ANGULAR 6

Lesson 07

COMPONENTS DEEP DIVE / ROUTING

Objectives

Components Deep Dive / Routing

- Component Life Cycle Hooks
- Reusable components in angular using <ng-content>
- ng-content and @ContentChild
- Inter-component communication (using Input, Output and Services)

- Navigating with Router links
- Understanding Navigation Paths
- Navigating Programmatically
- Passing Parameters to Routes
- Passing Query Parameters and Fragments
- Setting up Child (Nested) Routes
- Passing static data on routes
- Map() operator and switchMap() operator
- Redirecting and wildcard routes
- Outsourcing Route Configuration (create custom module)

COMPONENTS DEEP DIVE

Component Lifecycle

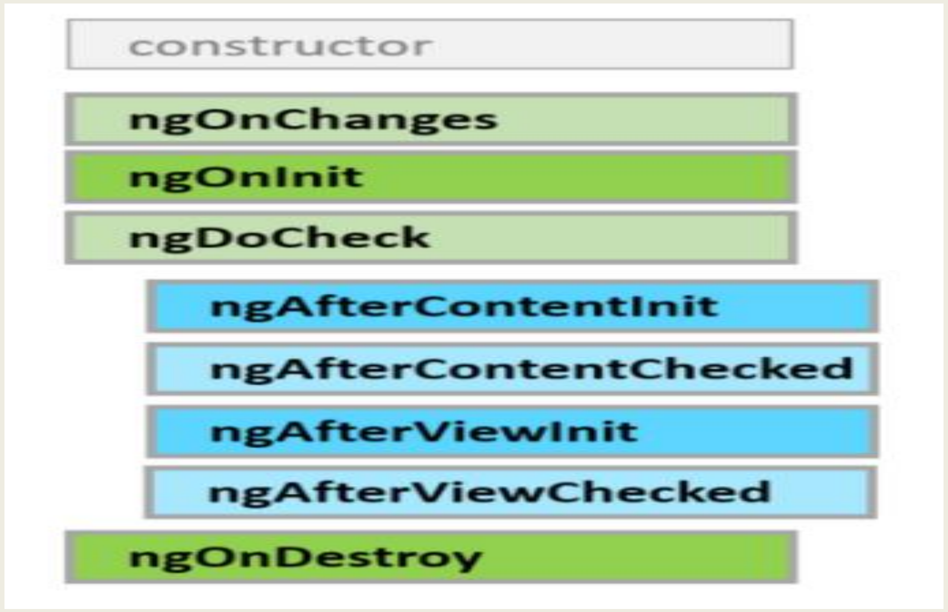
- Each Angular application goes through a lifecycle.
- If we want to access the value of an input - to load additional data from the server for example - you have to use a lifecycle phase.
- The constructor of the component class is called before any other component lifecycle hook.
- For best practice inputs of a component should not be accessed via constructor.
- To access the value of an input for instance to load data from server component's life cycle phase should be used.

Component Lifecycle (Contd…)

A component has a lifecycle managed by Angular.

Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.

Angular offers **lifecycle hooks** that provide visibility into these key life moments and the ability to act when they occur.



Component Life Cycle Hooks

After creating a component by calling its constructor, Angular calls the lifecycle hook methods in the following sequence at specific moments:

Hooks	Purpose and Timing
ngOnChanges()	Respond when Angular (re)sets data-bound input properties. The method receives a SimpleChanges object of current and previous property values. Called before ngOnInit() and whenever one or more data-bound input properties change.
ngOnInit()	Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties. Called once, after the first ngOnChanges().
ngDoCheck()	Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after ngOnChanges() and ngOnInit().

Component Life Cycle Hooks (Contd…)

Hooks	Purpose and Timing
ngAfterContentInit()	Respond after Angular projects external content into the component's view. Called once after the first ngDoCheck(). A component-only hook.
ngAfterViewInit()	Respond after Angular initializes the component's views and child views. Called once after the first ngAfterContentChecked(). A component-only hook.
ngAfterViewChecked()	Respond after Angular checks the component's views and child views. Called after the ngAfterViewInit and every subsequent ngAfterContentChecked(). A component-only hook.
ngOnDestroy()	Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks. Called *just before* Angular destroys the directive/component.

Reusable components in angular using <ng-content>

ng-content and @ContentChild

Inter-component communication (using Input, Output and Services)

ROUTING

Routing

- Routing means loading sub-templates depending upon the URL of the page.
- We can break out the view into a layout and template views and only show the view which we want to show based upon the URL the user is accessing.
- Routes are a way for multiple views to be used within a single HTML page. This enables you page to look more "app-like" because users are not seeing page reloads happen within the browser.
- Defining routes in application can:
  - *Separate different areas of the app*
  - *Maintain the state in the app*
  - *Protect areas of the app based on certain rules*

AngularJS Routes

- AngularJS routes enable us to create different URLs for different content in our application.
- Having different URLs for different content enables the user to bookmark URLs to specific content.
- In Angular 2 routes are configured by mapping paths to the component that will handle them.
- For instance, let consider an application with 2 routes:
  - *A main page route, using the /#/home path;*
  - *An about page, using the /#/about path;*
  - *And when the user visits the root path (/#/), it will redirect to the home path.*

Routing Setup

- To implement Routing to Angular Application
- Import RouterModule and Routes from ‘@angular/router’
  - *import { RouterModule, Routes} from '@angular/router';*
- Define routes for application
  - *const routes: Routes = [{ path: 'home', component: HomeComponent }];*

➢ *Install the routes using RouterModule.forRoot(routes) in the imports of NgModule*

- imports: [BrowserModule, RouterModule.forRoot(routes)]

Components of Angular routing

There are three main components are used to configure routing in Angular

Routes

- Describes the routes application supports

RouterOutlet

- A “placeholder” component that gets expanded to each route’s content

RouterLink

- Directive is used to link to routes

Router

Router Part	Meaning
Router	Displays the application component for the active URL. Manages navigation from one component to the next.
RouterModule	A separate Angular module that provides the necessary service providers and directives for navigating through application views.
Routes	Defines an array of Routes, each mapping a URL path to a component.
Route	Defines how the router should navigate to a component based on a URL pattern. Most routes consist of a path and a component type.
RouterOutlet	The directive (<router-outlet>) that marks where the router displays a view.

Router (Contd...)

Router Part	Meaning
RouterLink	The directive for binding a clickable HTML element to a route. Clicking an element with a routerLinkdirective that is bound to a string or a link parameters array triggers a navigation.
RouterLinkActive	The directive for adding/removing classes from an HTML element when an associated routerLink contained on or inside the element becomes active/inactive.
ActivatedRoute	A service that is provided to each route component that contains route specific information such as route parameters, static data, resolve data, global query params, and the global fragment.
RouterState	The current state of the router including a tree of the currently activated routes together with convenience methods for traversing the route tree.

Router (Contd...)

Router Part	Meaning
Link parameters array	An array that the router interprets as a routing instruction. You can bind that array to a RouterLink or pass the array as an argument to the Router.navigate method.
Routing component	An Angular component with a RouterOutlet that displays views based on router navigations.

Routes

- To define routes for application, create a Routes configuration and then use RouterModule.forRoot(routes) to provide application with the dependencies necessary to use the router.
  - *path specifies the URL this route will handle*
  - *component maps to the Component and its template*
  - *optional redirectTo is used to redirect a given path to an existing route*

```
const routes: Routes = [
      { path: '', redirectTo: 'home', pathMatch: 'full' },
      { path: 'home', component: HomeComponent },
      { path: 'about', component: AboutComponent },
      { path: 'contact', component: ContactComponent },
      { path: 'contactus', redirectTo: 'contact' },
];
```

RouterOutlet

- The router-outlet element indicates where the contents of each route component will be rendered.
- RouterOutlet directive is used to describe to Angular where in our page we want to render the contents for each route

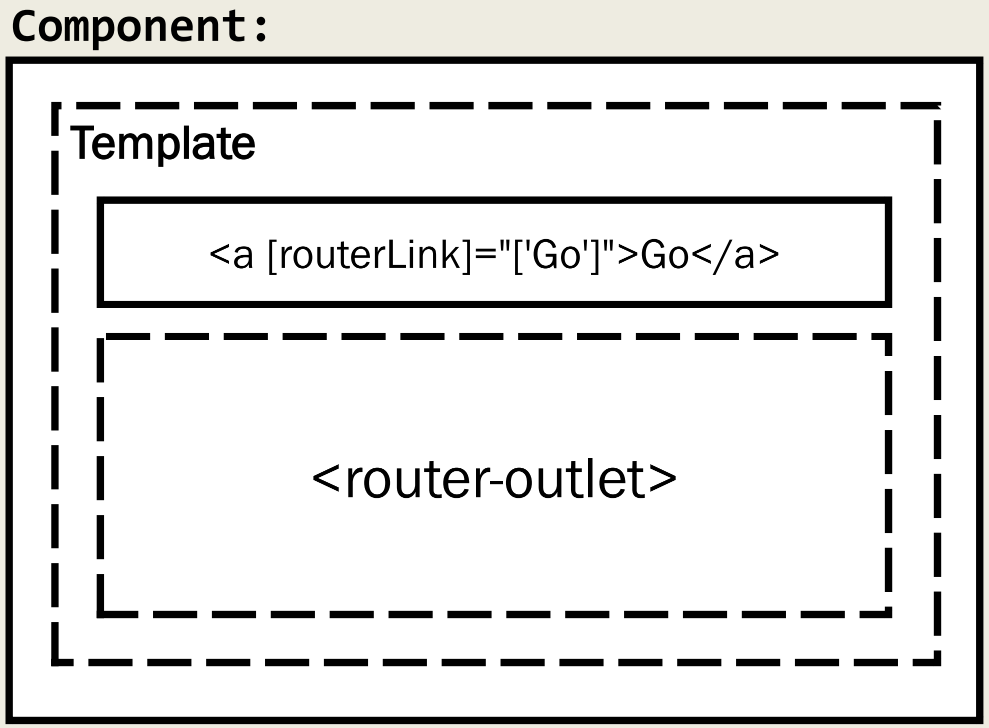
```
@Component({
selector: 'my-app',
template:`<div class="container">
        <router-outlet></router-outlet>
        </div>`
})
```

Navigating with Router links

- It generates link based on the route path.
- routerLink navigates to a route

```
<div>
    <a [routerLink]="['Home']">Home</a>
    <a [routerLink]="['About']">About Us</a>
</div>
```

RouterOutlet & RouterLink



HTML : <a [routerLink]="['Go']">Go</a>

Code : router.navigate(['Go']);

Routing Strategies

The way the Angular application parses and creates paths from and to route definitions is now location strategy.

HashLocationStrategy ('#/')

PathLocationStrategy (HTML 5 Mode Default)

```
//import LocationStrategy and HashLocationStrategy
import {LocationStrategy, HashLocationStrategy} from
'@angular/common';

//add that location strategy to the providers of NgModule
providers: [
  { provide: LocationStrategy, useClass: HashLocationStrategy }
]
```

Passing Parameters to Routes

- Route Parametes helps to navigate to a specific resource. For instance product with id 3
  - *⁄products/3*
- route takes a parameter by putting a colon : in front of the path segment
  - *⁄route/:param*
- To add a parameter to router configuration and to access the value refer the code given below

```
const routes: Routes =([
      { path:'/products/:id', name:‘Product', component:ProductComponent }
])

/*To access the parameter value */
routeParams.get('id')
```

ActivatedRoute

- In order to access route parameter value in Components, we need to import ActivatedRoute

```
import { ActivatedRoute } from '@angular/router’
```

- inject the ActivatedRoute into the constructor of our component

```
export class ProductComponent {
   id: string;

  constructor(private route: ActivatedRoute) {
     route.params.subscribe(params => { this.id = params['id']; });
  }
}
```

Understanding Navigation Paths

Navigating Programmatically

Passing Query Parameters and Fragments

Setting up Child (Nested) Routes

Passing static data on routes

Map() operator and switchMap() operator

Redirecting and wildcard routes

Outsourcing Route Configuration (create custom module)

Demo

- Demo Router
- Demo Router Passing Parameter