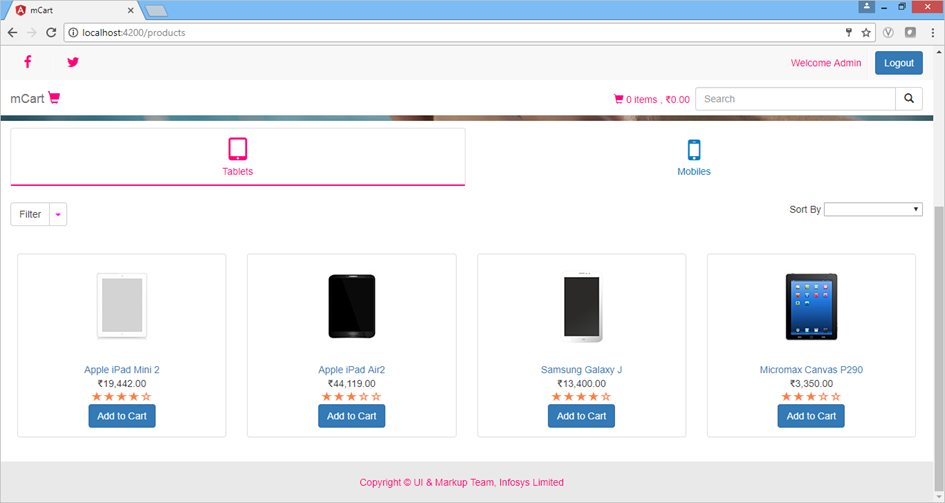
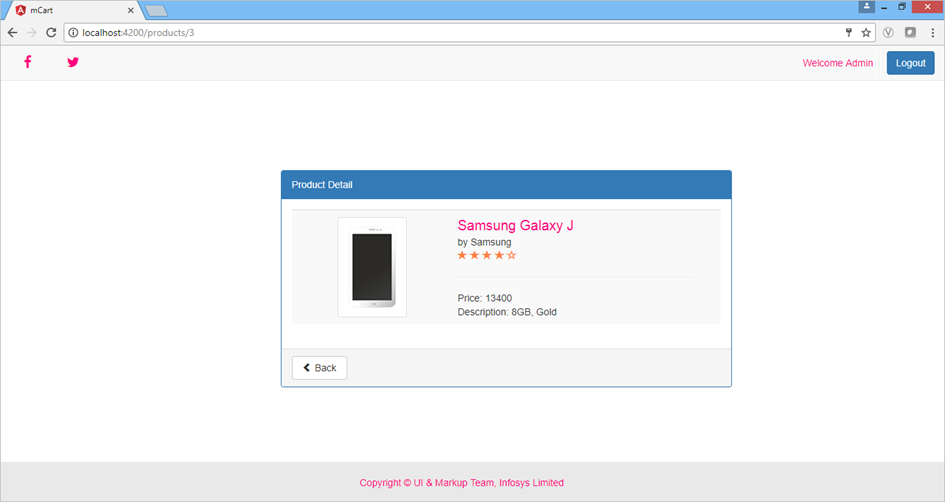
In this section, You will learn how to create a SPA(single page application) through routing.

mCart wants to add a feature where the additional details of the product will be rendered when the user clicks on the *product name* hyperlink as shown below.



After clicking on the product name link, it should render the following screen of ProductDetailComponent



You need to navigate from ProductListComponent to ProductDetailComponent by clicking on the product name hyperlink. This can be done by routing.

Routing is navigation between multiple views in a single page

Routing allows to express some aspects of the application's state in the URL. You can build the full application without ever changing the URL.

**Configuring Router**

Angular uses Component Router to implement routing.

The <base> tag should be present in the <head> tag in our root HTML page (index.html) to tell router how to compose navigation URLs. By default, a new angular app created with angular-cli will have the base tag. The href="/" represents the root of the application.

1. <base href="/">

Angular component router belongs to @angular/router module. To make use of routing, you should import Routes, RouterModule classes.

*RouterModule.****forRoot()*** method should be used when the routes are being configured in the root module of the app.

The use of these methods creates a singleton instance of the Router class, which is responsible for managing the routes.

You need to configure the routes and the router will look for a corresponding route when a browser **URL**changes. Routes is an array which contains all the route configurations. Then you need to pass this array to the *RouterModule.forRoot()* function in the application bootstrapping function.

You can move the routes declaration into a separate new module called ***app-routing.module.ts*** file.

**Example**:

**app-routing.module.ts**

1. ...
2. import { RouterModule, Routes } from '@angular/router';
3. ....
4. const appRoutes: Routes = [
5. { path: '/', redirectTo: '/login', pathMatch: 'full' },
6. { path: 'login', component: LoginComponent },
7. { path: 'bookDetails/:bookId', component: BookDetailsComponent }
8. ];
9. @NgModule({
10. imports: [RouterModule.forRoot(appRoutes)
11. ],
12. .....

**Line 2:** Imports Routes and RouterModule classes

**Line 6-9:**  Configure the routes where each route should contain the path to navigate and the component class which has to be invoked for a specific path

* You need to provide route configuration for default path i.e., path:'' and redirect it to the a specific route using redirectTo option. pathMatch is required if redirectTo option is used which specifies how the given path should match. Here pathMatch:full tells Router to match the given path completely.
* The path 'bookDetails/:bookId' where you specified the route parameter id which will receive different values based on the book selected

**Note**: You should not forget to **import the AppRoutingModule** in the AppModule. The AppRoutingModule should be added in the import array of the @NgModule() decorator of the corresponding module class (AppModule class in this case).

After configuring the routes, the next step is to decide how to navigate

There are two more parts of the router which helps in navigation.

* **RouterLink**- binds the router to links on a page and navigates to the appropriate application view when the user clicks a link.
* **RouterOutlet**  - used to specify where the views produced by the router is to be displayed.

The **difference**between href and routerLink is that **href**will reload the page, whereas **routerLink** will not reload the page.

**routerLink**is used to achieve the same functionality but since angular app are single page applications, where the page should not reload. routerLink navigates to new url and the component is rendered in place of <router-outlet> without reloading the page.

You can now use RouterLink directive to the anchor tag for navigation as shown below:

**app.component.ts**

1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'app-root',
4. styleUrls: ['./app.component.css'],
5. templateUrl: './app.component.html'
6. })
7. export class AppComponent {
8. title = 'Routing Demo';
9. }

**app.component.html**

1. <h1>{{title}}</h1>
2. <nav>
3. <a [routerLink]='["/bookList"]'>My Books</a>
4. </nav>
5. <router-outlet></router-outlet>

**Line 5 :** <router-outlet> is the place where the output of the component associated with the given path will be displayed. For example, if user click on My Books, it will navigate to /bookList which will execute BookComponent class as mentioned in the configuration details and the output will be displayed in the router-outlet.

You will learn to implement routing through the following demo.

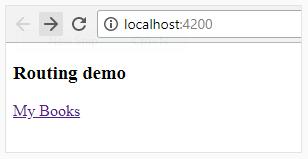
Highlights:

* Navigating between multiple components
* Understanding routing in an angular application

**Demo Steps:**

**Problem Statement:**Provide navigation between AppComponent to BookComponent. Output is as shown below

**Note:** You will add the **routing functionality** to **last demo** that we had created to implement service.



When the link **My Books** is clicked, it navigates to BookComponent as shown below:



You will modify the previous demo that you had created for services.

1. Modify **books.json** file under **assets folder** and add the following code

1. [
2. {
3. "id": 1,
4. "name": "HTML 5",
5. "stream": "JAVA"
6. },
7. {
8. "id": 2,
9. "name": "CSS 3",
10. "stream": "JAVA"
11. },
12. {
13. "id": 3,
14. "name": "Java Script",
15. "stream": "JAVA"
16. },
17. {
18. "id": 4,
19. "name": "VB.Net",
20. "stream": "MS"
21. },
22. {
23. "id": 5,
24. "name": "Asure",
25. "stream": "MS"
26. },
27. {
28. "id": 6,
29. "name": "C#",
30. "stream": "MS"
31. },
32. {
33. "id": 7,
34. "name": "Python",
35. "stream": "MS"
36. }
37. ]

2. Modify value of **booksUrl** in **book.service.ts**as shown below.

1. ....
2. export class BookService {
3. private booksUrl = './assets/books.json';
4. constructor(private http: HttpClient) { }
5. getBooks(): Observable<Book[]> {
6. return this.http.get<Book[]>(this.booksUrl);
7. }
8. }

3. Create a file, **app-routing.module.ts**and add the below code.

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { BookComponent } from './book/book.component';
4. const routes: Routes = [
5. {
6. path:'bookList', component:BookComponent
7. }
8. ];
9. @NgModule({
10. imports: [RouterModule.forRoot(routes)],
11. exports: [RouterModule]
12. })
13. export class AppRoutingModule { }

**Note:**You can use the **--routing** option with ng new to create **app-routing.module.ts** file when you create or initialize a project.

4. Modify the code in **app.component.html** file as shown

1. <div>
2. <h3>
3. Routing demo
4. </h3>
5. <a [routerLink]="['/bookList']">{{ title }}</a>
6. <br/>
7. <br/>
8. <router-outlet></router-outlet>
9. </div>

5. Save all the files and run the application.

You are now able to navigate from AppComponent to BookComponent.

You can also pass parameters to the URL while routing. Parameters passed along with URL are called route parameters. Let's take a look at route parameters next.

To navigate programmatically, you can use navigate() method of Router class. Inject the router class into the component and invoke navigate method as shown below:

1. this.router.navigate([url, parameters])

* url is the route path to which we want to navigate
* Parameters are the route values passed along with the url

**Example:**

1. import { Component, OnInit } from '@angular/core';
2. import { Router } from '@angular/router';
3. ....
4. export class SomeComponent implements OnInit {
5. constructor ( private router: Router, private dataService: DataService) { }
6. ngOnInit () {
7. .....
8. getDetails(vadiable : ClassType ) {
9. this.router.navigate(['/routePath', routeParam]);
10. }
11. }
12. }

**Line 2:** Import Router class from @angular/router module

**Line 7:** Inject into the component class through a constructor

**Line 12:** this.router.navigate() method is used to navigate to a specific URL programmatically. Navigate() method takes two arguments- first one is the path to navigate and the second one is the route parameter value to pass.

Let's take a look at how to implement route parameters with the help of a demo.

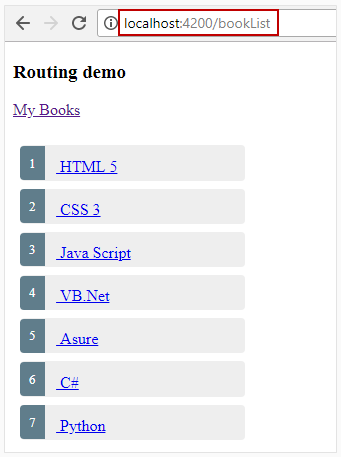
Highlights:

* Navigating between multiple views
* Implementing parameterised routing in Angular application

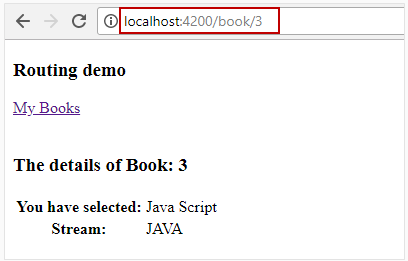
**Demo Steps:**

**Problem Statement:**Passing parameter while navigating from BookComponent to BookDetailsComponent. Output is as shown below

**Note:** We will add the **route parameters** to **Demo** that we had created to implement routing.



On clicking on a book title, the corresponding book details must be displayed as shown



1. Create BookDetails component using the following CLI command:

1. ng generate component bookDetails

2. Add the following code in **book-details.component.ts** file:

1. import { Component, OnInit } from '@angular/core';
2. import { ActivatedRoute} from '@angular/router';
3. import { Book } from '../book/book';
4. import { BookService } from '../book/book.service';
5. @Component({
6. selector: 'app-book-details',
7. templateUrl: './book-details.component.html',
8. styleUrls: ['./book-details.component.css']
9. })
10. export class BookDetailsComponent implements OnInit {
11. books!: Book[];
12. errorMessage: string='';
13. bookId: number=0;
14. chosenBook!: Book;
15. constructor(private bookService: BookService, private route: ActivatedRoute) { }
16. getBooks() {
17. this.bookService.getBooks().subscribe(
18. ibooks => {
19. this.books = ibooks; console.log(this.books);
20. this.route.params.subscribe(param => this.bookId = param['id']);
21. this.getBookDetails(this.bookId);
22. },
23. error => console.log(error));
24. }
26. getBookDetails(bookId:number) {
27. for (const iterator of this.books) {
28. if (iterator.id == bookId) {
29. this.chosenBook = iterator;
30. }
31. }
32. }
33. ngOnInit() {
34. this.getBooks();
35. }
36. }

**Line 2:** Imports ActivatedRoute class to access route parameters

**Line 19:** Injects ActivatedRoute class into the component class through a constructor

3. Add the following code in **book-details.component.html**file:

1. <div \*ngIf="chosenBook">
2. <div class="row">
3. <div class="col-md-4 col-md-offset-4 bordering-side">
4. <h3>The details of Book: {{chosenBook.id}}</h3>
5. <table [ngClass]="{'table':true}">
6. <tr>
7. <th>You have selected:</th>
8. <td>{{chosenBook.name}}</td>
9. </tr>
10. <tr>
11. <th>Stream:</th>
12. <td>{{chosenBook.stream}}</td>
13. </tr>
14. </table>
15. </div>
16. </div>
17. </div>

4. Modify the code in **book.component.html** file as given below:

1. <ul class="books">
2. <li \*ngFor="let book of books">
3. <a [routerLink]="['/book', book.id]">
4. <span class="badge">{{book.id}}</span> {{book.name}}</a>
5. </li>
6. </ul>
7. <div class="error" \*ngIf="errorMessage">{{errorMessage}}</div>

5. Modify the value of **routes array** in **app-routing.modules.ts**with the value given below

1. const routes: Routes = [
2. { path: 'bookList', component: BookComponent },
3. { path: 'book/:id', component: BookDetailsComponent}
4. ];

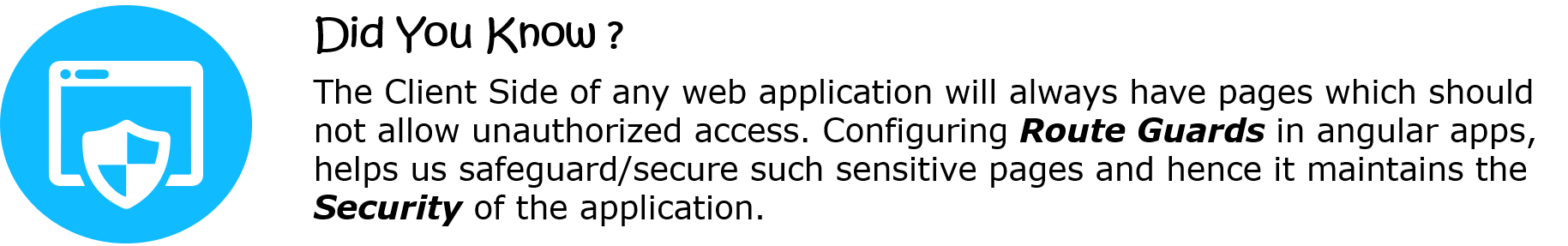
6. Save the files and run the application

Observe that while clicking on the book title, the BookDetailsComponent gets loaded by passing the id as parameter along with the route and specifies extra details on the selected book.

In Angular application, user can navigate to any url directly. Ideally, it should be checked whether a particular user is allowed to navigate to a particular url or not. This is achieved using **Route guards**.

Consider the following scenarios where we to need check if navigation is allowed -

* User has logged in first before accessing a component
* User is not authorized to access a component
* User has fetched data before displaying a component
* Pending changes have been saved before leaving a component



Next, we will learn more about route guards in detail.

In Angular, Route Guards are the interfaces using which we can communicate the Router module of angular, whether it should allow the user the navigate to the requested route or not. The route guard interfaces decide this based on the truthy or falsy values returned by the class which implements the guard interfaces.

* If it returns **true**, the navigation process continues
* If it returns **false**, the navigation process stops

Angular provides us with different interfaces to guard the routes in different scenarios. Those Interfaces are:

* **canActivate -**This is a guard Interface which decides whether a route can be activated or not. If the guard returns ***true***, the navigation continues else navigation is cancelled.
* **canActivateChild -**This is a guard Interface which decides whether a child route can be activated or not. If the guard returns ***true***, the navigation continues else navigation is cancelled.
* **canDeactivate -**This is a guard Interface which decides whether a route can be deactivated. If the guard returns ***true***, the navigation continues else navigation is cancelled.
* **canLoad -**This is a guard Interface which decides whether a children can be loaded or not. If the guard returns true, the navigation continues else navigation is cancelled.
* **Resolve -**This Interface can be implemented by the classes in angular app to be a data provider. We can use the data provider with the router for resolving data while navigation.

In this section we will see the use of only **CanActivate**Interface.

# How to Create Route Guards in Angular ?

We can create a class implementing required route Guard Interfaces using the below command:

1. ng generate guard auth

Once the guard is created successfully, Two new files i.e. **auth.guard.ts**and **auth.guard.spec.ts** are added to the project folder with the code as shown below:

1. Class AuthGuard implements CanActivate {
2. canActivate( ): boolean { }
3. }