**Resolvers:**

Consider a scenario where if there is a delay in fetching data from the server and the component is loaded empty. This kind of delay can lead to bad user experience(UX). To improve this behavior, we should pre-fetch the data from the server and then should load the component.

This can be implemented using Resolvers in Angular.

A Resolver in Angular is a service class that should inherit Resolve interface and override resolve() method. A resolver can return response of type observable or promise or any custom type.

A Resolver in Angular is a service class that should inherit Resolve interface and override resolve() method. A resolver can return response of type observable or promise or any custom type.

Consider the Book example. Let us create a resolver class to fetch the books data before loading BookComponent and DashboardComponent as shown below.

1. import { Injectable } from '@angular/core';
2. import { Observable } from 'rxjs';
3. import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
4. @Injectable({
5. providedIn: 'root'
6. })
7. export class BookResolver implements Resolve<any> {
8. constructor() { }
9. resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<any> {
11. }
12. }

In the above example, resolve() method takes two parameter.

* route: ActivatedRouteSnapshot => contains the route information of component such as route parameters
* state: RouterStateSnapshot => contains the information about state of router

route parameter in resolve method also holds the route parameters passed along with the URL.

**Example:**

1.     Create another service class called **book-resolver.service.ts** and add the following code

1. import { Injectable } from '@angular/core';
2. import { Resolve } from '@angular/router';
3. import { Observable, of } from 'rxjs';
4. import { map, catchError } from 'rxjs/operators';
5. import { BookService } from './book.service';
6. @Injectable({
7. providedIn: 'root'
8. })
9. export class BookResolver implements Resolve<any> {
10. constructor(
11. private bookService: BookService
12. ) { }
13. resolve(): Observable<any> {
14. return this.bookService.getBooks()
15. .pipe(
16. map(data => of({ bookList: data })),
17. catchError(error => of({ errorMsg: error }))
18. );
19. }
20. }

Invoking service in not required inside ngOnInit() of a component, instead the route parameter now holds the data sent from the services.

2. Add the following code to **dashboard.component.ts** file. Observe the code inside ngOnInit() method.

1. import { Component, OnInit } from '@angular/core';
2. import { Router, ActivatedRoute } from '@angular/router';
3. import { Book } from '../book';
4. @Component({
5. selector: 'app-dashboard',
6. templateUrl: './dashboard.component.html',
7. styleUrls: ['./dashboard.component.css']
8. })
9. export class DashboardComponent implements OnInit {
10. books: Book[] = [];
11. errorMessage: any;
12. constructor(
13. private router: Router,
14. private route: ActivatedRoute
15. ) { }
16. ngOnInit() {
17. const listData = this.route.snapshot.data.bookList;
18. if (listData) {
19. if (!listData.errorMsg) {
20. listData.subscribe(
21. data => this.books = data.bookList.slice(1, 5)
22. );
23. } else {
24. this.errorMessage = listData.errorMsg;
25. }
26. }
27. }
28. gotoDetail(book: Book) {
29. this.router.navigate(['/detail', book.id]);
30. }
31. }

Line 20: Resolved data can be accessed from route parameter with the help of **snapshot.data** property. Since the data is in observable format, it is necessary to use subscribe method to get the data in a readable-format

3. Add the following code to **book.component.ts** file. Observe the code written inside getBooks() method.

1. import { Component, OnInit } from '@angular/core';
2. import { Book } from '../book';
3. import { BookService } from '../book.service';
4. import { ActivatedRoute } from '@angular/router';
5. @Component({
6. selector: 'app-book',
7. templateUrl: './book.component.html',
8. styleUrls: ['./book.component.css']
9. })
10. export class BookComponent implements OnInit {
11. title1 = 'Demo on HttpClientModule';
12. books: any;
13. errorMessage: string;
14. newBook: Book = { id: 11, name: 'type' };
15. addBookFlag: boolean;
16. updateBookFlag: boolean;
17. deleteBookFlag: boolean;
18. constructor(
19. private bookService: BookService,
20. private route: ActivatedRoute
21. ) { }
22. getBooks() {
23. const listData = this.route.snapshot.data.bookList;
24. if (listData) {
25. if (!listData.errorMsg) {
26. listData.subscribe(
27. data => this.books = data.bookList
28. );
29. } else {
30. this.errorMessage = listData.errorMsg;
31. }
32. }
33. }
34. addBook(id, name) {
35. this.bookService.addBook({id, name})
36. .subscribe(hero => this.books.push(hero));
37. }
38. updateBook(id, name) {
39. this.bookService.updateBook({id , name})
40. .subscribe(hero => this.books = hero);
41. }
42. deleteBook(id: number) {
43. this.bookService.deleteBook(id)
44. .subscribe(hero => this.books = hero);
45. }
46. ngOnInit() {
47. this.getBooks();
48. }
49. }

Line 26: Resolved data can be accessed from route parameter with the help of **snapshot.data** property

4. Add the following code in **app-routing.module.ts.**Observe the usage of resolve property.

1. import { NgModule } from '@angular/core';
2. import { RouterModule, Routes } from '@angular/router';
3. import { BookComponent } from './book/book.component';
4. import { DashboardComponent } from './dashboard/dashboard.component';
5. import { BookDetailComponent } from './book-detail/book-detail.component';
6. import { BookResolver } from './book-resolver.service';
7. import { LoginGuardService } from './login/login.guard.service';
8. import { LoginComponent } from './login/login.component';
9. const appRoutes: Routes = [
10. {
11. path: 'dashboard',
12. component: DashboardComponent,
13. canActivate: [LoginGuardService],
14. resolve: { bookList: BookResolver }
15. },
16. {
17. path: '',
18. redirectTo: '/login',
19. pathMatch: 'full'
20. },
21. {
22. path: 'books',
23. component: BookComponent,
24. canActivate: [LoginGuardService],
25. resolve: { bookList: BookResolver }
26. },
27. {
28. path: 'login',
29. component: LoginComponent
30. },
31. {
32. path: 'detail/:id',
33. component: BookDetailComponent,
34. canActivate: [LoginGuardService]
35. }
36. ];
37. @NgModule({
38. imports: [
39. RouterModule.forRoot(appRoutes)
40. ],
41. exports: [
42. RouterModule
43. ]
44. })
45. export class AppRoutingModule { }

Line 15: Linking DashboardComponent to BookResolver in routing path with a route parameter value as *bookList*.

Adding resolve property in routing path will load the component only when the data is completely received in the component.