What is AngularJS ?

AngularJS is an open-source JavaScript framework developed by Google for building dynamic web applications. It was initially released in 2010 and gained significant popularity among web developers due to its ability to create single-page applications (SPAs) with a more structured and organized approach.

AngularJS follows the Model-View-Controller (MVC) architectural pattern, which helps separate concerns and makes it easier to manage complex applications. Here's a brief overview of the components in AngularJS:

1. **Model**: This represents the data and the business logic of the application. In AngularJS, the model is usually defined using JavaScript objects or JSON data.

2. **View**: The view is responsible for rendering the data and presenting it to the user. In the context of AngularJS, the view is typically defined using HTML templates with additional directives and expressions provided by the framework.

3. **Controller**: The controller acts as an intermediary between the model and the view. It handles user input, updates the model, and ensures that the view reflects the current state of the model. Controllers are defined using JavaScript functions in AngularJS.

AngularJS introduced several concepts that were innovative at the time, such as data binding (automatic synchronization between the model and the view), dependency injection (managing and injecting dependencies into components), and directives (extending HTML with custom behavior and functionality).

Concepts & Charactristics Of AngularJS

AngularJS introduced several key concepts that were instrumental in its popularity and success. These concepts helped developers build dynamic and maintainable web applications. Here are some of the core concepts of AngularJS:

1. **Two-Way Data Binding**: AngularJS introduced a powerful feature called two-way data binding, which allows automatic synchronization between the model (data) and the view (UI). When the model changes, the view is updated automatically, and vice versa. This reduces the need for manual DOM manipulation.

2. **Controllers**: Controllers in AngularJS are JavaScript functions that contain the business logic of a specific part of the application. They act as intermediaries between the model and the view. Controllers manipulate the model and respond to user interactions, updating the view accordingly.

3. **Directives**: Directives are markers on a DOM element that tell AngularJS to attach a certain behavior or functionality to that element. They enable the creation of reusable UI components and custom behaviors. Common directives include `ng-repeat` (for iterating over arrays), `ng-model` (for binding input elements to data), and `ng-show`/`ng-hide` (for toggling element visibility).

4. **Templates**: Templates are HTML files that define the structure of the user interface. AngularJS templates can include directives, which allow dynamic rendering and binding of data to the UI.

5. **Scope**: The scope is a JavaScript object that represents the context in which expressions are evaluated. Scopes serve as a bridge between controllers and views, enabling data binding and communication between the two.

6. **Filters**: Filters are used to format and transform data displayed in the view. They can be applied to expressions in templates to modify the way data is presented to the user.

7. **Services**: Services in AngularJS are singleton objects that provide specific functionality and can be injected into controllers, directives, and other components. Common examples of services include the `$http` service for making HTTP requests and the `$rootScope` service for managing the top-level scope.

8. **Dependency Injection**: AngularJS employs dependency injection to manage and provide instances of various components (controllers, services, etc.). This promotes modularization and testability of code by decoupling components and their dependencies.

9. **Routing**: AngularJS introduced the concept of routing to create single-page applications (SPAs) with different views. The `ngRoute` module provides tools for defining routes and managing navigation within an application.

10. **Modules**: AngularJS applications are typically organized into modules. Modules encapsulate related components, making the application more modular, maintainable, and testable.

These concepts collectively contributed to the development of dynamic and interactive web applications using AngularJS.

Expression in AngularJS

In AngularJS, expressions are a way to bind dynamic values to the HTML templates. They allow you to interpolate values, perform calculations, and manipulate data directly within the template. Expressions are typically enclosed in double curly braces `{{ }}`. Here's how expressions work for different data types:

1. **Numbers**:

```
<p>5 + 3 = {{ 5 + 3 }}</p>
<p>{{ 10 / 2 }}</p>
```

2. **Strings**:

```
<p>{{ 'Hello, ' + 'AngularJS' }}</p>
<p>{{ 'Length: ' + 'OpenAI'.length }}</p>
```

3. **Objects**: Assuming you have an object defined in your scope:

```
$scope.person = {     firstName: 'John',     lastName: 'Doe' };
```

You can use its properties in expressions:

```
<p>{{ person.firstName }} {{ person.lastName }}</p>
```

4. **Arrays**: Assuming you have an array defined in your scope:

```
$scope.numbers = [1, 2, 3, 4, 5];
```

You can use array elements in expressions:

```
<p>{{ numbers[0] }} {{ numbers[2] }}</p>
```

5. **Functions**: You can also call functions defined in your scope from expressions:

```
$scope.calculateSum = function(a, b) {     return a + b; };
```

```
<p>3 + 7 = {{ calculateSum(3, 7) }}</p>
```

6. **Conditional Expressions**: AngularJS expressions also support ternary conditional expressions:

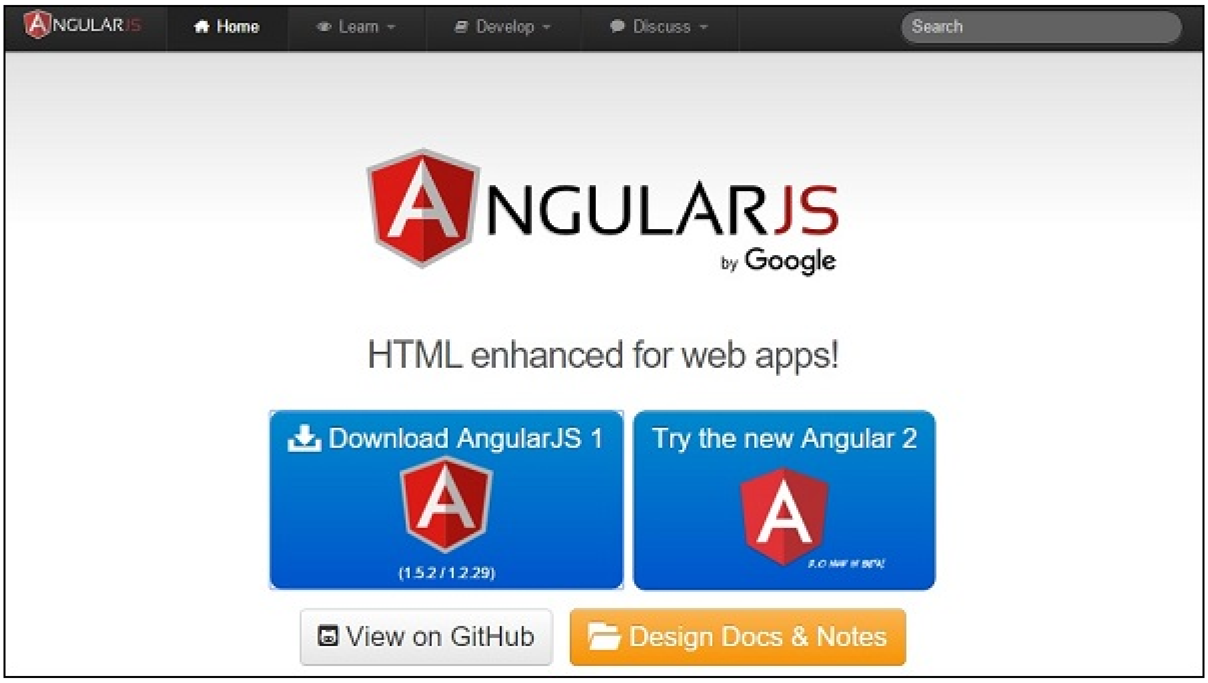
```
$scope.isEven = function(num) {     return num % 2 === 0; };
```

```
<p>{{ 8 }} is even: {{ isEven(8) ? 'Yes' : 'No' }}</p>
```

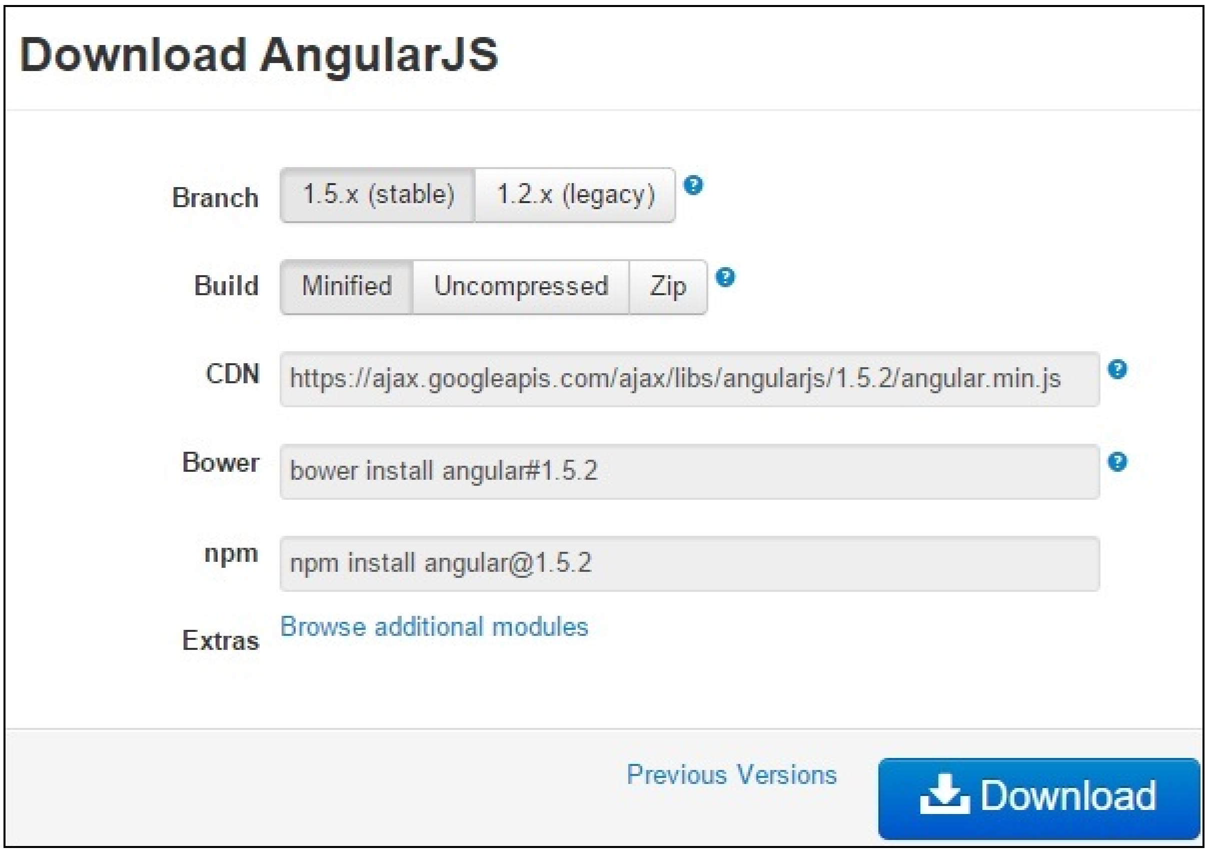
- It's important to note that AngularJS expressions are evaluated within the context of the current scope. The expressions are not full JavaScript; they are a subset designed specifically for use within AngularJS templates.
- Expressions do not support control flow statements (like loops or conditional statements) and cannot directly access global variables or execute arbitrary JavaScript code.
- Expressions are meant for relatively simple calculations and data binding within the context of templates.

Setting up Environment

- This chapter describes how to set up AngularJS library to be used in web application development. It also briefly describes the directory structure and its contents.
- When you open the link https://angularjs.org/, you will see there are two options to download AngularJS library –



- **View on GitHub** – By clicking on this button, you are diverted to GitHub and get all the latest scripts.
- **Download AngularJS 1** – By clicking on this button, a screen you get to see a dialog box shown as –



Download AngularJS

Branch: 1.5.x (stable) | 1.2.x (legacy)

Build: Minified | Uncompressed | Zip

CDN: `https://ajax.googleapis.com/ajax/libs/angularjs/1.5.2/angular.min.js`

Bower: `bower install angular#1.5.2`

npm: `npm install angular@1.5.2`

Extras: Browse additional modules

This screen gives various options of using Angular JS as follows –

- **Downloading and hosting files locally**
  - There are two different options : Legacy and Latest. The names themselves are self-descriptive. The Legacy has version less than 1.2.x and the Latest come with version 1.3.x.
  - We can also go with the minimized, uncompressed, or zipped version.
- **CDN access** – You also have access to a CDN. The CDN gives you access to regional data centers. In this case, the Google host. The CDN transfers the responsibility of hosting files from your own servers to a series of external ones. It also offers an advantage that if the visitor of your web page has already downloaded a copy of AngularJS from the same CDN, there is no need to re-download it.

> We are using the CDN versions of the library throughout this tutorial.

Example

Now let us write a simple example using AngularJS library. Let us create an HTML file *myfirstexample.html* shown as below –

```
<!doctype html>
<html>
   <head>
      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.5.2/angular.min.js"></script>
   </head>

   <body ng-app = "myapp">
      <div ng-controller = "HelloController" >
         <h2>Welcome {{helloTo.title}} to the world of Tutorialspoint!</h2>
      </div>

      <script>
         angular.module("myapp", [])

         .controller("HelloController", function($scope) {
            $scope.helloTo = {};
            $scope.helloTo.title = "AngularJS";
         });
      </script>
    </body>
</html>
```

Let us go through the above code in detail –

Include AngularJS We include the AngularJS JavaScript file in the HTML page so that we can use it –

```
<head>
   <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
   </script>
</head>
```

You can check the latest version of AngularJS on its official website.

Point to AngularJS app

Next, it is required to tell which part of HTML contains the AngularJS app. You can do this by adding the ng-app attribute to the root HTML element of the AngularJS app. You can either add it to the html element or the body element as shown below –

```
<body ng-app = "myapp">
</body>
```

View

```
<div ng-controller = "HelloController" >
   <h2>Welcome {{helloTo.title}} to the world of Tutorialspoint!</h2>
</div>
```

ng-controller tells AngularJS which controller to use with this view. *helloTo.title* tells AngularJS to write the model value named helloTo.title in HTML at this location.

Controller

The controller part is –

```
<script>
   angular.module("myapp", [])
      .controller("HelloController", function($scope) {
      $scope.helloTo = {};
      $scope.helloTo.title = "AngularJS";
   });
</script>
```

This code registers a controller function named HelloController in the angular module named *myapp*. The controller function is registered in angular via the angular.module(...).controller(...) function call.

The $scope parameter model is passed to the controller function. The controller function adds a *helloTo* JavaScript object, and in that object it adds a *title* field.

Execution

Save the above code as *myfirstexample.html* and open it in any browser. You get to see the following output –

```
Welcome AngularJS to the world of Tutorialspoint!
```

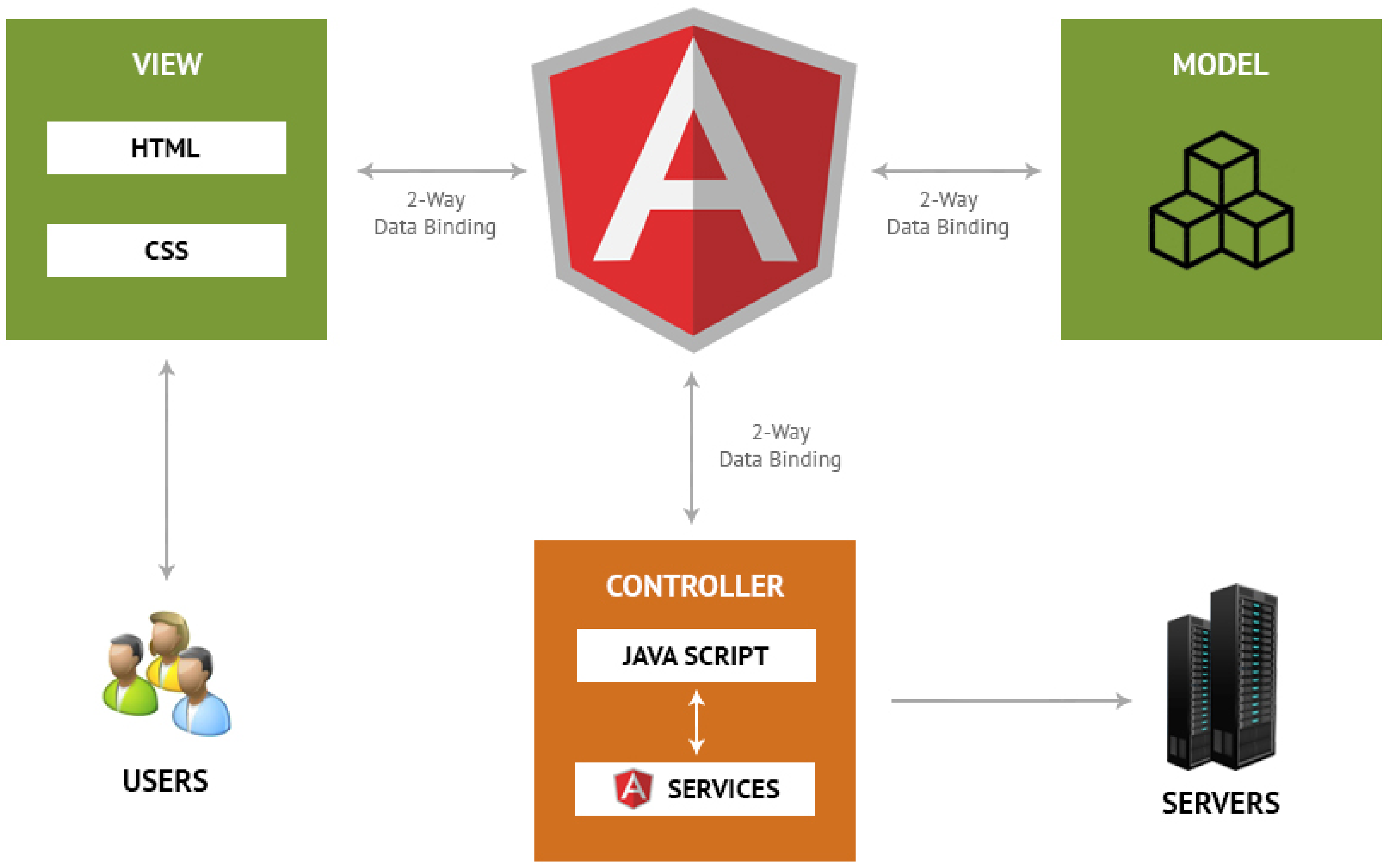
What happens when the page is loaded in the browser ? Let us see –

- HTML document is loaded into the browser, and evaluated by the browser.
- AngularJS JavaScript file is loaded, the angular *global* object is created.
- The JavaScript which registers controller functions is executed.
- Next, AngularJS scans through the HTML to search for AngularJS apps as well as views.
- Once the view is located, it connects that view to the corresponding controller function.

- Next, AngularJS executes the controller functions.
- It then renders the views with data from the model populated by the controller. The page is now ready.

Understanding MVC architecture

In AngularJS, the Model-View-Controller (MVC) architectural pattern is a fundamental concept that helps structure and organize your application code. AngularJS's interpretation of MVC is often referred to as MVVM (Model-View-ViewModel), but the principles are similar. Here's how MVC/MVVM is applied in AngularJS:



Model:

1.
    - In AngularJS, the model represents the data and the business logic of your application. It's the actual information that your application works with, such as user input, server responses, and other data.
    - The model in AngularJS is usually defined within the controllers or services. Controllers manage and manipulate the data that forms the model.
    - The `$scope` object in AngularJS is a key player here. It acts as the glue between the controller and the view, allowing data binding and communication between the two.
2. **View**:

- The view is the user interface that presents the data from the model to the user. In AngularJS, the view is typically defined using HTML templates enriched with AngularJS directives and expressions.
- Directives provide custom behaviors and enable data binding, which establishes a connection between the model and the view.
- AngularJS allows you to create dynamic views that update automatically when the underlying data changes.

3. **Controller**:

    - Controllers in AngularJS act as intermediaries between the model and the view. They handle user interactions, update the model, and ensure that the view accurately reflects the current state of the model.
    - Controllers contain the business logic that processes data and prepares it for presentation in the view.
    - The `$scope` object plays a significant role in controllers. It allows controllers to expose data and functions to the view, enabling data binding and interaction.

In the context of AngularJS, you can visualize the flow of MVC/MVVM like this:

1. **Model**: The application data is stored in JavaScript objects within controllers or services. These objects represent the state of the application.

2. **View**: HTML templates are used to define how the data from the model should be presented to the user. AngularJS directives and expressions are used to bind data and define behavior within the view.

3. **Controller**: Controllers contain the logic that manipulates the model data. They respond to user interactions and update the model accordingly. The updated model data is then automatically reflected in the view due to data binding.

AngularJS Directives

In AngularJS, directives are a powerful feature that extends HTML with new attributes or elements. They allow you to create reusable components and enhance the behavior of your application's UI. Directives play a significant role in separating concerns and making code more modular and maintainable. Here's an overview of AngularJS directives:

Built-in Directives: AngularJS comes with a set of built-in directives that you can use to enhance your application's functionality and create dynamic UI elements. Some common built-in directives include:

- `ng-repeat` : Iterates over a collection and generates HTML elements for each item.
- `ng-if` and `ng-show` / `ng-hide` : Conditionally show or hide elements based on expressions.
- `ng-model` : Binds an input element's value to a property in the model, enabling two-way data binding.
- `ng-click` : Attaches a click event to an element and executes an expression when clicked.
- `ng-class` and `ng-style` : Dynamically add classes or apply inline styles based on conditions.

Let Us Explore Some of Them in Detail:

1. **`ng-app`** : The `ng-app` directive is used to define the root of the AngularJS application. It marks the HTML element where the AngularJS framework will be initialized. You typically place this directive on the `<html>` or `<body>` element of your HTML document.

   Example:

   ```
   <!DOCTYPE html>

   <html ng-app="myApp">
    ...
   </html>
   ```

2. **`ng-init`** : The `ng-init` directive allows you to initialize values in the AngularJS scope. While it's convenient for simple cases, it's generally recommended to use controllers for more complex initialization logic.

   Example:

   ```
   <div ng-init="name = 'John'">
        <p>Hello, {{ name }}</p>
   </div>
   ```

3. **`ng-controller`** : The `ng-controller` directive associates a controller with a specific section of the HTML. It defines the scope for that section of the HTML where the controller's properties and functions can be accessed.

   Example:

   ```
   <div ng-controller="myController">
       <p>{{ greeting }}</p>
   </div>
   ```

4. `ng-model` : The `ng-model` directive binds an input, textarea, or select element to a property in the AngularJS scope. It enables two-way data binding, meaning changes to the input value update the scope, and changes to the scope value update the input.

   Example:

   ```
   <input type="text" ng-model="username">
   <p>Hello, {{ username }}</p>
   ```

5. `ng-repeat` : The `ng-repeat` directive is used to iterate over a collection (like an array or an object) and generate HTML elements for each item. It's particularly useful for rendering lists of data.

   Example:

   ```
   <ul>
       <li ng-repeat="item in items">{{ item }}</li>
   </ul>
   ```

- These directives are fundamental to building dynamic and interactive applications using AngularJS.
- They allow you to create a seamless connection between the UI and your application's logic, making it easier to manage and maintain your codebase.

Some Other Directives

1. `ng-class` : The `ng-class` directive allows you to conditionally apply CSS classes to an HTML element based on expressions in your AngularJS application. It's useful for dynamically changing the styling of elements.

   Example:

   ```
   <div ng-class="{ 'highlight': isHighlighted, 'error': isError }">
       Content
   </div>
   ```

   In this example, the class `highlight` will be applied if `isHighlighted` is truthy, and the class `error` will be applied if `isError` is truthy.

2. **`ng-animate`** (deprecated): The `ng-animate` directive was used in earlier versions of AngularJS to animate elements when they are added, removed, or updated in the DOM. However, this directive has been removed in favor of using the `ngAnimate` module.

3. **`ng-show` and `ng-hide`**: The `ng-show` and `ng-hide` directives are used to conditionally show or hide elements based on expressions. When the expression evaluates to `true`, the element is shown; when it evaluates to `false`, the element is hidden.

   Example:

   ```
   <div ng-show="showElement">
       This element is shown.
   </div>
   <div ng-hide="hideElement">
       This element is hidden.
   </div>
   ```

   In this example, the first `<div>` will be visible if `showElement` is truthy, and the second `<div>` will be hidden if `hideElement` is truthy.

Expressions and Controllers

1. **Expressions**:

   Expressions in AngularJS are used to bind dynamic values to the HTML templates. They are typically enclosed within double curly braces `{{ }}`. Expressions can contain variables, literals, operators, and function calls, allowing you to perform simple calculations and display data from the model.

   Example:

   ```
   <p> Hello, {{ name }} </p>
   <p> Age: {{ age }} </p>
   <p> Total: {{ price * quantity }} </p>
   ```

   In this example, `name`, `age`, `price`, and `quantity` are variables that are part of the model, and their values will be interpolated into the HTML.

2. **Controllers**:

   Controllers in AngularJS are JavaScript functions that act as the bridge between the model and the view. They contain the application's business logic and interact with the model to update data that's presented in the view. Controllers manage the scope of the application.

Example:

```
angular.module('myApp', [])
       .controller('myController', function($scope) {
             $scope.name = 'John';
             $scope.age = 30;
             $scope.price = 10;
             $scope.quantity = 5;
});
```

In this example, a controller named `myController` is defined. It attaches properties like `name`, `age`, `price`, and `quantity` to the `$scope` object, making them accessible to the view. The controller initializes the initial values of these properties.

You can then use this controller in your HTML:

```
<div ng-app="myApp" ng-controller="myController">
     <p>Hello, {{ name }}</p>
     <p>Age: {{ age }}</p>
     <p>Total: {{ price * quantity }}</p>
</div>
```

The `ng-controller` directive associates the controller with a specific section of the HTML.

- Controllers are important for keeping your application organized and maintainable.
- They allow you to encapsulate logic and data manipulation, ensuring that the view remains clean and focused on presentation.

Filters In AngularJS

In AngularJS, filters are used to format and transform data displayed in the UI. They allow you to modify the presentation of data before it's rendered to the user. Filters are applied to expressions within double curly braces `{{ }}` in templates. Here are some commonly used AngularJS filters:

1. **`currency`**: Formats a number as a currency string using the specified currency code.

```
<p>{{ price | currency:'USD' }}</p>
```

2. **`date`**: Formats a date object or string into a human-readable date string.

```
<p>{{ myDate | date:'yyyy-MM-dd HH:mm:ss' }}</p>
```

3. `uppercase` and `lowercase` : Converts a string to uppercase or lowercase.

   ```
   <p>{{ 'Hello World' | uppercase }}</p>
   <p>{{ 'Hello World' | lowercase }}</p>
   ```

4. `number` : Formats a number to a specified number of decimal places.

   ```
   <p>{{ value | number:2 }}</p>
   ```

5. `orderBy` : Orders an array of objects based on a specified property.

   ```
   <ul>
       <li ng-repeat="item in items | orderBy:'name'">{{ item.name }}</li>
   </ul>
   ```

6. `filter` : Filters an array based on a search term or criteria.

   ```
   <ul>
        <li ng-repeat="item in items | filter:searchText">{{ item.name }}</li>
   </ul>
   ```

7. `limitTo` : Limits the number of items displayed in an array or string.

   ```
   <p>{{ text | limitTo:100 }}</p>
   <ul>
       <li ng-repeat="item in items | limitTo:5">{{ item.name }}</li>
   </ul>
   ```

8. `Custom Filters` : You can also create custom filters to perform more specific transformations.

   ```
   angular.module('myApp').filter('customFilter', function() {
       return function(input) {
           // Custom logic to transform input
           return transformedOutput;
       };
   });
   ```

   ```
   <p>{{ data | customFilter }}</p>
   ```

- These are just a few examples of AngularJS filters. Filters are a powerful way to manipulate data directly within templates, reducing the need for complex logic in controllers.
- However, be mindful not to overuse filters, especially when dealing with large datasets, as they can impact performance.