# Location Strategy in Angular

7 Comments / 7 minutes of reading / March 9, 2023

← Angular Router | Angular Tutorial | Passing Parameters to Route →

In this tutorial, we look at the Location Strategy (Angular Routing Strategy) supported by the Angular routing module. Angular Supports **PathlocationStrategy** and **HashLocationStrategy**. We will look at what is client-side routing and how it works. We take a look at how hash style HTML5 routing works and the pros and cons of each. Finally, we will learn how to build the Angular application using the PathLocationStrategy (HTML5 routing) HashLocationStrategy .(hash style Routing)

**Table of Contents**

## Location Strategies in Angular Router

Being a Single Page Application, the Angular applications should not send the URL to the server and should not reload the page, every time the user requests a new page.

The URLs are strictly local in Angular Apps. The Angular router navigates to the new component and renders its template and updates the history and URL for the view. All this happens locally in the browser.

There are two ways, by which Angular achieves this. These are called Location Strategies.

The Location Strategy defines how our URL/Request is resolved. It also determines how your URL will look like

Angular supports two Location Strategies:

1. **HashLocationStrategy**
   Where URL looks like http://localhost:4200/#/product
2. **PathLocationStrategy**
   Where URL looks like http://localhost:4200/product

Before going further let's first understand what client-side routing is

### Client-Side Routing

In a Multi-page web application, Every time the application needs to display a page it has to send a request to the web server. You can do that by either typing the URL in the address bar, or clicking on the Menu link/ button. Every such action results in the browser sending a new request to the Web server

But, Angular Applications are single-page applications or SPA.

All the components are displayed on a single page

In a Typical Single Page Application, when the Web application is loaded it loads a single HTML page. Whenever the user interacts with the page, only a part of the page is dynamically updated.

If you open the `index.html` in any of the angular applications, you would see the following HTML markup

```
<body>
  <app-root>Loading...</app-root>
</body>
```

The `"app-root"` is a placeholder (selector), which is defined in the root component.

Angular generates and loads the view associated with the root component inside the `"app-root"` . Any subsequent components are also loaded dynamically inside the `"app-root"` selector

Angular does all this behind the scenes.

In such a scenario, we are not required to change the URL. But that brings a few cons

- You won't be able to refresh the page
- You won't be able to go to a particular view by typing the URL
- Sharing the URL with someone is not possible
- The Back button will not work as you cannot go back to the previous page
- SEO is not possible

That is where the client-side routing comes into the picture

The Client-side routing simply mimics server-side routing by running the process in the browser. It changes the URL in the browser address bar and updates the browser history, without actually sending the request to the server

### How Client-Side Routing works

The Client-side routing is handled in two ways

1. Hash style Routing
2. HTML 5 Routing

## Hash Style Routing

The Hash style routing uses the anchor tags technique to achieve client-side routing.

The anchor tags, when used along with the # allows us to jump to a place, within the web page.

For Example

### Index.html

```
<a name="contact">Contact Us</a>
```

When we visited the URL http://mysite.com/index.html#contact , the browser would scroll to the location of the Contact us label

When the requested anchor tag is on the current page, then the browser does not send the request to the Web server.

The Hashstyle Routing uses this technique to create the URL

The URL would look something like

```

```

```
http://www.example.com
http://www.example.com/#/about
http://www.example.com/#/contact

```

In all the above examples, only the URL sent to the server is `http://www.example.com` the URL's `"#/about"` and `#/contact` is never sent to the server

## HTML 5 routing

The introduction of HTML5, now allows browsers to programmatically alter the browser's history through the history object.

Using history.pushState() method, we can now programmatically add the browser history entries and change the location without triggering a server page request.

The history.pushState method accepts the following three parameters.

1. **State object:** A state object is a JavaScript object which is associated with the new history entry created by pushState()
2. **Title:** This is an optional title for the state
3. **URL:** The new history entry's URL. The browser won't jump to that page.

For example

```

var stateObj= { message: "some message" };
history.pushState(stateObj, "title", newUrl);

```

Using `history.pushState` the method, The browser creates new history entries that change the displayed URL without the need for a new request.

Example

When you request for `http://www.example.com` the server sends the `index.html`

Now, When you click on `ProductList` link, Angular use's the `history.pushState` method to push the state and change the URL to `http://www.example.com/ProductList`

Now, when you click on the specific Product, we again the use history method to push the state and change the URL to `http://www.example.com/product/1`

Here, when you click the back button, the browser will retrieve the `http://www.example.com/ProductList` from history and displays it.

But there are cons to this approach

1. Not all browsers support HTML 5
2. The older browser does not support HTML5. So if you want to support older browsers, you have to stick to the hash-style routing

## Why is Server Support Needed for HTML 5 routing

Now, consider the above example

What would happen, when you type the URL `http://www.example.com/ProductList` and hit the refresh button.

The browser will send the request to the web server. Since the page `ProductList` does not exist, it will return the 404 (page not found) error.

This problem could be solved, if we are able to redirect all the requests to the `index.html`

It means that when you ask from `http://www.example.com/ProductList`, the Web server must redirect it to index.html and return the request. Then in the Front-end Angular will read the URL and dynamically load the `ProductListComponent`.

To make HTML5 routing work you need to send the instruction to the webserver to serve /index.html for any incoming request, no matter what the path is.

## Location Strategy

As mentioned earlier, Angular implements both Hashstyle & HTML 5 Routing. `HashLocationstrategy` implements the Hashstyle routing & `PathIocationstrategy` implements the HTML5 style routing

## PathLocationStrategy Vs HashLocationStrategy

### PathLocationStrategy

**Pros:**

- Produces a clear URL like http://example.com/foo
- Supports Server-Side Rendering

Server-side Rendering is a technique that renders critical pages on the server that can greatly improve perceived responsiveness when the app first loads

Cons:

- Older browser does not support
- Server Support needed for this to work

### HashLocationStrategy

**Pros:**

- Supported by all browsers

**Cons:**

- Produces a URL like http://example.com/#foo
- Will not Support Server-Side Rendering

## PathLocationStrategy

The `PathLocationStrategy` is the default strategy in the Angular application.

To Configure the strategy, we need to add `<base href>` in the `<head>` section of the root page (index.html) of our application

```
<base href="/">
```

The Browser uses this element to construct the relative URLs for static resources (images, CSS, scripts) contained in the document.

If you do not have access to `<head>` Section of the `index.html`, then you can follow either of the two steps

Add the `APP_BASE_HREF` value as shown in the provider's section of the root module

```
import {Component, NgModule} from '@angular/core';
import {APP_BASE_HREF} from '@angular/common';

@NgModule({
providers: [{provide: APP_BASE_HREF, useValue: '/my/app'}]
})
class AppModule {}
```

or use the absolute path for all the static resources like CSS, images, scripts, and HTML files.

## HashLocationStrategy

You can use the HashLocationStrategy by providing the useHash: true in an object as the second argument of the RouterModule.forRoot in the AppModule.

```
1
2  @NgModule({
3  declarations: [
4     AppComponent,HomeComponent,ContactComponent,ProductComponent,ErrorComponer
5  ],
6  imports: [
7     BrowserModule,
8     FormsModule,
9     HttpModule,
10    Hashlocationstrategy RouterModule.forRoot(appRoutes, { useHash: true }
11 ],
12 providers: [ProductService],
13 bootstrap: [AppComponent]
14 })
15
```

## Which Location Strategy to Use

We recommend you use the HTML 5 style ( PathLocationStrategy ) as your location strategy.

Because

- It produces clean and SEO Friendly URLs that are easier for users to understand and remember.
- You can take advantage of the server-side rendering, which will make our application load faster, by rendering the pages in the server first before delivering them the client

Use the hash location strategy only if you have to support older browsers.

### Summary

Angular supports two different location strategies or Routing strategies in Angular. One is PathlocationStrategy and the other one is HashLocationStrategy . The