

# NextJS 13 / 14

## Basic Course

### Source:

<https://www.youtube.com/watch?v=A63UxsQsEbU&list=PL4cUxeGkcC9g9gP2onazU5-2M-AzA8eBw>

### Installation

If you need the latest version, run this command:

**npx create-next-app**

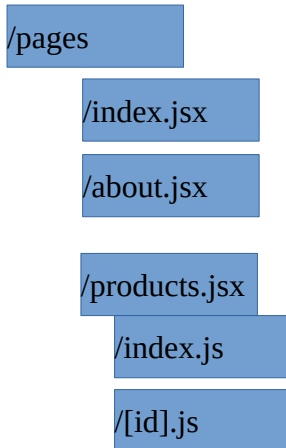
other versions:

**npx create-next-app@13**

## Basic Concept Routing in NEXTJS

For **pages Router**

In NextJS we use a file based routing network. That means, each route is created from a file



now we have an about page, homepage and `/products` page.

The `index.js` is only creating a `"/` route.

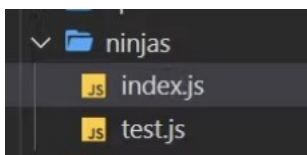
**Now – let us create some pages with dummy text.**

1. homepage
2. about page

extend with sub folders:

1. products

**here is the folder structure**



if you call `/ninjas/test` => `test.js` render

if you call `/ninjas` => the `index.js` will be rendered (creates a root path)

## More components:

there are other components – not only page components

these components will be stored by convention under „components“ folder.  
Similar to react folder structure

## Navigation

There is a slight difference to react links.

```
import Link from 'next/link'

const Navbar = () => {
  return (
    <nav>
      <div className="logo">
        <h1>Ninja List</h1>
      </div>
      <Link href="/"><a>Home</a></Link>
      <Link href="/about"><a>About</a></Link>
      <Link href="/ninjas"><a>Ninja Listing</a></Link>
    </nav>
  )
}
```



## Code splitting

nextjs by default will use code splitting.

That means that files will be provided from the server only if they get called.

### here is a test:

open the devtool => go to NETWORK

we start from the homepage!

let us navigate to the about page...you see the about.js will be served.

If you go back with the Arrow, and click again about → no server action and loading of about.js anymore.

## Layout Component

as well a slightly difference to react.

```
index.js  _app.js  Layout.js  •
comps > JS Layout.js > [⌕] Layout
1  import Footer from "../Footer"
2  import Navbar from "../Navbar"
3
4  const Layout = ({ children }) => {
5    return (
6      <div className="content">
7        <Navbar />
8        { children }
9        <Footer />
10     </div>
11   );
12 }
13
14 export default Layout;
```

```
1  import Layout from '../comps/Layout'
2  import '../styles/globals.css'
3
4  function MyApp({ Component, pageProps }) {
5    return (
6      <Layout>
7        <Component {...pageProps} />
8      </Layout>
9    )
10  }
11
12 export default MyApp
```

## Custom 404 page

You have to add a 404.js and you can have a custom error page.

**Further pages to custom:**

500 page

## Redirecting user

We will auto direkt the user after a certain time.

This redirection will be implemented to the 404 page

hint: router.go(-1) was an former function.

```
1 import Link from 'next/link'
2 import { useEffect } from 'react'
3 import { useRouter } from 'next/router'
4
5 const NotFound = () => {
6   const router = useRouter();
7
8   useEffect(() => {
9     setTimeout(() => {
10       // router.go(1)
11       router.push('/');
12     }, 3000)
13   }, [])
14
15   return (
16     <div className="not-found">
17       <h1>Oooops...</h1>
18       <h2>That page cannot be found.</h2>
19       <p>Go back to the <Link href="/"><a>Homepage</a></Link></p>
20     </div>
21   );
22 }
```

## getStaticProps

<https://youtu.be/zuelyEdRZQlk?si=upX2e1SuGee2Rdyu>

we want to fetch user data from json.placeholder to be rendered in an overview page.

Let us create a user path with an index.js

place this code:

```
1 export const getStaticProps = async () => {  
2  
3 }  
4  
5 const Ninjas = () => {  
6   return (  
7     <div>  
8       <h1>All Ninjas</h1>  
9     </div>  
10  );  
11 }  
12  
13 export default Ninjas;
```

the getStaticProps build in function is an async function which handles the fetch.

This function never runs in the browser ==> only on build time.

## Fetch Data with getStaticProps

Now we can provide the return props to the rendered component

let us create a users page in the pages folder:

/users/index.tsx

```
1  export const getStaticProps = async () => {
2
3    const res = await fetch('https://jsonplaceholder.typicode.com/users');
4    const data = await res.json();
5
6    return {
7      props: { ninjas: data }
8    }
9  }
10
11
12  const Ninjas = ({ ninjas }) => {
13    return (
14      <div>
15        <h1>All Ninjas</h1>
16      </div>
17    );
18  }
19
20  export default Ninjas;
```

finally map the data to the index.



## getStaticProps vs getServerSideProps

You should use **getServerSideProps** if you need to render a page that **relies on personalized user data**, or information that can only be known at request time. For example, authorization headers or a geolocation.

If you do not need to fetch the data at request time, or would prefer to cache the **data and pre-rendered HTML**, we recommend using [getStaticProps](#).

### Dynamic routes

<https://youtu.be/WPdJaBFquNc?si=1SvufEaOYv-IzXUN>

```
/ninja/id
/ninja/1
/ninja/2
/ninja/25
```

## set up dynamic routes in folder structure:



You can call [id] as well [slug] which is a convention.

Now we need to make the list of users as a clickable link for each user.

The link has to target „/users/userId“

```
const Ninjas = ({ ninjas }) => {
  return (
    <div>
      <h1>All Ninjas</h1>
      {ninjas.map(ninja => (
        <Link href={`/${ninja.id}`} key={ninja.id}>
          <a className={styles.single}>
            <h3>{ ninja.name }</h3>
          </a>
        </Link>
      ))}
    </div>
  )
}
```

## GetStaticPaths on dynamic routes

<https://youtu.be/mAHqpdVzJmA?si=1SuqWHRmRIXoiuzu>

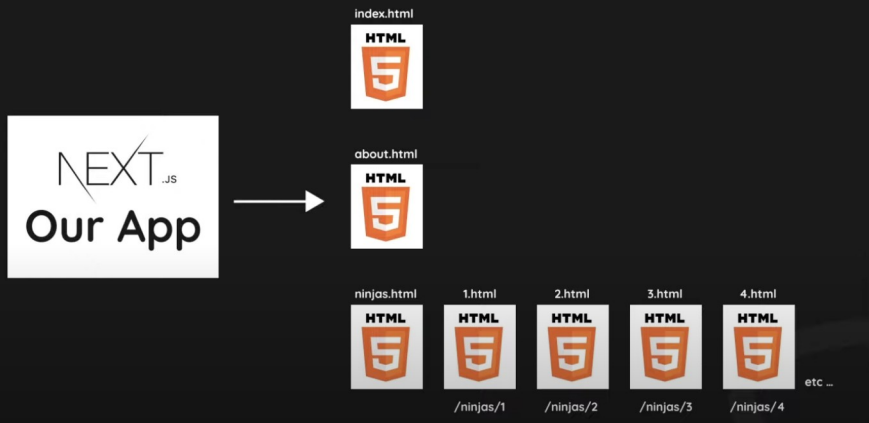
now we have a single user page.

Nextjs needs now to generate new pages for each user

nextjs does not know how many pages to be generated.

Now we need to tell next, how many pages has to be generated on build time.

# Static Site Generation



With `getStaticPaths` we are able to do so...

```

1  export const getStaticPaths = async () => {
2    const res = await fetch('https://jsonplaceholder.typicode.com/users');
3    const data = await res.json();
4
5    const paths = data.map(ninja => {
6      return {
7        params: { id: ninja.id.toString() }
8      }
9    })
10
11    return {
12      paths,
13      fallback: false
14    }
15  }
16
17  const Details = () => {
18    return (
19      <div>
20        <h1>Details Page</h1>
21      </div>
22    );
23  }
24
25  export default Details;

```



## What is going on there:

1. we fetch all data we want to have a page for => each user
2. we need to provide next the amount of pages based on the id.  
=> formating the number from the API to a string.
3. return the paths array

Now nextjs will create **on build time** the needed html pages for each user.

Next we need to tell the single component, what user we want to show

## getSingleUser

[https://youtu.be/2zRHlqc0\\_yw?si=0MzmMEm4YyNBKhBR](https://youtu.be/2zRHlqc0_yw?si=0MzmMEm4YyNBKhBR)

now we need as last the data for an individual user.

With `getStaticProps` we can gather the specific id from the context object

we add the `getStaticProps` to the Details Page. (line 17)

```
EDITORS 1 UNSAVED  pages > ninjas > [id].js > [id] getStaticProps > [id] id
LIST
.next
comps
node_modules
pages
api
ninjas
[id].js
index.js
_app.js
404.js
about.js
index.js
public
styles
globals.css
Home.module.css
Ninjas.module.css
.gitignore
package-lock.json
package.json
README.md

1  export const getStaticPaths = async () => {
2    const res = await fetch('https://jsonplaceholder.typicode.com/users');
3    const data = await res.json();
4
5    const paths = data.map(ninja => {
6      return {
7        params: { id: ninja.id.toString() }
8      }
9    })
10
11   return {
12     paths,
13     fallback: false
14   }
15 }
16
17 export const getStaticProps = async (context) => {
18   const id = context.params.id;
19 }
20
21 const Details = () => {
22   return (
23     <div>
24       <h1>Details Page</h1>
```

### Context looks like this:

```
const context:{
  id: value
}
```

Now we can work on the `getStaticProps` function to provide the individual userdata.

```
export const getStaticProps = async (context) => {
  const id = context.params.id;
  const res = await fetch('https://jsonplaceholder.typicode.com/users/' + id);
  const data = await res.json();

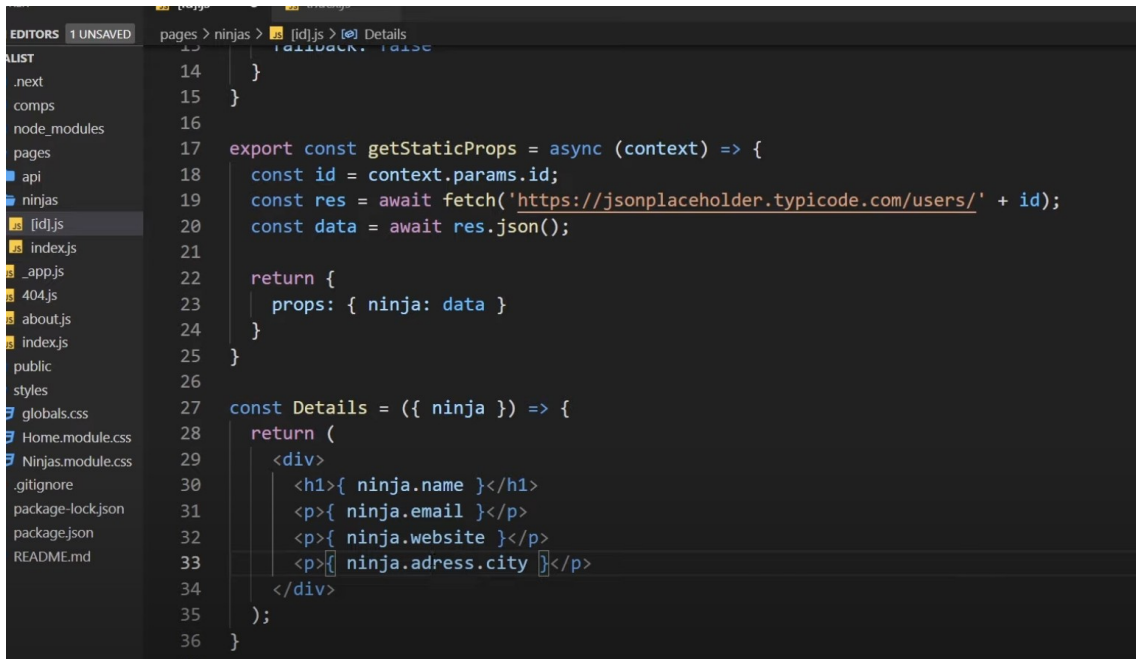
  return {
    props: { ninja: data }
  }
}

const Details = ({ ninja }) => {
  return (
    <div>
      <h1>Details Page</h1>
    </div>
  );
}
```

Based on that, we fetch the single user and provide the user in the return.

Now the `Details` element can work with the data.

We need to pass the the returned props from the getStaticProps to the Details page as an prop.



```
14 }
15 }
16
17 export const getStaticProps = async (context) => {
18   const id = context.params.id;
19   const res = await fetch('https://jsonplaceholder.typicode.com/users/' + id);
20   const data = await res.json();
21
22   return {
23     props: { ninja: data }
24   }
25 }
26
27 const Details = ({ ninja }) => {
28   return (
29     <div>
30       <h1>{ ninja.name }</h1>
31       <p>{ ninja.email }</p>
32       <p>{ ninja.website }</p>
33       <p>{ ninja.address.city }</p>
34     </div>
35   );
36 }
```

## Images and meta data

<https://youtu.be/rHncMH1CfCU?si=-6XHqlYN5X7ySpxT>

## Images in NextJS

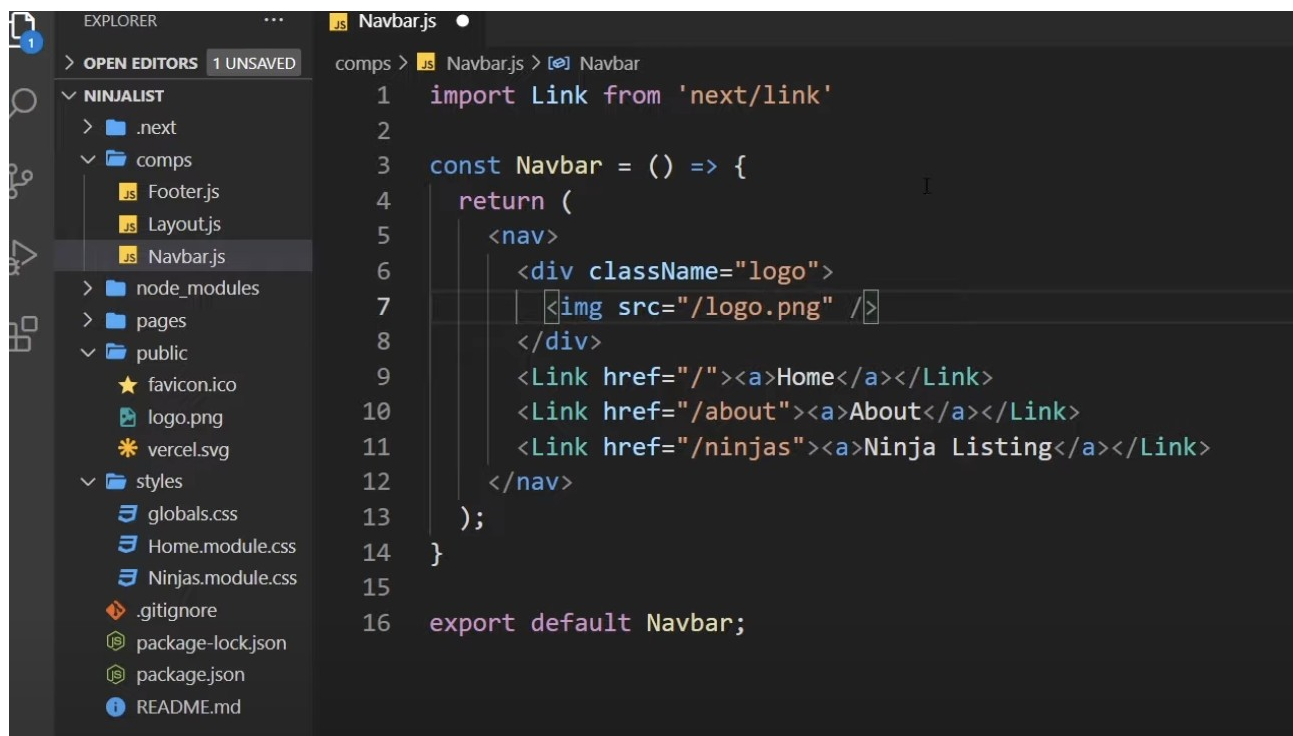
### Where to place the image?

The best and access easy place is the public folder for the image.

The client can access it anytime.

Let us place a new image in the public folder.

CHOOSE ANY IMAGE => place it to an component like the example below:



```
1 import Link from 'next/link'
2
3 const Navbar = () => {
4   return (
5     <nav>
6       <div className="logo">
7         
8       </div>
9       <Link href="/"><a>Home</a></Link>
10      <Link href="/about"><a>About</a></Link>
11      <Link href="/ninjas"><a>Ninja Listing</a></Link>
12    </nav>
13  );
14 }
15
16 export default Navbar;
```

Have a look at the image src. The link is calling directly the filename.

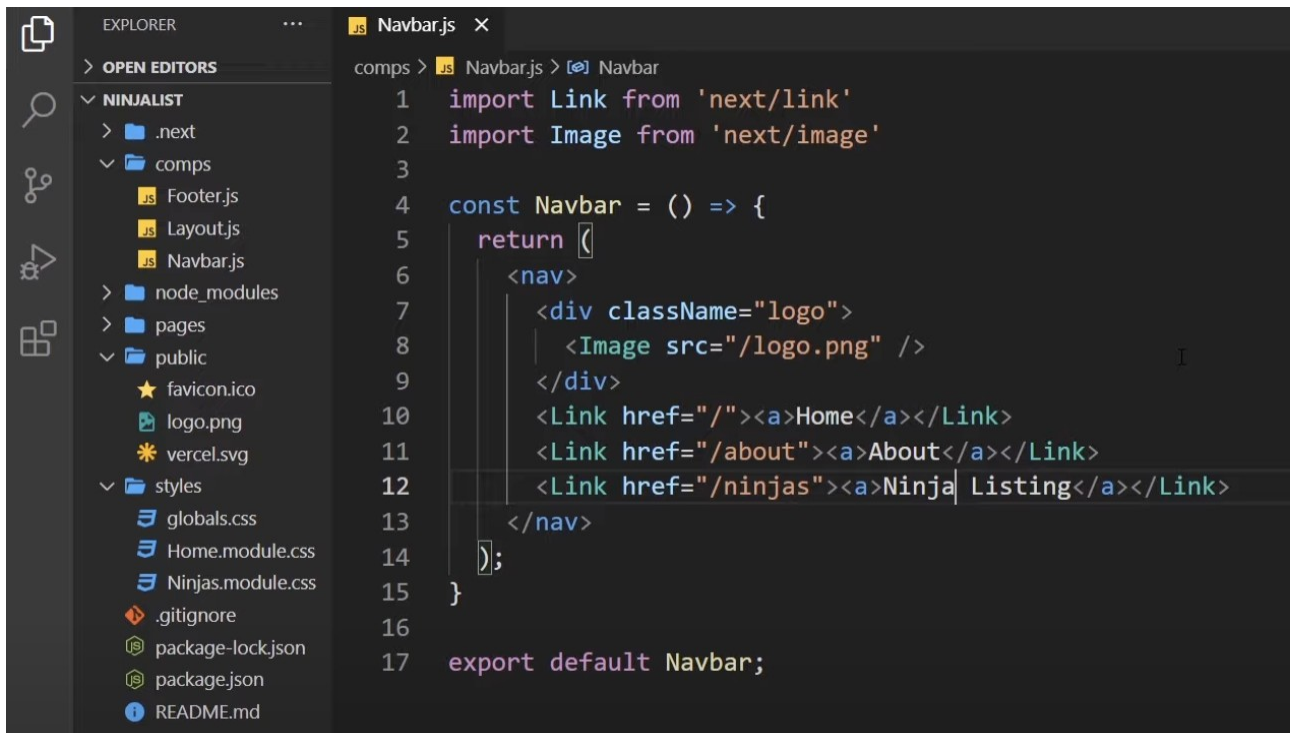
BUT.... The <img/> is not a perfect tag to use in nextjs.

Since Next 10 there is a special component <Image/>. We replace the <img/> with <Image/>



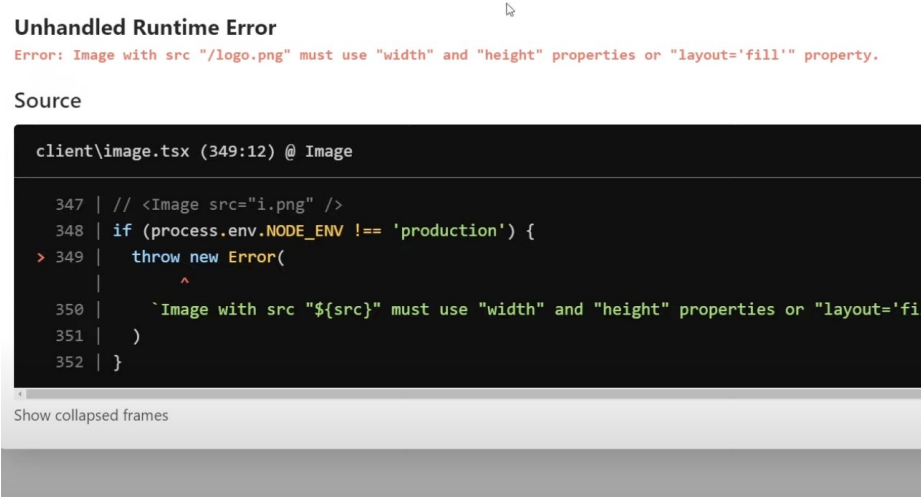
## Use the <Image/> tag

We now will change the tag to the new <Image/> component.



```
1 import Link from 'next/link'
2 import Image from 'next/image'
3
4 const Navbar = () => {
5   return (
6     <nav>
7       <div className="logo">
8         <Image src="/logo.png" />
9       </div>
10      <Link href="/"><a>Home</a></Link>
11      <Link href="/about"><a>About</a></Link>
12      <Link href="/ninjas"><a>Ninja Listing</a></Link>
13    </nav>
14  );
15 }
16
17 export default Navbar;
```

After reload –  
we get this error!



```
Unhandled Runtime Error
Error: Image with src "/logo.png" must use "width" and "height" properties or "layout='fill'" property.

Source

client\image.tsx (349:12) @ Image
347 | // <Image src="i.png" />
348 | if (process.env.NODE_ENV !== 'production') {
> 349 |   throw new Error(
      |         ^
350 |     `Image with src "${src}" must use "width" and "height" properties or "layout='fi
351 |   )
352 | }
```

## Add the width and the height !

```
Navbar.js x
comps > js Navbar.js > [e] Navbar
1  import Link from 'next/link'
2  import Image from 'next/image'
3
4  const Navbar = () => {
5    return (
6      <nav>
7        <div className="logo">
8          <Image src="/logo.png" width={128} height={77} />
9        </div>
10       <Link href="/"><a>Home</a></Link>
11       <Link href="/about"><a>About</a></Link>
12       <Link href="/ninjas"><a>Ninja Listing</a></Link>
13     </nav>
14   );
15 }
16
17 export default Navbar;
```

Now the image has no error and it looks better.

More benefits:

- lazy loading the image out of the box => if the image is in a lower area of the page, the image will loaded only on view enter

## Meta Data in NextJS

We can use the build in Component `<Head>` to place on the dedicated page the `<head>` information as we like.

Example:

```
ges > js index.js > Home
1  import Head from 'next/head'
2  import Navbar from '../comps/Navbar'
3  import Footer from '../comps/Footer'
4  import styles from '../styles/Home.module.css'
5  import Link from 'next/link'
6
7  export default function Home() {
8    return (
9      <>
10     <Head>
11     .....<title>Ninja List | Home</title>
12     .....<meta name="keywords" content="ninjas"/>
13     .....</Head>
14     <div>
15       <h1 className={styles.title}>Homepage</h1>
16       <p className={styles.text}>Lorem, ipsum dolor sit amet consectetur adipisicing eli
17       <p className={styles.text}>Lorem, ipsum dolor sit amet consectetur adipisicing eli
18       <Link href="/ninjas">
19       | <a className={styles.btn}>See Ninja Listing</a>
20       </Link>
21     </div>
22   </>
23 )
24 }
```

## Fonts in NextJS 13

<https://nextjs.org/docs/pages/building-your-application/optimizing/fonts>

## API Routes

With api routes we have a perfect place to manage request in our nextjs application directly to avoid direct calls to the real api data.

The API routes is based on ExpressJS / NodeJS and follows the same rules with slightly different parameters.

### Example:

```
1  data: {}
2  }
3
4  function handler(req: NextApiRequest, res:
5  NextApiResponse<Data>): Promise<void>
6
7  export default async function handler(
8    req: NextApiRequest,
9    res: NextApiResponse<Data>
10 ) {
11   console.log("request data", req)
12   if (req.method === 'POST') {
13     // Process a POST request
14   } else if (req.method === 'GET') {
15
16     const response = await fetch('https://jsonplaceholder.typicode.com/posts')
17     const data = await response.json()
18     res.status(200).json(data)
19
20     // Handle any other HTTP method
21     res.status(200).json({ name: 'John Doe' })
22   } else if (req.method === 'PUT') {
23     res.status(200).json({ msg: "Put method provided" })
24   }
25 }
26
27
28
29
30
```

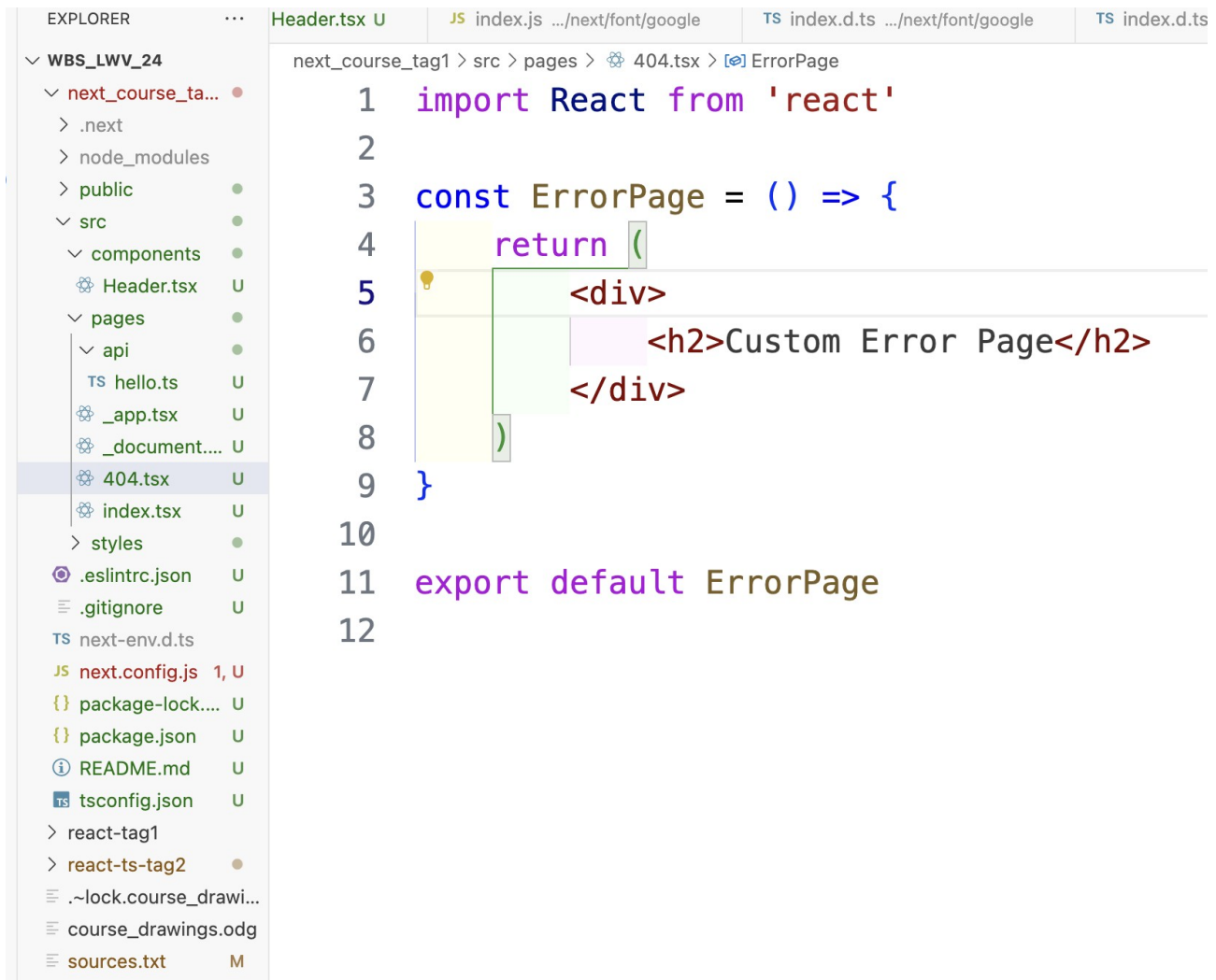
Above you see a dynamic handler. This handler function checks first, what kind of method the request has. An if/else statement handles the req method and call the function as provided.

CHECK WITH POSTPAN

## Custom Errors

<https://nextjs.org/docs/pages/building-your-application/routing/custom-error>

Some pages are reserved files like 500 / 404



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'pages' directory containing '404.tsx'. The code editor shows the content of '404.tsx', which is a custom error page component. The code is as follows:

```
1 import React from 'react'
2
3 const ErrorPage = () => {
4   return (
5     <div>
6       <h2>Custom Error Page</h2>
7     </div>
8   )
9 }
10
11 export default ErrorPage
12
```

## Lazy loading

We load a dynamic element <DynamicHeader/> to the index.tsx page in pages

next\_course\_ta...

- > .next
- > node\_modules
- > public
- > src
- > components
- > pages
  - > api
  - TS hello.ts
  - \_app.tsx
  - \_document.tsx
  - index.tsx**
- > styles
  - # globals.css
  - # Home.module.css
- .eslintrc.json
- .gitignore
- TS next-env.d.ts
- JS next.config.js
- () package-lock.json
- () package.json
- ① README.md
- tsconfig.json
- > react-tag1
- > react-ts-tag2
- course\_drawings.odg
- sources.txt

You, 43 minutes ago | 1 author (You)

```

1  import Head from 'next/head'
2  import Image from 'next/image'
3  import { Inter } from 'next/font/google'
4  import styles from '@styles/Home.module.css'
5
6  const inter = Inter({ subsets: ['latin'] })
7  import dynamic from 'next/dynamic'
8
9  const DynamicHeader = dynamic(() => import('../components/Header'), {
10    loading: () => <p>Loading...</p>,
11  })
12
13  You, 43 minutes ago * tag3
14  export default function Home() {
15    return (
16      <>
17      <Head>
18        <title>Create Next App</title>
19        <meta name="description" content="Generated by create next app" />
20        <meta name="viewport" content="width=device-width, initial-scale=1" />
21        <link rel="icon" href="/favicon.ico" />
22      </Head>
23      <DynamicHeader/>
24      <main className={` ${styles.main} ${inter.className}`}>
25        <div className={styles.description}>

```