ROUTING

Implementing Routing in Angular

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.

Angular Routing

Definition: Routing in Angular allows navigation between different parts (views) of an application. It enables moving from one component view to another by associating URL paths with specific components.

Steps to Implement Angular Routing:

1. **Install Angular Router:**
   - When you create a new Angular project with the Angular CLI, the `@angular/router` module is automatically included.
   - You can verify its presence in the `package.json` under dependencies:

```json
"@angular/router": "version"
```

How to define Routes

- Create a new route using **Routes** array and define some route objects inside that array.

2. **Define Routes:**
   - Routes are defined as an array of objects, where each object specifies a path and the corresponding component.
   - Example:

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: 'courses', component: CoursesComponent }
];
```

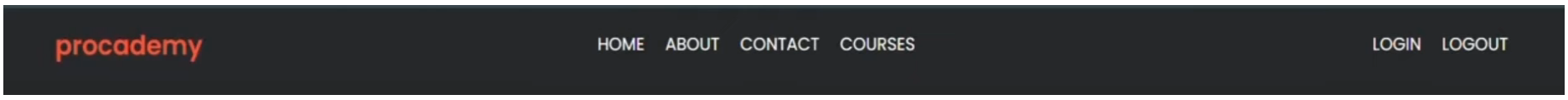
```
//DEFINE ROUTE
const routes: Routes = [
  {path: 'Home', component: HomeComponent},
  {path: 'About', component: AboutComponent},
  {path: 'Contact', component: ContactComponent},
  {path: 'Courses', component: CoursesComponent}
]
```

3. **Register the Routes:**
   - To register routes, you must import `RouterModule` and use its `forRoot()` method in the `imports` array of the `AppModule`.
   - Example:

```typescript
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

4. **Use `router-outlet` in Template**:
   - In the template (e.g., `app.component.html`), use the `router-outlet` directive to define where the routed component views should be rendered.

```html
<app-header></app-header>
<router-outlet></router-outlet>
<app-footer></app-footer>
```



Routing Example:

1. **App Component**:
   - The app component contains the layout with header and footer, and the dynamic content is rendered between them using the `router-outlet` directive.

```html
<app-header></app-header>
<router-outlet></router-outlet>
<app-footer></app-footer>
```

2. **Defining Routes**:

```typescript
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: 'courses', component: CoursesComponent },
  { path: '', redirectTo: '/home', pathMatch: 'full' } // Default route
];
```

3. **Navigation**:
   - When a user clicks on a link like `/home`, the `HomeComponent` view will be displayed between the header and footer.
   - Example:

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

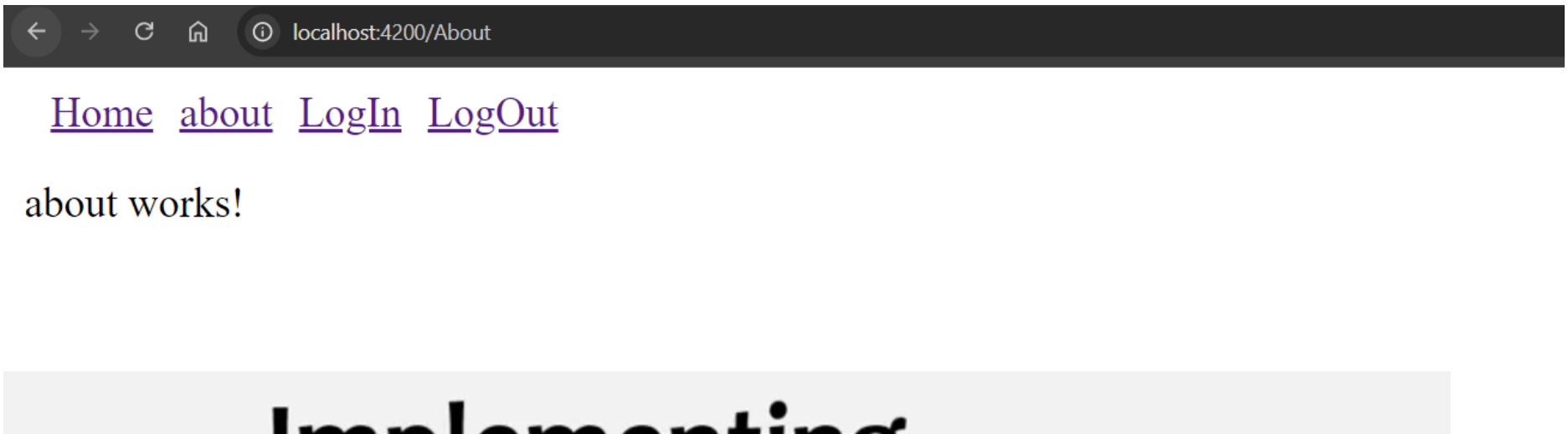
Redirecting to Default Route:

- You can configure a default route to display when no specific path is provided:

```typescript
{ path: '', redirectTo: '/home', pathMatch: 'full' }
```

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);`**



Implementing NotFound Route

Angular Routing with Wildcard Route

Definition:

In Angular, routing allows navigation between different views or components of an application. Wildcard routes are used to catch undefined routes and display a fallback view, typically a 404 error page.

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.

Key Concepts:

- **Routes**: Defined paths in the application, mapped to specific components.
- **Wildcard Route** (`**`): Matches any undefined URL that doesn't have a specific route defined.

Example:

1. **Routes Setup:**

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: '**', component: NotFoundComponent }  // Wildcard route
];
```

Here, `path: '**'` is a wildcard route that will match any path not already defined and display the `NotFoundComponent`.

2. **Not Found Component:**

html Copy code

```
<!-- not-found.component.html -->
<h1>404 Page Not Found</h1>
<p>The page you're looking for cannot be found. Please check the URL or try again late
```

3. **How It Works:**
   - If a user navigates to `/home`, the `HomeComponent` will render.
   - If the user navigates to an undefined route like `/services`, the wildcard route will activate, rendering the `NotFoundComponent` and displaying a "404 Page Not Found" message.

Important Notes:

- Wildcard routes must be **defined at the end** of the route list to prevent it from catching valid paths.
- Wildcard routes can match any path that doesn’t have a corresponding route, showing a fallback component.

Wildcard Route Example:

- URL: `rootURL/about` → Displays `AboutComponent`
- URL: `rootURL/non-existent` → Displays `NotFoundComponent`

error.component.html U

AngularRouting > src > app > error > error.component.html > ...

Go to component

```html
<div class="container">
  <h1>404 : Page Not Found</h1>
  <p>The page you are looking for is not found or you don't have access to it</p>
</div>
```

```ts
const routes: Routes = [
  { path: '', component: HomeComponent },
  // { path: '', redirectTo: 'Home', pathMatch: 'full' },
  { path: 'Home', component: HomeComponent },
  { path: 'About', component: AboutComponent },
  { path: 'Header', component: HeaderComponent },
  { path: '**', component: ErrorComponent }
]
```

Home about LogIn LogOut

404 : Page Not Found

The page you are looking for is not found or you don't have access to it

Configuring navigation Link for Route

Issue with Reloading:

- Clicking on links reloads the entire app.
- Default behavior: Sends a new request to the server and returns a new page.
- Undesirable as it restarts the app, leading to loss of application state.

header.component.html U

AngularRouting > src > app > header > header.component.html > ...

Go to component

```
<div class="container" style="display:flex">

  <div class="links">
    <a routerLink="/Home" class="margin">Home</a>
    <a [routerLink]="'/About'" class="margin">about</a>
    <a [routerLink]="['/Courses']" class="margin">Courses</a>
  </div>
```

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.

Angular Navigation and RouterLink Directive

1. Default HTML Navigation Links

- **Definition**: Traditional HTML uses anchor (`<a>`) tags with the `href` attribute to link between pages. This causes a full page reload.
- **Example**:

```
<a href="/home">Home</a>
<a href="/about">About</a>
```

Drawback: Using `href` causes the entire page to reload, sending new requests to the server every time a link is clicked. This results in reloading all necessary files (CSS, JavaScript bundles, etc.), which is inefficient and can lose the application's current state.

2. RouterLink Directive

- **Definition**: In Angular, the `RouterLink` directive is used for in-app navigation without reloading the entire page. It creates a single-page application (SPA) behavior by preventing full page reloads and dynamically switching between views.
- **Example**:

Copy code

```
<a routerLink="/home">Home</a>
<a routerLink="/about">About</a>
```

Advantages:

- Prevents full page reload.
- Only the necessary parts of the application are updated, improving performance.
- Maintains the application state.

3. **Preventing Default Anchor Behavior**

- **Definition**: By using `routerLink`, Angular prevents the default behavior of sending a new request to the server. It catches the click event and checks for a matching route in the application, navigating without reloading the page.
- **Key Concept**: **Single-Page Application (SPA)** behavior.

2. **RouterLink Syntax:**
   - Can be used without square brackets for a string value.
   - Alternatively, use square brackets for dynamic values or properties.

[routerLink]="'/about'"

- Array syntax for specifying multiple values, such as route parameters.

[routerLink]="['/about', 10]"

4. **RouterLink with Dynamic Binding**

- **Definition**: `routerLink` can be wrapped in square brackets to bind to a dynamic TypeScript expression. However, if using a string literal, it must be wrapped in single quotes.
- **Example**:

html Copy code

```
<a [routerLink]="'/about'">About</a>
```

5. **Performance Benefits**

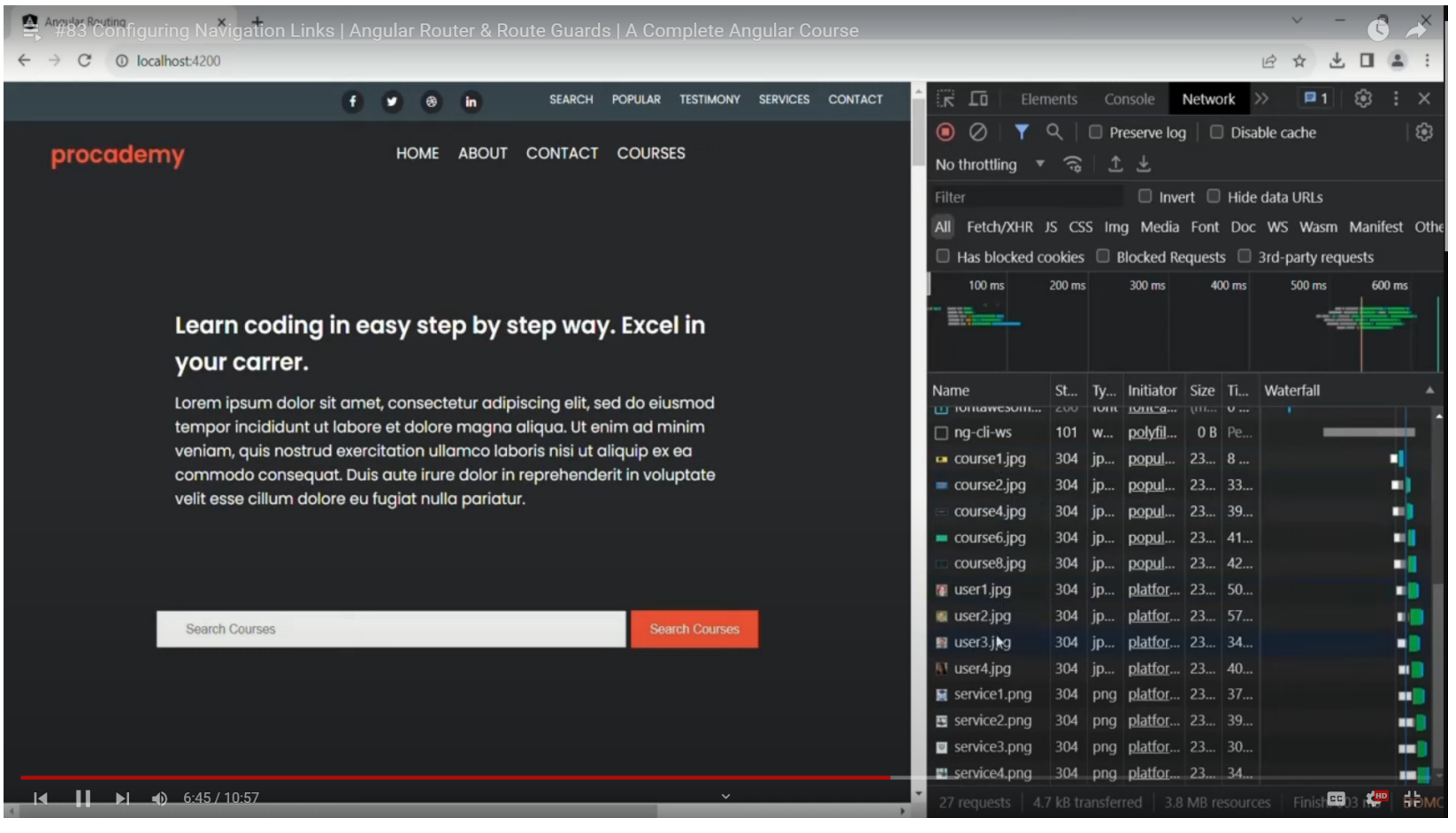
- **Definition**: When `routerLink` is used, only the required files for the current view (such as images) are downloaded, rather than reloading the entire application bundle. This improves the speed and efficiency of navigation within the application.
- **Example**: Navigating to different pages only triggers the download of the specific resources needed for that page, like an image or a component's CSS.

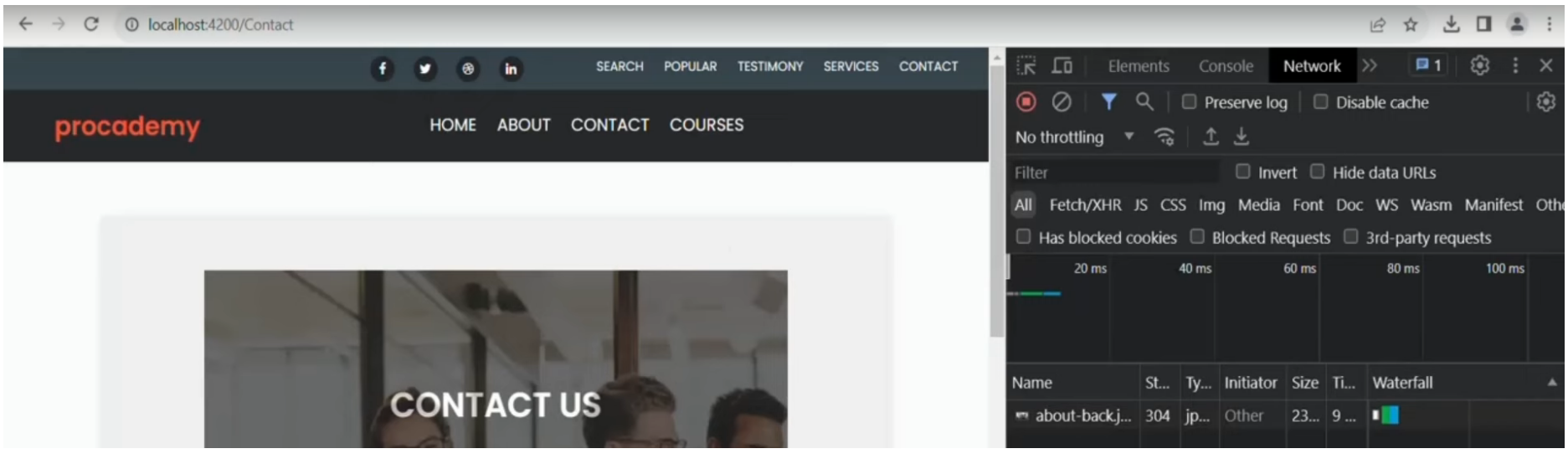
6. **Single-Page Application (SPA)**

- **Definition**: A single-page application dynamically updates the view without reloading the entire page. Angular achieves this behavior using `routerLink`, resulting in faster navigation and preserving the application state.
- **Key Concept**: **No page reload**, and **faster navigation** due to partial content updates.

7. **RouterLink vs. Href**

- **Difference**:
  - **Href**: Causes a full page reload, resulting in inefficiency and potential loss of the application state.
  - **RouterLink**: Prevents reloading, improving performance and maintaining the application's state.





Styling Active Router Link

4. **Styling Active Links:**
    - Current issue: Active link not visually distinguished in the navigation bar.
    - Desire: Styling the active link to inform the user about the current page.
5. **Upcoming Focus:**
    - Next lecture to cover the styling of active links in the navigation bar.
    - Goal: Provide a clear visual indication of the active link based on the current page.

RouterLinkActiveOptions Directive

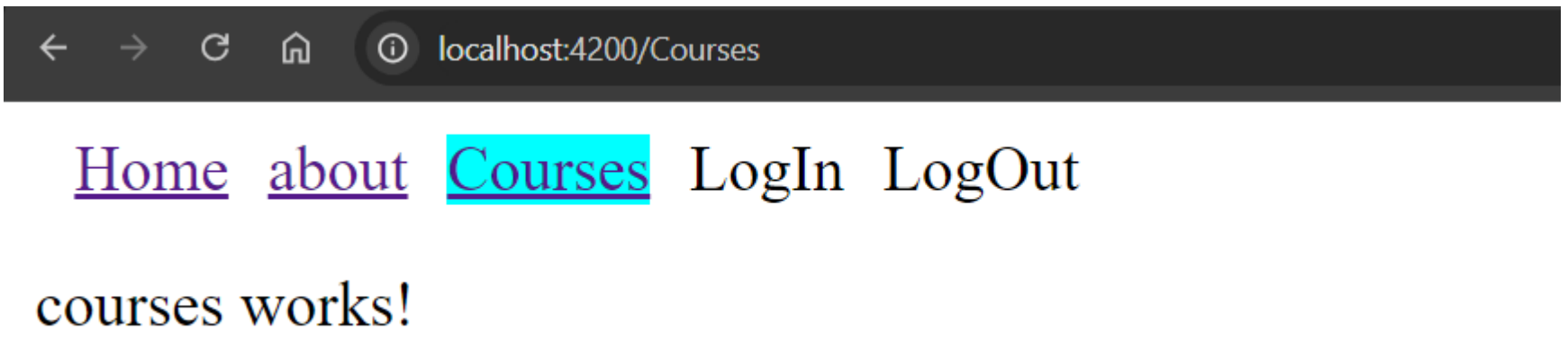
When a child route is active, then all the parent routes are also marked as active. In that case, **`routeLinkActive`** directive is applied to the active child route and all its parent routes.

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}"`

```
.Active {
    background-color: aqua;
}
```

```
<div class="links">

  <a routerLink="/Home" routerLinkActive="Active" [routerLinkActiveOptions]="{exact:true}
  " class="margin">Home</a>

  <a [routerLink]="'/About'" routerLinkActive="Active" class="margin">about</a>

  <a [routerLink]="['/Courses']" routerLinkActive="Active" class="margin">Courses</a>
</div>
```



Notes on Angular Routing: Styling Active Links

1. **Objective**: Style the active link in an Angular application to make it clear which page or view the user is currently on.
2. **Steps**:
    - **Step 1**: Create a CSS class in `styles.css`. Example:

```css
.active {
    font-size: larger;
    font-weight: bold;
}
```

    - **Step 2**: Use the `routerLinkActive` directive in your HTML to apply the CSS class.
        - Example:

```html
<a routerLink="/home" routerLinkActive="active">Home</a>
```

3. **Router Link Active Directive**:
    - The `routerLinkActive` directive dynamically adds/removes CSS classes from a link based on the active route.
    - Syntax:

```html
<a routerLink="/path" routerLinkActive="active">Link</a>
```

- If wrapped in square brackets, the directive expects a TypeScript expression, so wrap the CSS class name in single quotes.
  - Example:

```html
<a routerLink="/home" [routerLinkActive]="'active'">Home</a>
```

4. **Highlighting Active Links**:
   - When a route is active, its corresponding link will be styled with the `active` class.
   - Example behavior:
     - `/about`: The "About" link is highlighted.
     - `/contact`: The "Contact" link is highlighted.

5. **Issue with Parent Routes**:
   - If a child route is active (e.g., `/about`), its parent route (e.g., `/`) will also be marked as active. This results in both links being styled.
   - **Solution**: Use the `routerLinkActiveOptions` directive to ensure that only the exact route is styled.

```html
<a routerLink="/home" routerLinkActive="active" [routerLinkActiveOptions]="{ exact
```

6. **Fixing the Parent-Child Route Issue**:
   - Without `routerLinkActiveOptions`, both parent and child routes are marked as active.
   - By setting `exact: true` in the `routerLinkActiveOptions` directive, only the current route (not its parents) will be styled as active.

7. **Key Takeaway**:
   - The `routerLinkActiveOptions` directive helps avoid unwanted styling of parent routes when child routes are active.
   - Always ensure that the `exact` property is set to `true` for precise control over which link is highlighted.

Relative vs Absolute Route Path

Notes on **Relative** and **Absolute Route Paths** in Angular

Definitions:

1. **Absolute Route Path**:
   - **Definition**: An absolute route path starts with a `/` and is appended directly to the root URL, ignoring the currently active route.
   - **Usage**: When using a `/` before the path, it directs Angular to append the specified path to the root URL, no matter the current route.
   - **Example**:

```html
<a [routerLink]="/about">About</a>
```

   - Here, `/about` will append `about` directly to the root URL, e.g., `http://localhost:4200/about`.

2. **Relative Route Path**:
   - **Definition**: A relative route path does **not** begin with a `/` and is appended to the currently active route.
   - **Usage**: Without a `/`, the path is appended to the current URL or route.
   - **Example**:

```
<a [routerLink]="home">Home</a>
```

   - If the current active route is `/about`, the `home` path will be appended, resulting in `/about/home`.

Key Differences:

- **Absolute Path**: Appends to the root URL regardless of the current location.
- **Relative Path**: Appends to the currently active route.

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`

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`

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`

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`

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

Example Scenario:

In an Angular app, assume you are on the **About** page (`http://localhost:4200/about`):

1. **Absolute Path**:
   - Link:

     html — Copy code

     ```
     <a [routerLink]="/courses">Go to Courses</a>
     ```

   - Clicking the link will navigate to `http://localhost:4200/courses` .

2. **Relative Path**:
   - Link:

     html — Copy code

     ```
     <a [routerLink]="courses">Go to Courses</a>
     ```

   - Clicking this link will navigate to `http://localhost:4200/about/courses` .

Special Cases:

1. `./` **(Single Dot)**:
   - Indicates a relative path starting from the current route.
   - Example:

     html — Copy code

     ```
     <a [routerLink]="./home">Go Home</a>
     ```

   - Appends `home` to the current route.

2. `../` **(Double Dots)**:
   - Moves one level up in the current route before appending the new path.
   - Example:

```html
<a [routerLink]="../courses">Go to Courses</a>
```

     - If the current route is `/about/contact`, this will navigate to `/about/courses`.

Important Notes:

- If a **router link** is defined outside of the currently active route, even if the relative path is used, it will be appended to the root URL. This applies mainly to common components like headers, which are outside specific routes.

By understanding the difference between **relative** and **absolute** paths, you can control how navigation behaves in different contexts within your Angular application.

Navigating between Routes Programmatically

Notes: Navigating Between Routes Programmatically in Angular

1. Using `routerLink` Directive

- `routerLink` is a directive that binds a clickable HTML element to a route.
- Example: `<button [routerLink]="['/courses']">Go to Courses</button>`
- When the user clicks the element, Angular's router navigates to the associated route.

2. Navigating Programmatically

- Instead of using `routerLink`, you can navigate between routes programmatically using the `Router` class.
- Steps:
  1. Remove `routerLink` from the HTML element.
  2. Add a `(click)` event binding and call a method from the component class.

```html
<button (click)="navigateToCourses()">Go to Courses</button>
```

3. Injecting the `Router` Service

- To navigate programmatically, inject the `Router` service into your component class.
- Example:

```typescript
import { Router } from '@angular/router';
import { inject } from '@angular/core';

export class PopularComponent {
  router = inject(Router);

  navigateToCourses() {
    this.router.navigate(['/courses']);
  }
}
```

- `router.navigate()` accepts an array of route segments to construct the path.

4. Using `navigate()` Method

- `navigate()` method takes an array of path segments.
- Example:

```typescript
this.router.navigate(['/books', 'author', '101']);
```

- This navigates to `/books/author/101`.

5. Using `navigateByUrl()` Method

- `navigateByUrl()` method accepts a string representing the complete URL.
- Example:

```typescript
this.router.navigateByUrl('/courses');
```

- This navigates directly to the `/courses` page.

6. **Absolute vs Relative Paths**

- By default, paths passed to `navigate()` or `navigateByUrl()` are **absolute**.
- To use **relative** paths, you need to set the `relativeTo` option.

```
import { ActivatedRoute } from '@angular/router';
activeRoute = inject(ActivatedRoute);

this.router.navigate(['courses'], { relativeTo: this.activeRoute });
```

- This appends the `courses` segment to the currently active route.

7. **Example Scenario**

- Navigate programmatically when a button is clicked and append a relative path.

typescript Copy code

```
this.router.navigate(['courses'], { relativeTo: this.activeRoute });
```

- When the button is clicked, the `courses` path is appended to the current route.

This is the summary of programmatically navigating between routes in Angular using the `Router` service.

Working with Route Parameters

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
```

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
```

Notes on Route Parameters in Angular

Definition:

- A route parameter is a dynamic segment of a URL that can change, allowing the passing of extra information to a given route.

Example:

- URL Structure:
  - `/books/:id` (where `:id` is a route parameter)
  - `/bookauthor/:author` (where `:author` is a route parameter)

Defining Route Parameters:

- Use a colon (`:`) before the parameter name in the route definition.
  - Example:

javascript Copy code

```javascript
{ path: 'books/:id', component: BookDetailComponent }
```

Dynamic Routing Example:

- Consider a course application where users can view course details:
  - URL: `/courses/course/:id`
    - The `:id` can change based on which course detail is being viewed.
- To define this route in Angular:

javascript Copy code

```javascript
{ path: 'courses/course/:id', component: CourseDetailComponent }
```

Accessing Route Parameters:

1. **Using ActivatedRoute**:
   - Inject `ActivatedRoute` in the component:

```javascript
constructor(private activatedRoute: ActivatedRoute) {}
```

2. **Reading Parameter**:
   - Use `snapshot` to get the parameter:

```javascript
const courseId = this.activatedRoute.snapshot.params['id'];
```

   - Alternatively, use `paramMap` (recommended):

```javascript
this.activatedRoute.paramMap.subscribe(params => {
  const courseId = +params.get('id'); // Convert to number
});
```

Example in a Component:

- In `CourseDetailComponent`:

```javascript
ngOnInit() {
  this.activatedRoute.paramMap.subscribe(params => {
    const courseId = +params.get('id'); // Get course ID
    this.selectedCourse = this.courseService.getCourseById(courseId); // Filter course
  });
}
```

Router Link Usage:

- Use `routerLink` directive in HTML:

```html
<a [routerLink]="['/courses/course', course.id]">Details</a>
```

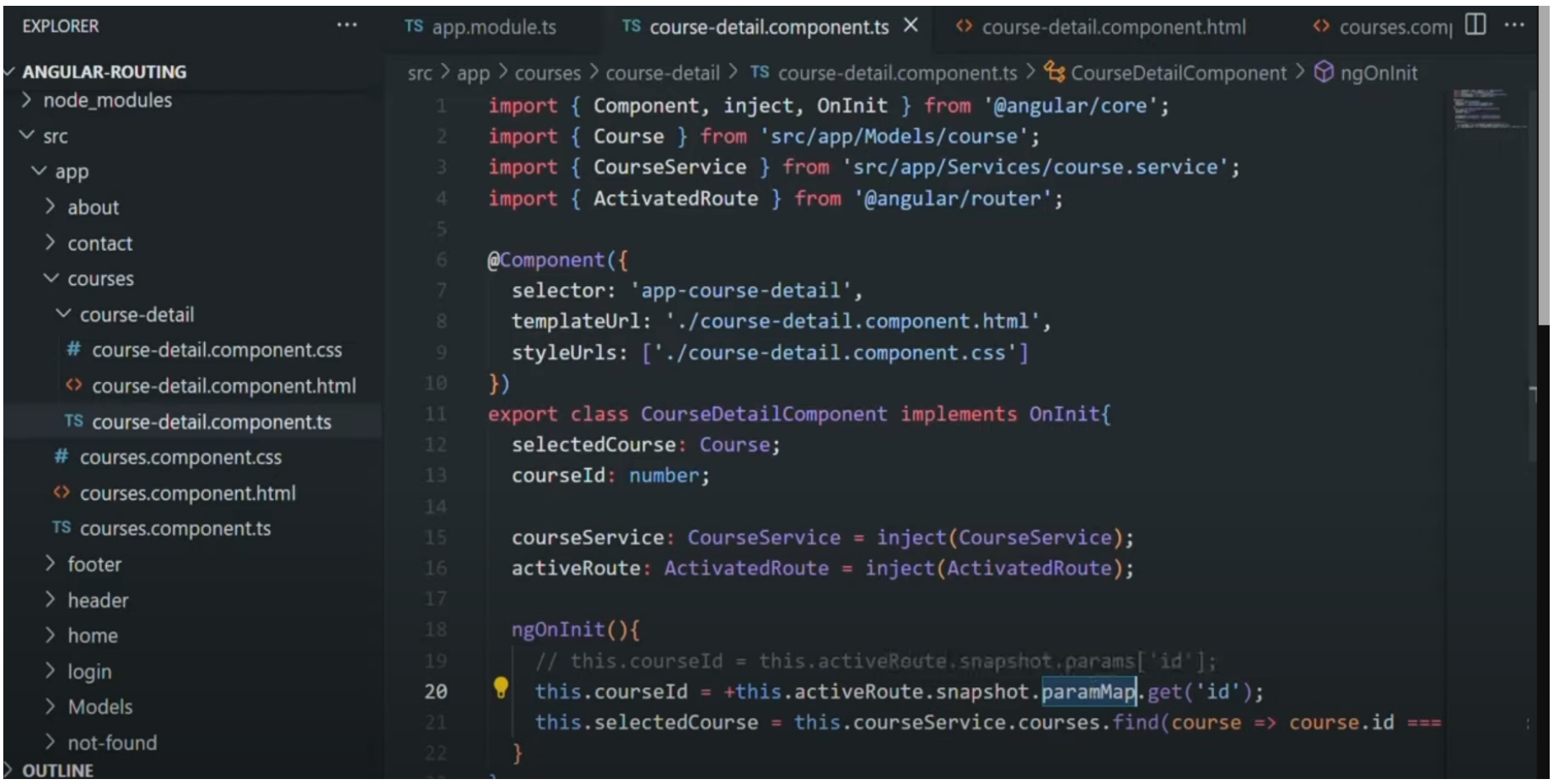
- Ensure paths are absolute by starting with a `/`.

Summary

- Route parameters are essential for dynamic routing in Angular applications, allowing the retrieval and display of variable data based on user interactions. Use `ActivatedRoute` to access these parameters and render appropriate views based on the dynamic segments in the URL.

```
{path: 'Courses/Course/:id/:name' }
{path: '**', component: NotFoundComponent},
```

```
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'];
    console.log(this.courseId);
  }
}
```



EXAMPLE :

Setting the parameter for route

```ts
const routes: Routes = [
  { path: '', component: HomeComponent },
  // { path: '', redirectTo: 'Home', pathMatch: 'full' },
  { path: 'Home', component: HomeComponent },
  { path: 'About', component: AboutComponent },
  { path: 'Courses', component: CoursesComponent },
  { path: 'Courses/:id', component: CoursesComponent },
  { path: 'Header', component: HeaderComponent },
  { path: '**', component: ErrorComponent }
]
```

```ts
})
export class CoursesComponent implements OnInit {
  courseId: number | null = null;

  active: ActivatedRoute = inject(ActivatedRoute);


  ngOnInit() {
    // this.courseId = this.active.snapshot.params['id'];
    const idParam = this.active.snapshot.paramMap.get('id');
    this.courseId = idParam ? +idParam : null;
    console.log(this.courseId);
  }
```

The error you are encountering stems from strict type checking in TypeScript. Here are the fixes:

1. **Property Initialization Error (TS2564)**:
   - TypeScript is complaining because `courseId` is not initialized, and you haven't marked it as optional. We can fix this by either assigning a default value or marking the property as optional (`undefined`).
2. **Type Assignment Error (TS2322)**:
   - The method `paramMap.get('id')` returns a `string | null`, but `courseId` expects a `number`. To fix this, you need to safely convert the value to a number, and handle the case when `null` is returned.

```ts
courseId: number | null = null;  // Assign null initially, or mark optional with `?`


active: ActivatedRoute = inject(ActivatedRoute);

ngOnInit() {
  // Fetch the courseId from the route params
  const courseIdParam = this.active.snapshot.paramMap.get('id');
  this.courseId = courseIdParam ? +courseIdParam : null;  // Safely convert to number
  console.log(this.courseId);
```

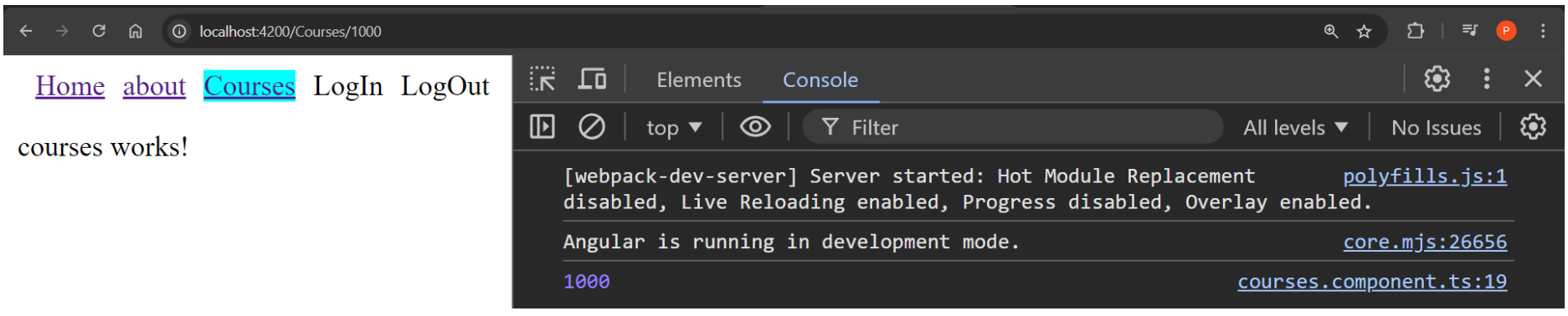
```
// Optionally listen to future route changes
this.active.paramMap.subscribe(params => {
  const courseIdParam = params.get('id');
  this.courseId = courseIdParam ? +courseIdParam : null;
  console.log('Updated courseId:', this.courseId);
});
```

Key Changes:

1. `courseId` **Initialization**:
   - Now initialized as `number | null`. You can also mark it as optional (`courseId?: number`) if that fits your use case.
2. **Type Conversion**:
   - `paramMap.get('id')` returns a `string | null`. It's converted to a number using `+` and checked for `null`. If `null`, we assign `null` to `courseId` instead of `undefined`.

3. **Handling Route Changes**:
   - Subscribing to `paramMap` ensures the component can handle route changes dynamically, updating `courseId` accordingly.

This solution should resolve both errors and allow the app to handle the route parameter correctly.



Using Observable to Retrieve Route Parameter

Route Parameters in Angular

1. **Snapshot Property**

- **Definition**: The snapshot property retrieves the initial value of route parameters when a component is loaded.
- **Example**:

```javascript
const id = this.route.snapshot.paramMap.get('id'); // Retrieves the initial 'id' param
```

2. **Limitations of Snapshot Property**

- **Definition**: The snapshot property does not update if the route parameter changes after the initial load.
- **Example**: If the route changes to `/courses/4`, calling `snapshot.paramMap.get('id')` will still return the old value if not reloaded.

3. **Router Link Property**

- **Definition**: A directive that allows navigation to a specific route.
- **Example**:

```html
<a [routerLink]="['/courses', selectedCourse.id - 1]">Previous Course</a>
<a [routerLink]="['/courses', selectedCourse.id + 1]">Next Course</a>
```

4. **Observable with Params**

- **Definition**: To dynamically update route parameters, you can subscribe to an observable.
- **Example**:

```javascript
this.route.paramMap.subscribe(params => {
    const id = +params.get('id'); // Using '+' to convert string to number.
});
```

5. **ParamMap vs. Params**

- **Definition**: `paramMap` is the recommended observable for reading route parameters, while `params` is deprecated.
- **Example**:

javascript Copy code

```
this.route.paramMap.subscribe(paramMap => {
    const id = +paramMap.get('id'); // Use 'get' method for retrieving values.
});
```

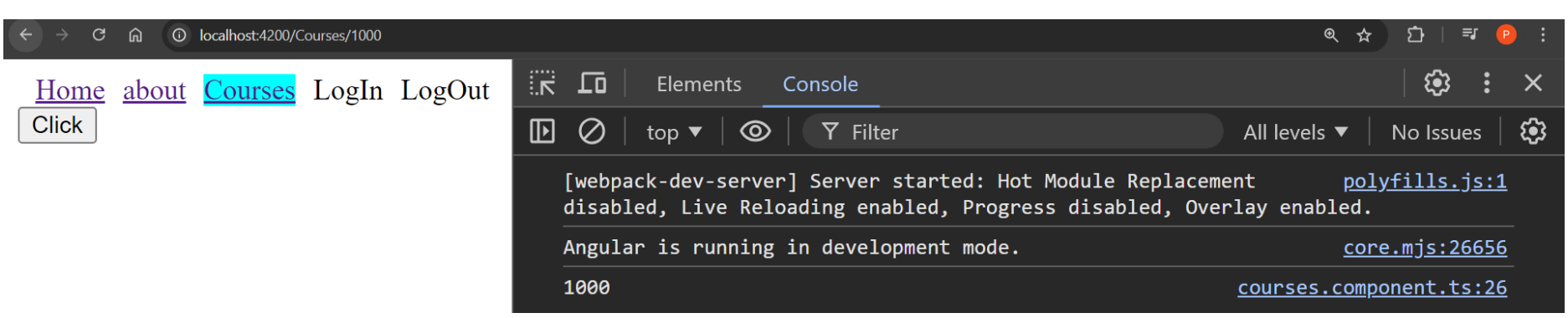
The paramMap Observable

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.

6. **Lifecycle Hooks**

- **Definition**: Lifecycle hooks allow you to manage component behavior at different stages.
- **Example**:
    - **ngOnInit**: Initializes the component.

```
ngOnInit() {
    this.route.paramMap.subscribe(params => {
        this.courseId = +params.get('id');
    });
}
```



```ts
export class CoursesComponent implements OnInit {

  courseId?: number;
  active: ActivatedRoute = inject(ActivatedRoute);


  ngOnInit() {
    // this.courseId = this.active.snapshot.params['id'];
    // const idParam = this.active.snapshot.paramMap.get('id');
    // this.courseId = idParam ? +idParam : null;
    this.active.params.subscribe((val) => {
      this.courseId = val['id'];
    })
    console.log(this.courseId);
  }
```

- **ngOnDestroy**: Cleans up subscriptions.

javascript Copy code

```ts
ngOnDestroy() {
    this.paramMapObs.unsubscribe(); // Unsubscribe to avoid memory leaks.
}
```

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.

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

```ts
export class CoursesComponent implements OnInit {

  courseId: number | null = null;
  active: ActivatedRoute = inject(ActivatedRoute);


  ngOnInit() {
    this.active.paramMap.subscribe((val) => {
      const CourseId = val.get('id');
      this.courseId = CourseId ? +CourseId : null;

    })
    console.log(this.courseId);
}
```

```ts
const CourseId = val.get('id');
this.courseId = CourseId ? +CourseId : null;
```

1. **Context**: These lines are within the `ngOnInit()` method of the `CoursesComponent` class. This method is called when the component is initialized, and it subscribes to the `paramMap` observable from the `ActivatedRoute`.

2. `val.get('id')`:
    - **What it does**: This line retrieves the value of the route parameter named `'id'` from the `paramMap` object, which contains all the route parameters as key-value pairs.
    - **Result**: `CourseId` will hold the value of the `id` parameter as a string. If the `id` is not present in the route, `CourseId` will be `null`.

3. `CourseId ? +CourseId : null`:
    - **Ternary Operator**: This line uses a conditional (ternary) operator to check if `CourseId` has a truthy value (i.e., it exists and is not `null` or an empty string).
    - `+CourseId`: If `CourseId` is truthy, it converts the string to a number using the unary plus operator (`+`). This is a shorthand way to parse the string into a number.
    - `null`: If `CourseId` is falsy (meaning it is `null` or an empty string), `this.courseId` is set to `null`.

4. **Final Assignment**: The result of the ternary operation is assigned to `this.courseId`. Thus, `this.courseId` will either hold the numeric value of the route parameter (if it exists) or `null`.

Complete Flow in `ngOnInit()`

- When the component is initialized, it subscribes to the `paramMap` observable.
- Each time the route parameters change, the callback function is triggered, and `val` holds the new parameters.
- The `id` parameter is fetched, and based on its existence, `this.courseId` is either set to its numeric value or `null`.
- Finally, the current value of `this.courseId` is logged to the console.

Example

- **Route**: `/courses/5`
- **Execution**:
    - `val.get('id')` retrieves `'5'`.
    - `this.courseId` is set to `5` (as a number).

- **Route**: `/courses/`
- **Execution**:
    - `val.get('id')` retrieves `null`.
    - `this.courseId` is set to `null`.

This approach ensures that the component reacts dynamically to route changes while maintaining proper typing for `courseId`.

Using Query String in Route

Query parameters are optional parameters that you pass to a route. These query parameters are added at the end of the URL separated by "?"

```
localhost:4200/Products?id=12345&&Name=iphone
```

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
```

In Angular, query parameters allow you to pass additional data through the URL, which can be accessed by the Angular components. They are typically used for filtering, pagination, or passing small amounts of data.

1. Setting Query Parameters

You can navigate with query parameters using the `Router.navigate()` method or directly in the template with the `[routerLink]` directive.

Example: Navigating with Query Parameters

Example: Navigating with Query Parameters

typescript — Copy code

```typescript
// In the component's TypeScript file
this.router.navigate(['/items'], { queryParams: { page: 2, sort: 'asc' } });
```

Here, you're navigating to the `/items` route and appending query parameters like `?page=2&sort=asc` to the URL.

2. Reading Query Parameters

You can read query parameters in a component using Angular's `ActivatedRoute` service.

Example: Reading Query Parameters

```typescript
import { ActivatedRoute } from '@angular/router';

constructor(private route: ActivatedRoute) {}

ngOnInit() {
  this.route.queryParams.subscribe(params => {
    const page = params['page'];  // e.g., page = 2
    const sort = params['sort'];  // e.g., sort = 'asc'
  });
}
```

3. Example of Full Usage

3.1. Routing Setup (app-routing.module.ts)

typescript Copy code

```typescript
const routes: Routes = [
  { path: 'items', component: ItemsComponent },
];
```

3.2. Component Navigation (items.component.ts)

typescript Copy code

```typescript
import { Router } from '@angular/router';

constructor(private router: Router) {}

navigateWithParams() {
  this.router.navigate(['/items'], { queryParams: { page: 1, sort: 'desc' } });
}
```

3.3. Reading Parameters (items.component.ts)

typescript Copy code

```typescript
import { ActivatedRoute } from '@angular/router';

constructor(private route: ActivatedRoute) {}

ngOnInit() {
  this.route.queryParams.subscribe(params => {
    console.log('Page:', params['page']);  // Output: 1
    console.log('Sort:', params['sort']);  // Output: 'desc'
  });
}
```

4. In Template with `[routerLink]`

You can also use query parameters in your HTML templates using the `[routerLink]` directive.

html Copy code

```html
<a [routerLink]="['/items']" [queryParams]="{ page: 3, sort: 'asc' }">Go to Page 3</a>
```

In this example, clicking the link will navigate to `/items?page=3&sort=asc`.

Let me know if you need more details on any part!

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

  activeRoute: ActivatedRoute = inject(ActivatedRoute);

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

```
;="home-page-banner-search">
:ype="text" class="search-input" placeholder="Search Courses" #searchVar>
 class="search-button" [routerLink]="'/Courses?search='+ (searchVar.target.value)"
```

```
h">
-input" placeholder="Search Courses" #searchVar>
outerLink]="'/Courses'" [queryParams]="{search: searchVar.value}">Search Courses<
```

Using Fragments in Route

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`

Fragments in Angular: Definition and Example

Definition:

A **fragment** in Angular routing is a part of a URL that points to a specific section of an HTML page. It is used to jump to a content section that has a corresponding `id` in the HTML. The fragment is appended to the URL after a pound/hash sign (`#`).

How Fragments Work:

- A fragment is specified in the URL after the route.
- For example, the URL `/home#services` includes the route `/home` and the fragment `#services`.
- When this URL is used, it scrolls to the section of the page with the `id="services"`.

Example:

1. **HTML Structure with IDs** (Target Sections):

html Copy code

```
<section id="search">Search Section</section>
<section id="courses">Popular Courses Section</section>
<section id="testimony">Testimony Section</section>
<section id="services">Services Section</section>
<section id="contact">Contact Section</section>
```

2. **Navigation Links with Fragments** (Fragment Links):

```html
<a [routerLink]="['/home']" fragment="search">Search</a>
<a [routerLink]="['/home']" fragment="courses">Popular Courses</a>
<a [routerLink]="['/home']" fragment="testimony">Testimony</a>
<a [routerLink]="['/home']" fragment="services">Services</a>
<a [routerLink]="['/home']" fragment="contact">Contact</a>
```

- Each link points to the `/home` route with a specific fragment like `#search`, `#courses`, etc.

3. **Reading the Fragment Value** (Using `ActivatedRoute`):

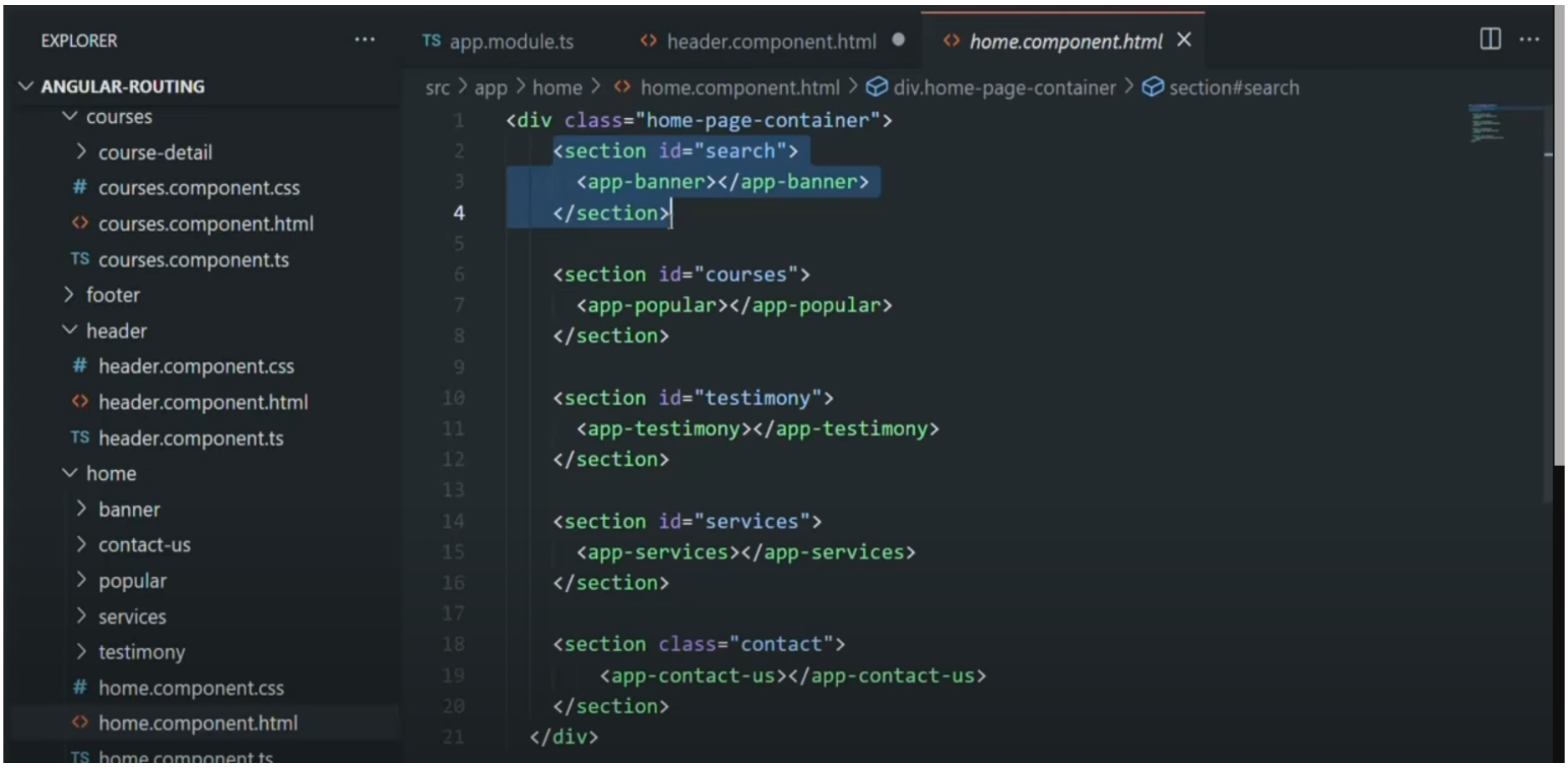
```typescript
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html'
})
export class HomeComponent implements OnInit {
  constructor(private activatedRoute: ActivatedRoute) {}

  ngOnInit(): void {
    this.activatedRoute.fragment.subscribe(fragment => {
      this.jumpToSection(fragment);
    });
  }

  jumpToSection(section: string): void {
    const element = document.getElementById(section);
    if (element) {
      element.scrollIntoView({ behavior: 'smooth' });
    }
  }
}
```

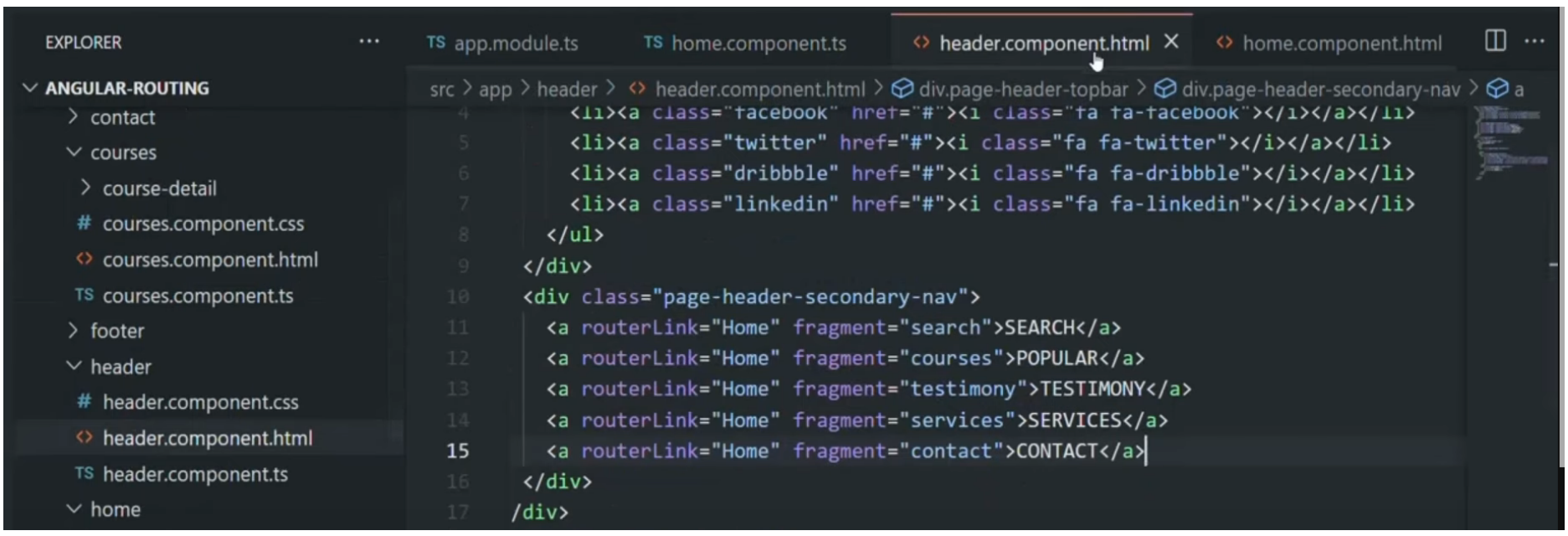
4. **How It Works:**
   - When a user clicks a navigation link, the URL will change to include the fragment (e.g., `/home#services`).
   - The `fragment` observable is used to read the fragment value from the URL.
   - The `jumpToSection` method scrolls smoothly to the corresponding section using `document.getElementById()` and `scrollIntoView()`.

Key Points:

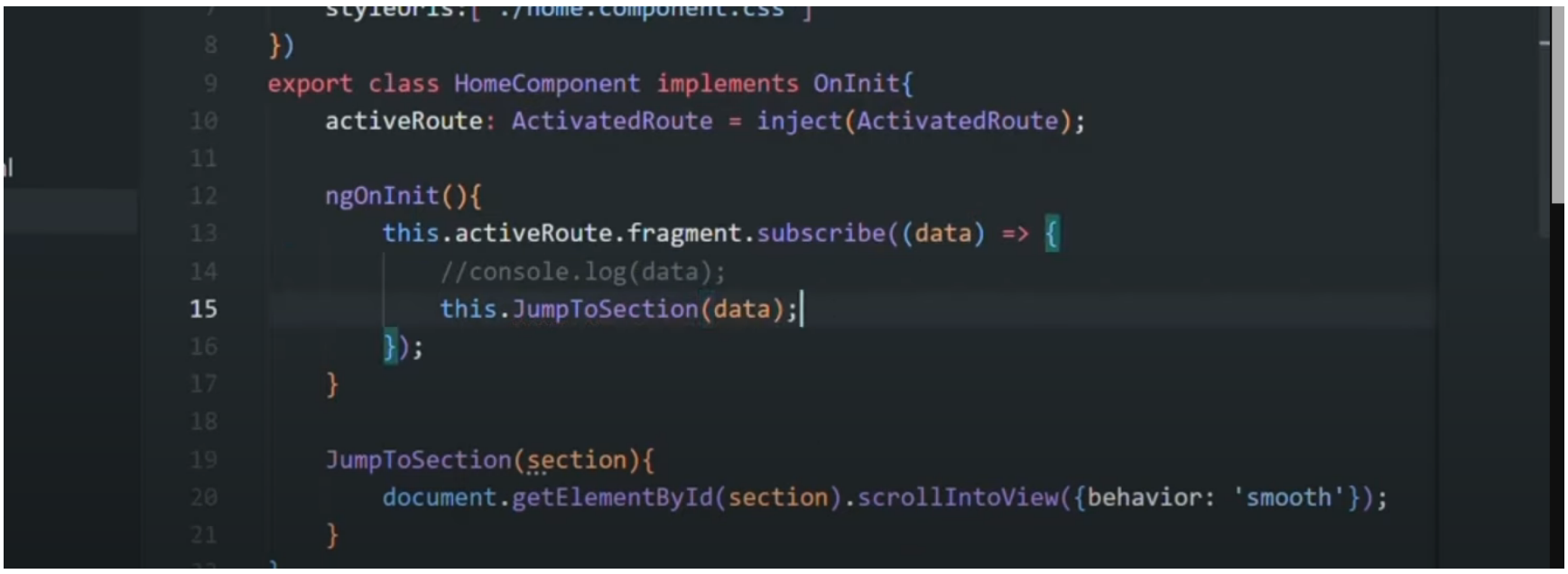
- **Fragment** is defined using the `[routerLink]` directive with the `fragment` attribute.
- The **fragment** is read using the `ActivatedRoute.fragment` observable.
- **Smooth scrolling** can be implemented with the `scrollIntoView()` method.

This approach makes it easy to navigate to specific sections of a page with smooth scrolling behavior in Angular applications.





The Fragment value which we are receiving is an id for the section



Working with Child Routes

```
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},
  // {path: 'Courses/Course/:id', component: CourseDetailComponent},
  {path: 'Courses', children: [
    {path: 'Course/:id', component: CourseDetailComponent}
  ]},
  {path: '**', component: NotFoundComponent},
```

Child Routes in Angular: Definition and Example

Definition:

A **child route** (or **nested route**) in Angular is a route that is defined within another route. It allows for a hierarchical organization of routes, making it easier to manage related components and views.

How to Create Child Routes:

1. Define a parent route.
2. Use the `children` property to specify an array of child routes.

Example:

1. **Parent Route Setup:**
   - Assume we have a parent route for displaying courses:

typescript Copy code

```typescript
const routes: Routes = [
  { path: 'courses', component: CoursesComponent, children: [
    // Child routes will be defined here
  ]}
];
```

2. **Defining Child Routes:**

   - Here, we add child routes for course details and a popular courses view:

```
const routes: Routes = [
  {
    path: 'courses',
    component: CoursesComponent,
    children: [
      { path: 'c/:id', component: CourseDetailComponent },  // Child route for course
      { path: 'popular', component: PopularComponent }      // Another child route for
    ]
  }
];
```

3. **Accessing Child Routes:**

   - When a user clicks a "Detail" button for a specific course, the URL will change to `/courses/c/{id}`, displaying the details for that course.

4. **Routing Example:**

   - **Navigate to Course Details:**

     - Clicking the detail button for the Node.js course might change the URL to `/courses/c/1`, where `1` is the ID of the course.

   - **Navigate to Popular Courses:**

     - Typing `/courses/popular` in the address bar would display the popular courses component.

5. **Final URL Structure:**

   - The final URLs for the routes could be:

     - `/courses/c/:id` for course details
     - `/courses/popular` for the popular courses view

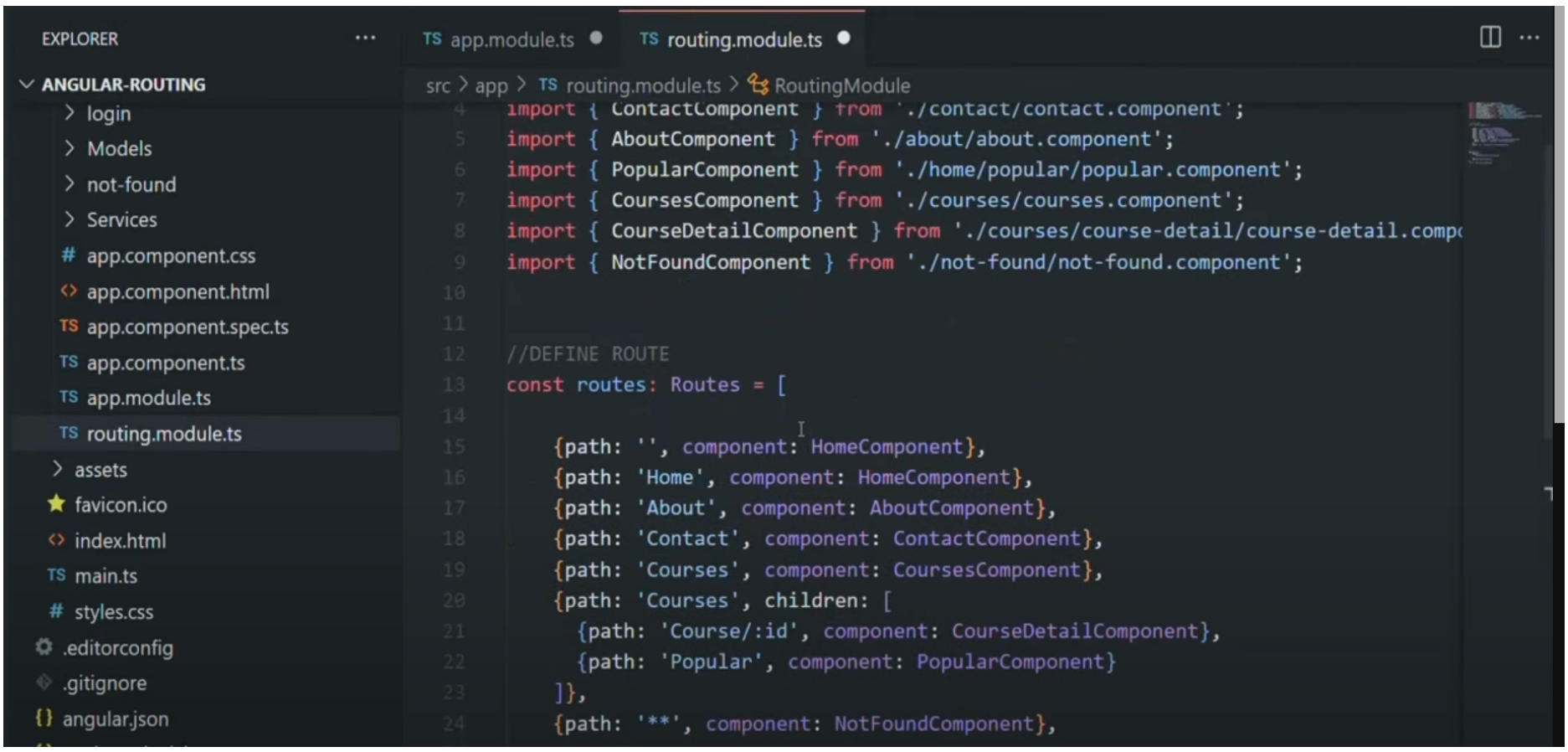
Key Points:

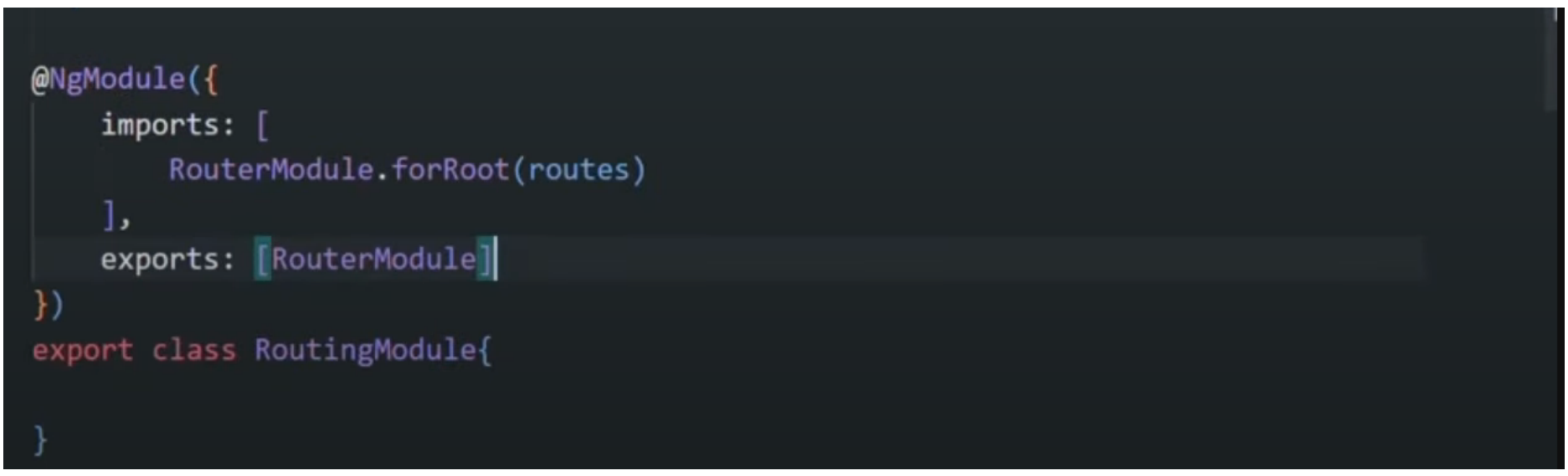
- **Parent Component:** The `CoursesComponent` acts as the parent component for the child routes.
- **Child Components:** The `CourseDetailComponent` and `PopularComponent` are the child components that render based on the specified child routes.
- **Multiple Child Routes:** You can define multiple child routes within the `children` array for better organization and navigation.

This hierarchical routing structure enhances the maintainability of Angular applications by clearly defining relationships between components and their routes.

Creating a Route Module File

Separate Routing Module





Just import routing Module in App modules imports Array

```
    NotFoundComponent
  ],
  imports: [
    BrowserModule,
    RoutingModule
  ],
  providers: [ServicesService, CourseService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Creating a Separate Routing Module in Angular

Overview:

In Angular applications, it's often beneficial to define routes in a separate file rather than in the main app module. This approach improves organization and maintainability, especially in larger applications with numerous routes.

Steps to Create a Routing Module:

1. **Create a New File:**
   - Inside the `app` folder, create a new file named `routing.module.ts`.
   - The file name convention should include "module" and have a `.ts` extension.
2. **Define the Routing Module:**
   - In `routing.module.ts`, create and export a class decorated with `@NgModule`.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  // Define your routes here
];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class RoutingModule {}
```

3. **Define Routes:**
    - Inside the `routes` array, define the application's routes as needed.
    - Example:

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: 'courses', component: CoursesComponent }
];
```

4. **Import the Routing Module in App Module:**
    - In your main app module (`app.module.ts`), import the `RoutingModule` .

```
import { RoutingModule } from './routing.module';

@NgModule({
  declarations: [
    // Your components
  ],
  imports: [
    BrowserModule,
    RoutingModule  // Add the routing module here
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

5. **Remove Routes from App Module:**
    - If routes were previously defined in `app.module.ts` , remove them to avoid duplication.
6. **Verify Functionality:**
    - After implementing the changes, run your application to ensure that routing still works as expected.
    - Check all defined routes (home, about, contact, courses) to confirm they display the correct components.

Benefits:

- **Improved Organization:** Separating routes makes the main module cleaner and easier to manage.
- **Scalability:** In larger applications with many routes, a dedicated routing module is essential for maintainability.
- **Reusability:** If needed, the routing module can be reused or imported into other modules in your application.

What is a Route Guard

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 use 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 use 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.**

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.

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

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 Route Guards Notes

Overview of Route Guards

- **Purpose**: Control user navigation to and from routes based on specific conditions.
- **Common Scenarios**:
  - Restricting access to unauthorized users.
  - Confirming unsaved changes before navigating away.
  - Validating route parameters.
  - Fetching data before displaying a component.

Types of Route Guards

1. **Can Activate / Can Activate Child**:
   - **Function**: Allows or restricts navigation to a route or child route.
   - **Use Case**: Protect routes from unauthorized access.
   - **Example**:

```typescript
canActivate: [AuthGuard]
```

2. **Can Deactivate**:
   - **Function**: Controls when a user can navigate away from the current route.
   - **Use Case**: Warns users about unsaved changes.
   - **Example**:

```typescript
canDeactivate: [UnsavedChangesGuard]
```

3. **Resolve**:
   - **Function**: Delays route activation until a task is complete, such as fetching data.
   - **Use Case**: Prefetching data from a backend API.
   - **Example**:

```typescript
resolve: { data: DataResolver }
```

4. **Can Load**:
   - **Function**: Prevents the loading of lazy-loaded modules.
   - **Use Case**: Restricting unauthorized users from accessing specific modules.
   - **Example**:

```typescript
canLoad: [AuthGuard]
```

Implementation of Route Guards

- **Angular 14 and Below**:
  - Create a service class that implements a specific route guard interface.
  - Assign the service to the route's guard property.
  - **Example**:

```
@Injectable({
  providedIn: 'root',
})
export class AuthGuard implements CanActivate {
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): boolean {
    return this.authService.isLoggedIn();
  }
}
```

- **Angular 15 and Higher**:
  - Introduces a functional approach to creating route guards.
  - Simply create a function and assign it to the route guard property.
  - **Example**:

```
const authGuard: CanActivateFn = (route, state) => {
  return authService.isLoggedIn();
};

routes: [
  { path: 'protected', canActivate: [authGuard], component: ProtectedComponent }
]
```

Key Takeaways

- Angular provides route guards to enhance security and control navigation within applications.
- Understanding when and how to use each type of route guard is crucial for effective routing management.
- Transitioning from service classes to functional guards simplifies the implementation process in the latest versions.

Next Steps

- Further exploration of each route guard's implementation with practical examples in upcoming lectures.

Creating an Auth Service