**Next.js**

1. **Installation**
2. *npx create-next-app@latest*
3. **Home Directory**
4. app route:
5. *src/app/page.jsx = / home page*
6. *src/app/layout.jsx = glob**al layout*
7. page route:
8. *src/pages/index.jsx = / home page*
9. *src/pages/\_app.jsx = global layout*
10. **public Folder**
11. in public folder store global assets.
12. Access public folder assets: */assest.name*
13. **Project Structure**
14. top-level folder:

|  |  |
| --- | --- |
| app | For App Router |
| pages | For Pages Router |
| public | Static assets to be served |
| src | Optional application source folder |

1. Top-level files:

|  |  |
| --- | --- |
| next.config.js |  |
| package.json |  |
| instrumentation.ts |  |
| middleware.ts |  |
| .env |  |
| .env.local |  |
| .env.production |  |
| .env.development |  |
| .eslintrc.json |  |
| .gitignore |  |
| next-env.d.ts |  |
| tsconfig.json |  |
| jsconfig.json |  |

1. **app Routing Conventions**
2. routing files:

|  |  |  |
| --- | --- | --- |
| layout | .js .jsx .tsx | Layout |
| page | .js .jsx .tsx | Page |
| loading | .js .jsx .tsx | Loading UI |
| not-found | .js .jsx .tsx | Not Found UI |
| error | .js .jsx .tsx | Error UI |
| global-error | .js .jsx .tsx | Global error UI |
| route | .js .ts | Server Side API Endpoint |
| template | .js .jsx .tsx | Re-rendered Layout |
| default | .js .jsx .tsx | Parrallel route fallback page |

1. Nested routes:

|  |  |
| --- | --- |
| folder | Route Segment |
| folder/folder | Nested route segmet |

1. Dynamic nested rotes:

|  |  |
| --- | --- |
| [folder] | Dynamic route segment |
| [...folder] | Catch-all route segment |
| [[...folder]] | Optional-catch-all-route-segment |

1. Route Groups and Private Folder:

|  |  |
| --- | --- |
| (folder) | Group routes without affecting routing |
| \_folder | Opt folder and all child segments out of routing |

1. Parallel and Intercepted Routes:

|  |  |
| --- | --- |
| @folder | Named slot |
| (.)folder | Intercept same level |
| (..)folder | Intercept one level above |
| (...)(...)folder | Intercept two level above |
| (..)folder | Intercept from root |

1. **Metadata File Conventions**
2. App Icons:

|  |  |  |
| --- | --- | --- |
| favicon | .ico | Favicon file |
| icon | .ico .jpg .jpeg .png .svg | App Icon File |
| icon | .js .ts .tsx | Generated App Icon |
| apple-icon | .jpg .jpeg, .png | Apple App Icon File |
| apple-icon | .js .ts .tsx | Generated Apple App Icon |

1. Open Graph and Twitter Images:

|  |  |  |
| --- | --- | --- |
| opengraph-image | .jpg .jpeg .png .gif | Open graph image file |
| opengraph-image | .js .ts .tsx | Generated open graph image |
| twitter-image | .jpg .jpeg .png .gif | Twitter image file |
| twitter-image | .js .ts .tsx | Generated twitter image |

1. SEO:

|  |  |  |
| --- | --- | --- |
| sitemap | .xml | Sitemap file |
| sitemap | .js .ts | Generated site map |
| robots | .txt | Robots File |
| robots | .js .ts | Generated robots file |

1. **Building Your Application**
2. Routing
3. Rendering
4. Data Fetching
5. Styling
6. Catching
7. Optimizing
8. Configuring
9. Deploying
10. Upgrading
11. **Routing**
12. terminology

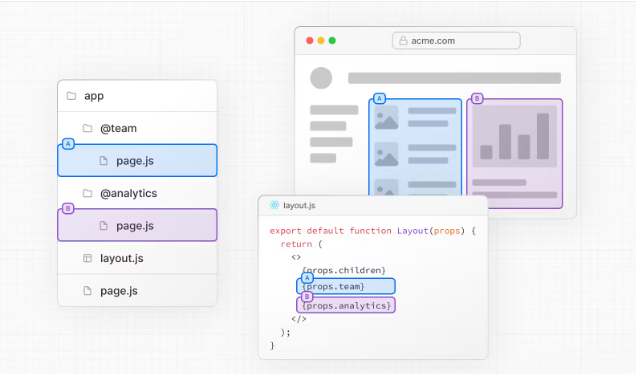
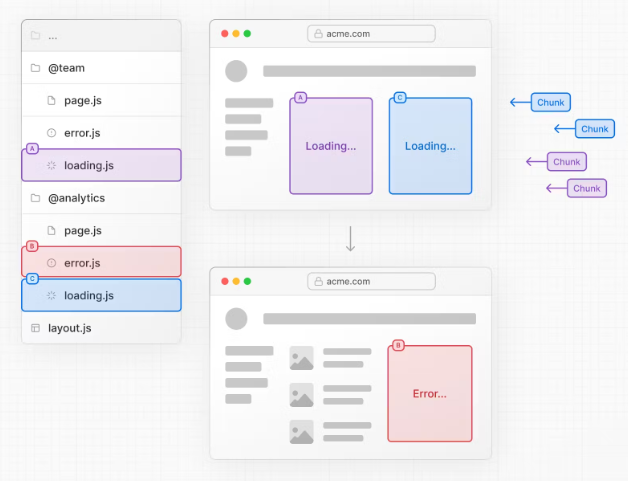
|  |  |
| --- | --- |
| Tree | A convention for visualizing a hierarchical structure. For example, a component tree with parent and children component, a folder structure etc. |
| Subtree | Part of a tree, starting at a new root (first) and ending at the leaves (last). |
| Root | The first node in a tree or subtree, such as root layout. |
| Leaf | Nodes in a subtree that have no children, such as the last segment in a URL Path. |
| URL Segment | Part of the URL path delimited by slashes. |
| URL Path | Part of the URL that comes after the domain (composed of segments) |

1. The app Router
2. app router built on server components
3. the app router works in a new directory named app.
4. By default, components inside app are React Server Components.
5. The app router takes priority over the pages router.
6. Components:
7. react server components
8. react client components
9. Roles of Folders and Files:
10. next.js uses file system based router
11. Folders: url path
12. Files: are used to create UI that is shown for a route segments.
13. Route Segments:
14. 📁dashboard = /dashboard
15. 📁dashboard📁setting = /*dashboard/setting*
16. *each folder in a route represents a route segment.*
17. *Nested Routes:*
18. *to create a nested route, you can nest folders inside each other.*
19. 📁dashboard📁setting = /*dashboard/setting*
20. */ = root segments*
21. *dashboard = segment*
22. *setting = leaf segment*
23. Reused Files In Nested Segments:

|  |  |
| --- | --- |
| layout.jsx | Shared UI for a segment and its children |
| error.jsx | Error UI for a segment and its children |
| loading.jsx | Loading UI for a segment and its children |
| page.jsx | Unique UI of a route and make routes publicly accessible |
| route.jsx | Server-side API endpoint |

1. Colocation:
2. you can write your own files or folder for compoenets,styles etc.
3. Because, only return contents of page.js and route.js are publicly accessible.
4. **Defining Routes**
5. 📁contact = /*contact*/
6. 📁contact📁users = /*contact*/users/
7. **Pages and Layouts and Templates**
8. **pages:**
9. 📁login:page.js = /*login*/page.js
10. 📁login📁user:page.js = /login/user/page.js
11. page.js file to make the route publicly accessible.
12. **Layouts:**
13. root layouts(required):
14. 📁app:layout.jsx = /app/layout.jsx
15. The root layout must define <html> and <body> tags.
16. You can use route groups to create multiple root layouts.
17. The app directory must include a root layout.
18. You should not manually add <head> tags such as <title> and <meta> to root layouts. Instead, you should use the Metadata API which automatically handles advanced requirements such as streaming and de-duplicating <head> elements.
19. Nesting layouts:
20. 📁app📁about:layout.jsx = /app/about/layout.jsx
21. it’s can’t define html and body tag.
22. Templates:
23. templates is like layouts.
24. But some reason has….
25. **Linking and Navigation**
26. **using <Link> component**
27. <Link href=“/dashboard”>Dashboard</Link>
28. it’s use client-side navigation
29. dynamic Link:
30. <Link href={`/blog/${post.slug}`}>{post.title}</Link>
31. active Link:
32. const pathname = usePathname();
33. Link className={pathname==='/about'?'active':'inactive'} href="/about">About</Link>
34. “use client”;
35. use server and client both components.
36. scrolling to an id:
37. <Link href="/dashboard#settings">Settings</Link>
38. **using useRouter() Hook**
39. const router = useRouter();
40. router.push(“/about”);
41. router.back();
42. router.forward();
43. The useRouter hook allows you to programmatically change routes.
44. This hook can only be used inside Client Components and is imported from next/navigation.
45. **Route Groups**
46. 📁(shop)📁products:page.js = *product*/page.js
47. A route group can be created by wrapping a folder's name in parenthesis: (folderName)
48. To organize routes without affecting the URL, create a group to keep related routes together. The folders in parenthesis will be omitted from the URL.
49. Different group layout:
50. 📁(shop):layout.jsx📁products:page.jsx;
51. you can create a different layout for each group by adding a layout.js file inside their folders.
52. Creating multiple root layouts:
53. To create multiple root layouts, remove the top-level layout.js file, and add a layout.js file inside each route groups.
54. **Dynamic Routes**
55. 📁post📁[id]📁[title]:page.jsx = /*post*/23/how-make-a-coffee
56. A Dynamic Segment can be created by wrapping a folder's name in square brackets
57. catch dynamic segments:
58. Dynamic Segments are passed as the params prop to layout, page, route, and generateMetadata functions.
59. ({params});params.id;params.title
60. catch all dynamic segments:
61. 📁post📁[...slug]:page.jsx = post/23/how-to-make-a-cup-tea
62. or
63. 📁post📁[[...slug]]:page.jsx = post/23/how-to-make-a-cup-tea
64. ({params}); params.slug[0];params.slug[1]
65. **Loading UI and Streaming**
66. **Instant Loading States**
67. just add loading.jsx file and her skeleton o spin within page.js file.
68. **Streaming With Suspense**
69. <Suspense fallback={<p>loading feed...</P>}><PostFeed/></Suspense>

1. **Error Handling**
2. create a error.jsx file in any segment.
3. ({error,reset})
4. **handling error in layouts**
5. just declare into any segment error.js file.
6. **Handling error in root layouts**
7. just declare into root segment error.js file.
8. **Parallel Routes**



1. Parallel Routing allows you to define independent error and loading states for each route as they're being streamed in independently.
2. **Intercepting Routes**
3. it’s a custom