Dynamic Routes

29 min

Previously, we've seen how we can create nested routes to define unique UIs based on the URL path. What if our path is displaying content relative to a specific user? For example, if our application displays the relevant user information depending on the user ID the content is for, should we define path segments for all possible IDs like /users/10 or /users/20?

The answer is *no*. This will not only be tedious as the path for each user will mostly contain the same UI, but it lacks scalability as the user base grows. To handle situations like this, we can use **dynamic segments**. Dynamic segments are dynamic portions of a URL that may change (like /users/10 and /users/20).

To define a dynamic segment in Next.js:

- Create an accessible segment (folder) as we've learned in the previous exercise.
- The folder name for a dynamic segment *must* be wrapped in square brackets ([]).
- The folder name given will serve as the identifier we use to retrieve the dynamic segment in our page.tsx.

For example, the following folder structure will display content for paths like /users/10 and /users/20:

```
├── app
|   ├── users
|   |   ├── [userId]
|   |   |   ├── page.tsx
```

We access the dynamic segment data by referencing the folder name as a property of the params prop in the page.tsx component. Take a look at the following example:

```
// destructure params which contain our identifier as a property
export default function MyUserPage( { params } : { params: { userId: string }}) {
  return (
    <h1>Greetings user: {params.userId}</h1>
  )
}
```

In this example:

- We destructure a params prop.
- We define the params type as an object containing the property userId.

- params properties are always of type string (or string[] — we'll see this later in the lesson).
- We access the dynamic segment data by using the params object and accessing the userId property.

As we work with dynamic segments, we may find that we'd like to store some analytics about the number of users that have navigated to the page. This sounds like a great place to use a shared UI in a layout.tsx file, but, because layouts don't rerender on nested segment navigation, we won't be able to rerun our API call.

To address this issue, we can use the template.tsx reserved file. template.tsx is similar to layout.tsx in that:

- template.tsx also defines a sharable UI for its nested segments.
- It must default export a React component.
- The default exported component receives a children prop.

But differs in that it gets *re-instansiated* on nested segment navigation (we'll learn more about navigation next). We can take advantage of this feature by calling an API to update our user's visited counter like:

```
// import statements

// re-instantiated on nested segment navigation
export default function MyTemplate({children}: {children : ReactNode}) {
  useEffect(() => {
    updateUsersCounter()  // API call
  }, [])  // runs on mount
  // other logic for MyTemplate
}
```

Let's practice adding a dynamic path and using template.tsx in our application.

Instructions

1. Checkpoint 1 Passed

1.

Currently, if you navigate to /categories and try to click on a category link you'll see a not-found (404) error.

To begin fixing this, create a nested dynamic segment in the /categories segment called name and create a page.tsx file. Then, add the following imports at the top of the file:

```
'use client'

import { CATEGORIES, ARTICLES } from '../../../store/data'
import Link from 'next/link'
```

Hint

Make sure to use square brackets ([]) in your folder name to define a dynamic segment.

2. Checkpoint 2 Passed

2.

Next, make the dynamic segment accessible by default exporting a React component called CategoryPage that receives a params prop. Add the following code to the function body of CategoryPage.

```
const categoryName = "REPLACE ME"
return (
 <section>
  <h2>Articles related to {categoryName}</h2>
  <ul>
   {CATEGORIES[categoryName]?.map(articleSlug => (
    <li key={articleSlug}>
     <Link href={`/articles/${articleSlug}`}>{ARTICLES[articleSlug].title}</Link>
    </li>
   ))}
  </ul>
 </section>
)
```

Hint

Make sure you default export a React component that receives a params prop and your imports are *above* the CategoryPage component.

3. Checkpoint 3 Passed

3.

Replace the value of categoryName with the dynamic segment data.

Run npm run dev. Once “Ready” appears, click the refresh button in the workspace browser.

If successful, you should see a list of articles generated for the category name in the dynamic URL segment.

Hint

Make sure you use the params prop received by dynamic segment components. Use the folder name to reference the dynamic segment data in params.

Enter the command npm run dev inside the terminal window.

The window can be refreshed when it displays “Ready.”

```
> dev
> next dev -p 4001

  ▲ Next.js 14.0.1
  - Local:        http://localhost:4001

 ✓ Ready in 1156ms
```

4. Checkpoint 4 Passed

4.

When looking at categories you’ll notice the time displayed of when you visited the /categories path. Let’s give users an indication of the last time articles were fetched.

To start, in /categories, create a shared UI reserved file that will be re-instantiated on any nested segment navigation. Add the following imports at the top of the file:

```
'use client'
import { ReactNode } from 'react'
```

Hint

Make sure to use the Next.js reserved file similar to layout.tsx that also gets re-instantiated on nested segment navigation.

5. Checkpoint 5 Passed

5.

Finally, default export a React component called LastUpdatedCategories and the following code as the function’s body:

```
const updatedTime = new Date().toLocaleTimeString()

return (
  <section>
    <h4 suppressHydrationWarning>Articles as of: {updatedTime}</h4>
    {children}
  </section>
```

)

Run npm run dev. Once “Ready” appears, click the refresh button in the workspace browser.

If successful, you should be able to click on category links and see Articles as of: display the last time (in UTC) you clicked on a category link.

Notice that the time in layout.tsx doesn’t update, do you know why?

Hint

Recall that a template.tsx component is defined similarly to layout.tsx. It differs in that it is re-instantiated on nested segment navigation.

Enter the command npm run dev inside the terminal window.

The window can be refreshed when it displays “Ready.”

```
> dev
> next dev -p 4001

  ▲ Next.js 14.0.1
  - Local:        http://localhost:4001

 ✓ Ready in 1156ms
```

page.tsx

```
'use client'


import { CATEGORIES, ARTICLES } from '../../../store/data'

import Link from 'next/link'


export default function CategoryPage( { params } : {

  params: { name: string }}) {

  const categoryName = params.name

  return (

    <section>

      <h2>Articles related to {categoryName}</h2>

      <ul>
```

```
        {CATEGORIES[categoryName]?.map(articleSlug => (
          <li key={articleSlug}>
            <Link href={`/articles/${articleSlug}`}>{ARTICLES[articleSlug].title}</Link>
          </li>
        ))}
      </ul>
    </section>
  )
}
```

template.tsx

```
'use client'
import { ReactNode } from 'react'

export default function LastUpdatedCategories({children}: {children: ReactNode}) {
  const updatedTime = new Date().toLocaleTimeString()
  return (
    <section>
      <h4 suppressHydrationWarning>Articles as of: {updatedTime}</h4>
      {children}
    </section>
  )
}
```