WT

Unit-3 AngularJS

Q:-1 What is AngularJS? Explain advantages/features of AngularJS.

AngularJS is a JavaScript-based open-source front-end web application framework. It was developed and is maintained by Google. AngularJS facilitates the development of dynamic, single-page web applications (SPAs) by providing a structured framework for building client-side applications.

Key features and advantages of AngularJS include:

1. **Two-way Data Binding:** AngularJS employs a two-way data binding system, which means that changes in the user interface (UI) are automatically reflected in the application's data model and vice versa. This simplifies the synchronization of the UI and the underlying data, reducing the need for manual DOM manipulation.
2. **MVC Architecture:** AngularJS follows the Model-View-Controller (MVC) architectural pattern. This helps in organizing and structuring the code in a way that enhances modularity, maintainability, and testability.
3. **Dependency Injection:** AngularJS has a built-in dependency injection system that makes it easy to manage and test components by injecting their dependencies rather than hard-coding them. This promotes modular and reusable code.
4. **Modularity:** AngularJS allows developers to build modular applications by breaking down the application into smaller, reusable components. These components can then be combined to create complex applications, making the codebase more manageable.
5. **Directives:** Directives in AngularJS are markers on a DOM element that tell AngularJS to attach a specified behavior to that DOM element or transform it in a certain way. They allow developers to create custom, reusable HTML elements and attributes, enhancing the extensibility of the framework.
6. **Templates:** AngularJS uses HTML as a declarative language for building UI. Templates in AngularJS are written in HTML and contain Angular-specific elements and attributes. This approach makes it easy for designers and developers to collaborate, as the separation of concerns is maintained.
7. **Testing Support:** AngularJS is designed with testability in mind. It includes features like dependency injection, which facilitates unit testing, and it provides tools like Jasmine and Karma for testing AngularJS applications.
8. **Single Page Application (SPA) Support:** AngularJS is well-suited for building SPAs, where a single HTML page is loaded initially, and subsequent interactions with the application are handled dynamically without the need for page reloads. This provides a smoother and more responsive user experience.
9. **Community and Ecosystem:** AngularJS has a large and active community, which means that developers can find a wealth of resources, tutorials, and third-party libraries. The ecosystem around AngularJS is robust, providing additional tools and extensions to enhance development.

10. **Declarative Code Style:** AngularJS promotes a declarative programming style, where developers describe what they want to achieve, and AngularJS takes care of the underlying implementation details. This can lead to more readable and maintainable code.

It's important to note that while AngularJS was widely used in the past, Angular (without the "JS") is the newer version (Angular 2 and above) that introduced significant changes and improvements. As of my last knowledge update in January 2022, developers are encouraged to use the latest version of Angular for new projects.

Q:-2 Enlist directives and explain with examples in AngularJS

In AngularJS, directives are a powerful feature that allows developers to extend the functionality of HTML by creating custom elements, attributes, classes, or comments. Directives can be used to create reusable components, manipulate the DOM, and add behavior to elements. Here are some commonly used directives in AngularJS along with examples:

1. **ng-app:**

- The `ng-app` directive is used to define the root element of the AngularJS application. It initializes the AngularJS application and specifies the top-level scope.

Example:

```html
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS App</title>
</head>
<body>
  <!-- The application content goes here -->
  <script src="angular.js"></script>
  <script src="app.js"></script>
</body>
</html>
```

2. **ng-model:**

- The `ng-model` directive binds an input, select, or textarea element to a property on the scope. It enables two-way data binding, automatically updating the model when the user interacts with the input.

Example:

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

3. **ng-repeat:**

- The `ng-repeat` directive is used for repeating elements (such as HTML elements or a set of data) in the DOM. It iterates over a collection and duplicates the HTML element for each item in the collection.

Example:

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

4. **ng-click:**

- The `ng-click` directive allows you to define behavior that should be executed when an element is clicked.

Example:

```html
<button ng-click="doSomething()">Click me</button>
```

5. **ng-if and ng-show/ng-hide:**

- `ng-if` removes or recreates an element based on a condition. `ng-show` and `ng-hide` show or hide an element based on a condition.

Example:

```html
<div ng-if="isUserLoggedIn">Welcome, User!</div>
<div ng-show="showDetails">Details are shown</div>
```

6. **ng-class and ng-style:**

- `ng-class` dynamically sets CSS classes based on conditions. `ng-style` dynamically sets inline styles based on conditions.

Example:

```html
<div ng-class="{ 'active': isActive, 'inactive': !isActive }">Status</div>
<div ng-style="{ 'color': textColor, 'font-size': fontSize }">Styled Text</div>
```

7. **ng-init:**

- The `ng-init` directive initializes variables in the AngularJS scope. While it's not recommended for complex logic, it can be useful for simple variable initialization.

Example:

```html
<div ng-init="count = 0">
  <p>Count: {{count}}</p>
  <button ng-click="count = count + 1">Increment</button>
</div>
```

8. **ng-submit:**

   - The `ng-submit` directive is used to define behavior that should be executed when a form is submitted.

   Example:

   ```html
   <form ng-submit="submitForm()">
     <!-- form fields go here -->
     <button type="submit">Submit</button>
   </form>
   ```

These are just a few examples of the many directives available in AngularJS. Directives play a crucial role in building dynamic and interactive applications using AngularJS.

Q:-3 Explain AngularJS filters with example.

Certainly! The `number` filter in AngularJS is used for formatting numbers. It allows you to control the number of decimal places and other aspects of number formatting. Here's an example:

```html
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>Number Filter Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>

<div ng-controller="NumberFilterController">
  <h2>Number Filter Example</h2>
```

```
  <p>Original Number: {{ originalNumber }}</p>
  <p>Formatted Number (default): {{ originalNumber | number }}</p>
  <p>Formatted Number (2 decimal places): {{ originalNumber | number:2 }}</p>
  <p>Formatted Number (currency): {{ originalNumber | currency }}</p>
</div>

<script>
  var app = angular.module('myApp', []);

  app.controller('NumberFilterController', function($scope) {
    $scope.originalNumber = 1234.5678;
  });
</script>

</body>
</html>
```

In this example:

- The `number` filter is used to format the `originalNumber`.
- The first usage (`{{ originalNumber | number }}`) applies the default formatting.
- The second usage (`{{ originalNumber | number:2 }}`) formats the number with 2 decimal places.
- The third usage (`{{ originalNumber | currency }}`) formats the number as currency.

When you run this example, you will see how the number is formatted differently based on the specified filter. The `number` filter is versatile and can be customized to suit various formatting requirements for numeric data in AngularJS applications.

Q:-4 Write AngularJS script for addition of two numbers or to find the sum of n numbers

```
<!DOCTYPE html>
<html ng-app="additionApp">
<head>
  <title>AngularJS Addition Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>

<div ng-controller="AdditionController">
  <h2>Addition of Two Numbers</h2>

  <form>
    <label>Enter First Number:</label>
    <input type="number" ng-model="num1" required>

    <label>Enter Second Number:</label>
    <input type="number" ng-model="num2" required>

    <button ng-click="addNumbers()">Add</button>
  </form>

  <p ng-if="result !== undefined">Result: {{ result }}</p>
</div>

<script>
  var app = angular.module('additionApp', []);

  app.controller('AdditionController', function($scope) {
```

```
  $scope.addNumbers = function() {
    // Convert input values to numbers and calculate the sum
    var num1 = parseFloat($scope.num1);
    var num2 = parseFloat($scope.num2);

    // Check if both numbers are valid
    if (!isNaN(num1) && !isNaN(num2)) {
      $scope.result = num1 + num2;
    } else {
      $scope.result = undefined;
      alert("Please enter valid numbers.");
    }
  };
 });
</script>

</body>
</html>
```

Q:-5 Discuss ng-repeat with example.

`ng-repeat` is an AngularJS directive used for iterating over a collection (such as an array or object) and generating HTML elements for each item in the collection. It allows you to dynamically create content in the view based on the data in the collection. Here's an example to illustrate how **`ng-repeat`** works:

```
<!DOCTYPE html>
<html ng-app="ngRepeatExample">
<head>
 <title>ng-repeat Example</title>
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>
```

```
<div ng-controller="RepeatController">
  <h2>ng-repeat Example</h2>

  <ul>
    <!-- Using ng-repeat to iterate over an array -->
    <li ng-repeat="fruit in fruits">{{ fruit }}</li>
  </ul>

  <table>
    <!-- Using ng-repeat to iterate over an array of objects -->
    <tr ng-repeat="person in people">
      <td>{{ person.name }}</td>
      <td>{{ person.age }}</td>
    </tr>
  </table>
</div>

<script>
  var app = angular.module('ngRepeatExample', []);

  app.controller('RepeatController', function($scope) {
    // Example with an array
    $scope.fruits = ['Apple', 'Banana', 'Orange', 'Grapes'];

    // Example with an array of objects
    $scope.people = [
      { name: 'John', age: 25 },
      { name: 'Jane', age: 30 },
      { name: 'Bob', age: 22 }
    ];
```

```
});
</script>

</body>
</html>
```

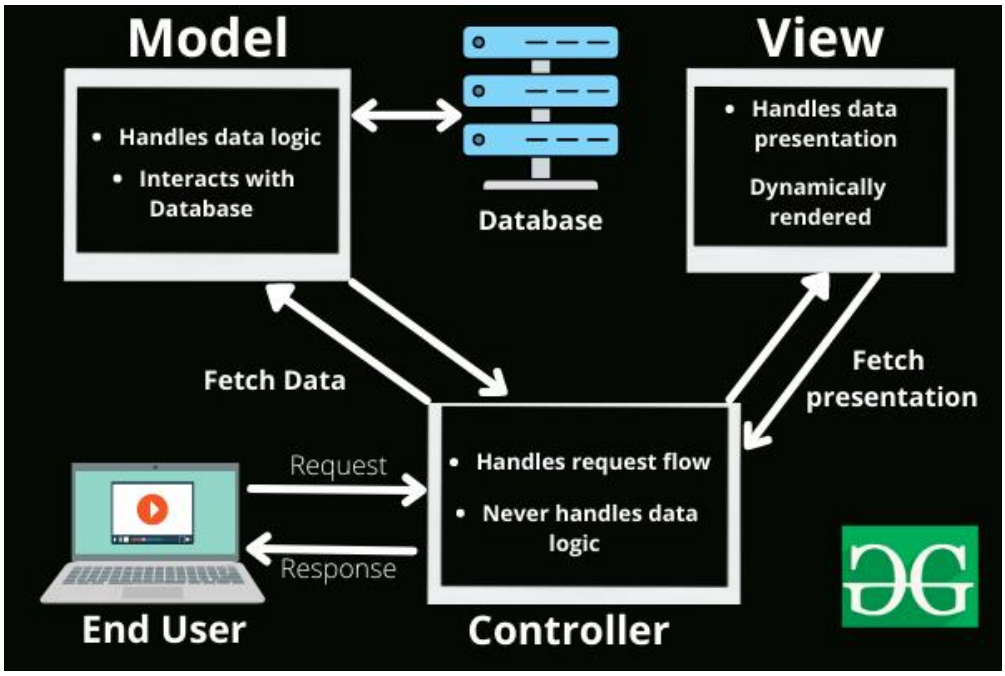
In this example:

1. The `ng-repeat` directive is used to iterate over an array of fruits and create an `<li>` element for each fruit in the list.
2. Another usage of `ng-repeat` is demonstrated inside a `<table>`, where it iterates over an array of objects (people) and creates a table row (`<tr>`) for each person with two columns.

When you run this example, you will see a list of fruits and a table with information about people. The content is dynamically generated based on the data provided in the `$scope.fruits` and `$scope.people` arrays.

This is a fundamental use case for `ng-repeat`, and it's widely used in AngularJS applications to display dynamic content based on data from the controller.

Q:-6 Discuss MVC architecture with example in detail.



The Model-View-Controller (MVC) framework is an architectural/design pattern that separates an application into three main logical components Model, View, and Controller. Each architectural component is built to handle specific development aspects of an application. It isolates the business logic and presentation layer from each other. It was traditionally used for desktop graphical user interfaces (GUIs). Nowadays, MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects. It is also used for designing mobile apps.
MVC was created by Trygve Reenskaug. The main goal of this design pattern was to solve the problem of users controlling a large and complex data set by splitting a large application into specific sections that all have their own purpose.

Features of MVC :

- It provides a clear separation of business logic, Ul logic, and input logic.
- It offers full control over your HTML and URLs which makes it easy to design web application architecture.
- It is a powerful URL-mapping component using which we can build applications that have comprehensible and searchable URLs.
- It supports Test Driven Development (TDD).

Components of MVC :
The MVC framework includes the following 3 components:

- Controller
- Model
- View

Controller:
The controller is the component that enables the interconnection between the views and the model so it acts as an intermediary. The controller doesn't have to worry about handling data logic, it just tells the model what to do. It processes all the business logic and incoming requests, manipulates data using the Model component, and interact with the View to render the final output.
View:
The View component is used for all the UI logic of the application. It generates a user interface for the user. Views are created by the data which is collected by the model component but these data aren't taken directly but through the controller. It only interacts with the controller.

Model:
The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. It can add or retrieve data from the database. It responds to the controller's request because the controller can't interact with the database by itself. The model interacts with the database and gives the required data back to the controller.

Working of the MVC framework with an example:

Let’s imagine an end-user sends a request to a server to get a list of students studying in a class. The server would then send that request to that particular controller that handles students. That controller would then request the model that handles students to return a list of all students studying in a class.

The model would query the database for the list of all students and then return that list back to the controller. If the response back from the model was successful, then the controller would ask the view associated with students to return a presentation of the list of students. This view would take the list of students from the controller and render the list into HTML that can be used by the browser.

The controller would then take that presentation and returns it back to the user. Thus ending the request. If earlier the model returned an error, the controller would handle that error by asking the view that handles errors to render a presentation for that particular error. That error presentation would then be returned to the user instead of the student list presentation.

Q:-7 Discuss scope and services in AngularJS.

In AngularJS, both scopes and services play crucial roles in organizing and managing the application's structure and behavior. Let's delve into each concept:

Scope in AngularJS:

A scope in AngularJS is an object that refers to the application model. It acts as a glue between the controller and the view, holding the data and functions that are accessible in both. Scopes are hierarchical and follow the DOM structure, allowing for the creation of a clear data-binding context.

Key aspects of scopes:

1. **Hierarchical Structure:** Scopes form a hierarchy based on the DOM structure. Each controller creates a new child scope, and these scopes are organized in a tree-like structure.
2. **Two-Way Data Binding:** Scopes facilitate two-way data binding, meaning that changes in the model (in the scope) automatically reflect in the view, and vice versa.
3. **Controller-View Interaction:** Controllers interact with the scope by attaching properties and functions to it. These properties and functions are then accessible in the associated view.

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>Scope Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>

<div ng-controller="MyController">
  <h2>{{ greeting }}</h2>
</div>

<script>
  var app = angular.module('myApp', []);

  app.controller('MyController', function($scope) {
    $scope.greeting = 'Hello, Angular!';
  });
</script>

</body>
</html>
```

In this example, the `MyController` controller attaches the `greeting` property to the scope, which is then displayed in the associated view.

Services in AngularJS:

Services are objects or functions in AngularJS that are responsible for carrying out specific tasks, such as data retrieval, business logic, or utility functions. They are designed to be reusable and provide a way to organize and share code across different components of an application.

Key aspects of services:

1. **Singleton Pattern:** Services in AngularJS are often implemented as singletons. There is only one instance of a service, and it is shared across different parts of the application.
2. **Dependency Injection:** AngularJS uses dependency injection to make services available to controllers, directives, and other services. This promotes modularity and code reuse.
3. **Built-In Services:** AngularJS comes with several built-in services like `$http` for making HTTP requests, `$timeout` for handling timeouts, and `$log` for logging.

Example:

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
<p>The url of this page is:</p>
<h3>{{myUrl}}</h3>
</div>

<p>This example uses the built-in $location service to get the absolute url of the page.</p>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $location) {
   $scope.myUrl = $location.absUrl();
});
</script>

</body>
</html>
```