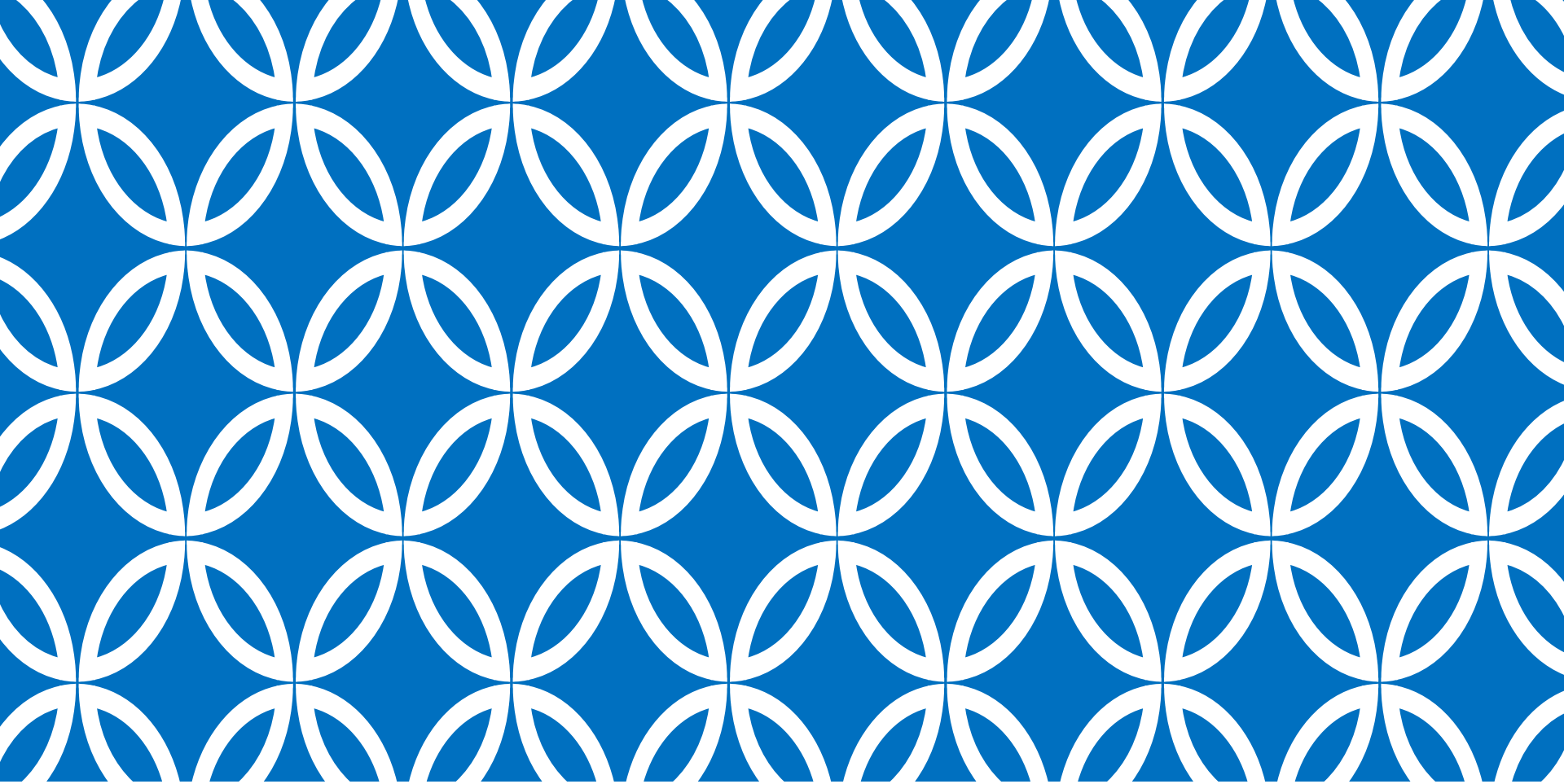
LEARN. DO. EARN

ACADGILD

# FRONT END DEVELOPMENT (WITH ANGULARJS)

Website : http://www.acadgild.com
LinkedIn : https://www.linkedin.com/company/acadgild
Facebook : https://www.facebook.com/acadgild



# Session 15 – AngularJS Directives

# Agenda – AngularJS Directives

1. Custom Directives
2. Creating Custom Directives Steps
3. Restrict Property
4. Understanding Directive Priority
5. Using templateUrl For External Templates
6. transclude Property
7. Introduction To Scopes

# Custom Directives

- Custom directives are used in AngularJS to extend the functionality of HTML.
- Custom directives are defined using **"directive"** function.
- A custom directive simply replaces the element for which it is activated.
- AngularJS provides support to create custom directives for following type of elements.
  - Element directives - Directive activates when a matching element is encountered.
  - Attribute - Directive activates when a matching attribute is encountered.
  - CSS - Directive activates when a matching css style is encountered.
  - Comment - Directive activates when a matching comment is encountered.

# Creating Custom Directives Steps

**Step 1:** You register a directive with a module. Here is an example of how that looks:

```
myapp = angular.module("myapp", []);
myapp.directive('div', function() { var directive = {};
directive.restrict = 'E'; /* restrict this directive to elements */
directive.template = "My first directive: {{textToInsert}}";
return directive; });
```

**Step 2:** Imagine that your HTML page has this HTML:

```
<div ng-controller="MyController" >
<div>This div will be replaced</div>
</div>
```

**Step 3:** Then the added directive would be activated when AngularJS finds the inner div element. Instead of this divelement, this HTML will be inserted:

```
My first directive: {{textToInsert}}
```

# Restrict Property

- The restrict property declares if the directive can be used in a template as an element, attribute, class, comment, or any combination.

| | | |
|---|---|---|
| "E": | Element | <my-directive> |
| "A": | Attribute | <div my-directive> |
| "C": | Class | <div class="my-directive“> |
| "M": | Comment: | <!-- directive: my-directive exp --> |

- Combine (e.g. "EA") for more flexibility

# Understanding Directive Priority

- Directives have priority property.
- In case there are multiple directives in an element, the directive with the higher priority gets applied first.
- If we have multiple directives on a single DOM element and if the order in which they're applied matters, we can use the priority property to order their application.

- **Note** that higher number priority directives run first. The default priority is 0 if we don't specify one.

- **Example:** For **ng-repeat**, we use a priority value of 1000.

# Using templateUrl For External Templates

- The template property has a drawback of getting the html code in the JavaScript code.
- A better approach is to keep the html code in a separate html file.
- We can achieve this by using the **templateUrl** property.

# transclude Property

- We can also move the original content within the new template through the transclude property.

- If set to true, the directive will delete the original content within the directive, but make it available for reinsertion within the template through a directive called **ng-transclude**.

# Introduction to Scopes

- Scope is an object that refers to the application model.
- Scopes are arranged in hierarchical structure which mimic the DOM structure of the application.

- New scope - We can create that inherits from our enclosing controller's scope. Here, we will have the ability to read all the values in the scopes and this scope will be shared with any other directives on our DOM element that request this kind of scope and can be used to communicate with them.

- Isolated scope – It does not any model properties from its parent. We can use this type of scope when we need to isolate the operation of our directive from the parent scope usually while creating reusable components.

# Agenda – AngularJS Scope & Controller

1. Scope Continued
2. Isolated Scope
3. Create Isolated Scope
4. One Way & Two Way Text Binding
5. Method Binding Using &
6. Using Controllers In Directives
7. Using controllerAs For Controllers
8. Using External Controller Within A Directive
9. Using ‘require’ Property For Directives
10. Using $transclude Function

# Scope Continued

**Three types of scopes are available for directives**:

- **The existing scope** of our directive's DOM element.
- **A new scope -** We can create that inherits from our enclosing controller's scope. Here, we will have the ability to read all the values in the scopes and this scope will be shared with any other directives on our DOM element that request this kind of scope and can be used to communicate with them.
- **An isolate scope** that inherits no model properties from its parent. We can use this option when we need to isolate the operation of our directive from the parent scope usually while creating reusable components.

# Isolated Scope

- Isolate scopes doesn't inherit model properties, they are still children of their parent scope. Like all other scopes, they have a $parent property that references their parent.
- When you create an isolate scope, you don’t have access to anything in the parent scope’s model by default. You can, however, specify that you want specific attributes passed into your directive. You can think of these attribute names as parameters to the function.
- To create a isolate scope we just need to write scope:{} in the DDO(directive definition Object).
- The new scope also known as Isolated scope because it is completely detached from its parent scope.

# Create Isolated Scope

- We have 3 modes in which we can communicate with parent properties within our directive for an isolated scope.
- AngularJS gives us flexibility to choose what properties we want to inherit from the parent and at what level of behavior( i.e. one way or two way binding) .
- This way we can selectively choose what we need within our directive.
- **Following are the modes**:
  1. "@" one way text binding to parent's scope properties
  2. "=" two-way binding to parents scope properties
  3. "&" One way Behavior or Method binding

# One Way & Two Way Text Binding

**One-way binding:**

- We can create text bindings which are prefixed with @.
- They are always strings.
- Whatever we write as attribute value, it will be parsed and returned as strings.
- Note that we need to keep the parent property value inside an expression. Its inside double curly braces.

**Two-way binding:**

- Two way binding are prefixed by = and they can be of any type.
- These work like actual bindings i.e. any changes to a bound value will be reflected in everywhere.

# Method Binding Using &

- The third type of binding is using &.
- This binding binds a value, object or a function.
- This type of binding returns a function for a text property or object.
- We need to call the function to access the value.

# Using Controllers in Directives

- Directives can also have controllers to attach data or functions within their scope. We just need to add the controller property to the DDO with the function that we want to be our controller.
- **Syntax:**

```
angular.module('app', [])
.directive('testDirective', function() { restrict: 'E',
Controller:
function($scope, $ element, $ attrs, $transclude) { // The controller logic goes here
}
})
```

- **Note** that we can inject the Angular injectors here like $scope, $element, $attrs and $transclude.
  1. **$element** represents the element reference of the directive.
  2. **$attrs** contains the attributes object for the current element.
  3. **$transclude** is the function to modify the DOM and play with the content within the directive if we need.

# Using controllerAs For Controllers

- We have another option in DDO to create an alias of a controller reference. Though it's very simple concept, yet its very powerful and very helpful.
- Using controllers as option, we no longer need to use the **$scope** within our controller. We can simply use the this current controller instance.
- The important point is that when we are trying to access any property or function within the html, we can use the reference as the string defined in the controllerAs option in the directive.

# Using External Controller Within A Directive

- We can also use external controller defined within the application.
- This controllers can be reused within the directives.
- We just need to specify the controller name within the directive.
- **Note** that the name of the controller is a string so we have to keep within single or double quotes.

# Using ‘require’ Property For Directives

- One directive can use controller of other directive.
- To do it we need to use the require property in the DDO and set its value to the directive whose controller we want to use.

# Using $transclude Function

- Angular gives us power to also use the $transclude function by which we can put the contents inside directive tags in directive view.
- Using $transclude function we get a lot of flexibility to decide where and how we want to modify the content.
- Note that we pass a function to the $transclude function

# Lets Discuss Assignments