Managing Unpredictable URLs

26 min

We've discussed how we can create dynamic segments and nested segments, but what if our application supports any number of the same dynamic segments? For example, if our application has a feature to compare usage details for its users, the URL paths may look like:

- /users/123: Show usage details for users with ID 123.
- /users/123/987: Show comparison of usage details for users with ID's 123 and 987.

How should we handle this? Should we use multiple dynamic segments like /users/[userId1]/[userId2]/...? Not quite. This approach will have the same scaling issue we solved by using dynamic segments. Instead, Next.js provides a **catch-all** segments solution.

Catch-all segments, as the name suggests, will match all paths like /users/123, and /users/123/987 using a single dynamic segment. To create one, we define our dynamic segment as we've done before (using a folder and []) but also add an ellipses (...) prefix to the dynamic segment name like:

```
├── app
|   ├── users
|   |   ├── [...userIds]  // catch-all dynamic segment
|   |   |   ├── page.tsx
```

To access the data in userIds, we similarly will receive the params in our page.tsx component, except that the type of our property will no longer be a string, but string[]. For example:

```
// destructure params which contain our identifier as property
export default function MyUserPage( { params } : { params: { userIds: string[] }}) {
const userIds = params.userIds   // access params property with our dynamic segment data.
  return (
    <section>
      {/* display data for all userIds*/}
      {userIds.map((userId) => (
        <UsageDetails key={userId} userId={userId}/>
      ))}
    </section>
  )
}
```

This example will retrieve all userIds from the catch-all dynamic segment [...userIds] from a params property named userIds of type string[].

Before we wrap up dynamic segments, let's add a new feature where a user will be able to select the users to compare if they navigate to the base /users URL segment. Navigating to that path now will yield a 404 error.

One way we can fix this is by taking advantage of Next.js *optional* catch-all segments. Optional catch-all segments, as the name suggests, are optional and will match paths like: /users, /users/123, and /users/123/987. To create one, we wrap *another* pair of square brackets ([]) around our catch-all segment folder like [[...userIds]].

To access our data, we make a slight type modification to the type where the property is now optional, like:

```
// Destructure params which contain our identifier as property
export default function MyUserPage( { params } : { params: { userIds?: string[] }}) {
  // Using optional chaining
  // In path /users, `userIds` will not exist in the object
  const userIds = params?.userIds

  // ...body
}
```

Great job learning about dynamic catch-all segments. Let's practice using them by adding one to our application.

Instructions

1. Checkpoint 1 Passed

1.

Currently, the /articles path yields a 404 error. Ideally, the content in this path should display a list of article links and when a user clicks on an article link, the selected articles should render beneath the list.

To complete this functionality, create an /articles directory. Inside, create an *optional catch-all* segment called titles and create the following files:

- page.tsx
- page.module.css
- layout.tsx

Note: We'll complete these files next.

Hint

Make sure your folder name contains double square brackets ([[]]) and ellipses (...).

2. Checkpoint 2 Passed

2.

Next, add in the starter code for layout.tsx:

layout.tsx

```
'use client'
import { ARTICLES } from '../../../store/data'
import Link from 'next/link'
import { usePathname } from 'next/navigation'
import { ReactNode } from 'react'
export default function ArticlesLayout({ children }: { children: ReactNode }) {
  const pathname = usePathname()
  return (
    <>
      <section>
        <h1>Articles</h1>
        <ul>
          {Object.values(ARTICLES)
            .filter(article => !pathname.includes(article.slug))
            .map(article => (
              <li key={article.slug}>
                <Link href={`${pathname}/${article.slug}`}>{article.title}</Link>
              </li>
            ))}
        </ul>
      </section>
      {children}
  </>
  )
}
```

Then, for page.module.css:

page.module.css

```
.articlesContainer {
  margin-top: 1rem;
}
.title~p {
  margin-top: 0;
  margin-bottom: 1rem;
}
.notFound {
  margin: 2, 5rem;
```

```
}
```

And finally, for page.tsx:

page.tsx

```
import { ARTICLES } from '../../../store/data'
import Link from 'next/link'
import ReactMarkdown from 'react-markdown'
import styles from './page.module.css'
export default function ArticlePage({ params }: { params: { titles?: string[] } }) {
  const foundArticles = []
  return (
    <section className={styles.articlesContainer}>
      {foundArticles?.map((article, idx) => (
        article ?
          (<article key={article.slug}>
            <h1 className={styles.title}>{article.title}</h1>
            <p>By <Link href={`/authors/${article.author}`}>{article.author}</Link></p>
            <ReactMarkdown>{article.body}</ReactMarkdown>
          </article>)
          : (<h1 className={styles.notFound}>No article found for with title:
{params.titles?.[idx]}</h1>))}
      {(!foundArticles || foundArticles.length === 0) && <h1>No articles to display...</h1>}
  </section>
  )
}
```

Run the code using npm run dev in the terminal. Once you see a console message indicating “Ready,” click the Refresh Browser button in the workspace browser bar.

If successful, you should see a list of article links displayed.

Note that layout.tsx will contain a shared UI with all the available article links. page.tsx will display the individual article content for all articles.

Hint

layout.tsx is a reserved file used to create a shared UI and the export component will receive a children prop of type ReactNode.

page.tsx is a reserved file used to create a unique and accessible UI for a path segment.

Enter the command 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
```

3. Checkpoint 3 Passed

3.

In page.tsx, fix the value of the constant name foundArticles. The new value should be a list of article objects with the same title as the catch-all segment data.

Use the imported ARTICLES object, where the key is the article title and value is the article object.

Run the code using npm run dev in the terminal. Once you see a console message indicating “Ready,” click the Refresh Browser button in the workspace browser bar.

If successful, you should be able to click on articles and see the article contents displayed below each other.

Hint

Make sure to .map() over the list of titles in params to retrieve the article object with the matching title in ARTICLE.

Enter the command 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

```
import { ARTICLES } from '../../../store/data'
import Link from 'next/link'
import ReactMarkdown from 'react-markdown'
import styles from './page.module.css'
export default function ArticlePage({ params }: { params: { titles?: string[] } }) {
  const foundArticles = params.titles?.map(title => ARTICLES[title]) || [];

  return (
    <section className={styles.articlesContainer}>
      {foundArticles?.map((article, idx) => (
        article ?
          (<article key={article.slug}>
            <h1 className={styles.title}>{article.title}</h1>
            <p>By <Link href={`/authors/${article.author}`}>{article.author}</Link></p>
            <ReactMarkdown>{article.body}</ReactMarkdown>
          </article>)
          : (<h1 className={styles.notFound}>No article found for with title:
{params.titles?.[idx]}</h1>))}
      {(!foundArticles || foundArticles.length === 0) && <h1>No articles to display...</h1>}
  </section>
  )
}
```

page.module.css

```
.articlesContainer {
   margin-top: 1rem;
 }
 .title~p {
   margin-top: 0;
   margin-bottom: 1rem;
 }
 .notFound {
   margin: 2, 5rem;
 }
```

layout.tsx

```
'use client'
import { ARTICLES } from '../../../store/data'
import Link from 'next/link'
import { usePathname } from 'next/navigation'
import { ReactNode } from 'react'
export default function ArticlesLayout({ children }: { children: ReactNode }) {
 const pathname = usePathname()
 return (
   <>
     <section>
       <h1>Articles</h1>
       <ul>
         {Object.values(ARTICLES)
           .filter(article => !pathname.includes(article.slug))
           .map(article => (
             <li key={article.slug}>
```

```
            <Link href={`${pathname}/${article.slug}`}>{article.title}</Link>
          </li>
        ))}
      </ul>
    </section>
    {children}
  </>
 )
}
```