Life Cycle of a Component

Day 62/100

Just like we say in react, Angular components have their own life cycle events that are mainly maintained by Angular itself. Below is the list of life cycle events of any Angular components. In Angular, every component has a life–cycle, a number of different stages it goes through from initialization to destruction. There are eight different stages in the component lifecycle. Every stage is called a life cycle hook event. So, we can use these component lifecycle events in different stages of our application to obtains complete controls on the components.

- NgOnChanges – This event executes every time a value of an input control within the component has been changed. This event activates first when the value of a bound property has been changed.
- NgOnInit – This event executed at the time of Component initialization. This event is called only once, just after the ngOnChanges() events. This event is mainly used to initialize data in a component.
- NgDoCheck – This event is executed every time when the input properties of a component are checked. We can use this life cycle method to implement the checking on the input values as per our own logic check.
- NgAfterContentInit – This lifecycle method is executed when Angular performs any content projection within the component views. This method executes only once when all the bindings of the component need to be checked for the first time. This event executes just after the ngDoCheck() method.
- NgAfterContentChecked – This life cycle hook method executes every time the content of the component has been checked by the change detection mechanism of Angular.

This method is called after the ngAfterContentInit() method. This method is can also be executed on every execution of ngDoCheck() event.

• NgAfterViewInit – This life cycle method executes when the component completes the rendering of its view full. This life cycle method is used to initialize the component's view and child views. It is called only once, after ngAfterContentChecked(). This lifecycle hook method only applies to components.

•NgAfterViewChecked – – This method is always executed after the ngAterViewInit() method. Basically, this life cycle method is executed when the change detection algorithm of the angular component occurs. This method automatically executed every execution time of the ngAfterContentChecked().

• NgOnDestroy – This method will be executed when we want to destroy the Angular components. This method is very useful for unsubscribing the observables and detaching the event handlers to avoid memory leaks. It is called just before the instance of the component being destroyed. This method is called only once, just before the component is removed from the DOM.

Nested Component.

On Monday, we discussed the different aspects of components, like the definition, metadata, and life–cycle events. So, when we develop an application, it is very often that there are some requirements or scenarios where we need to implement a nested component. A nested component is one component inside another component, or we can say it is a parent–child component.

The first question that might be raised in our mind: “Does Angular framework support these types of components?” The answer is yes. We can put any number of components within another component. Also, Angular supports the nth level of nesting in general.

Demo 1: Basic Component

Now, we need to create a new component in our Angular 8 Projects. So, either we can create a new project or using the same project as the previous article. In the project folder, we have a component file named app.component.ts. Now, we will first develop a component with inline HTML contains. For that purpose, we will make the below changes in the existing app.component.ts file.

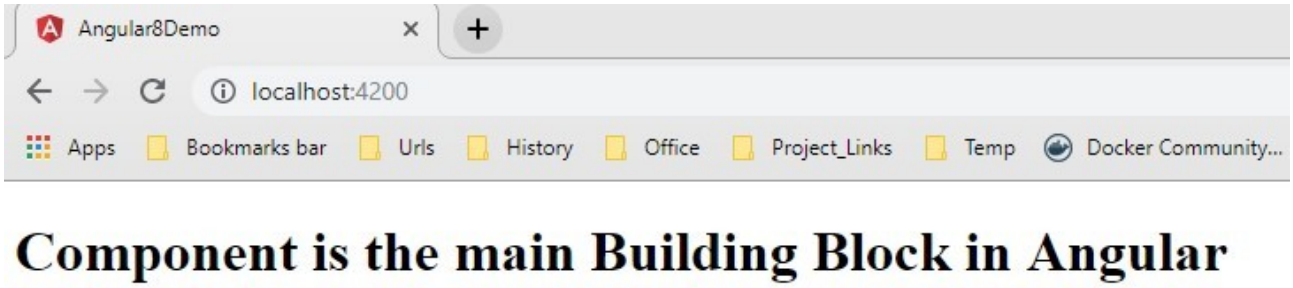
App.component.ts:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: '<h1>Component is the main Building Block in Angular</h1>'
})
export class AppComponent {

}
```

The output will be:



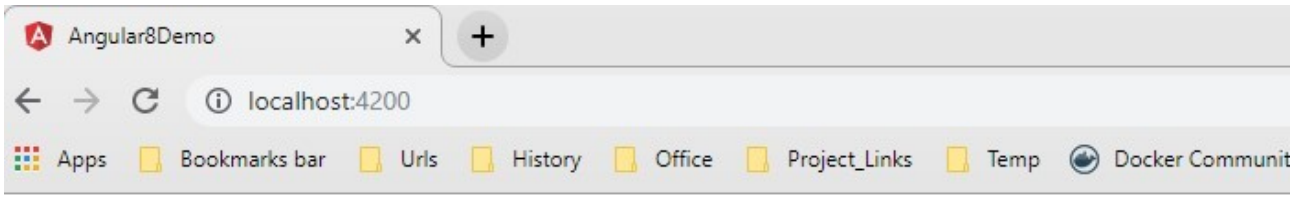
Demo 2: Apply Style into the Content

Now, we need to apply the styles in the above component. So, for that, we will make the below changes in the component. First, we will apply the inline styles in the component.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: '<h1>Component is the main Building Block in Angular</h1> <h2>Angular 8 Samples</h2>',
  styles: ['h1{color:red;font-weight:bold}','h2{color:blue}']
})
export class AppComponent {

}
```

The output will be:



Component is the main Building Block in Angular

Angular 8 Samples

Demo 3: Use External Stylesheet File for Component

Try removing the inline css styles, add them in a custom.css file and link them in the above demo. Share the results with your peers.

Demo 4: Use External HTML File for Component Content

Try removing the Markup logic and try separating them it from the typescript logic, Add your markup code in a file code app.components.html and link it to your app.components.ts

Demo 5: Component Life Cycle Demo

Now in this demo, we will demonstrate the life cycle events of a component. For that, add the below code in the app.component.ts file –

app.component.ts file:

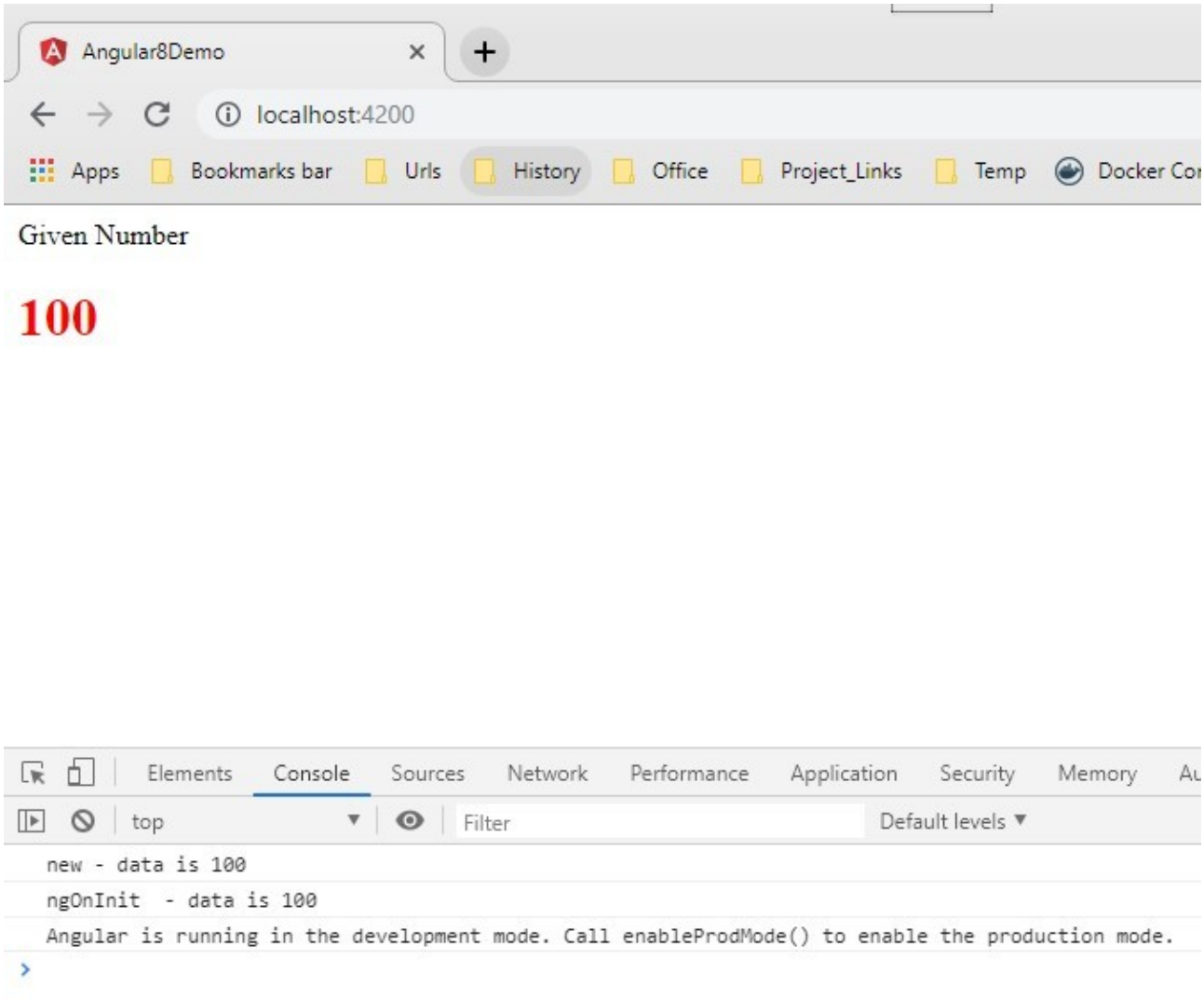
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls : [ './custom.css' ]
})
export class AppComponent {
  data:number=100;
    constructor() {
        console.log(`new - data is ${this.data}`);
    }
    ngOnChanges() {
        console.log(`ngOnChanges - data is ${this.data}`);
    }
    ngOnInit() {
        console.log(`ngOnInit  - data is ${this.data}`);
    }
}
```

Now add the below code in app.component.html file –

```
<span class="setup">Given Number</span>
<h1 class="punchline">{{ data }}</h1>
```

When the broswer refreshes the output will be:



Demo 6: Nested or Parent-Child Component

In Angular, we can develop any component as a parent-child concept. For that purpose, we need to use the child component selector within the parent component HTML file. So, first, we need to develop a child component as below.

child.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'child',
  templateUrl: './child.component.html',
  styleUrls : [ './custom.css' ]
})
export class ChildComponent {

}
```

child.component.html

```
<h2>It is a Child Component</h2>
<p>
    A component is a Reusable part of the application.
</p>
```

app.component.ts

```
import { Component } from '@angular/core';

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

}
```

app.component.html

```
<h1>Demostration of Nested Component in Angular</h1>
<h3>It is a Parent Component</h3>
<child></child>
```

Now include the child component into the app.module.ts file as below –

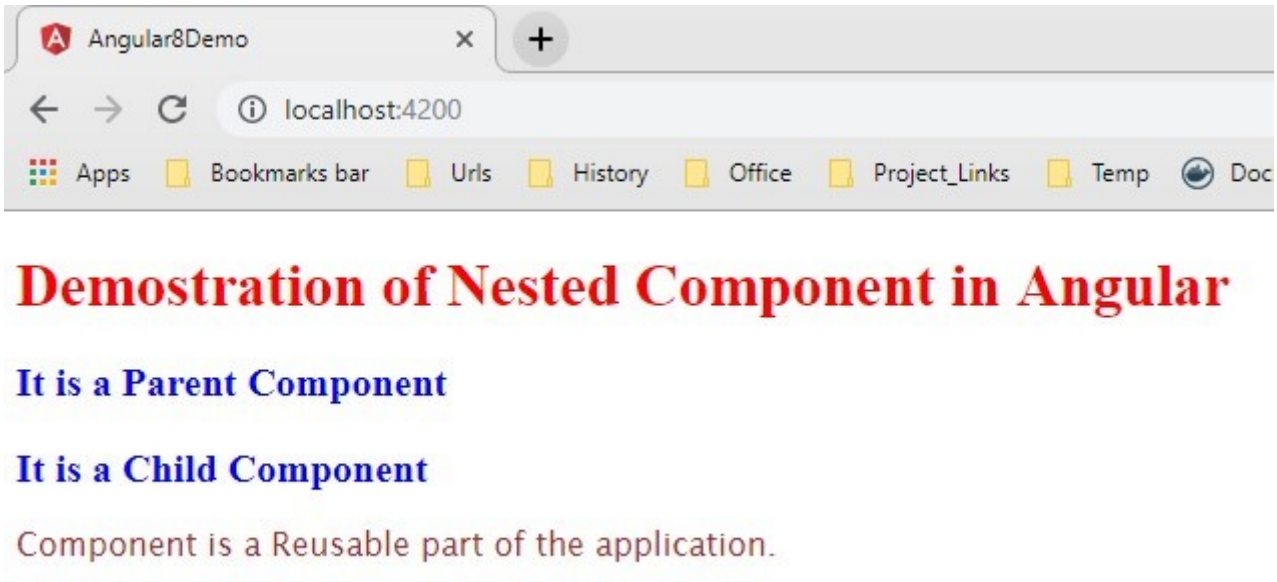
app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { ChildComponent } from './child.component';

@NgModule({
  declarations: [
    AppComponent,ChildComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Now, refresh the browser to check the output -



Tips:

Alway have all your components files in a folder components. Alway seperate you HTML MARK UP logics from the typescript Logics.

All the best, Lux Tech Academy