**NEXT JS**

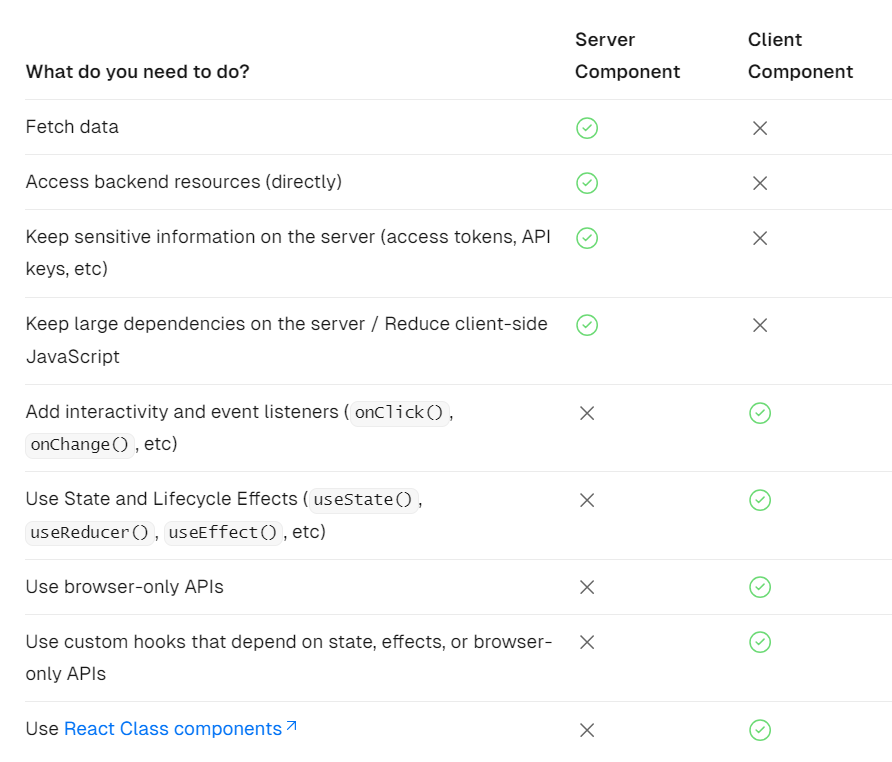
1. **Defination :** Next.js is an open-source web development framework built on top of React, focusing on server-side rendering and static website generation.Created by Vercel (formerly known as Zeit), it provides a set of tools and configurations aimed at improving web application performance, SEO, and developer productivity.
2. **Installation :** npx create-next-app@latest
3. **Components :**
4. **Client Components :**

* Render on the client (browser).
* Client Components in Next.js enable you to write interactive UI that can be rendered on the client at request time.
* **"use client"** directive is used to make a default server component to the client component.
* All the UI related code and events should be written here.

1. **Server Components :**

* Render on the server.
* Support data fetching on the server.
* Improve performance, security, and caching.
* Useful for server-rendered applications and static site generators.
* Default in Next.js 13.
* Server Components are rendered on the server and can be used to optimize the rendering process and improve user experience

**Note :** We can use the both components together without any problem.

****

**Example : Server Component :**

export default function Home() {

  return (

    <div>

      <h1>Server Component</h1>

**<button onClick={()=>alert("Hello")} type="button">Click Me</button> 🡨 Error as it is event**

    </div>

  )

}

**Example : Client Component :**

**"use client"**

export default function Home() {

  return (

    <div>

      <h1>Server Component</h1>

**<button onClick={()=>alert("Hello")} type="button"Click Me</button>**

    </div>

  )

}

**Note :** “use client” 🡪 The "use client" directive in Next.js marks components to be rendered on the client side, enabling interactivity and access to browser features.

1. **Routing :** Routing in Next.js is a way to manage the URLs and pages of a web application.

Next.js uses a **file-system-based router**, where folders represent route segments that map to URL segments.

Each folder represents a route segment that maps to a URL segment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Types of Route   |  |  | | --- | --- | | 1. App Route | 1. Page Route | | The App Router is a newer feature introduced in Next.js version 13. It is designed to address some limitations of the Pages Directory approach.  The App Router still uses folder directories for routing but with a slightly different convention. It allows for shared layouts, nested routing, loading states, error handling, and more. | The Pages Router is the traditional way of creating routes in Next.js. It uses a file-system-based router where each file automatically becomes a route based on its name.  Routes are defined by creating React components in the pages directory with supported file extensions like .js, .jsx, .ts, or .tsx.  How To Create :  Src/folder name/page.jsx  How To Access On Browser  <http://localhost:3000/folder> name  file should be always named as page.jsx in both the route app , page .  Jo route rahega us naam se folder banao inside src and the add page.js within that folder. | |

**Layout.jsx :** In Next.js, a layout is a user interface (UI) component that is shared between multiple routes. Layouts in Next.js serve to maintain state, interactivity, and prevent unnecessary re-renders when navigating between different pages. Note : Share Common areas or components like navbar , footer among multiple pages.

* A layout is UI that is **shared** between multiple routes. On navigation, layouts preserve state, remain interactive, and do not re-render. Layouts can also be [nested](https://nextjs.org/docs/app/building-your-application/routing/pages-and-layouts#nesting-layouts).
* You can define a layout by default exporting a React component from a layout.js file. The component should accept a children prop that will be populated with a child layout (if it exists) or a page during rendering.

|  |
| --- |
| Layout.jsx  import React from 'react';  // IMPORT COMPONENTS  import Navbar from '../components/Navbar';  export default function layout({ children }) {      return (          <div className="main">              <Navbar />              {children}          </div>      )  } |

**Page.jsx :** A page.jsx file in Next.js is a React component that is automatically available as a route based on its file name.

1. **Linking and Navigating :**

* **Linking :** using anchor tag to navigate between the page is called linking.

**<Link>** is a built-in component that extends the HTML <a> tag to provide [prefetching](https://nextjs.org/docs/app/building-your-application/routing/linking-and-navigating#2-prefetching) and client-side navigation between routes. It is the primary and recommended way to navigate between routes in Next.js.

Example : <Link href="/routes/auth/signup">Sign Up</Link>

* **Navigating :** using button to navigate between the page is called navigating.

[**useRouter() hook**](https://nextjs.org/docs/app/building-your-application/routing/linking-and-navigating#userouter-hook)

The useRouter hook allows you to programmatically change routes from [Client Components](https://nextjs.org/docs/app/building-your-application/rendering/client-components).

Example :

const router = useRouter()

<button type="button" onClick={()=>router.push("/routes/auth/signin”) >

Sign In

</button>

**usePathname() hook**

You can use [usePathname()](https://nextjs.org/docs/app/api-reference/functions/use-pathname) to determine if a link is active.

Example :

Const pathname = usePathname()

Console.log(pathname)

Output : /routes/auth/signup

1. **Nested Routing :** Nested routing in Next.js involves organizing routes in a hierarchical structure, allowing for more intuitive layout and navigation within a web application.

/routes/college

/routes/college/students

/routes/college/teachers

1. **Dynamic Routing :** Dynamic routing in Next.js allows for creating routes based on dynamic data, enabling flexible and customizable URLs.

Dynamic segments are placeholders filled at request time or prerendered at build time. They are denoted by square brackets, like [segmentName], such as [id] or [slug]. These segments can be accessed using useRouter

1. **Catch-all Segments :** Catch-all segments in Next.js are a feature that allows developers to create dynamic routes that match any number of URL segments. […folderName]

**Kabhi Kabhi kya hota hai ki hum log ko post ke andhar bhi sare routes chale if agar waha route nhi hai toh bhi kuch na kuch dikhe but 404 na dikhe for such situation this concept is used.**

**Jaha hume pata na ho ki url mein kitne segment hoge vaha use kiya jata hai isko**

Step : create […foldername] within the parent folder.

<http://localhost:3000/routes/college/teachers/jdjff/fffffffff/ffrrrrrrrrr>

|  |
| --- |
| import React from 'react'  export default function page({ params }) {      console.log(params);      return (          <div>              <h1>Hello i am catch-all segments</h1>          </div>      )  }  **OUTPUT :**  { practicals: [ 'jdjff', 'fffffffff', 'ffrrrrrrrrr', 'ffff' ] } |

1. **404 Page :**
2. **Global Route 404 Page :** In Next.js, a global 404 page is a custom page that is displayed when a user visits a URL that does not have a matching route in your application.this is applicable to all the routes if page not found.

**Syntax :** create **not-found.jsx** file in app folder

1. **Specifc Route 404 Page :** In Next.js, you can create a specific 404 page to handle unmatched routes with custom content.

**Syntax :** create a folder named as […foldername] an add page.jsx file within it .

1. **Middleware :**

Middleware allows you to run code before a request is completed. Then, based on the incoming request, you can modify the response by rewriting, redirecting, modifying the request or response headers, or responding directly.

Middleware runs before cached content and routes are matched.

Step : create a middleware.js file within a src folder.

|  |
| --- |
| import { NextResponse } from 'next/server'  import type { NextRequest } from 'next/server'    // This function can be marked `async` if using `await` inside  export function middleware(request: NextRequest) {  return NextResponse.redirect(new URL('/home', request.url))  }  // See "Matching Paths" below to learn more  export const config = {      // SINGLE MATCH      matcher: "/routes/:path\*"      // MULTIPLE MATCH      matcher : ["/routes/college/:path\*","/routes/blogs/:path\*"]  } |

1. **Rendering :** Rendering in Next.js refers to the process of converting code into user interfaces.

**Types :**

1. **Static Site Generation (SSG) :** HTML is generated at build time and reused for each request. SSG is beneficial for SEO as pre-rendered HTML improves page performance and SEO ranking factors
2. **Server-Side Rendering (SSR) :**  HTML is generated at request time, making it suitable for dynamic pages. SSR is useful when generating pages at build time for all scenarios is not feasible
3. **Client-Side Rendering (CSR) :** In CSR, the browser downloads minimal HTML and JavaScript to render the page. The full page rendering occurs after all JavaScript is downloaded, parsed, and executed. CSR can impact SEO and user experience due to initial loading delays and potential performance issues.

Note :

* **Static rendering** involves generating content at build time or during revalidation, where the result is cached and distributed via a Content Delivery Network (CDN). This pre-rendered content can be quickly accessed by users globally, reducing server load and improving SEO by facilitating easier indexing for search engines
* **Dynamic rendering** generates content on the server for each user at request time. This approach allows for real-time or frequently updated data display, personalized content delivery, and access to request-time information like cookies or URL parameters. Dynamic rendering is ideal for applications with constantly changing data or user-specific content

1. **Fetch Data :**
2. **fetch API data in Client component :**

|  |
| --- |
| "use client" // converting server site component to client site component using this.  import React, { useEffect, useState } from 'react';  export default function page() {      const [products, setProducts] = useState([]);      async function fetchProducts() {          const url = "https://fakestoreapi.com/products";          const request = await fetch(url);          const response = await request.json();          console.log("Client Site Response :", response);          setProducts(response)      }      useEffect(() => {          fetchProducts();      }, [])      return (          <div className='p-5'>              <h1 className='text-xl font-semibold'>Client Site API fetching</h1>              {                  products.length !== 0 &&                  <ul>                      {                          products.map((item, index) => (                              <li key={index}>{item.id} {item.title}</li>                          ))                      }                  </ul>              }          </div>      )  } |

1. **fetch API data in Server component :**

|  |
| --- |
| async function fetchProducts() {      const url = "https://fakestoreapi.com/products";      const request = await fetch(url);      const response = await request.json();      return response;  }  export default async function page() {      let products = await fetchProducts();      console.log("Server Site Response :", products);      return (          <div className='p-5'>              <h1 className='text-xl font-semibold'>Server Site API fetching</h1>              <ul>                  {                      products.map((item, index) => (                          <li key={index}>                              {item.id}                              {item.category}                          </li>                      ))                  }              </ul>          </div>      )  } |

1. **CSS Module :**
2. **Whatever style we implement in global.css it will applicable to the entire app.**
3. **Sometime we have a same class “main” to multiple html element if you want to apply different style to same class but for different element then the css module in introduce.**

Step : create a “filename.module.css” file near a global.css

Step : Import it in other file like this : import mycss from “path to css module file”

Step : apply this mycss as mycss.stylename like : mycss.btn , mycss.main

1. **If you want to create a folder the holds the entire css create that folder with the src directory**

**Create files named as “filename.module.css” an add some styles in it.**

**Now you can import this folder dynamically like this :**

import Style from "@/styles/style1.module.css";

|  |
| --- |
| import React from 'react';  // Import Css Module  import Css from "../../css.module.css";  import Css2 from "../../css1.module.css";  // Import Style Outside The App Folder  import Style from "@/styles/style1.module.css";  export default function page() {    return (      <div>        Blogs route        <button type="button" className="btn bg-blue-500 text-white px-5 py-2">All Posts</button>        <h1 className={Css.btn}>Heading 1</h1>        <h2 className={Style.main}>Heading 2</h2>        <p className={Css2.btn}>All Posts</p>      </div >    )  } |

1. To Implement Id Style use this :

|  |  |
| --- | --- |
| Style.css  #main {      color: blue;  } | React Component  <h1 className={Css.btn} id={Css2.main}>Heading 1</h1> |

1. **Image Optimization in nextJs :**

|  |
| --- |
| import Image from 'next/image'    export default function Page() {  return (  **<Image**  **src="/path to public folder/imagename.png"**  **width={500}**  **height={500}**  **alt="Picture of the author"**  **/>**  )  } |
| Notes : store images to the public folder |

**To Add Image Directly from website then you have to configure the nextjs app like this**

Go to : next.config.js add below code :

|  |
| --- |
| /\*\* @type {import('next').NextConfig} \*/  const nextConfig = {      images:{          domains : ["images.pexels.com"]      }  };  export default nextConfig;  **Implementating :** <Image src={"https://images.pexels.com/photos/2049422/pexels-photo-2049422.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=2"} height={200} width={200} /> |

1. **Fonts Optimization :**

|  |
| --- |
| import { Roboto } from 'next/font/google';  const roboto = Roboto({    weight:"100",    subsets:["latin"],    display:"swap"  })  Implement :          <h1 className={roboto.className} >Layout.jsx NextJs default fonts</h1> |

1. **generateMetadata for Dynamic meta data in next.js :**

|  |
| --- |
| **Just simply add the following lines of code in you main file : note : function name must be generateMetadata only.**  export function generateMetadata() {    return {      title: "This is generate meta data",      description: "signin page meta data"    }  } |

1. **Script component**

* **HTML SCRIPT TAG :** It loads the script to the entire app not to the specifc components.
* **So** Instead of applying script to the entire app we can apply it as per choice in specifc components.

|  |
| --- |
| import Script from 'next/script';  **create myscript or a script in public folder always**  **add the below code in specifc component.**   <Script src='/myscript.js' onLoad={() => {            console.log("Script loaded successfully");          }} /> |

1. **Loader with API data in Next.js 13.4 :**

|  |
| --- |
| "use client"  import React, { useEffect, useState } from 'react'  export default function page() {      const [list, setList] = useState([]);      async function fetchProducts() {          const request = await fetch("https://fakestoreapi.com/products");          const response = await request.json();          console.log(response);          setList(response);      }      useEffect(() => {          fetchProducts();      }, [])      return (          <div>              <ul>                  {                      list.length === 0 ?                          "loading data"                          :                          (                              list.map((item, index) => (                                  <li>                                      <h1>{index} {item.title}</h1>                                  </li>                              ))                          )                  }              </ul>          </div>      )  } |

1. **Static Assets in next.js 13.4**

* **Step-1 : just create a file , or add image to public folder of a nodejs app.**
* **Step-2 : import is like this where you wants : “./filename”**

1. **Types of build :**

* **Development build : you will notice an nextjs folder in your app “npm run dev”.**
* **Production build : run this command “npm run build” then “npm run start”.**

1. **Export Static HTML Page with Build :**

Suppose if our project size is small which have min 5-6 pages so instead of generating whole production build you can generate the static html pages which can be deploy our used any where so please implement the below configs :

1. Create some routes or pages.
2. Add the code to vercel.config file

/\*\* @type {import('next').NextConfig} \*/

const nextConfig = {

    output: "export"

};

export default nextConfig;

1. Run this command : npm run build
2. This will generate both production as well as static html file build
3. **Static Site Generation (SSG) :**

If a page uses **Static Generation**, the page HTML is generated at **build time**. That means in production, the page HTML is generated when you run next build. This HTML will then be reused on each request.

1. **Redirect :**

|  |
| --- |
| Method : 1  import React from 'react';  import {redirect} from "next/navigation";  export default function page() {    redirect("/routes/seller")    return (      <div>        <h1>Export Static HTML Page with Build </h1>        <h1>Products Route</h1>      </div>    )  }  Here there is one issue if someday you delete this page and other pages are related to this page (link) this leads to system issue and SEO related problem so following method 2 is best solution for it.  Note : whenever you access this page you will be redirect to the seller page from products page access. |

|  |
| --- |
| Method 2 :  Next.config() file :  /\*\* @type {import('next').NextConfig} \*/  const nextConfig = {      // output: "export",      redirects: async () => {          return [              {                  source: "/routes/products",                  destination: "/",                  permanent: false              },              {                  source: "/routes/products/:product\_id",                  destination: "/",                  permanent: false              },              {                  source: "/routes/sellers",                  destination: "/",                  permanent: false              }          ]      }  };  export default nextConfig; |

1. **Environment variable :**

|  |
| --- |
| import Navbar from '@/app/components/Navbar'  import React from 'react'  export default function page() {    console.log(process.env.NODE\_ENV);    return (      <div>        <Navbar />        <h1>Seller Route</h1>        {          process.env.NODE\_ENV === "development"            ?            <h1>{">>"} You are in development mode</h1>            :            <h1>{">>"} You are in production mode</h1>        }      </div>    )  } |

Add or create new environment variable :

.env.local file in my-app or root of the project and add

FNAME = "PIYUSH"

LNAME = "Thaware" >> access it like this : process.env.FNAME

1. **API ROUTES :**

|  |
| --- |
| **Create an API** |
| 1. Create an api folder within the app folder. 2. The route of api is based on the file based route system like nextJS.so whatever the name of the folder it will be your route , 3. Create a router.js file within that folder an and the below code :   export async function GET(request) {      return new Response("Hello,Next.js!")  }   1. Access the api route on the browser as : http://localhost:3000/api/routeFolderName |

|  |
| --- |
| **ALTERNATE OF ABOVE CODE**  import { NextResponse } from "next/server";  export async function GET(request) {      return NextResponse.json({          success: true,          fname: "Piyush",          lname: "Thaware"      },          {              status: 200          })  }  Access : http://localhost:3000/api/routeFolderName |

1. **GET METHOD :**

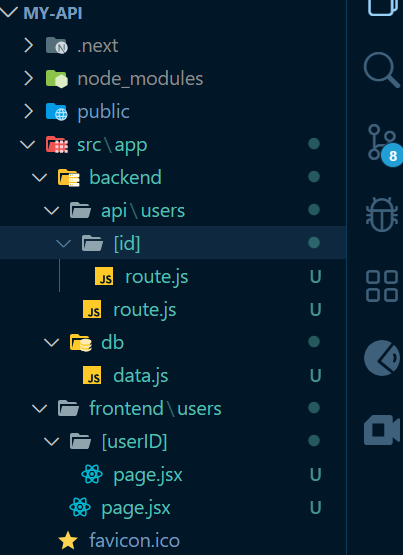
|  |
| --- |
| **ACCESS WITHOUT PARAMETERS LIKE ID :** http://localhost:3000/backend/api/users |
| import { NextResponse } from "next/server";  import { users } from "../../db/data";  export function GET() {      const mydata = users;      return NextResponse.json(mydata, { status: 200 });  } |

|  |
| --- |
| **ACCESS WITH PARAMETERS LIKE ID :** http://localhost:3000/backend/api/users/id |
| **Create an folder :** [id] or [slug] inside the main user route  import { NextResponse } from "next/server";  import { users } from "../../../db/data";  export function GET(request, content) {      const mydata = users;      const ID = content.params.id;      const singleUserFilter = mydata.find((item) => item.id == ID);      if (typeof singleUserFilter === "undefined") {          // Empty          return NextResponse.json({ message: "unable to found the data with this ID." }, { status: 404 });      }      else {          // Found          return NextResponse.json({ message: singleUserFilter }, { status: 200 });      }  }  **Example :** http://localhost:3000/backend/api/users/3 |

**Connect Above Backend With Frontend :**

|  |
| --- |
| 1. **Fetch All Users List :** http://localhost:3000/frontend/users |
| import React from 'react';  import { Poppins } from 'next/font/google';  const poppins = Poppins({      subsets: ["latin"],      weight: "500"  });  import { FiExternalLink } from "react-icons/fi";  import Link from 'next/link';  **// CONNECT BACKEND WOTH FRONTEND >> Server Site Rendering**  **async function getUsers() {**  **const request = await fetch("http://localhost:3000/backend/api/users");**  **const response = await request.json();**  **return response;**  **}**  export default **async** function page() {  **const data = await getUsers();**  **console.log(data);**      return (          <div className={`${poppins.className} p-10`}>              <div className="flex flex-nowrap items-center gap-2">                  <h1 className='text-xl'>Fetch Users List : </h1>                  <h1 className='px-4 py-1 tracking-[1px] text-sm rounded-md bg-green-500 text-white'>GET</h1>              </div>              <table className='my-5 border-[1.5px] border-slate-900'>                  <thead>                      <tr className='bg-slate-700 text-white'>                          <td className='px-10 py-2 border-2 border-slate-900'>Id</td>                          <td className='px-10 py-2 border-2 border-slate-900'>Name</td>                          <td className='px-10 py-2 border-2 border-slate-900'>Email</td>                          <td className='px-10 py-2 border-2 border-slate-900'>Age</td>                          <td className='px-10 py-2 border-2 border-slate-900'>Preview</td>                      </tr>                  </thead>                  <tbody>  **{**  **data.length !== 0**  **&&**  **(**  **data.map((item, index) => (**  **<tr key={index} className=''>**  **<td className='px-10 py-2 border-2 border-slate-900'>{item.id}</td>**  **<td className='px-10 py-2 border-2 border-slate-900'>{item.name}</td>**  **<td className='px-10 py-2 border-2 border-slate-900'>{item.email}</td>**  **<td className='px-10 py-2 border-2 border-slate-900'>{item.age}</td>**  **<td className='px-10 py-2 border-2 border-slate-900 cursor-pointer'><Link href={`/frontend/users/${item.id}`} className='inline-flex text-sm gap-1 items-center' >Checkout <FiExternalLink /></Link></td>**  **</tr>**  **))**  **)**  **}**                  </tbody>              </table>          </div>      )  } |

|  |
| --- |
| 1. **Fetch Single User Only Based On The User ID :** http://localhost:3000/frontend/users/id |
| import React from 'react';  import { Poppins } from 'next/font/google';  const poppins = Poppins({      subsets: ["latin"],      weight: "500"  });  **// CONNECT BACKEND WOTH FRONTEND**  **async function getUsers(id) {**  **const request = await fetch(`http://localhost:3000/backend/api/users/${id}`);**  **const response = await request.json();**  **return response;**  **}**  export default **async** function page**({ params })** {  **const ID = params.userID;**  **const data = await getUsers(ID);**      console.log(data);      return (          <div className={`${poppins.className} p-10`}>              <div className="flex flex-nowrap items-center gap-2">                  <h1 className='text-xl'>Fetch User Based On ID : </h1>                  <h1 className='px-4 py-1 tracking-[1px] text-sm rounded-md bg-green-500 text-white'>GET</h1>              </div>              <table className='my-5 border-[1.5px] border-slate-900'>                  <thead>                      <tr className='bg-slate-700 text-white'>                          <td className='px-10 py-2 border-2 border-slate-900'>Id</td>                          <td className='px-10 py-2 border-2 border-slate-900'>Name</td>                          <td className='px-10 py-2 border-2 border-slate-900'>Email</td>                          <td className='px-10 py-2 border-2 border-slate-900'>Age</td>                      </tr>                  </thead>                  <tbody>  **<tr className=''>**  **<td className='px-10 py-2 border-2 border-slate-900'>{data.message.id}</td>**  **<td className='px-10 py-2 border-2 border-slate-900'>{data.message.name}</td>**  **<td className='px-10 py-2 border-2 border-slate-900'>{data.message.email}</td>**  **<td className='px-10 py-2 border-2 border-slate-900'>{data.message.age}</td>**  **</tr>**                  </tbody>              </table>          </div>      )  } |

****

1. **POST METHOD :**

|  |
| --- |
| **Backend** |
| import { NextResponse } from "next/server";  import { users } from "../../db/data";  export function GET() {      const mydata = users;      return NextResponse.json(mydata, { status: 200 });  }  export async function POST(request, respose) {      // GETTING DATA FROM FRONTEND      let payload = await request.json();      console.log(payload);      const mydata = users;      mydata.push(payload);      return NextResponse.json({ success: true, message: mydata }, { status: 201 });  } |

|  |
| --- |
| **Frontend : frontend > users > addUser > page.jsx** |
| "use client"  import React, { useState } from 'react';  import { Poppins } from 'next/font/google';  const poppins = Poppins({      subsets: ["latin"],      weight: "500"  });  export default function page() {      const [name, setName] = useState("");      const [email, setEmail] = useState("");      const [age, setAge] = useState("");      const [message, setMessage] = useState('');      async function handleSubmit(event) {          event.preventDefault();          const request = await fetch("http://localhost:3000/backend/api/users", {              method: "POST",              headers: {                  "Content-Type": "application/json"              },              body: JSON.stringify({ name: name, age: age, email: email })          });          const response = await request.json();          console.log(response);          setMessage("Data Inserted Successfully");      }      return (          <div className={`${poppins.className} p-10`}>              <div className="flex flex-nowrap items-center gap-2">                  <h1 className='text-xl'>Add New User : </h1>                  <h1 className='px-4 py-1 tracking-[1px] text-sm rounded-md bg-orange-500 text-white'>POST</h1>              </div>              <div className="my-10 border-[1.5px] border-slate-500 bg-slate-100 rounded-md p-5 max-w-full sm:max-w-[60%] md:max-w-[50%] lg:max-w-[30%]">                  <form action="" onSubmit={handleSubmit}>                      <input required value={name} onChange={(e) => setName(e.target.value)} type="text" placeholder='Enter Name' className='border-[1.6px] w-full rounded-md my-3 p-3' />                      <br />                      <input required value={email} onChange={(e) => setEmail(e.target.value)} type="email" placeholder='Enter Email' className='border-[1.6px] w-full rounded-md my-3 p-3' />                      <br />                      <input required value={age} onChange={(e) => setAge(e.target.value)} type="number" placeholder='Enter Age' className='border-[1.6px] w-full rounded-md my-3 p-3' />                      <button type="submit" className='px-3 py-2 bg-slate-900 text-white rounded-md text-sm' >Submit</button>                  </form>              </div>              <div className="">                  Message :                  {message !== ""                      &&                      <p>{message}</p>                  }              </div>          </div>      } |

1. **PUT METHOD :**

**NOTE :** We have to create a put method always in a nested folder like [id] or [slug] as we are getting some parameter from url.

|  |
| --- |
| **BACKEND** |
| import { NextResponse } from "next/server";  import { users } from "../../../db/data";  export function GET(request, content) {      const mydata = users;      const ID = content.params.id;      const singleUserFilter = mydata.find((item) => item.id == ID);      if (typeof singleUserFilter === "undefined") {          // Empty          return NextResponse.json({ message: "unable to found the data with this ID." }, { status: 404 });      }      else {          // Found          return NextResponse.json({ message: singleUserFilter }, { status: 200 });      }  }  // ^ NOTE : We have to create a put method always in a nested folder like [id] or [slug] as we are getting some parameter from url so  export async function PUT(request, content) {      const payload = await request.json();      const userID = content.params.id;      console.log(userID);      return NextResponse.json({ success: true, message: payload }, { status: 200 });  } |

|  |
| --- |
| **FRONTEND :** frontend>users>userID>update (as update is a dynamic route so it should always be inside the [id] or [slug] folder. |
| "use client"  import React, { useEffect, useState } from 'react';  import { Poppins } from 'next/font/google';  const poppins = Poppins({      subsets: ["latin"],      weight: "500"  });  export default function page({ params }) {      let id = params.userID;      const [name, setName] = useState("");      const [email, setEmail] = useState("");      const [age, setAge] = useState("");      const [message, setMessage] = useState('');      async function getUser() {          const request = await fetch(`http://localhost:3000/backend/api/users/${id}`);          const response = await request.json();          setName(response.message.name);          setEmail(response.message.email);          setAge(response.message.age);      }      useEffect(() => {          getUser();      }, [])  **async function handleSubmit(event) {**  **event.preventDefault();**  **const request = await fetch(`http://localhost:3000/backend/api/users/${id}`, {**  **method: "PUT",**  **headers: {**  **"Content-Type": "application/json"**  **},**  **body: JSON.stringify({ id:id,name: name, age: age, email: email })**  **});**  **const response = await request.json();**  **setMessage("Data Updated Successfully");**  **}**        return (          <div className={`${poppins.className} p-10`}>              <div className="flex flex-nowrap items-center gap-2">                  <h1 className='text-xl'>Update User : </h1>                  <h1 className='px-4 py-1 tracking-[1px] text-sm rounded-md bg-blue-500 text-white'>PUT</h1>              </div>              <div className="my-10 border-[1.5px] border-slate-500 bg-slate-100 rounded-md p-5 max-w-full sm:max-w-[60%] md:max-w-[50%] lg:max-w-[30%]">                  <form action="" onSubmit={handleSubmit}>                      <input required value={name} onChange={(e) => setName(e.target.value)} type="text" placeholder='Enter Name' className='border-[1.6px] w-full rounded-md my-3 p-3' />                      <br />                      <input required value={email} onChange={(e) => setEmail(e.target.value)} type="email" placeholder='Enter Email' className='border-[1.6px] w-full rounded-md my-3 p-3' />                      <br />                      <input required value={age} onChange={(e) => setAge(e.target.value)} type="number" placeholder='Enter Age' className='border-[1.6px] w-full rounded-md my-3 p-3' />                      <button type="submit" className='px-3 py-2 bg-slate-900 text-white rounded-md text-sm' >Submit</button>                  </form>              </div>              <div className="">                  Message :                  {message !== ""                      &&                      <p>{message}</p>                  }              </div>          </div>      )  } |

1. **DELETE METHOD :**

|  |
| --- |
| **BACKEND :** backend >users >[id] > page.js |
| export async function DELETE(request, content) {      const userID = content.params.id;      const mydata = users;      const newArray = mydata.filter(record => record.id != userID);      return NextResponse.json({ message: newArray }, { status: 200 });  } |

|  |
| --- |
| **FRONTEND :** |
| **Create a deleteUser Component :** because the event are not usable with server components like onClick()  "use client";  import { RiDeleteBin2Line } from "react-icons/ri";  import React from 'react'  export default function DeleteUser(props) {      async function deleteUser() {          const request = await fetch(`http://localhost:3000/backend/api/users/${props.id}`, {              method: "DELETE",          })          const response = await request.json();          alert("Record Deleted Successfully !!!");      }      return (          <button type='button' className='inline-flex text-sm gap-1 items-center' onClick={deleteUser} >Delete <RiDeleteBin2Line /></button>      )  }  **Import that component in the file where you want to use it :**  <td className='px-10 py-2 border-2 border-slate-900 cursor-pointer'>  <DeleteUser id={data.message.id} />  </td> |

**// CATCH ALL API ROUTES :**

|  |
| --- |
| **Create a file inside the backend folder :** backend>api> […foldername] > route.js  import { NextResponse } from "next/server";  export async function GET(request, content) {      console.log(content.params);      return NextResponse.json({          // message: "Catch All API Route"          message: content.params      }, { status: 200 })  }  **Access :** <http://localhost:3000/backend/api/ronak/4/json/6>  **Output :** {"message": {"users": ["ronak","4","json","6"]}} |