CodeyOverflow Forum

In this project, you'll practice your knowledge of the App Router in Next.js. You will be building the CodeyOverflow Forum application, which users will use to view topics, questions, and user information. Currently, many paths aren't working properly and the application itself does not start. To fix this, you will need to:

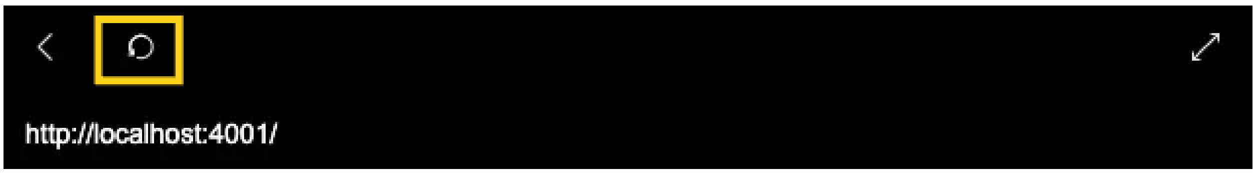
- enable the App Router
- define the /topics route
- define a dynamic nested segment in /topics
- optimize links
- use shared layouts
- handle errors

Before you start coding, take a look at the files currently present in the workspace. At the root level, you'll notice a /components folder containing reusable React components, a /data folder containing the application data, and a /lib folder containing some utility functions. In the /app folder you'll see:

- the root page.tsx file
- theroute groups (account), (discussion), (info)
- a /about path defined
- a /questions path defined
- a /questions/[id] dynamic path defined.
- a /users path defined
- a /users/[userId] dynamic path defined

Once you begin, you can use npm run dev in the workspace terminal to start the Next.js application.

How do I start the development server?


http://localhost:4001/

Tasks

20/20 complete

Mark the tasks as complete by checking them off

Routing Setup

1.

Currently, the App Router is missing a root layout.

Begin by creating the root layout file and adding the following imports to the file:

```
import './globals.css'
import Nav from '../components/navigation/Nav'
import UrlBar from '../lib/UrlBar/UrlBar'
```

Hint

Make sure to create the layout file at the root of the App Router.

2.

Next, in your root layout, default export a React component called RootLayout.

Hint

Make sure you default export a React component with a children prop.

3.

Finally, return the imported <UrlBar> and <Nav> elements followed by children.

Once complete, you can start the dev server and you should be able to see your (limited) forum application with access to the following routes:

- /.
- /about.
- /users.
- /questions

Note the <UrlBar> element is a utility component you can use to change the URL path.

Hint

Make sure you return the <UrlBar> and <Nav> elements followed by children within the <html> and <body> elements.

Routing Topics

4.

When you click on "Topics" in the header (or navigate to the /topics path) you'll notice that we get a 404 error.

Let’s begin to fix this by creating the /topics path in the (discussion) route group, which helps you organize segments without affecting the URL path structure.

Hint

Recall that Next.js uses file-system based routing.

5.

Next, make the /topics path accessible by setting up the appropriate folder structure.

For the path, import the following:

```
import Button from '../../../components/button/Button'
import { CATEGORIES } from '../../../data/postData'
import { capitalize } from '../../../lib/utils'
```

Then, add the following code to the path component.

```
<>
  <h1>All Topics</h1>
  <ul>
    {CATEGORIES.map((category) => (
      <li key={category}>
        <Button
          href={`/topics/${category}`}
          label={capitalize(category)}
        ></Button>
      </li>
    ))}
  </ul>
</>
```

Hint

Recall that a path segment is made accessible using a page.tsx file and default exporting a React component.

6.

Notice that clicking a topic category leads to another 404 error.

To fix this, begin by creating a nested dynamic segment called topicId in the /topics segment.

Hint

A dynamic segment is created with square brackets [] around the folder name.

7.

Make this dynamic segment accessible by setting up the appropriate folder structure.

For the path, import the following:

```
'use client'
import React, { useEffect, useState } from 'react'
import Button from '../../../../components/button/Button'
import { capitalize, fetchPostByTopic, Post } from '../../../../lib/utils'
```

Then, add the following code to the component:

```
const topicId = 'REPLACE ME'
const [topicPosts, setTopicPosts] = useState<Post[]>([])
useEffect(() => {
  fetchPostByTopic(topicId).then((fetchedPosts) => {
    setTopicPosts(fetchedPosts)
  })
}, [topicId])
return (
  <div>
    <h1>{capitalize(topicId)}</h1>
    <ul>
      {topicPosts.map((post) => (
        <li key={post.id}>
          <Button
            href={`/questions/${post.id}`}
            label={post.title}
          ></Button>
        </li>
      ))}
    </ul>
  </div>
)
```

Hint

Make sure to default export a React component and add the starter code provided.

8.

Clicking on a topic no longer results in a 404 error, but no related questions are found for the topic.

In your dynamic segment component, fix the topicId constant so it correctly gets the data from the dynamic segment.

If successful, you should be able to see questions related to the topic category selected.

Hint

Make sure you use the params object passed into the React component as a prop to retrieve the topicId dynamic segment data.

9.

Wrap up /topics/topidId by adding a shared layout that will include a “Back To All Topics” button when you navigate to a specific topic.

For the path, import the following:

```
import Button from '../../../../components/button/Button'
```

Then, add the following code to the component:

```
<div>
  {/* REPLACE ME */}
  <footer>
    <Button href="/topics" label="Back To All Topics"></Button>
  </footer>
</div>
```

Hint

Make sure to default export a React component that receives a children prop.

10.

Finally, in the shared component you just created, replace the {/* REPLACE ME */} comment with the content in page.tsx.

Hint

Make sure you return children in your shared layout.

Using Links

11.

Notice when looking at /questions, clicking on a post causes an error.

Fix this in components/post/Post.tsx by updating the href props with the values of /questions/REPLACE and /users/REPLACE with dynamic links using id and userId respectively.

Hint

A dynamic link can be created using template literals.

12.

After fixing the dynamic links, you'll notice clicking on a post no longer results in an error but your browser refreshes on navigation.

Fix this by replacing all the <a> elements with <Link> elements.

Hint

Make sure to import Link from next/link.

13.

Finally, notice the header links don't indicate which is currently active.

To fix this in components/navigation/Nav.tsx, begin by replacing pathname with the current path in the browser.

Hint

Make sure to use the Next.js hook that returns the current path.

14.

Finally, apply the active class to each <li> element when the href matches the current path.

Hint

Make sure to use the styles object to apply the active class.

Showing Last Updated Time

15.

If you navigate to a specific user like /users/1 there is no indication of how stale the user data is.

Create a shared UI in (account)/users/[userId] that will be re-instantiated on nested segment navigation and default exports a client component.

Hint

Make sure to default export a React component that receives a children prop and use the use client directive.

16.

Finally, return the following code and update the {/* REPLACE ME */} comment to render the page.tsx content passed in as a child.

```
const requestTime = new Date()
return (
  <div>
    {/* REPLACE ME */}
    <footer>Last Checked: {requestTime.toLocaleTimeString()}</footer>
  </div>
)
```

If successful, you should be able to see the time (in UTC) the user data was last loaded.

Note: A hydration error may appear. This error is typically due to a discrepancy in the time generated on the server versus the time generated on the client during the page's hydration process. It's safe to disregard this error.

Hint

Make sure you replace {/* REPLACE ME */} with children.

Handling Errors

17.

The forum application is almost complete. If you navigate to /questions/101 you'll notice the application crashes because the question does not exist.

Handle this error in (discussion)/questions/[id] by creating the Next.js reserved file that wraps the content in an <ErrorBoundary> and default export a React component.

Hint

Make sure you default export a React component that receives an error object of type Error.

18.

Next, import the following:

```
'use client'
import Button from '../../../../components/button/Button'
import { useEffect } from 'react'
```

Then, add the following code to the component:

```
useEffect(() => {
 console.error(error)
}, [error])
const goHome= () => {
 // Fix me
}
const goToAllQuestions = () => {
 // Fix me
}
return (
 <div>
  <h2>This Question Does Not Exist!</h2>
```

```
    <Button onClick={goHome} label="Return Home" />
    <Button onClick={goToAllQuestions} label="All Questions" />
  </div>
)
```

If successful, when navigating to /questions/34326, you should see the fallback UI with “Return Home” and “All Questions” buttons.

Hint

Make sure your component receives an error prop.

19.

Notice the “Return Home” and the “All Questions” buttons do not work.

Fix this so that when a user clicks on the “Return Home” button they are taken to / and when clicking on “All Questions” they are taken to /questions.

If successful, both buttons should work as expected.

Hint

Make sure to use the router object from Next.js.

Conclusion

20.

In this project, you practiced Next.js routing by creating a forum webpage where users can navigate to pages for user profiles, topics, and comments. Great work!

Feel free to take a look at our solution by navigating to the project walkthrough video in the **Get Unstuck** section.