**Experiment 3**

**Shashwat Tripathi  
D15A Batch C  
RollNo: 64**

**AIM : To study AngularJS**

**THEORY:**

**1. What are directives? Name some of the most commonly used directives in AngularJS application.**

Directives in AngularJS are extended HTML attributes with the prefix ng-. They allow you to attach new behavior to HTML elements.

* ng-app: Initializes an AngularJS application. It identifies the root element of the application.
* ng-model: Binds the value of HTML controls (like input, select, textarea) to application data, providing features like type validation, status indicators, and form binding.
* ng-bind: Sets the inner text of an HTML element to the value of an AngularJS expression.
* ng-repeat: Repeats a series of HTML elements once per item from a collection, cloning the HTML elements once for each item.
* ng-if: Conditionally includes an HTML element in the document based on the value of an AngularJS expression.
* ng-show / ng-hide: Shows or hides the given HTML element based on the truthiness of the expression provided.
* ng-class: Dynamically sets CSS classes on an HTML element by binding an expression to one or more CSS classes.
* ng-style: Applies inline styles to an HTML element by binding an expression to CSS style names.
* ng-include: Fetches, compiles and includes an external HTML fragment into the current DOM.
* ng-view: Manages the transition between different parts of your application as users navigate through it.

**2. What is data binding in AngularJS?**

Data binding in AngularJS refers to the automatic synchronization between the model and the view. In AngularJS applications, the data model represents a collection of data available for the application, and the view is the HTML container where the application is displayed. The view has access to the model and can display the model data in various ways.

* Using ng-bind: You can use the ng-bind directive to bind the innerHTML of an element to a specified model property. For instance, <p ng-bind="firstname"></p> would bind the "firstname" property from the model to the paragraph element.
* Using double braces {{ }}: Double braces are used for string interpolation, allowing you to display content from the model directly in the view. For example, <p>First name: {{firstname}}</p> would display the value of "firstname" from the model.
* Using ng-model: The ng-model directive is used to bind data from the model to the view on HTML controls such as input, select, and textarea. This creates a two-way binding, meaning any changes in the model are reflected in the view, and any changes in the view update the model. For example, <input ng-model="firstname"> would bind the input field to the "firstname" property in the model.

**3. How is form validation done in angularJS**

* Built-in Directives: AngularJS offers directives like ng-required, ng-minlength, ng-maxlength, etc., that can be added to form inputs to apply validation rules. For example, using required attribute will ensure that the field must be filled out.
* Form State Tracking: AngularJS keeps track of the state of form inputs. It knows whether an input has been visited ($touched), modified ($dirty), or if it is valid ($valid) or invalid ($invalid). These properties can be used to display error messages conditionally.
* ng-show and ng-hide: AngularJS uses ng-show and ng-hide directives to conditionally display error messages based on the state of the form. For example, ng-show="form1.username.$pristine && form1.username.$invalid" will show an error message only if the username field has not been touched and its content is invalid.
* Custom Validation: Developers can create custom validation functions by adding a new directive to the application. This involves defining a function within the directive that performs the validation check and then sets the $valid or $invalid property of the form control accordingly.
* Preventing Form Submission: To prevent form submission when the form is invalid, you can use ng-submit directive with a condition that checks the form's validity before executing the submit function. For example, <form ng-submit="myForm.$valid && submitFunction()" name="myForm"> will only call submitFunction() if myForm is valid.
* Server-Side Validation: While client-side validation improves user experience by providing immediate feedback, it does not replace server-side validation. Client-side validation can be bypassed, so server-side validation is necessary to ensure data integrity and security.

**4. What is the use of AngularJS Controllers in the application?**

AngularJS Controllers are central to AngularJS applications. They are JavaScript functions that manage the data and behavior of a view. Controllers are where you define your application's behavior and manipulate the data received from the model.

* Managing Data: Controllers are responsible for initializing the data that the view will present. They can set default values for variables that are bound to the view.
* Handling User Interactions: Controllers respond to user actions by updating the model and changing the view accordingly. They contain functions that are called in response to events triggered by the view.
* Binding View to Model: Controllers act as a bridge between the model and the view. They expose the model's data and functions to the view, and they listen for view events that change the model.
* Using Dependency Injection: AngularJS Controllers often depend on services that perform tasks like fetching data from a server or performing calculations. AngularJS's dependency injection system allows these services to be easily passed into the controller.
* Creating Scopes: Every controller gets its own scope, which is an object that the controller and the view can use to communicate. The scope contains model data and functions that the view can access.
* Implementing Logic: Controllers contain the business logic of an application. They decide what happens when a user interacts with the view, such as clicking a button or submitting a form.

**5. What is the use of AngularJS Filters in the application?**

AngularJS Filters are used to format the data for display purposes without changing the actual data. They are added to expressions or directives using the pipe | symbol and can be used to transform data of various data types.

* currency: Converts a number into a currency format 3.
* date: Converts a date into a specified format 3.
* filter: Filters array and object elements and returns the filtered items 3.
* json: Converts a JavaScript object into JSON 3.
* limitTo: Returns an array or string containing a specified number of elements 3.
* lowercase: Converts a string into lowercase letters 3.
* uppercase: Converts a string into uppercase letters 3.
* number: Converts a number into a string or text 3.
* orderBy: Sorts an array based on specified predicate expressions 3.

**Problem Statement:**

**1. Demonstrate with an AngularJS code one way data binding and two way data binding in AngularJS**

**Code 1:**

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<meta charset="UTF-8" />

<title>AngularJS One-Way Data Binding</title>

<script src="./angular.min.js"></script>

</head>

<body>

<div ng-controller="OneWayController">

<h2>One-Way Data Binding</h2>

<p>{{message}}</p>

</div>

<script>

angular

.module("myApp", [])

.controller("OneWayController", function ($scope) {

$scope.message = "Hello, Shashwat here! I am learning AngularJS";

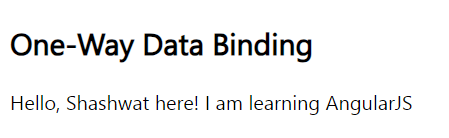
});

</script>

</body>

</html>

**Output:**



**Code 2:**

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<meta charset="UTF-8" />

<title>AngularJS Two-Way Data Binding</title>

<script src="./angular.min.js"></script>

</head>

<body>

<div ng-controller="TwoWayController">

<h2>Two-Way Data Binding</h2>

<p>Enter your name: <input type="text" ng-model="userName" /></p>

<p>Hello, {{userName}}!</p>

</div>

<script>

angular

.module("myApp", [])

.controller("TwoWayController", function ($scope) {

$scope.userName = "";

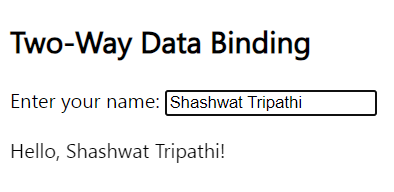
});

</script>

</body>

</html>

**Output:**

****

**2. Implement a basic authentication system for a web application using AngularJS. Create a simple login page that takes a username and password, and upon submission, checks for a hardcoded set of credentials. If the credentials are valid, display a success message; otherwise, show an error message.**

**Demonstrate AngularJS controller, module and form directives.**

**Code :**

<!DOCTYPE html>

<html lang="en" ng-app="authApp">

<head>

<meta charset="UTF-8" />

<title>AngularJS Authentication Example</title>

<script src="./angular.min.js"></script>

</head>

<body>

<div ng-controller="AuthController">

<h2 style="text-align: center">Login</h2>

<form ng-submit="login()">

<label for="username">Username:</label>

<input type="text" id="username" ng-model="user.username" required />

<label for="password">Password:</label>

<input

type="password"

id="password"

ng-model="user.password"

required

/>

<button type="submit">Login</button>

</form>

<div ng-show="showMessage" class="message">

<p ng-show="isSuccess" style="color: green">Login successful!</p>

<p ng-show="!isSuccess" style="color: red">

Invalid credentials. Please try again.

</p>

</div>

</div>

<script>

angular

.module("authApp", [])

.controller("AuthController", function ($scope) {

// Hardcoded set of credentials

var validCredentials = {

username: "demo",

password: "password123",

};

$scope.user = {

username: "",

password: "",

};

$scope.showMessage = false;

$scope.isSuccess = false;

$scope.login = function () {

// Check if entered credentials match the hardcoded values

if (

$scope.user.username === validCredentials.username &&

$scope.user.password === validCredentials.password

) {

$scope.isSuccess = true;

} else {

$scope.isSuccess = false;

}

// Display the message

$scope.showMessage = true;

// Optional: You can redirect to another page or perform additional actions upon successful login.

};

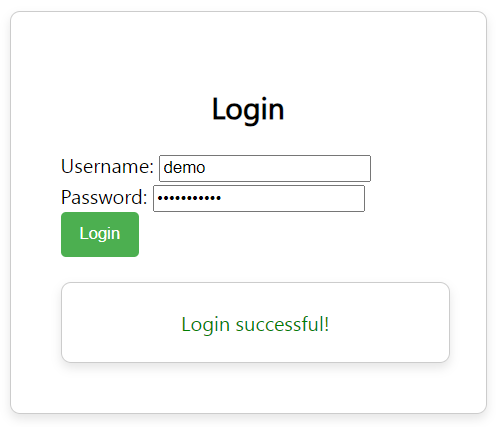
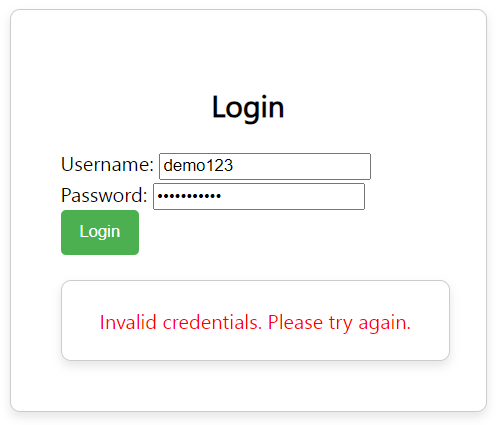
});

</script>

</body>

</html>

**Output :**

**3. Services**

In AngularJS, services are objects that are wired together using dependency injection (DI). They are primarily used for sharing data and functionality across controllers, directives, filters, and other services.

Singleton Pattern: Services in AngularJS follow the singleton pattern. This means that only one instance of a service is created and shared throughout the application. This is particularly useful for sharing configuration, caching, and other data that needs to persist across controllers.

Shared Functionality: Services are great for abstracting common functionality that doesn't belong to a single controller, directive, or module. They can be used to create utility functions or to manage the communication between controllers and backend services.

Encapsulation: Services encapsulate the logic and state that isn't associated with a particular view. This helps to keep controllers and directives lightweight and focused on their specific responsibilities.

Dependency Injection: Services are injected into other components, such as controllers, via the constructor function. This allows for easy testing and modularity as dependencies can be swapped out during testing.

Decoupling: Services help to decouple the codebase, making it easier to maintain and extend. They separate concerns and allow for better organization of code.

Statefulness: Since services are singletons, they can hold state over the lifetime of an application. This makes them suitable for storing user preferences, session data, and other information that needs to be persistent.

**4. Filters**

AngularJS Filters are used to format the data for display purposes without changing the actual data. They are added to expressions or directives using the pipe | symbol and can be used to transform data of various data types.

Filters can be applied to directives like ng-repeat to format the output dynamically. For example, you might use the currency filter to display prices in a list of products, or the date filter to format dates in a user profile.

Custom filters can also be created by registering a filter factory function in a module. A custom filter should be a pure function, producing the same result for a given input without affecting external states.

**CONCLUSION: Thus, we studied the angularjs.**