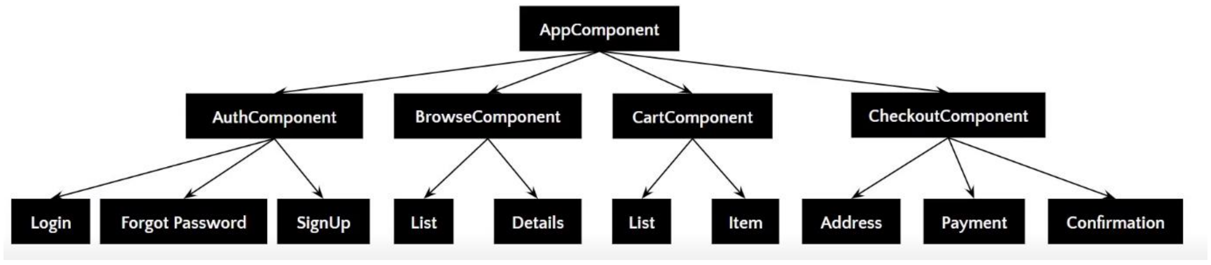
What are Angular Components?

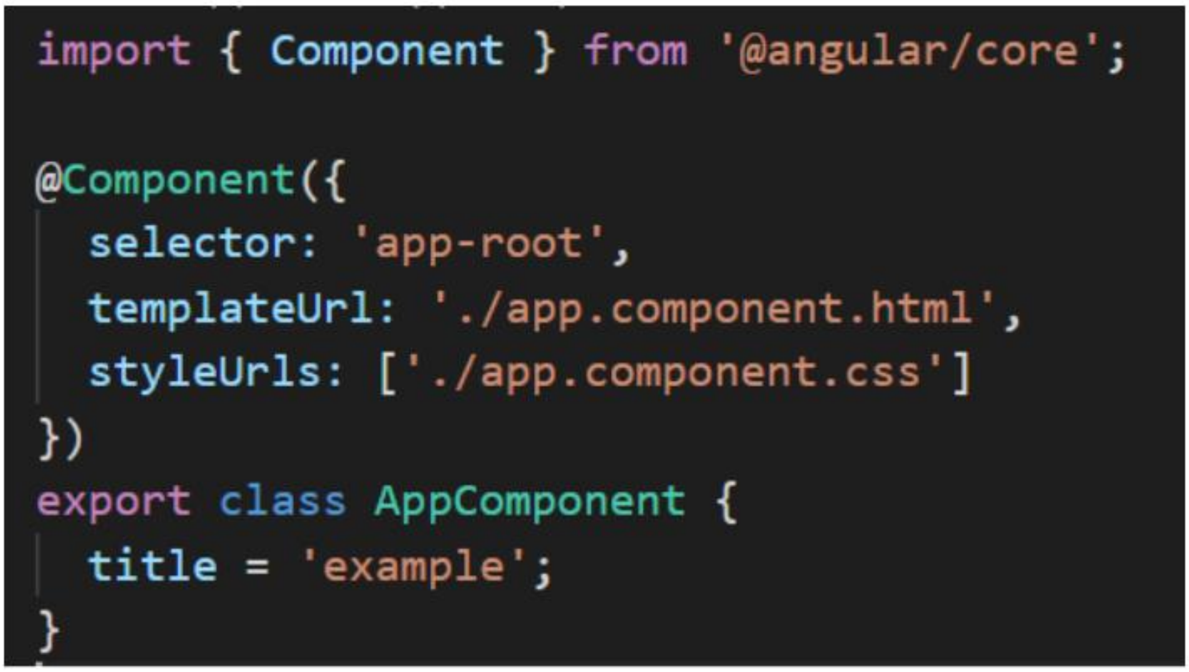
- Angular components are the building blocks of a UI in an Angular application. These components are associated with a template and are a subset of directives.



The above image shows the classification tree structure. A root component, the AppComponent, branches out into other components, creating a hierarchy.

Here are some of the features of Angular Component:

- Components are typically custom HTML elements, and each of these elements can instantiate only one component.
- A TypeScript class is used to create a component. This class is then decorated with the "@Component" decorator.
- The decorator accepts a metadata object that gives information about the component.
- A component must belong to the NgModule for it to be usable by another component or application.



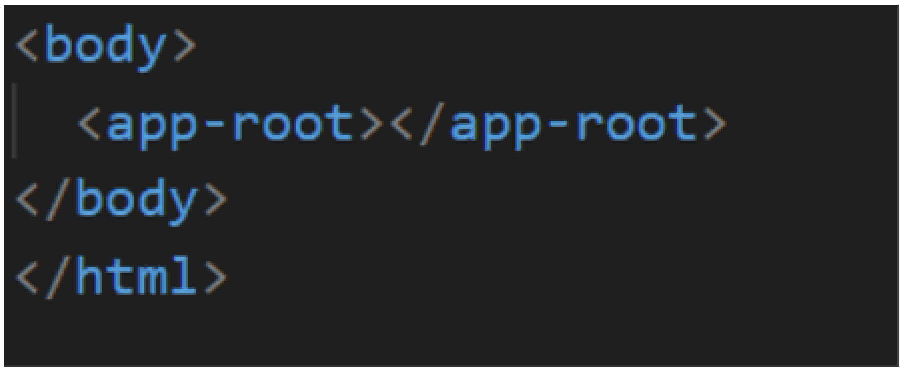
```
import { Component } from '@angular/core';

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

The above image shows an AppComponent, a pure TypeScript class decorated with the "@Component" decorator. The metadata object provides properties like selector, templateUrl, and so on—the templateUrL points to an HTML file that defines what you see on your application.

In the index.html file, the <app-root> tag corresponds to the component's selector. By doing so, Angular will inject the corresponding template for the element.

AppComponent

When you create a new Angular project using the Angular cli command (ng new project-name), by default it will create a component called AppComponent in src/app.



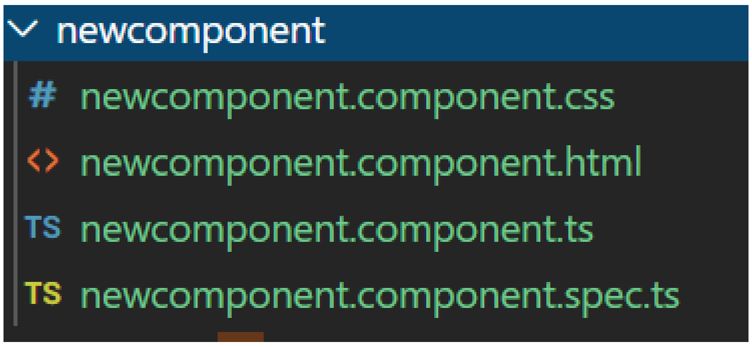
```
<body>
  <app-root></app-root>
</body>
</html>
```

Creating Your First Angular Component

- Angular CLI is used to create an Angular component. In the terminal, type in the command,

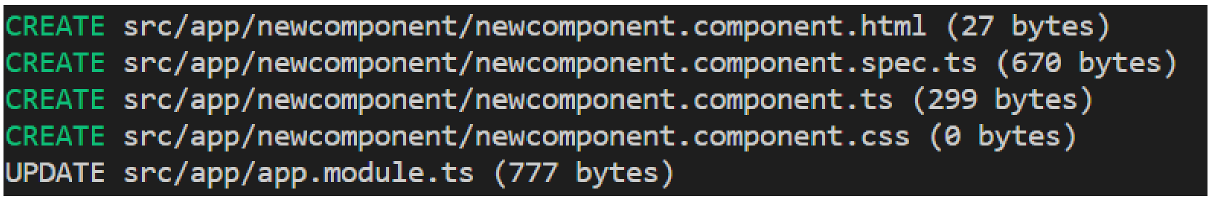
ng g c component-name

- This will create a folder named component-name with four files.



```
newcomponent
  newcomponent.component.css
  newcomponent.component.html
  newcomponent.component.ts
  newcomponent.component.spec.ts
```

You'll also receive a message:



```
CREATE src/app/newcomponent/newcomponent.component.html (27 bytes)
CREATE src/app/newcomponent/newcomponent.component.spec.ts (670 bytes)
CREATE src/app/newcomponent/newcomponent.component.ts (299 bytes)
CREATE src/app/newcomponent/newcomponent.component.css (0 bytes)
UPDATE src/app/app.module.ts (777 bytes)
```

The update message indicates that the component created is included in the declarations array of the main component.

Angular needs to know which component to run next and its features. For that, some metadata is created. The following section addresses the component metadata.

Component Decorator Metadata

As mentioned earlier, the @Component decorator accepts a metadata object that provides information about the component. Here's a list of properties of the metadata object:

```
@Component({
 selector: 'app-root',
 template: `<h1>Hello! Welcome</h1>`,
 templateUrl: './app.component.html',
 styles: [`
  h3{
   color: blue;
  }
 `],
 styleUrls: ['./app.component.css']
```

Selector

It is the CSS selector that identifies this component in a template. This corresponds to the HTML tag that is included in the parent component. You can create your HTML tag. However, the same has to be included in the parent component.

Template

It is an inline-defined template for the view. The template can define some markup, typically including headings or paragraphs displayed on the UI.

templateUrl

It is the URL for the external file containing the template for the view.

Styles

These are inline-defined styles to be applied to the component's view.

styleUrls

List of URLs to stylesheets to be applied to the component's view.

Providers

It is an array where certain services can be registered for the component.

Angular Lifecycle Hooks

- Every component has a lifecycle, like us . The lifecycle starts when a component is instantiated, continues with change detection, and ends when the component template is removed from the DOM.
- Directives have a similar lifecycle.
- Angular gives us lifecycle hook methods to take advantage of events in the lifecycle

How can we use these lifecycle methods?

- You can import these interfaces from the @angular/core library.
- Every lifecycle interface has only 1 method. There is standardization for naming methods. It takes an ng prefix with the interface name. For example, look at the OnInit interface from the @angular/core library.

Sequence of Lifecycle methods

- ngOnChanges
- ngOnInit
- ngDoCheck
- ngAfterContentInit
- ngAfterContentChecked
- ngAfterViewInit
- ngAfterViewChecked
- ngOnDestroy
- When the component is rendered to the template, these methods will run at the appropriate time. Of course, above all, the constructor runs first.

ngOnChanges()

- This one is called before **ngOnInit**, **only if you have an input value.** If you don't have an input value, this method won't run.
- When the input value resets or updates, this method will be triggered. ngOnChanges method gets a **SimpleChanges** object, which shows the **currentValue, firstChange, previousValue.**
- Your input values might change frequently, so making an HTTP request or something heavy might affect your application’s performance.
- By the way, **If your input is an object and if you change its property it wont catch by Angular, because it still referencing to the same address**. Let me show you an example.

ngOnInit()

- This method is **called only once** during the component lifecycle and is the most used hook in Angular.
- It is called after the **ngOnChanges** method. Even if ngOnChanges is not called, it will still be called.
- We can use ngOnInit for initialization tasks. This is a good place for fetching data from a server, and **it's a best practice**.
- You may ask why not the constructor? Because **you can't reach input variables in the constructor.**

ngDoCheck()

- this method is **called almost every change detection.**
- In previous video i have shown you that angular couldn't catch the input object property changes, **so we can implement our own custom change detection here.**

ngAfterContentInit()

- This hook is called only once after the first **ngDoCheck().**
- Before going in, I should explain what **content** is. If you are putting HTML elements inside the child component tags, it means you are **projecting content** to the child.
- Catch this content with the ng-content tag in the child component.
- So if you are working with ng-content and want to do something after this content is initialized you can use this lifecycle hook. Because this is the first time that we can access to the **ElementRef** of the**ContentChild**.
- **in this hook we only have access to the projected content**. while the template is not initialized yet we cannot access any other elements. it will be available to access on the ngAfterViewInit hook.

ngAfterContentChecked()

- This hook will be called after **ngAfterContentInit** and after every **ngDoCheck**.
- You may use this hook to react to changes in the projected content.
- **You shouldn't do heavy things here** because **ngDoCheck** triggers this hook.

ngAfterViewInit()

- This hook is called only once after the first **ngAfterContentChecked**. With this hook, we understand that the component and child views are initialized.
- This is the first time that we can access the **ElementRef** of the **ViewChildren**.

ngAfterViewChecked()

- This hook will be called after **ngAfterViewInit** and after every **ngAfterContentChecked**.
- This is called after angular checked component views and its child views.
- The triggering chain is like this;
- **ngDoCheck** → **ngAfterContentChecked** → **ngAfterViewChecked** so like others, this hook will be called almost every change detection. **You shouldn't make heavy operations in this method.**

ngOnDestroy()

- **Called before Angular removes the component template from the DOM**.
- You can use this hook to unsubscribe from observables (**use async pipe as much as possible**), clean up your intervals, timeouts, local storage, etc. Eventually, this is a good place to handle memory leaks.