

Handling Error in Next.js

In Next.js, several special files play key roles in managing the application's behavior and user experience. These files have already been explained one by one in a previous guide. Their names are `page.tsx`, `layout.tsx`, `template.tsx`, `notfound.tsx`, and `loading.tsx`. Now, I will explain `error.tsx`.

error.tsx:

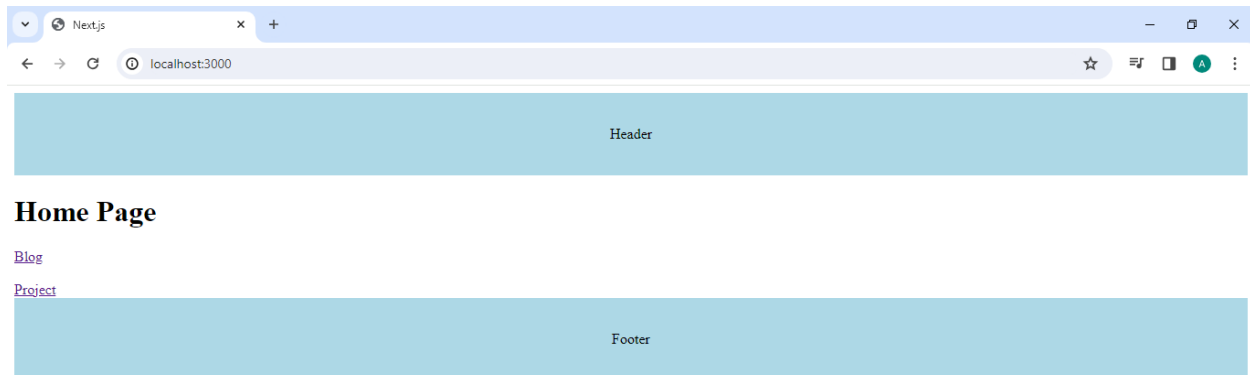
The `error.tsx` file in Next.js allows you to automatically handle errors that occur within a specific part of your website, called a route segment, and its nested components.

You can create custom error user interfaces (UI) for different segments of your website by organizing files in your project's folder structure. This helps you to pinpoint and deal with errors only in the affected segments while keeping the rest of your website running smoothly.

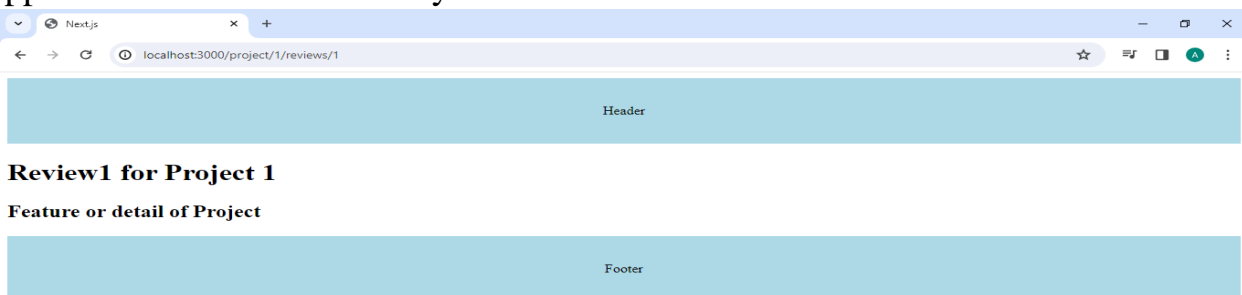
Moreover, with `error.tsx`, you can try to recover from errors without having to reload the entire webpage. This means you can provide users with a better experience by attempting to fix issues seamlessly within the affected segment of your website.

" Handling Error in Next.js : Step-by-Step Guide with Screenshots"

"Run the command `npm run dev` and navigate to `localhost:3000` in your browser. This will open the home page of our application."



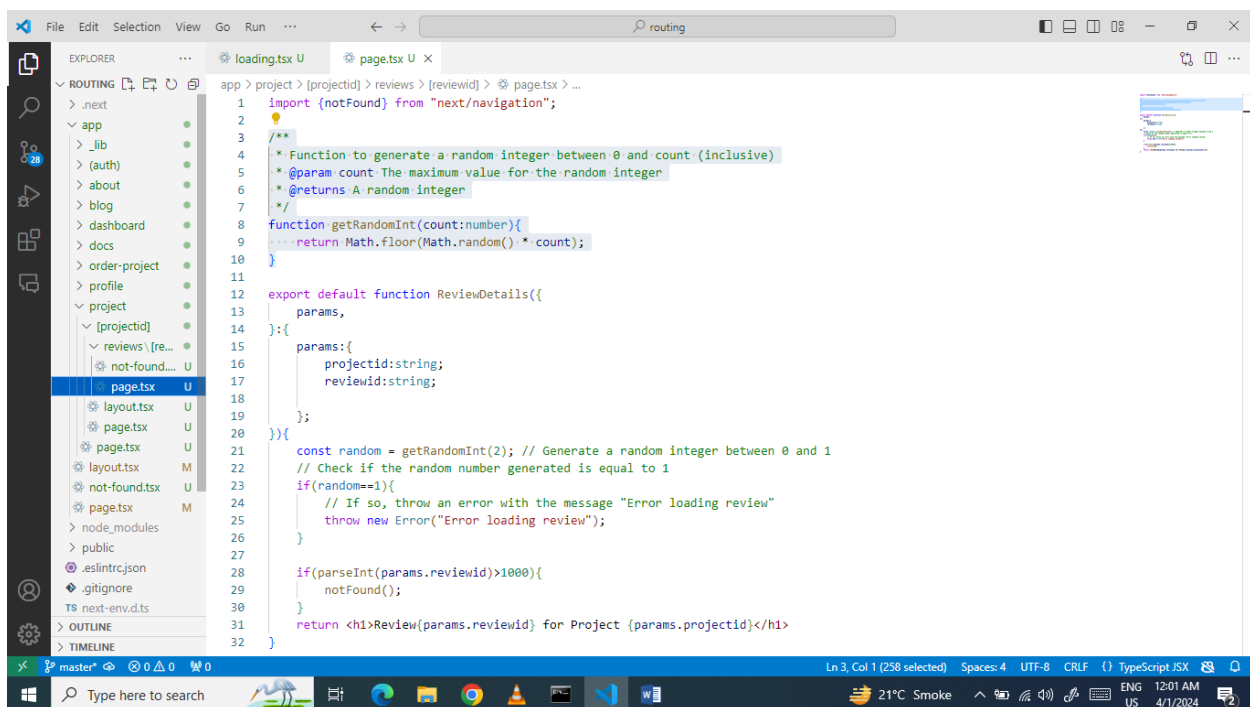
Now, in the browser, navigate to `localhost:3000/project/1/reviews/1`. Our application functions smoothly when we visit this URL.



When accessing our application's URL, it accurately displays the first review for the initial project. This particular URL is linked to a file named `page.tsx` located within the review ID folder, ensuring precise rendering in the browser.

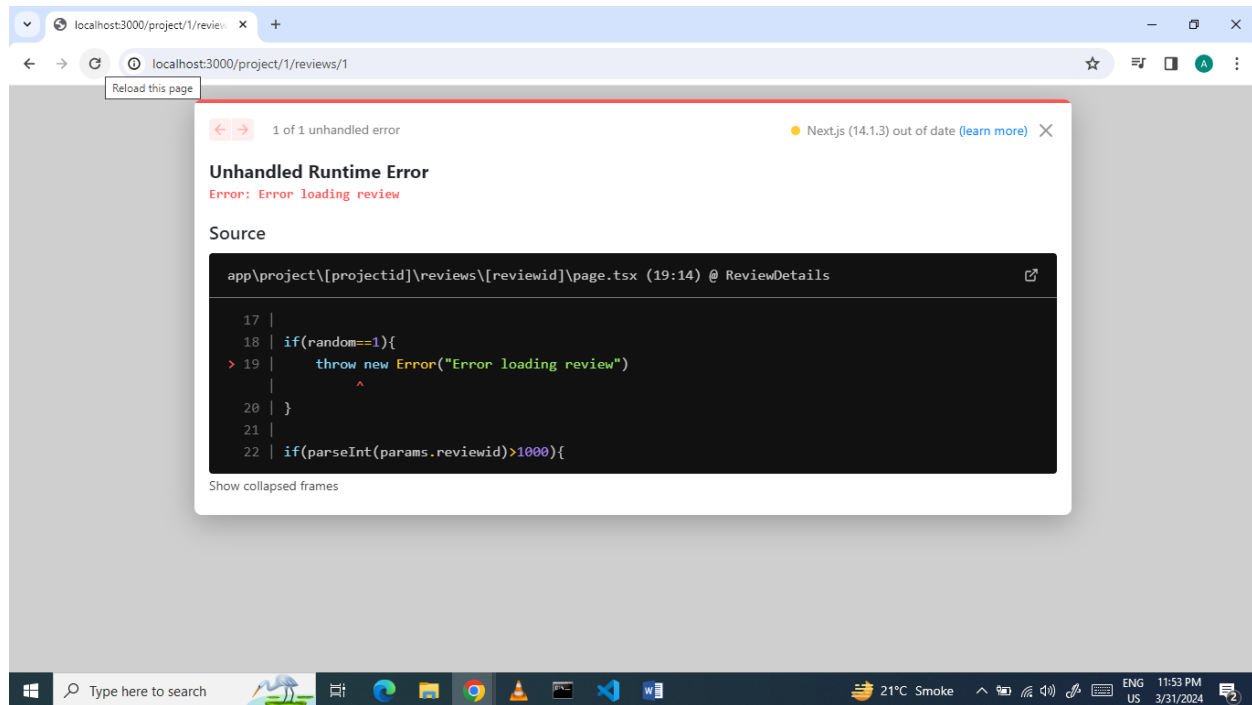
However, errors are bound to happen. For example, a network request might fail when trying to fetch a review. Let's create a scenario where an error occurs within our review ID page

Open the page.tsx file located inside the review ID folder and update the code according to the highlighted and comments provided in the screenshots below.



```
1 import {notFound} from "next/navigation";
2
3 /**
4  * Function to generate a random integer between 0 and count (inclusive)
5  * @param count The maximum value for the random integer
6  * @returns A random integer
7  */
8 function getRandomInt(count:number){
9   return Math.floor(Math.random()*count);
10 }
11
12 export default function ReviewDetails({
13   params,
14 }):{
15   params:{
16     projectid:string;
17     reviewid:string;
18   };
19 }
20
21 const random = getRandomInt(2); // Generate a random integer between 0 and 1
22 // Check if the random number generated is equal to 1
23 if(random==1){
24   // If so, throw an error with the message "Error loading review"
25   throw new Error("Error loading review");
26 }
27
28 if(parseInt(params.reviewid)>1000){
29   notFound();
30 }
31 return <h1>Review{params.reviewid} for Project {params.projectid}</h1>
32 }
```

The function mentioned above will result in either 0 or 1, which is then stored in a constant named "random". If the outcome is 1, we throw a new error with the message "Error loading review", similar to the screenshot provided below from the browser.



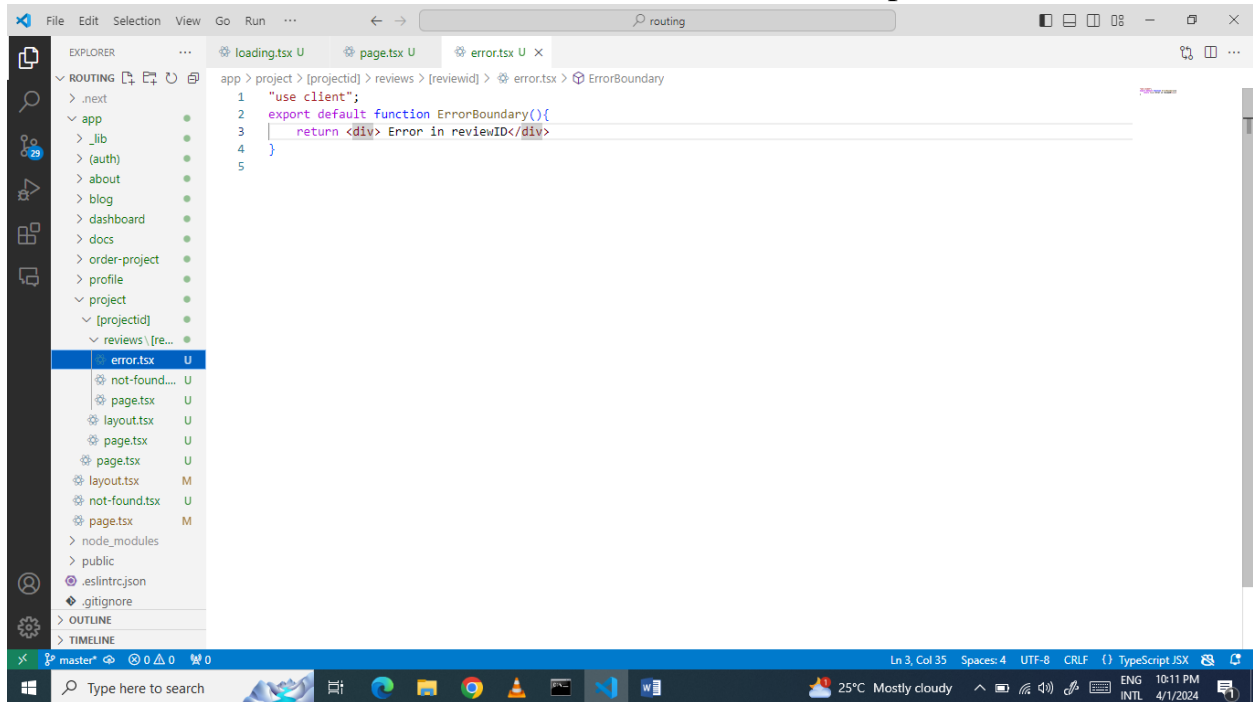
upon refreshing the page multiple times in the browser, an error is triggered when the generated random number equals 1. This error is not handled at runtime, and the displayed message is "Error loading review." Such occurrences are common during development mode.

In above given screenshot encountering a problem while using our app. When we're working on it and something goes wrong, we might see a basic error message like "There's a problem with the app - something didn't work on the server." But that message doesn't really explain what happened or what we should do next.

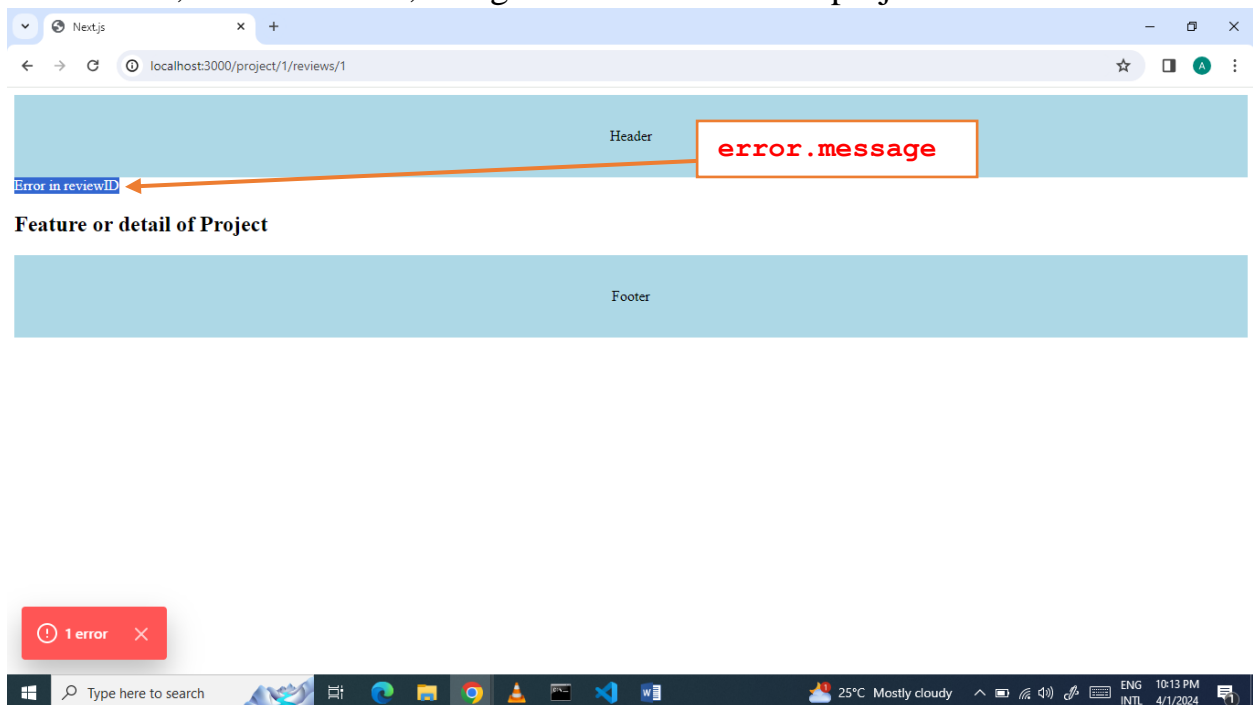
And sometimes, if the problem is deep inside the app, it can mess up everything, which is definitely not good.

To handle these issues better and make sure only the part of the app that's having trouble gets affected, we have a special file in Next.js called `error.tsx`. It's like a detective that helps us find out what's wrong and how to fix it without breaking the rest of the app. This file sits in the folder related to the part of the app where the problem occurred. For example, let's say we have a part of the app called `review`, and something goes wrong there.

Create error.tsx file inside review folder and add react component.



Now, in the browser, navigate to localhost:3000/project/1/reviews/1.

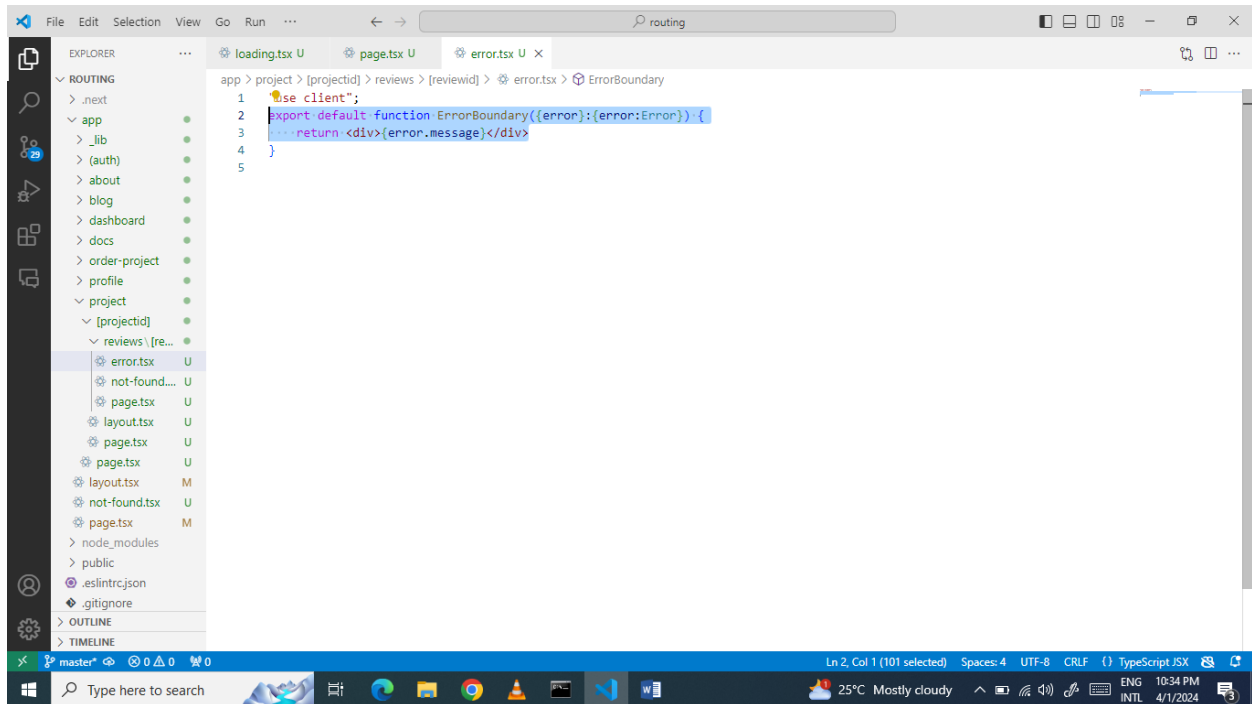


In the above screenshot, when encountering an error during app usage, developers now receive a precise error message pinpointing the issue with the

segment linked to the review ID, instead of receiving a generic one. The advantageous aspect is that only this specific part of the app is impacted, while the remainder continues to function normally.

Ideally, this error message would integrate better with the application's design. The `error.tsx` component can also be improved to provide more detailed information about the error, assisting users and developers in understanding what occurred.

Open the `error.tsx` file and update it according to the screenshot provided. Replace hard-coded text with `error.message`. Additionally, ensure that this component receives an error object as a prop to display more information. You can find this message within our component (page.tsx file resides inside the review folder, which was discussed at the beginning of this guide.).

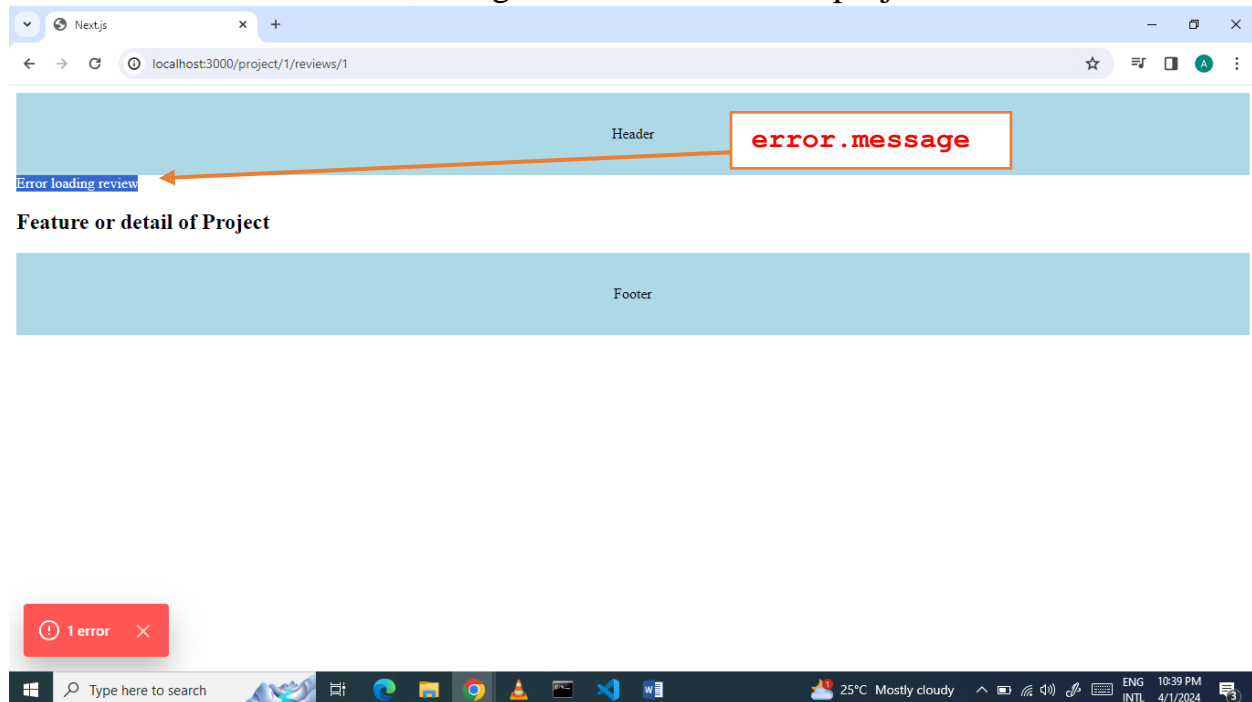


```
File Edit Selection View Go Run ... routing
EXPLORER
  ROUTING
  .next
  app
  _lib
  (auth)
  about
  blog
  dashboard
  docs
  order-project
  profile
  project
  [projectid]
    reviews [re...]
      error.tsx U
      not-found... U
      page.tsx U
      layout.tsx U
      page.tsx U
      layout.tsx M
      not-found.tsx U
      page.tsx M
  node_modules
  public
  .eslintrc.json
  .gitignore
  OUTLINE
  TIMELINE

app > project > [projectid] > reviews > [reviewid] > error.tsx > ErrorBoundary
1 use client;
2 export default function ErrorBoundary({error}:{error:Error}) {
3   return <div>{error.message}</div>
4 }
5

Ln 2, Col 1 (101 selected) Spaces: 4 UTF-8 CRLF TypeScript JSX
25°C Mostly cloudy 10:34 PM 4/1/2024
```

Now, in the browser, navigate to `localhost:3000/project/1/reviews/1`.

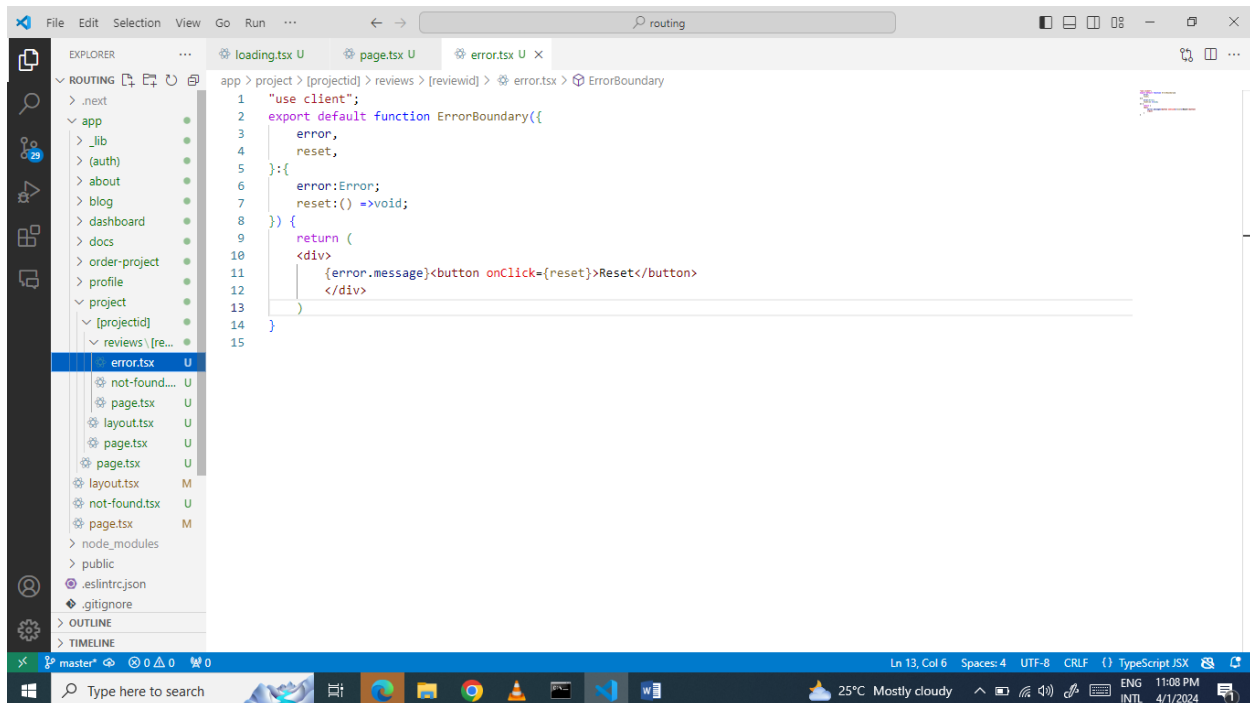


In summary, the ``error.tsx`` file automatically encapsulates a route segment and its nested components within a React error boundary. This functionality permits programmers to design custom error user interfaces for specific segments of the application, confining errors to affected areas while ensuring the continued functionality of the rest of the application. Moreover, it empowers developers to attempt error recovery without the necessity of reloading the entire page.

"Recovering from error in Next.js : Step-by-Step Guide with Screenshots"

The error boundary, defined within "error.tsx," offers a crucial feature: a reset function. Through object destructuring alongside the error object, developers gain access to this reset function. Additionally, we specify the function type within the tsx. By integrating a button within our tsx, labeled "Reset," that triggers the execution of the reset function upon clicking, we establish a mechanism for attempting to render the component once more on the page.

Open the "error.tsx" file and update it according to the screenshot of VS Code below.



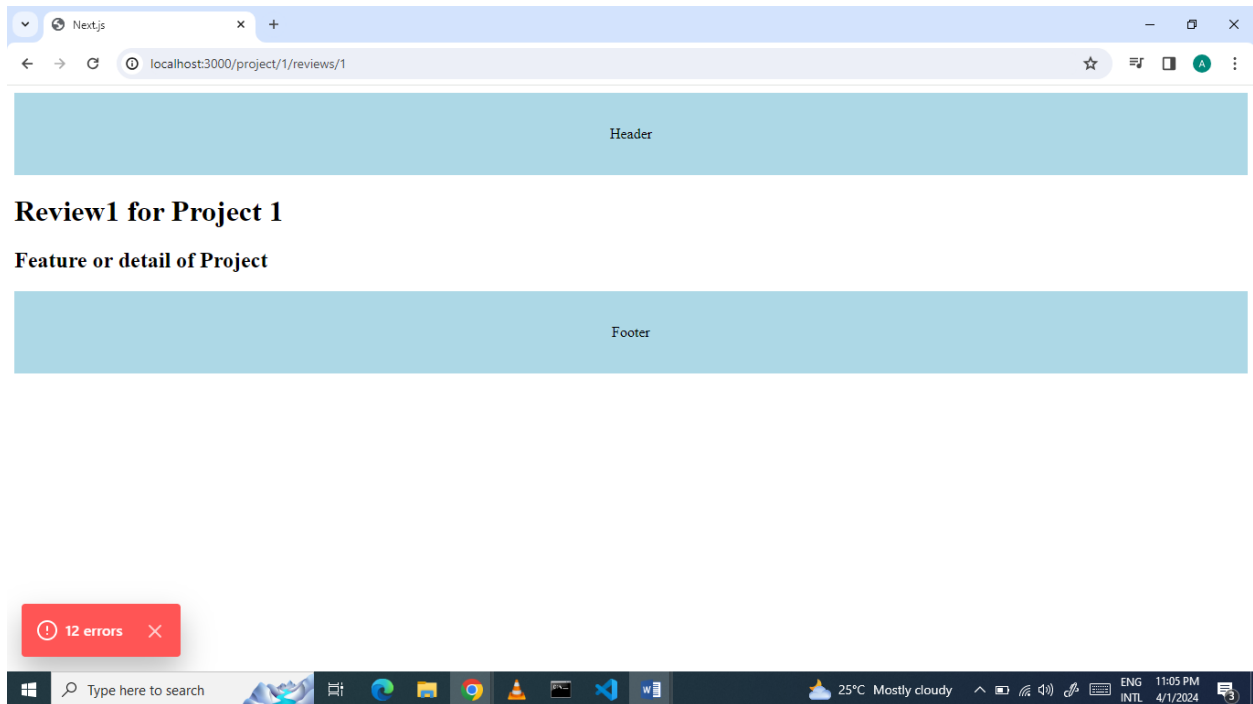
```
1 "use client";
2 export default function ErrorBoundary({
3   error,
4   reset,
5 }): {
6   error: Error;
7   reset: () => void;
8 } {
9   return (
10    <div>
11      {error.message}<button onClick={reset}>Reset</button>
12    </div>
13  )
14 }
15
```

For client-side recovery assurance, it's necessary to convert our page from "page.tsx" to a client component. This transition is facilitated by including "use client" at the beginning of the file. Upon reloading the browser and encountering an error, developers will notice a "reset" button positioned alongside the error message. Clicking this button triggers the re-rendering of the review ID component, effectively resolving the error and loading the intended page content.

The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows a project structure with a 'project' folder containing 'reviews' and 'error.tsx'. The 'page.tsx' file is selected in the editor. The code in 'page.tsx' is as follows:

```
1  "use client";
2  import {notFound} from "next/navigation";
3
4  /**
5   * Function to generate a random integer between 0 and count (inclusive)
6   * @param count The maximum value for the random integer
7   * @returns A random integer
8   */
9  function getRandomInt(count:number){
10     return Math.floor(Math.random() * count);
11 }
12
13 export default function ReviewDetails({
14   params,
15 }):{
16   params:{
17     projectId:string;
18     reviewId:string;
19   };
20 }
21
22 const random = getRandomInt(2); // Generate a random integer between 0 and 1
23 // Check if the random number generated is equal to 1
24 if(random==1){
25   // If so, throw an error with the message "Error loading review"
26   throw new Error("Error loading review");
27 }
28
29 if(parseInt(params.reviewId)>1000){
30   notFound();
31 }
32 return <h1>Review{params.reviewId} for Project {params.projectId}</h1>
```

Now, in the browser, navigate to localhost:3000/project/1/reviews/1.



Refresh the browser, and you will see the reset button along with the error message.
By clicking this button, the browser will reset.

