AngularJS

---

1. Define AngularJS

- **AngularJS** is a popular, open-source JavaScript-based framework developed by Google.
- It is designed specifically for building dynamic, single-page web applications (SPAs) by extending HTML with additional attributes and capabilities.
- AngularJS simplifies development by enabling declarative programming and offering
    - two-way data binding
    - dependency injection
    - MVC (Model-View-Controller) architecture.

Key Features of AngularJS

2. What are the key features of AngularJS? Explain them

1. **Two-Way Data Binding**:
    - Keeps the view (HTML) and model (JavaScript objects) in sync automatically.
    - Reduces code and simplifies DOM manipulation.
2. **Dependency Injection**:
    - Manages dependencies between different components.
    - Increases modularity and testability of applications.
3. **Directives**:
    - Extend HTML with custom attributes and behaviors (ng-bind, ng-repeat, etc.).
    - Enable the creation of reusable components.
4. **MVC Architecture**:
    - Encourages separation of concerns:
        i. **Model**: Manages application data.

ii. **View**: Handles the user interface.
iii. **Controller**: Manages logic and data flow.

5. **Template Engine**:
   - AngularJS uses HTML as a declarative template language, making templates easy to write and understand.
6. **Routing**:
   - Built-in module (`ngRoute`) for creating SPAs.
   - Allows navigation without reloading the page.
7. **Filters**:
   - Format data for display (e.g., `currency`, `date`, `uppercase`).
8. **Testing**:
   - Integrated support for unit testing and end-to-end testing using tools like Jasmine and Protractor.

---

3. Explain Angular-JS and its significance in web application development.

- **Explanation of Angular-JS** **2 marks**
- **AngularJS** is an open-source JavaScript framework developed and maintained by Google. It is designed specifically for building dynamic, single-page web applications (SPAs) by extending HTML with additional attributes and capabilities.
- **The significance in web application development.** **5 marks**

1. **Simplifies Development:**
2. **Single-Page Applications (SPA):**
3. **Code Reusability and Modularity:**
4. **Two-Way Data Binding:**
5. **Declarative Programming:**
6. **Enhanced Testability:**
7. **Community and Ecosystem:**
8. **Cross-Platform Development:**

AngularJS expressions

4. What are AngularJS expressions?

- These are used to bind data between the application and the HTML view.
- They resemble JavaScript expressions but operate in the AngularJS context.
- AngularJS expressions can be written inside double curly braces `{{ }}` or within the `ng-bind` directive.

---

5. Explain the syntax of AngularJS Expressions with examples

- **Basic Syntax**: `{{ expression }}`
- **Attributes**: Can be used with directives like `ng-bind`.
- **Execution Context**: They are evaluated against the current scope, not the global window object.

- **Examples of AngularJS Expressions**

a. Arithmetic Operations

```
<div ng-app="" ng-init="x=10; y=5">
  Sum: {{ x + y }} <br>
  Product: {{ x * y }} <br>
  Division: {{ x / y }}
</div>
```

Output:

- Sum: 15
- Product: 50
- Division: 2

---

b. String Concatenation

```
<div ng-app="" ng-init="firstName='John'; lastName='Doe'">
```

```
  Full Name: {{ firstName + ' ' + lastName }}
</div>
```

Output: Full Name: John Doe

---

c. Conditionals

```
<div ng-app="" ng-init="isLoggedIn=true">
  Status: {{ isLoggedIn ? 'Welcome Back!' : 'Please Login' }}
</div>
```

Output: Status: Welcome Back!

---

d. Using AngularJS Objects

```
<div ng-app="" ng-init="person={name:'Alice', age:25}">
  Name: {{ person.name }} <br>
  Age: {{ person.age }}
</div>
```

Output:

- Name: Alice
- Age: 25

---

e. Array Access

```
<div ng-app="" ng-init="colors=['Red', 'Green', 'Blue']">
  Favorite Color: {{ colors[0] }}
</div>
```

Output: Favorite Color: Red

---

f. Expression in Directives

Using the `ng-bind` directive:

```
<div ng-app="" ng-init="greeting='Hello, World!'">
  <p ng-bind="greeting"></p>
</div>
```

Output: Hello, World!

AngularJS Modules

6. What is an AngularJS module? What are usage of An AngularJS module? How can you create a module explain with syntax?

is a container for different parts of an AngularJS application, such as controllers, directives, filters, services, and configurations.

Why Use Modules?

1. Organize your code into manageable parts.
2. Create reusable components.
3. Manage dependencies easily.
4. Avoid polluting the global namespace.

Creating a Module

```
var app = angular.module('myApp', []);
```

- **`myApp`**: The name of the module.
- **`[]`**: An array of dependencies (other modules).

Using the Module

After creating a module, you can attach components like controllers or directives to it.

7. Explain the concept of Data Binding in Angular-JS. What are the different types of data binding available in Angular-JS?

- **Explanation of Data Binding in Angular-JS.** **2 marks**
  Data Binding is a key feature of AngularJS that provides a way to synchronize the data between the model (JavaScript objects) and the view (HTML elements) in the user interface. It allows automatic synchronization of data between the controller and the view, making the development of dynamic web applications much easier and more efficient.
- **The different types of data binding available in Angular-JS?** **8 marks**
  1. **One-Way Data Binding**

In AngularJS, **one-way data binding** allows the view to be updated when the model changes but not vice versa. This means that data flows in a single direction—from the model (JavaScript variables) to the view (HTML).

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>One-Way Data Binding Example</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>
  <div ng-controller="MyController">
    <h1>One-Way Data Binding</h1>
    <!-- Displaying the model in the view -->
    <p>Message: {{ message }}</p>

    <!-- Modifying the input will not update the model -->
    <input type="text" value="{{ message }}" />
  </div>

  <script>
    // Define the AngularJS module and controller
    var app = angular.module('myApp', []);
    app.controller('MyController', function($scope) {
      // Define a model
      $scope.message = "Hello, AngularJS!";
    });
  </script>
</body>
</html>
```

2. **Two-Way Data Binding**

Two-way data binding in AngularJS allows synchronization between the model and the view. Any changes in the model are immediately reflected in the view, and vice versa. This is achieved using the `ng-model` directive.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Two-Way Data Binding Example</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-app="myApp" ng-controller="myController">

  <h1>Two-Way Data Binding Example</h1>
```

```
    <!-- Input field bound to the model -->
    <input type="text" ng-model="name">

    <!-- Output reflects changes in real-time -->
    <p>Hello, {{ name }}!</p>

</body>
<script>
    // Define the AngularJS application and controller
    var app = angular.module('myApp', []);
    app.controller('myController', function($scope) {
        $scope.name = "World"; // Initialize the model
    });
</script>
</html>
```

8. What is Event Binding? Explain with an example.

Event binding in AngularJS connects the user interaction (such as clicks, mouse events, keyboard input, etc.) with a function defined in the controller. It allows AngularJS to respond to user actions and trigger specific behavior.

The **`ng-click`** directive is one of the most commonly used directives for event binding. Other event directives include `ng-dblclick`, `ng-mouseover`, `ng-keyup`, etc.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Event Binding Example</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-app="eventApp" ng-controller="eventController">

    <h1>Event Binding Example</h1>

    <!-- Button with event binding -->
    <button ng-click="increment()">Click Me</button>

    <!-- Display the count -->
    <p>You clicked the button {{ count }} times!</p>

</body>
<script>
    // Define the AngularJS application and controller
    var app = angular.module('eventApp', []);
    app.controller('eventController', function($scope) {
        $scope.count = 0; // Initialize count
```

```
    // Function to increment the count
    $scope.increment = function() {
      $scope.count += 1;
    };
  });
</script>
</html>
```

9. List and explain key directives Used for Form Validation in Angular-JS.

- **`ng-required`**
  - Makes an input field required.
  - Example: `<input type="text" ng-model="username" ng-required="true">`
  - If this field is empty, the form will be marked as invalid.
- **`ng-minlength` and `ng-maxlength`**
  - Specifies the minimum and maximum number of characters allowed in an input.
  - Example: `<input type="text" ng-model="password" ng-minlength="6" ng-maxlength="12">`
  - If the input length is outside these bounds, AngularJS flags it as invalid.
- **`ng-pattern`**
  - Validates input using a regular expression pattern.
  - Example: `<input type="text" ng-model="phone" ng-pattern="/^\d{10}$/">`
  - Ensures that the input matches the specified pattern, such as a 10-digit phone number.
- **`ng-change`**
  - Executes a custom function when the value of the input changes.
  - Example: `<input type="text" ng`
  - `-model="email" ng-change="validateEmail()">`
  - Useful for custom validation logic or other dynamic updates.
- **`ng-model-options`**
  - Controls when AngularJS updates the model (e.g., after a delay or on blur).
  - Example: `<input type="text" ng-model="username" ng-model-options="{ updateOn: 'blur' }">`
  - Helps to control validation frequency, such as delaying validation until after the user stops typing.
- **`ng-trim`**
  - Trims whitespace from the beginning and end of the input by default, but can be disabled if needed.
  - Example: `<input type="text" ng-model="name" ng-trim="false">`
  - Ensures that unnecessary spaces do not affect validation.
- **`novalidate`**
  - Prevents the browser's default HTML5 validation, allowing AngularJS to handle validation.
  - Example: `<form name="myForm" novalidate> ... </form>`

    - This helps in managing validation purely within AngularJS.
- **`ng-submit`**
    - Handles form submission and can prevent submission if the form is invalid.
    - Example: `<form name="myForm" ng-submit="submitForm()">`
    - Allows you to specify actions when the form is submitted, based on validation.

10. What are the Controllers in XML? Explain their role in an XML-based system.

In the context of XML-based systems, **controllers** are not a direct feature of XML itself but are part of the architectural pattern used when integrating XML into systems, such as in **Model-View-Controller (MVC)** or similar paradigms. Here's an explanation of controllers in such systems:

What Are Controllers in XML-Based Systems?

Controllers act as intermediaries or coordinators in XML-based systems. They manage the flow of data and interactions between:

1. **Models (Data)**: The underlying data structures or sources, often represented in XML or another format.
2. **Views (Presentation)**: The user interface or visual representation of the data, which might be styled using technologies like XSLT (XML Stylesheet Language Transformations) or CSS.

Controllers handle:

- **Request Parsing**: They interpret requests (e.g., user inputs, API calls) and decide how to handle them.
- **Data Transformation**: They fetch, transform, or manipulate XML data to meet the application's needs.
- **Communication**: They manage the interaction between the XML data model and the view.

Role of Controllers in XML-Based Systems

1. **Facilitate Interaction**:
    - Controllers handle user interactions, API calls, or events that require data manipulation or view updates.

    - For example, a controller might parse a user's input form, retrieve corresponding data from an XML file, and prepare it for display.
2. **Data Transformation**:
    - Controllers often use XML parsers (e.g., SAX, DOM, StAX) to read or manipulate XML data.
    - They may also apply XSLT for transforming XML into HTML or other formats for presentation.
3. **Separation of Concerns**:
    - By decoupling the view and the model, controllers allow XML-based systems to be modular and more maintainable.
    - This makes it easier to update the view or modify the data model independently.
4. **Validation and Logic Execution**:
    - Controllers ensure the XML conforms to the required schema (e.g., XSD or DTD validation).
    - They implement business logic, ensuring that only valid data or requests proceed to further stages.

Example in Action

Imagine a web application that uses XML to store data and XSLT to present it:

1. A **user** submits a request to view a product catalog.
2. The **controller** receives this request and:
    - Retrieves the product data stored in an XML file.
    - Applies an XSLT stylesheet to transform the XML into an HTML representation.
    - Sends the transformed HTML to the user's browser.

In this setup, the controller ensures smooth coordination between the XML data (model) and its visual presentation (view).

Technologies Related to Controllers in XML Systems

- **Frameworks**: Many web frameworks incorporate controllers for XML, such as Spring MVC (Java) or ASP.NET MVC.
- **XML Processing Tools**: Tools like Apache Xerces, JAXB, or LINQ to XML often work alongside controllers.
- **Middleware**: Systems like RESTful APIs use controllers to interact with XML data.

AngularJS and ReactJS are popular JavaScript-based frameworks and libraries used for building web applications. While they serve similar purposes, they differ significantly in their approaches, features, and use cases. Here's a comparison:

1. Type

- **AngularJS**: A full-fledged **framework** for building single-page applications (SPAs). It provides a comprehensive solution for developing web apps, including routing, state management, and templating.
- **ReactJS**: A **library** focused on building user interfaces, particularly the view layer in MVC (Model-View-Controller). It is often combined with other libraries for a complete solution.

---

2. Creator and Ownership

- **AngularJS**: Developed and maintained by **Google**.
- **ReactJS**: Developed and maintained by **Facebook (now Meta)**.

---

3. Language and Syntax

- **AngularJS**: Uses **HTML** with **Directives** to extend its syntax and allows developers to bind data and add logic directly in HTML.
- **ReactJS**: Uses **JSX (JavaScript XML)**, which allows HTML-like syntax within JavaScript code, offering a more integrated experience for building components.

---

4. Architecture

- **AngularJS**: Follows an **MVC (Model-View-Controller)** or **MVVM (Model-View-ViewModel)** architecture.
- **ReactJS**: Follows a **component-based architecture** where the UI is divided into reusable components.

---

5. Data Binding

- **AngularJS**: Supports **two-way data binding**, meaning changes in the model update the view, and vice versa.
- **ReactJS**: Uses **one-way data binding**, where data flows from parent to child components, improving control and predictability.

---

6. Performance

- **AngularJS**: Performance can be slower in large applications due to its two-way data binding and digest cycle.
- **ReactJS**: Offers better performance because of the **Virtual DOM**, which efficiently updates only the parts of the real DOM that have changed.

---

7. Dependency Management

- **AngularJS**: Has a built-in **dependency injection system**, making it easier to manage dependencies.
- **ReactJS**: Does not have a built-in dependency injection system. Developers use external libraries like Redux or Context API for state and dependency management.

---

8. Learning Curve

- **AngularJS**: Steeper learning curve due to its comprehensive nature and reliance on concepts like directives, dependency injection, and scopes.
- **ReactJS**: Comparatively easier to learn, especially for developers familiar with JavaScript.

---

9. Flexibility

- **AngularJS**: Opinionated, offering a prescribed way of developing applications.
- **ReactJS**: Highly flexible and can be integrated with other libraries and tools to suit specific needs.

---

10. Community and Ecosystem

- **AngularJS**: Strong community with extensive documentation and resources but losing popularity to newer versions of Angular (Angular 2+).

- **ReactJS**: Larger community, more actively maintained, and widely adopted in modern web development.

---

11. Use Cases

- **AngularJS**: Ideal for building enterprise-level, large-scale applications where a full framework is beneficial.
- **ReactJS**: Preferred for dynamic, high-performance applications, especially when a lightweight, component-based approach is desired.

Summary Table:

Feature	AngularJS	ReactJS
Type	Framework	Library
Creator	Google	Facebook
Language/Syntax	HTML + Directives	JSX (JavaScript + HTML)
Architecture	MVC/MVVM	Component-Based
Data Binding	Two-Way	One-Way
Performance	Slower (Digest Cycle)	Faster (Virtual DOM)
Learning Curve	Steeper	Moderate
Flexibility	Opinionated	Flexible
Dependency Management	Built-In	External Libraries
Use Cases	Enterprise-Level SPAs	Dynamic, Interactive Apps

Each has its strengths and weaknesses, so the choice between AngularJS and ReactJS depends on the project's requirements and the team's familiarity with the tools.

Creating an external controller file to control object properties is common in frameworks like AngularJS or when using modular JavaScript. Below is an example snippet for AngularJS:

Example: External Controller File in AngularJS

Step 1: Define the Controller in an External File

Create a file named `myController.js`:

```
javascript
```

```
Copy code
// myController.js
angular.module('myApp', [])
    .controller('MyController', ['$scope', function($scope) {
        // Define object properties
        $scope.person = {
            name: "John Doe",
            age: 30,
            occupation: "Engineer"
        };

        // Define a method to update properties
        $scope.updateOccupation = function(newOccupation) {
            $scope.person.occupation = newOccupation;
        };
    }]);
```

Step 2: Link the External Controller File

Ensure the `myController.js` file is included in your HTML:

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

    <h1>Person Details</h1>
    <p>Name: {{ person.name }}</p>
    <p>Age: {{ person.age }}</p>
    <p>Occupation: {{ person.occupation }}</p>

    <input type="text" ng-model="newOccupation" placeholder="New Occupation">
    <button ng-click="updateOccupation(newOccupation)">Update
Occupation</button>

</body>
</html>
```

Explanation:

1. **Controller File (`myController.js`):**
    - Defines a module `myApp` and a controller `MyController`.

- Uses `$scope` to bind object properties (`person`) and methods (`updateOccupation`).

2. **HTML Integration:**
   - The `ng-app` directive initializes the AngularJS app.
   - The `ng-controller` directive binds the `MyController` to the view.
   - Data binding (`{{ ... }}`) dynamically displays and updates object properties.

This modular approach ensures separation of concerns, making the code more maintainable.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Mouse Move Example</title>

</head>

<body>

    <h1 id="header">Hover over me!</h1>

    <p>Mouse move count: <span id="count">0</span></p>


    <script>

        // Initialize the count variable

        let count = 0;


        // Get references to the H1 element and the count display

        const header = document.getElementById('header');
```

```
        const countDisplay = document.getElementById('count');


        // Add a mousemove event listener to the H1 element
        header.addEventListener('mousemove', () => {
            count++; // Increment the count
            countDisplay.textContent = count; // Update the display
        });
    </script>
</body>
</html>
```

Lab

In **AngularJS**,

👉 `ng-app` is a **directive** that is used to define the **root element** of an AngularJS application.

Key Points:

1. **Starting point**: It tells AngularJS where the application should start executing.
2. **Scope**: Everything inside the element with `ng-app` is controlled by AngularJS.
3. **Bootstrap**: AngularJS automatically bootstraps (initializes) the application when it sees `ng-app`.

`ng-app` initializes AngularJS application and tells Angular where to start working.