AngularJS

Module 3

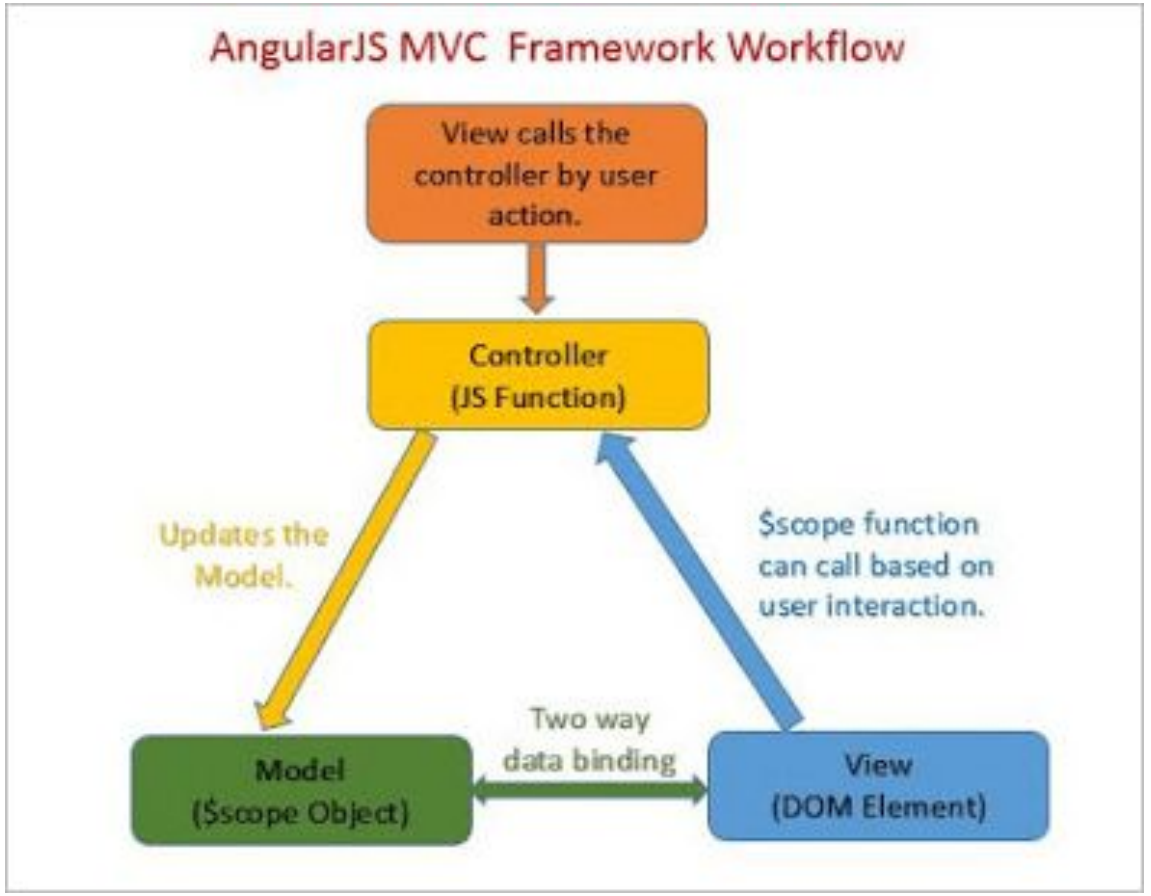
Overview of AngularJS

- **Client Side** Web Application Framework using **Javascript**
- AngularJS is developed and maintained by Google.
- **Open** Source and **Free**
- released under the MIT License
- Used to create **front-end** web framework for developing single-page applications **SPA** (Single Page application), built using HTML, JS and Ts
- originally developed by Misko Hevery and Adam Abrons at Google.

- **Separation of Concerns**
  - AngularJS is an **Model-View-Controller** framework that is similar to the JavaScript framework
  - Separate DOM from the application logic
    - Component, templates, Services
- Can create **RIA,** with including a wide range of third-party libraries, tools, and extensions
- **Cross Browser Compatible** It can be easily added to HTML pages with simply a <script> tag
- named angular after HTML as HTML contains angular brackets.

Module View Controller (MVC)

- AngularJS designs the applications in MVC style
- Divides application into 3 parts:
  - **Views**. Views are the components that display the application's user interface (UI).
  - **Models**. Model objects are the parts of the application that implement the logic for the application's data domain. Often, model objects retrieve and store model state in a database.



    - Controllers. Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI.
- In an MVC application, the view only displays information; the controller handles and responds to user input and interaction.
- View is independent of Model
- Reusable Pattern
- Decouple the Modules
- Parallel Development
- Multiple view support

Need of AngularJS in real web

- Easy to **Learn**
  - HTML, JavaScript and CSS
- It has a **Two-Way Binding** Feature (bidirectional data flow)
  - AngularJS allows for an immediate synchronization between the view and the model. If any data is altered in the model, it reflects in the view. When changes are made in the view data, the model is revised accordingly.
- Supports **SPA** features
  - faster website transition
- Has a **Declarative UI**
- **Supported** by large Community and Google
- supported by different **browsers**
- Incorporates the concepts of various languages including JavaScript and server-side languages.

- It is a Powerful Framework
  - robust solution for faster front-end development
  - It has multiple features such as **MVC pattern, directives, and dependency injection**.
  - It is a common platform among developers because it is **free**ly available.
  - This allows developers to expand the HTML syntax and build client-side applications.
- Real-Time **Testing**
  - end-to-end and unit testing

- AngularJS extension that is the Angular apps are built using the **TypeScript** language, which eliminates the errors while writing the codes and ensures high performance and security.
- The **time investment** in Angular for the projects is less as compared to other web development frameworks. The Angular CLI tool allows the developers to create initial projects, perform testing, and integrate features in the same.
- Reduce development time as Angular uses HTML to define the User-interface of an application, which is less complex as compared to JavaScript.

Environment Setup for AngularJS

- Download the desired version of AngularJS (https://angularjs.org/. OR use CDN
  - https://angularjs.org/

Including script from CDN:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularJS/1.7.2/angular.min.js"
type="text/javascript">
</script>
```

Including script from local machine:

```
<script    src=" scripts/angular.js"    type="text/javascript">

</script>
```

- AngularJS projects often use Node.js for development tools (https://nodejs.org/en) and npm (Node Package Manager) for managing dependencies.
- To Create Project
  - Create a new directory for your AngularJS project and navigate to it using the terminal or command prompt.
  - Install AngularJS

    npm i @angular/core (OR npm install angular@1.7.9)

    - This step generates a skeleton application and includes all dependencies
    - Include html and other file

Sample Code

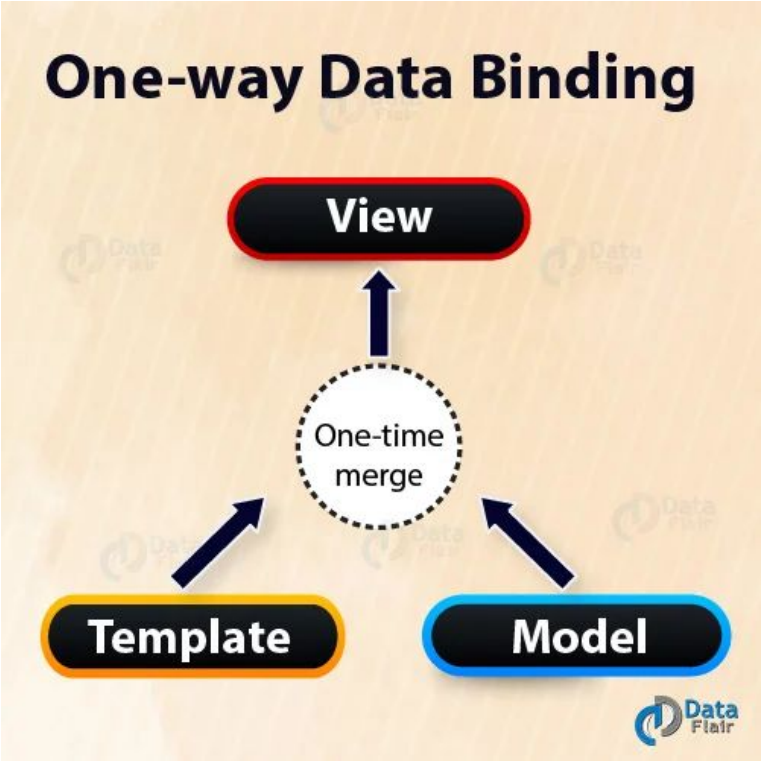

```
<!DOCTYPE html>
<html>
    <head>
        <title>My First Program in AngularJS - Wikitechy</title>
        <script src="https://ajax.googleapis.
            com/ajax/libs/angularjs/1.5.6/angular.min.js"></script>
    </head>

    <body ng-app="">
        Enter Text : <input ng-model="text">
        <h2 ng-bind="text"></h2>
    </body>
</html>
```

Data Binding

- The **Data Binding** refers to the *synchronization* of data between the model and view.
- Binding is an important concept as it acts as a bridge between the view and the logic of the AngularJS app
- Data Binding In AngularJS is achieved by using Directives.
- There are 2 major components of Data Binding in AngularJS:
  - **Model:** It is responsible for maintaining data in the application.
  - **View:** It is the HTML container where the app is displayed to the user.
- AngularJS provides two types of Data Binding:
  - One-way data binding
  - Two-way data binding

One Way Data Binding:

- In one-way data binding, the flow of data is in **one direction** only i.e. from model to view.
- A value is taken from the data model, inserted in an HTML element, and displayed to the user.
- But there is no way to update the model according to the input given by the user which means that the data can’t flow from the view to the model.
- One-Way Data Binding can be achieved by:
  - Interpolation
  - Using ng-bind directive



Interpolation/ expressions

- AngularJS expressions can contain literals, operators or variables, unlike the traditional JavaScript.
- add the ng-app directive, else the expression will be displayed as it is, without being solved.
- AngularJS expressions could be written inside double curly braces or can be written inside a directive.

Syntax:

Inside curly braces:

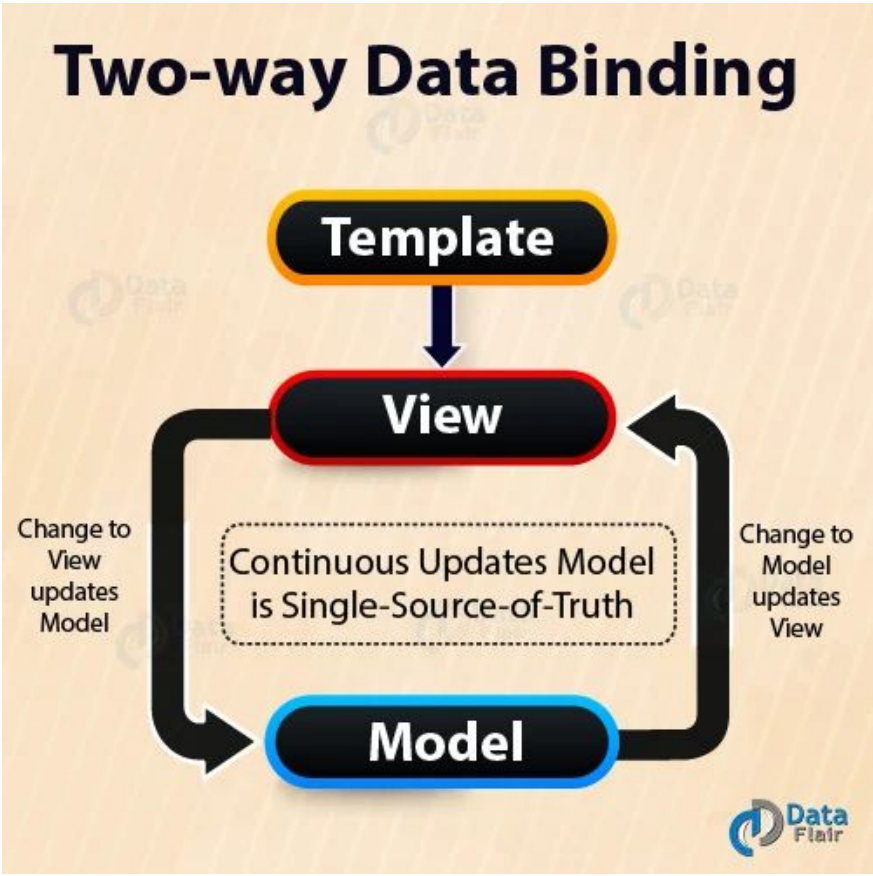
```
<div ng-app=””>
<p>My first expression in Angular JS: {{3+3}}</p>
  </div>
```

Inside directive:

<div ng-controller=”Name of your controller”>

<element ng-bind="your expression"></element>

- **Two way Data binding**
  - the flow of data is bidirectional i.e the data can flow from the model to the view as well as from the view to the model.
  - when the data in the model changes, the changes are reflected in the view and when the data in the view changes the model is also updated
  - Two-way data binding is achieved by using the **ng-model** directive, transfers data from the controller to the view and vice versa.



Datatype

- Numbers

  <p>The value of 5 times 10 is : {{5*10}}<p>

- Strings
  - Strings can be initialized using ng-init directive or ng-controller directive. The concatenation of strings is also possible when the + operator is used within the expression.
  - Strings also could be used as expressions within double curly braces or use ng-bind directive just like the AngularJS numbers.

Inside curly braces:

<div ng-app=”” ng-init=”first string variable name=’your first string’; second string variable name=’your second string’ ”>

My first string expression in Angular JS: {{ first string variable name + second string variable name }}

Inside directive:

```
<div ng-app=”” ng-init=” first string variable name=’your first string’;
second string variable name=’your second string’”>
<p>My first string expression in Angular JS:<span ng-bind=
" first string variable name + second string variable name "></span>
</p>
</div>
```

- **AngularJS Objects**
  - AngularJS objects behave the same way in which the JavaScript objects behave.
  - The items within an object could be accessed using the dot operator.

Inside curly braces

```
<div ng-app="" ng-init="your object name={first variable name='your first value',second variable name='your second value'}">

<p>My first object in Angular JS: {{ your object name.second variable name }}</p>

</div>
```

Inside directive

```
<div ng-app=”” ng-init=”your object name={first variable name=’your first value’,second variable name=’your second value’}”>

<p>My first object in Angular JS:<span ng-bind=" your object name.second variable name "></span>

</p>

</div>
```

- **AngularJS Arrays**
  - AngularJS arrays behave the same way in which the JavaScript arrays behave.
  - The items within an array could be accessed by denoting the value's index number within square braces.
  - indexing starts from zero always

Inside curly braces:

```
<div ng-app="" ng-init="your array name=[your first value,your secondValue]">
<p>My first array in Angular JS: {{ your array name[1] }}</p>
  </div>
```

Inside directive:

<div ng-app=”” ng-init=”your array name=[your first value,your second value]”>

My first array in Angular JS:

note, AngularJS:

- Does not support conditionals, loops, and exceptions in expressions.
- Does not support function declaration (even inside the ng-init directive) in expressions.
- Does not support bitwise, comma, void and new operator in expressions.
- Ignores the null or undefined properties in expressions.
- Expressions are evaluated belonging to the scope object and not the global window.

Create Project

An AngularJS application consists of following three main parts −

- ng-app− Defines and links application to HTML.
- ng-model− Binds the values of application data to HTML input controls.
- ng-bind− Binds the Application data to HTML tags.

Step 1: Load framework

<Script> tag.

Step 2: Define Application using ng-app directive

```
<div ng-app = "">

//code
  </div>
```

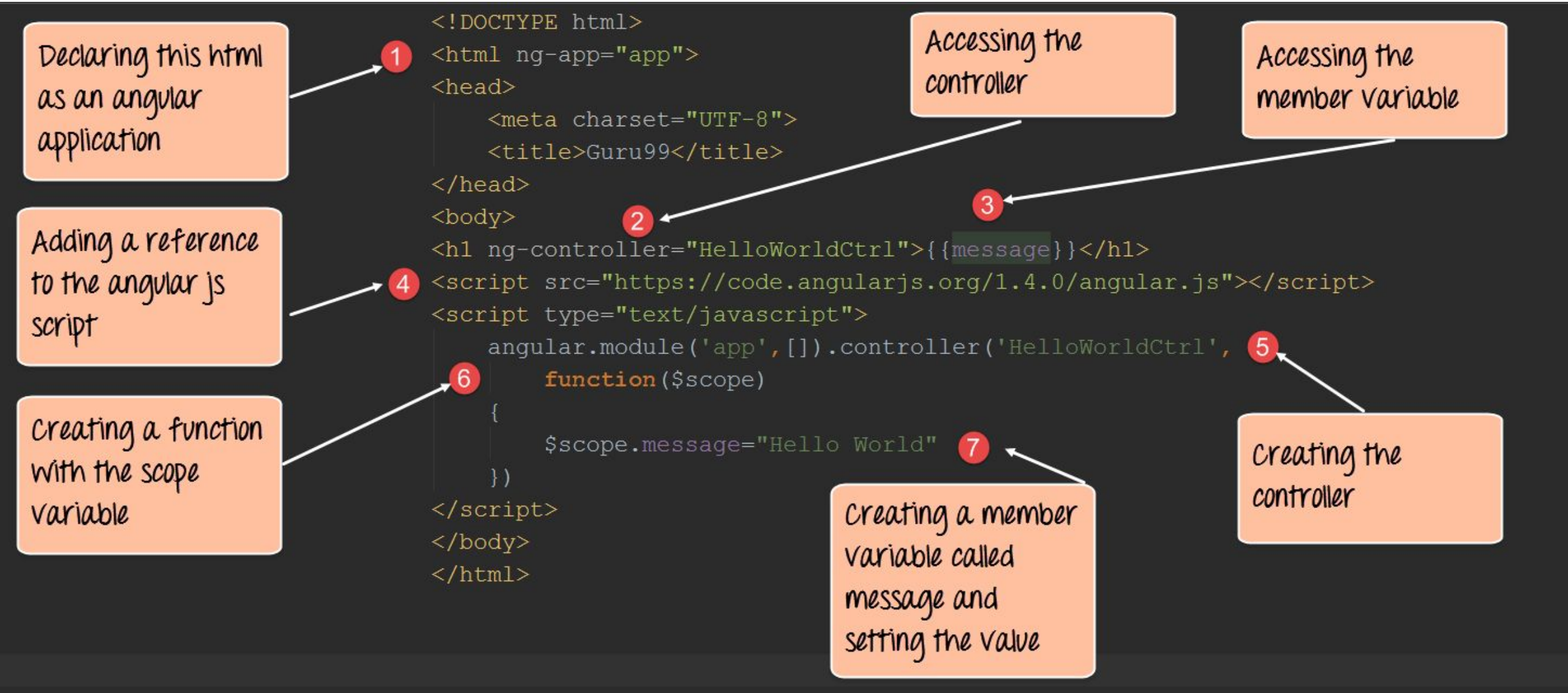
Step 3: Define a model name using ng-model directive

```
  <p>Enter Text: <input type = "text"  ng-model = "name"></p>
```

Step 4: Bind the value of above model defined using ng-bind directive.

```
  <p>Hello<span ng-bind = "name"></span></p>
```

Sample Code


Declaring this html as an angular application
Adding a reference to the angular js script
Creating a function with the scope variable
Accessing the controller
Accessing the member variable
Creating the controller
Creating a member variable called message and setting the value
1
2
3
4
5
6
7
<!DOCTYPE html>
<html ng-app="app">
<head>
<meta charset="UTF-8">
<title>Guru99</title>
</head>
<body>
<h1 ng-controller="HelloWorldCtrl">{{message}}</h1>
<script src="https://code.angularjs.org/1.4.0/angular.js"></script>
<script type="text/javascript">
angular.module('app',[]).controller('HelloWorldCtrl',
function($scope)
{
$scope.message="Hello World"
})
</script>
</body>
</html>

Directives

- Tells the DOM what action (controller) need to be done.
- Directives are used like Attributes
- 2 types
  - Built in Directives - a set of built-in directives which offers functionality to your applications.
  - Custom Directives - lets you define your own directives.

Built in Directives

- Start as ng-*
- Eg:
  - **ng-app** − This directive defines and links an AngularJS application to HTML. start the AngularJS application.
  - **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
  - **ng-bind** − This directive binds the AngularJS Application data to HTML tags.
  - ng-value - bind value with a tag

- ng-controller - to associate with a controller
- **ng-show, ng-hide** - show or hide any control from DOM
- ng-init - initialise any variable
- ng-repeat - used for looping
- **ng-click** - is like onclick(), is based on an event
- ng-required - used for form validation
- **ng-disabled** - enable or disable the control at runtime

Custom Directives

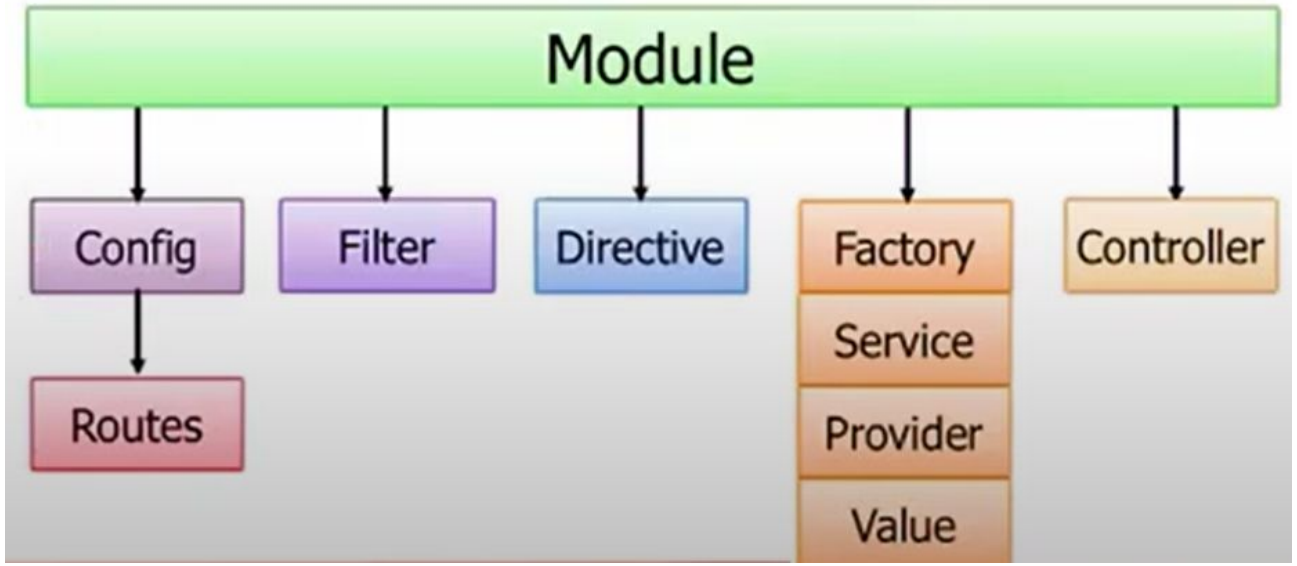
- To implement our own directives.
- AngularJS enables us to create new directives so that we can encapsulate and simplify DOM manipulation.
- Custom directives are used to extend the functionality of HTML.
- A custom directive simply replaces the element for which it is activated.
- Custom directive can be for the following types.
    - **Element directives:** Directive activates when a matching element is encountered. Restrict mode is defined by “E”.
        - **Example:** <ng-directives></ng-directives>

- **Attribute**: Directive activates when a matching attribute is encountered. Restrict mode is defined by “A”.
  - Example: <span ng-directive>
- **CSS**: Directive activates when a matching css style is encountered.
  - Restrict mode is defined by “C”.
  - Example:
- **Comment**: Directive activates when a matching comment is encountered.
  - Restrict mode is defined by “M”.
  - Example <!-- directive: ng-directive -->

AngularJS Modules

- In AngularJS, a module defines an application.
- It is a container for the different parts of your application like controller, services, filters, directives etc.
- A module is used as a Main() method. Controller always belongs to a module.
- Modules are used to separate logic such as services, controllers, application etc. from the code and maintain the code clean
- A module is a collection of providers, services, directives etc., and optionally config and run blocks which get applied to the application during the bootstrap process.

- Modules are said to be the containers for different parts of an application



- 2 type of Modules
  - Application Module − used to initialize an application with controller(s).
  - Controller Module − used to define the controller.

- Creating a Module in AngularJS:

  var app = angular.module("Module-name", []);

  - [], we can add a list of components needed but we are not including any components in this case.

  *** ***

  The code in which the module is required.

  *** ***

- **Adding a Controller:**

  app.controller("Controller-name", function($scope) {

  $scope.variable-name= "";

  });

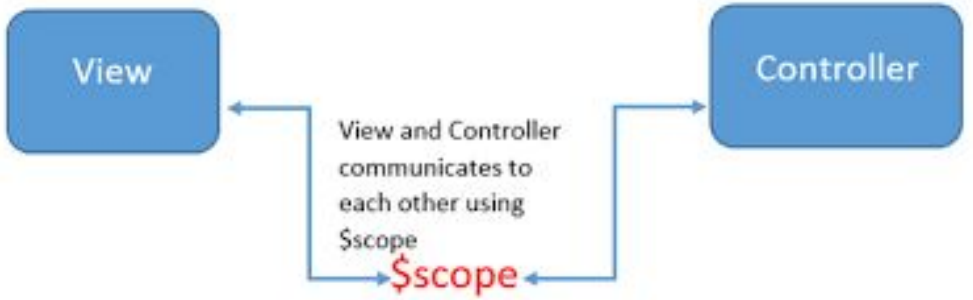
  - Here, we can add any number of variables in the controller and use them in the HTML files, and the body of the tag in which the controller is added to that tag by writing:

Consolidating……

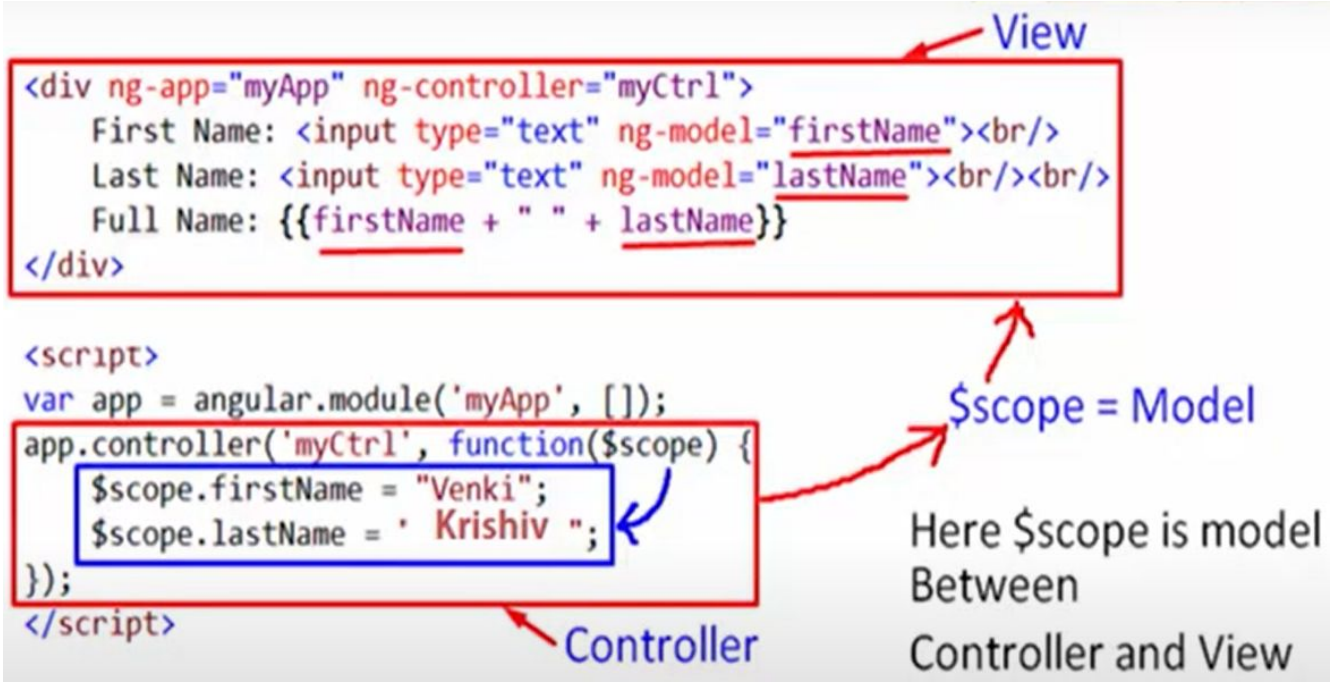
```
<body>
    <div ng-app="Module-name">
        <div ng-controller="Controller-name">
            {{variable-name}}
        </div>

    <!-- This wont get printed since its not part of the div in which
        controller is included -->
        {{variable-name}}
    </div>
</body>
```

- Modules and controllers can be in the same file along with the HTML file which requires it however we may want to use this module in some other file.
- Hence this will lead to redundancy so we will prefer to create Module, Controller, and HTML files separately.
- The Module and Controller are to be stored by using .js files.

AngularJS scope

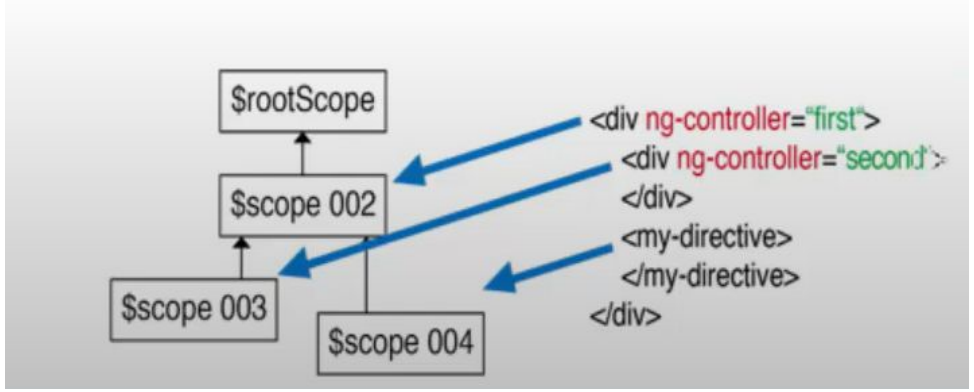


- Scope is an object that refers to application model
- It is used to bind between the view and the controller
- Scope is accessible to both view and controller



AngularJS controllers

- Controllers **control** the application
- controls the f**low of data** in that application
- These are the **JavaScript Objects** that hold attributes and functions.
- They are defined by a **JavaScript constructor function** that is used to augment the AngularJS Scope.
- By using specific controller constructor function Angular will initiate a **new constructor object** and a **new child scope** is available as injectable parameter to the Controllers constructor function.
- Scopes are organized in hierarchical structure which is initiated the DOM structure of the application



- It is defined by ng-controller.
- It uses $scope as a parameter. $scope objects communicate with the view and expose the model to the view.

```
<div ng-app = "" ng-controller = "controller_name">
//code
</div>
```

- Define in 2 ways
  - as an application module
  - as a JavaScript function.

- **An Application Module**

```
angular.module("myapp", [])
.controller("appController", function($scope) {
// definition of controller
  } );
```

- **JavaScript Function**

```
function controllerAsFunction($scope){
//definition of controller
}
```

- Controllers are also in external files (see in Module)

- Controllers are used to **set up the initial state** of the `$scope` object.
- Add behavior to the `$scope` object.
- When an application is created we need to set up the initial state for the AngularJS $scope.
- You set up the initial state of a scope by attaching properties to the $scope object.
- The properties contain the view model (the model that will be presented by the view).
- All the $scope properties will be available to the template at the point in the DOM where the Controller is registered.
- Also behaviour can be added to the Scope Object

Do not use controllers to:

- Manipulate DOM — Controllers should contain only business logic. Putting any presentation logic into Controllers significantly affects its testability. AngularJS has databinding for most cases and directives to encapsulate manual DOM manipulation.
- Format input — Use AngularJS form controls instead.
- Filter output — Use AngularJS filters instead.
- Share code or state across controllers — Use AngularJS services instead.
- Manage the life-cycle of other components (for example, to create service instances).

AngularJS Filters

- Filter are functions that are used to format, transform, and filter data in expressions.
- They allow you to modify the data before it is displayed to the user.
- Filters can be applied in the view templates to format data without changing the underlying model
- Filters can be applied to expressions in view templates using the following syntax:

  {{ expression | filter }}

  also {{ expression | filter1 | filter2 | ... }}

- Filtering the data to show the data in a customized format
  - Used to modify the data representation
  - To specify search criteria
  - Can be clubbed with expressions
- Eg:
  - Uppercase/ Lowercase
  - Currency
  - OrderBy
  - Filter

  {{ name | uppercase }}

  {{ amount | currency }}

  {{ date | date:'MM/dd/yyyy' }}

- Custom Filters
  - A custom filter can be created by registering a filter factory function in a module.
  - They are functions that return a function
  - Use the filter function to create a custom filter
  - The AngularJS filter function should be a pure function, meaning it should produce the same result as per the given input, and it should not have any effect on an external state.

AngularJS Forms

- **AngularJS** performs **form validation** on the client side
- 2 ways
  - AngularJS adds CSS classes to forms and input fields depending on their states.
    - ng-untouched The field has not been touched yet
    - ng-touched The field has been touched
    - ng-pristine The field has not been modified yet
    - ng-dirty The field has been modified
    - ng-valid The field content is valid
    - ng-invalid The field content is not valid

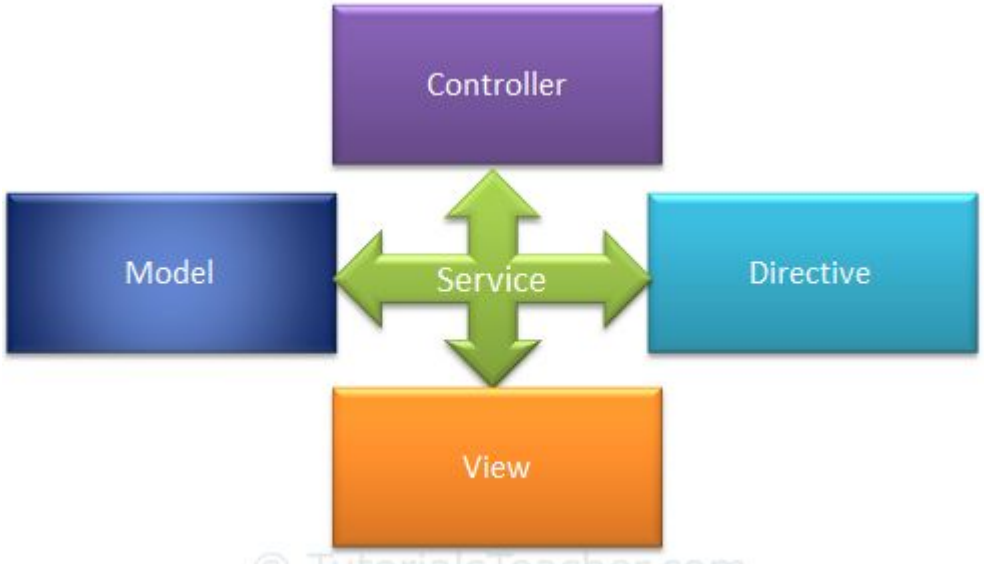
- Form State and Input State
  - AngularJS is constantly updating the state of both the form and the input fields
    - $untouched The field has not been touched yet
    - $touched The field has been touched
    - $pristine The field has not been modified yet
    - $dirty The field has been modified
    - $invalid The field content is not valid
    - $valid The field content is valid

AngularJS Services

- In AngularJS, Services are objects that provide some sort of service that can be reused with in an angular application.
- Instead of writing the code for every component separately to perform the same function, Angular service is used to write the code once and inject it into every component
- Like other JS objects they too have has properties and methods
- 2 types
  - Built in Services
    - AngularJS has lot of built in services
    - Eg: $http, $log
  - Custom Services

- Why do we need services in an angular application
  - The primary responsibility of the controller is to build the model for the view. The controller should not be doing too many things if the logic with in your controller, is becoming too large or too complex, then use service.
  - Reusability :
    - In a service you usually have a logic that you want to reuse with in your entire application.
  - Dependency Injection : Another benefit of services, is that, they can simply be injected into controllers or other services that need them.
  - Testability : Since services are injected into controllers or other services that need them, it becomes very easy to test them.

- Services can be used to organize and share code across your app.
- AngularJS services are: Lazily instantiated – AngularJS only instantiates a service when an application component depends on it.
- AngularJS services interact with the controller, model or custom directives.
- However, some services interact with view (UI) also for UI specific tasks
- They follow Singleton Patterns



$http service

- Used to make HTTP requests to remote server
- $http service is used to send or receive data from the remote server using browser's XMLHttpRequest
- it has a single input parameter i.e a configuration object.

```
$http({
    method: 'GET',
    url: 'EmployeeService.asmx/GetAllEmployees'
});
```

- includes following shortcut methods

Method	Description
$http.get()	Perform Http GET request.
$http.head()	Perform Http HEAD request.
$http.post()	Perform Http POST request.
$http.put()	Perform Http PUT request.
$http.delete()	Perform Http DELETE request.
$http.jsonp()	Perform Http JSONP request.
$http.patch()	Perform Http PATCH request.

- $http service returns a promise object.
- This means the functions are executed asynchronously and the data that these functions return may not be available immediately.

Routing

- In AngularJS we build Single Page Application (SPA) with AngularJS.
- It is a web app that loads a single HTML page and dynamically updates that page as the user interacts with the web app.
- AngularJS **ngRoute** module provides routing
- download angular-route.js script or use CDN
- ngRoute module routes your application to different pages without reloading the entire application.

[Install AngularJS http server if required

Install server
npm install -g http-server

Move to the project folder

To run server
http-server
]

1. load the ngRoute module in your AngularJS application by adding it as a dependent module
2. $routeProvider is used to configure the routes.
    - use the ngRoute config() to configure the $routeProvider.
    - The config() takes a function which takes the $routeProvider as parameter and the routing configuration goes inside the function.
    - $routeProvider has a simple API, accepting either the when() or otherwise() method.
3. application needs a container to put the content provided by the routing
    - ngView directive is used to display the HTML templates or views in the specified routes.
    - Every time the current route changes, the included view changes with it according to the configuration of the $route service.

AngularJS dependency injection

- Dependency Injections are
  - Software Design Pattern
  - Create components with Single Responsibility,
    - Reusability
    - Loosely coupled code
    - Maintainability
    - testing becomes easy
- It allows the AngularJS application to divide into smaller modules
  - One injected to another
- Modularization

- DI are implemented in AngularJS as
  - Value
    - Injecting a Value
  - Factory
    - Injecting Values into a Factory
  - Service
    - Injecting Values into a Service
  - Providers
    - Configuring a Provider
  - Constants
    - Dependencies Between Modules

- Value

```
myMod.value("numberValue", 10);
myMod.value("stringValue", "aaa");
myMod.value("objectValue", { field1 : 123, field2 : "abc"} );
```

- Constant
  - Config Phase of AngularJS has some restrictions for injected values, as you cannot inject values into the module.config() function.
  - Instead constants are used to pass values at config phase.
  - To define a constant value we use the constant() function of the defined module.
  - Then for the constant parameter we provide a constant value.

- Factory
  - factory is a function that is used to return a value.
  - This function creates the value on demand whenever a service or controller needs any value injected from the factory.
  - AngularJS applications normally use a factory function to calculate and return a value.
  - difference between an AngularJS service and factory is that a service is a constructor function and a factory is not. That is why, in the case of a factory, we return an object literal instead of using this.

- Service
  - Services are created by using service() function on a module and then injecting it into controllers.
- Providers
  - In AngularJS, a provider is used to internally create other values or services or factory during config phase (the phase during which AngularJS bootstraps itself).
  - Provider can be considered to be the most flexible form of factory you can create.
  - Provider is a special factory method with a get() function which is used to return the value or service or factory

- **Controllers** are to do with view related business logic.
- **Services**, on the other hand, are to do with reusable business logic independent of the views.
- **Factory** is that a service is a constructor function and a factory is not. That is why, in the case of a factory, we return an object literal instead of using this

Built-in Helper Functions

angular.copy
angular.equals
angular.forEach
angular.fromJson
angular.identity
angular.isArray
angular.isDate
angular.isDefined
angular.isUndefined
angular.isElement
angular.isFunction
angular.isNumber
angular.isObject
angular.isString
angular.merge
angular.noop
angular.toJson

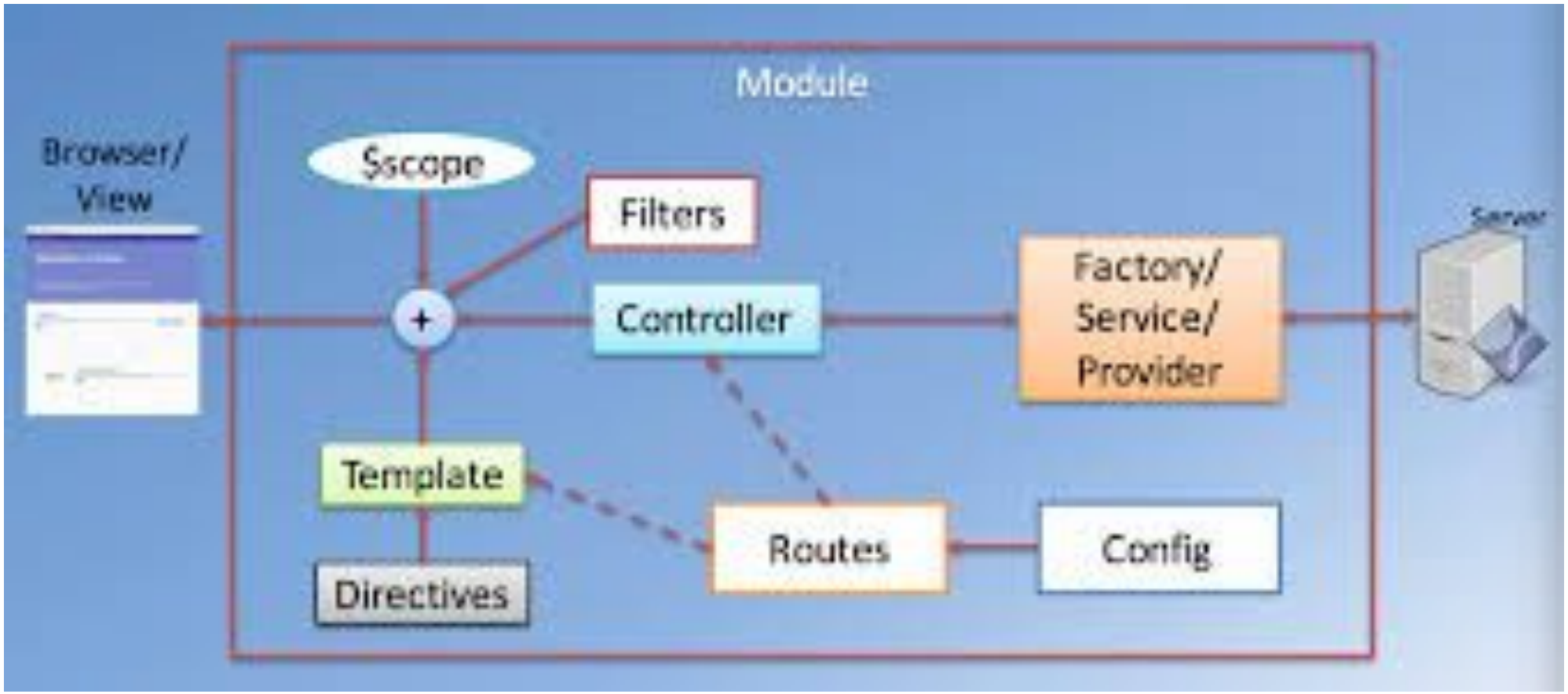
Using Angular JS with Typescript

- enhance the development experience by adding static typing and other features provided by TypeScript.

```
npm install -g typescript
```

```
npm install angular
```

```
npm install --save-dev @types/angular
```


Module
Browser/
View
$scope
Filters
Controller
Factory/
Service/
Provider
Server
Template
Routes
Config
Directives