Contents

[Components 2](#_Toc19633194)

[- Selector 2](#_Toc19633195)

[- Providers 2](#_Toc19633196)

[- Pipes 2](#_Toc19633197)

[- Directives 2](#_Toc19633198)

[o Components 2](#_Toc19633199)

[o Structural 2](#_Toc19633200)

[o Attribute directives 2](#_Toc19633201)

[Template 2](#_Toc19633202)

[Directive 2](#_Toc19633203)

[Binding 3](#_Toc19633204)

[- Event binding 3](#_Toc19633205)

[- Property binding 3](#_Toc19633206)

[- Two-way data binding 3](#_Toc19633207)

[Service 3](#_Toc19633208)

[NgModules 3](#_Toc19633209)

[o Declarations 3](#_Toc19633210)

[o providers 3](#_Toc19633211)

[Dependency injection (DI) 3](#_Toc19633212)

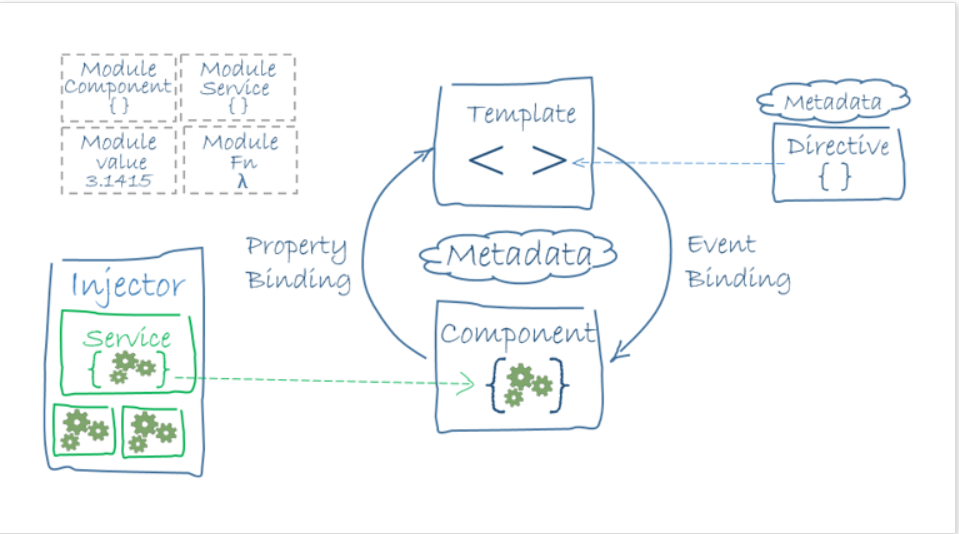
[Template expressions 4](#_Toc19633213)

[Template statements 4](#_Toc19633214)

[@Input() 4](#_Toc19633215)

[@Output() 4](#_Toc19633216)

[Safe navigation operator ( ? ) 4](#_Toc19633217)



# Components

Component defines a class that contains application data and logic, and is associated with an HTML template that defines a view.

* Selector: tells Angular to create and insert an instance of this component wherever it finds the corresponding tag in template HTML.
* Providers: A provider is an object that tells an injector how to obtain or create a dependency. An array of providers for services that the component requires.
* Pipes: to transform data before it is displayed
* Directives: to apply app logic to what gets displayed.
  + Components: directives with a template.
  + Structural: change the DOM layout by adding and removing DOM elements (NgFor and NgIf.)
  + Attribute directives: change the appearance or behavior of an element, component, or another directive (NgStyle)

# Template

A template combines HTML with Angular markup that can modify HTML elements before they are displayed.

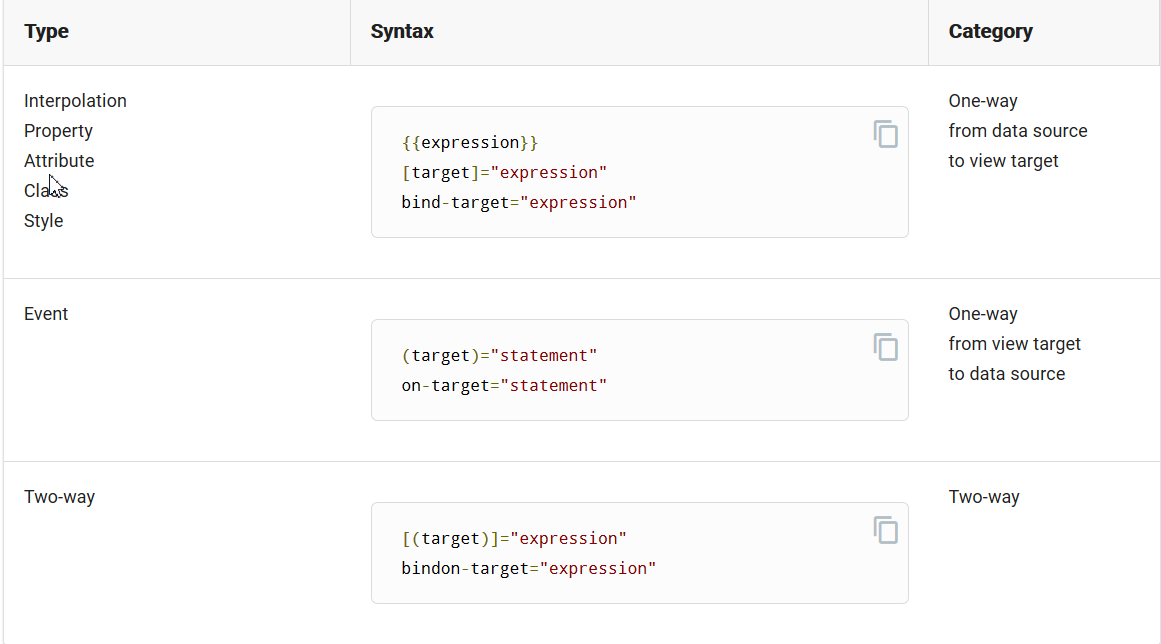
# Directive

Directives provide program logic

# Binding

Binding markup connects our application data and the DOM. Two types of data binding:

* Event binding lets your app respond to user input in the target environment by updating your application data
* Property binding lets you interpolate values that are computed from your application data into the HTML
* Two-way data binding (used mainly in template-driven forms) combines property and event binding in a single notation.
  + Sets a specific element property.
  + Listens for an element change event.



# Service

For data or logic that isn't associated with a specific view, and that we want to share across components

* Fetching data from the server, validating user input.

# NgModules

* are containers for a cohesive block of code dedicated to an application domain
* can contain components, service providers, and other code files
  + Declarations: The components, directives, and pipes that belong to this NgModule.
  + providers: The same instance of a service is available to all components in that NgModule.

# Dependency injection (DI)

* DI is wired into the Angular framework and used everywhere to provide new components with the services or other things they need
* An injector creates dependencies, and maintains a container of dependency instances that it reuses if possible

# Template expressions

A template expression produces a value and appears within the double curly braces, {{ }}. Angular executes the expression and assigns it to a property of a binding target.

* Simplicity
* Quick execution
* No visible side effects

# Template statements

A template statement responds to an event raised by a binding target such as an element, component, or directive. A template statement has a side effect.

# @Input()

@Input() decorator in a child component or directive to let Angular know that a property in that component can receive its value from its parent component.

# @Output()

Use the @Output() decorator in the child component or directive to allow data to flow from the child out to the parent.

# Safe navigation operator ( ? )

*{{item?.name}}* If item is null, the view still renders but the displayed value is blank

# Lifecycle sequence

## OnInit()

Use ngOnInit() for two main reasons:

* To perform complex initializations shortly after construction
* To set up the component after Angular sets the input properties.

## OnChanges()

Angular calls its ngOnChanges() method whenever it detects changes to input properties of the component

## AfterView

The AfterView sample explores the AfterViewInit() and AfterViewChecked() hooks that Angular calls after it creates a component's child views

## AfterContent

The AfterContent sample explores the AfterContentInit() and AfterContentChecked() hooks that Angular calls after Angular projects external content into the component

# Component Interaction

* Pass data from parent to child with input binding
* Parent listens for child event
  + The child component exposes an EventEmitter property with which it emits events when something happens. The parent binds to that event property and reacts to those events
* Intercept input property changes with a setter
  + Use an input property setter to intercept and act upon a value from the parent.
* Intercept input property changes with ngOnChanges()
* Parent calls an @ViewChild()
* Parent and children communicate via a service

# ComponentFactoryResolver

use ComponentFactoryResolver to add components dynamically.

A simple registry that maps Components to generated ComponentFactory classes that can be used to create instances of components