Query Parameters in Angular

7 Comments / 10 minutes of reading / March 9, 2023

← Child/Nested Routes Angular Tutorial Navigation Between Routes →

Query Parameters in Angular allow us to generate URLs with **query strings**. They allow us to pass optional parameters like page number, sorting & filter criteria to the component. In this tutorial, we look at how to pass the query parameters using the **queryParams property** of the **router.navigate** method or **routerlink** directive. We then look at how to read the parameter in the component using the **ActivatedRoute Service**. We also find out the difference between the Query and Route parameters and which one to use and when. Finally, we create an Angular Query Params Example application.

Table of Contents

What are query parameters?

Query params (or Query Parameters) are key-value pairs that appear to the right of the ? in a URL. Each query parameter is separated by `&`.

```

/product?page=2&filter=all

```

In the above example, **page=2&filter=all** is the query params. It contains two query parameters. One is **Page** whose value is **2** and the other one is **Filter** whose value is **all**.

They are also known as **query strings**.

Difference between Query parameters and Route parameters

The route parameters are required and Angular Router uses them to determine the route. They are part of the route definition.

For Example, when we define the route as shown below, the `id` is the route parameter.

```

{ path: 'product', component: ProductComponent }
{ path: 'product/:id', component: ProductDetailComponent }

```

The above route matches the following URL The angular maps the values 1 & 2 to the `id` field

```

URL
/product       matches => path: 'product'
/product/1      matches => path: 'product/:id'
/product/2      matches => path: 'product/:id'

```

The Angular Router will not navigate to the `ProductDetailComponent` route if the id is not provided. It will navigate to ProductComponent instead. If the product route is not defined, it will result in an error.

However, **the query parameters are optional**. The missing parameter does not stop angular from navigating to the route. The query parameters are added to the end of the URL Separated by Question Mark

Which one to use? Route Parameters or Query Parameters?

The best practice is to use Route Parameters to identify a specific resource or resources. For example a specific product or group of products.

```

//All Products
/products

//specific Product
/product/:id

```

Use query parameters to sort, filter, paginate, etc. For example sort products based on name, rating, etc. Filter based on price, color, etc.

```

 //All Products sorted on rating
/products?sort=rating

//All Products sorted on rating with color=red
/products?sort=rating&color=red

//All Products sorted on rating with color=red & second page
/products?sort=rating&color=red&page=2

```

The route parameters determine the route. The angular Router Module determines which component to load using them. Hence they are required.

Query Parameters are optional. Hence if you do not require the value then use the query parameter.

Adding Query Parameters

The Query parameters are not part of the route. Hence you do not define them in the routes array like route parameters. There are two ways in which you can pass a Query Parameter to Route

1. Using `routerlink` directive
2. Using `router.navigate` method.
3. Using the router.navigateByUrl method

Using routerlink Directive in Template

We use the `queryParams` property of the `routerlink` directive to add the query parameter. We add this directive in the template file.

```
<a [routerLink]="['product']" [queryParams]="{ page:2 }">Page 2</a>
```

The `router` will construct the URL as

```
/product?page=2
```

You can pass multiple Query Parameters as shown below

```
<a [routerLink]="['products']"
                    [queryParams]="{ color:'blue' , sort:'name'}">Products</a>
```

The router will construct the URL as

```

/products?color=blue&sort=name

```

Using router.navigate method in Component

You can also navigate programmatically using the `navigate` method of the Router service as shown below

```

goTo() {
   this.router.navigate(
      ['/products'],
      { queryParams: { page: 2, sort:'name'} }
   );
}

```

The above code will navigate to the following URL

```

/products?page=2&sort=name

```

Using the router.navigateByUrl method in the component

You can also use the `navigateByUrl` method of the router. navigateByUrl expects an absolute URL, Hence we need to build our query string programmatically. Also, the `navigateByUrl` method does not have `queryParamsHandling` option.

```

this.router.navigateByUrl('product?pageNum=2');

```

Reading Query Parameters

Reading the Query parameters is similar to reading the Router Parameter. There are two ways by which you can retrieve the query parameters.

1. Subscribing to the **queryParamMap** or **queryParams** observable
2. Using **queryParamMap** or **queryParams** property of the snapshot property

Both the above are part of the **ActivatedRoute service**. Hence we need to inject it into our component class.

Using queryParamMap observable

The `queryParamMap` is a Observable that returns a ParamMap of the query parameters available to the current route. We can use this to retrieve values from the query parameter. The `queryParamsMap` is accessible via `ActivatedRoute`

Hence, we need to inject the `ActivatedRoute` in the constructor of the component/service, where we want to read the query parameter as shown below

```

constructor(private Activatedroute:ActivatedRoute,
            private router:Router){
}

```

You can subscribe to the `queryParamMap` of the `ActivatedRoute`, which returns the observable of type ParamMap.

The ParamMap object contains three methods

get method retrieves the value of the given parameter.

getAll method retrieves all parameters

has method returns true if the ParamMap contains a given parameter else false

We use the `get` method to read the query parameter as shown below. The following code reads the value of the `pageNum` query parameter.

```

this.sub = this.Activatedroute.queryParamMap
      .subscribe(params => {
          this.pageNum = +params.get('pageNum')||0;
});

```

Using queryParams observable

The `queryParams` is a Observable that returns a Params. The Params array is a list of parameter values, indexed by name.

The following code shows how to subscribe and retrieve the value of `pageNum` query parameter.

```

this.sub = this.Activatedroute.queryParams
      .subscribe(params => {
        this.pageNum = +params.['pageNum']||0;
});

```

Using snapshot.queryParamMap property

You can also read the value of the query parameter from `queryParamMap` using the `snapshot` property of the `ActivatedRoute` as shown below

```

this.Activatedroute.snapshot
      .queryParamMap.get('pageNum')||0;;

```

Using snapshot.queryParams property

You can also read the value of the query parameter from `queryParams` property of the `snapshot` property.

```

this.Activatedroute.snapshot
      .queryParams['pageNum']||0;;

```

Which one to use? snapshot or observable

We usually retrieve the value of the Query parameter in the **ngOninit** life cycle hook, when the component is initialized.

When the user navigates to the component again, and if the component is already loaded then, Angular does not create the new component but reuses the existing instance. In such circumstances, the `ngOnInit` method of the component is not called again.

If you are using the Snapshot to read the value of the Query parameters in ngOnInit, then you will be stuck with the values that you read when the component is loaded for the time. This is because Snapshot is not observable. Hence it will not notify you if the value changes.

By subscribing to the paramMap observable (or to the params observable), you will get a notification when the value changes. Hence you can retrieve the latest value of the parameter and update the component accordingly.

queryParamsHandling

The query parameter is lost when the user navigates to another route.

For Example, if a user navigates to the product page with the route `/product?pageNum=2` and later navigates to the product detail page, angular removes the query parameter from the URL. This is the default behavior

You can change this behavior by configuring the **queryParamsHandling strategy**. This Configuration strategy determines how the angular router handles query parameters when the user navigates away from the current route. It has three options

1. “”

2. preserve
3. merge

queryParamsHandling:”

This is the default option. The angular removes the query parameter from the URL when navigating to the next route.

```
this.router.navigate(['product'],
      { queryParams: { pageNum: this.pageNo + 1 },
      queryParamsHandling :''}   );
```

```
<a [routerLink]="['product']"
                    [queryParams]="{ pageNum:2 }">Page 2</a>
```

queryParamsHandling preserve

The Angular preserves or carries forwards the query parameter of the current route to the next navigation. Any query parameters of the next route are discarded

```
this.router.navigate(['product'],
       { queryParams: { pageNum: this.pageNo + 1 },
         queryParamsHandling :"preserve"}
);
```

```
<a [routerLink]="['product']"
                    [queryParams]="{ pageNum:2 }"
                    queryParamsHandling="preserve">Page 2</a>
```

queryParamsHandling merge

The Angular merges the query parameters from the current route with that of the next route before navigating to the next route.

```

this.router.navigate(['product'],
             { queryParams: { pageNum: this.pageNo + 1 },
             queryParamsHandling :"merge"}
 );

```

```

<a [routerLink]="['product']"
                     [queryParams]="{ pageNum:2 }"
                     queryParamsHandling="merge">Page 2</a>

```

Note that preserveQueryParams is DEPRECATED

Angular Query Params Example

Let us build a small application to demonstrate the use of Query parameters. You can download the code from the Stackblitz.

The Query Params example application has two components (AppComponent & ProductComponent). The Navigation menu uses bootstrap 3 CSS.

app.component.ts

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `

  <div class="container">
  <nav class="navbar navbar-default">
    <div class="container-fluid">
      <div class="navbar-header">
        <a class="navbar-brand" href="#"
          ><strong> {{ title }} </strong></a
        >
      </div>

      <ul class="nav navbar-nav">
        <li><a [routerLink]="['home']">Home</a></li>
        <div class="navbar-form navbar-left">
          <div class="form-group">
            <input
             type="text"
             class="form-control"
             placeholder="Page No"
             [(ngModel)]="pageNum"
            />
          </div>
        </div>
        <li class="nav">
          <a [routerLink]="['product']" [queryParams]="{ pageNum: pageNum }"
            >Product</a
          >
        </li>
        <li><a [routerLink]="['contact']">Contact us</a></li>
      </ul>
    </div>
  </nav>

  <router-outlet></router-outlet>

  `,
})
export class AppComponent {
  title = 'Routing Module - Query Parameters';
  pageNum = 0;
}

```

We have defined the pageNum variable in the component.

```

export class AppComponent {
  title = 'Routing Module - Query Parameters';
  pageNum = 0;
}

```

The `AppComponent` has the Navigation Menu. It contains the link to the product page. The link is created using the `routerlink` directive. The `routerlink` directive contains the `queryParams` property, where we pass the `pageNum` to `"pageNum"` variable.

```

<li class="nav"><a [routerLink]="['product']"
                         [queryParams]="{ pageNum: pageNum }">Product</a></li>

```

The Angular will construct the URL as

```

/product?pageNum=xxx

```

An input box pageNum is provided so that the user can change page no.

```

        <div class="form-group">
          <input
            type="text"
            class="form-control"
            placeholder="Page No"
            [(ngModel)]="pageNum"
          />
        </div>
      </div>

```

product.component.ts

```

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  template: `

  <h1>Product Page</h1>

 <p>Current Page No <strong> {{pageNo}} </strong></p>
 <p>snapshot Page No  <strong> {{ snapshotPageNo }} </strong></p>

<button (click)="nextPage()">Next Page</button>
  `,
})
export class ProductComponent implements OnInit {
  pageNo = 0;
  snapshotPageNo = 0;

  constructor(private Activatedroute: ActivatedRoute,
                    private router: Router) {
  }

  ngOnInit() {

    this.snapshotPageNo =
      +this.Activatedroute.snapshot.queryParamMap.get('pageNum') || 0;

    this.Activatedroute.queryParamMap.subscribe((params) => {
      this.pageNo = +params.get('pageNum') || 0;
      console.log('Query params ', this.pageNo);
    });
  }

  nextPage() {
    this.router.navigate(['product'], {
      queryParams: { pageNum: this.pageNo + 1 },
    });
  }
}


```

The ProductComponent does not display any products. But it displays the query parameters received

First, we inject both ActivatedRoute and Router Service in the constructor of the ProductComponent

```

constructor(private Activatedroute:ActivatedRoute,
            private router:Router){
}


```

Next, in the ngOnInit lifecycle hook, we use the Activatedroute.snapshot method to retrieve the pageNum and update the snapshotPageNo variable.

We also subscribe to the queryParams property. We are updating our local variable pageNo with the page number obtained from the Queryparams

```

ngOnInit() {

   this.snapshotPageNo = this.Activatedroute.snapshot.queryParams['pageNum'] || 0;

   this.Activatedroute.queryParams
      .subscribe(params => {
         this.pageNum = +params['pageNum']||0;
        console.log('Query params ',this.pageNum)
   });
}

```

In our template, we display both snapshotPageNo pageNo variable.

```

<p>Current Page No <strong> {{pageNo}} </strong></p>
<p>snapshot Page No  <strong> {{ snapshotPageNo }} </strong></p>

```

And a button to take us to the next page.

```

<button (click)="nextPage()">Next Page</button>

```

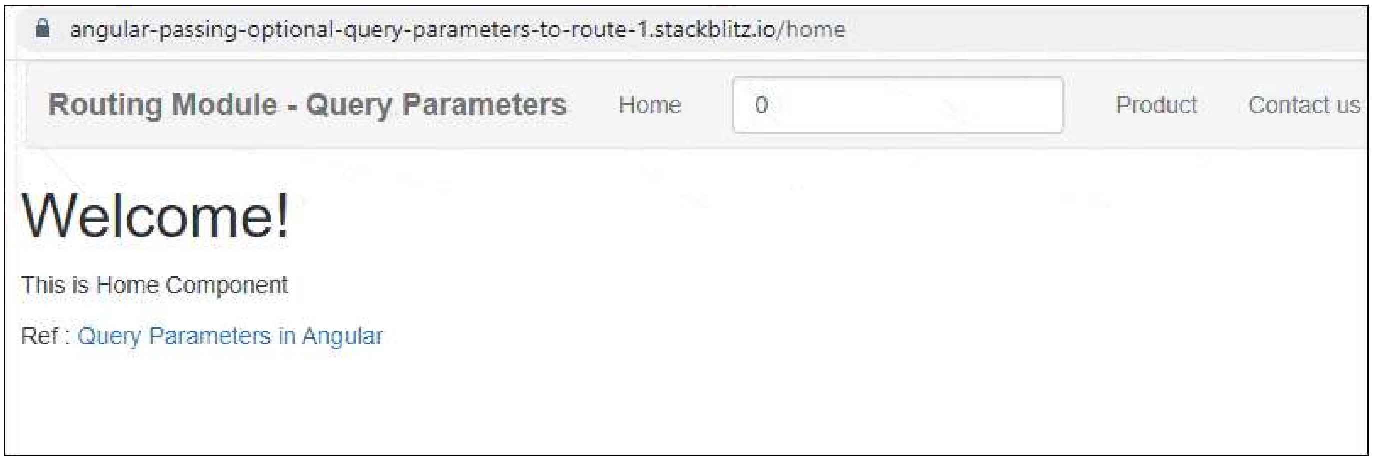
We also have nextPage method. It uses the router.navigate method to take us to the next page.

```

nextPage() {
     this._router.navigate(['./'],
                          { queryParams: { pageNum: this.pageNum + 1 },
                            relativeTo: this._Activatedroute }   );
  }

```

Here we are using the router.navigate method to navigate to the next page. This actually does not change the route as we are already on the page. The angular router does not re-create the component, but since we have subscribed to the Query parameters, we will be notified of the change in page Number. The pageNum variable is updated to the new value, while the snapshotPageNo does not change

You can refer to the StackBltiz for rest of the code

Query Params Example Application in Action


angular-passing-optional-query-parameters-to-route-1.stackblitz.io/home
Routing Module - Query Parameters
Home
0
Product
Contact us

Welcome!

This is Home Component

Ref : Query Parameters in Angular

1.

Enter the Page Number (for example 2) and click on the Product Link. You will navigate to the Product Page with the URL /product/pageNum=2. The Product Component reads the pageNum from the Activatedroute service. You will observe that the snapshot Page Number is the same as the page number obtained from the observable

Click on Next Page will change the URL and Page No. **But Angular will not reload the Component because the component is already loaded**. Hence the snapshot Page Number will remain the same.

queryParamsHandling Example

Open the app.component.ts and add the **queryParamsHandling='preserve'** to the contact page routerLink

```

<a [routerLink]="['contact']"
      queryParamsHandling='preserve'
          >Contact us</a>

```

Now navigate to the Product Page and then to the Contact us page. You will see that the URL will retain the query strings.