhost:3000/docsd

## Page not found

Could not find requested resource

```
//src/app/not-found.tsx

export default function NotFound() {
  return(
    <>
      <h2>Page not found</h2>
      <p>Could not find requested resource</p>
    </>
  )
}
```

### *Private Folders:*

*Excluded from routing – Folders prefixed with an underscore*

*(\_folderName) are ignored by Next.js routing.*

*Used for organizing internal files, separating UI logic from routing, and avoiding naming conflicts.*

### *Key Uses:*

*1. Keeps internal files separate from route-based files.*

*2. Improves project structure and maintainability.*

*3. Prevents unintended routing conflicts.*

## Including Underscores in URLs

Use %5F instead of \_ (e.g., %5FfolderName) to include underscores in URL segments.



Private folders help maintain clean and organized Next.js projects!

```
//src/app/_lib/page.tsx
```

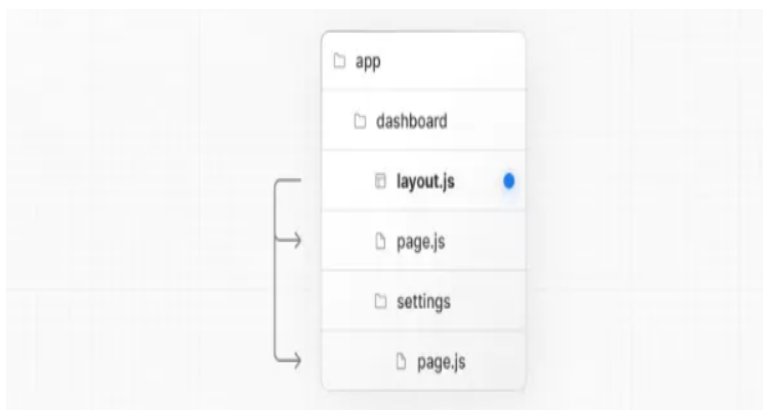
```
export default function PrivateRoute() {  
  return <h1>You can't view this in the browser</h1>  
}
```

## Layouts-

A layout is a UI that is shared between multiple pages in the app, like the header and footer.

We can define a layout by default exporting a React component from a layout.js or layout.tsx file. The component should accept a children props that will be populated with a child page during rendering.

For example, the layout will be shared with the /dashboard and /dashboard/settings pages:



## Root Layout-

The root layout is applied to all routes.

```
export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="en">
      <body>
        <header>
          <p>Header</p>
        </header>

        {childer}

        <footer>
          <p>Footer</p>
        </footer>
      </body>
    </html>
  )
}
```

## Nested layout-

Layouts defined inside a folder (e.g. `app/dashboard/layout.js`) apply to specific route segments and render when those segments are active.

## Metadata API-

What is Metadata in Next.js?

Metadata improves SEO, social sharing, and search engine visibility by defining titles, descriptions, and Open Graph tags for pages.

## How to Use Metadata API?

1. Static Metadata (Defined Directly in a Page File)

tsx

```
export const metadata = {  
  title: "My Next.js Page",  
  description: "This is an example of Next.js metadata.",  
};
```

---

## 2. Dynamic Metadata (Using generateMetadata() for Dynamic Data)

---

tsx

```
export async function generateMetadata() {  
  return {  
    title: "Dynamic Page",  
    description: "This metadata is generated dynamically.",  
  };  
}
```

---

### Supported Metadata Fields-

---

1. title – Sets the page title.
  2. description – Adds a meta description.
  3. keywords – Defines SEO keywords.
  4. openGraph – Adds Open Graph tags for social media sharing.
  5. twitter – Sets Twitter-specific metadata.
  6. robots – Controls search engine indexing (index, follow).
-

## Benefits-

1. Boosts SEO & visibility
2. Enhances social media previews
3. Provides better indexing & ranking

## Navigation-

### Client-Side Navigation with <Link>

Next.js provides the component for fast, client-side navigation between pages. It extends the tag <a> but prefetches links for better performance.

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

const router = useRouter();

const handleClick = () => {
  console.log("Placing order");
  router.push("/");
};

return (
  <button onClick={handleClick}>Place Order</button>
)
```

## Key Benefits

1. Faster navigation with prefetching.
2. No full page reloads, improving UX.
3. Built-in support for dynamic routes.

## Templates-

### What are Templates?

Similar to layouts but do not preserve state between navigations.

Recreates DOM elements and re-synchronizes effects on route change.

### How to Define a Template?

Create a `template.js` or `template.tsx` file and export a React component:

```
tsx

export default function Template({ children }: { children: React.ReactNode }) {
  return <div>{children}</div>;
}
```

### Key Differences from Layouts

1. New component instance on each navigation.
2. Does not retain state or effects.
3. Useful for isolating page transitions.

## loading.tsx-

### What is loading.tsx?

Used to show loading states while a route segment's content is being fetched.

Improves user experience by providing instant feedback during navigation.

## Key Benefits-

1. Instant feedback for users.
2. Enhances perceived performance.
3. Can include spinners or skeleton loaders.



## error.tsx-

Create error UI tailored to specific segments using the file-system hierarchy to adjust granularity

Isolate errors to affected segments while keeping the rest of the application functional.

Add functionality to attempt to recover from an error without a full page reload.

```

'use client' // Error components must be Client Components

import { useEffect } from 'react'

export default function Error({
  error,
  reset,
}): {
  error: Error & { digest?: string }
  reset: () => void
} {
  useEffect(() => {
    // Log the error to an error reporting service
    console.error(error)
  }, [error])

  return (
    <div>
      <h2>Something went wrong!</h2>
      <button
        onClick={
          // Attempt to recover by trying to re-render the segment
          () => reset()
        }
      >
        Try again
      </button>
    </div>
  )
}

```

## Handling errors in nested routes-

1. An `error.tsx` file will cater to errors for all its nested child segments
2. By positioning `error.tsx` files at different levels in the nested folders of a route, you can achieve a more granular level of error handling

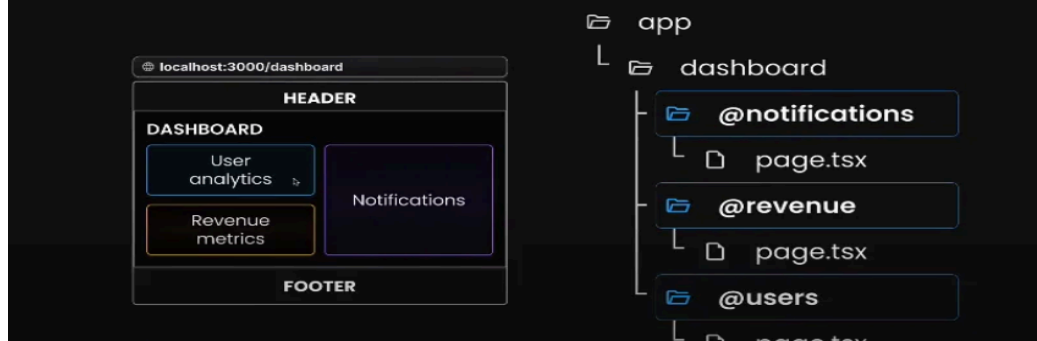
## Parallel Routes-

Parallel routes are an advanced routing mechanism that allows for the simultaneous rendering of multiple pages within the same layout.

Parallel routes in Next.js are defined using a feature known as slots. Slots help structure our content in a modular fashion.

To define a slot, we use the '@folder' naming convention. Each slot is then passed as a prop to its corresponding 'layout.tsx' file.

## Parallel Routes contd.



*For example, considering a dashboard, you can use parallel routes to simultaneously render the users, Revenue and notifications pages:*

```
// dashboard/layout.tsx

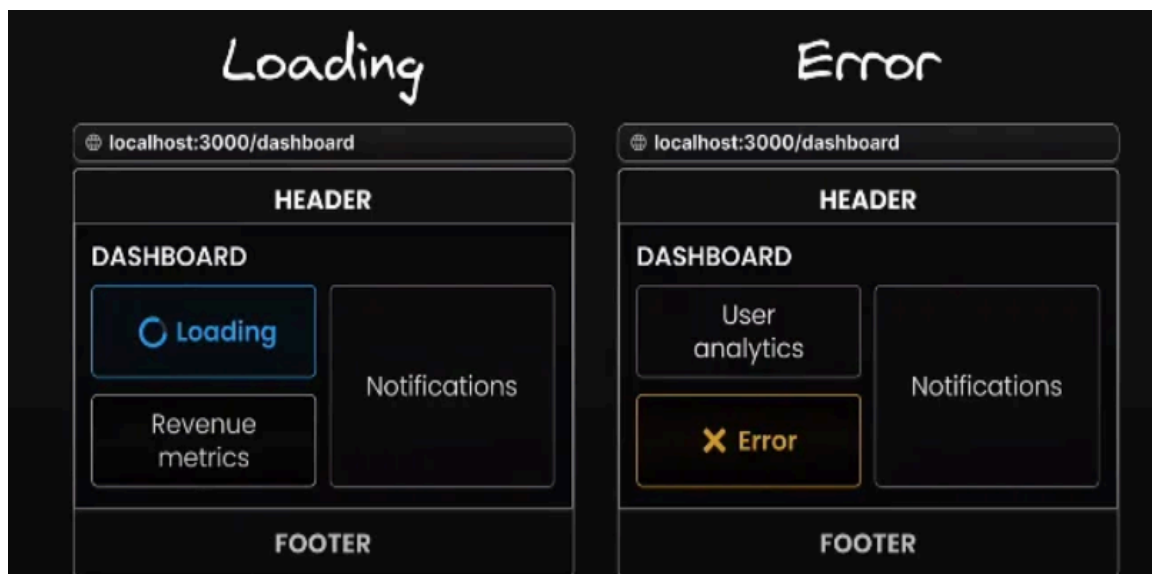
export default function DashboardLayout({
  children,
  users,
  revenue,
  notifications }: {
  children: React.ReactNode;
  users: React.ReactNode;
  revenue: React.ReactNode;
  notifications: React.ReactNode;
}) {
  return (<>
    <div>{children}</div>
    <div>{users}</div>
    <div>{revenue}</div>
    <div>{notifications}</div>
  </>);
}
```

*A benefit of parallel routes is their ability to split a single layout into various slots, making the code more manageable.*

## Independent route handling-

1. Each slot of your layout, such as user analytics or revenue metrics, can have its own loading and error states.

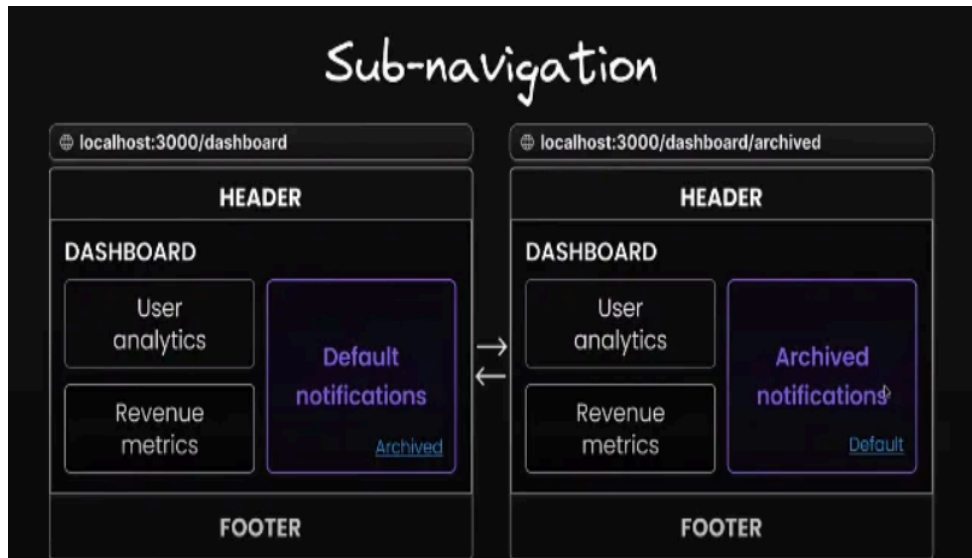
2. This granular control is particularly beneficial in scenarios where different sections of the page load at varying speeds or encounter unique errors



## Sub-Navigation in Routes-

Each section of a complex app (e.g., a dashboard) can have its own navigation and state management.

Useful for managing independent sections within the same app.



### Example: Sub-Navigation in a Dashboard

#### 1. Notifications Page (`/dashboard/@notifications/page.tsx`)

```
tsx

import Link from "next/link";

export default function Notifications() {
  return (
    <div>
      <h2>Notifications</h2>
      <Link href="/dashboard/@notifications/archived">Archived</Link>
    </div>
  );
}
```

## 2. Archived Notifications ( /dashboard/@notifications/archived/page.tsx )

tsx

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

export default function ArchivedNotifications() {
  return (
    <div>
      <h2>Archived Notifications</h2>
      <Link href="/dashboard/">Default</Link>
    </div>
  );
}
```

### Key Benefits-

1. Independent navigation for different sections.
2. Better organization for complex apps.
3. Preserves state within each section.