

Link Component Navigation in Next.js

Navigation:

Navigation is simply moving from one page to another within a website or web application. Think of it like traveling from one place to another on the internet.

Navigation refers to the process of moving between different pages or routes within a web application. In the context of Next.js, navigation typically involves moving between different pages within a Next.js application, which follows file-based routing

File based routing:

In Next.js, the structure of your project's folders and files determines the URLs of your web pages. Each page typically corresponds to a specific file in your project. So, when you visit a URL, Next.js looks for the corresponding file to display the page.

Users navigate through a web application either by manually entering URLs in the browser's address bar, clicking on UI elements like links, or through programmatic navigation, such as after completing an action.

Manually Entering URLs:

This is when you type a web address directly into your browser's address bar to go to a specific page. For example, typing "<http://localhost:3000/blog> " in the address bar would take you to the blog page of the website.

UI Elements for Navigation:

Instead of typing URLs, users often navigate through a website by clicking on buttons or links. These are called UI elements. Clicking on them directs users to different pages or sections of the website.

UI Navigation:

This refers to the process of moving from one page to another within a website by clicking on UI elements. For instance, if you're on the home page of a website and you click on a "Blog" link, you expect to be taken to the blog page.

Link Component Navigation:

In Next.js, UI navigation is facilitated through components like the `<Link>` component. The `<Link>` component enables client-side navigation by extending the HTML `<a>` element. It allows users to navigate between routes within a Next.js application without full page refreshes, enhancing the user experience.

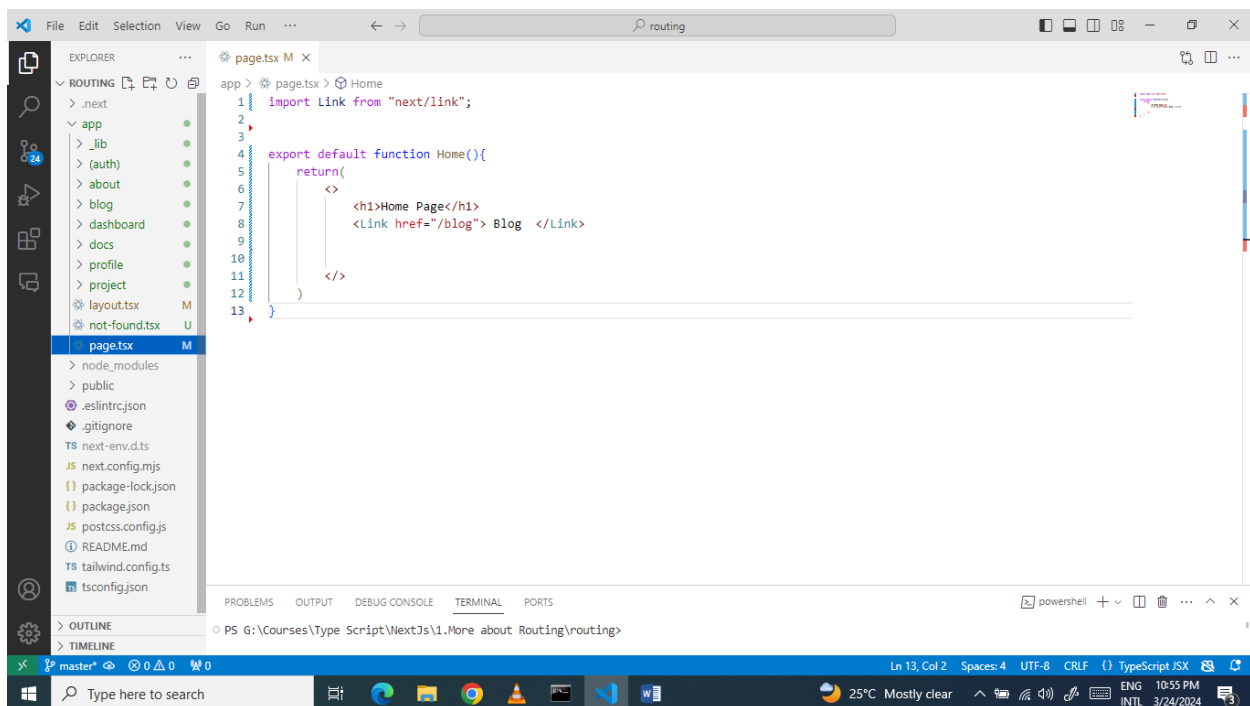
In Next.js, the `<Link>` component is a special tool for creating links between pages within your application. It's like a magic button that takes users from one page to another without reloading the entire page. You import the `<Link>` component from the "next/link" module, and then you use it to create clickable links in your application. This way, when users click on a link created with the `<Link>` component, Next.js handles the navigation smoothly without a full-page refresh.

So, in summary, Link Component Navigation in Next.js makes it easy to create clickable links that smoothly take users from one page to another within your web application, enhancing the user experience by providing fast and seamless navigation.

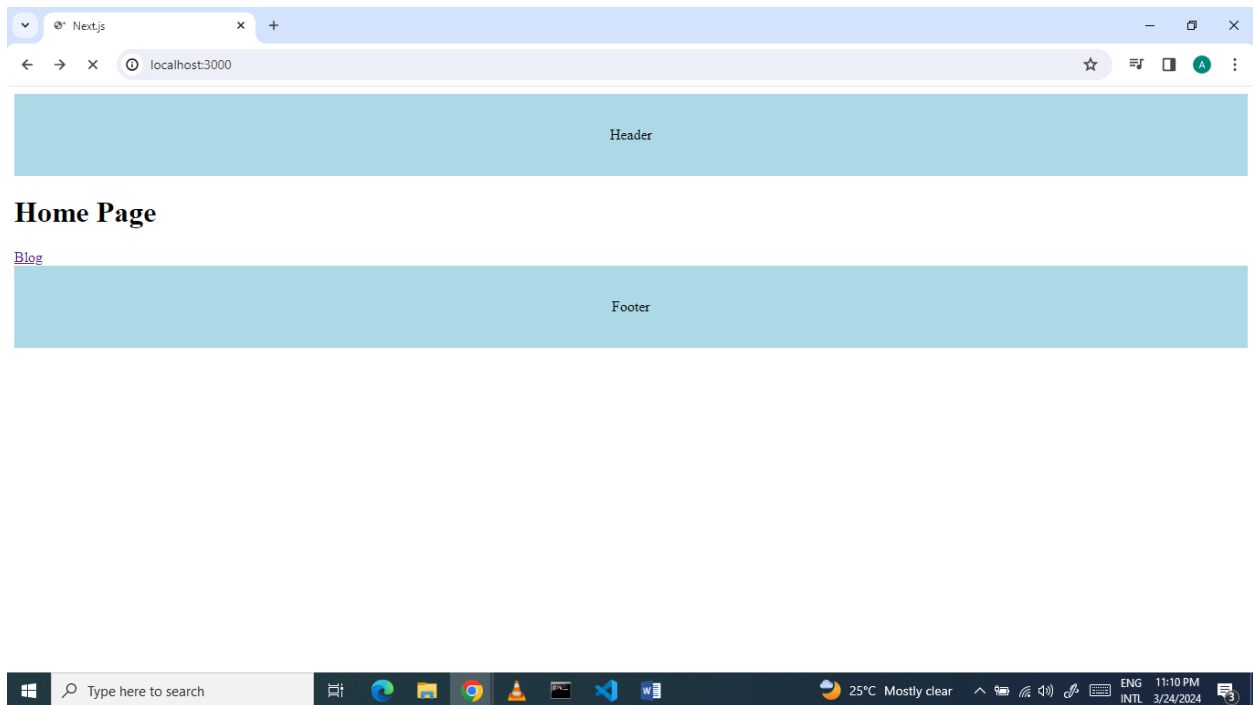
"Next.js Link Component Navigation : Step-by-Step Guide with Screenshots"

To use the `<Link>` component in Next.js, you need to import it from the `next/link` module. This component is crucial for implementing navigation within your Next.js application while leveraging the benefits of client-side routing provided by Next.js.

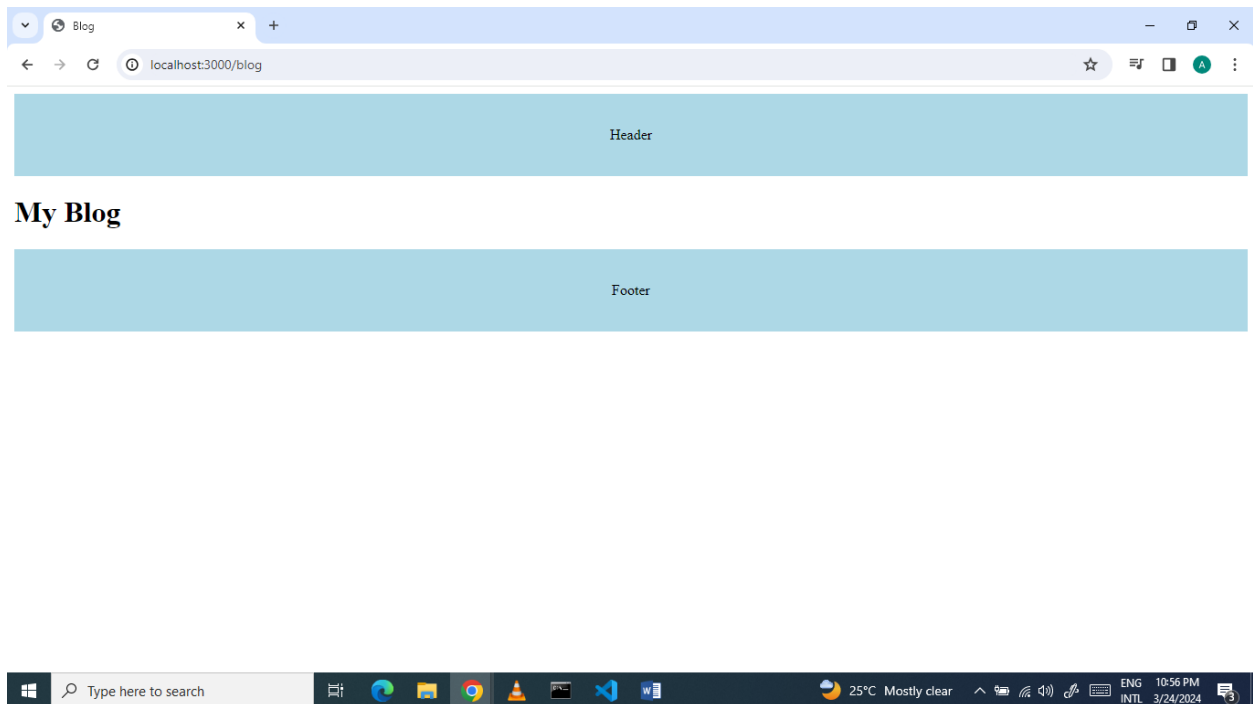
"Go to the **page.tsx** file inside the **app** folder and import the Link component. We can use the Link component to navigate to any route in our application."



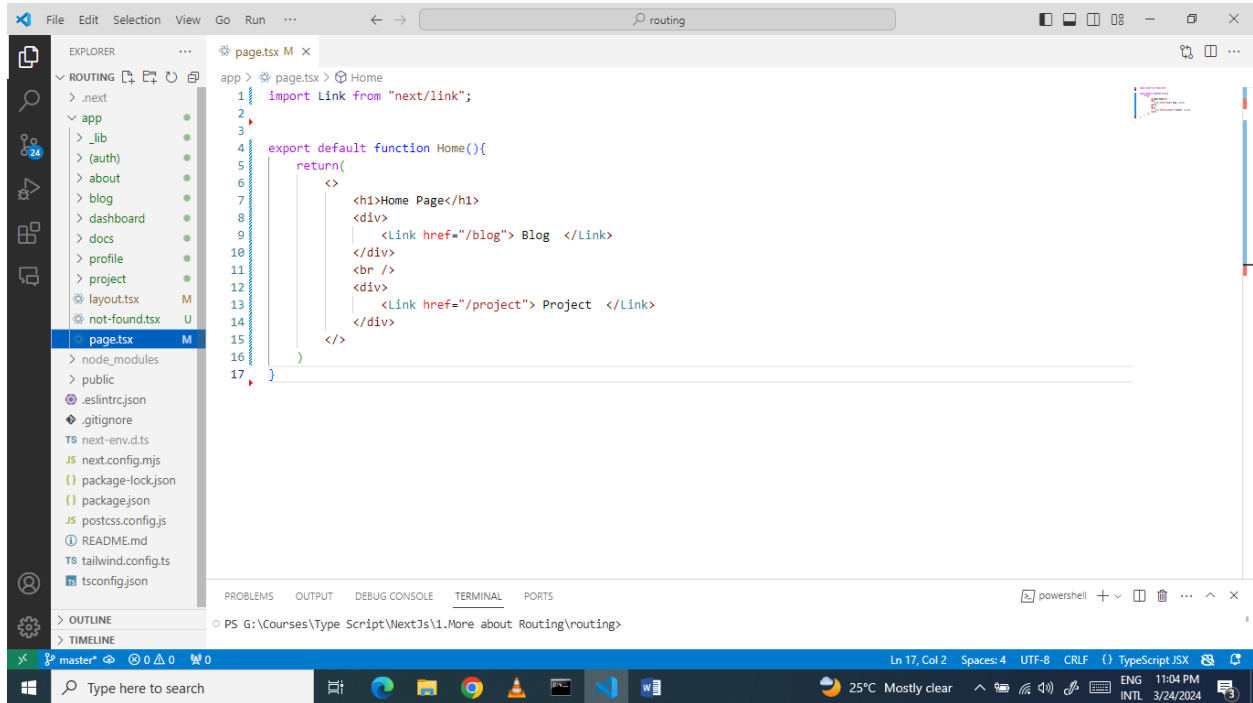
"Go to the browser and observe the rendered Blog link."



"The blog link is now active, and clicking on it will successfully navigate to the blog page."



"Second example: To navigate to the project page, add a link in the page.tsx file inside the app folder."

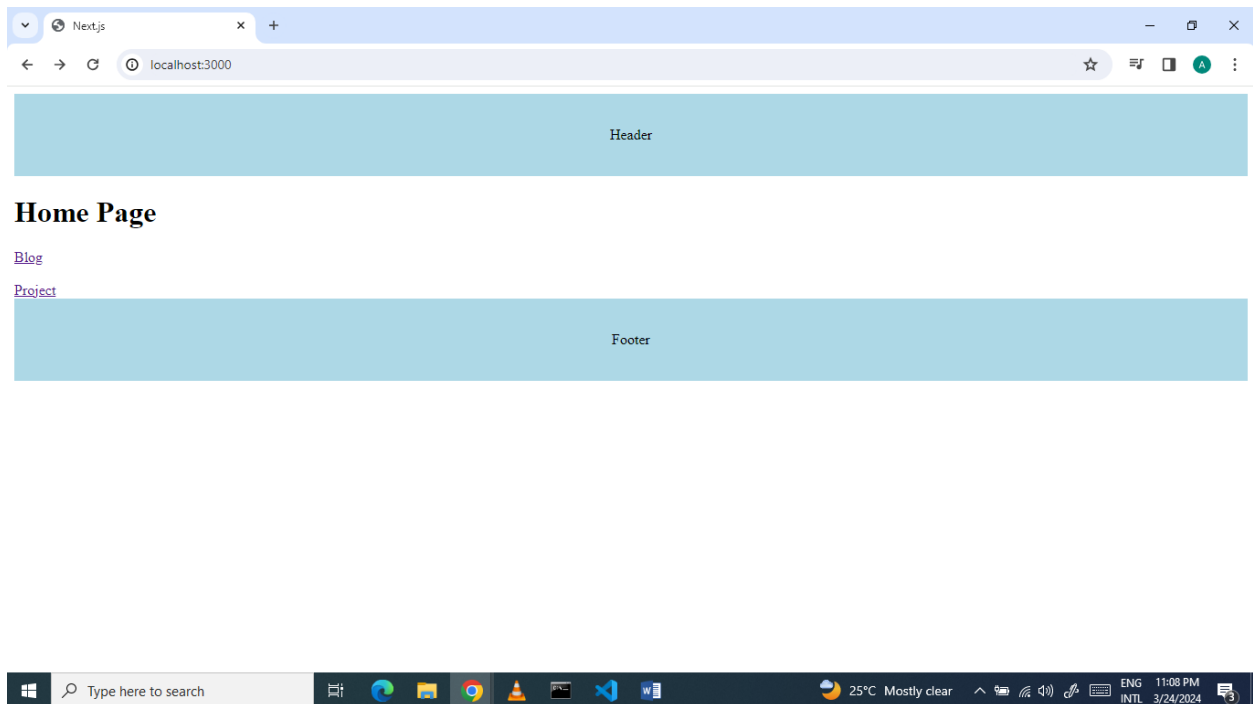


The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows the project structure with the 'app' folder expanded, and 'page.tsx' selected. The main editor displays the content of 'page.tsx':

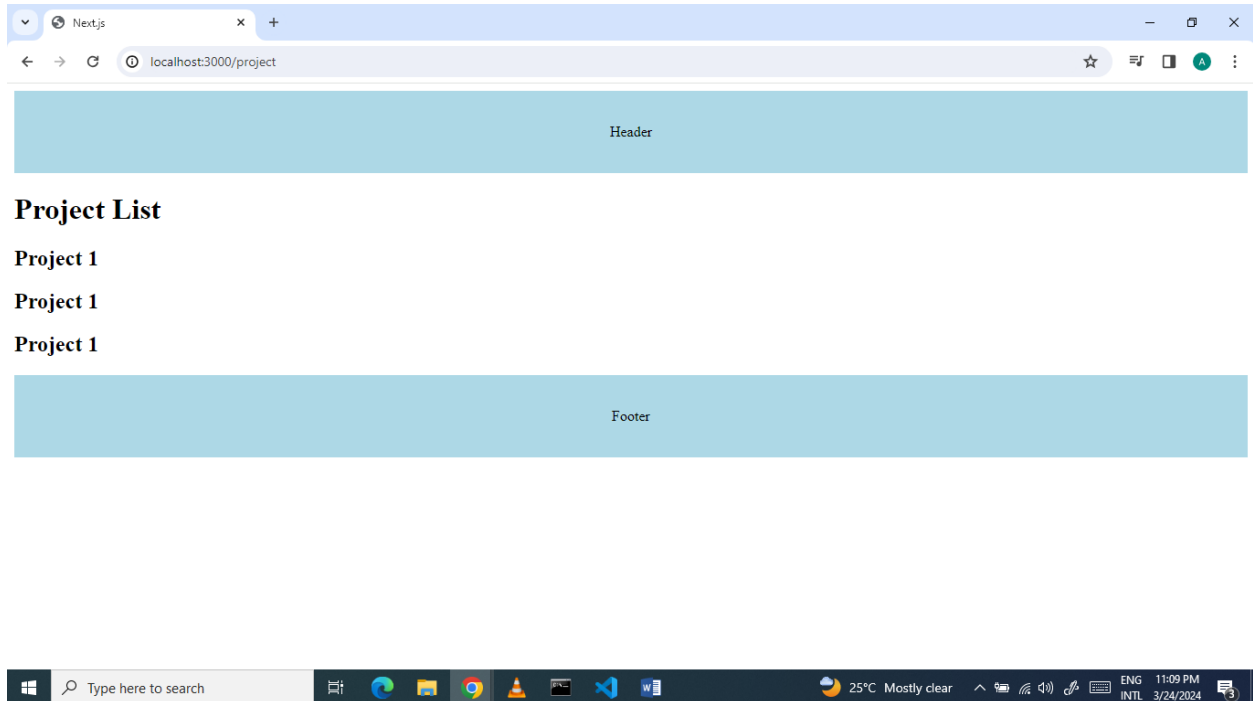
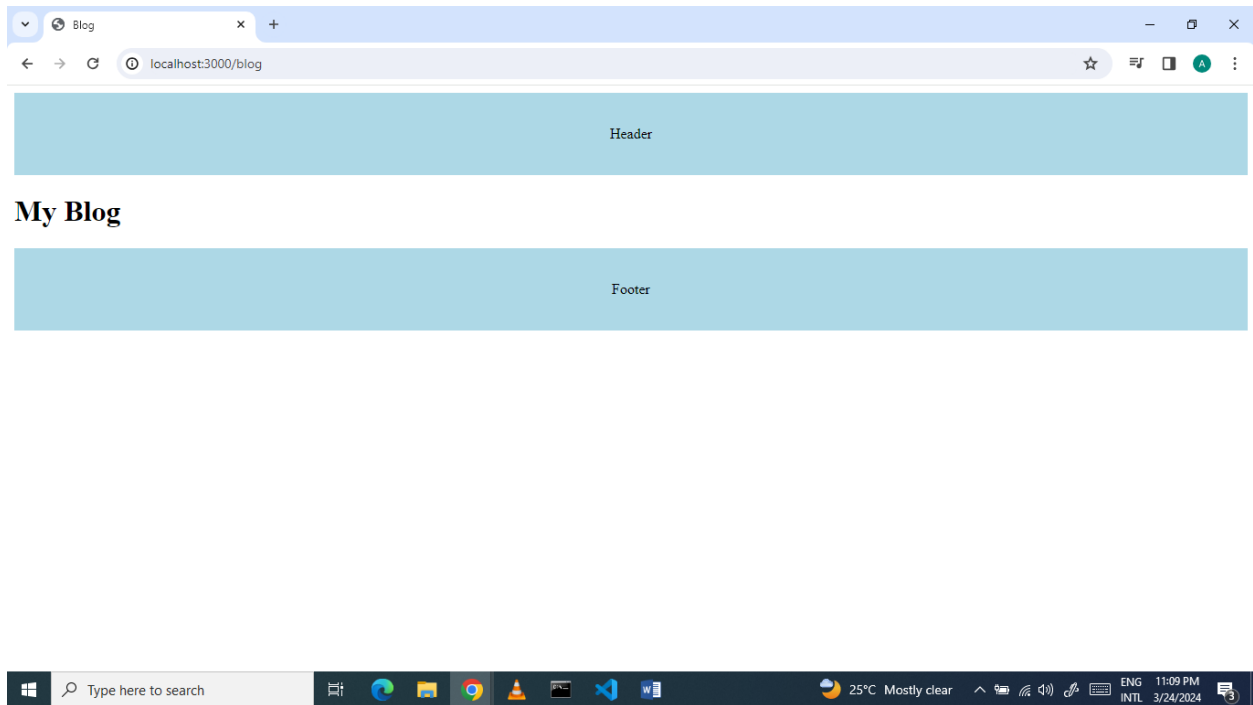
```
1 import Link from "next/link";
2
3
4 export default function Home(){
5   return(
6     <>
7       <h1>Home Page</h1>
8       <div>
9         <Link href="/blog"> Blog </Link>
10      </div>
11      <br />
12      <div>
13        <Link href="/project"> Project </Link>
14      </div>
15    </>
16  )
17 }
```

The bottom status bar shows the file is 'TypeScript JSX' and the terminal is open with the command 'PS G:\Courses\Type Script\NextJs\1.More about Routing\routing>'.

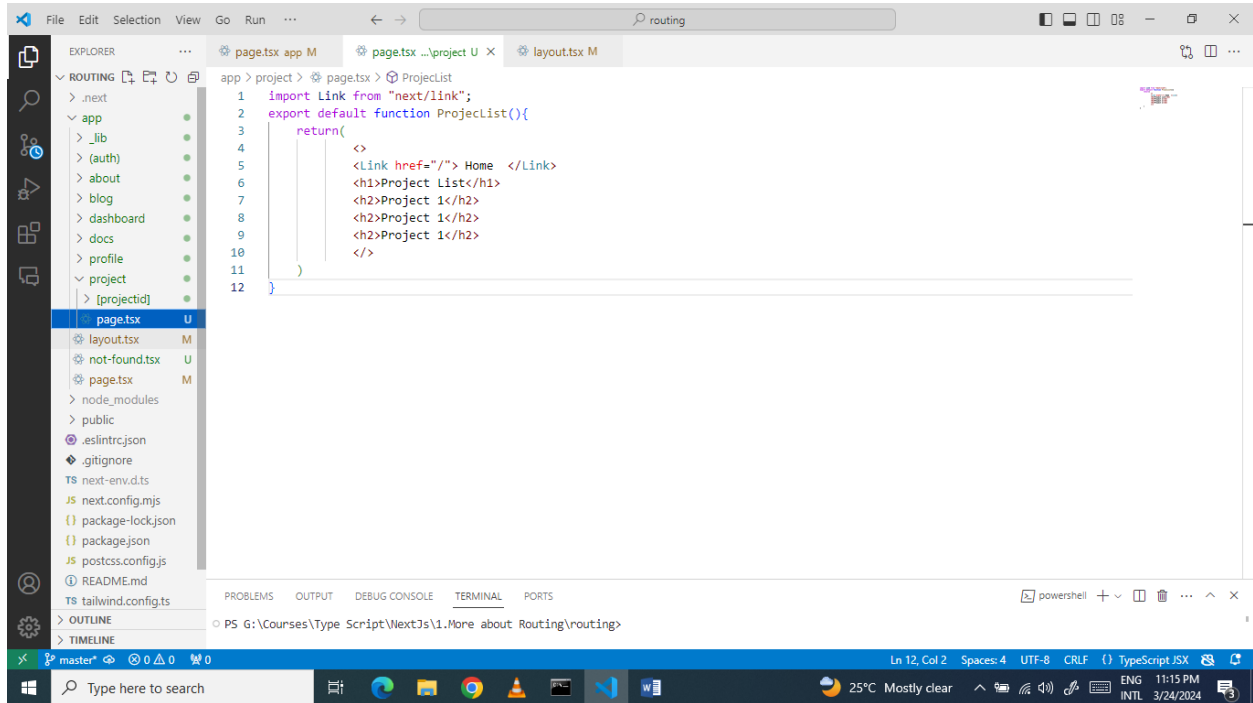
"Go to the browser and see the rendered Blog and Project links."



"The Blog and Project links are now accessible. Clicking on the Blog link will successfully navigate to the blog page, while clicking on Project will successfully navigate to the project page."

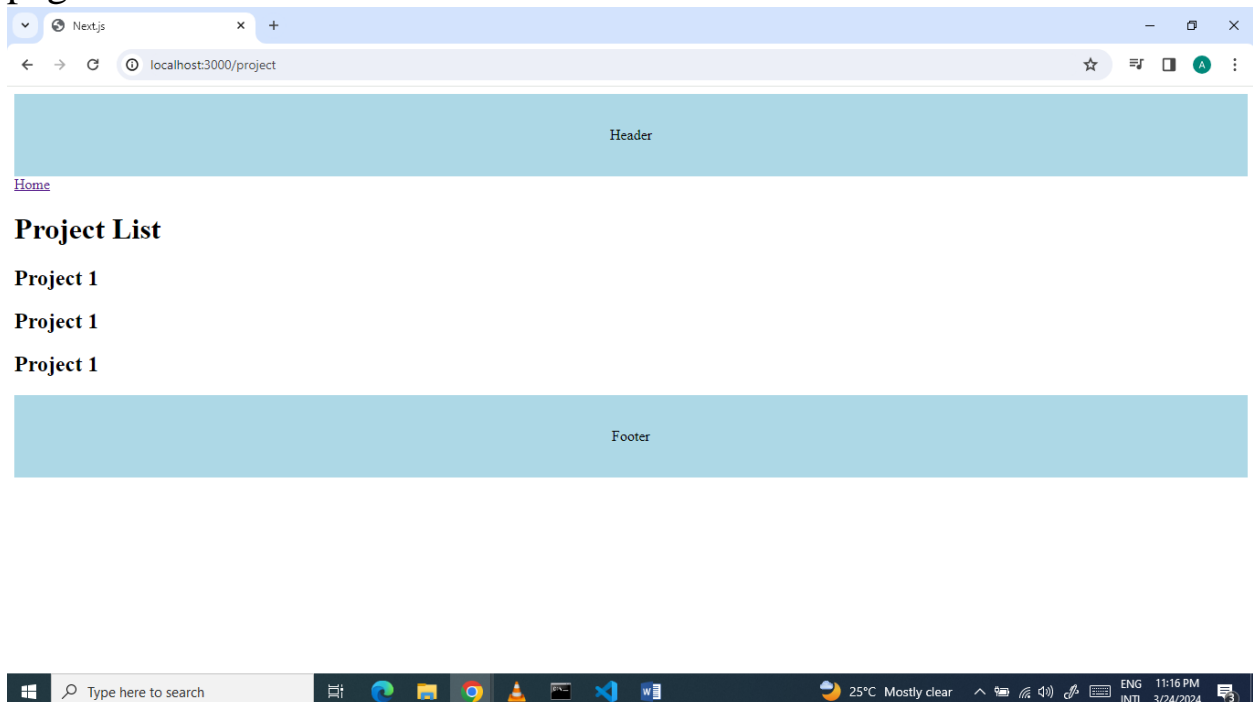


"Now, add a link on the project or blog page to navigate back to the Home Page. Go to the 'page.tsx' file inside the 'project' folder and add the Link component."

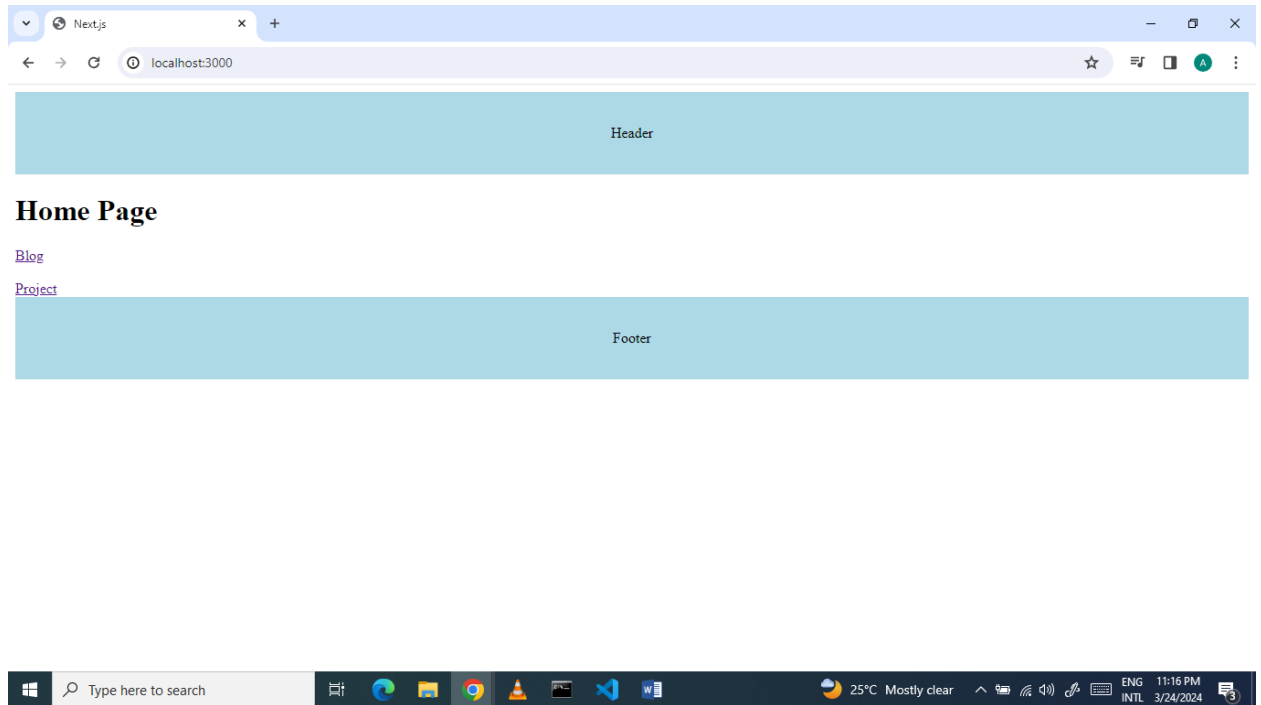


```
1 import Link from "next/link";
2 export default function ProjectList(){
3   return(
4     <>
5     <Link href="/"> Home </Link>
6     <h1>Project List</h1>
7     <h2>Project 1</h2>
8     <h2>Project 1</h2>
9     <h2>Project 1</h2>
10    </>
11  )
12 }
```

"Go to the browser and see the rendered 'Home' link on the project page."

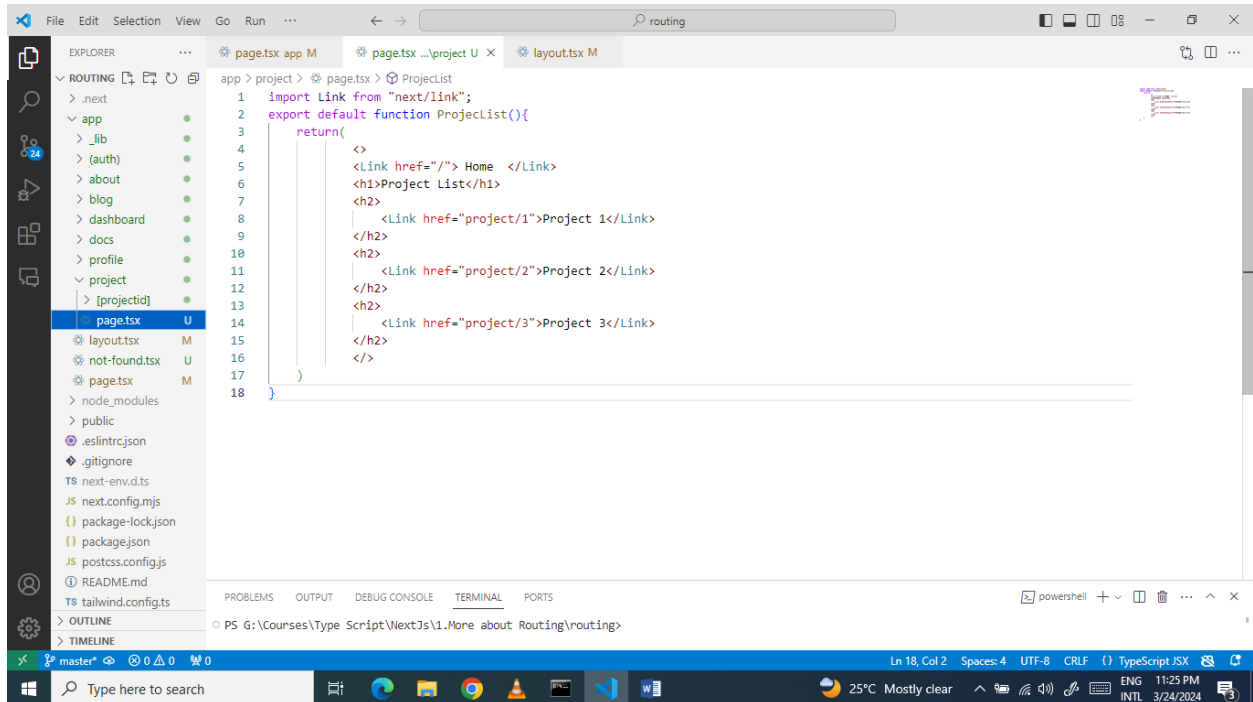


"The Home link is now open, and by clicking on it, you will successfully navigate to the home page."



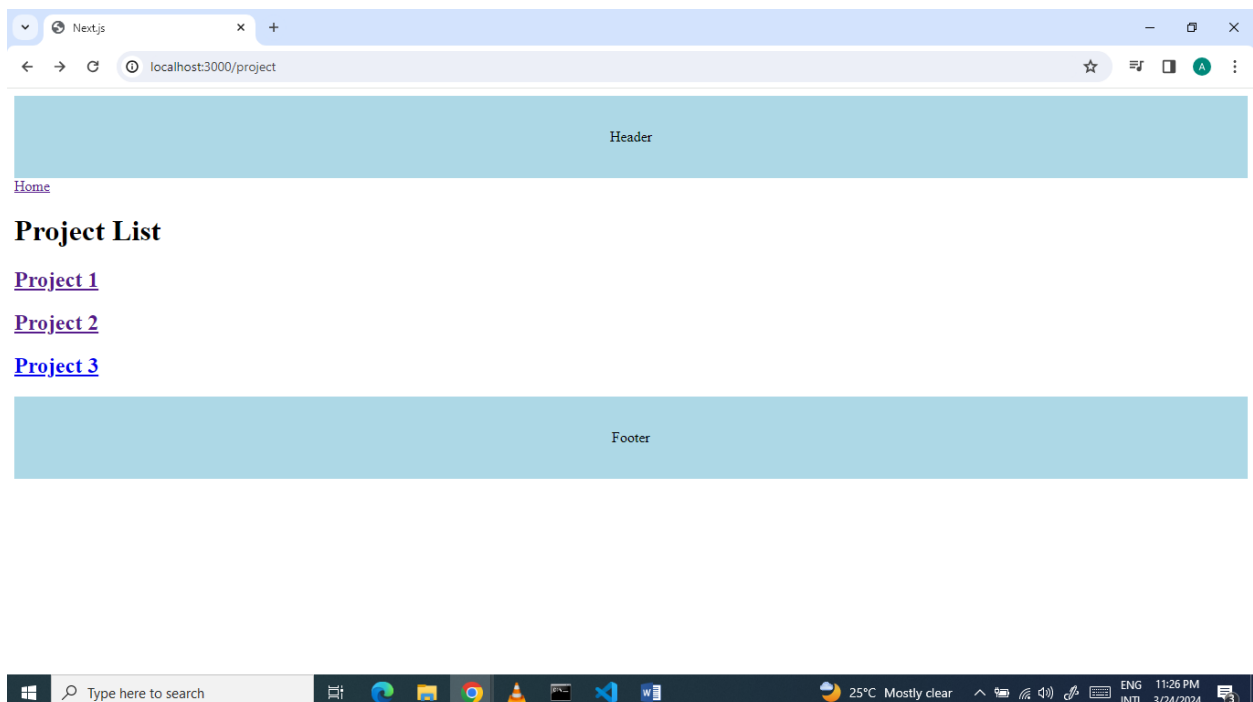
"Go to the next example to navigate to dynamic routes."

"In the page.tsx file, include Link components in each heading tag for all available projects."

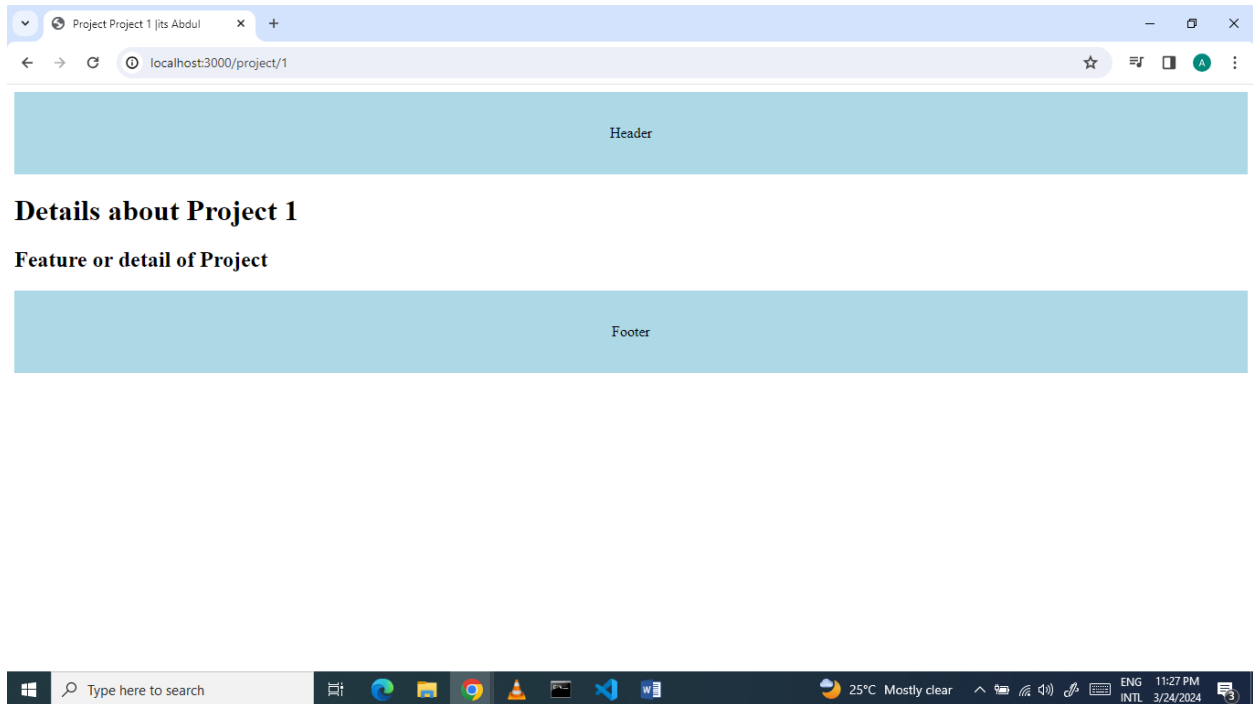


```
1 import Link from "next/link";
2 export default function ProjectList() {
3   return (
4     <>
5     <Link href="/"> Home </Link>
6     <h1>Project List</h1>
7     <h2>
8       <Link href="project/1">Project 1</Link>
9     </h2>
10    <h2>
11      <Link href="project/2">Project 2</Link>
12    </h2>
13    <h2>
14      <Link href="project/3">Project 3</Link>
15    </h2>
16    </>
17  )
18 }
```

"Go to the browser and view the rendered Project links on the project page."

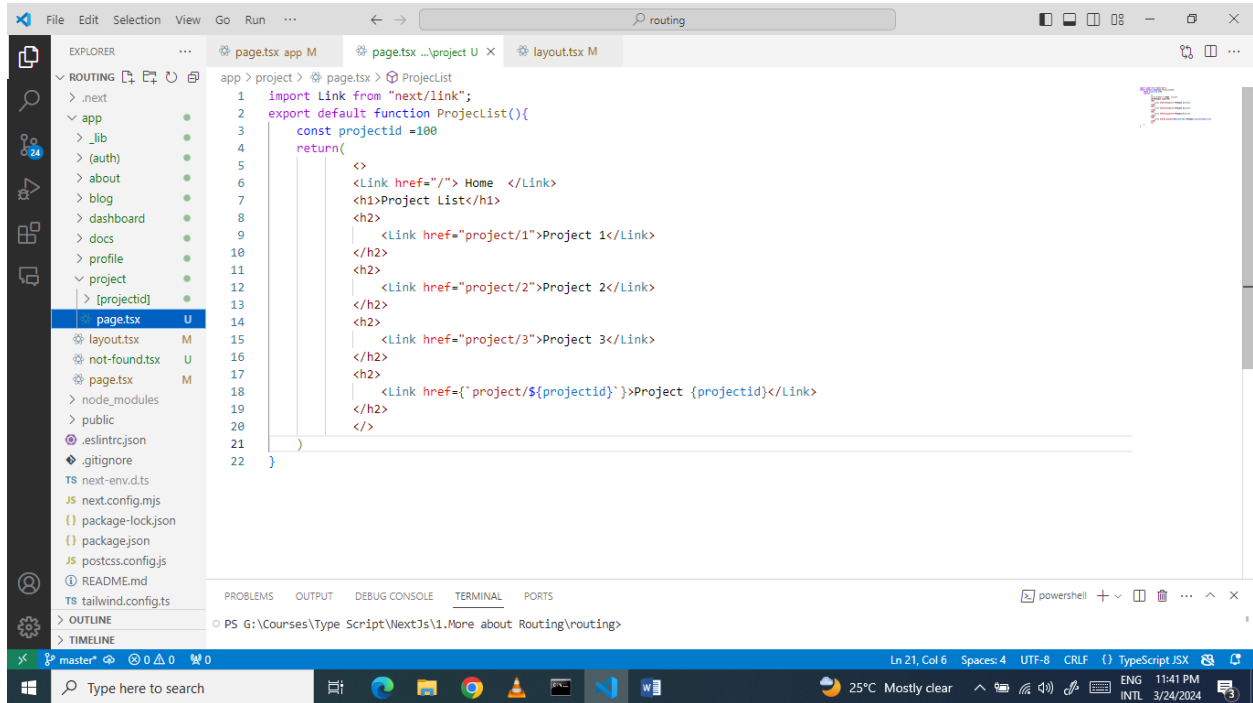


The dynamic link is now open. All three projects should be clickable, and by clicking on a project, it will successfully navigate to the project detail page and Successfully implemented navigation for dynamic routes.



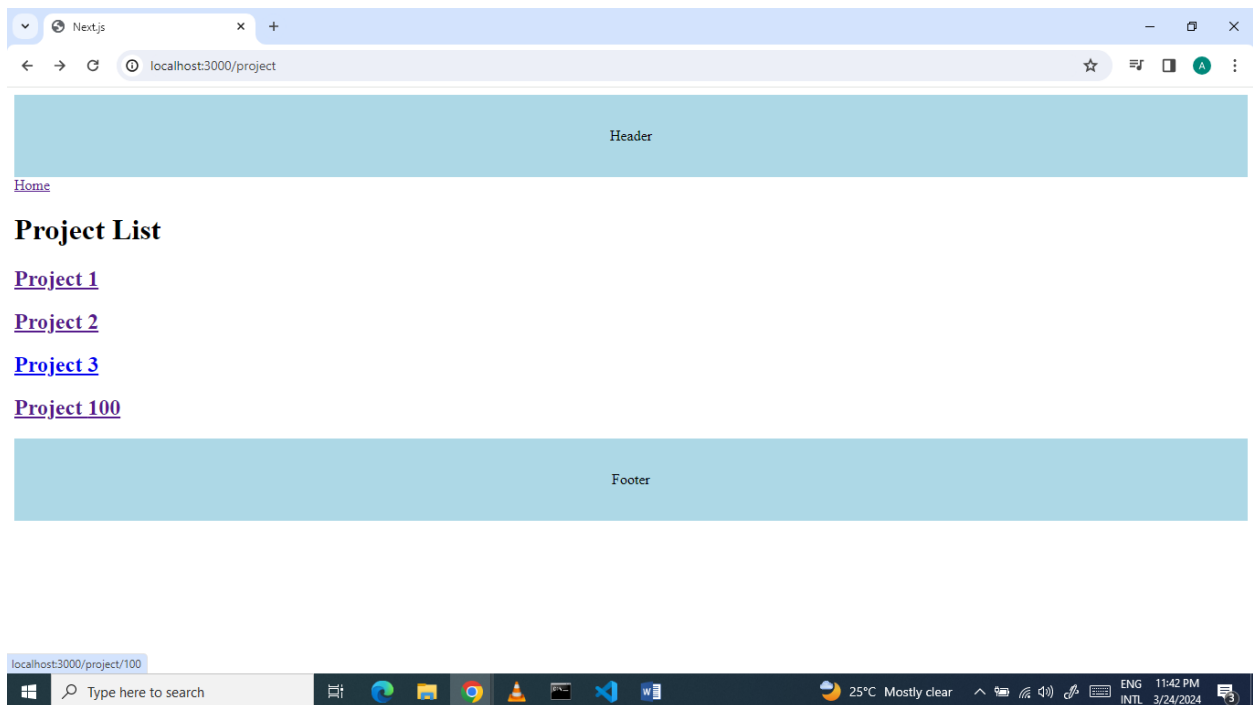
By using fixed routes can be okay sometimes, but it might not work well for all situations. In those cases, you can send the projectid as a prop to the components. For example, let's say we want to go to a project with an ID of 100. We can set const projectid to 100 to go to that specific project. By using string interpolation with the href attribute, we can easily change the link destination.

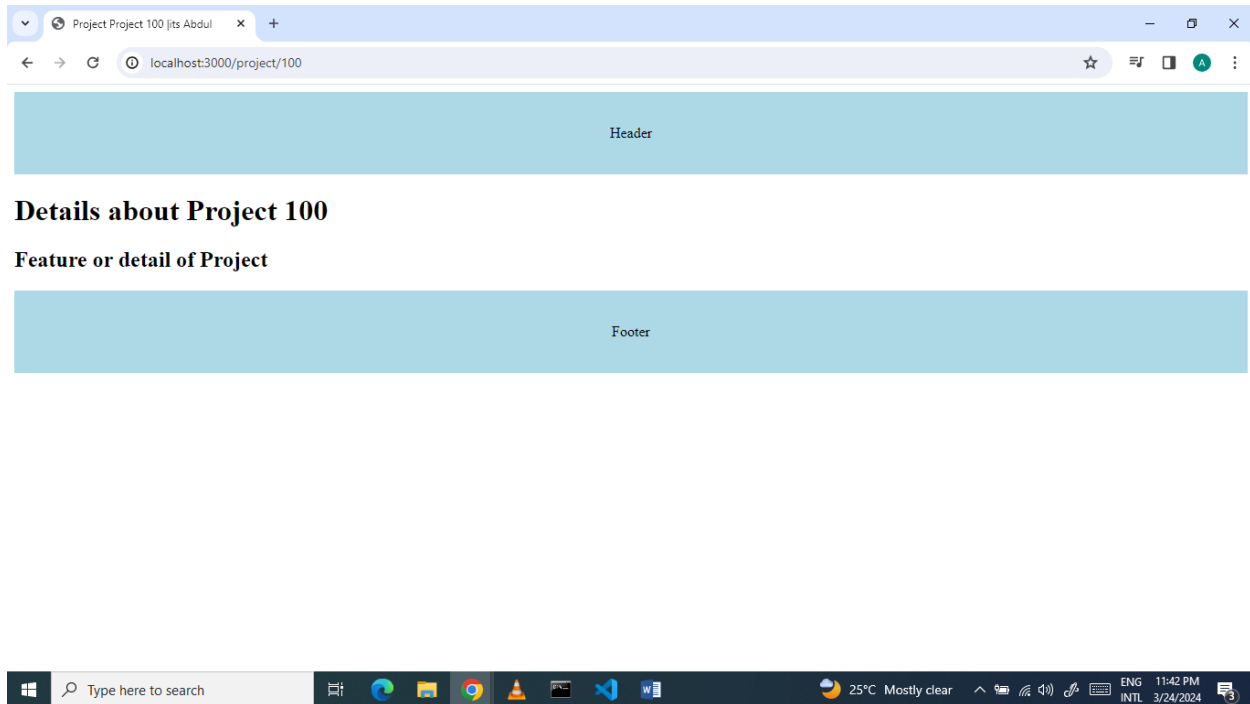
"In the page.tsx file inside the project folder, add the code as shown in the screenshots below."



```
1 import Link from "next/link";
2 export default function ProjectList(){
3   const projectId =100
4   return(
5     <>
6     <Link href="/"> Home </Link>
7     <h1>Project List</h1>
8     <h2>
9       <Link href="project/1">Project 1</Link>
10    </h2>
11    <h2>
12      <Link href="project/2">Project 2</Link>
13    </h2>
14    <h2>
15      <Link href="project/3">Project 3</Link>
16    </h2>
17    <h2>
18      <Link href={`project/${projectId}`}>Project {projectId}</Link>
19    </h2>
20    </>
21  )
22 }
```

"You should now see 'Project100', and clicking on it will navigate you to the detail page of the same project."



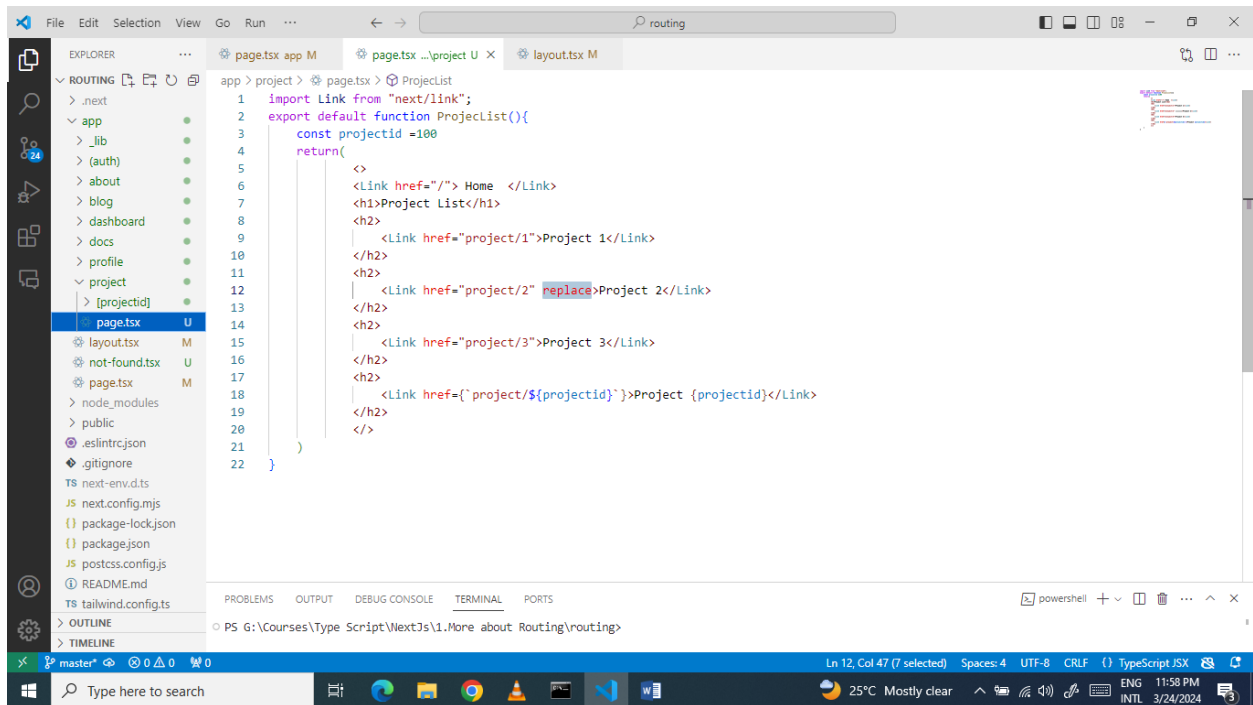


Our link buttons that help us move around the website are working well. I want to talk about one special thing about these buttons called the 'replace' feature. Let's see how this 'replace' feature works when we click on a specific project (see below screen shots).

So, the 'replace' feature changes the current page instead of adding a new one to the history.

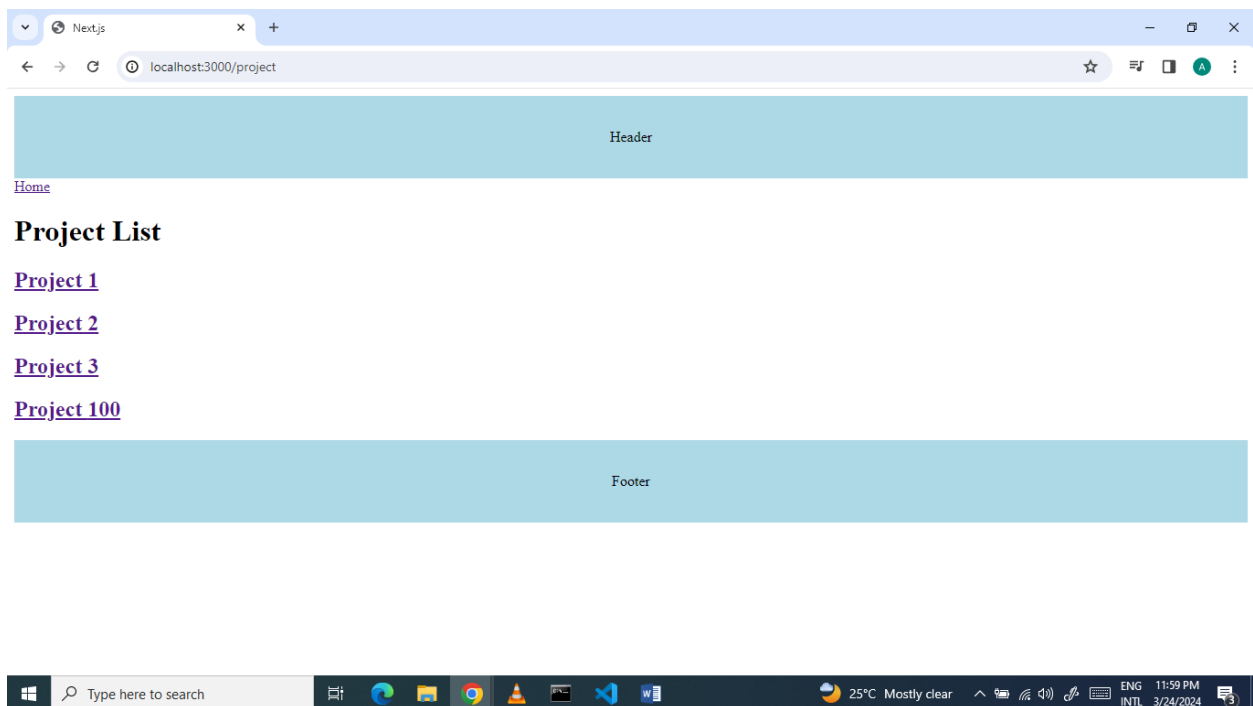
To sum up, the Link component in Next.js helps us move between pages using buttons. To use it, just import the Link component from 'next/link' and add it to your page with the text you want to show and the 'href' attribute to specify where it goes.

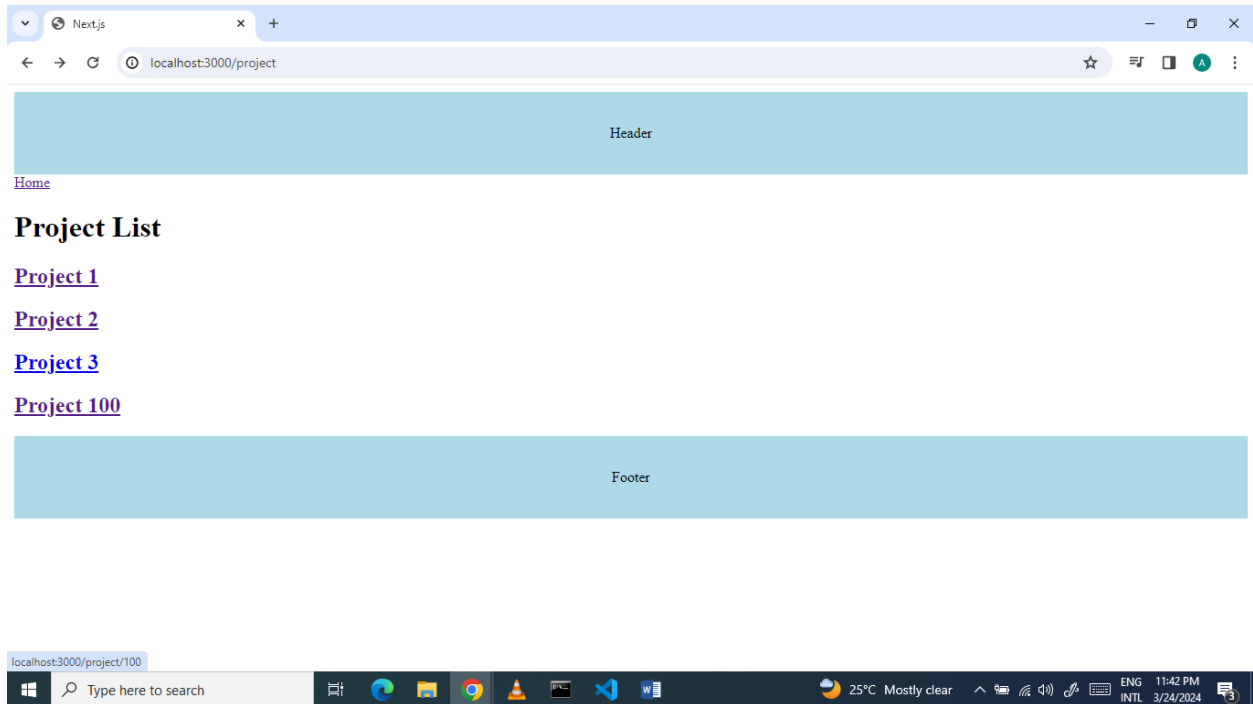
"In the page.tsx file inside the project folder, add the replace feature in project 2 heading."



```
1 import Link from "next/link";
2 export default function ProjectList(){
3   const projectid =100
4   return(
5     <>
6     <Link href="/"> Home </Link>
7     <h1>Project List</h1>
8     <h2>
9       <Link href="project/1">Project 1</Link>
10    </h2>
11    <h2>
12      <Link href="project/2" replace>Project 2</Link>
13    </h2>
14    <h2>
15      <Link href="project/3">Project 3</Link>
16    </h2>
17    <h2>
18      <Link href={`project/${projectid}`}>Project {projectid}</Link>
19    </h2>
20    </>
21  )
22 }
```

We'll start from the main page, then go to a project, maybe 'project 1', and then click 'back'





When we click 'back' after 'project two', we end up back on the main page instead of the project list page.

