Tutorial: Creating custom route matches

The Angular Router supports a powerful matching strategy that you can use to help users navigate your application. This matching strategy supports static routes, variable routes with parameters, wildcard routes, and so on. Also, build your own custom pattern matching for situations in which the URLs are more complicated.

In this tutorial, you'll build a custom route matcher using Angular's `UrlMatcher`. This matcher looks for a Twitter handle in the URL.

For a working example of the final version of this tutorial, see the .

Objectives

Implement Angular's `UrlMatcher` to create a custom route matcher.

Prerequisites

To complete this tutorial, you should have a basic understanding of the following concepts:

- JavaScript
- HTML
- CSS
- Angular CLI

If you are unfamiliar with how Angular's router works, review Using Angular routes in a single-page application.

Create a sample application

Using the Angular CLI, create a new application, *angular-custom-route-match*. In addition to the default Angular application framework, you will also create a *profile* component.

1. Create a new Angular project, *angular-custom-route-match*.

   ng new angular-custom-route-match
   When prompted with `Would you like to add Angular routing?`, select `Y`.

   When prompted with `Which stylesheet format would you like to use?`, select `CSS`.

   After a few moments, a new project, `angular-custom-route-match`, is ready.

2. From your terminal, navigate to the `angular-custom-route-match` directory.

3. Create a component, *profile*.

ng generate component profile

1. In your code editor, locate the file, `profile.component.html` and replace the placeholder content with the following HTML.

2. In your code editor, locate the file, `app.component.html` and replace the placeholder content with the following HTML.

Configure your routes for your application

With your application framework in place, you next need to add routing capabilities to the `app.module.ts` file. As a part of this process, you will create a custom URL matcher that looks for a Twitter handle in the URL. This handle is identified by a preceding `@` symbol.

1. In your code editor, open your `app.module.ts` file.
2. Add an `import` statement for Angular's `RouterModule` and `UrlMatcher`.
3. In the imports array, add a `RouterModule.forRoot([])` statement.
4. Define the custom route matcher by adding the following code to the `RouterModule.forRoot()` statement.

This custom matcher is a function that performs the following tasks:

- The matcher verifies that the array contains only one segment.
- The matcher employs a regular expression to ensure that the format of the username is a match.
- If there is a match, the function returns the entire URL, defining a `username` route parameter as a substring of the path.
- If there isn't a match, the function returns null and the router continues to look for other routes that match the URL.

A custom URL matcher behaves like any other route definition. Define child routes or lazy loaded routes as you would with any other route.

Subscribe to the route parameters

With the custom matcher in place, you now need to subscribe to the route parameters in the `profile` component.

1. In your code editor, open your `profile.component.ts` file.
2. Add an `import` statement for Angular's `ActivatedRoute` and `ParamMap`.
3. Add an `import` statement for RxJS `map`.
4. Subscribe to the `username` route parameter.
5. Inject the `ActivatedRoute` into the component's constructor.

Test your custom URL matcher

With your code in place, you can now test your custom URL matcher.

1. From a terminal window, run the `ng serve` command.

ng serve

1. Open a browser to `http://localhost:4200`.

   You should see a single web page, consisting of a sentence that reads `Navigate to my profile`.

2. Click the **my profile** hyperlink.

   A new sentence, reading `Hello, Angular!` appears on the page.

Next steps

Pattern matching with the Angular Router provides you with a lot of flexibility when you have dynamic URLs in your application. To learn more about the Angular Router, see the following topics:

- In-app Routing and Navigation
- Router API

This content is based on Custom Route Matching with the Angular Router, by Brandon Roberts.