**. Passing Data from URLs**

Consider that we are navigating from one page to another in which the previous page is destroyed and we are landing on another page. If there’s not a lot of data to pass through (e.g., id of an object), we can use the URL to pass the data.

There are two ways to pass the data through URLs in Angular:

* Router Parameters
* Query Params

If the parameter is mandatory for the component then we have to use *router parameter*. Otherwise, we can use *query params*.

**Using Router Parameters**

Router parameters are required parameters. We have to register the parameter with the URL in the router module like this:

app-router.module.ts

const routes: Routes = [

{ path: 'list/:id', component: AppListComponent }

];

Copy

In this example, list is the route URL and :id is the router param that is mandatory to pass and AppListComponent is the component to mount on that route.

**Passing Router Param Through routerLink Directive**

<button

type="button"

[routerLink]="['/list', id]"

>

Show List

</button>

Copy

In this example, id is the variable initialized in that component’s code and the /list is the route on which we want to navigate.

**Passing Router Param Through route Service**

app.component.ts

id = 28;

constructor (private router: Router) {}

route() {

this.router.navigate(['/list', this.id]);

}

Copy

**Reading Router Params**

Here’s how to read a router parameter from the component routed to:

app-list.component.ts

constructor(

private activatedroute: ActivatedRoute

) {

this.activatedroute.params.subscribe(data => {

console.log(data);

})

}

Copy

**Using Query Parameters**

Query params are optional params. There is no need to register a separate URL for the query params.

app-router.module.ts

const routes: Routes = [

{ path: 'list', component: AppListComponent }

];

Copy

In this example, list is the route URL and AppListComponent is the component.

**Passing Query Param Through routerLink Directive**

<button

type="button"

[routerLink]="['/list']"

[queryParams]="{id: '24'}"

>

Show List

</button>

Copy

In this example, id is the key and 24 is the static value. You can also pass dynamic value through a variable.

**Passing Query Param Through route Service**

app.component.ts

id = 28;

constructor (private router: Router) {}

route() {

this.router.navigate(['/list'], {queryParams: {id: this.id}});

}

Copy

**Reading Query Params**

app-list.component.ts

constructor(

private activatedroute: ActivatedRoute

) {

this.activatedroute.queryParams.subscribe(data => {

console.log(data);

})

}

Copy

**Note:** Get more details on [Angular router params in this article](https://www.digitalocean.com/community/tutorials/angular-query-parameters).

**2. Passing Data Through @Input and @Output**

If we want to pass data from a child to a parent or a parent to a child component we can use @Input and @Output.

app-parent.component.html

<app-child

[jsonData]="data"

(outputData)="data = $event"

></app-child>

Copy

Here data is a variable initialized in the component’s code.

app-child.component.ts

import { Component, Input, OnInit } from '@angular/core';

@Component({

selector: 'app-child',

template: ''

})

export class AppChild implements OnInit {

@Input()

jsonData;

@Output()

outputData = new EventEmitter();

constructor() {}

ngOnInit() {

console.log(this.jsonData);

}

emitData(data) {

this.outputData(data);

}

}

Copy

In this way, we can pass data from child to parent and from parent to child.

**Note;** Get more detail on [@Input() in this article](https://www.digitalocean.com/community/tutorials/angular-inputs-angular) and on [@Output() in this article](https://www.digitalocean.com/community/tutorials/angular-outputs-angular).

**3. Passing Data Through a Service Using Observables**

If two components are siblings or the level of a component in the hierarchy is more distant, then it’s good to use a service for passing the data using observables.

This example uses [RxJS subject](https://www.digitalocean.com/community/tutorials/rxjs-subjects) for creating an observable:

app.service.ts

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

import { Subject } from 'rxjs';

@Injectable({providedIn: 'root'})

export class AppService {

observer = new Subject();

public subscriber$ = this.observer.asObservable();

emitData(data) {

this.observer.next(data);

}

}

To emit the data you can call the emitData method of this service and for getting the data you have to subscribe to subsciber$ like this:

constructor(private appService: AppService) {}

ngOnInit() {

this.appService.subscriber$.subscribe(data => {

console.log(data);

});

}