Using Angular routes in a single-page application

This tutorial describes how to build a single-page application, SPA that uses multiple Angular routes.

In a Single Page Application (SPA), all of your application's functions exist in a single HTML page. As users access your application's features, the browser needs to render only the parts that matter to the user, instead of loading a new page. This pattern can significantly improve your application's user experience.

To define how users navigate through your application, you use routes. Add routes to define how users navigate from one part of your application to another. You can also configure routes to guard against unexpected or unauthorized behavior.

To explore a sample application featuring the contents of this tutorial, see the .

Objectives

- Organize a sample application's features into modules.
- Define how to navigate to a component.
- Pass information to a component using a parameter.
- Structure routes by nesting several routes.
- Check whether users can access a route.
- Control whether the application can discard unsaved changes.
- Improve performance by pre-fetching route data and lazy loading feature modules.
- Require specific criteria to load components.

Prerequisites

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

- JavaScript
- HTML
- CSS
- Angular CLI

You might find the Tour of Heroes tutorial helpful, but it is not required.

Create a sample application

Using the Angular CLI, create a new application, *angular-router-sample*. This application will have two components: *crisis-list* and *heroes-list*.

1. Create a new Angular project, *angular-router-sample*.

   ng new angular-router-sample

When prompted with `Would you like to add Angular routing?`, select `N`.

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

After a few moments, a new project, `angular-router-sample`, is ready.

2. From your terminal, navigate to the `angular-router-sample` directory.
3. Create a component, *crisis-list*.

ng generate component crisis-list

1. In your code editor, locate the file, `crisis-list.component.html` and replace the placeholder content with the following HTML.
2. Create a second component, *heroes-list*.

ng generate component heroes-list

1. In your code editor, locate the file, `heroes-list.component.html` and replace the placeholder content with the following HTML.
2. In your code editor, open the file, `app.component.html` and replace its contents with the following HTML.
3. Verify that your new application runs as expected by running the `ng serve` command.

ng serve

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

   You should see a single web page, consisting of a title and the HTML of your two components.

Import `RouterModule` from `@angular/router`

Routing lets you display specific views of your application depending on the URL path. To add this functionality to your sample application, you need to update the `app.module.ts` file to use the module, `RouterModule`. You import this module from `@angular/router`.

1. From your code editor, open the `app.module.ts` file.
2. Add the following `import` statement.

Define your routes

In this section, you'll define two routes:

- The route `/crisis-center` opens the `crisis-center` component.
- The route `/heroes-list` opens the `heroes-list` component.

A route definition is a JavaScript object. Each route typically has two properties. The first property, `path`, is a string that specifies the URL path for the route. The second property, `component`, is a string that specifies what component your application should display for that path.

1. From your code editor, open the `app.module.ts` file.
2. Locate the `@NgModule()` section.
3. Replace the `imports` array in that section with the following.

This code adds the `RouterModule` to the `imports` array. Next, the code uses the `forRoot()` method of the `RouterModule` to define your two routes. This method takes an array of JavaScript objects, with each object defining the properties of a route. The `forRoot()` method ensures that your application only instantiates one `RouterModule`. For more information, see Singleton Services.

Update your component with `router-outlet`

At this point, you have defined two routes for your application. However, your application still has both the `crisis-list` and `heroes-list` components hard-coded in your `app.component.html` template. For your routes to work, you need to update your template to dynamically load a component based on the URL path.

To implement this functionality, you add the `router-outlet` directive to your template file.

1. From your code editor, open the `app.component.html` file.
2. Delete the following lines.
3. Add the `router-outlet` directive.

View your updated application in your browser. You should see only the application title. To view the `crisis-list` component, add `crisis-list` to the end of the path in your browser's address bar. For example:

http://localhost:4200/crisis-list

Notice that the `crisis-list` component displays. Angular is using the route you defined to dynamically load the component. You can load the `heroes-list` component the same way:

http://localhost:4200/heroes-list

Control navigation with UI elements

Currently, your application supports two routes. However, the only way to use those routes is for the user to manually type the path in the browser's address bar. In this section, you'll add two links that users can click to navigate between the `heroes-list` and `crisis-list` components. You'll also add some CSS

styles. While these styles are not required, they make it easier to identify the link for the currently-displayed component. You'll add that functionality in the next section.

1. Open the `app.component.html` file and add the following HTML below the title.

   This HTML uses an Angular directive, `routerLink`. This directive connects the routes you defined to your template files.

2. Open the `app.component.css` file and add the following styles.

If you view your application in the browser, you should see these two links. When you click on a link, the corresponding component appears.

Identify the active route

While users can navigate your application using the links you added in the previous section, they don't have a straightforward way to identify what the active route is. Add this functionality using Angular's `routerLinkActive` directive.

1. From your code editor, open the `app.component.html` file.
2. Update the anchor tags to include the `routerLinkActive` directive.

View your application again. As you click one of the buttons, the style for that button updates automatically, identifying the active component to the user. By adding the `routerLinkActive` directive, you inform your application to apply a specific CSS class to the active route. In this tutorial, that CSS class is `activebutton`, but you could use any class that you want.

Adding a redirect

In this step of the tutorial, you add a route that redirects the user to display the `/heroes-list` component.

1. From your code editor, open the `app.module.ts` file.
2. In the `imports` array, update the `RouterModule` section as follows.

   Notice that this new route uses an empty string as its path. In addition, it replaces the `component` property with two new ones:

   - `redirectTo`. This property instructs Angular to redirect from an empty path to the `heroes-list` path.
   - `pathMatch`. This property instructs Angular on how much of the URL to match. For this tutorial, you should set this property to `full`. This strategy is recommended when you have an empty string for a path. For more information about this property, see the Route API documentation.

Now when you open your application, it displays the `heroes-list` component by default.

Adding a 404 page

It is possible for a user to try to access a route that you have not defined. To account for this behavior, the best practice is to display a 404 page. In this section, you'll create a 404 page and update your route configuration to show that page for any unspecified routes.

1. From the terminal, create a new component, `PageNotFound`.

   ng generate component page-not-found

2. From your code editor, open the `page-not-found.component.html` file and replace its contents with the following HTML.

3. Open the `app.module.ts` file. In the `imports` array, update the `RouterModule` section as follows.

   The new route uses a path, `**`. This path is how Angular identifies a wildcard route. Any route that does not match an existing route in your configuration will use this route.

   Notice that the wildcard route is placed at the end of the array. The order of your routes is important, as Angular applies routes in order and uses the first match it finds.

Try navigating to a non-existing route on your application, such as `http://localhost:4200/powers`. This route doesn't match anything defined in your `app.module.ts` file. However, because you defined a wildcard route, the application automatically displays your `PageNotFound` component.

Next steps

At this point, you have a basic application that uses Angular's routing feature to change what components the user can see based on the URL address. You have extended these features to include a redirect, as well as a wildcard route to display a custom 404 page.

For more information about routing, see the following topics:

- In-app Routing and Navigation
- Router API