



[View, add and edit your notes in the app](#)

Angular Router and Route Guards | Angular Concepts made easy | Procademy Classes

Generated on February 21, 2024

Summary

Notes

Screenshots

Bookmarks

0

190

0

Implementing Routing in Angular

- Angular routing allows for seamless navigation between different components in an application using the angular/router module.

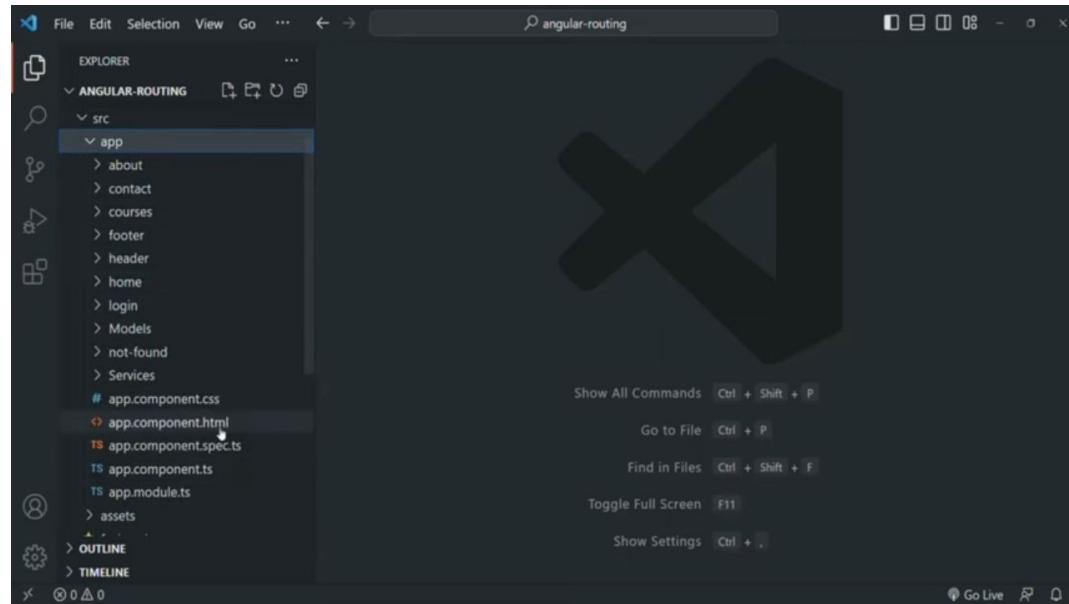
0:05

What is Routing

Routing allows us to navigate from one part of our application to another part. In Angular, using routing, we can move from view of one component to view of another component.

To implement routing in Angular, we use a built-in `@angular/router` module.

▶ 0:15



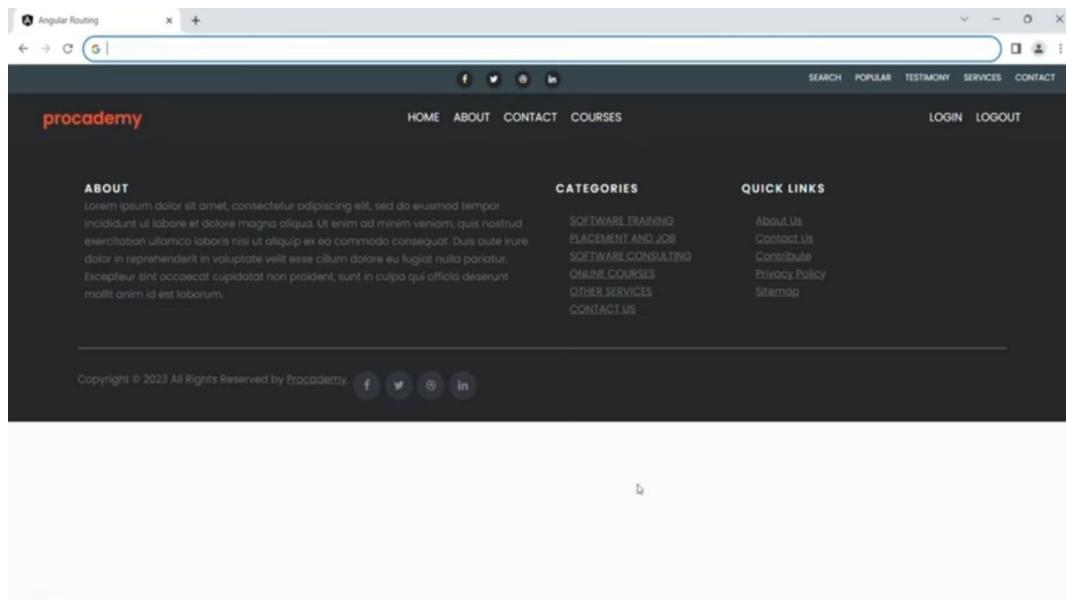
- ★ In the Source folder, the app component serves as the root component and utilizes selectors for both header and footer components.

▶ 1:10

```
<div class="main-page-container">
  <app-header></app-header>
  <!-->
  <!-->
  <!-->
  <!-->
  <!-->
</div>
```

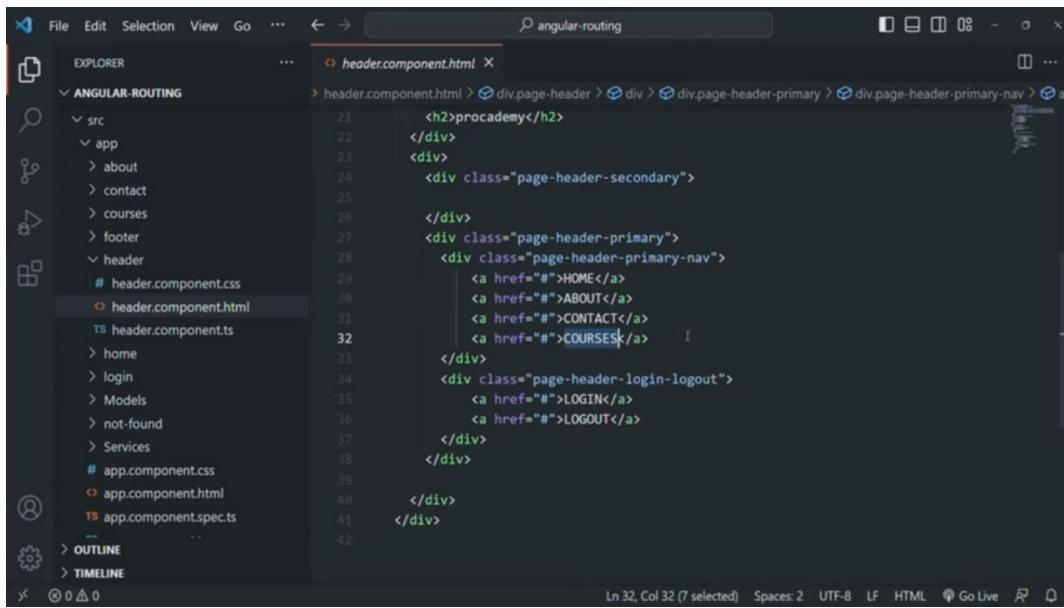
- ★ App component is the root component used in index.html file, rendering the content of header and footer components.

▶ 1:30



- ★ Our current focus is on the main pages of our angular application, including home, about, contact, and courses links.

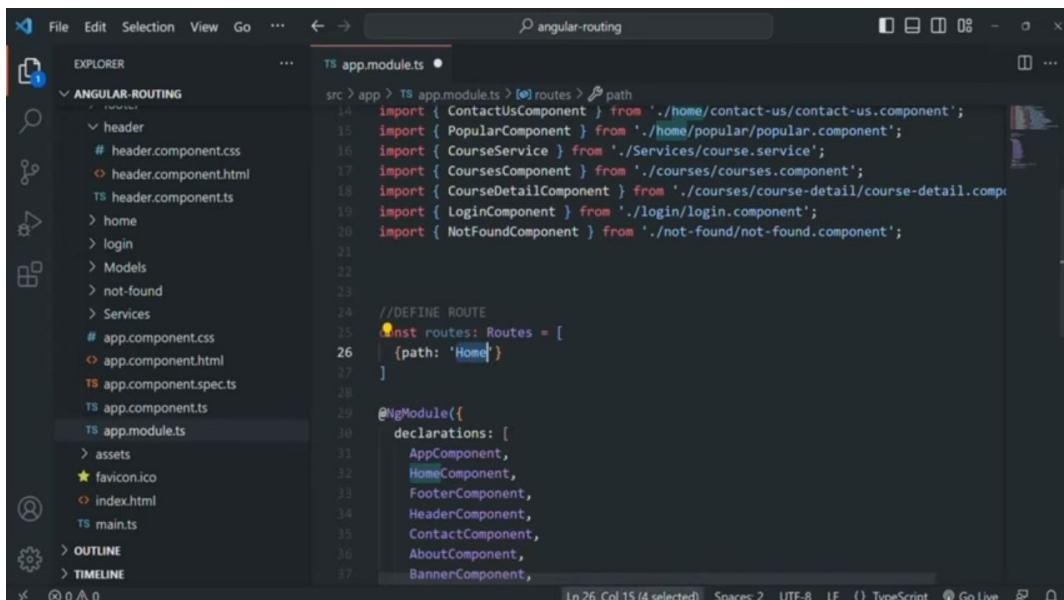
▶ 2:28



```
<h2>procademy</h2>
</div>
<div>
  <div class="page-header-secondary">
    </div>
    <div class="page-header-primary">
      <div class="page-header-primary-nav">
        <a href="#">HOME</a>
        <a href="#">ABOUT</a>
        <a href="#">CONTACT</a>
        <a href="#">COURSES</a>
      </div>
      <div class="page-header-login-logout">
        <a href="#">LOGIN</a>
        <a href="#">LOGOUT</a>
      </div>
    </div>
  </div>
</div>
```

- ★ A new web page has been designed to display different components when specific links are clicked.

▷ 2:53

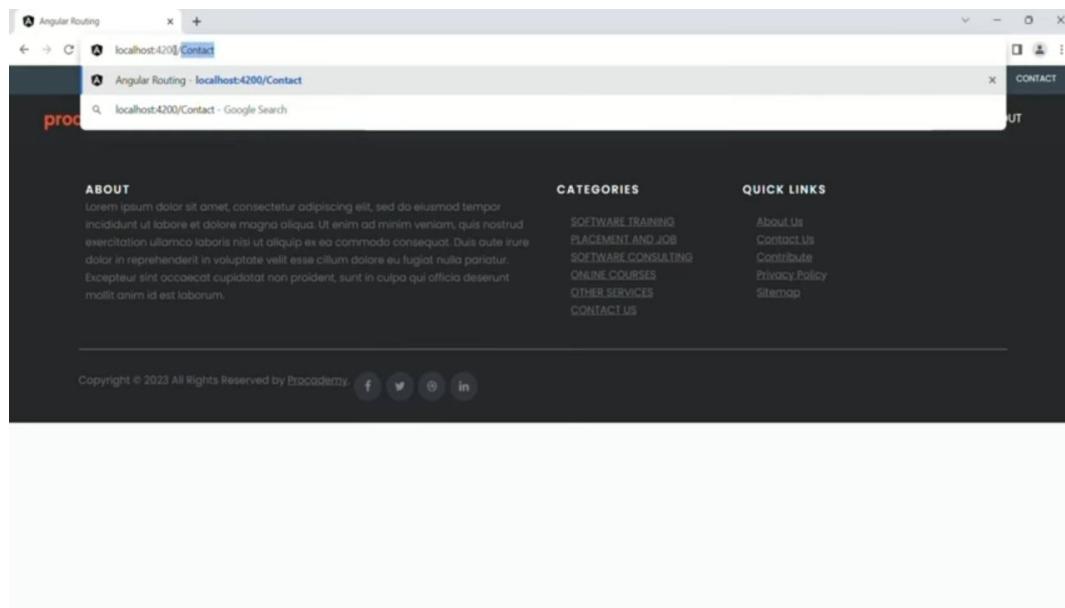


```
const routes: Routes = [
  {path: 'Home'}
]

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    FooterComponent,
    HeaderComponent,
    ContactComponent,
    AboutComponent,
    BannerComponent,
    ContactUsComponent,
    PopularComponent,
    CoursesComponent,
    CourseDetailComponent,
    LoginComponent,
    NotFoundComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [
    CourseService
  ],
  bootstrap: [AppComponent]
})
```

- ★ In the app module, routes are defined directly before the NG module directive.

▷ 3:56



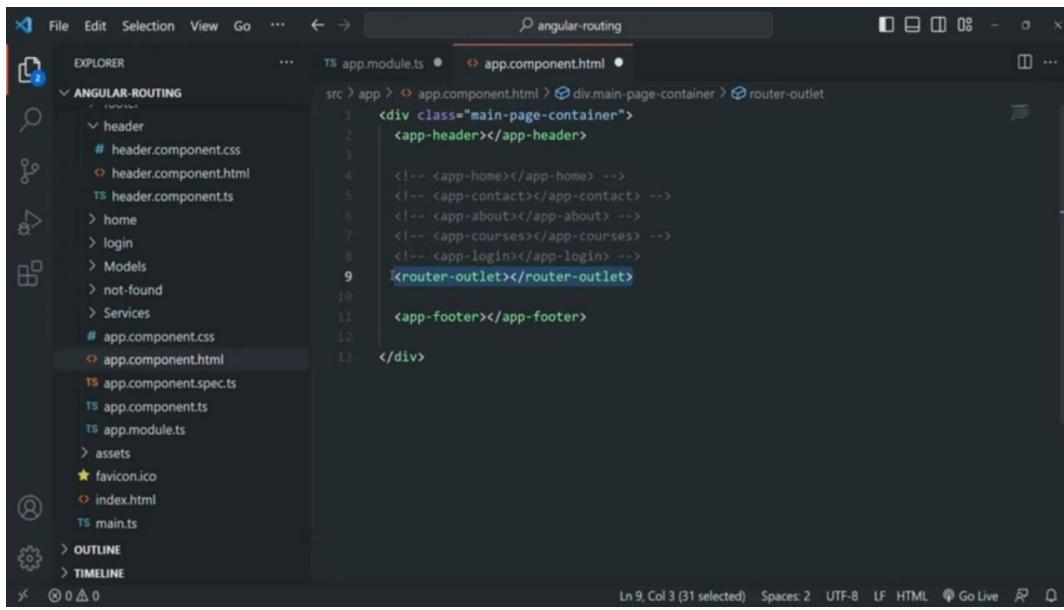
- 💡 Understanding the difference between URL and path is crucial for navigating websites effectively.

▷ 5:55

```
src > app > TS app.module.ts •
14 import { ContactUsComponent } from './home/contact-us/contact-us.component';
15 import { PopularComponent } from './home/popular/popular.component';
16 import { CourseService } from './Services/course.service';
17 import { CoursesComponent } from './courses/courses.component';
18 import { CourseDetailComponent } from './courses/course-detail/course-detail.component';
19 import { LoginComponent } from './login/login.component';
20 import { NotFoundComponent } from './not-found/not-found.component';
21
22
23
24 //DEFINE ROUTE
25 const routes: Routes = [
26   {path: 'Home', component: HomeComponent}
27 ]
28
29 @NgModule({
30   declarations: [
31     AppComponent,
32     HomeComponent,
33     FooterComponent,
34     HeaderComponent,
35     ContactComponent,
36     AboutComponent,
37     BannerComponent,
```

- 💡 When a user types "home" in the address bar, the web page will display the view of the Home component.

▷ 6:20



The screenshot shows the VS Code interface with the file `app.component.html` open. The code contains a `<router-outlet>` directive at line 9, which is highlighted. The code is as follows:

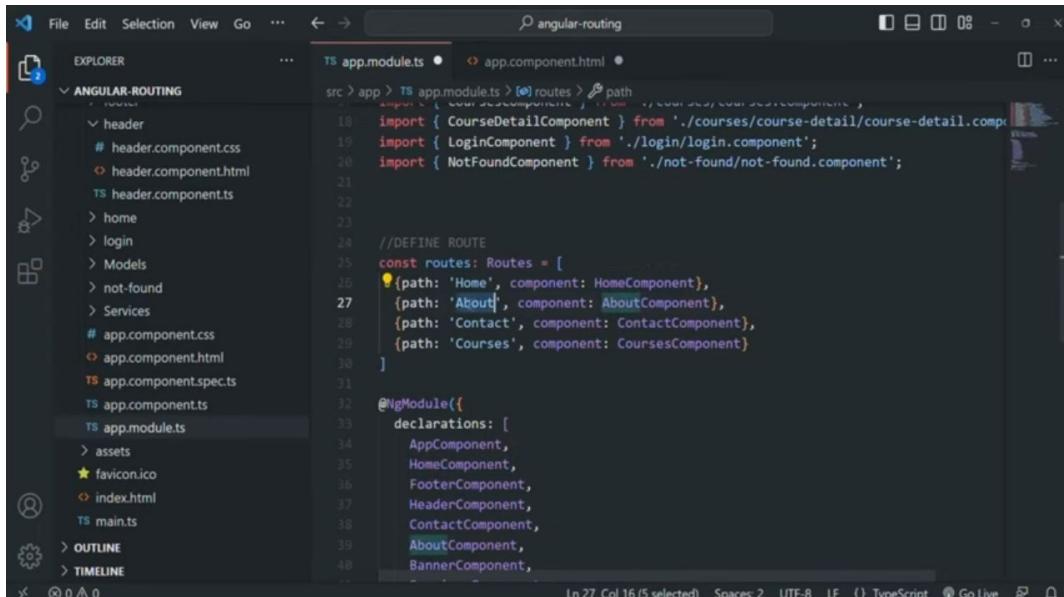
```
<div class="main-page-container">
  <app-header></app-header>

  <!--> <app-home></app-home> -->
  <!--> <app-contact></app-contact> -->
  <!--> <app-about></app-about> -->
  <!--> <app-courses></app-courses> -->
  <!--> <app-login></app-login> -->
  <router-outlet></router-outlet>

  <app-footer></app-footer>
</div>
```

Angular uses a directive called router Outlet to render view templates between the header and footer components based on the URL path.

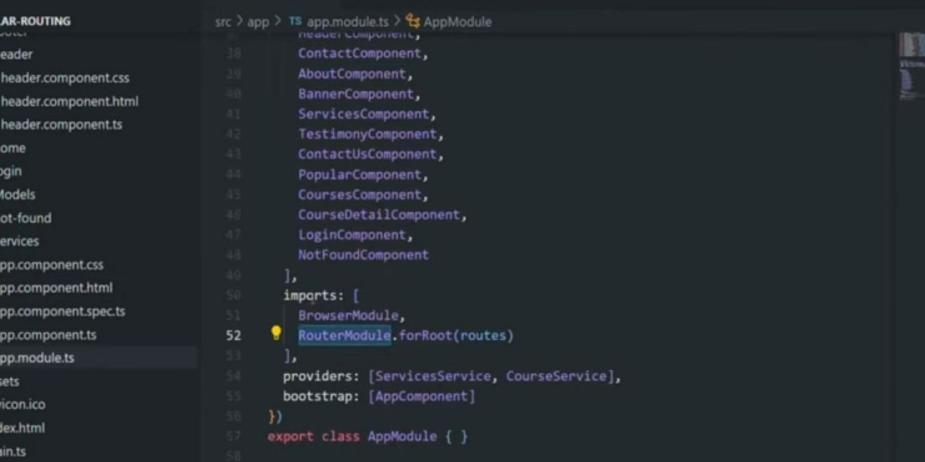
▶ 10:34



The screenshot shows the VS Code interface with the file `app.module.ts` open. The code defines routes in the `routes` array at lines 25-30. The routes are:

```
const routes: Routes = [
  {path: 'Home', component: HomeComponent},
  {path: 'About', component: AboutComponent},
  {path: 'Contact', component: ContactComponent},
  {path: 'Courses', component: CoursesComponent}
]
```

▶ 10:43

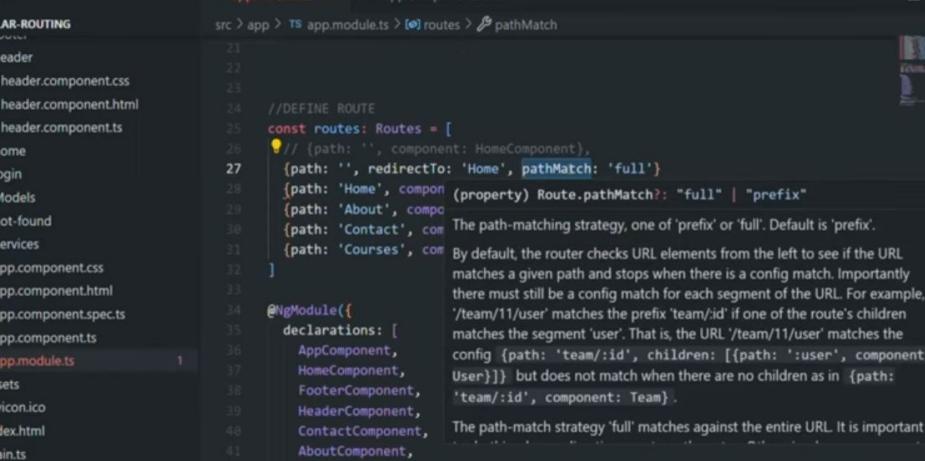


The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "ANGULAR-ROUTING". The "app.module.ts" file is currently selected.
- Code Editor (Center):** Displays the content of "app.module.ts". The code defines the AppModule with routes, imports BrowserModule and RouterModule.forRoot(routes), providers [ServicesService, CourseService], bootstrap [AppComponent], and an empty export class AppModule {}.
- Status Bar (Bottom):** Shows "Ln 52, Col 17 (12 selected)" and "Spaces: 2 - UTF-8 - LF - TypeScript - Go Live".

- The router Outlet directive is automatically imported when the router module is imported, eliminating the need for explicit import statements.

▶ 11:16



The screenshot shows the VS Code interface with the title bar "angular-routing". The Explorer sidebar on the left lists the project structure under "ANGULAR-ROUTING", including files like "header.component.css", "header.component.html", "header.component.ts", "app.module.ts", and "app.component.ts". The main editor area displays "app.module.ts" with code for defining routes:

```
//DEFINE ROUTE
const routes: Routes = [
  // (path: '', component: HomeComponent),
  {path: '', redirectTo: 'Home', pathMatch: 'full'}
  // (path: 'Home', component: HomeComponent),
  // (path: 'About', component: AboutComponent),
  // (path: 'Contact', component: ContactComponent),
  // (path: 'Courses', component: CoursesComponent)
]

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    FooterComponent,
    HeaderComponent,
    ContactComponent,
    AboutComponent,
    BannerComponent,
    ServicesComponent,
    TestimoniumComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [
    environment.provide
  ],
  bootstrap: [AppComponent]
})
```

The code editor has a tooltip explaining the "pathMatch" property:

The path-matching strategy, one of 'prefix' or 'full'. Default is 'prefix'.
By default, the router checks URL elements from the left to see if the URL matches a given path and stops when there is a config match. Importantly there must still be a config match for each segment of the URL. For example, '/team/11/user' matches the prefix 'team/:id' if one of the route's children matches the segment 'user'. That is, the URL '/team/11/user' matches the config '{path: 'team/:id', children: [{path: ':user', component: User}]} but does not match when there are no children as in {path: 'team/:id', component: Team}.

The path-match strategy 'full' matches against the entire URL. It is important

- Angular's path mesh property allows for easy configuration of URL matching and redirection.

▶ 15:26

How to define Routes

- Create a new route using **Routes** array and define some route objects inside that array.
- Register the route using **RouterModule.forRoot(routeName);**
- Use **<router-outlet>** where you want to render the view of specified route component.

Angular application now has five implemented routes for routing.

▶ 16:50

Implementing NotFound Route

Angular application now includes five implemented routes for Home, About, Contact, and Courses.

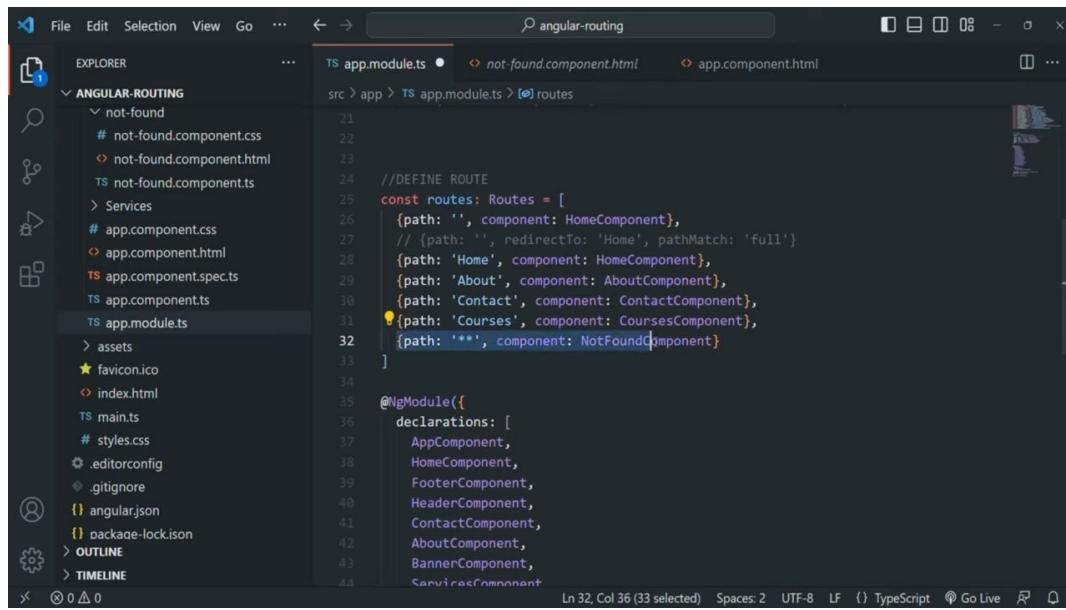
▶ 16:58

- Developer console reveals error in website routing, causing redirection to root URL.

▶ 18:28

- Developing a custom 404 page for your website can help improve user experience and provide helpful information when a page cannot be found.

▶ 20:26

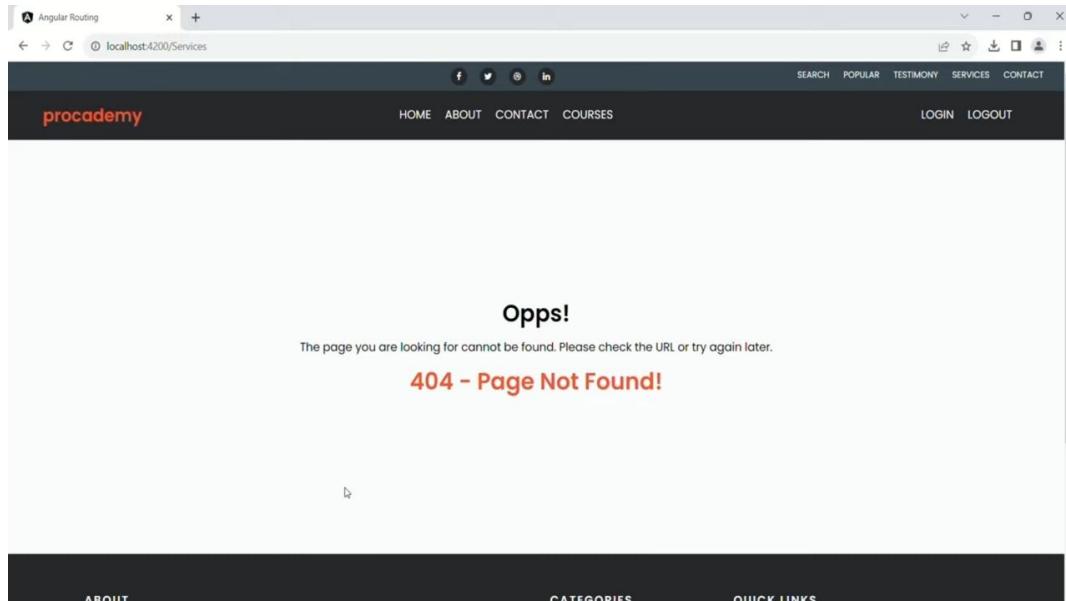


```
21
22
23
24 //DEFINE ROUTE
25 const routes: Routes = [
26   {path: '', component: HomeComponent},
27   // {path: '', redirectTo: 'Home', pathMatch: 'full'}
28   {path: 'Home', component: HomeComponent},
29   {path: 'About', component: AboutComponent},
30   {path: 'Contact', component: ContactComponent},
31   {path: 'Courses', component: CoursesComponent},
32   {path: '**', component: NotFoundComponent}
33 ]
34
35 @NgModule({
36   declarations: [
37     AppComponent,
38     HomeComponent,
39     FooterComponent,
40     HeaderComponent,
41     ContactComponent,
42     AboutComponent,
43     BannerComponent,
44     ServicesComponent
45   ]
46 })
47 export class AppModule { }
```

Ln 32, Col 36 (33 selected) Spaces: 2 UTF-8 LF {} TypeScript ⚡ Go Live ⌂ ⌂

✿ New routing component allows for dynamic rendering of not found views based on user input in the address bar.

▷ 21:11



✿ The not found component view is rendered on the web page and no errors are present in the developer console.

▷ 22:40

Wild Card Route

A wild card route is that route which matches every route path. In angular, the wild card route is specified using ** signs.



NOTE: A wild card route must be specified at the end of all the defined routes.

- ★ The wildcard route in Angular is used to match any route path and must be specified at the end of all defined routes.

▷ 24:01

Configuring navigation Link for Route

- ★ Developing functional navigation links in an Angular application is essential for displaying the correct views when users click on specific links.

▷ 28:57

The screenshot shows a web browser window with the URL localhost:4200. The page content includes a header with links to HOME, ABOUT, CONTACT, and COURSES. Below the header is a section with the text "Learn coding in easy step by step way. Excel in your career." followed by a paragraph of placeholder text (Lorem ipsum). At the bottom is a search bar with the placeholder "Search Courses" and a red "Search Courses" button. To the right of the browser window is a Network tab from the developer tools, showing a timeline of requests. The timeline shows several requests for files like styles.css, runtime.js, and vendor.js, which are listed in the table below. The table details the request information for each file.

Name	Status	Type	Initiator	Size	Tl.	Waterfall
localhost	304	d...	Other	23...	16...	
styles.css	304	st...	(index)	23...	20...	
runtime.js	304	sc...	(index)	23...	21...	
polyfills.js	304	sc...	(index)	23...	37...	
vendor.js	304	sc...	(index)	23...	78...	
main.js	304	sc...	(index)	23...	50...	
css2?family=...	200	st...	styles...	(m...	0 ...	
font-aweso...	200	st...	styles...	(m...	0 ...	
pxiGyp8kv8J...	200	font	css2f...	(m...	0 ...	
pxityp8kv8J...	200	font	css2f...	(m...	0 ...	
pxityp8kv8J...	200	font	css2f...	(m...	0 ...	
fontawesom...	200	font	font-a...	(m...	0 ...	
styles.js	304	sc...	(index)	23...	25...	
ng-cl-ws	101	w...	polyfill	0 B	Pe...	

- Runtime.js, polyfill.js, and vendor.js bundles have been downloaded for the application, but files are being re-downloaded when navigating to different pages.

▶ 33:16

The screenshot shows the Visual Studio Code interface with the file `header.component.html` open. The code is an Angular template for a header component. It includes an `<h2>procademy</h2>` heading, a secondary navigation area, a primary navigation area with links to Home, About, Contact, and Courses, and a login/logout area. The code uses Angular directives like `>`, `</>`, and `<div>`.

```

<h2>procademy</h2>
</div>
</div>
<div class="page-header-secondary">
</div>
<div class="page-header-primary">
  <div class="page-header-primary-nav">
    <a href="#">HOME</a>
    <a href="#">ABOUT</a>
    <a href="#">CONTACT</a>
    <a href="#">COURSES</a>
  </div>
  <div class="page-header-login-logout">
    <a href="#">LOGIN</a>
    <a href="#">LOGOUT</a>
  </div>
</div>

```

- Developing Angular applications requires proper use of directives to ensure desired behavior.

▶ 34:23

The screenshot shows the VS Code interface with the file `header.component.html` open. The code defines a header with a secondary navigation bar containing links for Home, About, Contact, and Courses, each wrapped in a `routerLink` directive. The primary navigation bar below it also contains links for Home, About, Contact, and Courses, but these are standard `a` href links.

```
<h2>procademy</h2>
</div>
<div>
  <div class="page-header-secondary">
    </div>
    <div class="page-header-primary">
      <div class="page-header-primary-nav">
        <a routerLink="Home">HOME</a>
        <a routerLink="About">ABOUT</a>
        <a routerLink="Contact">CONTACT</a>
        <a routerLink="Courses">COURSES</a>
      </div>
      <div class="page-header-login-logout">
        <a href="#">LOGIN</a>
        <a href="#">LOGOUT</a>
      </div>
    </div>
  </div>
</div>
```

- Using router links in applications can prevent the need to constantly download the complete application file when navigating.

▶ 36:43

The screenshot shows the same code as above, but with a syntax error highlighted. The line `<a [routerLink]="'About'">ABOUT` has a red squiggly underline under the opening bracket [and the closing bracket]. This indicates that square brackets cannot be used for string interpolation in this context; instead, single quotes must be used.

```
<h2>procademy</h2>
</div>
<div>
  <div class="page-header-secondary">
    </div>
    <div class="page-header-primary">
      <div class="page-header-primary-nav">
        <a routerLink="Home">HOME</a>
        <a [routerLink]="'About'">ABOUT</a> I
        <a routerLink="Contact">CONTACT</a>
        <a routerLink="Courses">COURSES</a>
      </div>
      <div class="page-header-login-logout">
        <a href="#">LOGIN</a>
        <a href="#">LOGOUT</a>
      </div>
    </div>
  </div>
</div>
```

- Square brackets cannot be used to assign a string value, instead the value must be wrapped in single quotes.

▶ 37:35

RouterLink Directive

The **RouterLink** is a directive that binds the HTML element to a Route. When the HTML element on which we have used the **RouterLink** is clicked, it will result in navigation to that Route.

NOTE: **RouterLink** directive is an attribute directive and we can also pass additional parameters to it.

- ★ The router link directive in our example binds HTML elements to specific routes, allowing for easy navigation within the website.

▷ 37:56

Styling Active Router Link

- ★ The implementation of routing in our Angular application has allowed for dynamic content changes and seamless navigation through the use of links.

▷ 39:40

The screenshot shows the VS Code interface with the file `styles.css` open. The code defines a new CSS class `.active` with the following properties:

```
src > # styles.css > .active
1  /* You can add global styles to this file, and also import other style files */
2  @import url('https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,400;1,500;1,600;1,700;1,800;1,900');
3  @import url('https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css');
4  *{
5      margin: 0px;
6      padding: 0px;
7      box-sizing: border-box;
8  }
9  body{
10     font-family: 'Poppins', sans-serif;
11     background-color: #FBFCFC;
12 }
13
14 .active{
15     font-size: 18px;
16     font-weight: bold;
17 }
```

- In the style.css file, a new CSS class called "active" has been created to apply font size and weight to active links.

▶ 41:16

The screenshot shows the VS Code interface with the file `header.component.html` open. The code contains an `a` tag with the `[routerLinkActive]` directive set to a string value:

```
header.component.html > div.page-header > div > div.page-header-primary > div.page-header-primary-nav > a
21     <h2>proacademy</h2>
22   </div>
23   <div>
24     <div class="page-header-secondary">
25
26     </div>
27     <div class="page-header-primary">
28       <div class="page-header-primary-nav">
29         <a [routerLink="Home" [routerLinkActive]="'active'>">HOME</a>
30         <a [routerLink]="'About'>">ABOUT</a>
31         <a [routerLink]="'Contact'>">CONTACT</a>
32         <a [routerLink]="'Courses'>">COURSES</a>
33
34       </div>
35       <div class="page-header-login-logout">
36         <a href="#">LOGIN</a>
37         <a href="#">LOGOUT</a>
38
39       </div>
40     </div>
41   </div>
```

- The router link active directive requires single quotes to assign a string value to the attribute.

▶ 42:42

The screenshot shows the VS Code interface with the title bar "angular-routing". The left sidebar has a tree view with "EXPLORER" expanded, showing "ANGULAR-ROUTING", "node_modules", "src", "app", "about", "contact", "courses", "footer", "header", "header.component.css", "header.component.html", "header.component.ts", "home", "login", "Models", "not-found", "Services", "app.component.css", "app.component.html", and "app.component.spec.ts". The right pane displays the "header.component.html" file with code highlighting. The code includes a header with a logo, secondary navigation, primary navigation with links for Home, About, Contact, and Courses, and a login/logout section. The "Courses" link is highlighted with a larger font size and bold text, indicating it is the active route.

```
<h2>procademy</h2>
</div>
<div>
  <div class="page-header-secondary">
    ...
  </div>
  <div class="page-header-primary">
    <div class="page-header-primary-nav">
      <a routerLink="Home" [routerLinkActive]="'active'">HOME</a>
      <a routerLink="About" [routerLinkActive]="'active'">ABOUT</a>
      <a routerLink="Contact" [routerLinkActive]="'active'">CONTACT</a>
      <a routerLink="Courses" [routerLinkActive]="'active'">COURSES</a>
    </div>
    <div class="page-header-login-logout">
      <a href="#">LOGIN</a>
      <a href="#">LOGOUT</a>
    </div>
  </div>
</div>
```

- Website developer enhances user experience by highlighting active links with larger font size and bold text.

▶ 43:04

RouterLinkActive Directive

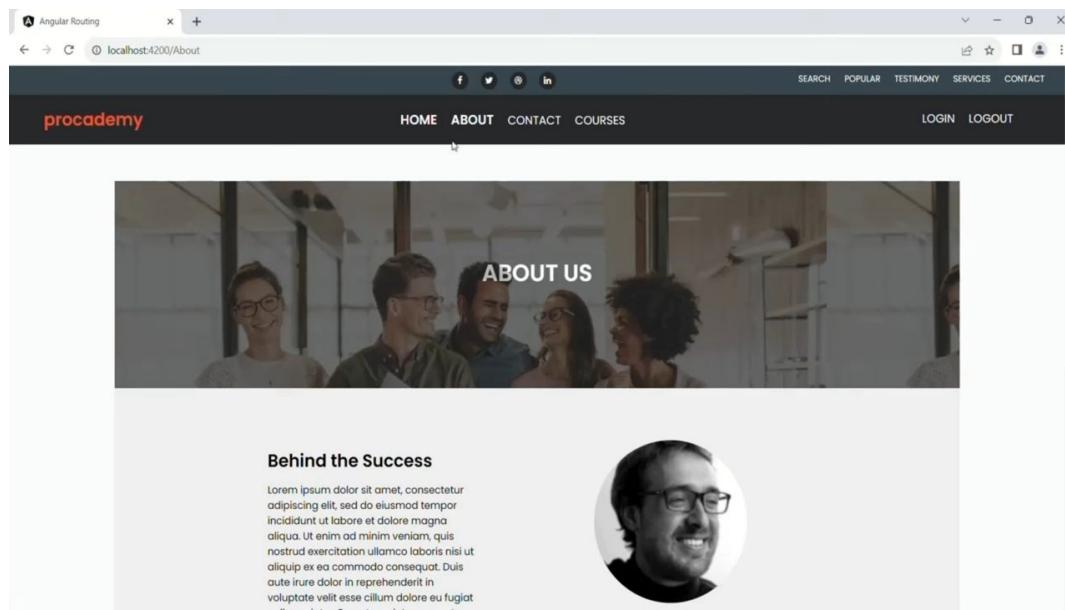
The **RouterLinkActive** is a directive for adding or removing classes from an HTML element that is bound to **RouteLink**.

Using **RouterLinkActive** directive, we can toggle CSS classes for active route link based on the current router state.

The main use case of **RouterLinkActive** directive is to highlight which route is currently active.

- The router link active directive is a useful tool for highlighting the active route in an Angular application.

▶ 43:37



- The about page has been rendered and is now the active link, but the home page is also shown as active due to its relationship as a child link.

▷ 45:26

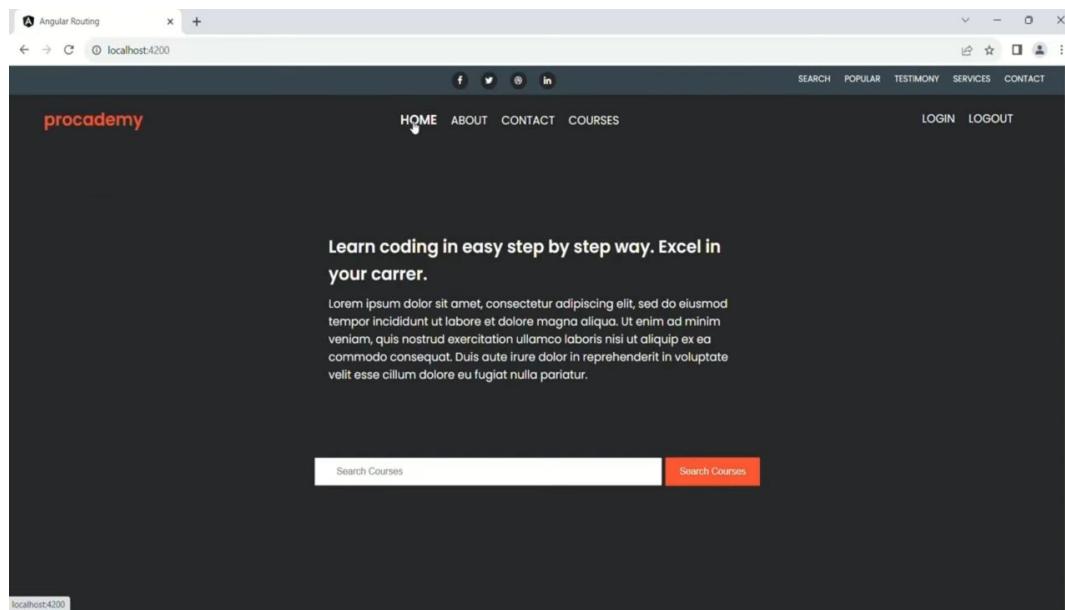
```

<!-- page-header-primary -->
<a href="#" [routerLinkActive]="'active'" [routerLinkActiveOptions]={{exact: true}}>
  <!-- routerLink="About" [routerLinkActive]="'active'" -->ABOUT</a>
<a href="#" [routerLinkActive]="'active'">CONTACT</a>
<a href="#" [routerLinkActive]="'active'">COURSES</a>

```

- Using the attribute directive, we can assign an object to ensure that the active CSS class only gets applied to routes whose path matches the current URL.

▷ 47:17



✨ Web page styling issue resolved with new route changes.

▷ 48:30

```
File Edit Selection View Go ... < > angular-routing
EXPLORER
  ANGULAR-ROUTING
    node_modules
    src
      app
        about
        contact
        courses
        footer
        header
          header.component.css
          header.component.html
          header.component.ts
        home
        login
        Models
        not-found
        Services
        app.component.css
        app.component.html
        app.component.spec.ts
  OUTLINE
  TIMELINE
  0 △ 0
```

```
header.component.html > div.page-header > div > div.page-header-primary > div.page-header-primary-nav > a
21
22
23
24  ondary">
25
26
27  mary">
28  primary-nav">
29  [routerLinkActive]="'active'" [routerLinkActiveOptions]="{{exact: true}}">>HOME</a>
30  [routerLinkActive]="'active'" [routerLinkActiveOptions]="{{exact: true}}">>ABOU
31  t" [routerLinkActive]="'active'" [routerLinkActiveOptions]="{{exact: true}}">>CONTAC
32  ts" [routerLinkActive]="'active'" [routerLinkActiveOptions]="{{exact: true}}">>COURSE
33
34  login-logout">
35  .
36  >
37
38
39
40
41
42
```

Ln 29, Col 106 (4 selected) Spaces: 2 UTF-8 LF HTML ⚡ Go Live ⌂ ⌂

✨ The issue of styling has been resolved in the lecture, with a focus on changing the path for the home link.

▷ 48:45

RouterLinkActiveOptions Directive

When a child route is active, then all the parent routes are also marked as active. In that case, **routeLinkActive** directive is applied to the active child route and all its parent routes.

Using **RouterLinkActiveOptions** directive, we can set some options for **routerLinkActive** directive. One of the options we can set is the **exact** property which tells how to match the route path for styling the active route.

```
[routerLinkActiveOptions] = "{exact: true}"
```

▶ 48:55

Relative vs Absolute Route Path

- Understanding the difference between relative and absolute route paths is crucial for navigating an Angular application effectively.

▶ 49:29

Absolute Path

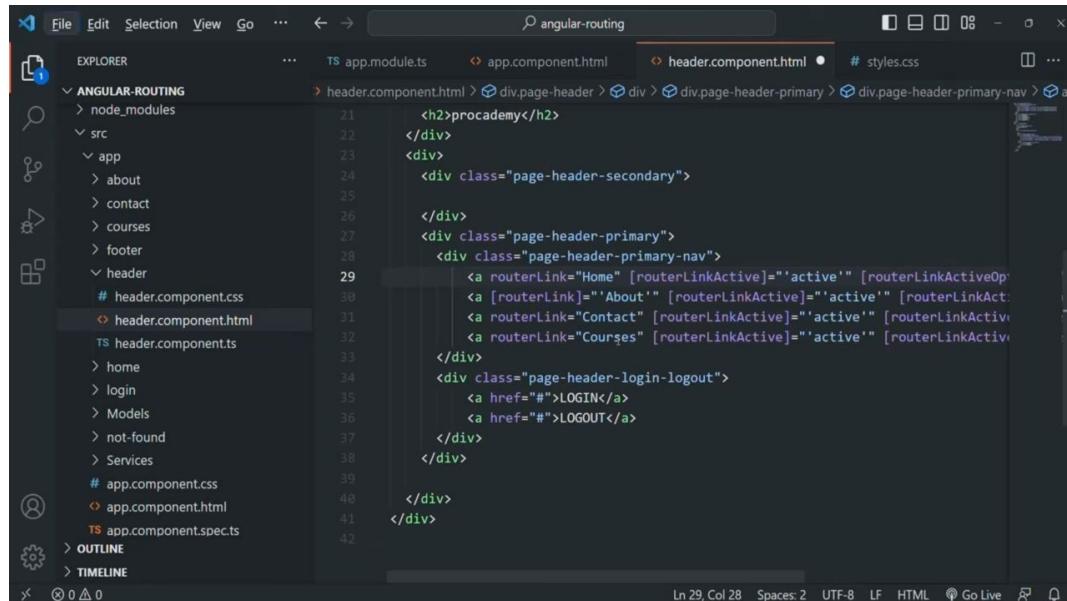
When we use a slash ("/") before the router link path, in that case it uses absolute path and the path is directly appended to root url.

```
<a routerLink="/About">About</a>
```

URL: localhost:4200/About

- When creating links in web development, the path is directly appended to the root URL.

▶ 50:53



The screenshot shows the VS Code interface with the title bar "angular-routing". The left sidebar shows the file structure under "ANGULAR-ROUTING": node_modules, src, app (with about, contact, courses, footer, header), Models, not-found, Services, and app component files. The main editor area displays the content of "header.component.html". The code includes an H2 tag "procademy", a secondary header section, and a primary navigation bar with five items: Home, About, Contact, Courses, and Log In/Out. The "About" item is highlighted with a red background, indicating it is the active route. The code uses routerLink and routerLinkActive directives from the RouterModule.

```
<h2>procademy</h2>
<div>
  <div class="page-header-secondary">
    <div>
      <div class="page-header-primary">
        <div class="page-header-primary-nav">
          <a routerLink="Home" [routerLinkActive]="'active'" [routerLinkActiveOp
          <a [routerLink]="'About'" [routerLinkActive]="'active'" [routerLinkAct
          <a routerLink="Contact" [routerLinkActive]="'active'" [routerLinkActiv
          <a routerLink="Courses" [routerLinkActive]="'active'" [routerLinkActiv
        </div>
        <div class="page-header-login-logout">
          <a href="#">LOGIN</a>
          <a href="#">LOGOUT</a>
        </div>
      </div>
    </div>
  </div>
</div>
```

- Understanding the use of slashes in file paths is crucial for navigating and linking within a website.

▶ 51:37

Relative Path

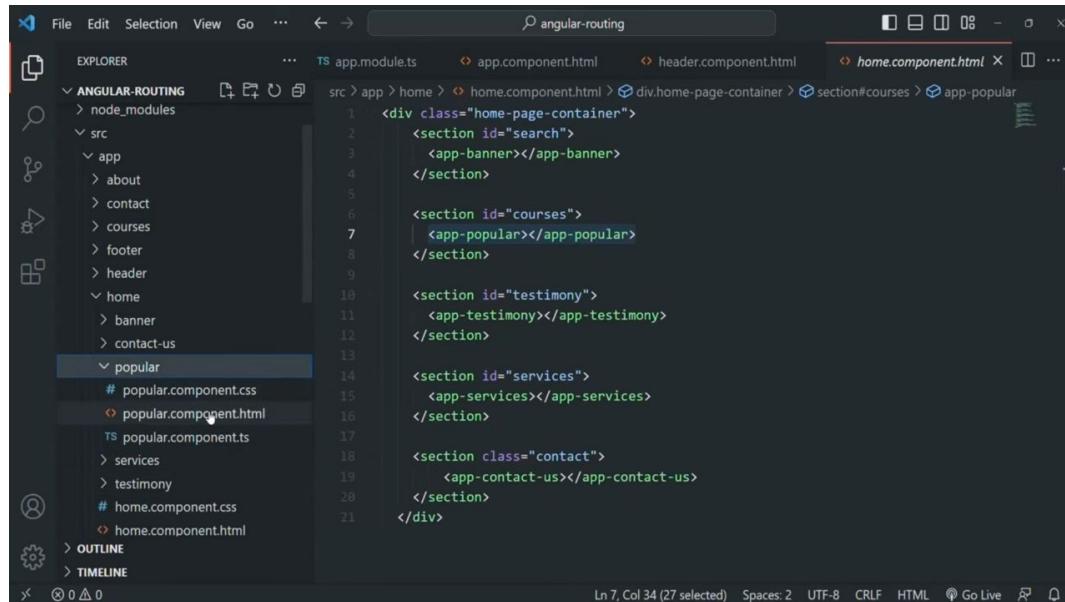
When we do not use a slash (“/”) before the router link path, in that case it uses relative path and the path is appended to the currently active route.

For example, let's say the currently active route is **About**. And a link is defined in About page to go to home page as shown below. If that link is clicked, the path will be appended to currently active route.

```
<a routerLink="Home">Go to Home</a>
```

URL: localhost:4200/About/Home

▷ 51:51



```
<div class="home-page-container">
  <section id="search">
    <app-banner></app-banner>
  </section>

  <section id="courses">
    | <app-popular></app-popular>
  </section>

  <section id="testimony">
    <app-testimony></app-testimony>
  </section>

  <section id="services">
    <app-services></app-services>
  </section>

  <section class="contact">
    <app-contact-us></app-contact-us>
  </section>
</div>
```

- ★ New app component "Popular" is being used to display popular causes and includes a button with a router link directive.

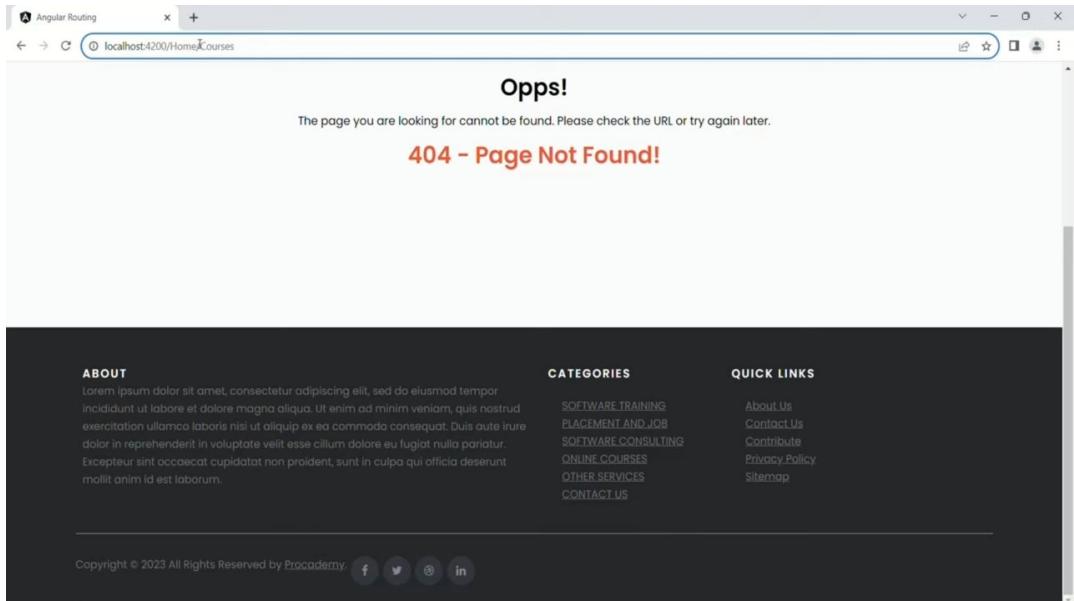
▷ 53:27

A screenshot of a code editor window titled "angular-routing". The left sidebar shows a file tree with "ANGULAR-ROUTING" expanded, containing "node_modules", "src", "app", "about", "contact", "courses", "footer", "header", "home", "popular", and "services". The "popular" folder contains "popular.component.css", "popular.component.html", and "popular.component.ts". The main pane displays the "popular.component.html" file with the following code:

```
<div class="course-title">
  {{ course.title }}
</div>
<div class="course-desc">
  {{ course.desc.substring(0, 120) }}
</div>
<div class="course-price-rating">
  <div class="course-price"><b>PRICE:</b> {{ course.price | currency }}</div>
  <div class="course-rating"><b>RATINGS:</b> {{ course.rating }}</div>
</div>
<div class="course-action-buttons">
  <button class="btn">BUY NOW</button>
  <button class="btn">DETAILS</button>
</div>
<div class="popular-course-bottom">
  <button class="go-to-course-button" routerLink="Courses">
    Go to Courses Page
    <i class="fa fa-arrow-right" aria-hidden="true"></i>
  </button>
</div>
```

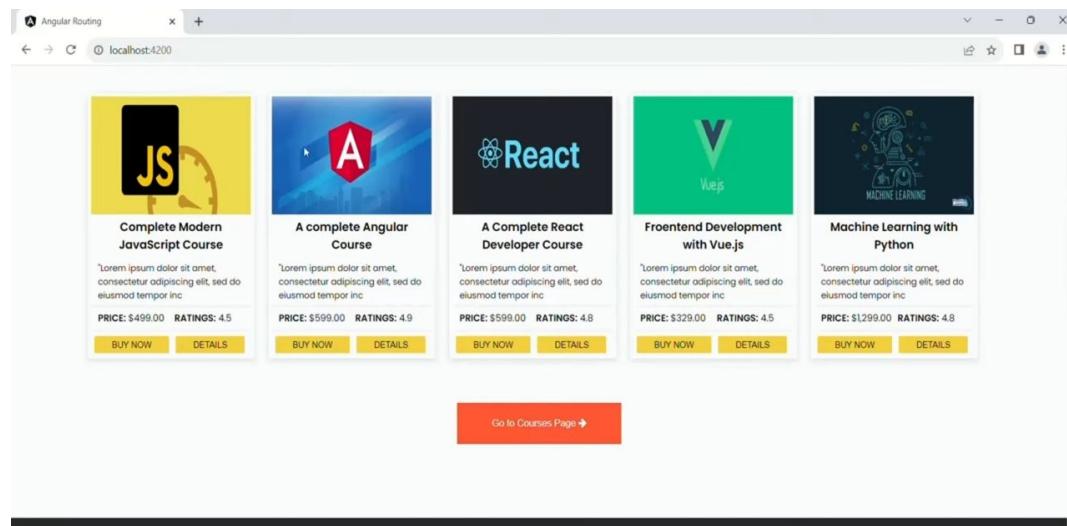
★ Learn how to use router links in HTML elements to redirect users to specific pages on your website.

▷ 54:04



★ Using relative paths in URLs can lead to unexpected results when navigating through a website.

▷ 54:51



What Our Customer Thinks

💡 Navigating through the website's routes and components is made easy with the seamless appending of paths.

▷ 55:25

```
<div class="course-title">
    {{ course.title }}
</div>
<div class="course-desc">
    {{ course.desc.substring(0, 120) }}
</div>
<div class="course-price-rating">
    <div class="course-price"><b>PRICE:</b> {{ course.price | currency }}</div>
    <div class="course-rating"><b>RATINGS:</b> {{ course.rating }}</div>
</div>
<div class="course-action-buttons">
    <button class="btn">BUY NOW</button>
    <button class="btn">DETAILS</button>
</div>
<div class="popular-course-bottom">
    <button class="go-to-course-button" routerLink="/Courses">
        Go to Courses Page
        <i class="fa fa-arrow-right" aria-hidden="true"></i>
    </button>
</div>
```

💡 Using absolute paths in web development ensures that the specified URL will always be appended to the root URL, regardless of the current active route.

▷ 56:17

Relative Path

When we use a dot & a slash (".") before the router link path, in that case it uses relative path and the path is appended to the currently active route.

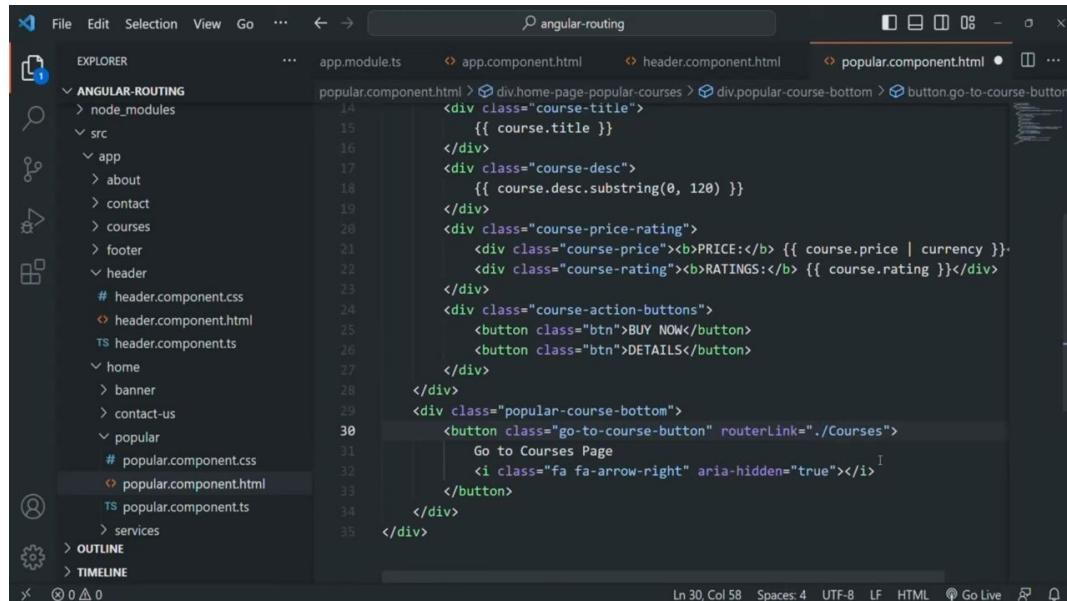
Current Active Router Link Path: `localhost:4200/Books/Author`

```
<a routerLink=".//Stephen-King">Stephen King</a>
```

URL: `localhost:4200/Books/Author/Stephen-King`

- Understanding the difference between absolute and relative paths is crucial for navigating URLs effectively.

▶ 1:00:02



The screenshot shows a code editor interface with the title bar "angular-routing". The left sidebar has sections for "EXPLORER", "ANGULAR-ROUTING", "OUTLINE", and "TIMELINE". The "ANGULAR-ROUTING" section is expanded, showing a tree structure of files: "node_modules", "src", "app", "about", "contact", "courses", "footer", "header", "home", "popular", and "services". The "header" folder contains "header.component.css", "header.component.html", and "header.component.ts". The "popular" folder contains "popular.component.css", "popular.component.html", and "popular.component.ts". The "app.module.ts" file is open in the main editor area, showing the following code:

```
14     <div class="course-title">
15         {{ course.title }}
16     </div>
17     <div class="course-desc">
18         {{ course.desc.substring(0, 120) }}
19     </div>
20     <div class="course-price-rating">
21         <div class="course-price"><b>PRICE:</b> {{ course.price | currency }}</div>
22         <div class="course-rating"><b>RATINGS:</b> {{ course.rating }}</div>
23     </div>
24     <div class="course-action-buttons">
25         <button class="btn">BUY NOW</button>
26         <button class="btn">DETAILS</button>
27     </div>
28 </div>
29 <div class="popular-course-bottom">
30     <button class="go-to-course-button" routerLink=".//Courses">
31         Go to Courses Page
32         <i class="fa fa-arrow-right" aria-hidden="true"></i>
33     </button>
34 </div>
35 </div>
```

- Understanding the difference between absolute and relative paths is crucial for navigating web pages effectively.

▶ 1:00:22

Relative Path

When we use (“..”) before the router link path, in that case it will move one level up and the path will be appended to the parent path

Current Active Router Link Path: **localhost:4200/Books/Author**

```
<a routerLink="..../Stephen-King">Stephen King</a>
```

URL: **localhost:4200/Books/Stephen-King**

- Understanding how to use two dot and a slash before the router link path can help navigate through URL paths more effectively.

▷ 1:01:16

Relative Path

When we use (“..../..”) before the router link path, in that case it will move two level up and the path will be appended after remove last two paths from the URL

Current Active Router Link Path: **localhost:4200/Books/Author**

```
<a routerLink="..../..../Stephen-King">Stephen King</a>
```

URL: **localhost:4200/Stephen-King**

- URL paths can be manipulated to navigate to different levels and append new paths.

▷ 1:02:17

Navigating between Routes Programmatically

- ★ Absolute and relative paths are essential for defining router link paths in Angular, allowing for seamless navigation between routes.

▷ 1:02:56

RouterLink Directive

RouterLink is a directive which we use to bind any clickable HTML element to a route. When that HTML element is clicked, the router will navigate to the associated route.

RouterLink is a directive which we use in the view template of a component to navigate between routes.

```
<a routerLink="Home">Home</a>
```

- ★ In this lecture, you will learn how to navigate between routes programmatically in Angular.

▷ 1:03:13

The screenshot shows a code editor interface with the title bar "angular-routing". The left sidebar is titled "EXPLORER" and lists project files under "ANGULAR-ROUTING". The main editor area displays the "popular.component.html" file. The code is an Angular template for a course card. It includes sections for course title, description, price, rating, and action buttons for buying or viewing details. A button at the bottom right is annotated with a tooltip: "Go to Courses Page" and "RATINGS". The status bar at the bottom indicates "Ln 30, Col 71 (17 selected)" and "Spaces: 4".

```
<div class="course-title">
    {{ course.title }}
</div>
<div class="course-desc">
    {{ course.desc.substring(0, 120) }}
</div>
<div class="course-price-rating">
    <div class="course-price"><b>PRICE:</b> {{ course.price | currency }}</div>
    <div class="course-rating"><b>RATINGS:</b> {{ course.rating }}</div>
</div>
<div class="course-action-buttons">
    <button class="btn">BUY NOW</button>
    <button class="btn">DETAILS</button>
</div>
<div class="popular-course-bottom">
    <button class="go-to-course-button" (click)="navigateToCourses()">
        Go to Courses Page
        <i class="fa fa-arrow-right" aria-hidden="true"></i>
    </button>
</div>
```

- ★ A new method called "getCourses" has been assigned to the popular component class for retrieving courses.

▷ 1:04:35

navigate() Method

Using **navigate()** method, we can navigate from one route to another programmatically. The navigate method takes an array as an argument and in that array we can specify route segments.

Let's say, we want to navigate the user to following route:

URL: `localhost:4200/Books/Author/101`

We can pass each segment of above URL as an element of the passed array to navigate method.

```
this._router.navigate(['Books', 'Author', 101]);
```

- ★ The navigate method on a router allows for programmatically navigating between routes using an array of route segments as arguments.

▷ 1:06:28

```
File Edit Selection View Go ... angular-routing
EXPLORER
ANGULAR-ROUTING
src > app > home > popular > TS popular.component.ts > PopularComponent > navigateToCourses
1 import { Component, inject } from '@angular/core';
2 import { Course } from 'src/app/Models/course';
3 import { CourseService } from 'src/app/Services/course.service';
4 import { Router } from '@angular/router';
5
6 @Component({
7   selector: 'app-popular',
8   templateUrl: './popular.component.html',
9   styleUrls: ['./popular.component.css']
10})
11 export class PopularComponent {
12   courseService = inject(CourseService);
13   popularCourses: Course[] = [];
14   router: Router = inject(Router);
15
16   ngOnInit(){
17     this.popularCourses = this.courseService.courses.filter(c => c.rating >= 4.5);
18   }
19
20   navigateToCourses(){
21     this.router.navigate(['Courses']);
22   }
23 }

```

Ln 21, Col 25 (8 selected) Spaces: 2 UTF-8 LF () TypeScript ⓘ Go Live ⌂ ⌂

- ★ Successful redirection to the courses page using the `navigate` method on the `router` object has been achieved.

▶ 1:08:34

navigateByUrl() Method

Using `navigateByUrl()` method, we can navigate from one route to another programmatically. The `navigateByUrl` method takes a string value as an argument and that string value should contain all the route segments.

Let's say, we want to navigate the user to following route:

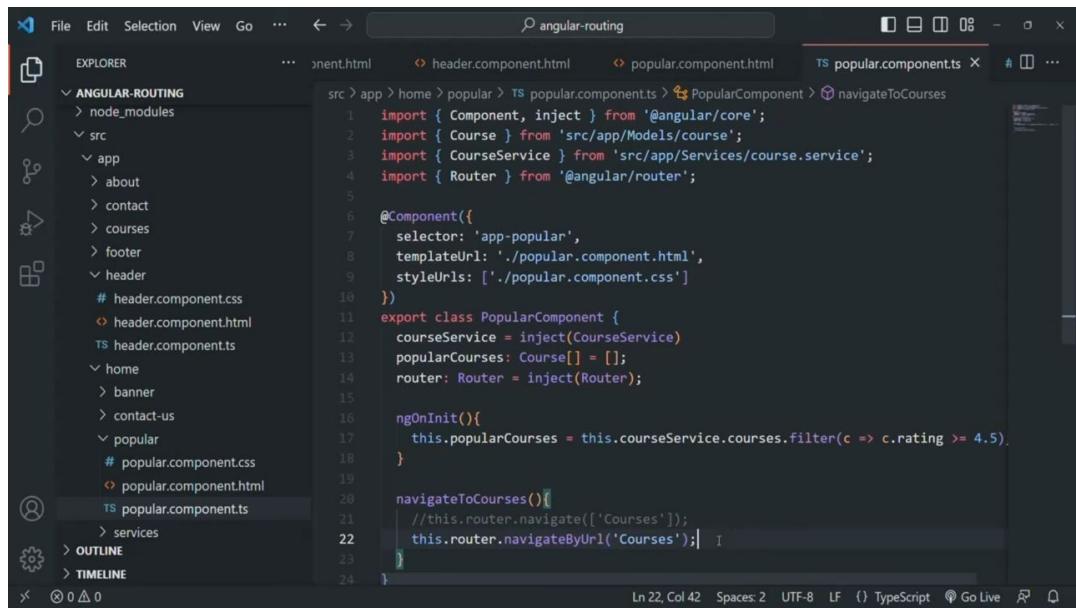
URL: `localhost:4200/Books/Author/101`

We can pass each segment of above URL as an element of the passed array to `navigate` method.

```
this._router.navigateByUrl('Books/Author/101');
```

- ★ The `navigate by URL` method on the `router` object allows for programmatic navigation between routes using a string value containing all route segments.

▶ 1:08:58

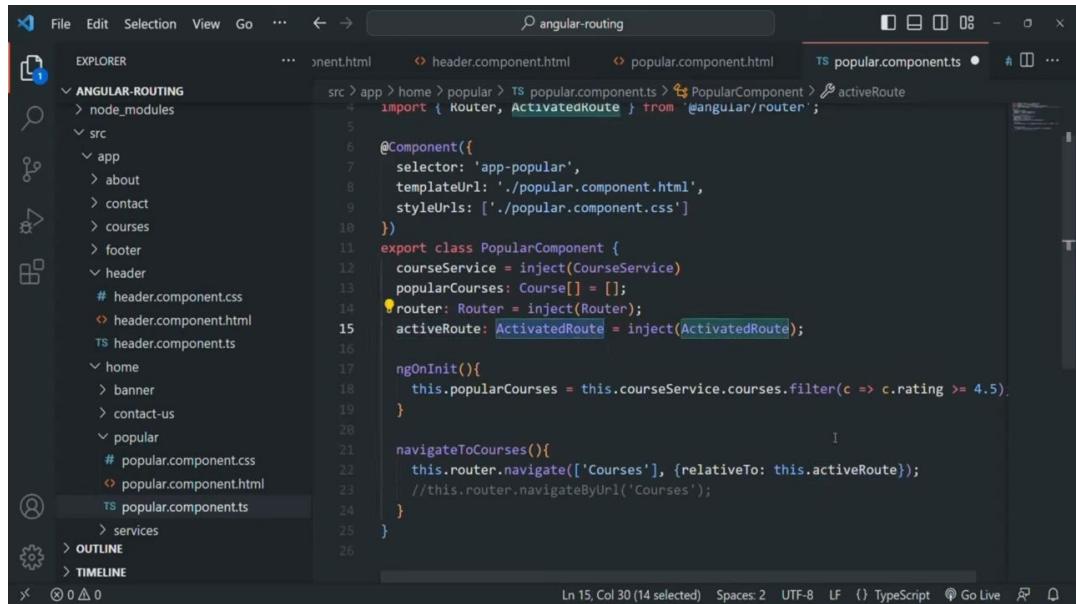


```
src > app > home > popular > TS popular.component.ts > PopularComponent > navigateToCourses
1 import { Component, inject } from '@angular/core';
2 import { Course } from 'src/app/Models/course';
3 import { CourseService } from 'src/app/Services/course.service';
4 import { Router } from '@angular/router';
5
6 @Component({
7   selector: 'app-popular',
8   templateUrl: './popular.component.html',
9   styleUrls: ['./popular.component.css']
10})
11 export class PopularComponent {
12   courseService = inject(CourseService);
13   popularCourses: Course[] = [];
14   router: Router = inject(Router);
15
16 ngOnInit(){
17   this.popularCourses = this.courseService.courses.filter(c => c.rating >= 4.5);
18 }
19
20 navigateToCourses(){
21   //this.router.navigate(['Courses']);
22   this.router.navigateByUrl('Courses');
23 }
24 }
```

Ln 22, Col 42 Spaces: 2 UTF-8 LF () TypeScript ⓘ Go Live ⌂ ⌂

The navigation method on the website successfully directs users to the courses page, providing a seamless user experience.

▶ 1:09:46



```
src > app > home > popular > TS popular.component.ts > PopularComponent > activeRoute
1 import { Router, ActivatedRoute } from '@angular/router';
2
3 @Component({
4   selector: 'app-popular',
5   templateUrl: './popular.component.html',
6   styleUrls: ['./popular.component.css']
7 })
8 export class PopularComponent {
9   courseService = inject(CourseService);
10  popularCourses: Course[] = [];
11  router: Router = inject(Router);
12  activeRoute: ActivatedRoute = inject(ActivatedRoute);
13
14 ngOnInit(){
15   this.popularCourses = this.courseService.courses.filter(c => c.rating >= 4.5);
16 }
17
18 navigateToCourses(){
19   this.router.navigate(['Courses'], {relativeTo: this.activeRoute});
20   //this.router.navigateByUrl('Courses');
21 }
22
23 }
```

Ln 15, Col 30 (14 selected) Spaces: 2 UTF-8 LF () TypeScript ⓘ Go Live ⌂ ⌂

The active route in the navigate method stores information about the currently active route.

▶ 1:13:17

Working with Route Parameters

- Angular route parameters allow for dynamic values to be passed to a given route, providing extra information and functionality.

▷ 1:14:56

Route Parameter

The route parameters are the dynamic part of the route whose value can change. These parameters provides a way to pass extra information to a given route.

```
localhost:4200/Books/Author/Stefen-King
localhost:4200/Books/Author/Sarah-Williams
localhost:4200/Books/Author/John-Smith
```

▷ 1:15:05

Route Parameter

The route parameters are the dynamic part of the route whose value can change. These parameters provides a way to pass extra information to a given route.

```
localhost:4200/Books/23432
localhost:4200/Books/54367
localhost:4200/Books/31728
```

- ✿ Using route parameters in Angular allows for dynamic changes to the URL path, providing flexibility and customization for web applications.

▷ 1:15:50

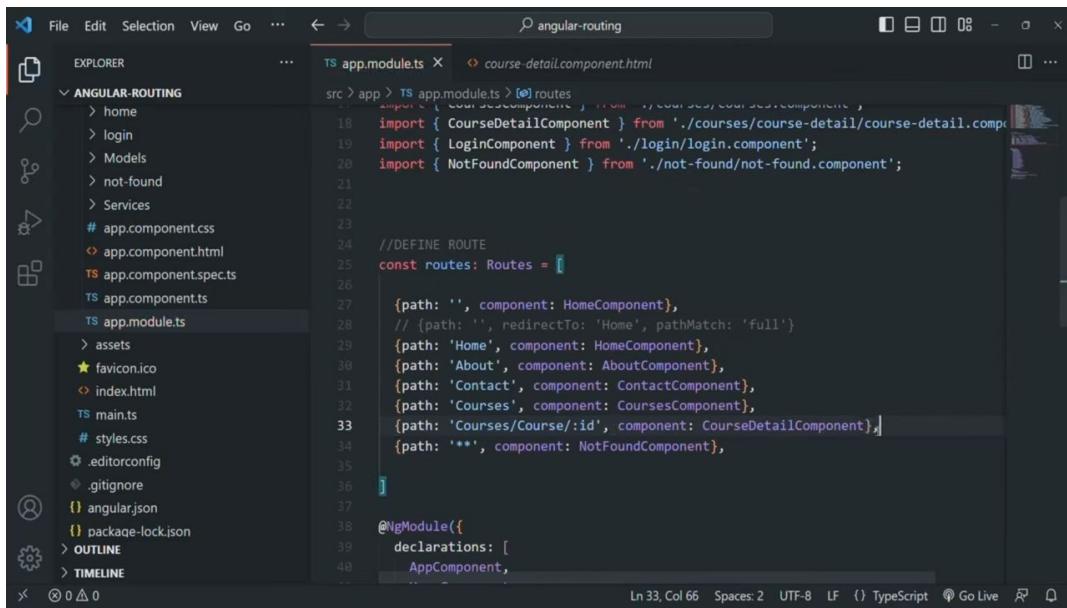
Route Parameter

The route parameters are the dynamic part of the route whose value can change. These parameters provides a way to pass extra information to a given route.

```
localhost:4200/Books/:id
localhost:4200/Books/Author/:author
```

- ✿ Angular allows for dynamic routing by using route parameters, which can be specified with a colon before the parameter name.

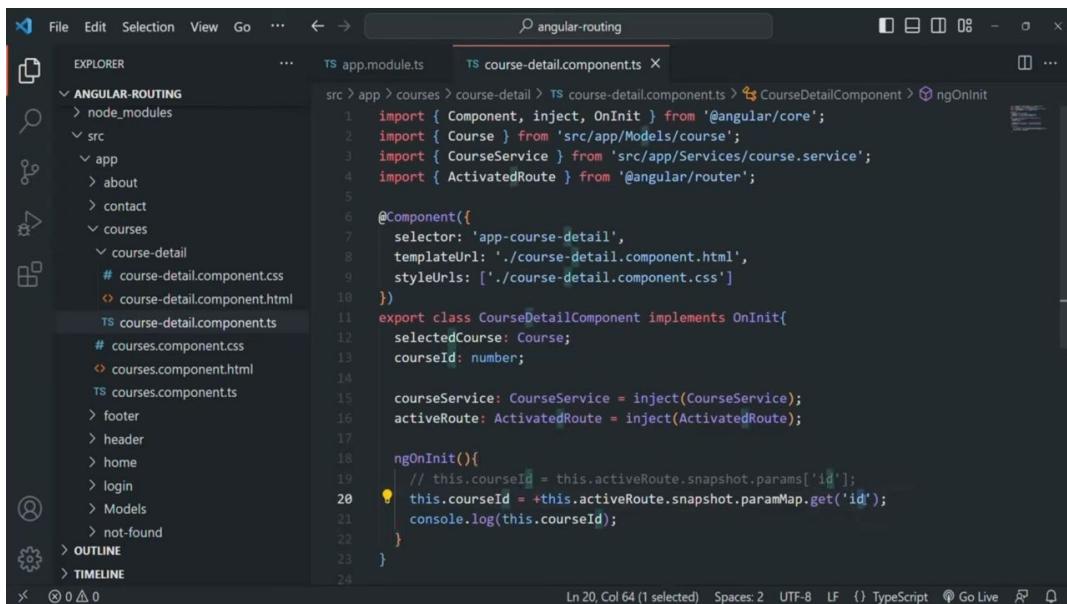
▷ 1:16:20



```
src > app > TS app.module.ts > routes
18 import { CourseDetailComponent } from './courses/course-detail/course-detail.component';
19 import { LoginComponent } from './login/login.component';
20 import { NotFoundComponent } from './not-found/not-found.component';
21
22
23 //DEFINE ROUTE
24 const routes: Routes = [
25   {path: '', component: HomeComponent},
26   // {path: '', redirectTo: 'Home', pathMatch: 'full'}
27   {path: 'Home', component: HomeComponent},
28   {path: 'About', component: AboutComponent},
29   {path: 'Contact', component: ContactComponent},
30   {path: 'Courses', component: CoursesComponent},
31   {path: 'Courses/Course/:id', component: CourseDetailComponent},
32   {path: '**', component: NotFoundComponent},
33
34
35
36
37
38 @NgModule({
39   declarations: [
40     AppComponent,
```

- Using the component property, users can assign course detail components to specific URLs for easy access.

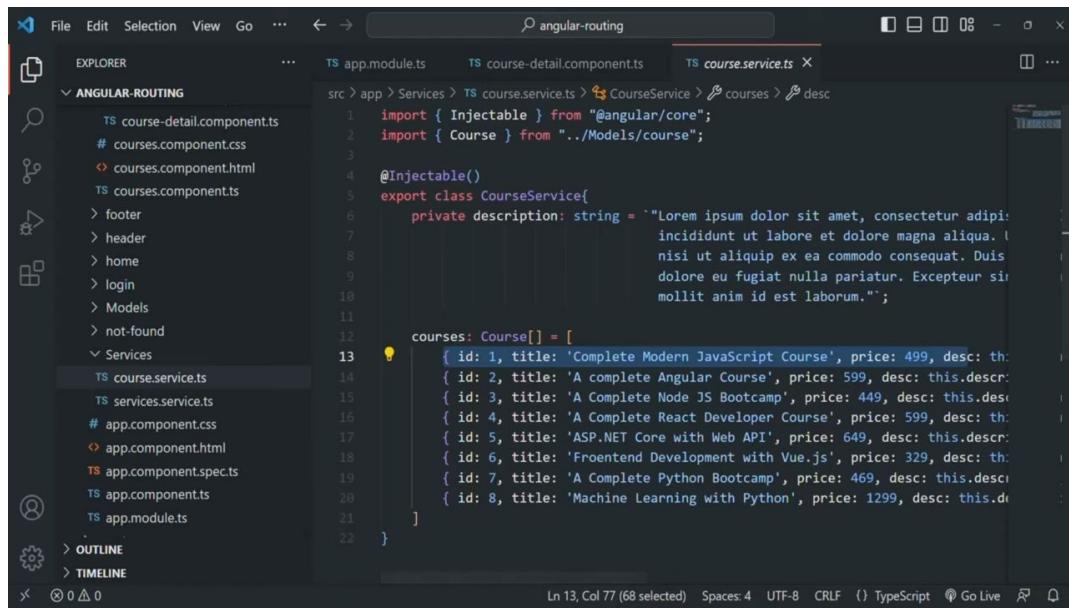
▶ 1:22:11



```
src > app > courses > course-detail > TS course-detail.component.ts > CourseDetailComponent > ngOnInit
1 import { Component, inject, OnInit } from '@angular/core';
2 import { Course } from 'src/app/Models/course';
3 import { CourseService } from 'src/app/Services/course.service';
4 import { ActivatedRoute } from '@angular/router';
5
6 @Component({
7   selector: 'app-course-detail',
8   templateUrl: './course-detail.component.html',
9   styleUrls: ['./course-detail.component.css']
10 })
11 export class CourseDetailComponent implements OnInit {
12   selectedCourse: Course;
13   courseId: number;
14
15   courseService: CourseService = inject(CourseService);
16   activeRoute: ActivatedRoute = inject(ActivatedRoute);
17
18   ngOnInit(){
19     // this.courseId = this.activeRoute.snapshot.params['id'];
20     this.courseId = +this.activeRoute.snapshot.paramMap.get('id');
21     console.log(this.courseId);
22   }
23 }
```

- A new method of logging and filtering course values based on ID route parameters has been implemented in the courses service.

▶ 1:30:26



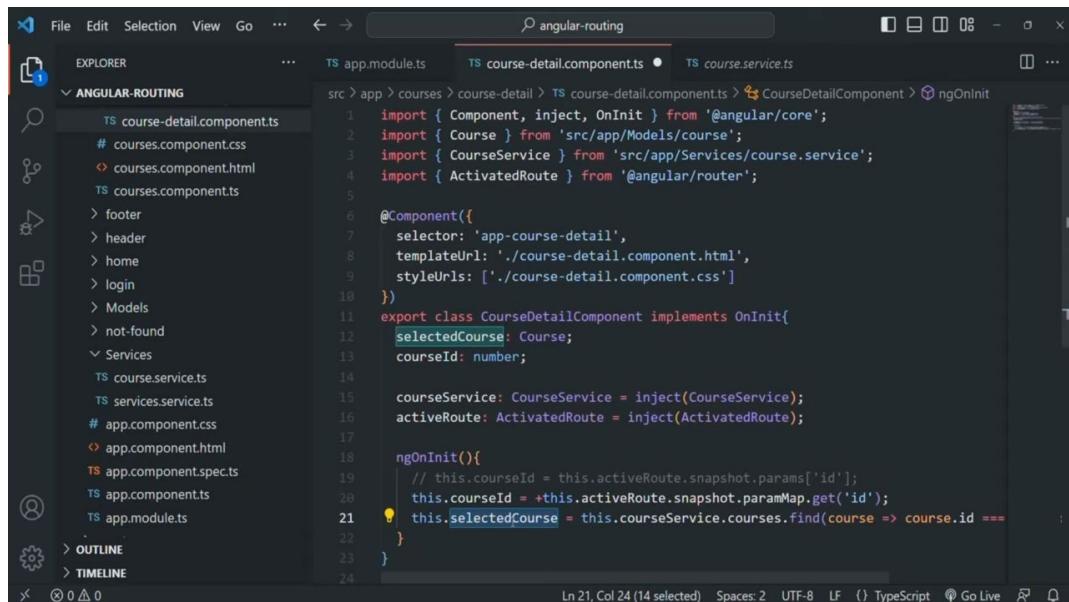
```
import { Injectable } from '@angular/core';
import { Course } from '../Models/course';

@Injectable()
export class CourseService{
  private description: string = `Lorem ipsum dolor sit amet, consectetur adipisci  
incididunt ut labore et dolore magna aliqua. Ut  
nisi ut aliquip ex ea commodo consequat. Duis  
dolore eu fugiat nulla pariatur. Excepteur si  
mollit anim id est laborum.`;

  courses: Course[] = [
    { id: 1, title: 'Complete Modern JavaScript Course', price: 499, desc: this.description },
    { id: 2, title: 'A Complete Angular Course', price: 599, desc: this.description },
    { id: 3, title: 'A Complete Node JS Bootcamp', price: 449, desc: this.description },
    { id: 4, title: 'A Complete React Developer Course', price: 599, desc: this.description },
    { id: 5, title: 'ASP.NET Core with Web API', price: 649, desc: this.description },
    { id: 6, title: 'Frontend Development with Vue.js', price: 329, desc: this.description },
    { id: 7, title: 'A Complete Python Bootcamp', price: 469, desc: this.description },
    { id: 8, title: 'Machine Learning with Python', price: 1299, desc: this.description }
  ]
}
```

Filtering course details based on the ID route parameter is a crucial step in displaying specific course information in the course detail component.

▶ 1:30:57



```
import { Component, inject, OnInit } from '@angular/core';
import { Course } from 'src/app/Models/course';
import { CourseService } from 'src/app/Services/course.service';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-course-detail',
  templateUrl: './course-detail.component.html',
  styleUrls: ['./course-detail.component.css']
})
export class CourseDetailComponent implements OnInit{
  selectedCourse: Course;
  courseId: number;

  courseService: CourseService = inject(CourseService);
  activeRoute: ActivatedRoute = inject(ActivatedRoute);

  ngOnInit(){
    // this.courseId = this.activeRoute.snapshot.params['id'];
    // this.courseId = +this.activeRoute.snapshot.paramMap.get('id');
    this.selectedCourse = this.courseService.courses.find(course => course.id === this.courseId);
  }
}
```

New method for displaying course details has been implemented in the course detail component.

▶ 1:33:18

```
File Edit Selection View Go ... < > angular-routing
EXPLORER ... .ts course-detail.component.ts course-detail.component.html courses.component.html ...
ANGULAR-ROUTING iv.courses-page-all-courses > div.courses-page-course-list > div.course-card > div.course-action-buttons > button.btn
> about 18 <div class="course-desc">
> contact 19 {{ course.desc.substring(0, 120) }}
courses 20 </div>
<div class="course-price-rating">
21 <div class="course-price"><b>PRICE:</b> {{ course.price | currency }}</div>
22 <div class="course-rating"><b>RATINGS:</b> {{ course.rating }}</div>
23 </div>
24 <div class="course-action-buttons">
25 <button class="btn" [routerLink]="/Courses/Course/" + course.id>
26 BUY NOW</button>
27 </div>
28 </div>
29 </div>
30 </div>
31 </div>
32 </div>
33 </div>
TS course-detail.component.ts
# course-detail.component.css
course-detail.component.html
TS course-detail.component.ts
# courses.component.css
courses.component.html
TS courses.component.ts
> footer
> header
> home
> login
> Models
> not-found
Services
TS course.service.ts
TS courses.service.ts
> OUTLINE
> TIMELINE
x 0 0 △ 0
```

Line 27, Col 60 Spaces: 4 UTF-8 LF HTML ⚡ Go Live ⌂ ⌂

Using absolute paths instead of relative paths is crucial for ensuring that the correct URLs are accessed in web development.

▶ 1:37:33



The new course details display feature in the app module allows for easy navigation and access to specific course information.

▶ 1:37:59

Using Observable to Retrieve Route Parameter

- ★ Learn how to dynamically display content based on route parameters in Angular.

▷ 1:39:29

Snapshot property

The snapshot property contains the current value of the route. If the value of the route parameter changes after we have retrieved the value of route, that change will not get captured.

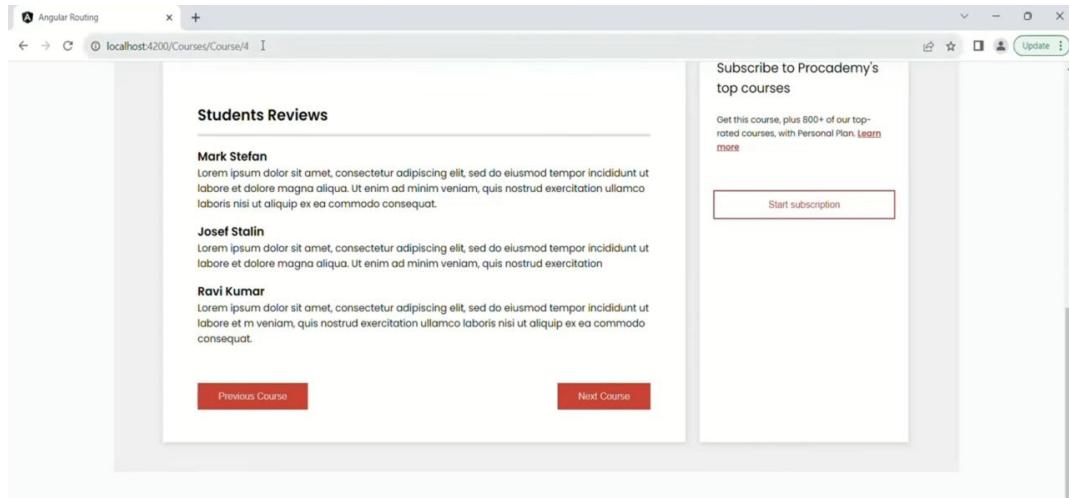
```
this.activeRoute.snapshot.paramMap.get('id');
```

▷ 1:39:37

A screenshot of a code editor window titled "angular-routing". The left sidebar shows a file tree with "ANGULAR-ROUTING" expanded, containing "node_modules", "src", "app", "about", "contact", "courses", and "course-detail". Under "course-detail", there are files: "course-detail.component.html", "course-detail.component.ts", "course.component.css", "courses.component.html", "courses.component.ts", "footer", "header", "home", "login", "Models", and "not-found". The main pane displays a portion of "course-detail.component.html" with code related to course reviews and navigation buttons. The status bar at the bottom shows "Ln 51, Col 103" and "Spaces: 4".

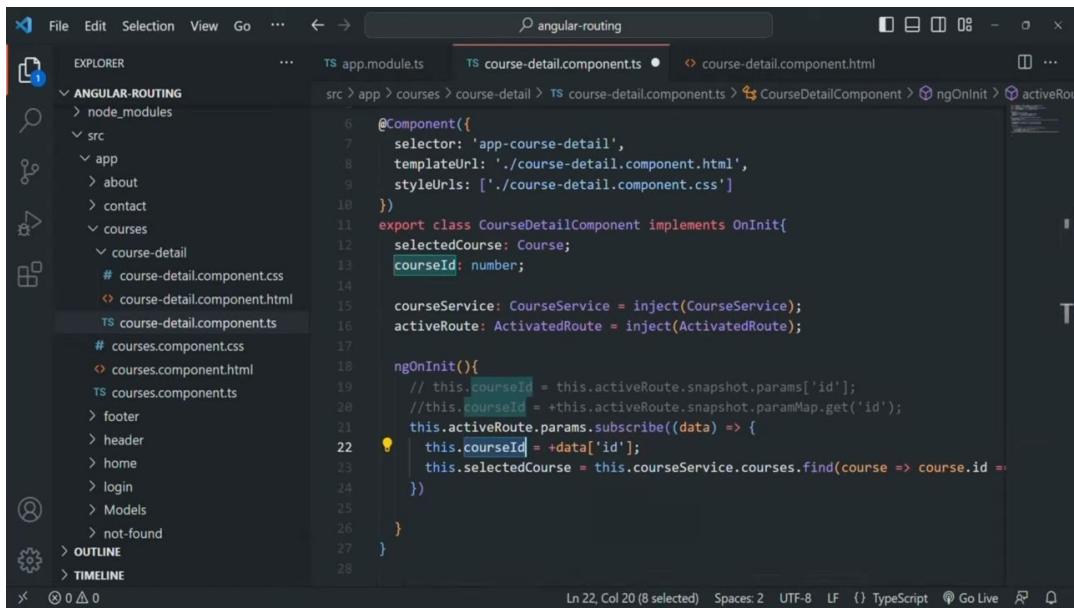
- ★ A programming error was fixed on the website, allowing users to access course details without any issues.

▷ 1:42:30



- ★ Course navigation on the new platform has been causing issues for users due to a glitch in the ID system.

▷ 1:43:10

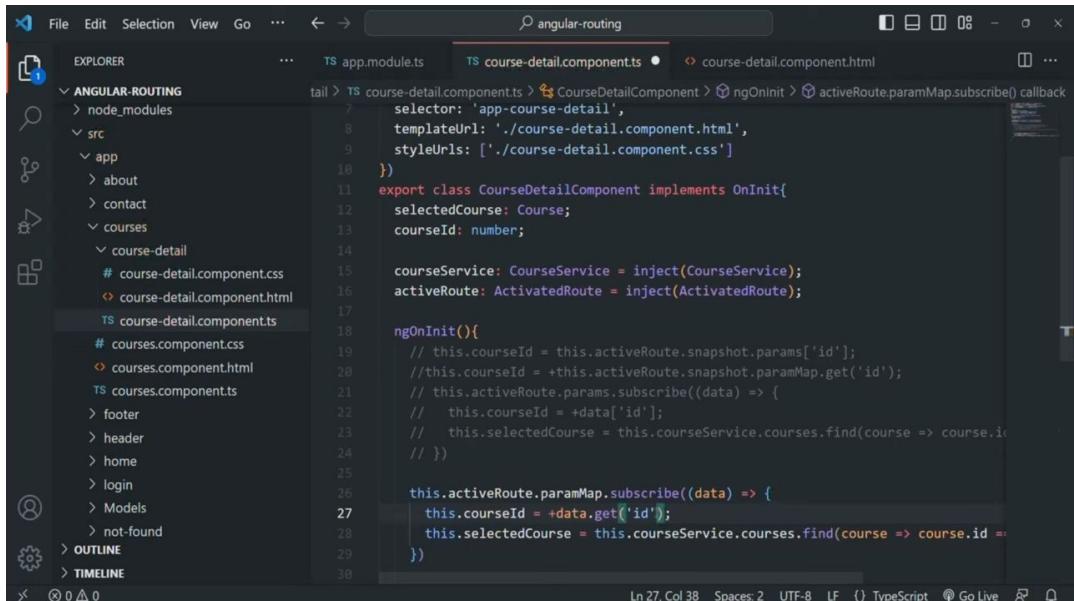


```
6  @Component({
7    selector: 'app-course-detail',
8    templateUrl: './course-detail.component.html',
9    styleUrls: ['./course-detail.component.css']
10   })
11  export class CourseDetailComponent implements OnInit{
12    selectedCourse: Course;
13    courseId: number;
14
15    courseService: CourseService = inject(CourseService);
16    activeRoute: ActivatedRoute = inject(ActivatedRoute);
17
18    ngOnInit(){
19      // this.courseId = this.activeRoute.snapshot.params['id'];
20      //this.courseId = +this.activeRoute.snapshot.paramMap.get('id');
21      this.activeRoute.params.subscribe((data) => {
22        this.courseId = +data['id'];
23        this.selectedCourse = this.courseService.courses.find(course => course.id =
24      })
25
26    }
27  }
28
```

Ln 22, Col 20 (8 selected) Spaces: 2 UTF-8 LF {} TypeScript Go Live

- ✿ New route values are being used to select courses based on the ID parameter in a web page.

▷ 1:49:16



```
tail > TS course-detail.component > CourseDetailComponent > ngOnInit > activeRoute.paramMap.subscribe() callback
6  @Component({
7    selector: 'app-course-detail',
8    templateUrl: './course-detail.component.html',
9    styleUrls: ['./course-detail.component.css']
10   })
11  export class CourseDetailComponent implements OnInit{
12    selectedCourse: Course;
13    courseId: number;
14
15    courseService: CourseService = inject(CourseService);
16    activeRoute: ActivatedRoute = inject(ActivatedRoute);
17
18    ngOnInit(){
19      // this.courseId = this.activeRoute.snapshot.params['id'];
20      //this.courseId = +this.activeRoute.snapshot.paramMap.get('id');
21      // this.activeRoute.params.subscribe((data) => {
22      //   this.courseId = +data['id'];
23      //   this.selectedCourse = this.courseService.courses.find(course => course.id =
24      // })
25
26      this.activeRoute.paramMap.subscribe((data) => {
27        this.courseId = +data.get('id');
28        this.selectedCourse = this.courseService.courses.find(course => course.id =
29      })
30    }
31  }
32
```

Ln 27, Col 38 Spaces: 2 UTF-8 LF {} TypeScript Go Live

- ✿ Utilizing the get method, we can retrieve the value of a parameter from the currently active route in order to ensure seamless functionality on web pages.

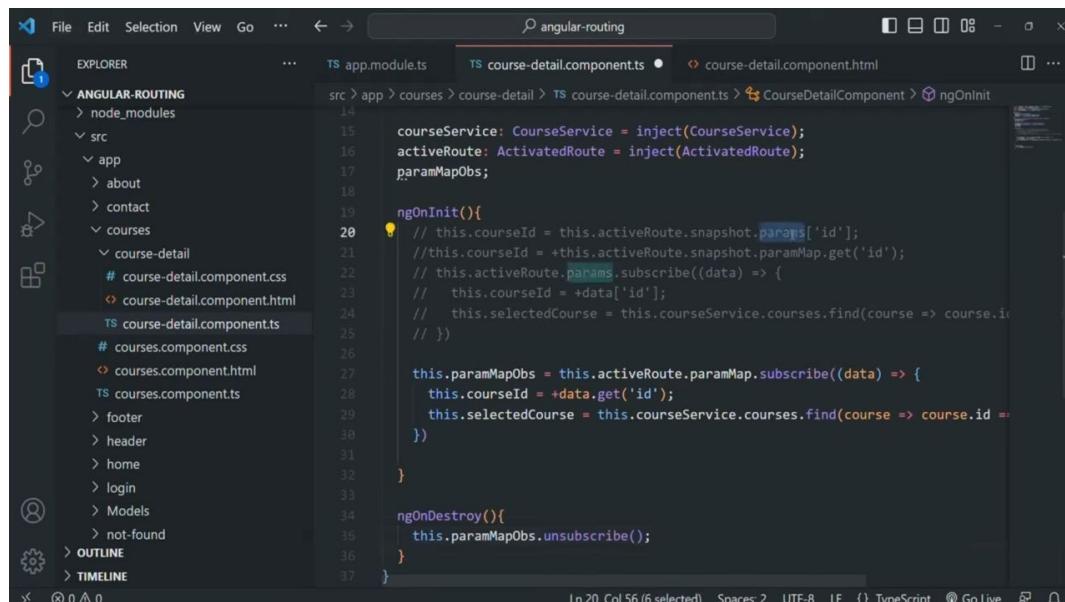
▷ 1:50:58

The paramMap Observable

By subscribing to the paramMap observable (or to 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.

- ★ Subscribing to route parameter observables allows for real-time updates and ensures proper component functionality.

▷ 1:51:49



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "ANGULAR-ROUTING".
- Code Editor:** Displays the file `course-detail.component.ts`. The code uses the `paramMap` observable to handle route parameters:

```
courseService: CourseService = inject(CourseService);
activeRoute: ActivatedRoute = inject(ActivatedRoute);
paramMapObs;

ngOnInit(){
    // this.courseId = this.activeRoute.snapshot.params['id'];
    // this.courseId = +this.activeRoute.snapshot.paramMap.get('id');
    // this.activeRoute.params.subscribe((data) => {
    //     this.courseId = +data['id'];
    //     this.selectedCourse = this.courseService.courses.find(course => course.id === this.courseId);
    // })

    this.paramMapObs = this.activeRoute.paramMap.subscribe((data) => {
        this.courseId = +data.get('id');
        this.selectedCourse = this.courseService.courses.find(course => course.id === this.courseId);
    })
}

ngOnDestroy(){
    this.paramMapObs.unsubscribe();
}
```

- Status Bar:** Shows the current line (Ln 20, Col 56), spaces used (Spaces: 2), encoding (UTF-8), line endings (LF), and the TypeScript language service status.

▷ 1:53:51

Using Query String in Route

- ★ The use of the params observable is deprecated in Angular, and it is recommended to use the param map property instead for reading Route parameters.

▷ 1:54:03

What is Query String

Query Strings are the optional data that we can pass to a component through a route. These query strings are added at the end of the route after a ?

```
localhost:4200/Books/Author?name=stefen-king  
localhost:4200/Products/product?id=12345&name=iphone
```

- ★ Learn how to specify and retrieve query string values in Angular routes for passing optional data to components.

▷ 1:54:19

The screenshot shows the VS Code interface with the file `courses.component.ts` open. The code implements the `OnInit` interface and defines a search functionality. It uses the `.filter()` method to search through an array of courses based on a query parameter.

```
11 export class CoursesComponent implements OnInit{
12   coursesService = inject(CourseService);
13   AllCourses: Course[];
14   searchString: string;
15
16   activeRoute: ActivatedRoute = inject(ActivatedRoute);
17
18   ngOnInit(){
19     this.searchString = this.activeRoute.snapshot.queryParams['search'];
20     console.log(this.searchString);
21
22     if(this.searchString === undefined || this.searchString === ''){
23       this.AllCourses = this.coursesService.courses;
24     }else{
25       this.AllCourses = this.coursesService.courses.filter(x => x.title.includes(
26         this.searchString));
27     }
28   }
29 }
```

- Filter method used to search for courses with the term "JavaScript"
returned no results.

▷ 2:03:24

The screenshot shows the same code as above, but with a modification in the `ngOnInit` method. The `.filter()` method now converts both the search term and the course titles to lowercase before performing the comparison.

```
11 export class CoursesComponent implements OnInit{
12   coursesService = inject(CourseService);
13   AllCourses: Course[];
14   searchString: string;
15
16   activeRoute: ActivatedRoute = inject(ActivatedRoute);
17
18   ngOnInit(){
19     this.searchString = this.activeRoute.snapshot.queryParams['search'];
20     console.log(this.searchString);
21
22     if(this.searchString === undefined || this.searchString === ''){
23       this.AllCourses = this.coursesService.courses;
24     }else{
25       this.AllCourses = this.coursesService.courses
26         .filter(x => x.title.toLowerCase()
27           .includes(this.searchString.toLowerCase()));
28     }
29   }
30 }
```

- Search strings are converted to lowercase before comparison, allowing
for more accurate and visible results on the web page.

▷ 2:04:12

```
src/app/courses/courses.component.ts
11 component implements OnInit{
12 inject(CourseService);
13 ;
14 ;
15 ;
16 activatedRoute = inject(ActivatedRoute);
17 ;
18 ;
19 :ng = this.activeRoute.snapshot.queryParams['search'];
20 = this.activeRoute.snapshot.queryParamMap.get('search');
21 searchString;
22 ;
23 :ng === undefined || this.searchString === '' || this.searchString === null){
24 = this.coursesService.courses;@NonNullableAssert
25 @AngularCompiler
26 = this.coursesService.courses @NonNullableFormBuilder
27 .title.toLowerCase()
28 .searchString.toLowerCase());
29 ;
30 ;
31 ;
32 ;
```

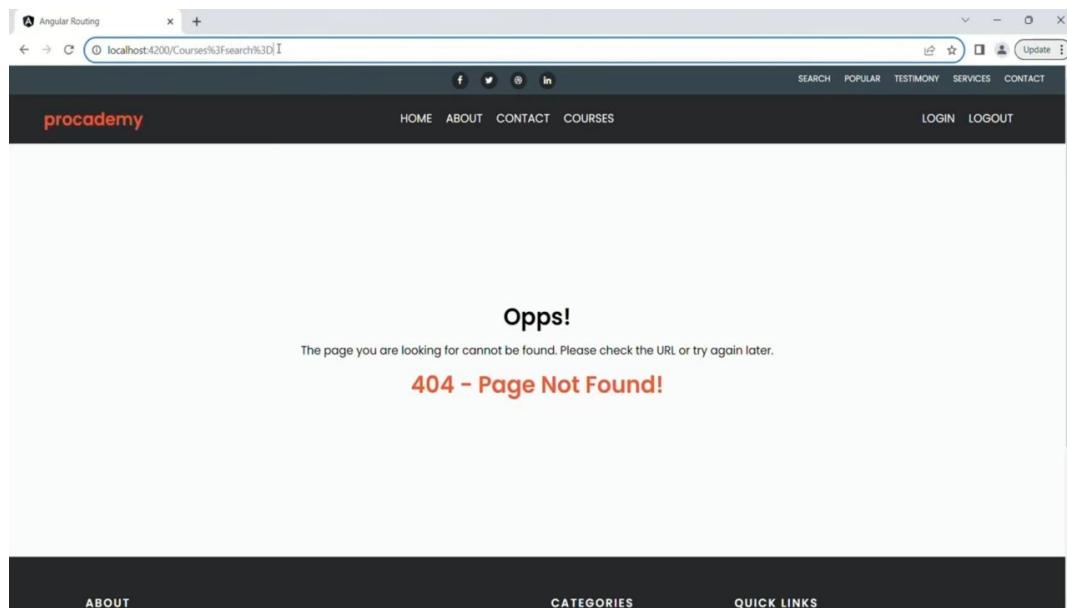
- 💡 New feature added to web page allows for improved course search functionality.

▷ 2:06:14

```
banner > banner.component.html > div.home-page-banner > div.home-page-banner-search > button.search-button
1 e-banner">
2 -page-banner-slogan">
3 ng in easy step by step way. Excel in your career.</h1>
4 ;
5 dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut
6 labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea comm
7 ure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
8 ;
9 ;
10 -page-banner-search">
11 ext" class="search-input" placeholder="Search Courses" #searchVar>
12 "search-button" [routerLink]="/Courses?search="+ {searchVar.value}>Search Courses
```

- 💡 Template reference variables in JavaScript can be used to directly access input element properties.

▷ 2:09:35



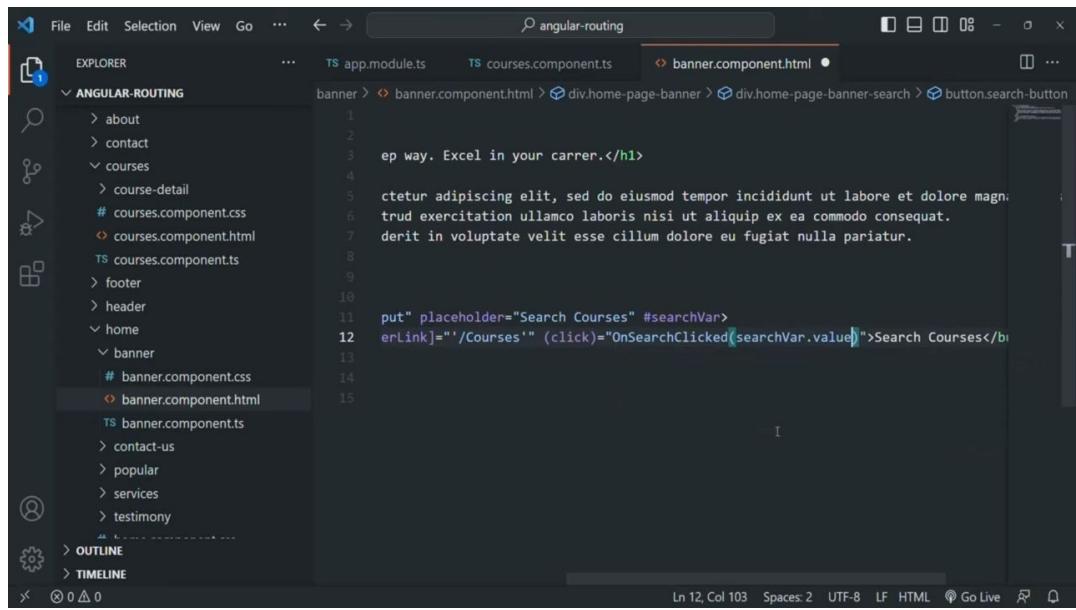
- Website search function not displaying correct results due to improper URL formatting, leading to 404 error.

▷ 2:10:17

```
1<div>
2  <input type="text" placeholder="Search Courses" #searchVar>
3  <a href="/Courses" [queryParams]="{search: searchVar.value}">Search Courses</a>
4</div>
```

- In order to specify query parameters, the query params directive is used to assign an object with the query string name and its value.

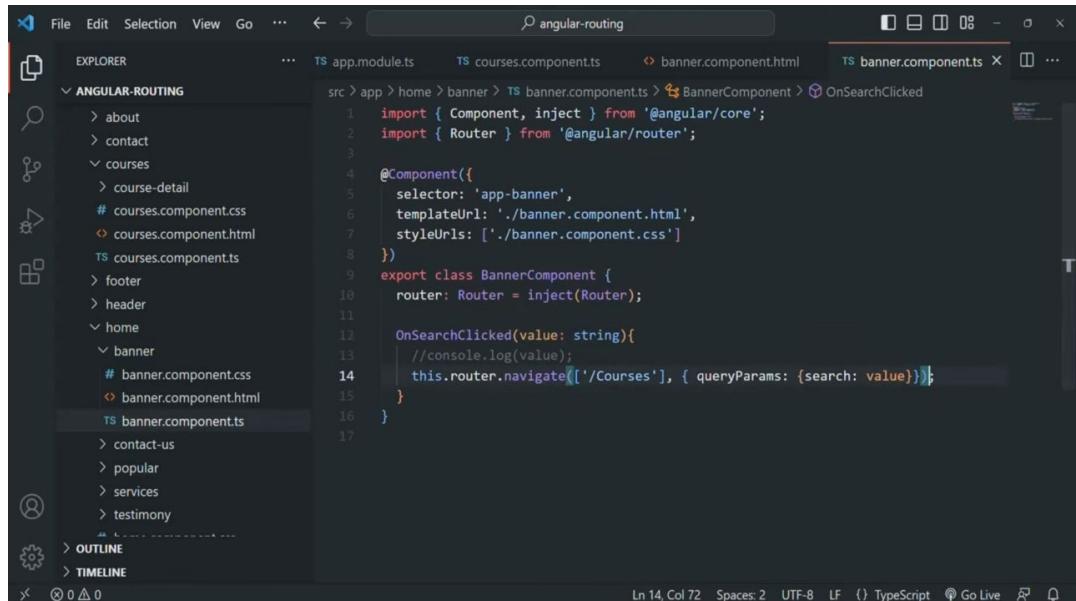
▷ 2:11:41



```
1 <h1>The best way. Excel in your career.</h1>
2
3 <p>ctetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
4 <p>Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
5
6
7
8
9
10
11 <input type="text" placeholder="Search Courses" #searchVar>
12 <a href="/Courses" (click)="OnSearchClicked(searchVar.value)">Search Courses</a>
13
14
15
```

- The Banner component in the search input element allows users to enter a value and trigger a search click method.

▶ 2:13:58



```
1 import { Component, inject } from '@angular/core';
2 import { Router } from '@angular/router';
3
4 @Component({
5   selector: 'app-banner',
6   templateUrl: './banner.component.html',
7   styleUrls: ['./banner.component.css']
8 })
9 export class BannerComponent {
10   router: Router = inject(Router);
11
12   OnSearchClicked(value: string){
13     //console.log(value);
14     this.router.navigate(['/Courses'], { queryParams: { search: value } });
15   }
16 }
```

- The ability to specify query strings in an object allows for efficient filtering of search results on web pages.

▶ 2:16:47

The screenshot shows the Visual Studio Code interface with the title bar "angular-routing". The Explorer sidebar on the left lists files under "ANGULAR-ROUTING", including "courses.component.html" which is currently selected. The main editor area displays the following code:

```
src > app > courses > courses.component.html > div.courses-page-main-container > a
14   </div>
15   <div class="course-title">
16     {{ course.title }}
17   </div>
18   <div class="course-desc">
19     {{ course.desc.substring(0, 120) }}
20   </div>
21   <div class="course-price-rating">
22     <div class="course-price"><b>PRICE:</b> {{ course.price | currency }}</div>
23     <div class="course-rating"><b>RATINGS:</b> {{ course.rating }}</div>
24   </div>
25   <div class="course-action-buttons">
26     <button class="btn">BUY NOW</button>
27     <button class="btn" [routerLink]="/Courses/Course/' + course.id">View Details</button>
28   </div>
29 </div>
30 </div>
31 </div>
32 <a routerLink="/Courses" [queryParams]="{search: 'JavaScript'}">Show JavaScript Courses</a>
33 </div>
34 </div>
```

At the bottom of the editor, status bar items include "Ln 32, Col 65", "Spaces: 4", "UTF-8", "LF", "HTML", "Go Live", and file icons.

💡 New feature on website allows users to easily search for JavaScript courses with a simple click.

▶ 2:19:16

The screenshot shows the Visual Studio Code interface with the title bar "angular-routing". The Explorer sidebar on the left lists files under "ANGULAR-ROUTING", including "courses.component.html" which is currently selected. The main editor area displays the same code as the previous screenshot, but with a cursor on the closing brace of the "queryParams" object in the anchor tag's routerLink attribute.

```
src > app > courses > courses.component.html > div.courses-page-main-container > a
14   </div>
15   <div class="course-title">
16     {{ course.title }}
17   </div>
18   <div class="course-desc">
19     {{ course.desc.substring(0, 120) }}
20   </div>
21   <div class="course-price-rating">
22     <div class="course-price"><b>PRICE:</b> {{ course.price | currency }}</div>
23     <div class="course-rating"><b>RATINGS:</b> {{ course.rating }}</div>
24   </div>
25   <div class="course-action-buttons">
26     <button class="btn">BUY NOW</button>
27     <button class="btn" [routerLink]="/Courses/Course/' + course.id">View Details</button>
28   </div>
29 </div>
30 </div>
31 </div>
32 <a routerLink="/Courses" [queryParams]="{search: 'JavaScript'}">Show JavaScript Courses</a>
33 </div>
34 </div>
```

At the bottom of the editor, status bar items include "Ln 32, Col 70", "Spaces: 4", "UTF-8", "LF", "HTML", "Go Live", and file icons.

▶ 2:19:23

The screenshot shows a web browser window with the URL <localhost:4200/Courses?search=JavaScript>. The page displays a course card for 'A Complete React Developer Course'. The card features the React logo, the title 'A Complete React Developer Course', a short description, a price of '\$599.00', a rating of '4.8', and two buttons: 'BUY NOW' and 'DETAILS'. Below the card is a link 'Show JavaScript Courses'.

- JavaScript course does not display as expected due to the logic being written inside the NG on init lifecycle hook.

▷ 2:20:07

The screenshot shows the VS Code editor with the file `courses.component.ts` open. The code implements the `OnInit` interface and uses the `ActivatedRoute.queryParamMap.subscribe()` observable to handle search queries. The code is as follows:

```
export class CoursesComponent implements OnInit {
  coursesService = inject(CourseService);
  AllCourses: Course[];
  searchString: string;

  activeRoute: ActivatedRoute = inject(ActivatedRoute);

  ngOnInit(){
    // this.searchString = this.activeRoute.snapshot.queryParams['search'];
    // this.searchString = this.activeRoute.snapshot.queryParamMap.get('search');
    // console.log(this.searchString);

    this.activeRoute.queryParamMap.subscribe((data) => {
      this.searchString = data.get('search');

      if(this.searchString === undefined || this.searchString === '' || this.searchString === null){
        this.AllCourses = this.coursesService.courses;
      }else{
        this.AllCourses = this.coursesService.courses
          .filter(x => x.title.toLowerCase()
            .includes(this.searchString.toLowerCase())));
      }
    })
  }
}
```

- Using the query param map observable, users can retrieve specific values from a map using the get method.

▷ 2:24:18

Route Parameter vs Query String

Route parameters are required and it is used by Angular router to determine the route. Route Parameters are part of route definition.

Query strings are optional data which we pass through route. If the query parameter is missing in the URL, then it will not stop angular from navigating to the route.

- Route parameters are required for displaying specific results, while query parameters are optional data passed through a route.

▷ 2:26:59

Using Fragments in Route

- The fragment in a route is a link that jumps to a specific section or content within an HTML page.

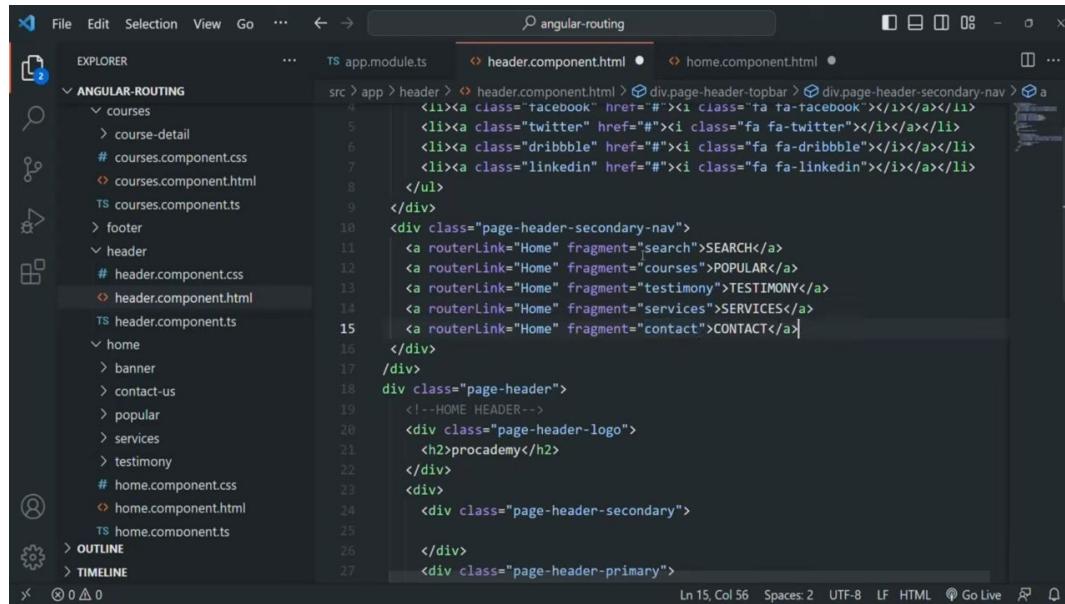
▷ 2:27:16

What is Query String

A fragment in a route is a link which jumps to a section or a content in the HTML page, which contains the ID mentioned in the fragment. A fragment comes after a # sign

localhost:4200/Home#Services

▶ 2:27:27

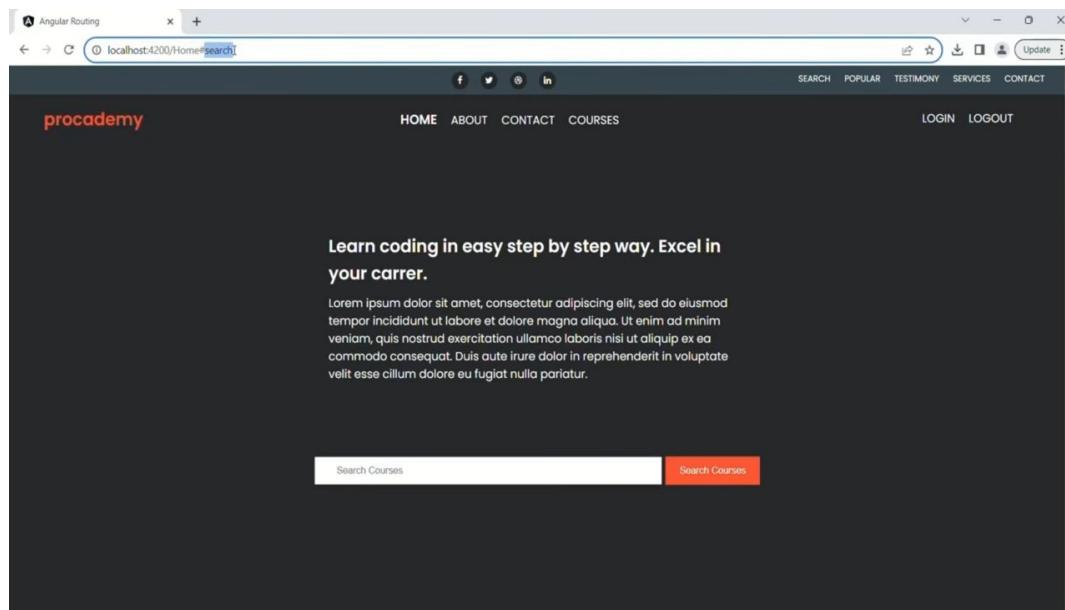


The screenshot shows a code editor with the file 'header.component.html' open. The code is part of an Angular application and includes routing definitions for various sections like Home, Popular, Services, Testimony, and Contact. It also includes social media links for Facebook, Twitter, Dribbble, and LinkedIn.

```
src > app > header > header.component.html > div.page-header-topbar > div.page-header-secondary-nav > a
  4   <li><a class="facebook" href="#"><i class="fa fa-facebook"></i></a></li>
  5   <li><a class="twitter" href="#"><i class="fa fa-twitter"></i></a></li>
  6   <li><a class="dribbble" href="#"><i class="fa fa-dribbble"></i></a></li>
  7   <li><a class="linkedin" href="#"><i class="fa fa-linkedin"></i></a></li>
  8 </ul>
  9 </div>
10 <div class="page-header-secondary-nav">
11   <a routerLink="Home" fragment="search">SEARCH</a>
12   <a routerLink="Home" fragments="courses">POPULAR</a>
13   <a routerLink="Home" fragments="testimony">TESTIMONY</a>
14   <a routerLink="Home" fragments="services">SERVICES</a>
15   <a routerLink="Home" fragment="contact">CONTACT</a>
16 </div>
17 </div>
18 <div class="page-header">
19   <!-- HOME HEADER -->
20   <div class="page-header-logo">
21     <h2>procademy</h2>
22   </div>
23   <div>
24     <div class="page-header-secondary">
25       </div>
26     <div class="page-header-primary">
27       </div>
```

- ✿ Developing a web page with specified fragments for different sections is the first step in creating a seamless user experience.

▶ 2:32:58



★ Web page navigation using fragment values is demonstrated by clicking on different links and observing the changes in the URL.

▷ 2:33:17

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows the project structure with files like 'header.component.html', 'header.component.ts', 'home', 'home.component.css', 'home.component.html', and 'home.component.ts'. The main editor area displays the 'home.component.ts' file, which contains TypeScript code for an 'HomeComponent'. It includes imports for 'Component', 'OnInit', 'ActivatedRoute', and 'Router'. The component selector is 'app-home', and it has a template URL pointing to 'home.component.html'. The 'ngOnInit' method uses the 'activeRoute.fragment.subscribe()' callback to log data and scroll to sections. The status bar at the bottom shows 'Ln 15, Col 38' and other file-related information.

★ New method "jump to section" implemented for smoother navigation on web pages.

▷ 2:40:32

What is Query String

A fragment in a route is a link which jumps to a section or a content in the HTML page, which contains the ID mentioned in the fragment. A fragment comes after a # sign

`localhost:4200/Home#Services`



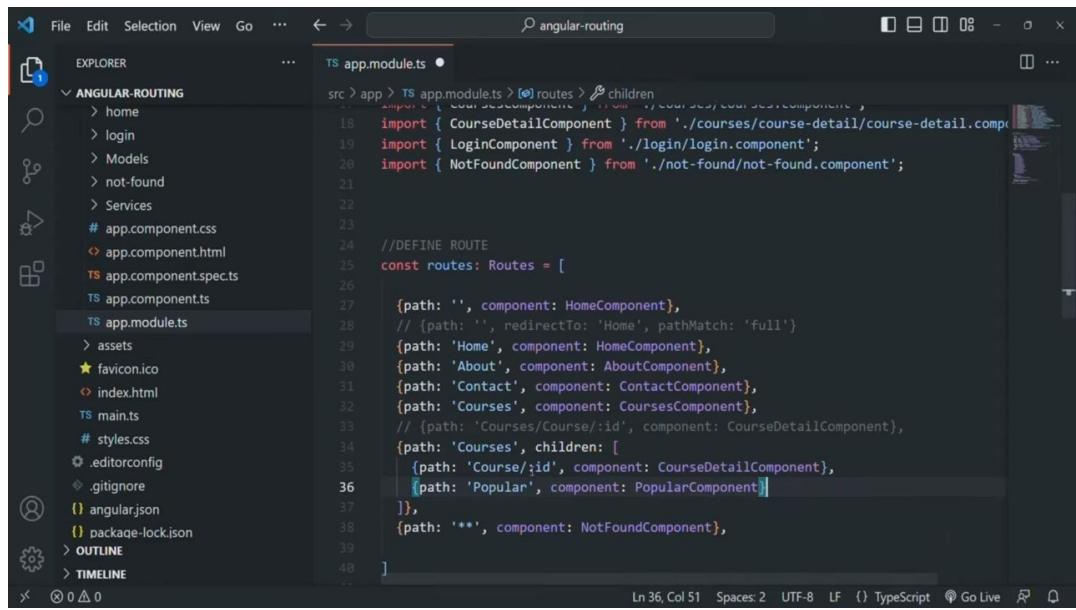
- Utilizing fragments in HTML pages allows for easy navigation to specific sections of content.

▷ 2:41:29

Working with Child Routes

- In this lecture, we will learn how to create and use a child route in Angular.

▷ 2:41:50



The screenshot shows the VS Code interface with the title bar "angular-routing". The left sidebar shows a tree view of the project structure under "ANGULAR-ROUTING", including files like "app.component.css", "app.component.html", "app.component.spec.ts", "app.module.ts", "main.ts", and "styles.css". The main editor area displays the "app.module.ts" file with the following TypeScript code:

```
src > app > TS app.module.ts > routes > children
18 import { CourseDetailComponent } from './courses/course-detail/course-detail.component';
19 import { LoginComponent } from './login/login.component';
20 import { NotFoundComponent } from './not-found/not-found.component';
21
22
23 //DEFINE ROUTE
24 const routes: Routes = [
25   {path: '', component: HomeComponent},
26   // [path: '', redirectTo: 'Home', pathMatch: 'full']
27   {path: 'Home', component: HomeComponent},
28   {path: 'About', component: AboutComponent},
29   {path: 'Contact', component: ContactComponent},
30   {path: 'Courses', component: CoursesComponent},
31   // {path: 'Courses/Course/:id', component: CourseDetailComponent},
32   {path: 'Courses', children: [
33     {path: 'Course/:id', component: CourseDetailComponent},
34     {path: 'Popular', component: PopularComponent}
35   ]},
36   {path: '**', component: NotFoundComponent},
37
38 ]
39
40 ]
```

At the bottom of the editor, status bar items include "Ln 36, Col 51", "Spaces: 2", "UTF-8", "LF", "TypeScript", "Go Live", and file icons.

✿ New Popular component added with two child routes for enhanced user experience.

▷ 2:46:43

Creating a Route Module File

✿ In the first lecture of this section, we learned how to define child routes in an Angular application.

▷ 2:47:27

The screenshot shows the VS Code interface with the title bar "angular-routing". The Explorer sidebar on the left shows files like app.module.ts, routing.module.ts, and app.component.ts. The main editor area displays the routing module code:

```
src > app > TS routing.module.ts > RoutingModule
14     {path: '', component: HomeComponent},
15     {path: 'Home', component: HomeComponent},
16     {path: 'About', component: AboutComponent},
17     {path: 'Contact', component: ContactComponent},
18     {path: 'Courses', component: CoursesComponent},
19     {path: 'Courses', children: [
20         {path: 'Course/:id', component: CourseDetailComponent},
21         {path: 'Popular', component: PopularComponent}
22     ]},
23     {path: '**', component: NotFoundComponent},
24 ]
25
26 @NgModule({
27     imports: [
28         RouterModule.forRoot(routes)
29     ],
30     exports: [RouterModule]
31 })
32 export class RoutingModule{}
```

At the bottom, status bar items include "Ln 32, Col 28", "Spaces: 4", "UTF-8", "CRLF", "TypeScript", "Go Live", and file icons.

Exporting and importing router modules allows for seamless integration of routing functionality across different files and modules in a web application.

▶ 2:54:40

The screenshot shows the VS Code interface with the title bar "angular-routing". The Explorer sidebar on the left shows files like app.module.ts, routing.module.ts, and app.component.ts. The main editor area displays the AppModule code:

```
src > app > TS app.module.ts > AppModule
31     FooterComponent,
32     HeaderComponent,
33     ContactComponent,
34     AboutComponent,
35     BannerComponent,
36     ServicesComponent,
37     TestimonyComponent,
38     ContactUSComponent,
39     PopularComponent,
40     CoursesComponent,
41     CourseDetailComponent,
42     LoginComponent,
43     NotFoundComponent
44 ],
45     imports: [
46         BrowserModule,
47         NgModule,
48         RouterModule,
49         providers: [ServicesService, CourseService],
50         bootstrap: [AppComponent]
51     }
52     export class AppModule { }
```

At the bottom, status bar items include "Ln 47, Col 18", "Spaces: 2", "UTF-8", "LF", "TypeScript", "Go Live", and file icons.

▶ 2:54:52

What is a Route Guard

- Route protection is essential for controlling access to certain parts of an application.

▷ 2:57:11

Why Route Guard

Allowing the users to navigate to all parts of the application is not a good idea. And we need to restrict the user from accessing certain routes, until the user performs a specific action like login to application. So, you can use route guards for following scenario's

- Restrict a user from accessing a route.
- Ask user to save changes before moving away from view.
- Validating the route parameters before navigating to the route.
- Fetch some data before you display component view of a route.

- Angular route guards are a crucial tool for securing and controlling access to routes in web applications.

▷ 2:58:15

Type of Route Guard

CanActivate

This guard decides if a route can be accessed by a user or not. This guard is useful in the circumstance where the user is not authorized to navigate to the target component.

CanActivateChild

This guard decides, if a user can leave a route or not. This guard is useful in case where the user might have some pending changes, which was not saved.

CanDeactivate

This guard determines whether a child route can be activated or not.



- Angular introduces five route guards, including the connectivate and can activate child route guard, to control user access to routes.

▷ 2:58:56

Type of Route Guard

Resolve

This guard delays the activation of the route until some tasks are complete. You can use the guard to pre-fetch the data from the backend API, before activating the route

CanLoad

The **CanLoad** Guard ^b prevents the loading of the Lazy Loaded Module. We generally use this guard when we do not want an unauthorized user to be able to even see the source code of the module.

- New route guards in navigation systems allow for better control over user navigation and data prefetching.

▷ 3:00:42

Route Guard Implementation

Angular 14 & Lower Versions

1. Create a service which inherits from specific route guard interface.
2. Implement the method provided by that route guard interface.
3. Return a boolean value / data from that method.
4. Assign the service to route guard property of route object.

Angular 15 & Higher Versions

1. Create a function which should return a boolean value / data.
2. Assign that function to specific route guard property of route object.

Angular has introduced a new way of implementing route guards in versions 15 and higher.

▶ 3:02:08

Creating an Auth Service

In this lecture, we will explore the implementation of route guards in Angular 14 and lower versions as well as in Angular 15 and higher versions.

▶ 3:04:47

The screenshot shows the VS Code interface with the file `TS user.service.ts` open. The code defines a `UserService` class with an array of `User` objects. The `User` class has properties `id`, `name`, `username`, and `password`. A constructor initializes these properties.

```
import { Injectable } from '@angular/core';
import { User } from '../Models/user';

@Injectable({
  providedIn: 'root'
})
export class UserService{
  users: User[] = [
    new User(1, 'John Smith', 'js', '12345'),
    new User(2, 'Merry Jane', 'mj', '12345'),
    new User(3, 'Mark Vought', 'mv', '12345'),
    new User(4, 'Sarah King', 'sk', '12345')
  ]
}
```

A new user model has been implemented with the ability to create a username and password.

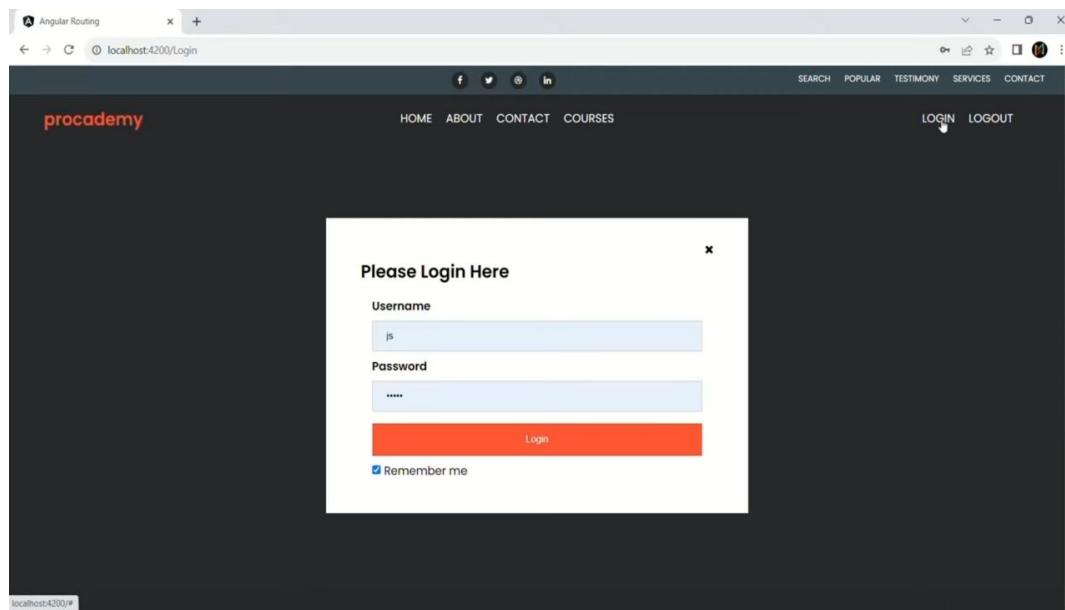
▶ 3:06:08

The screenshot shows the VS Code interface with the file `TS user.ts` open. The code defines a `User` class with properties `id`, `name`, `username`, and `password`. A constructor initializes these properties.

```
export class User{
  id: number;
  name: string;
  username: string;
  password: string;

  constructor(id: number, name: string, username: string, password: string){
    this.id = id;
    this.name = name;
    this.username = username;
    this.password = password;
  }
}
```

▶ 3:06:11



✿ Developing a login page for a website involves integrating the login component with the header component.

▷ 3:14:41

```
src > app > login > login.component.html
1  -- <div class="overlay"></div> -->
2  <div class="login-form-container">
3    <div class="login-form">
4      <div class="cancel-button-div">
5        <i class="fa fa-times" aria-hidden="true"></i>
6      </div>
7      <h2>Please Login Here</h2>
8      <div class="container">
9        <label for="uname"><b>Username</b></label>
10       <input type="text" placeholder="Enter Username" required #username>
11
12       <label for="psw"><b>Password</b></label>
13       <input type="password" placeholder="Enter Password" required #password>
14
15       <button type="submit" (click)="OnLoginClicked()">Login</button>
16     <label>
17       <input type="checkbox" checked="checked" name="remember"> Remember me
18     </label>
19   </div>
20 </div>
21 </div>
```

✿ A new method for handling login button clicks has been added to the login component.

▷ 3:18:27

```
<h2>proacademy</h2>
</div>
<div class="page-header-secondary">
</div>
<div class="page-header-primary">
    <a routerLink="Home" [routerLinkActive]="'active'" [routerLinkActiveOp:>
        <a [routerLink]="About" [routerLinkActive]="'active'" [routerLinkAct:>
            <a routerLink="Contact" [routerLinkActive]="'active'" [routerLinkAct:>
                <a routerLink="Courses" [routerLinkActive]="'active'" [routerLinkAct:>
</div>
<div class="page-header-login-logout">
    <a routerLink="/Login">LOGIN</a>
    <a routerLink="/Logout" [queryParam]={"logout: true}">LOGOUT</a>
</div>
</div>
```

- ➡ Developers are working to assign an object and specify a log out query parameter for a web page.

▷ 3:26:18

```
import { Injectable, inject } from '@angular/core';
import { UserService } from './user.service';

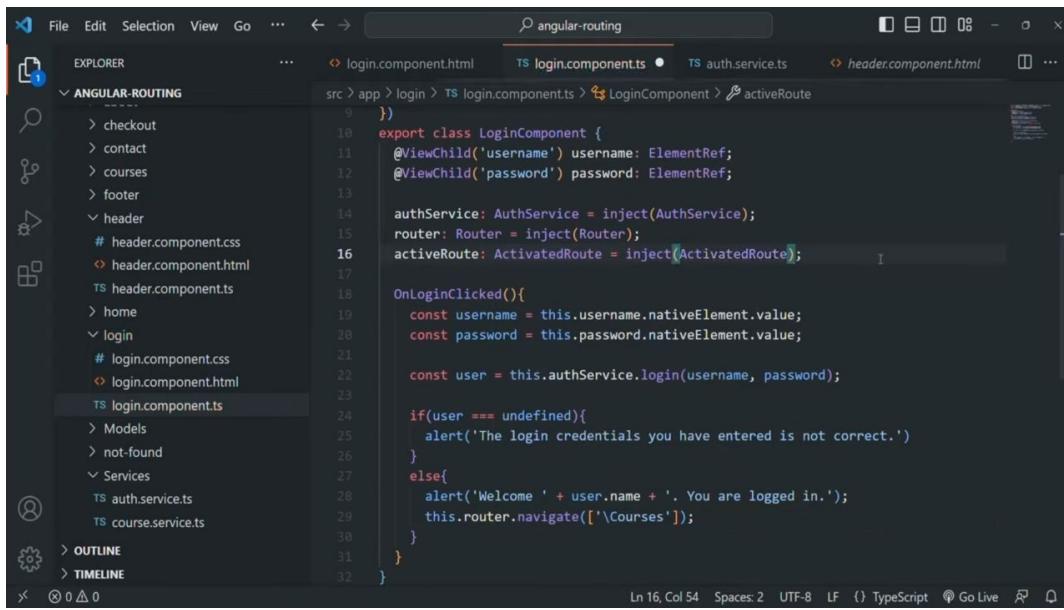
@Injectable({
    providedIn: 'root'
})
export class AuthService{
    isLoggedIn: Boolean = false;
    userService: UserService = inject(UserService);

    login(username: string, password: string){
        let user = this.userService.users.find((u) => u.username === username && u.password === password);
        if(user === undefined)
            this.isLoggedIn = false;
        else
            this.isLoggedIn = true;
        return user;
    }

    logout(){
        this.isLoggedIn = false;
    }
}
```

- ➡ The logout method of the auth service will be called in the login component to read the query parameter and perform the necessary logic.

▷ 3:27:34

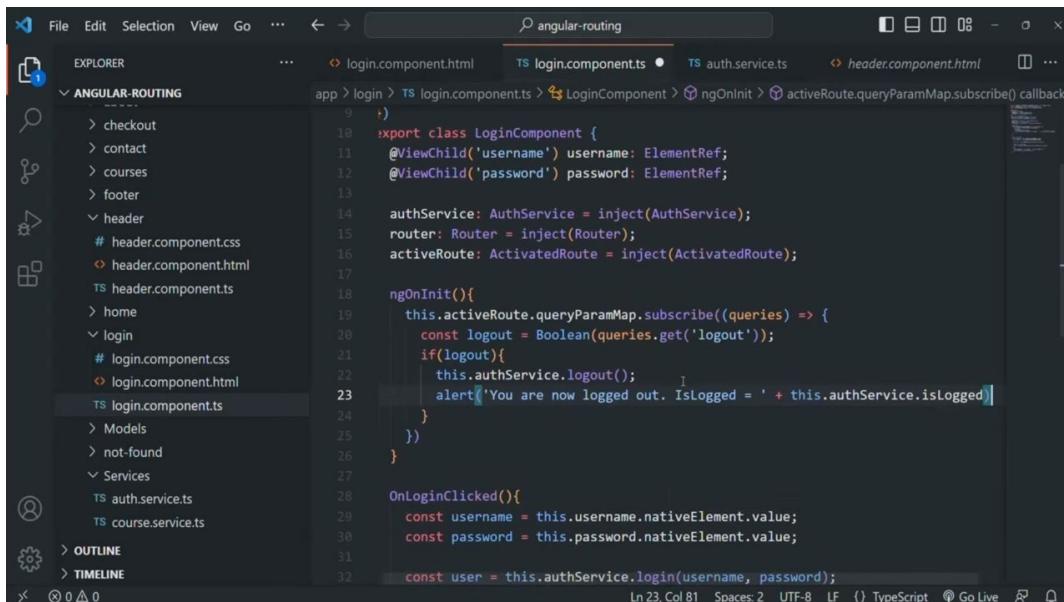


```
src > app > login > login.component.ts > LoginComponent > activeRoute
9 }
10 export class LoginComponent {
11   @ViewChild('username') username: ElementRef;
12   @ViewChild('password') password: ElementRef;
13
14   authService: AuthService = inject(AuthService);
15   router: Router = inject(Router);
16   activeRoute: ActivatedRoute = inject(ActivatedRoute);
17
18   OnLoginClicked() {
19     const username = this.username.nativeElement.value;
20     const password = this.password.nativeElement.value;
21
22     const user = this.authService.login(username, password);
23
24     if(user === undefined) {
25       alert('The login credentials you have entered is not correct.')
26     } else {
27       alert('Welcome ' + user.name + '. You are logged in.');
28       this.router.navigate(['Courses']);
29     }
30   }
31 }
32 }
```

Ln 16, Col 54 Spaces: 2 UTF-8 LF () TypeScript ⓘ Go Live ⌂ ⌂

- Angular developers are implementing the NG on init lifecycle hook to inject an instance of the activated route class inside the login component.

▶ 3:28:37



```
app > login > login.component.ts > LoginComponent > ngOnInit > activeRoute.queryParamMap.subscribe() callback
9 }
10 export class LoginComponent {
11   @ViewChild('username') username: ElementRef;
12   @ViewChild('password') password: ElementRef;
13
14   authService: AuthService = inject(AuthService);
15   router: Router = inject(Router);
16   activeRoute: ActivatedRoute = inject(ActivatedRoute);
17
18   ngOnInit() {
19     this.activeRoute.queryParamMap.subscribe((queries) => {
20       const logout = Boolean(queries.get('logout'));
21
22       if(logout){
23         this.authService.logout();
24         alert('You are now logged out. IsLogged = ' + this.authService.isLoggedIn);
25       }
26     })
27   }
28
29   OnLoginClicked() {
30     const username = this.username.nativeElement.value;
31     const password = this.password.nativeElement.value;
32     const user = this.authService.login(username, password);
33   }
34 }
```

Ln 23, Col 81 Spaces: 2 UTF-8 LF () TypeScript ⓘ Go Live ⌂ ⌂

- The authentication service's log property is being tested to ensure it is set to false when the user logs out.

▶ 3:31:03

CanActivate Route Guard

- ★ In this article, we will explore how to implement route guards in Angular and understand the purpose of the can activate route guard.

▷ 3:32:05

CanActivate Route Guard

The Angular CanActivate route guard decides if a route can be activated or not. We use this route guard when we want to check on some condition, before showing the component view to user.

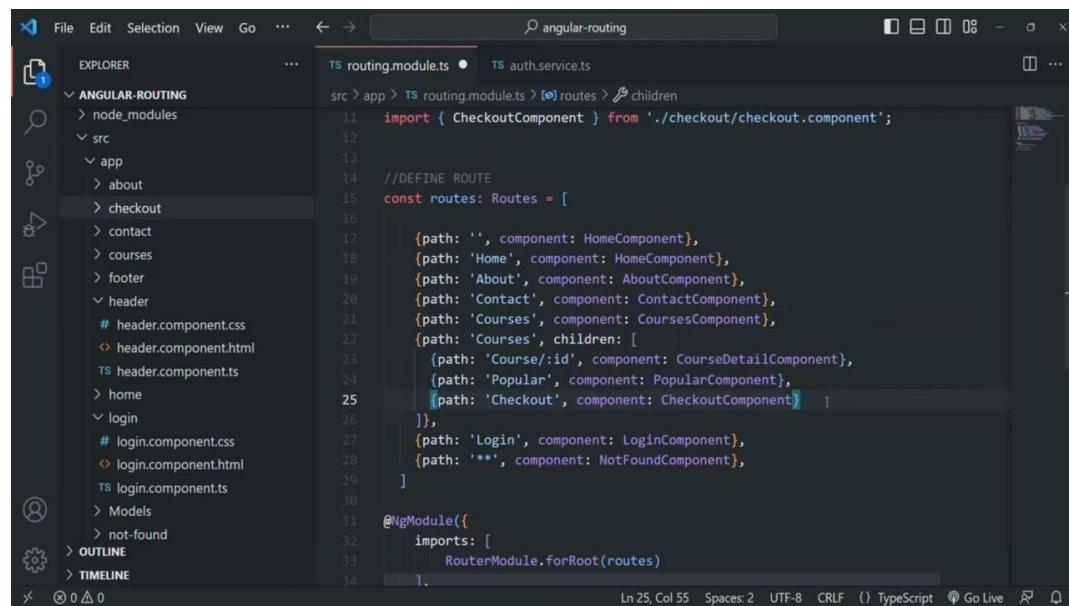
- ★ Learn how to implement route cards in Angular and understand the purpose of each route guard, starting with the can activate route guard.

▷ 3:32:27

CanActivate Route Guard Use Case

The common use case where we can use a CanActivate route guard is when we want to protect a route from an unauthorized user. For example, let's say there are some routes which can only be accessed by a logged in user, we can use CanActivate guard.

▶ 3:32:38



```
File Edit Selection View Go ... < > angular-routing
EXPLORER TS routing.module.ts ● TS auth.service.ts
src > app > TS routing.module.ts > [o] routes > ↴ children
11 import { CheckoutComponent } from './checkout/checkout.component';
12
13
14 //DEFINE ROUTE
15 const routes: Routes = [
16
17   {path: '', component: HomeComponent},
18   {path: 'Home', component: HomeComponent},
19   {path: 'About', component: AboutComponent},
20   {path: 'Contact', component: ContactComponent},
21   {path: 'Courses', component: CoursesComponent},
22   {path: 'Courses', children: [
23     {path: 'Course/:id', component: CourseDetailComponent},
24     {path: 'Popular', component: PopularComponent},
25     {path: 'Checkout', component: CheckoutComponent}
26   ]},
27   {path: 'Login', component: LoginComponent},
28   {path: '**', component: NotFoundComponent},
29 ]
30
31 @NgModule({
32   imports: [
33     RouterModule.forRoot(routes)
34   ],
35 })
36 
```

- ★ New route and component properties have been specified for the checkout process in the courses component.

▶ 3:35:15

```

<div>
  <div>
    <div>
      <div>{{ course.title }}</div>
      <div>{{ course.desc.substring(0, 120) }}</div>
      <div>
        <div>{{ course.price | currency }}</div>
        <div>{{ course.rating }}</div>
      </div>
      <div>
        <button class="btn" routerLink="/Courses/Checkout">Buy Now</button>
        <button class="btn" [routerLink]="/Courses/Course/+ course.id">View Details</button>
      </div>
    </div>
  </div>
</div>

```

Line 26 highlights the button element with the value "Buy Now".

💡 New feature added to website allows for dynamic redirection to checkout page based on selected course.

▷ 3:35:48

Payment Information		Complete Modern JavaScript Course	
Full Name	Satoshi Nakamoto	449.50 x 1 Course	449.00 USD
Card Number	4111-2222-3333-4444	Discount	0.00 USD
Expires on:		Subtotal	449.00 USD
Month	Month: ▾	Tax	47.41 USD
CVV	415	Total	546.41 USD

A large orange button at the bottom left says "Buy Now".

▷ 3:35:59

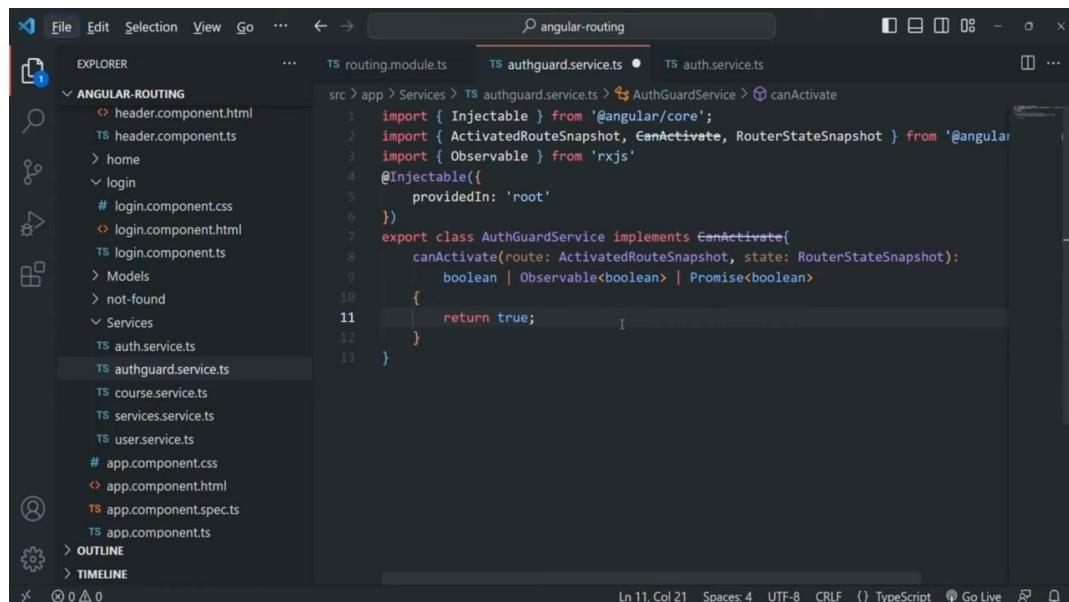
CanActivate Guard: Angular 14 & Lower versions

1. Create a service which inherits from CanActivate interface.
2. Implement CanActivate method provided by CanActivate interface.
3. Return a boolean value from the CanActivate method based on if the route can be activated or not.
4. Assign the service to canActivate property of route object.

↳

- 💡 New Angular 15 introduces a new way to implement route guards, offering flexibility for developers.

▷ 3:37:21



```
src > app > Services > TS authguard.service.ts > AuthGuardService > canActivate
1 import { Injectable } from '@angular/core';
2 import { ActivatedRouteSnapshot, CanActivate, RouterStateSnapshot } from '@angular/router';
3 import { Observable } from 'rxjs';
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class AuthGuardService implements CanActivate{
8   canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
9     boolean | Observable<boolean> | Promise<boolean>
10  {
11    return true;
12  }
13 }
```

- 💡 New service implementation successfully resolves connectivity issues and creates a route activation card.

▷ 3:42:51

CanActivate Guard: Angular 14 & Lower versions

1. Create a service which inherits from CanActivate interface.
2. Implement CanActivate method provided by CanActivate interface.
3. Return a boolean value from the CanActivate method based on if the route can be activated or not.
4. Assign the service to canActivate property of route object.

▷ 3:42:55

```
File Edit Selection View Go ... < > angular-routing
EXPLORER File Edit Selection View Go ... < > angular-routing
ANGULAR-ROUTING
src > app > Services > auth.service.ts > AuthService > isAuthenticated
isLoggedIn: Boolean = false;
userService: UserService = inject(UserService);

login(username: string, password: string){
    let user = this.userService.users.find((u) => u.username === username
                                                && u.password === password);
    if(user === undefined)
        this.isLoggedIn = false;
    else
        this.isLoggedIn = true;
    return user;
}

logout(){
    this.isLoggedIn = false;
}

isAuthenticated(){
    return this.isLoggedIn;
}
```

- ★ Service dot is authenticated method returns a Boolean value based on the logged status.

▷ 3:47:56

The screenshot shows a code editor with the Angular Routing module selected in the Explorer sidebar. The current file is authguard.service.ts, which contains the following TypeScript code:

```
import { Injectable, inject } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from './auth.service';
@Injectable({
  providedIn: 'root'
})
export class AuthGuardService implements CanActivate{
  authService: AuthService = inject(AuthService);
  router: Router = inject(Router);

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean>
  {
    if(this.authService.isAuthenticated()){
      return true;
    }else{
      this.router.navigate(['/Login']);
      return false;
    }
  }
}
```

The code defines a service that implements the `CanActivate` interface. It checks if the user is authenticated. If yes, it returns `true`. If no, it navigates to the `/Login` page and returns `false`.

- ➡ Developers are implementing a new navigation feature to direct users to the login page on a web application.

▷ 3:49:23

CanActivateFn Route Guard

- ➡ Learn how to implement can activate route guard in Angular 15 and higher versions in the next lecture.

▷ 3:51:24

CanActivate Route Guard

The Angular CanActivate route guard decides if a route can be activated or not. We use this route guard when we want to check on some condition, before showing the component view to user.

▶ 3:51:30

CanActivate Route Guard Use Case

The common use case where we can use a CanActivate route guard is when we want to protect a route from an unauthorized user. For example, let's say there are some routes which can only be accessed by a logged in user, we can use CanActivate guard.

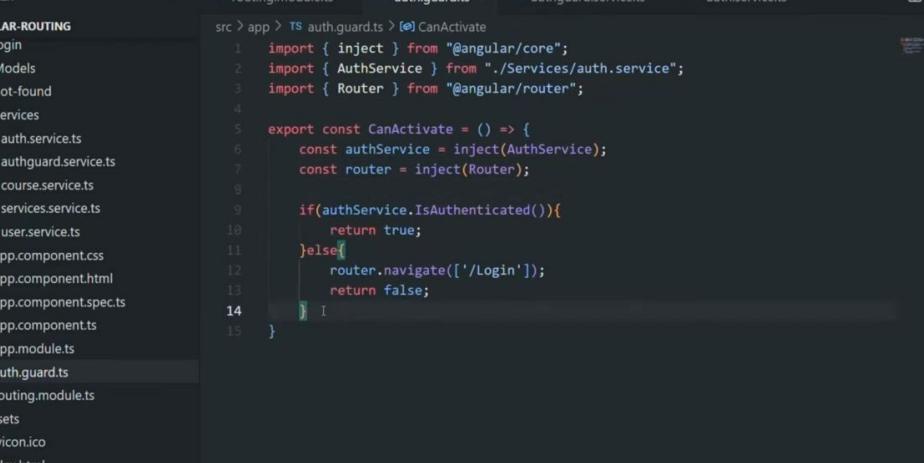
- ★ New Implementation Method for Connectivate Cloud in Angular 15 and Higher Versions.

▶ 3:51:57

The screenshot shows the Angular API List page for the router module. The left sidebar contains navigation links for Tutorials, Updates and releases, Reference (with sub-links for Conceptual reference, CLI Command Reference, and API reference), Error reference, Extended diagnostic reference, Example applications, Angular glossary, Angular coding style, Documentation, and contributors guide. The main content area has a search bar with 'canAct' entered and a 'GO TO SURVEY' button. Below the search bar, there are filters for Type (All) and Status (All). The router section is expanded, showing five API entries: CanActivate (selected), CanActivateChild, CanActivateFn, mapToCanActivate, and mapToCanActivateChild. Each entry has a small icon and a link to its documentation.

- Angular has deprecated the connective function in favor of the new can activate function in version 15 and higher.

▶ 3:53:05



```
src > app > TS auth.guard.ts > [o] CanActivate
1 import { inject } from "@angular/core";
2 import { AuthService } from "./Services/auth.service";
3 import { Router } from "@angular/router";
4
5 export const CanActivate = () => {
6   const authService = inject(AuthService);
7   const router = inject(Router);
8
9   if(authService.IsAuthenticated()){
10     return true;
11   }else{
12     router.navigate(['/Login']);
13     return false;
14   }
15 }
```

[†] Accessing router variables and importing functions for routing modules

▶ 3:59:00

The screenshot shows a code editor with the file `TS routing.module.ts` open. The code defines a route named 'NE ROUTE' with a single child route. This child route has several path segments and components defined. A specific line of code is highlighted: `canActivate: [CanActivate]`. The code editor interface includes a navigation bar with File, Edit, Selection, View, Go, etc., and a status bar at the bottom.

```
src > app > TS routing.module.ts [8] routes > children
  8 : { CourseDetailComponent } from './courses/course-detail/course-detail.component';
  9 : { NotFoundComponent } from './not-found/not-found.component';
10 : { LoginComponent } from './login/login.component';
11 : { CheckoutComponent } from './checkout/checkout.component';
12 : { AuthGuardService } from './Services/authguard.service';
13 : { CanActivate } from './auth.guard'
14 :
15 :
16 NE ROUTE
17 routes: Routes = [
18   {
19     path: '',
20     component: HomeComponent,
21     path: 'Home', component: HomeComponent,
22     path: 'About', component: AboutComponent,
23     path: 'Contact', component: ContactComponent,
24     path: 'Courses', component: CoursesComponent,
25     path: 'Courses', children: [
26       {path: 'Course/:id', component: CourseDetailComponent},
27       {path: 'Popular', component: PopularComponent},
28       {path: 'Checkout', component: CheckoutComponent, canActivate: [CanActivate]} ],
29     path: 'Login', component: LoginComponent},
30     path: '**', component: NotFoundComponent},
```

💡 New update allows for seamless activation of connectivity function in application, ensuring continued smooth operation.

▷ 3:59:50

CanActivateChild Route Guard

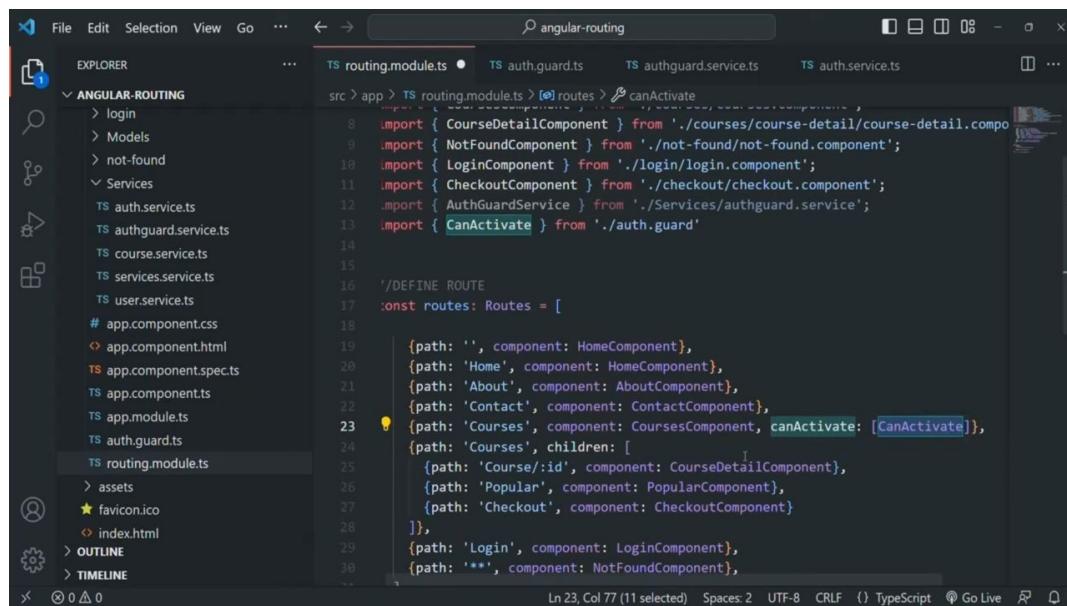
💡 Learn about the Connectivate Child Route Guard and its implementation in Angular.

▷ 4:02:13

CanActivateChild Route Guard

The Angular CanActivateChild route guard runs before we navigate to a child route guard. This guard is very similar to CanActivate route guard and it protects a child route from unauthorized access.

▶ 4:02:22



```
File Edit Selection View Go ... < > angular-routing
EXPLORER TS routing.module.ts TS auth.guard.ts TS authguard.service.ts TS auth.service.ts ...
ANGULAR-ROUTING
> login
> Models
> not-found
Services
TS auth.service.ts
TS authguard.service.ts
TS course.service.ts
TS services.service.ts
TS user.service.ts
# app.component.css
○ app.component.html
TS app.component.spec.ts
TS app.component.ts
TS app.module.ts
TS auth.guard.ts
TS routing.module.ts
> assets
★ favicon.ico
○ index.html
> OUTLINE
> TIMELINE
Ln 23, Col 77 (11 selected) Spaces: 2 UTF-8 CRLF () TypeScript ⚡ Go Live ⌂ ⌂
```

```
src > app > TS routing.module.ts > [e] routes > canActivate
    8 import { CourseDetailComponent } from './courses/course-detail/course-detail.component';
    9 import { NotFoundComponent } from './not-found/not-found.component';
   10 import { LoginComponent } from './login/login.component';
   11 import { CheckoutComponent } from './checkout/checkout.component';
   12 import { AuthGuardService } from './Services/authguard.service';
   13 import { CanActivate } from './auth.guard'
   14
   15 '/DEFINE ROUTE
   16 const routes: Routes = [
   17
   18     {path: '', component: HomeComponent},
   19     {path: 'Home', component: HomeComponent},
   20     {path: 'About', component: AboutComponent},
   21     {path: 'Contact', component: ContactComponent},
   22     {path: 'Courses', component: CoursesComponent, canActivate: [CanActivate]},
   23     {path: 'Courses', children: [
   24         {path: 'Course/:id', component: CourseDetailComponent},
   25         {path: 'Popular', component: PopularComponent},
   26         {path: 'Checkout', component: CheckoutComponent}
   27     ]},
   28     {path: 'Login', component: LoginComponent},
   29     {path: '**', component: NotFoundComponent},
   30 ];
```

- Connectivate can be used to protect both parent and child routes on a course route.

▶ 4:03:18

The screenshot shows the VS Code interface with the file 'TS routing.module.ts' open. The code defines routes for various components like HomeComponent, AboutComponent, ContactComponent, CoursesComponent, CourseDetailComponent, PopularComponent, CheckoutComponent, LoginComponent, and NotFoundComponent. It also imports canActivate from auth.guard and canActivate from auth.guard.ts.

```
import { CourseDetailComponent } from './courses/course-detail/course-detail.component';
import { NotFoundComponent } from './not-found/not-found.component';
import { LoginComponent } from './login/login.component';
import { CheckoutComponent } from './checkout/checkout.component';
import { AuthGuardService } from './Services/authguard.service';
import { CanActivate } from './auth.guard'

//DEFINE ROUTE
const routes: Routes = [
  {path: '', component: HomeComponent},
  {path: 'Home', component: HomeComponent},
  {path: 'About', component: AboutComponent},
  {path: 'Contact', component: ContactComponent},
  {path: 'Courses', component: CoursesComponent},
  {path: 'Courses', children: [
    {path: 'Course/:id', component: CourseDetailComponent, canActivate: [CanActivate]},
    {path: 'Popular', component: PopularComponent, canActivate: [CanActivate]},
    {path: 'Checkout', component: CheckoutComponent, canActivate: [CanActivate]}
  ]},
  {path: 'Login', component: LoginComponent},
  {path: '**', component: NotFoundComponent},
]
```

- 💡 New feature allows for more granular access control on child routes within an application.

▷ 4:04:18

The screenshot shows the 'TS authguard.service.ts' file. The AuthGuardService implements CanActivate and CanActivateChild interfaces. It injects AuthService and Router. The canActivate method checks if the user is authenticated. The canActivateChild method delegates to the canActivate method of the AuthService.

```
export class AuthGuardService implements CanActivate, CanActivateChild{
  authService: AuthService = inject(AuthService);
  router: Router = inject(Router);

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
    boolean | Observable<boolean> | Promise<boolean>
  {
    if(this.authService.isAuthenticated()){
      return true;
    }else{
      this.router.navigate(['/Login']);
      return false;
    }
  }

  canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot):
    boolean | Observable<boolean> | Promise<boolean>
  {
    return this.canActivate(childRoute, state);
  }
}
```

- 💡 Auth Guard Service now implements the Connectivity Child interface, allowing for seamless assignment to the can activate child property.

▷ 4:10:17

The screenshot shows the VS Code interface with the file 'TS routing.module.ts' open. The code defines routes for various components like HomeComponent, AboutComponent, ContactComponent, CoursesComponent, CourseDetailComponent, PopularComponent, and CheckoutComponent. It also includes a canActivate guard for the Courses route.

```
import { CourseDetailComponent } from './courses/course-detail/course-detail.component';
import { NotFoundComponent } from './not-found/not-found.component';
import { LoginComponent } from './login/login.component';
import { CheckoutComponent } from './checkout/checkout.component';
import { AuthGuardService } from './Services/authguard.service';
import { CanActivate } from './auth.guard'

//DEFINE ROUTE
const routes: Routes = [
  {path: '', component: HomeComponent},
  {path: 'Home', component: HomeComponent},
  {path: 'About', component: AboutComponent},
  {path: 'Contact', component: ContactComponent},
  {path: 'Courses', component: CoursesComponent},
  {path: 'Courses', canActivateChild: [AuthGuardService], children: [
    {path: 'Course/:id', component: CourseDetailComponent},
    {path: 'Popular', component: PopularComponent},
    {path: 'Checkout', component: CheckoutComponent}
  ]},
  {path: 'Login', component: LoginComponent},
  {path: '**', component: NotFoundComponent},
]
```

✿ New security measures implemented to restrict access to certain web pages.

▷ 4:10:58

The screenshot shows the VS Code interface with the file 'TS auth.guard.ts' open. It contains two export statements: 'CanActivate' and 'CanActivateChild'. The 'CanActivate' function checks if the user is authenticated using the AuthService. If authenticated, it returns true; otherwise, it navigates to the '/Login' page and returns false. The 'CanActivateChild' function is defined as returning the result of the 'CanActivate' function.

```
import { inject } from "@angular/core";
import { AuthService } from "./Services/auth.service";
import { Router } from "@angular/router";

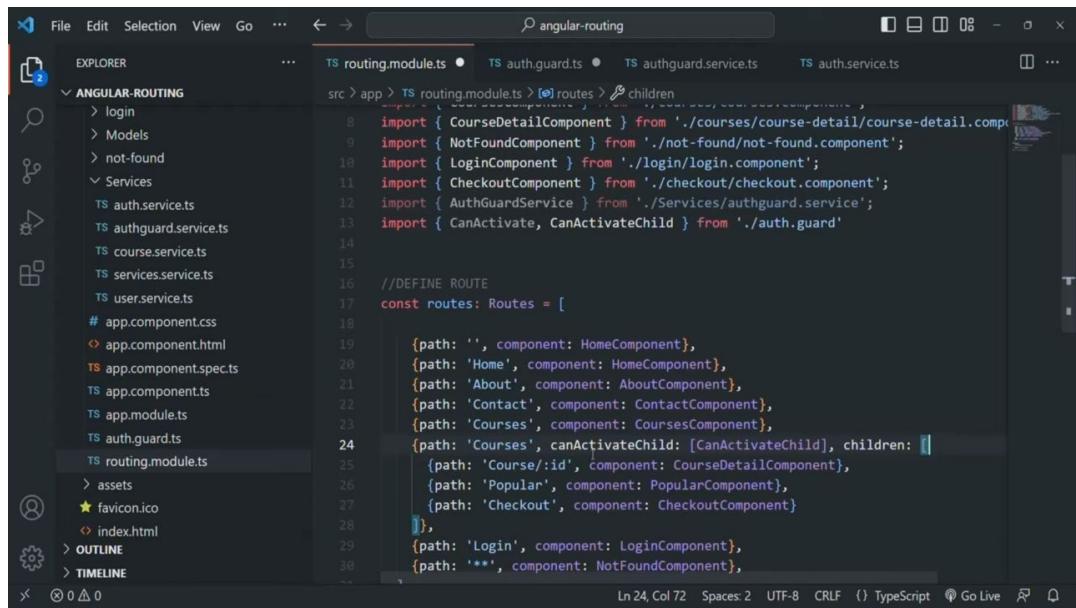
export const CanActivate = () => {
  const authService = inject(AuthService);
  const router = inject(Router);

  if(authService.isAuthenticated()){
    return true;
  }else{
    router.navigate(['/Login']);
    return false;
  }
}

export const CanActivateChild = () => [
  return CanActivate();
]
```

✿ The connectivity function returns a Boolean value and is imported into the routing module.

▷ 4:15:12



The screenshot shows a code editor with the file `routing.module.ts` open. The code defines routes for a single component, `HomeComponent`, which has several child routes. The `canActivateChild` method is used to protect these child routes. The code is as follows:

```
const routes: Routes = [
  {path: '', component: HomeComponent},
  {path: 'Home', component: HomeComponent},
  {path: 'About', component: AboutComponent},
  {path: 'Contact', component: ContactComponent},
  {path: 'Courses', component: CoursesComponent},
  {path: 'Courses', canActivateChild: [CanActivateChild], children: [
    {path: 'Course/:id', component: CourseDetailComponent},
    {path: 'Popular', component: PopularComponent},
    {path: 'Checkout', component: CheckoutComponent}
  ]},
  {path: 'Login', component: LoginComponent},
  {path: '**', component: NotFoundComponent},
]
```

- 💡 New feature in Angular allows for protection of child routes using the `connectactivate` function.

▷ 4:15:34

CanDeactivate Route Guard

- 💡 Learn about the `can deactivate` route guard in Angular and how it can be used to protect routes from navigation.

▷ 4:17:12

CanDeactivate Route Guard

We use CanDeactivate route guard to decide if the user can navigate away from a route or not. This route guard is called whenever we try to navigate away from the current route.

▷ 4:17:26

CanDeactivate Route Guard Use Case

The best use case where we can use CanDeactivate route guard is, when the user has filled a form like registration form and tries to navigate away without submitting. In that case, before navigating the user away, we might want to confirm if he wants to submit the data.

- ★ A can deactivate route guard is essential for preventing users from navigating away from a form before submitting it.

▷ 4:17:34

The screenshot shows the VS Code interface with the file 'routing.module.ts' open in the editor. The code defines routes for various components like HomeComponent, AboutComponent, ContactComponent, CoursesComponent, and CheckoutComponent, along with their respective paths and components. It also includes canActivate and canActivateChild guards.

```
src > app > TS routing.module.ts > [x] routes > [x] canActivate > [x] canActivateChild
11 import { CheckoutComponent } from './checkout/checkout.component';
12 import { AuthGuardService } from './Services/authguard.service';
13 import { CanActivate, CanActivateChild } from './auth.guard';
14
15 //DEFINE ROUTE
16 const routes: Routes = [
17   {
18     path: '',
19     component: HomeComponent,
20     # Home , component: HomeComponent,
21     path: 'About', component: AboutComponent,
22     path: 'Contact', component: ContactComponent, canActivate: [AuthGuardService]
23     path: 'Courses', component: CoursesComponent,
24     path: 'Courses', canActivateChild: [CanActivateChild], children: [
25       {
26         path: 'Course/:id', component: CourseDetailComponent,
27         path: 'Popular', component: PopularComponent,
28         path: 'Checkout', component: CheckoutComponent
29     }],
30     path: 'Login', component: LoginComponent,
31     path: '**', component: NotFoundComponent,
32   }
33 ]
34 @NgModule({
35   imports: [
36     RouterModule.forRoot(routes)
37   ]
38 })
39 export class AppRoutingModule { }
```

- Property assignment and implementation of route guards for the contact page.

▶ 4:23:33

The screenshot shows the VS Code interface with the file 'contact.component.html' open in the editor. It contains an HTML form with input fields for name, last name, and country, and a large text area for a message. The text area is bound to the 'ngModel' property.

```
<div>
  <div>
    <div>
      <form>
        <input type="text" placeholder="Your name.." [(ngModel)]="firstName">
        <input type="text" placeholder="Your last name.." [(ngModel)]="lastName">
        <select [(ngModel)]="country">
          <option>Australia</option>
        </select>
        <br>
        <label>
          <input type="text" placeholder="Write something.." style="height:200px" [(ngModel)]="message">
        </label>
      </form>
    </div>
  </div>
</div>
```

- Angular developers are now able to bind text areas to message properties using the NG model and forms module.

▶ 4:27:57

The screenshot shows the Visual Studio Code interface with the file 'contact.component.html' open. The code defines a form for a contact submission:

```
14  <div class="container">
15    <form>
16      <label for="fname">First Name</label>
17      <input type="text" class="contact-input-el" placeholder="Your name.." [(ngModel)]="firstName" name="fname">
18
19      <label for="lname">Last Name</label>
20      <input type="text" class="contact-input-el" placeholder="Your last name.." [(ngModel)]="lastName" name="lname">
21
22      <label for="country">Country</label>
23      <select id="country" name="country" [(ngModel)]="country">
24        <option value="australia">Australia</option>
25        <option value="canada">Canada</option>
26        <option value="usa">USA</option>
27      </select>
28
29      <label for="subject">Subject</label>
30      <textarea class="contact-input-el" placeholder="Write something.." style="height: 100px;" [(ngModel)]="message" name="subject">
31
32      <input type="submit" value="Submit" (click)="OnSubmit()">
33    </form>
34  </div>
35
36
```

- A new method for handling form submission and navigation has been introduced in the latest component class.

▶ 4:29:37

The screenshot shows the Visual Studio Code interface with the file 'contact.component.ts' open. The code defines a ContactComponent class with properties for first name, last name, country, and message, and methods for OnSubmit and canExit:

```
7  })
8  export class ContactComponent {
9    firstName: string = '';
10   lastName: string = '';
11   country: string = 'usa';
12   message: string = '';
13
14   isSubmitted: boolean = false;
15
16   OnSubmit(){
17     this.isSubmitted = true;
18   }
19
20   canExit(){
21     if(this.firstName || this.lastName || this.message){
22       return confirm('You have unsaved changes. Do you want to navigate away?')
23     }
24     else{
25       return true;
26     }
27   }
28
29
```

- New method implemented in contact component for activating route card.

▶ 4:32:16

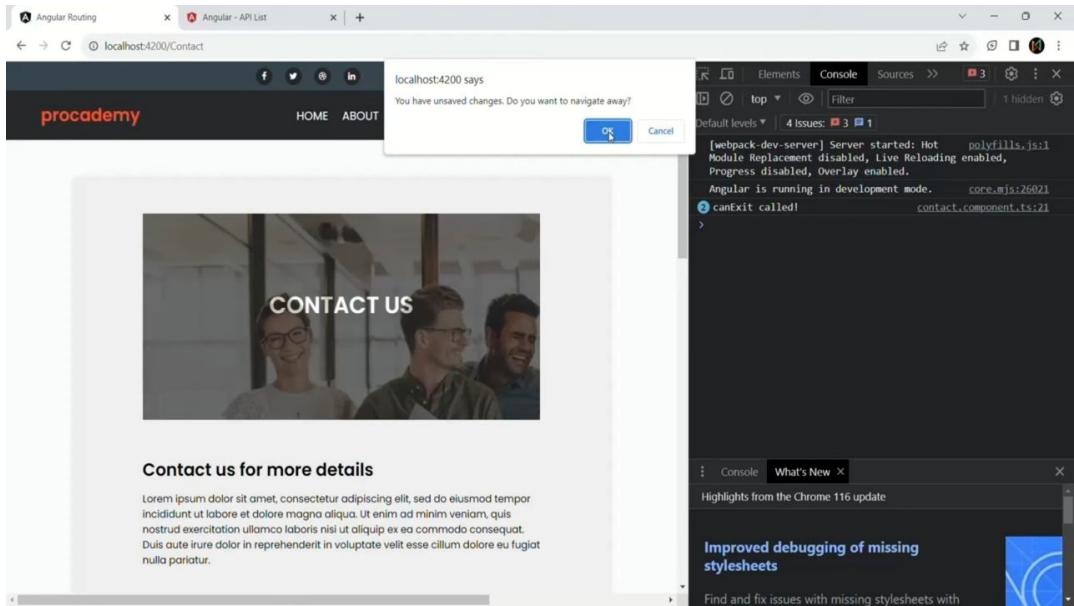
```
11  router: Router = inject(Router);
12
13  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
14    boolean | Observable<boolean> | Promise<boolean>
15  {
16    if(this.authService.isAuthenticated()){
17      return true;
18    }else{
19      this.router.navigate(['/Login']);
20      return false;
21    }
22  }
23
24  canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot):
25    boolean | Observable<boolean> | Promise<boolean>
26  {
27    return this.canActivate(childRoute, state);
28  }
29
30  canDeactivate(component: ContactComponent, currentRoute: ActivatedRouteSnapshot,
31  currentState: RouterStateSnapshot, nextState: RouterStateSnapshot)
32  {
33    return component.canExit();
34  }

```

Ln 32, Col 36 Spaces: 4 UTF-8 CRLF () TypeScript Go Live

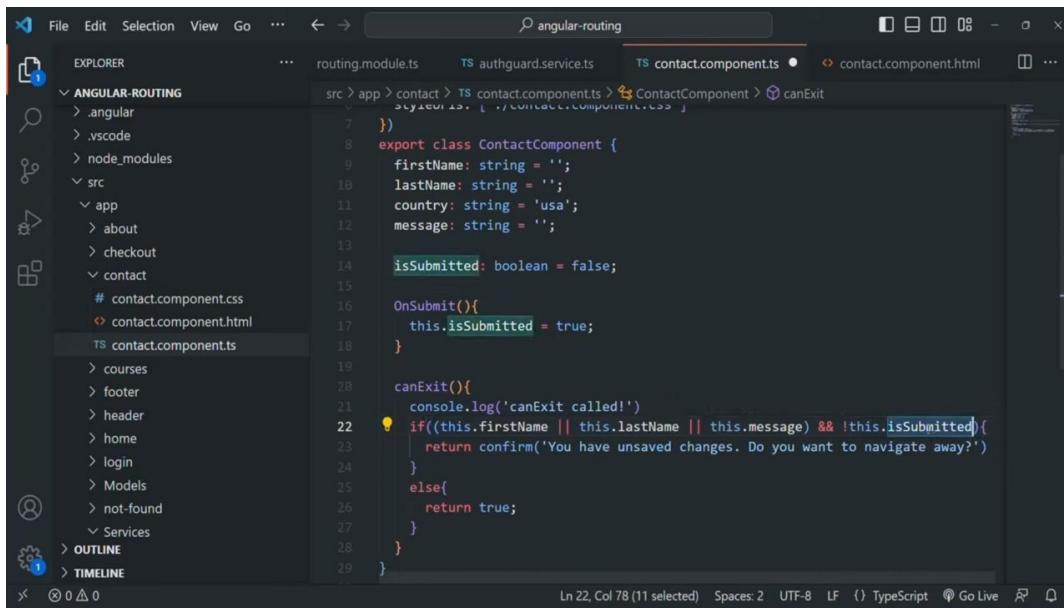
- 💡 New implementation on the contact page allows users to navigate away without filling out the form.

▷ 4:32:46



- 💡 Navigating away from unsaved changes in a form can be easily managed using the deactivate route card.

▷ 4:34:19

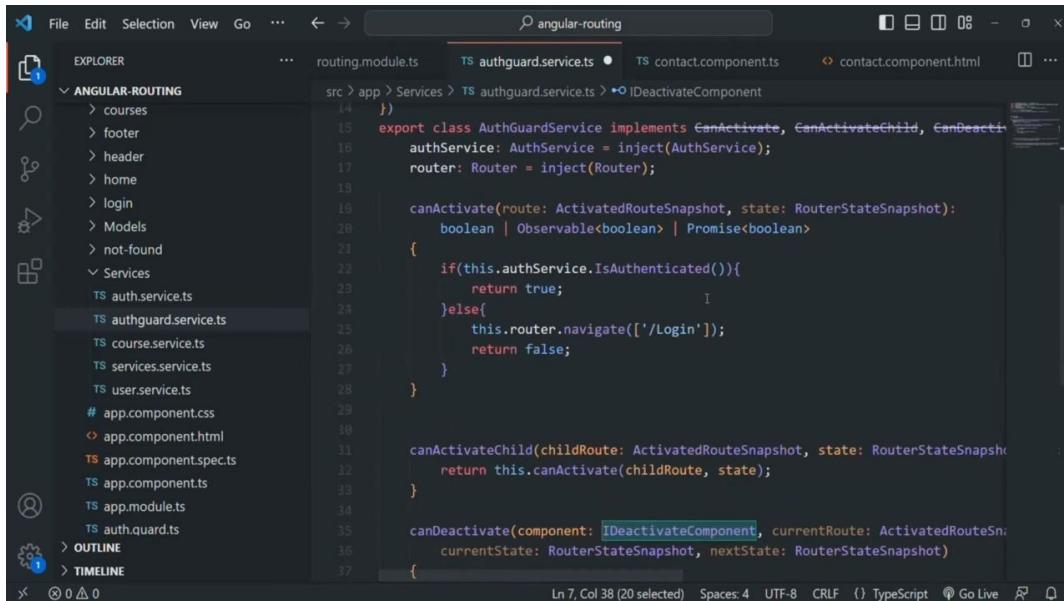


```
7  })
8  export class ContactComponent {
9    firstName: string = '';
10   lastName: string = '';
11   country: string = 'usa';
12   message: string = '';
13
14   isSubmitted: boolean = false;
15
16   OnSubmit(){
17     this.isSubmitted = true;
18   }
19
20   canExit(){
21     console.log('canExit called!');
22     if((this.firstName || this.lastName || this.message) && !this.isSubmitted){
23       return confirm('You have unsaved changes. Do you want to navigate away?')
24     }
25     else{
26       return true;
27     }
28   }
29 }
```

Ln 22, Col 78 (11 selected) Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live

💡 New feature in author guard service.ts file prevents users from navigating away without saving changes.

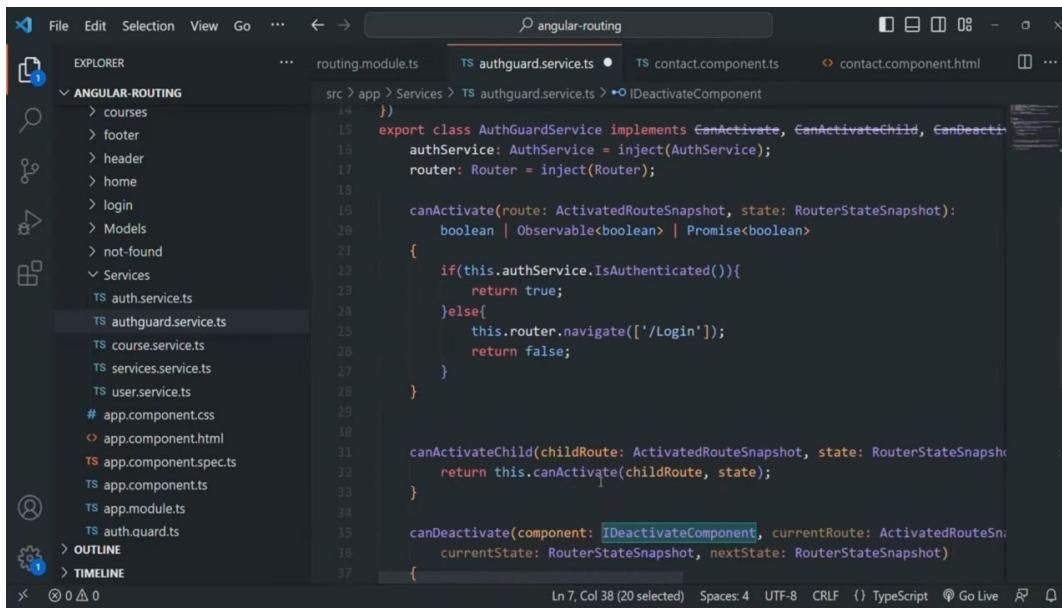
▶ 4:35:51



```
14  })
15  export class AuthGuardService implements CanActivate, CanActivateChild, CanDeactivate<any> {
16    authService: AuthService = inject(AuthService);
17    router: Router = inject(Router);
18
19    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean> {
20      if(this.authService.isAuthenticated()){
21        return true;
22      }
23      else{
24        this.router.navigate(['/Login']);
25        return false;
26      }
27    }
28
29    canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean> {
30      return this.canActivate(childRoute, state);
31    }
32
33    canDeactivate(component: IDeactivateComponent, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot, nextState: RouterStateSnapshot): Observable<boolean> {
34      return of(true);
35    }
36  }
```

Ln 7, Col 38 (20 selected) Spaces: 4 UTF-8 CRLF () TypeScript ⚡ Go Live

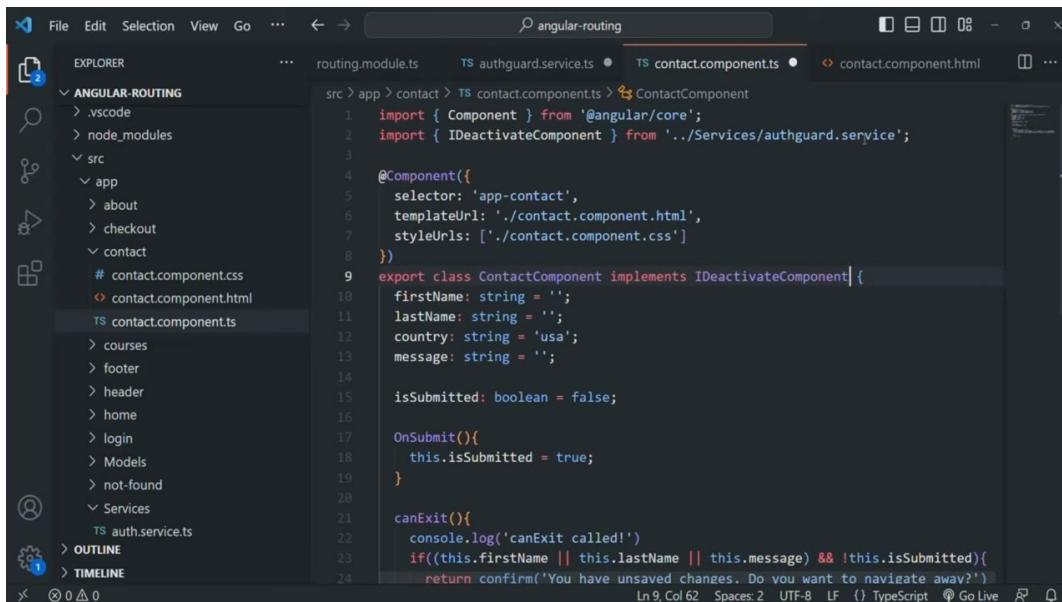
▶ 4:38:39



```
14    })
15    export class AuthGuardService implements CanActivate, CanActivateChild, CanDeactivate<any> {
16      authService: AuthService = inject(AuthService);
17      router: Router = inject(Router);
18
19      canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean>
20      {
21        if(this.authService.isAuthenticated()){
22          return true;
23        }else{
24          this.router.navigate(['/Login']);
25          return false;
26        }
27      }
28
29      canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean>
30      {
31        return this.canActivate(childRoute, state);
32      }
33
34      canDeactivate(component: IDeactivateComponent, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot, nextState: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean>
35      {
36
37    }
```

Developing a contact component that inherits the can deactivate route functionality from the IDeactivateComponent interface is essential for ensuring smooth navigation within the application.

4:38:39



```
1  import { Component } from '@angular/core';
2  import { IDeactivateComponent } from '../Services/authguard.service';
3
4  @Component({
5    selector: 'app-contact',
6    templateUrl: './contact.component.html',
7    styleUrls: ['./contact.component.css']
8  })
9  export class ContactComponent implements IDeactivateComponent {
10    firstName: string = '';
11    lastName: string = '';
12    country: string = 'usa';
13    message: string = '';
14
15    isSubmitted: boolean = false;
16
17    onSubmit(){
18      this.isSubmitted = true;
19    }
20
21    canExit(){
22      console.log('canExit called!');
23      if((this.firstName || this.lastName || this.message) && !this.isSubmitted){
24        return confirm('You have unsaved changes. Do you want to navigate away?');
25      }
26    }
27  }
```

Component inheritance is achieved through the use of the "implements" keyword and implementing from a specific interface.

4:38:56

```
11 lastName: string = '';
12 country: string = 'usa';
13 message: string = '';
14
15 isSubmitted: boolean = false;
16
17 OnSubmit(){
18     this.isSubmitted = true;
19 }
20
21 canExit(){
22     console.log('canExit called!');
23     if((this.firstName || this.lastName || this.message) && !this.isSubmitted){
24         return confirm('You have unsaved changes. Do you want to navigate away?')
25     }
26     else{
27         return true;
28     }
29 }
30
31 }
```

- Software developers are implementing a new feature to prompt user confirmation before navigating away from a page.

▶ 4:39:14

```
11 { checkout/checkout.component';
12 'services/authguard.service';
13 } from './auth.guard'
14
15
16
17
18
19 :},
20 onActivate,
21 onDeactivate,
22 { component, canDeactivate: [(comp: ContactComponent) => {return comp.canExit();}]
23 :isComponent|,
24 [CanActivateChild], children: [
25 { component: CourseDetailComponent },
26 { component: LarComponent },
27 { component: CheckoutComponent }
28
29 :onDeactivate,
30 :onComponent|,
31
32
33
34 }
```

- Deactivate route card is a feature used in Angular applications to implement the can deactivate function.

▶ 4:42:42

Resolve Route Guard

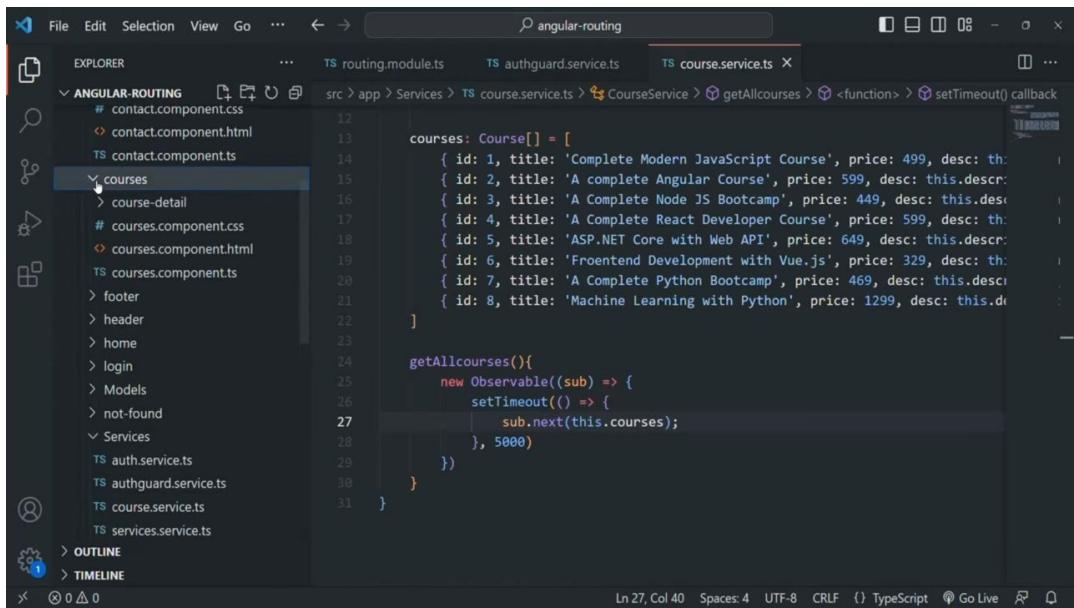
- 💡 Learn about the resolved route guard in Angular and how it can be used to load data before navigating to a route.

▷ 4:43:05

Resolve Route Guard

Resolve route guard in Angular can be used when we want to load some data before we navigate to a route.

▷ 4:43:10

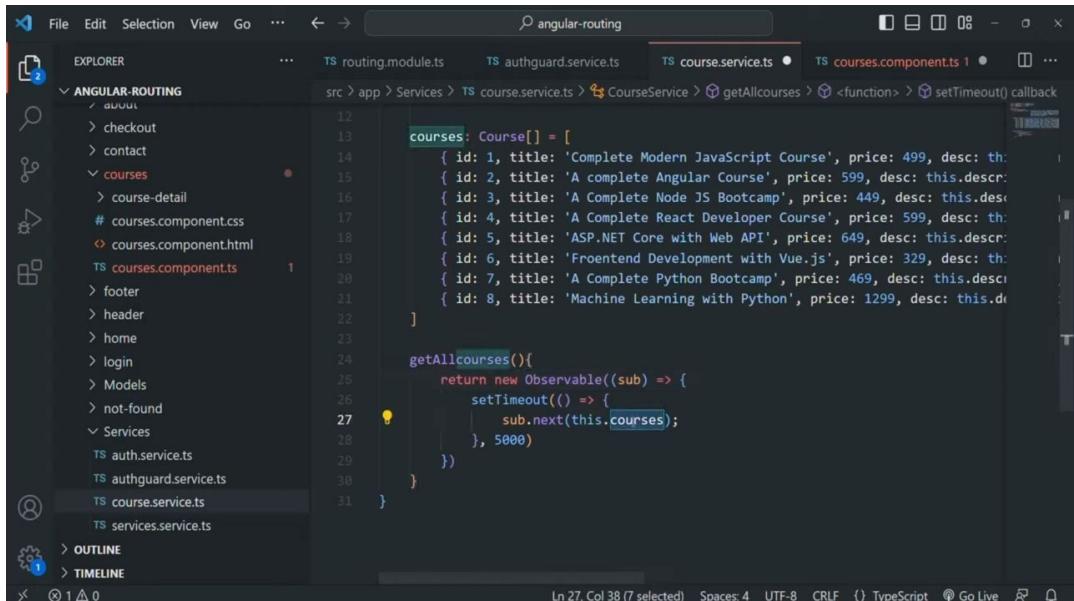


```
courses: Course[] = [
  { id: 1, title: 'Complete Modern JavaScript Course', price: 499, desc: 'This course covers the basics of JavaScript, including ES6 features, and how to build complex web applications using frameworks like React and Angular.' },
  { id: 2, title: 'A Complete Angular Course', price: 599, desc: 'This course is designed for beginners and covers the fundamentals of Angular, including routing, services, and state management.' },
  { id: 3, title: 'A Complete Node JS Bootcamp', price: 449, desc: 'This course teaches you how to build server-side applications using Node.js and MongoDB, including RESTful APIs and real-time communication with Socket.io.' },
  { id: 4, title: 'A Complete React Developer Course', price: 599, desc: 'This course is perfect for those who want to learn how to build user interfaces using the React library and Redux for state management.' },
  { id: 5, title: 'ASP.NET Core with Web API', price: 649, desc: 'This course covers the basics of ASP.NET Core and how to build web APIs using Entity Framework Core and Swagger documentation.' },
  { id: 6, title: 'Frontend Development with Vue.js', price: 329, desc: 'This course is for front-end developers who want to learn how to build dynamic web applications using the Vue.js framework.' },
  { id: 7, title: 'A Complete Python Bootcamp', price: 469, desc: 'This course covers the basics of Python programming and how to build web applications using Django and Flask frameworks.' },
  { id: 8, title: 'Machine Learning with Python', price: 1299, desc: 'This course is for data scientists and machine learning enthusiasts who want to learn how to build machine learning models using Python and libraries like TensorFlow and PyTorch.' }
]

getAllcourses(){
  return new Observable((sub) => {
    setTimeout(() => {
      sub.next(this.courses);
    }, 5000)
  })
}
```

- The courses component uses the "all courses" property to display course information.

▷ 4:46:27

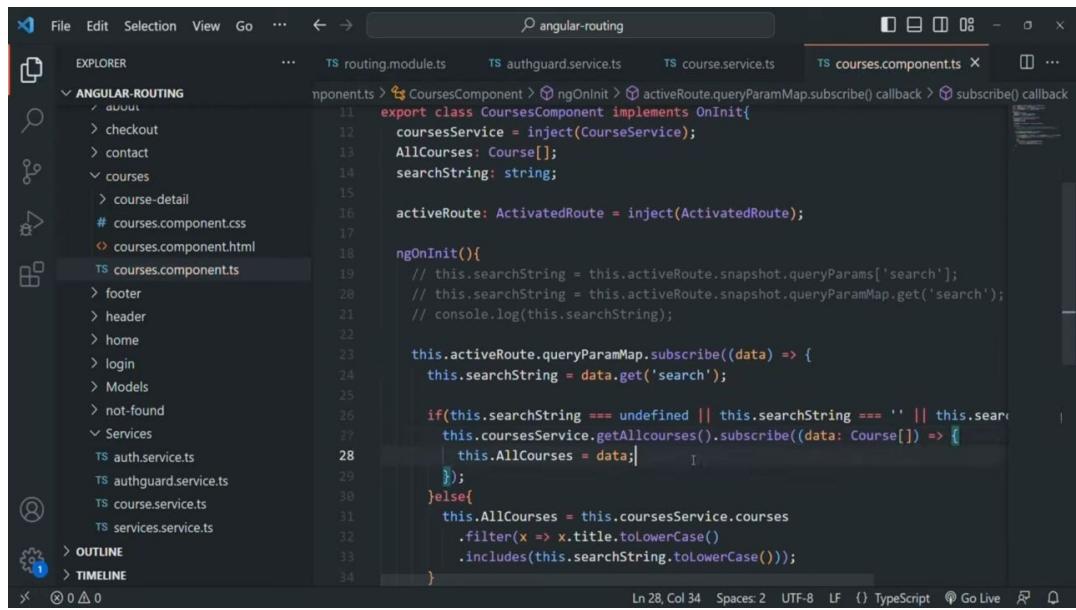


```
courses: Course[] = [
  { id: 1, title: 'Complete Modern JavaScript Course', price: 499, desc: 'This course covers the basics of JavaScript, including ES6 features, and how to build complex web applications using frameworks like React and Angular.' },
  { id: 2, title: 'A Complete Angular Course', price: 599, desc: 'This course is designed for beginners and covers the fundamentals of Angular, including routing, services, and state management.' },
  { id: 3, title: 'A Complete Node JS Bootcamp', price: 449, desc: 'This course teaches you how to build server-side applications using Node.js and MongoDB, including RESTful APIs and real-time communication with Socket.io.' },
  { id: 4, title: 'A Complete React Developer Course', price: 599, desc: 'This course is perfect for those who want to learn how to build user interfaces using the React library and Redux for state management.' },
  { id: 5, title: 'ASP.NET Core with Web API', price: 649, desc: 'This course covers the basics of ASP.NET Core and how to build web APIs using Entity Framework Core and Swagger documentation.' },
  { id: 6, title: 'Frontend Development with Vue.js', price: 329, desc: 'This course is for front-end developers who want to learn how to build dynamic web applications using the Vue.js framework.' },
  { id: 7, title: 'A Complete Python Bootcamp', price: 469, desc: 'This course covers the basics of Python programming and how to build web applications using Django and Flask frameworks.' },
  { id: 8, title: 'Machine Learning with Python', price: 1299, desc: 'This course is for data scientists and machine learning enthusiasts who want to learn how to build machine learning models using Python and libraries like TensorFlow and PyTorch.' }
]

getAllcourses(){
  return new Observable((sub) => {
    setTimeout(() => {
      sub.next(this.courses);
    }, 5000)
  })
}
```

- The observable is set to emit a list of courses, but the data parameter type needs to be specified in order to fix an error in the code.

▷ 4:47:43



```
export class CoursesComponent implements OnInit {
  coursesService = inject(CourseService);
  AllCourses: Course[];
  searchString: string;

  activeRoute: ActivatedRoute = inject(ActivatedRoute);

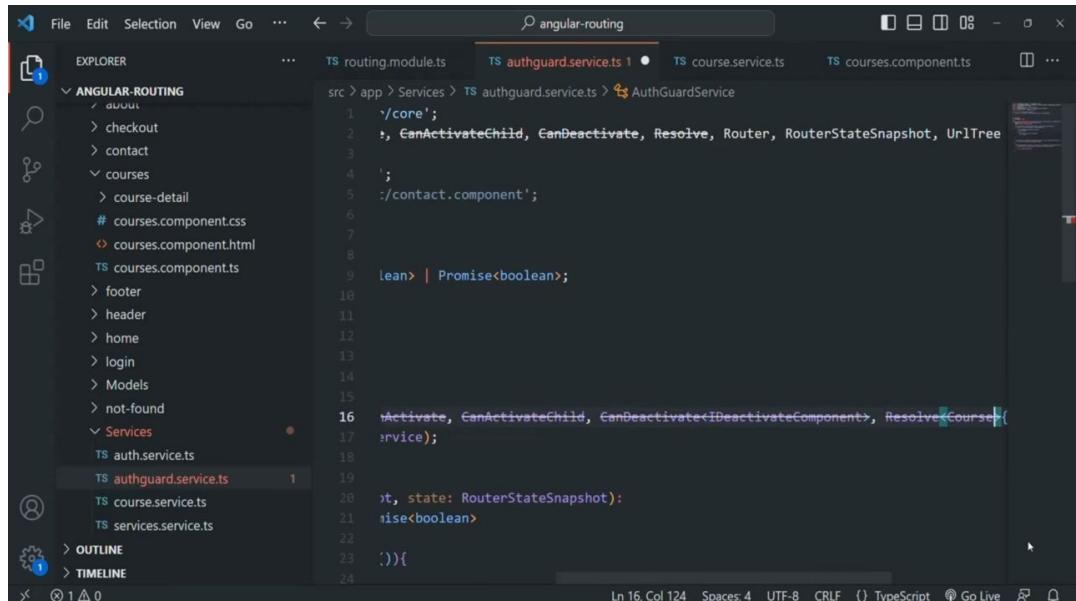
  ngOnInit(){
    // this.searchString = this.activeRoute.snapshot.queryParams['search'];
    // this.searchString = this.activeRoute.snapshot.queryParamMap.get('search');
    // console.log(this.searchString);

    this.activeRoute.queryParamMap.subscribe((data) => {
      this.searchString = data.get('search');

      if(this.searchString === undefined || this.searchString === '' || this.searchString === null){
        this.coursesService.getAllcourses().subscribe((data: Course[]) => {
          this.AllCourses = data;
        });
      }else{
        this.AllCourses = this.coursesService.courses
          .filter(x => x.title.toLowerCase()
          .includes(this.searchString.toLowerCase()));
      }
    })
  }
}
```

- The course list on the web page will be available after five seconds of navigating to the courses link.

▷ 4:48:09



```
src > app > Services > TS authguard.service.ts 1 ● TS course.service.ts
1   'core';
2   *, CanActivateChild, CanDeactivate, Resolve, Router, RouterStateSnapshot, UrlTree
3
4   ';
5   ./contact.component';
6
7
8
9   lean> | Promise<boolean>;
10
11
12
13
14
15
16   activate, CanActivateChild, CanDeactivate<IDeactivateComponent>, Resolve<Course>,
17   service);
18
19
20   pt, state: RouterStateSnapshot):
21   nise<boolean>
22
23 }()
24
```

- Service classes inheriting from resolved route guards must specify the type of data they will emit, such as an array of courses.

▷ 4:51:09

```
src > app > Services > authguard.service.ts > resolve
    this.router.navigate(['/Login']);
    return false;
}

canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return this.canActivate(childRoute, state);
}

canDeactivate(component: IDeactivateComponent, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot, nextState: RouterStateSnapshot): boolean {
    return component.canExit();
}

resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Course[] {
    let courseList: Course[] = [];
    this.courseService.getAllcourses().subscribe((courses: Course[]) => {
        courseList = courses;
    });
    return courseList;
}
```

Resolve guards in Angular can be used to load data before navigating to a route by using the let keyword instead of const and returning the course list from the resolve method.

▶ 4:53:55

```
src > app > TS routing.module.ts > routes > resolve > courses
    routes = [
        { path: '', component: HomeComponent },
        { path: 'about', component: AboutComponent },
        { path: 'contact', component: ContactComponent, canDeactivate: [(comp: ContactComponent) => {
            if (!comp.isDirty) {
                return true;
            }
            return comp.onLeave();
        }] },
        { path: 'courses', component: CoursesComponent, resolve: {courses: AuthGuardService} },
        { path: 'course/:id', component: CourseDetailComponent },
        { path: 'popular', component: PopularComponent },
        { path: 'checkout', component: CheckoutComponent },
        { path: 'login', component: LoginComponent },
        { path: '**', component: NotFoundComponent },
    ];
}

module.forRoot(routes)
```

New Auth Card Service Implements Resolve Interface for Course Data Assignment.

▶ 4:55:37

The screenshot shows the VS Code interface with the title bar "angular-routing". The Explorer sidebar on the left shows the project structure under "ANGULAR-ROUTING", including files like "courses.component.ts", "auth.guard.service.ts", and "course.service.ts". The main editor area displays the "courses.component.ts" file, which contains code for a CoursesComponent that implements OnInit. It uses the ActivatedRoute.queryParamMap observable to handle search queries and the course service to get all courses. The status bar at the bottom shows "Ln 31, Col 69" and other settings.

```
src > app > courses > TS courses.component.ts > CoursesComponent > ngOnInit > activeRoute.queryParamMap.subscribe
11 export class CoursesComponent implements OnInit{
12   coursesService = inject(CourseService);
13   AllCourses: Course[];
14   searchString: string;
15
16   activeRoute: ActivatedRoute = inject(ActivatedRoute);
17
18   ngOnInit(){
19     // this.searchString = this.activeRoute.snapshot.queryParams['search'];
20     // this.searchString = this.activeRoute.snapshot.queryParamMap.get('search');
21     // console.log(this.searchString);
22
23     this.activeRoute.queryParamMap.subscribe((data) => {
24       this.searchString = data.get('search');
25
26       if(this.searchString === undefined || this.searchString === '' || this.searchString === null) {
27         // this.coursesService.getAllcourses().subscribe((data: Course[]) => {
28         //   this.AllCourses = data;
29         // });
30
31       this.AllCourses = this.activeRoute.snapshot.data['courses'];
32     }else{
33       this.AllCourses = this.coursesService.courses
34         .filter(x => x.title.toLowerCase()
```

Website Redirect Issue Causes Course List to Disappear.

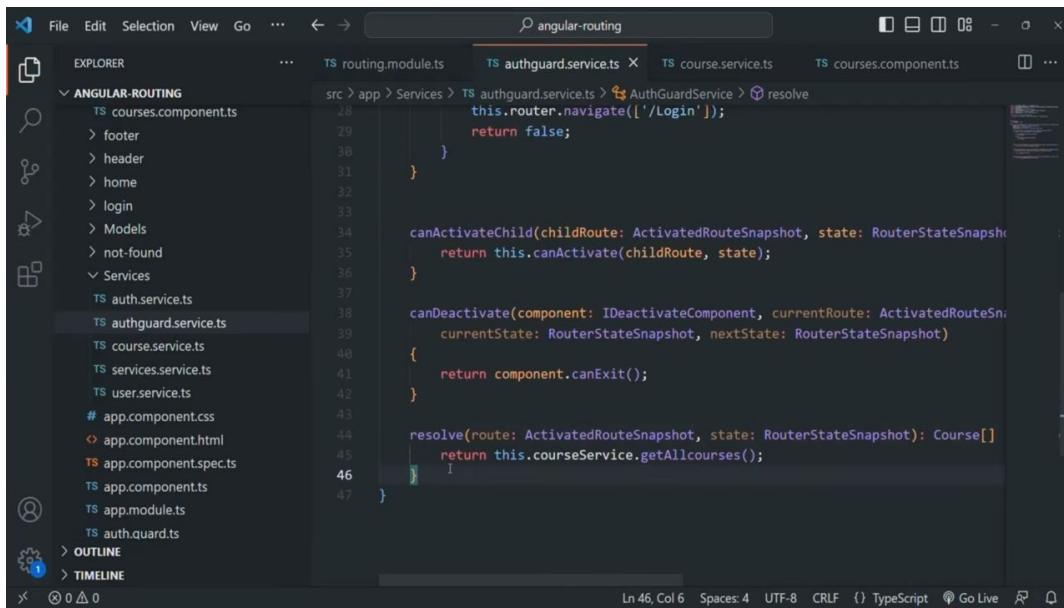
▶ 4:57:46

The screenshot shows the VS Code interface with the title bar "angular-routing". The Explorer sidebar on the left shows the project structure under "ANGULAR-ROUTING", including files like "courses.component.ts", "auth.guard.service.ts", and "course.service.ts". The main editor area displays the "course.service.ts" file, which contains the CourseService class. It includes an observable "courses" array with 8 course objects and an "getAllcourses" method that returns an observable. The status bar at the bottom shows "Ln 25, Col 51" and other settings.

```
src > app > Services > TS course.service.ts > CourseService > getAllcourses > <function>
12   courses: Course[] = [
13     { id: 1, title: 'Complete Modern JavaScript Course', price: 499, desc: 'This course covers everything you need to know about modern JavaScript development.' },
14     { id: 2, title: 'A complete Angular Course', price: 599, desc: 'This course is designed for beginners and covers the basics of Angular 10+.' },
15     { id: 3, title: 'A Complete Node JS Bootcamp', price: 449, desc: 'This course is perfect for those who want to learn how to build server-side applications using Node.js.' },
16     { id: 4, title: 'A Complete React Developer Course', price: 599, desc: 'This course is ideal for those who want to learn how to build user interfaces using React.' },
17     { id: 5, title: 'ASP.NET Core with Web API', price: 649, desc: 'This course is designed for those who want to learn how to build web APIs using ASP.NET Core.' },
18     { id: 6, title: 'Frontend Development with Vue.js', price: 329, desc: 'This course is perfect for those who want to learn how to build modern front-end applications using Vue.js.' },
19     { id: 7, title: 'A Complete Python Bootcamp', price: 469, desc: 'This course is ideal for those who want to learn how to build web applications using Python and Django.' },
20     { id: 8, title: 'Machine Learning with Python', price: 1299, desc: 'This course is designed for those who want to learn how to build machine learning models using Python and TensorFlow.' }
21   ]
22
23   getAllcourses(){
24     return new Observable<Course[]>((sub) => {
25       setTimeout(() => {
26         sub.next(this.courses);
27       }, 5000)
28     })
29   }
30 }
```

Developing observable data types for a web page.

▶ 5:00:04



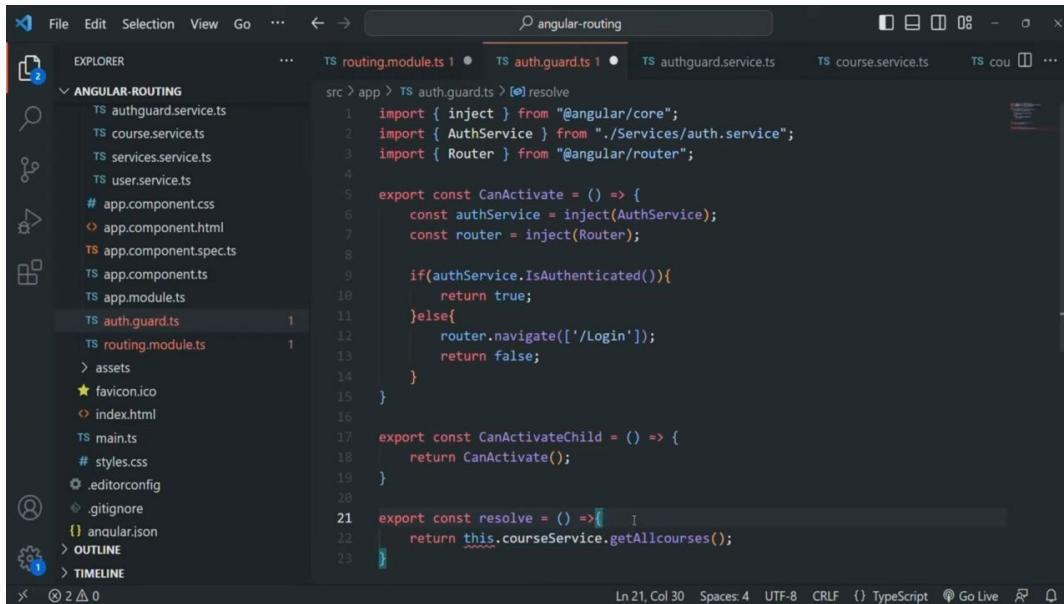
```
src > app > Services > TS authguard.service.ts > AuthGuardService > resolve
    this.router.navigate(['/Login']);
    return false;
}

canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return this.canActivate(childRoute, state);
}

canDeactivate(component: IDeactivateComponent, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot, nextState: RouterStateSnapshot): boolean {
    return component.canExit();
}

resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Course[] {
    return this.courseService.getAllcourses();
}
```

▶ 5:00:17



```
src > app > TS auth.guard.ts > resolve
    import { inject } from '@angular/core';
    import { AuthService } from './Services/auth.service';
    import { Router } from '@angular/router';

    export const CanActivate = () => {
        const authService = inject(AuthService);
        const router = inject(Router);

        if(authService.isAuthenticated()){
            return true;
        }else{
            router.navigate(['/Login']);
            return false;
        }
    }

    export const CanActivateChild = () => {
        return CanActivate();
    }

    export const resolve = () => {
        return this.courseService.getAllcourses();
    }

```

- ★ In the resolve method, we create a variable called course service and ask Angular to inject an instance of this service.

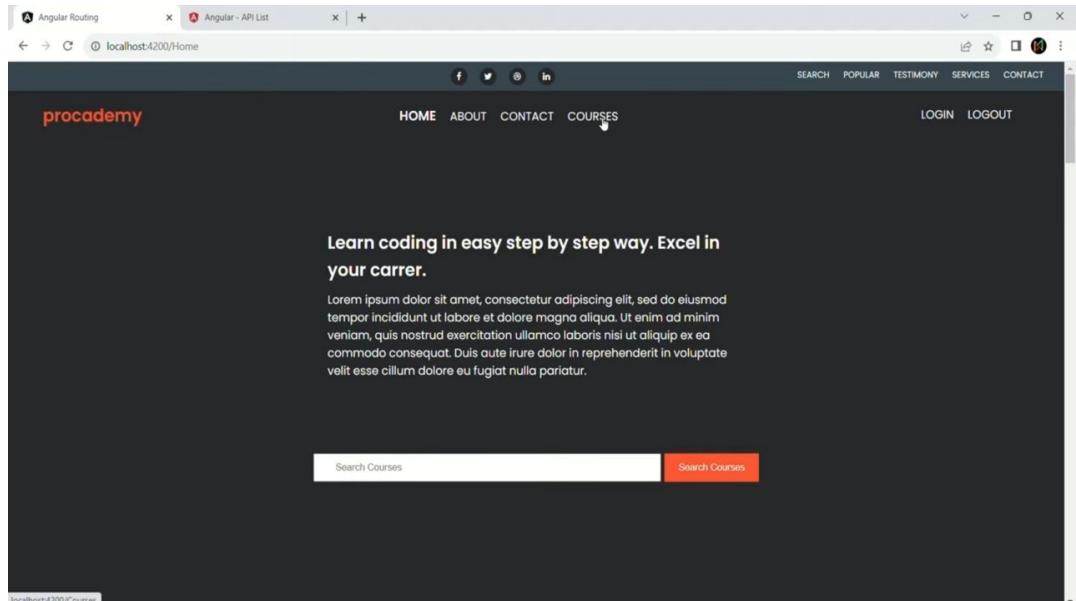
▶ 5:03:23

The screenshot shows the VS Code interface with the file 'TS routing.module.ts' open. The code defines routes for various components like HomeComponent, AboutComponent, ContactComponent, CoursesComponent, CourseDetailComponent, NotFoundComponent, LoginComponent, and CheckoutComponent. It also includes imports for AuthGuardService and its methods CanActivateChild, resolve, and canActivate.

```
src > app > TS routing.module.ts > [0] routes
  8 import { CourseDetailComponent } from './courses/course-detail/course-detail.component';
  9 import { NotFoundComponent } from './not-found/not-found.component';
10 import { LoginComponent } from './login/login.component';
11 import { CheckoutComponent } from './checkout/checkout.component';
12 import { AuthGuardService } from './Services/authguard.service';
13 import { CanActivate, CanActivateChild, resolve } from './auth.guard'
14
15
16 //DEFINE ROUTE
17 const routes: Routes = [
18   {
19     path: '',
20     component: HomeComponent,
21   },
22   {
23     path: 'About',
24     component: AboutComponent,
25   },
26   {
27     path: 'Contact',
28     component: ContactComponent,
29     canDeactivate: [(comp: ContactComponent) => comp.canDeactivate()]
30   },
31   {
32     path: 'Courses',
33     component: CoursesComponent,
34     resolve: {courses: resolve}
35   },
36   {
37     path: 'Courses/:id',
38     component: CourseDetailComponent,
39   },
40   {
41     path: 'Popular',
42     component: PopularComponent,
43   },
44   {
45     path: 'Checkout',
46     component: CheckoutComponent
47   },
48   {
49     path: 'Login',
50     component: LoginComponent,
51   },
52   {
53     path: '**',
54     component: NotFoundComponent,
55   }
56 ];
```

- ✿ New functionality has been added to the courses page, allowing for a delay in displaying data.

▷ 5:04:16



- ✿ Resolve route card is a useful tool for ensuring that data is available before navigating to a specific page on a website.

▷ 5:04:33

Router Navigation Events

- ★ Learn how to implement a loading spinner during data loading and understand navigation events in Angular.

▷ 5:05:21

What is Navigation Events

In Angular, when we navigate from one route to another route, there is a sequence of navigation events that gets triggered by Angular router. We can subscribe to these events and execute some logic if we want..

▷ 5:05:32

The screenshot shows a code editor with the file `src/app/routing.module.ts` open. The code defines a `RouterModule` with various routes and configurations. A specific line of code is highlighted:

```
  {path: 'Courses', canActivate: true, component: CoursesComponent}
```

This line includes the `canActivate` guard set to `true`. The code editor interface includes tabs for `File`, `Edit`, `Selection`, `View`, `Go`, and `EXPLORER`. The status bar at the bottom shows the line number (Ln 35), column number (Col 58), and other settings.

- Developers can enable tracing to log navigation events in the browser's developer console by specifying an "enable tracing" property as true in an anonymous object.

▶ 5:06:46

The screenshot shows a web browser displaying a website for "proacademy". The page has a header with links for `SEARCH`, `POPULAR`, `TESTIMONY`, `SERVICES`, and `CONTACT`. Below the header is a navigation bar with `HOME`, `ABOUT`, `CONTACT`, and `COURSES`. The main content area features a large image of four people and a section titled "ABOUT US". To the left, there's a sidebar with "Behind the Success" and some placeholder text. On the right, the developer console is open, showing a list of navigation events logged in the browser's developer tools. The events listed include:

- GuardsCheckEnd
- Router Event: ResolveStart
- ResolveStart(id: 1, url: '/About', route: Route{url: '', path: ''})
- ResolveStart
- Router Event: ResolveEnd
- ResolveEnd(id: 1, url: '/About', route: Route{url: '', path: ''})
- Router Event: ActivationEnd
- ActivationEnd(path: '/About')
- ActivationEnd
- Router Event: ChildActivationEnd
- ChildActivationEnd(path: '')
- ChildActivationEnd
- Router Event: NavigationEnd
- NavigationEnd(id: 1, url: '/About', route: Route{url: '', path: ''})
- NavigationEnd
- Router Event: Scroll
- Scroll(anchor: 'null', position: 'null')
- Scroll

- Developers demonstrate how to log navigation events in a web application using the developer console.

▶ 5:07:08

The screenshot shows the VS Code interface with the Angular Routing project open. The Explorer sidebar on the left lists files like routing.module.ts, app.component.html, and styles.css. The main editor area displays the contents of styles.css, which includes CSS for a loader and a keyframe rotation animation.

```
src > # styles.css > .loader
24 border-radius: 50%; 
25 display: inline-block; 
26 box-sizing: border-box; 
27 animation: rotation 1s linear infinite; 
28 left: 50%; 
29 position: absolute; 
30 top: 50%; 
31 transform: translate(-50%, -50%); 
32 } 

33 @keyframes rotation { 
34 0% { 
35     transform: rotate(0deg); 
36 } 
37 100% { 
38     transform: rotate(360deg); 
39 } 
40 } 

41 .overlay{ 
42     left: 0; 
43     top: 0; 
44     width: 100%; 
45     height: 100%; 
46 } 
```

- 💡 New CSS classes have been added to the style.css file for a loading indicator on web pages.

▷ 5:11:56

The screenshot shows the VS Code interface with the Angular Routing project open. The Explorer sidebar on the left lists files like routing.module.ts, app.component.html, and styles.css. The main editor area displays the contents of app.component.html, which includes an ngIf directive for the loader component.

```
src > app > app.component.html > div.main-page-container > div.overlay
1 <div class="main-page-container">
2   <app-header></app-header>
3
4   <!-- <app-home></app-home> -->
5   <!-- <app-contact></app-contact> -->
6   <!-- <app-about></app-about> -->
7   <!-- <app-courses></app-courses> -->
8   <!-- <app-login></app-login> -->
9   <router-outlet></router-outlet>
10
11   <div class="overlay" *ngIf="showLoader">
12     <div class="loader"></div>
13   </div>[ 
14
15   <app-footer></app-footer>
16
17 </div>
```

- 💡 Loading indicator removed from web page based on show loader value in appcomponent.ts file.

▷ 5:13:04

```
src > app > TS app.component.ts > AppComponent > ngOnInit > router.events.subscribe() callback
import { Router, Event, NavigationStart, NavigationEnd } from '@angular/router'

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular Routing';

  showLoader: boolean = false;

  ngOnInit(){
    this.router.events.subscribe((routerEvent: Event) => {
      if(routerEvent instanceof NavigationStart){
        this.showLoader = true;
      }

      if(routerEvent instanceof NavigationEnd){
        this.showLoader = false;
      }
    })
  }
}

Ln 24, Col 33  Spaces: 2  UTF-8  LF  {}  TypeScript  Go Live
```

- Storing an instance of navigation end and setting the show loader to false can improve user experience on web pages.

▶ 5:17:43

```
src > app > TS app.component.ts > AppComponent > ngInit > router.events.subscribe() callback
import { Router, Event, NavigationStart, NavigationEnd, NavigationCancel } from '@angular/router'

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular Routing';

  showLoader: boolean = false;

  ngOnInit(){
    this.router.events.subscribe((routerEvent: Event) => {
      if(routerEvent instanceof NavigationStart){
        this.showLoader = true;
      }

      if(routerEvent instanceof NavigationEnd || routerEvent instanceof NavigationCancel){
        this.showLoader = false;
      }
    })
  }
}

Ln 23, Col 89  Spaces: 2  UTF-8  LF  {}  TypeScript  Go Live
```

- To handle the router event and hide the loading indicator, we need to import navigation cancel from angular/router.

▶ 5:20:34

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "ANGULAR-ROUTING".
- Editor:** Displays the file "app.component.ts" with the following code:

```
src > app > TS app.components.ts > AppComponent > ngOnInit > router.events.subscribe() callback
9  })
10 export class AppComponent {
11   title = 'Angular Routing';
12
13   showLoader: boolean = false;
14
15   router: Router = inject(Router);
16
17   ngOnInit(){
18     this.router.events.subscribe((routerEvent: Event) => {
19       if(routerEvent instanceof NavigationStart){
20         this.showLoader = true;
21       }
22
23       if(routerEvent instanceof NavigationEnd
24       || routerEvent instanceof NavigationCancel
25       || routerEvent instanceof NavigationError)
26       {
27         this.showLoader = false;
28       }
29     })
30   }
31 }
```

Bottom status bar: Ln 25, Col 50 Spaces: 2 UTF-8 LF {} TypeScript Go Live

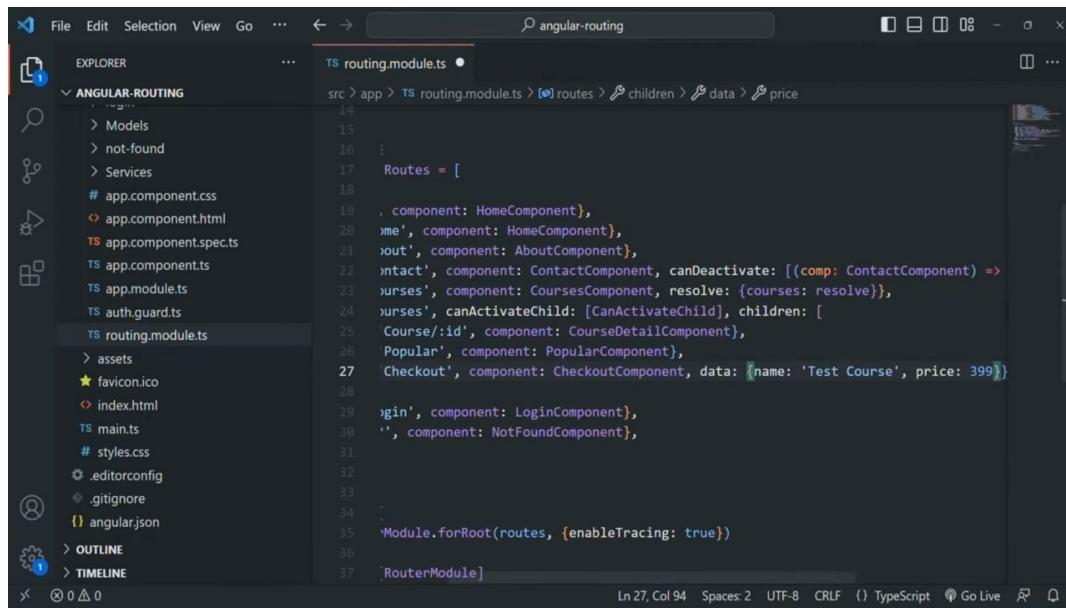
💡 Using router events in Angular to track navigation errors and outcomes.

▶ 5:22:03

Passing Data to Route

💡 Angular allows for passing data through routes, both statically and dynamically, providing flexibility in application development.

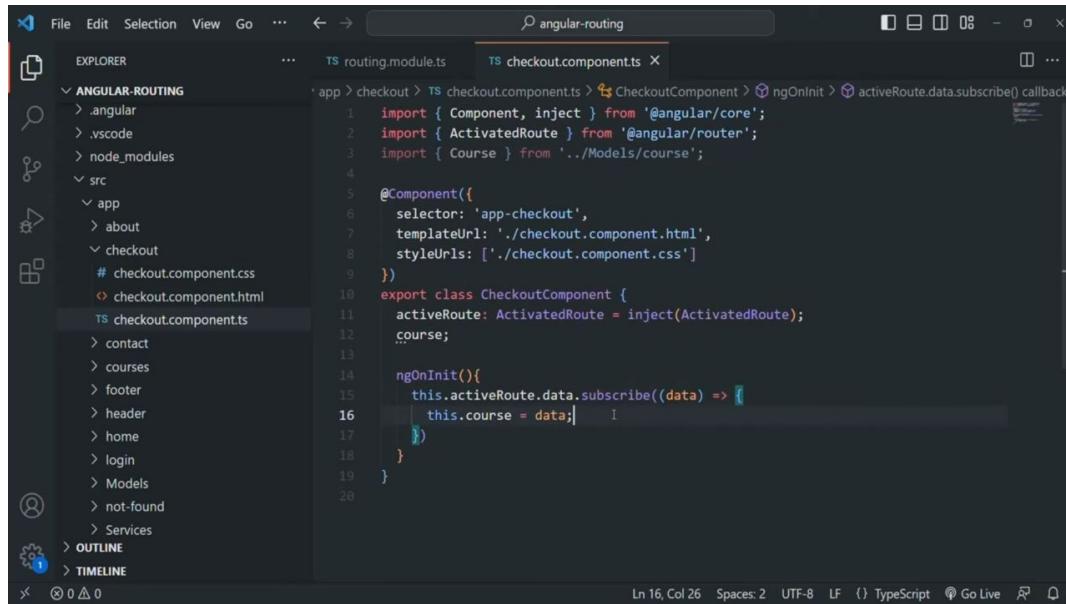
▶ 5:23:19



```
src > app > TS routing.module.ts > [o] routes > [o] children > [o] data > [o] price
14
15
16 ...
17 Routes = [
18
  { component: HomeComponent },
19  { name: 'home', component: HomeComponent },
20  { about: 'about', component: AboutComponent },
21  { contact: 'contact', component: ContactComponent, canDeactivate: [(comp: ContactComponent) =>
22    comp.message] },
23  { courses: 'courses', component: CoursesComponent, resolve: {courses: resolve} },
24  { course: 'course/:id', component: CourseDetailComponent },
25  { popular: 'popular', component: PopularComponent },
26  { checkout: 'checkout', component: CheckoutComponent, data: {name: 'Test Course', price: 399} }
27
28  { login: 'login', component: LoginComponent },
29  { error: '', component: NotFoundComponent },
30
31
32
33
34
35  RouterModule.forRoot(routes, {enableTracing: true})
36
37 ]
```

A new course, "Test Course," has been introduced with a price of \$399 and will be available for users to navigate to.

▶ 5:25:35



```
src > checkout > TS checkout.component.ts > [o] CheckoutComponent > [o] ngOnInit > [o] activeRoute.data.subscribe() callback
1 import { Component, inject } from '@angular/core';
2 import { ActivatedRoute } from '@angular/router';
3 import { Course } from '../Models/course';
4
5 @Component({
6   selector: 'app-checkout',
7   templateUrl: './checkout.component.html',
8   styleUrls: ['./checkout.component.css']
9 })
10 export class CheckoutComponent {
11   activeRoute: ActivatedRoute = inject(ActivatedRoute);
12   course;
13
14   ngOnInit(){
15     this.activeRoute.data.subscribe((data) => {
16       this.course = data;
17     });
18   }
19 }
```

▶ 5:28:44

```
<div>
  <label for="credit-card-num">Card Number</label>
  <span class="card-logos">
    <i class="fa fa-cc-visa" aria-hidden="true"></i>
    <i class="fa fa-cc-amex" aria-hidden="true"></i>
    <i class="fa fa-cc-mastercard" aria-hidden="true"></i>
    <i class="fa fa-cc-discover" aria-hidden="true"></i>
  </span>
<input id="credit-card-num" name="credit-card-num" placeholder="1111-2222-3333-4444" required type="text"/>
<div>
  <p class="expires">Expires on:</p>
  <div class="card-expiration">
    <label for="expiration-month">Month</label>
    <select id="expiration-month" name="expiration-month" required>
      <option value="">Month:</option>
```

- Developers are advised to remove the type in order to avoid errors and use string interpolation syntax for displaying course names in HTML files.

▶ 5:28:46

```
<button class="btn">
  <i class="fa fa-lock" aria-hidden="true"></i> Buy Now
</button>
</form>
</div>
<div class="right-side">
  <div class="receipt">
    <h2 class="receipt-heading">{{ course.name }}</h2>
    <div>
      <table class="table">
        <tr>
          <td>{{ course.proce }} x 1 Course</td>
          <td class="price">449.00 USD</td>
        </tr>
        <tr>
          <td>Discount</td>
          <td class="price">0.00 USD</td>
        </tr>
        <tr>
          <td>Subtotal</td>
```

- A new feature has been added to display the value of the name property and price in a software application.

▶ 5:29:22

The screenshot shows the Angular Routing code editor. The left sidebar displays the project structure under 'EXPLORER' with 'ANGULAR-ROUTING' expanded, showing files like 'checkout.component.html', 'courses.component.html', and 'courses.component.ts'. The main editor area shows the template for 'courses.component.html' (line numbers 11-34). The template uses an ngFor loop to iterate over 'AllCourses'. Each course item contains a 'course-card' div with an image, title, description, price, rating, and action buttons. The first button's routerLink is set to '/Courses/Checkout' with a state named 'course'.

```
<div class="course-card" *ngFor="let course of AllCourses">
  <div class="course-image">
    <img [src]="course.image">
  </div>
  <div class="course-title">
    {{ course.title }}
  </div>
  <div class="course-desc">
    {{ course.desc.substring(0, 120) }}
  </div>
  <div class="course-price-rating">
    <div class="course-price"><b>PRICE:</b> {{ course.price | currency }}</div>
    <div class="course-rating"><b>RATINGS:</b> {{ course.rating }}</div>
  </div>
  <div class="course-action-buttons">
    <button class="btn" routerLink="/Courses/Checkout" [state]="">CHECKOUT</button>
    <button class="btn" [routerLink]="/Courses/Course/">DETALLES</button>
  </div>
</div>
</div>
```

- Pass dynamic data with routes by assigning the current course object to the State Property in order to access it in the checkout component view.

▶ 5:32:30

This screenshot shows the same code editor as the previous one, but the 'checkout' button's routerLink has been modified to include the 'course' state variable. The state is defined as 'course' in the line above the button.

```
<button class="btn" routerLink="/Courses/Checkout" [state]="">CHECKOUT</button>
```

▶ 5:32:31

```
import { Component, inject } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { Course } from '../Models/course';

@Component({
  selector: 'app-checkout',
  templateUrl: './checkout.component.html',
  styleUrls: ['./checkout.component.css']
})
export class CheckoutComponent {
  activeRoute: ActivatedRoute = inject(ActivatedRoute);
  router: Router = inject(Router);
  course;

  ngOnInit() {
    this.activeRoute.data.subscribe((data) => {
      // this.course = data;
      // })
    this.course = this.router.getCurrentNavigation().extras.state;
  }
}
```

>Error in course type causes issues with website functionality, prompting changes to be made.

5:34:51

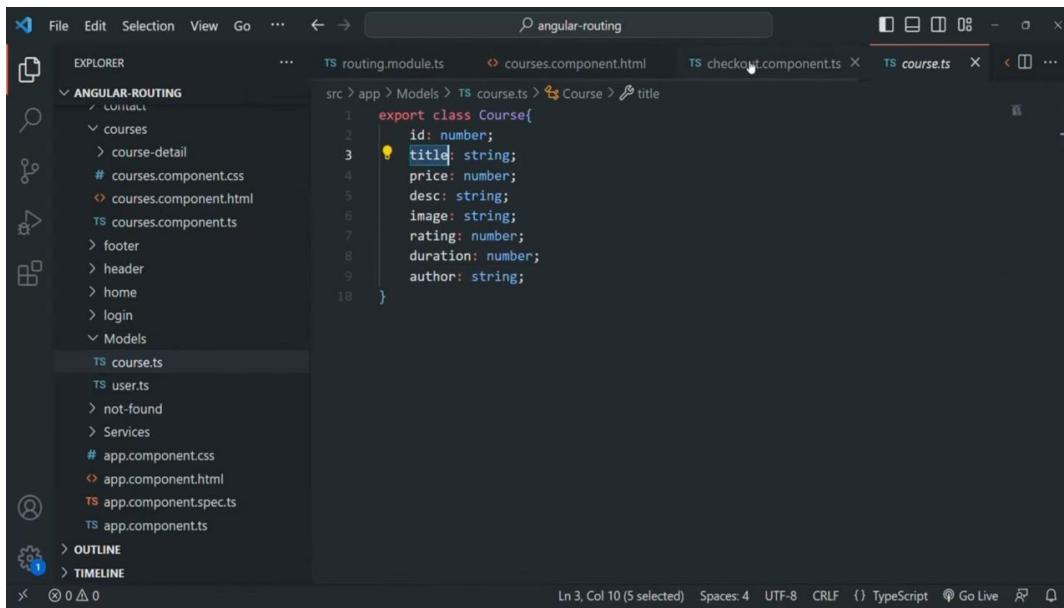
```
import { Component, inject } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { Course } from '../Models/course';

@Component({
  selector: 'app-checkout',
  templateUrl: './checkout.component.html',
  styleUrls: ['./checkout.component.css']
})
export class CheckoutComponent {
  activeRoute: ActivatedRoute = inject(ActivatedRoute);
  router: Router = inject(Router);
  course;

  ngOnInit() {
    this.activeRoute.data.subscribe((data) => {
      // this.course = data;
      // })
    this.course = this.router.getCurrentNavigation().extras.state;
  }
}
```

New React course now available with displayed price but missing name property.

5:36:48

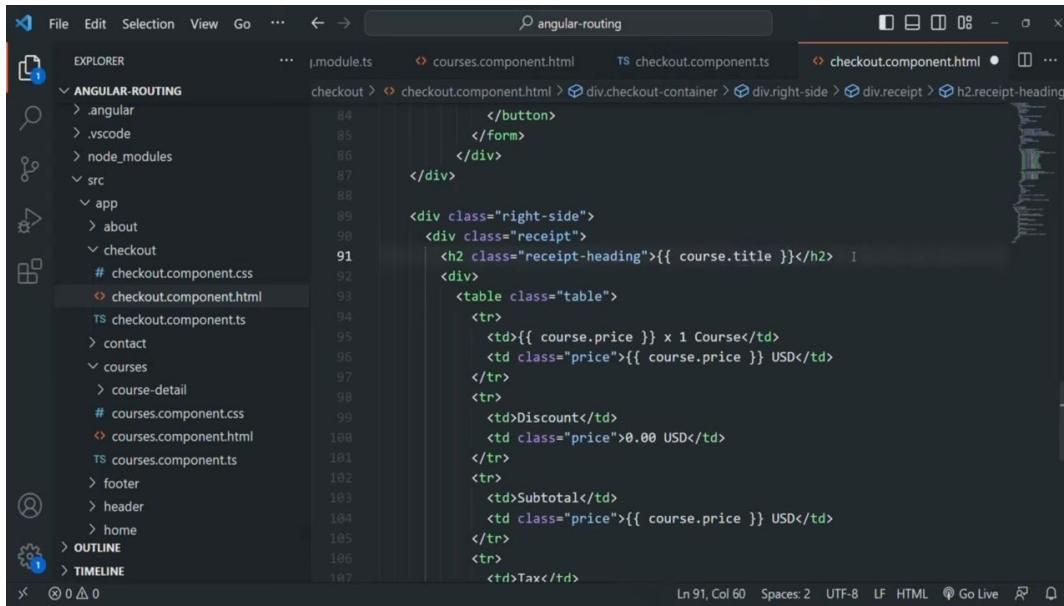


The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under the **ANGULAR-ROUTING** folder, including files like `courses.component.ts`, `course.ts`, and `checkout.component.ts`.
- Editor:** Displays the `course.ts` file with the following code:

```
1 export class Course{  
2     id: number;  
3     title: string;  
4     price: number;  
5     desc: string;  
6     image: string;  
7     rating: number;  
8     duration: number;  
9     author: string;  
10}
```
- Status Bar:** Shows "Ln 3, Col 10 (5 selected) Spaces: 4 UTF-8 CRLF () TypeScript Go Live" and a small preview icon.

▶ 5:36:57



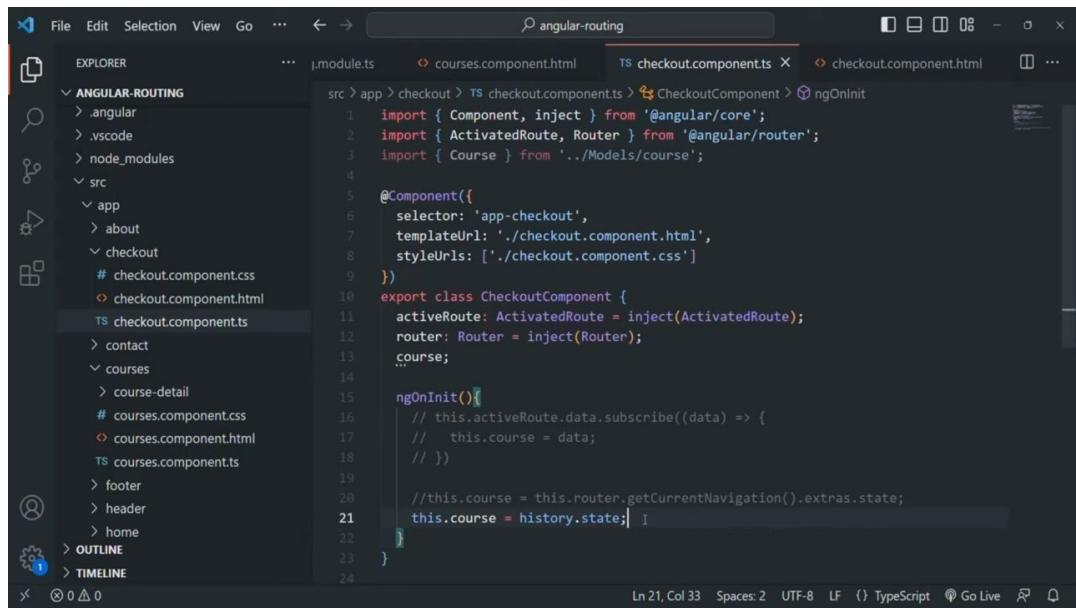
The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under the **ANGULAR-ROUTING** folder, including files like `checkout.component.ts` and `checkout.component.html`.
- Editor:** Displays the `checkout.component.html` file with the following code:

```
84             <button>  
85                 </button>  
86             </div>  
87         </div>  
88  
89         <div class="right-side">  
90             <div class="receipt">  
91                 <h2 class="receipt-heading">{{ course.title }}</h2>  
92             </div>  
93             <table class="table">  
94                 <tr>  
95                     <td>{{ course.price }} x 1 Course</td>  
96                     <td class="price">{{ course.price }} USD</td>  
97                 </tr>  
98                 <tr>  
99                     <td>Discount</td>  
100                    <td class="price">0.00 USD</td>  
101                </tr>  
102                <tr>  
103                    <td>Subtotal</td>  
104                    <td class="price">{{ course.price }} USD</td>  
105                </tr>  
106                <tr>  
107                    <td>Tax</td>
```
- Status Bar:** Shows "Ln 91, Col 60 Spaces: 2 UTF-8 LF HTML Go Live" and a small preview icon.

★ Checkout component.html has been updated to use the title property instead of the name property for a smoother user experience.

▶ 5:37:07



```
src > app > checkout > TS checkout.component.ts <-- CheckoutComponent > ngOnInit
1 import { Component, inject } from '@angular/core';
2 import { ActivatedRoute, Router } from '@angular/router';
3 import { Course } from '../Models/course';
4
5 @Component({
6   selector: 'app-checkout',
7   templateUrl: './checkout.component.html',
8   styleUrls: ['./checkout.component.css']
9 })
10 export class CheckoutComponent {
11   activeRoute: ActivatedRoute = inject(ActivatedRoute);
12   router: Router = inject(Router);
13   course;
14
15 ngOnInit() {
16   // this.activeRoute.data.subscribe((data) => {
17   //   this.course = data;
18   // })
19
20   //this.course = this.router.getCurrentNavigation().extras.state;
21   this.course = history.state; | I
22 }
23 }
```

Ln 21, Col 33 Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ⌂

★ Learn how to pass static and dynamic data to a route in Angular and master routing and Route guards with real world examples.

▶ 5:38:51