# Angular 2

Lesson 5—Binding and Events

## Learning Objectives

- ✓ What is a Template Model
- ✓ How Angular2 binding works and the types of bindings
- ✓ Understand Angular2 built-in directives
- ✓ Basics of Webpack and SystemJS

## Binding and Events

Topic 1—Template Model

## Templates

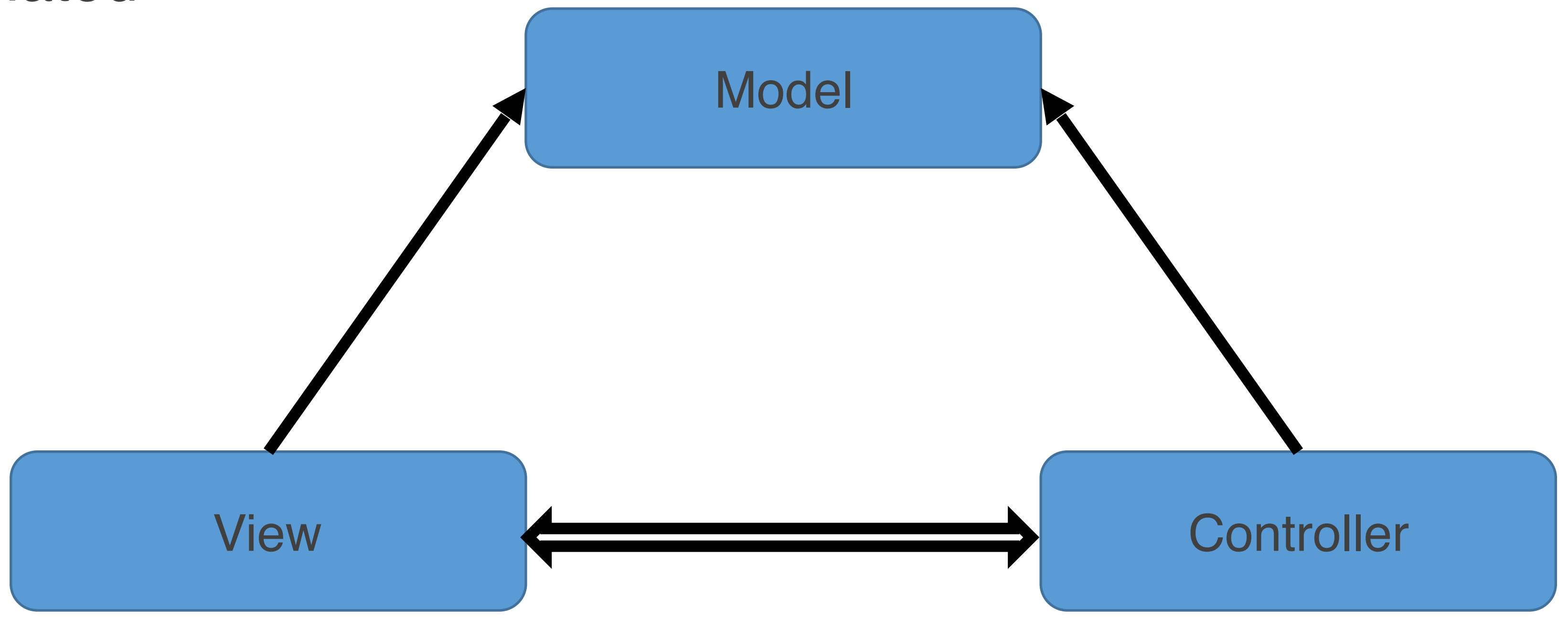
Angular Application manages access to users through the interaction between the component class instance (Component) and its user-facing template.

## Templates—MVC With Angular 2

The **Model-View-Controller (MVC)** is an architectural pattern which segregates an application into three logical components.

**Model:** The Model corresponds to all the data-related logic.

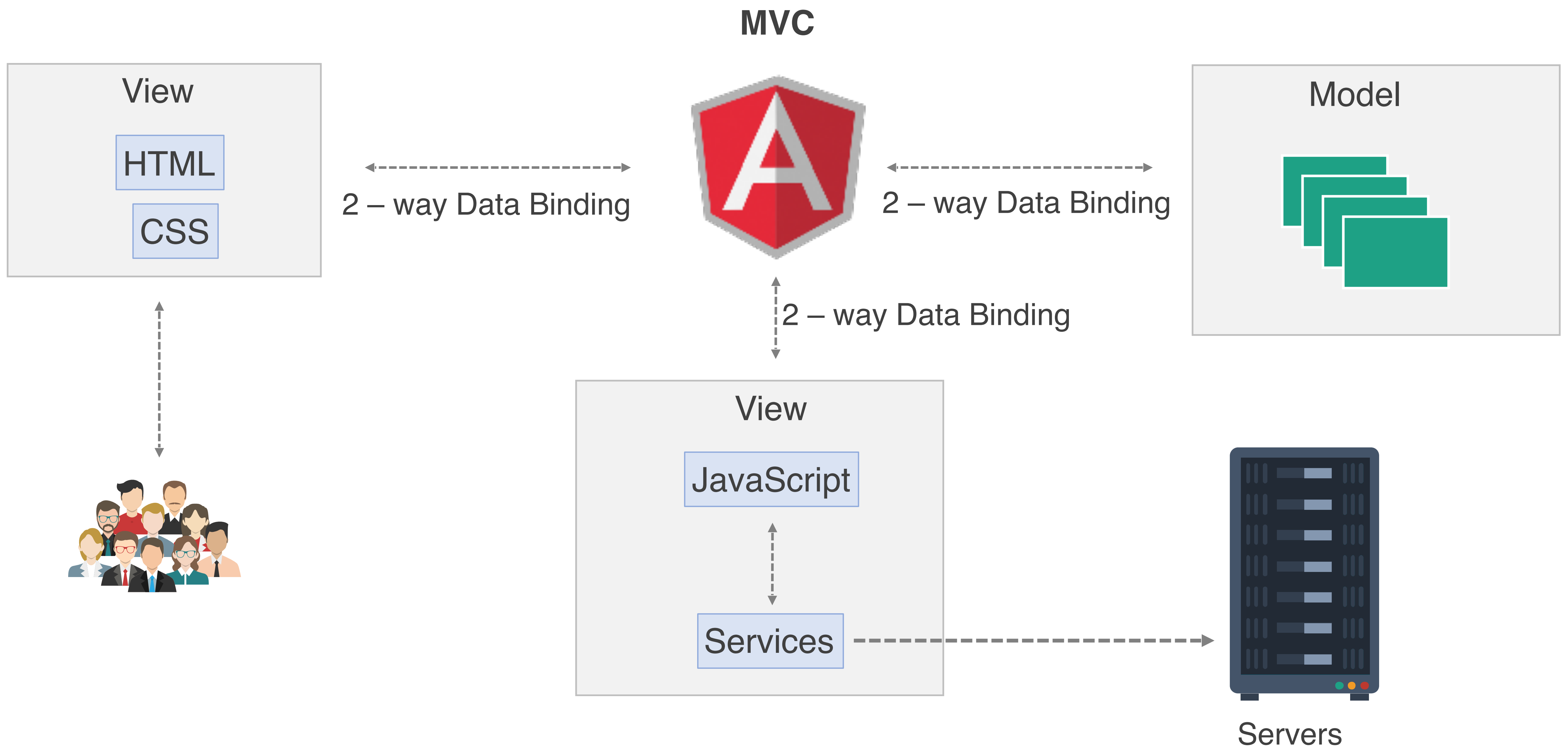
**View:** The View is used for all the UI logic of the application.



**Controller:** Controller acts as a connection between Model and View to process all the business logic and data manipulation-related tasks.

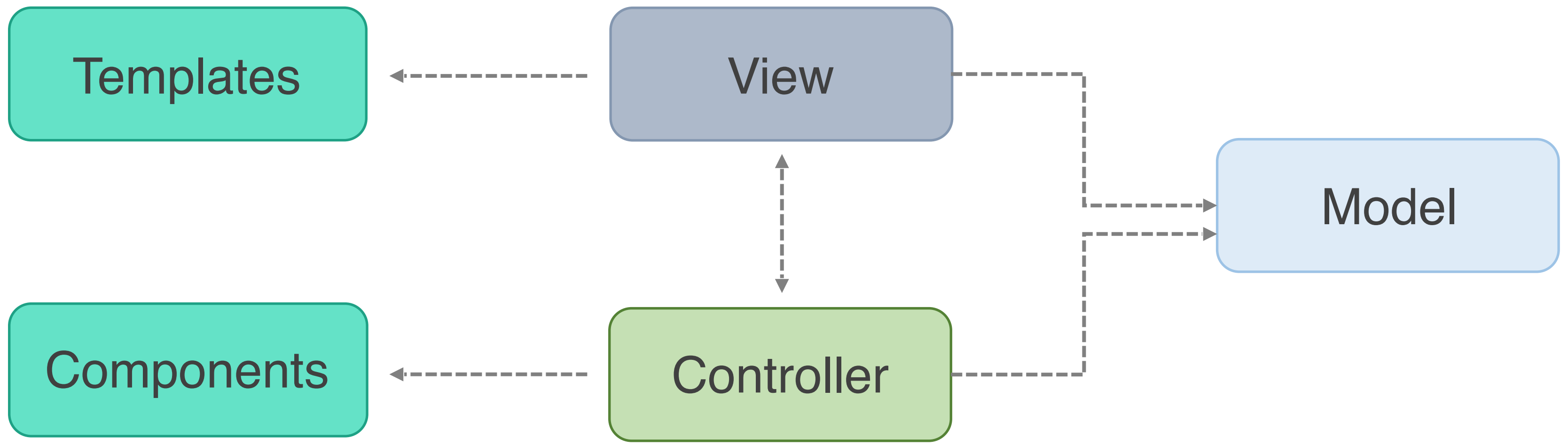
## Templates

You will be familiar with component/template duality if you have worked on Model-View-Controller (MVC) or Model-View-ViewModel (MVVM).



## Templates

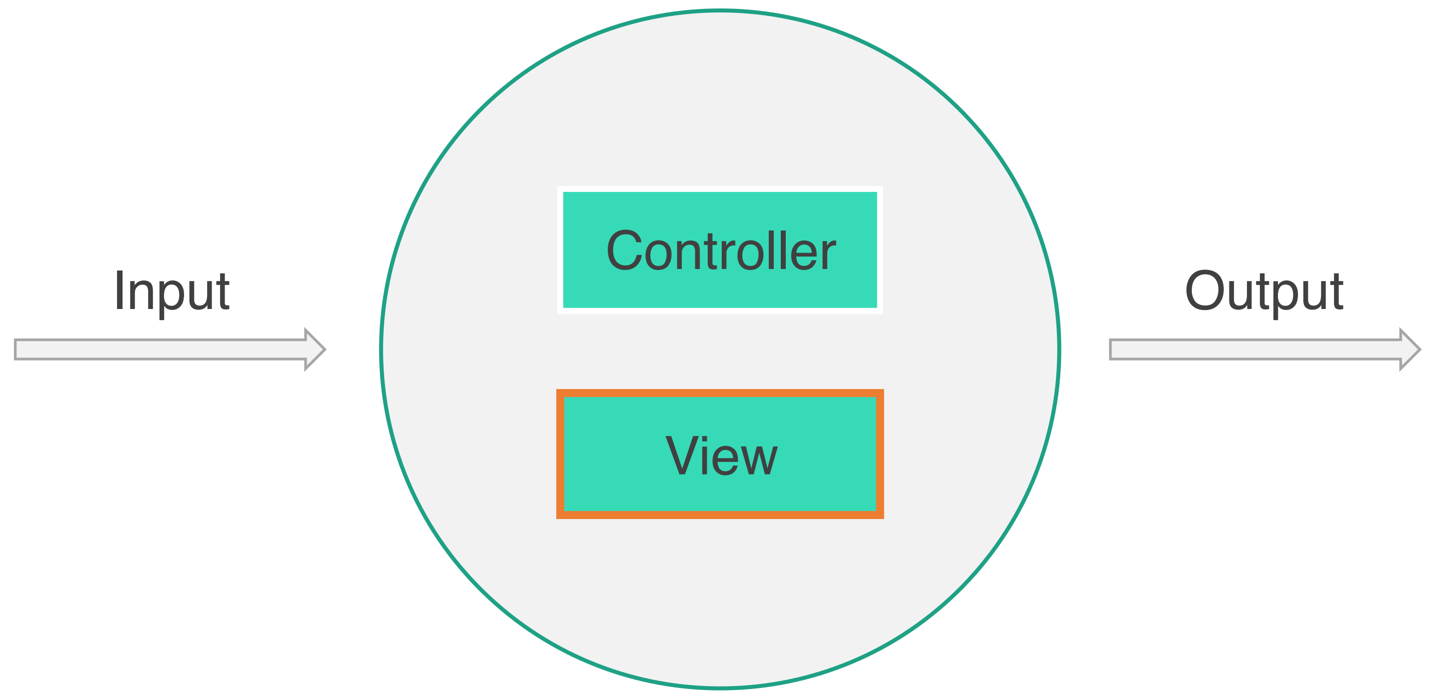
In Angular 2, the component plays the part of the controller and the template represents the view.



Angular 2 templating system provides a syntax to express the dynamic part of HTML.

## Templates

In Angular 2, a component needs to have a view.

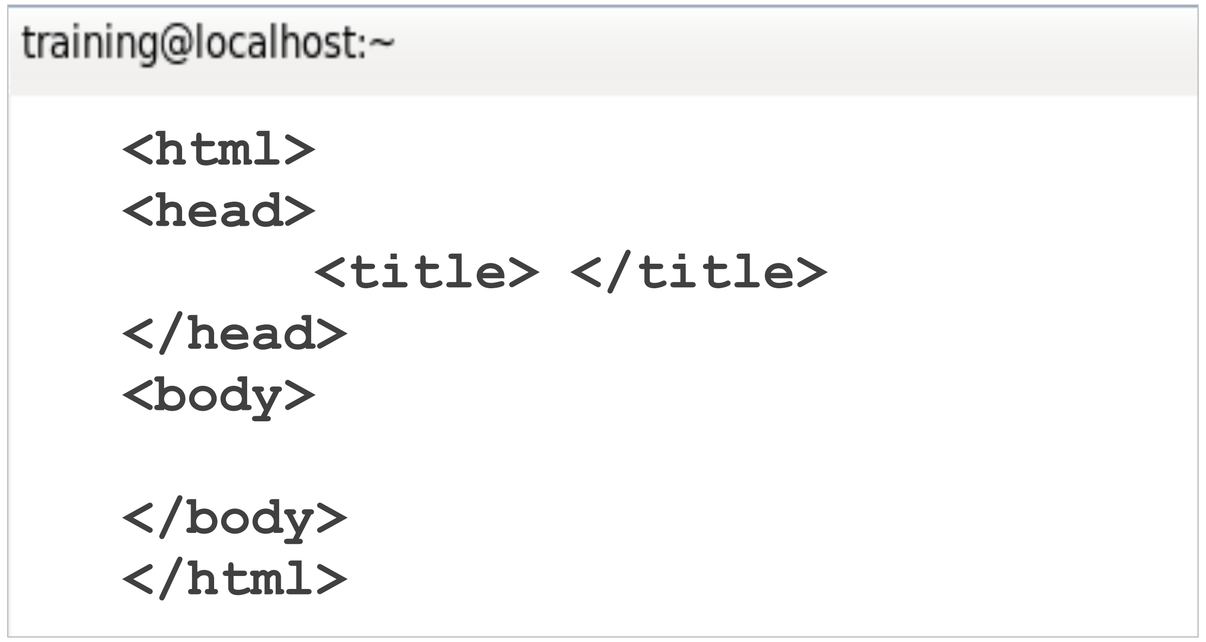


To define view, you can define a template inline (using template) or in a separate file (using templateUrl).

## HTML in Templates

HTML is the language of the Angular template.

Almost all HTML syntax is valid template syntax.



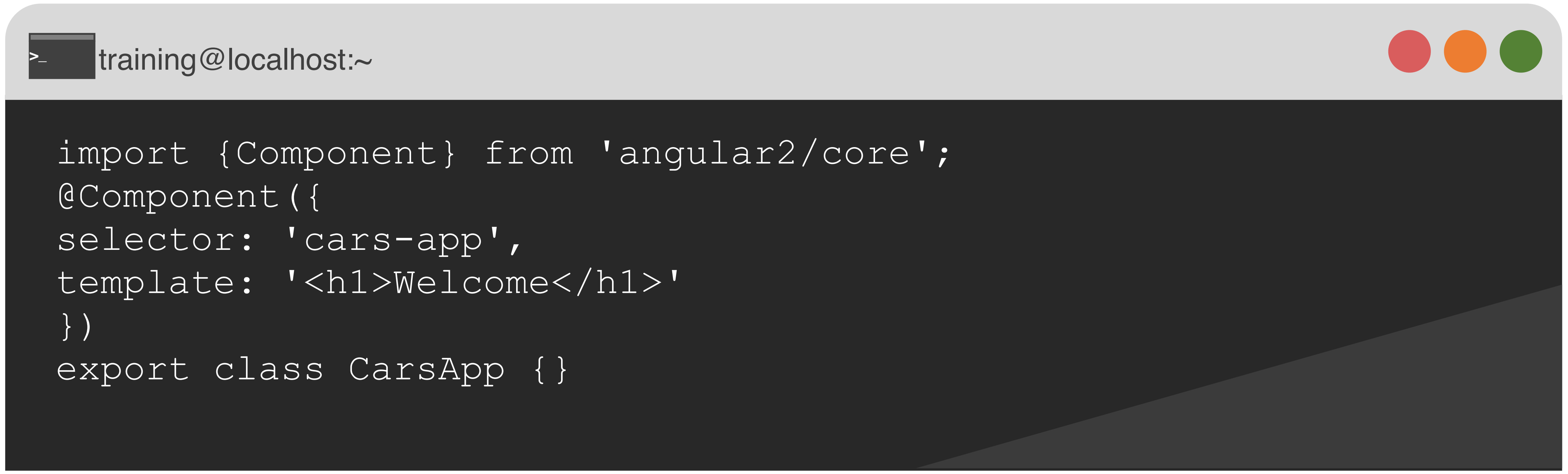
```
<html>
<head>
      <title> </title>
</head>
<body>

</body>
</html>
```

**Important Points:**

- The *<script>* element is a notable exception; it is forbidden as it carries the risk of script injection attacks.
- If *<script>* is ignored, a warning appears in the browser console.
- Some legal HTML tags like *<html>*, *<body>,* and*<base>* have no role.

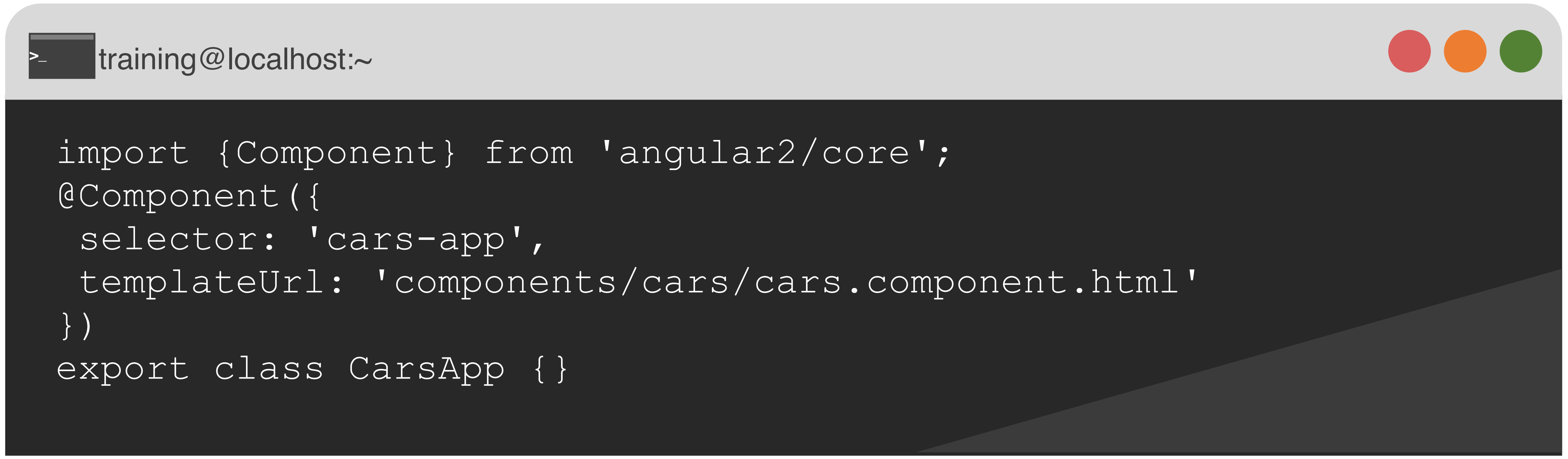
# Templates Syntax

**Template Inline**



```
import {Component} from 'angular2/core';
@Component({
selector: 'cars-app',
template: '<h1>Welcome</h1>'
})
export class CarsApp {}
```

**Separate file**



```
import {Component} from 'angular2/core';
@Component({
 selector: 'cars-app',
 templateUrl: 'components/cars/cars.component.html'
})
export class CarsApp {}
```

## Templates Symbols

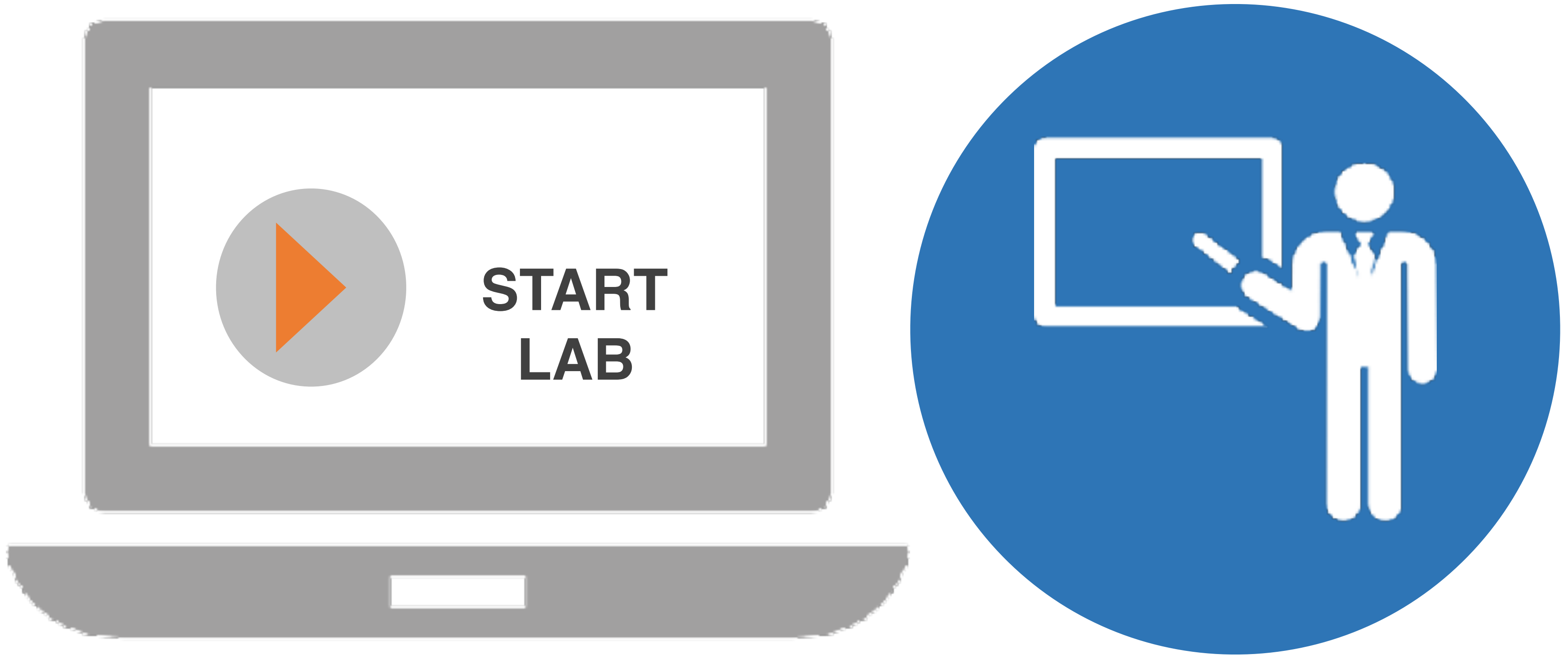
```
<html>
<title>HTML</title>
<body>
This is HTML!
</body>
</html>
```

- A template helps to render HTML with some dynamic parts, depending on the data.
- It allows you to express data, property binding, event binding, and templating concerns.

To be able to express those behaviors, Angular 2 comes with its own symbols:

| | |
|---|---|
| {{ }} | ---- Interpolation |
| [] | ---- Property Binding |
| () | ---- Event Binding |
| # | ---- Variable Declaration |
| * | ---- Structural Directives |

## Lab—Demo



## Binding and Events

Topic 2—Binding

## Binding and Component Data Binding

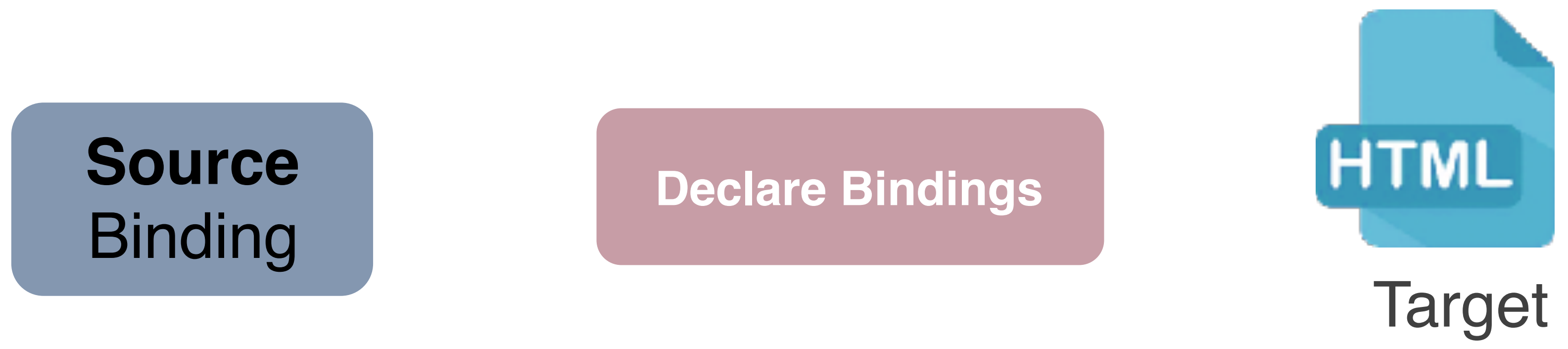
Binding is a mechanism for coordinating between what users see and data values in the application.

**Binding**

While pushing values to and pulling values from HTML, it will be easier to write, read, and maintain the application if you turn these chores over to a binding framework.

## Binding and Component Data Binding

We need to declare bindings between sources and target HTML elements to let the framework perform the task.



## Binding

Angular provides many types of data binding.

{{Data binding}}

Binding types can be grouped into three categories that are classified based on the direction of data flow between:

- Source-to-view
- View-to-source
- In the two-way sequence it is ’view-to-source-to-view’

## Binding

**Types of Binding supported by Angular 2 are:**

- Property Binding
- Class Binding
- Style Binding
- Event Binding
- Two-Way Binding
- Component Binding

## Property Binding

Every DOM property can be written via special attributes on HTML elements using square brackets []. An HTML attribute can start with anything.

Angular 2 maintains the properties and attributes in sync when you use them, so if you are familiar with Angular 1.x directives, such as ng-hide, you can work directly with the hidden property and no more ng-hide is needed.

```
<div ng-hide="isHidden">Hidden element or not?</div>
```

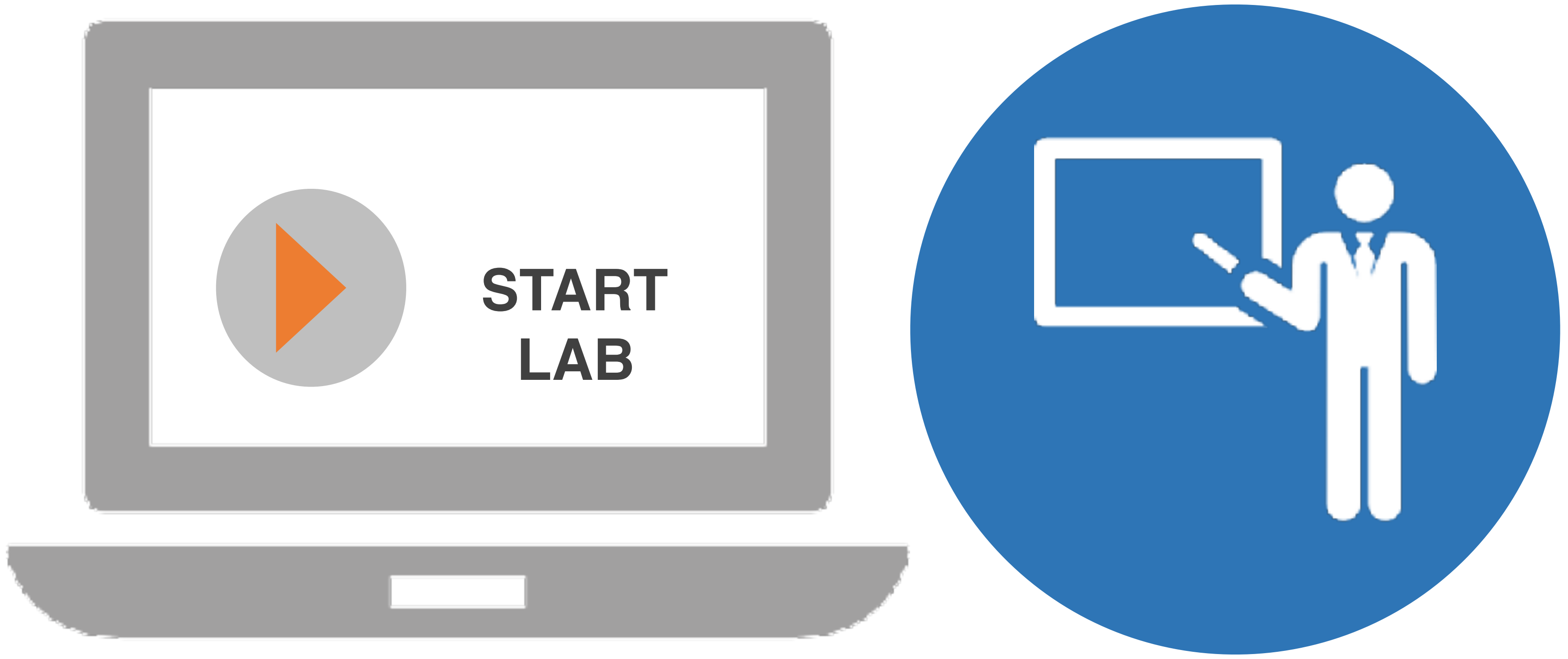
The most common property binding sets an element property to a component property value.

training@localhost:~

```
<img [src]="heroImageUrl">

<button [disabled]="isUnchanged">Cancel is disabled</button>
```

# Lab—Demo

Property Binding



## Class Binding

You can add and remove CSS class names from an element's class attribute with a class binding.

Class binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix class, optionally followed by a dot (.) and the name of a CSS class: [class.class-name].

The following examples show how to add and remove the application's "special" class with class bindings. Here's how to set the attribute without binding:

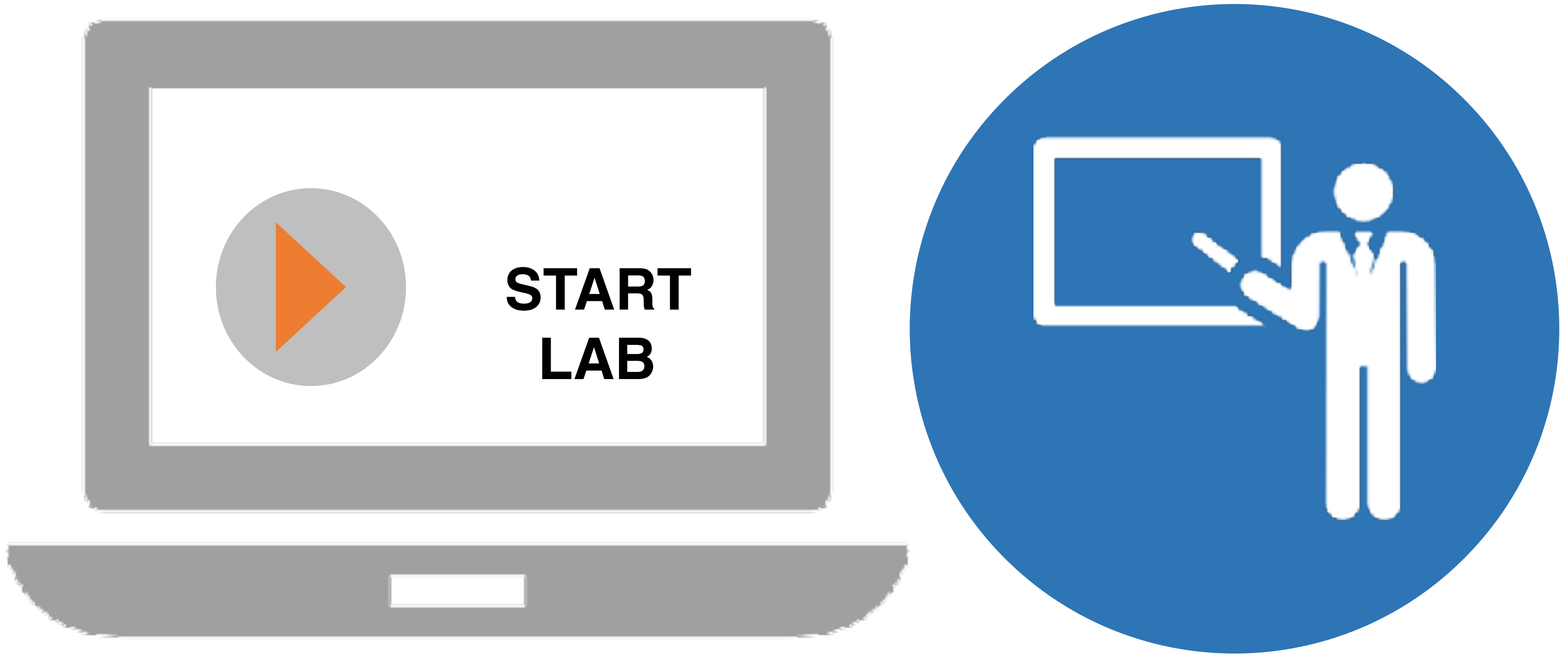
```
<!-- standard class attribute setting-->
<div class="bad curly special">Bad curly special</div>
```

Class binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix class, optionally followed by a dot (.) and the name of a CSS class: [class.class-name].

```
<!-- reset/override all class names with a binding-->
<div class="bad curly special‶[class]="badCurly">Bad curly</div>
```

## Lab—Demo

Class Binding



## Style Binding

You can set inline styles with a style binding.

Style binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix style, followed by a dot (.) and the name of a CSS style property: [style.style-property].

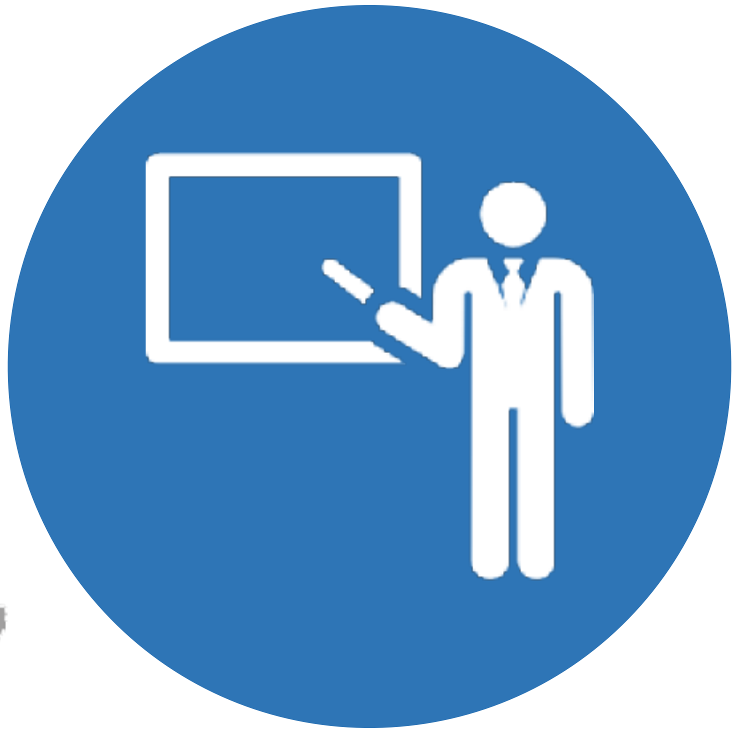
```
<button [style.color]="isSpecial ? 'red': 'green'">Red</button>
<button [style.background-color]="canSave ? 'cyan': 'grey'"
>Save</button>
```

Some style binding styles have a unit extension. The following example conditionally sets the font size in “em” and “%” units.

```
<button [style.font-size.em]="isSpecial ? 3 : 1" >Big</button>
<button [style.font-size.%]="!isSpecial ? 150 : 50" >Small</
button>
```

## Lab—Demo

Style Binding



## Event Binding

The bindings directives you've met so far allow data to flow in one direction: from a component to an element.

Users don't just stare at the screen. They enter text into input boxes. They pick items from lists. They click buttons. Such user actions may result in a flow of data in the opposite direction: from an element to a component.

The only way to know about a user action is to listen for certain events such as keystrokes, mouse movements, clicks, and touches. You declare your interest in user actions through Angular event binding.

Event binding syntax consists of a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.

```
<button (click)="onSave()">Save</button>
```

## Lab—Demo

Event Binding

## Two-Way Binding

You often want to both display a data property and update that property when the user makes changes.

On the element side, it takes a combination of setting a specific element property and listening for an element change event.

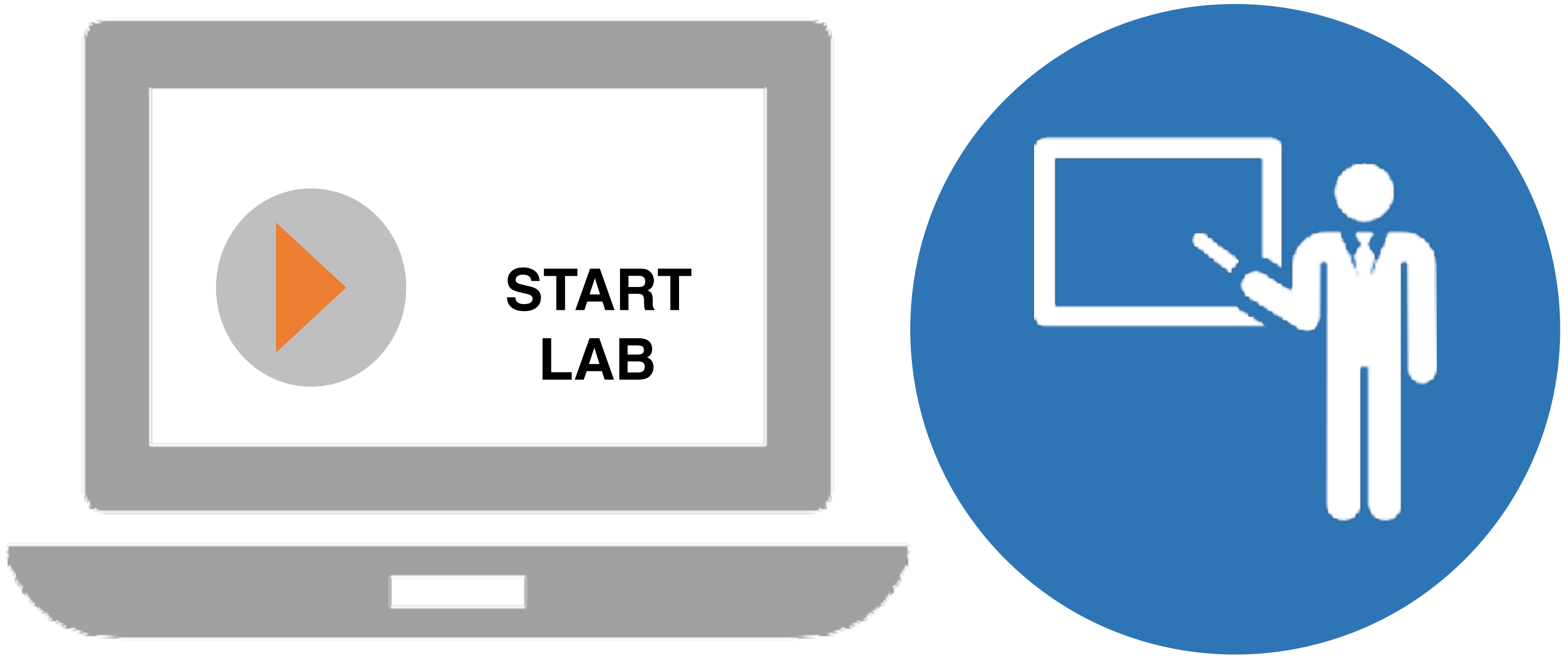
Angular offers a special two-way data binding syntax for this purpose, [(x)]. The [(x)] syntax combines the brackets of property binding, [x], with the parentheses of event binding, (x).

The two-way binding syntax is really just syntactic sugar for a property binding and an event binding. Angular desugars the SizerComponent binding into this:

```
<my-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></my-sizer>
```

## Lab—Demo

Two-Way Binding



## Component Binding

Angular components use events to notify parent components that something has changed.

There’s no longer two-way data binding; it’s designed around a unidirectional data flow system that adopts a much more reasonable approach to application development.

Let’s understand the basics of parent-child and child-parent communication by introducing EventEmitter and @Output.

EventEmitter is an Angular2 abstraction, and its only purpose is to emit events in components.

@Output is used to define output property to achieve custom event binding.

## Lab—Demo

Component Binding

## Binding and Events

Topic 3—Built-in Directives

## Introduction to ngModel

It would be convenient to use two-way binding with HTML form elements like <input> and <select>. However, no native HTML element follows the x value and xChange event pattern.

NgModel - Two-way binding to form elements with [(ngModel)]

When developing data entry forms, you often both display a data property and update that property when the user makes changes.

Fortunately, the Angular NgModel directive is a bridge that enables two-way binding to form elements.

```
<input [(ngModel)]="currentHero.name">
```

## Lab—Demo

ngModel

## Built-in Directives

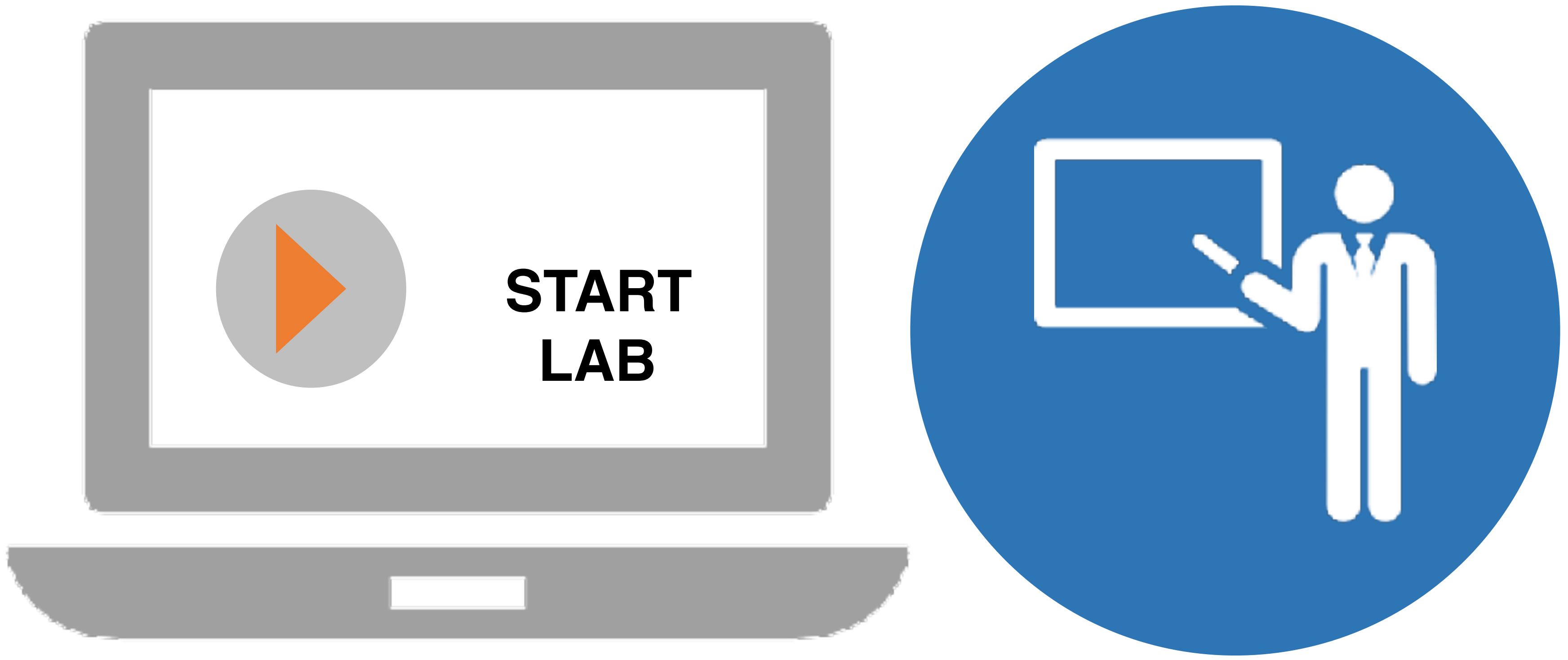
- Structural directives are responsible for HTML layout.
- They shape or reshape the DOM's structure, typically by adding, removing, and manipulating the host elements to which they are attached.

Here, we discuss the common structural directives:

| | |
|---|---|
| NgIf | Conditionally add or remove an element from the DOM |
| NgFor | Repeat a template for each item in a list |
| NgSwitch | A set of directives that switch among alternative views |

## Lab—Demo

Built-in Directives



Binding and Events

## Topic 4—Basics of Webpack and SystemJS

## Webpack

01 Webpack is essentially a code bundler.

02 It takes your code, transforms and bundles it, then returns a new “compiled” version.

03 This is bundled in Angular CLI version.

04 Webpack is a very complex tool, and most people do not need to know how every part of it works.

# Webpack

There are three key parts to a basic Webpack configuration.

**Entry**

- The starting point of your application

**Loaders**

- The transformations you want to make on your code

**Output**

- Where you want your compiled code to go

## SystemJS

01 SystemJS is a universal module loader capable of loading modules of different formats: AMD, CommonJs, globals. It works both in node and in the browser.

02 SystemJS is composed of an ES6 module loader polyfill and a compatibility layer that allow it to use different module formats.

03 SystemJS can be used independently of jspm, but it’s best to use the two tools together.

04 This is very powerful: we can use ES6 modules in production today in a transparent way. It works even in an ES5-only project (using the required syntax).

## Key Takeaways

- ✓ Angular 2 templating system provides a syntax to express the dynamic part of HTML.
- ✓ Binding is a mechanism for coordinating what users see, with application data values.
- ✓ AngularJS Built-in Directives are responsible for HTML layout.
- ✓ Webpack is essentially a code bundler. It takes your code, transforms and bundles it, and then returns a new “compiled” version. SystemJS is a universal module loader capable of loading modules of different formats.

## ? Quiz

QUIZ

1

**On the opposite side of event bindings (()) lie Angular's square-bracket syntax ([]) which signifies a________________.**

a. property binding

b. class binding

c. style binding

d. a and b

**QUIZ**

**1**

**On the opposite side of event bindings (()) lie Angular's square-bracket syntax ([]) which signifies a_______________.**

a. property binding

b. class binding

c. style binding

d. a and b

The correct answer is **a.**

**On the opposite side of event bindings (()) lie Angular's square-bracket syntax ([]) which signifies a property binding.**

**QUIZ**

2

**Angular 2 can detect when component data changes, and then automatically re-renders the view to reflect that change.**

a. True

b. False

## QUIZ

## 2

**Angular 2 can detect when component data changes, and then automatically re-renders the view to reflect that change.**

a. True

b. False

The correct answer is **a.**

**Angular 2 can detect when component data changes, and then automatically re-renders the view to reflect that change.**

**QUIZ**

**3**

## What are the code bundlers used in market?

a. Webpack

b. Grunt

c. Gulp

d. All of the above

QUIZ

3

## What are the code bundlers used in market?

a. Webpack

b. Grunt

c. Gulp

d. All of the above

The correct answer is **d.**

**Webpack, Grunt, and Gulp are the code bundlers that are used in market.**

## QUIZ 4

**What are the possible ways for two-way data binding in Angular 2?**

a. {(ngmodel)}

b. bindon-ngModel

c. Bind-src

d. a and b

QUIZ

4

**What are the possible ways for two-way data binding in Angular 2?**

a. {(ngmodel)}

b. bindon-ngModel

c. Bind-src

d. a and b

The correct answer is **d.**

**{(ngmodel)} and bindon-ngModel are the possible ways for two-way data binding in Angular 2.**

Thank You