**Angular JS**

**Introducing Angular JS**

**Starting out with Angular JS**

**Basic AngularJS**

**Introduction to AngularJS**

AngularJS is a JavaScript framework developed by Google for building dynamic, single-page web applications (SPAs). It extends HTML by adding directives and allows for data binding, making the development process smoother and more efficient.

Let’s go through a simple example step by step to understand AngularJS.

**Step 1: Setup**

To get started with AngularJS, you'll need to include the AngularJS library in your HTML file. You can use a CDN link to include AngularJS.

Here’s how you can set up your HTML file:

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<h1>{{ greeting }}</h1>

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

<input type="text" ng-model="name" placeholder="Enter your name">

<button ng-click="changeMessage()">Click me!</button>

</div>

<script src="app.js"></script>

</body>

</html>

**Step 2: Create AngularJS Module and Controller**

Next, you need to define the AngularJS application. This is done by creating a module, and a controller to handle the logic. Create a file called app.js.

javascript

Copy

// Create an AngularJS module

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

// Create a controller for the module

app.controller("myCtrl", function($scope) {

// Define a simple greeting message

$scope.greeting = "Hello, AngularJS!";

// Define a dynamic message

$scope.message = "Welcome to the AngularJS tutorial.";

// Define a function that changes the message when the button is clicked

$scope.changeMessage = function() {

$scope.message = "Hello, " + $scope.name + "!";

};

});

**Step 3: Explanation of Code**

1. **HTML Structure**:
   * ng-app="myApp": This directive initializes the AngularJS application, linking it to the module myApp we defined in app.js.
   * ng-controller="myCtrl": This connects the HTML to the myCtrl controller from app.js.
   * {{ greeting }} and {{ message }}: These are **expressions** that AngularJS will automatically replace with values from the controller.
   * ng-model="name": This binds the input field to a variable name on the AngularJS scope, making it two-way bound. The value of name will automatically update as the user types in the input field.
   * ng-click="changeMessage()": This triggers the changeMessage() function in the controller when the button is clicked.
2. **JavaScript (app.js)**:
   * **Module**: angular.module("myApp", []) creates an AngularJS module. The empty array [] means there are no dependencies.
   * **Controller**: app.controller("myCtrl", function($scope) {...}) defines a controller named myCtrl. The controller contains data ($scope.greeting, $scope.message) and a method ($scope.changeMessage) that are bound to the view.
   * **$scope**: $scope is an object that acts as a bridge between the view (HTML) and the controller (JavaScript). Any data stored in $scope is accessible in the HTML using AngularJS expressions.

**Step 4: Run the Application**

To test the application:

1. Save the HTML file as index.html.
2. Save the JavaScript file as app.js.
3. Open the index.html file in a web browser.

You should see:

* A greeting message: "Hello, AngularJS!"
* A welcome message: "Welcome to the AngularJS tutorial."
* An input field where you can type your name.
* A button that, when clicked, changes the message to greet the user by name.

**2)Directives and Controllers**

### Directives and Controllers in AngularJS

AngularJS allows us to create dynamic and powerful web applications using **directives** and **controllers**. Let's go through both concepts step by step with examples.

### 1. ****Directives in AngularJS****

Directives are markers on DOM elements (like attributes, elements, classes, etc.) that tell AngularJS to attach specific behavior to those elements.

AngularJS provides many built-in directives, but you can also create your own custom directives.

#### Built-in Directives

Here are some of the most common built-in directives:

* ng-app: Initializes the AngularJS application.
* ng-model: Binds the value of an HTML element to the scope variable.
* ng-repeat: Repeats an element for each item in an array or object.
* ng-if: Conditionally includes or removes an element from the DOM.
* ng-click: Specifies an expression to evaluate when an element is clicked.

#### Example: Using ng-repeat and ng-click Directives

Let's create an example where we display a list of items and use ng-repeat to repeat the list items dynamically.

**HTML (index.html)**

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Directives Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<h1>My Favorite Fruits</h1>

<ul>

<!-- ng-repeat directive to loop through the fruits array -->

<li ng-repeat="fruit in fruits" ng-click="showFruit(fruit)">

{{ fruit }}

</li>

</ul>

<p ng-if="selectedFruit">You selected: {{ selectedFruit }}</p>

</div>

<script src="app.js"></script>

</body>

</html>

**JavaScript (app.js)**

javascript

Copy

// Create an AngularJS module

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

// Create a controller for the module

app.controller("myCtrl", function($scope) {

// Define an array of fruits

$scope.fruits = ["Apple", "Banana", "Cherry", "Date", "Elderberry"];

// Function to handle the selection of a fruit

$scope.showFruit = function(fruit) {

$scope.selectedFruit = fruit; // Update the selected fruit

};

});

### Explanation of Code:

* **ng-repeat**: Loops through the fruits array and creates a list item (<li>) for each fruit. It displays the name of each fruit and binds the click event to the showFruit function.
* **ng-click**: This directive triggers the showFruit(fruit) function when a fruit is clicked.
* **ng-if**: The ng-if directive displays the message "You selected: [fruit]" only when a fruit is selected.

When you click on any fruit name, it will display the selected fruit below the list using ng-if.

### 2. ****Controllers in AngularJS****

A **controller** in AngularJS is a JavaScript function that is responsible for handling the logic of the application. It manages the data ($scope) and behavior (functions) that the view will use.

Controllers are attached to the view via the ng-controller directive.

#### Example: Controller to Manage Input and Output

Let’s take a simple example where we manage a user's input and display it dynamically.

**HTML (index2.html)**

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Controller Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<h1>AngularJS Controllers</h1>

<p>Enter your name:</p>

<input type="text" ng-model="userName">

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

<button ng-click="reset()">Reset</button>

</div>

<script src="app2.js"></script>

</body>

</html>

**JavaScript (app2.js)**

javascript

Copy

// Create an AngularJS module

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

// Create a controller for the module

app.controller("myCtrl", function($scope) {

// Function to reset the user's name

$scope.reset = function() {

$scope.userName = ''; // Clears the input field

};

});

### Explanation of Code:

* **ng-model="userName"**: This directive binds the input field to the userName variable in the controller. Any changes in the input field will automatically update the userName in the scope.
* **{{ userName }}**: This expression displays the value of userName in real-time as the user types.
* **ng-click="reset()"**: This directive calls the reset() function when the button is clicked, which clears the userName field.

### 3. ****Custom Directive Example****

You can create your own custom directives in AngularJS to extend the functionality of HTML.

#### Example: Creating a Custom Directive

Let’s create a custom directive that changes the background color of a div when hovered.

**HTML (index3.html)**

<!DOCTYPE html>

<html lang="en" ng-app="myApp"> <!-- Moved ng-app here -->

<head>

<meta charset="UTF-8">

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

<title>AngularJS Custom Directive</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<style>

div {

width: 300px;

height: 200px;

border: 2px solid black;

display: flex;

justify-content: center;

align-items: center;

font-size: 18px;

text-align: center;

}

.hovered {

background-color: yellow !important;

}

</style>

</head>

<body>

<div ng-controller="myCtrl">

<h1>Custom Directive Example</h1>

<!-- Custom Directive in action -->

<div hover-color>

Hover over this box to change its background color!

</div>

</div>

<script src="app3.js"></script>

</body>

</html>

**JavaScript (app3.js)**

### // Create an AngularJS module

### var app = angular.module("myApp", []);

### // Create a controller to ensure AngularJS is working

### app.controller("myCtrl", function($scope) {

### console.log("Controller loaded!"); // Debugging log

### });

### // Create a custom directive called hoverColor

### app.directive("hoverColor", function() {

### return {

### restrict: 'A', // A for attribute directive

### link: function(scope, element, attrs) {

### console.log("Directive applied!"); // Debugging log

### element.on('mouseenter', function() {

### console.log("Mouse entered! Changing color."); // Debugging log

### element.addClass('hovered'); // Add class instead of setting CSS

### });

### element.on('mouseleave', function() {

### console.log("Mouse left! Resetting color."); // Debugging log

### element.removeClass('hovered');

### });

### }

### };

### });

### Explanation of Code:

* **hover-color**: This is the custom directive we created. We applied it as an attribute to the div.
* **Directive Definition**: In the directive, we defined a link function that adds event listeners to the div. When the mouse enters, the background color changes to yellow, and when the mouse leaves, the background color returns to transparent.

### Summary:

* **Directives**: AngularJS uses directives to add functionality to HTML elements. You can use built-in directives like ng-repeat, ng-click, and ng-if, or you can create custom directives.
* **Controllers**: Controllers in AngularJS manage the application's logic. They bind data and behavior to the view using $scope.
* **Custom Directives**: You can extend HTML by creating your own custom directives to add specific behavior to DOM elements.

These two concepts (directives and controllers) form the foundation of an AngularJS application.

**AngularJS Modules**

**Ans:**

### ****AngularJS Modules Example with Step-by-Step Process****

In **AngularJS**, a module is a container for different parts of an application such as controllers, directives, services, etc. It helps in maintaining modularity and code organization.

## ****Step-by-Step Process****

### ****Step 1: Create an HTML File****

Create an index.html file and include the AngularJS library.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Modules Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

<h1>{{ message }}</h1>

</div>

<script src="app.js"></script>

</body>

</html>

### ****Step 2: Create an AngularJS Module****

Create a JavaScript file app.js and define an AngularJS module.

// Define an AngularJS module

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

Here:

* "myApp" is the name of the module.
* [] is an array where dependencies can be added.

### ****Step 3: Create a Controller****

Now, add a controller to the module.

// Define the controller inside the module

app.controller("myController", function($scope) {

$scope.message = "Hello, AngularJS!";

});

Here:

* "myController" is the name of the controller.
* $scope is used to bind data between the controller and view.

### ****Step 4: Link Controller with HTML****

* The ng-app="myApp" directive in the <body> tag tells AngularJS to use this module.
* The ng-controller="myController" directive binds the controller to the <div>.
* {{ message }} displays the data stored in $scope.message.

### ****Step 5: Run the Application****

* Open index.html in a browser.
* You should see Hello, AngularJS! displayed.

## ****Enhancing the Example with a Service****

Let's add a service to fetch a message.

### ****Step 6: Create a Service****

Modify app.js to include a service:

js

CopyEdit

// Define the AngularJS module

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

// Create a service

app.service("messageService", function() {

this.getMessage = function() {

return "Hello from AngularJS Service!";

};

});

// Define the controller and inject the service

app.controller("myController", function($scope, messageService) {

$scope.message = messageService.getMessage();

});

Now, instead of directly defining the message in the controller, we fetch it from a service.

### ****Conclusion****

* We created an **AngularJS module**.
* Added a **controller** to handle data.
* Introduced a **service** to fetch data dynamically.

**Creating First Controller, working with and Displaying**

**Ans:**

A **controller** in AngularJS is a JavaScript function that is used to build the business logic of an application. It manages the data and interactions between the view (HTML) and the model (JavaScript objects).

## ****Step 1: Set Up HTML File****

Create a basic HTML file (index.html) and include the AngularJS library.

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS First Controller</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-app="myApp">

<!-- Attach the controller to this div -->

<div ng-controller="myController">

<h1>{{ message }}</h1>

<p>Name: {{ name }}</p>

</div>

<script src="app.js"></script>

</body>

</html>

### ****Explanation:****

* The ng-app="myApp" directive initializes the AngularJS application.
* The ng-controller="myController" directive links the myController controller to the <div>.
* {{ message }} and {{ name }} are AngularJS expressions that bind data to the view.

## ****Step 2: Define the Controller in JavaScript****

Create a file named app.js and define the AngularJS module and controller.

// Create the AngularJS module

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

// Define the controller

app.controller("myController", function($scope) {

// Define variables

$scope.message = "Welcome to AngularJS!";

$scope.name = "John Doe";

});

### ****Explanation:****

* We create a module named **myApp**.
* We define a controller **myController** inside the module.
* $scope is used to pass data between the controller and the view.
* The values **"Welcome to AngularJS!"** and **"John Doe"** are bound to the view using AngularJS expressions.

## ****Step 3: Run the Application****

1. Open index.html in a browser.
2. You should see the following output:

vbnet

CopyEdit

Welcome to AngularJS!

Name: John Doe

## ****Enhancing the Controller with User Input****

We can extend this example to allow users to update the name dynamically.

Modify index.html:

html

CopyEdit

<div ng-controller="myController">

<h1>{{ message }}</h1>

<p>Name: {{ name }}</p>

<!-- Input field to update the name -->

<input type="text" ng-model="name">

</div>

### ****Explanation:****

* The ng-model="name" directive binds the input field to the $scope.name variable.
* When the user types in the input field, the value updates in real time.

**Arrays in AnjularJS**

**Ans:**

In **AngularJS**, arrays can be used inside controllers to store multiple values and display them dynamically in the view using **ng-repeat** or other AngularJS directives.

## ****Step 1: Basic Array Example in AngularJS****

### ****1. Create an HTML File (****index.html****)****

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Arrays Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

<h2>List of Fruits</h2>

<ul>

<li ng-repeat="fruit in fruits">{{ fruit }}</li>

</ul>

</div>

<script src="app.js"></script>

</body>

</html>

### ****2. Define the AngularJS Module and Controller (****app.js****)****

js

CopyEdit

// Create an AngularJS module

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

// Define the controller

app.controller("myController", function($scope) {

// Define an array of fruits

$scope.fruits = ["Apple", "Banana", "Mango", "Orange", "Grapes"];

});

### ****Explanation:****

* We define a module called **myApp**.
* We create a controller **myController**, which contains an array **$scope.fruits**.
* We use **ng-repeat="fruit in fruits"** in the HTML to loop through the array and display each fruit.

## ****Step 2: Array of Objects in AngularJS****

Instead of a simple array, we can also store objects in an array.

### ****1. Modify**** index.html ****to Display an Array of Objects****

html

CopyEdit

<div ng-controller="myController">

<h2>List of Students</h2>

<table border="1">

<tr>

<th>Name</th>

<th>Age</th>

<th>Grade</th>

</tr>

<tr ng-repeat="student in students">

<td>{{ student.name }}</td>

<td>{{ student.age }}</td>

<td>{{ student.grade }}</td>

</tr>

</table>

</div>

### ****2. Modify**** app.js ****to Define an Array of Objects****

js

CopyEdit

app.controller("myController", function($scope) {

// Define an array of student objects

$scope.students = [

{ name: "Alice", age: 18, grade: "A" },

{ name: "Bob", age: 19, grade: "B" },

{ name: "Charlie", age: 17, grade: "A+" },

{ name: "David", age: 20, grade: "B+" }

];

});

### ****Explanation:****

* The **students** array contains objects with **name, age, and grade** properties.
* We use **ng-repeat** to display each student's details in a table.

## ****Step 3: Adding Data to an Array Dynamically****

Let's allow users to add new students dynamically.

### ****1. Modify**** index.html

html

CopyEdit

<div ng-controller="myController">

<h2>Student List</h2>

<input type="text" ng-model="newName" placeholder="Enter Name">

<input type="number" ng-model="newAge" placeholder="Enter Age">

<input type="text" ng-model="newGrade" placeholder="Enter Grade">

<button ng-click="addStudent()">Add Student</button>

<table border="1">

<tr>

<th>Name</th>

<th>Age</th>

<th>Grade</th>

</tr>

<tr ng-repeat="student in students">

<td>{{ student.name }}</td>

<td>{{ student.age }}</td>

<td>{{ student.grade }}</td>

</tr>

</table>

</div>

### ****2. Modify**** app.js ****to Add a Function****

js

CopyEdit

app.controller("myController", function($scope) {

// Define an array of student objects

$scope.students = [

{ name: "Alice", age: 18, grade: "A" },

{ name: "Bob", age: 19, grade: "B" }

];

// Function to add a new student

$scope.addStudent = function() {

if ($scope.newName && $scope.newAge && $scope.newGrade) {

$scope.students.push({

name: $scope.newName,

age: $scope.newAge,

grade: $scope.newGrade

});

// Clear input fields after adding a student

$scope.newName = "";

$scope.newAge = "";

$scope.newGrade = "";

}

};

});

## ****Conclusion****

* We learned how to use **arrays** in AngularJS.
* Displayed data using **ng-repeat**.
* Used **arrays of objects** to store structured data.
* Allowed users to **add new data dynamically**.

**More Directives**

**Ans:**

### ****AngularJS Directives with Examples****

**Directives** in AngularJS are special attributes that extend HTML functionality. They are used to bind data, manipulate DOM, and create reusable components.

## ****1. Commonly Used AngularJS Directives****

| **Directive** | **Description** |
| --- | --- |
| **ng-app** | Declares an AngularJS application. |
| **ng-controller** | Attaches a controller to an element. |
| **ng-model** | Binds input fields to scope variables. |
| **ng-repeat** | Loops through arrays and objects. |
| **ng-if** | Conditionally displays elements. |
| **ng-show/ng-hide** | Show or hide elements dynamically. |
| **ng-click** | Adds click event handlers. |
| **ng-class** | Dynamically applies CSS classes. |

## ****2. Example of AngularJS Directives****

Let's see these directives in action.

### ****Step 1: Create an HTML File (****index.html****)****

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Directives Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<style>

.highlight { color: red; font-weight: bold; }

</style>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

<!-- ng-model: Two-way data binding -->

<h2>ng-model Example</h2>

<input type="text" ng-model="name">

<p>Your name is: {{ name }}</p>

<!-- ng-repeat: Loop through array -->

<h2>ng-repeat Example</h2>

<ul>

<li ng-repeat="fruit in fruits">{{ fruit }}</li>

</ul>

<!-- ng-if: Conditional display -->

<h2>ng-if Example</h2>

<p ng-if="isVisible">This text is visible!</p>

<!-- ng-show / ng-hide -->

<h2>ng-show / ng-hide Example</h2>

<button ng-click="toggleVisibility()">Toggle</button>

<p ng-show="isVisible">This is shown when isVisible is true.</p>

<!-- ng-click: Handling button clicks -->

<h2>ng-click Example</h2>

<button ng-click="incrementCounter()">Click Me</button>

<p>Button clicked: {{ counter }} times</p>

<!-- ng-class: Dynamically apply CSS class -->

<h2>ng-class Example</h2>

<p ng-class="{'highlight': isHighlighted}">This text changes color when highlighted.</p>

<button ng-click="toggleHighlight()">Toggle Highlight</button>

</div>

<script src="app.js"></script>

</body>

</html>

### ****Step 2: Create AngularJS Controller (****app.js****)****

// Create an AngularJS module

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

// Define the controller

app.controller("myController", function($scope) {

// ng-model example

$scope.name = "John Doe";

// ng-repeat example

$scope.fruits = ["Apple", "Banana", "Mango", "Orange"];

// ng-if example

$scope.isVisible = true;

// ng-show / ng-hide example

$scope.toggleVisibility = function() {

$scope.isVisible = !$scope.isVisible;

};

// ng-click example

$scope.counter = 0;

$scope.incrementCounter = function() {

$scope.counter++;

};

// ng-class example

$scope.isHighlighted = false;

$scope.toggleHighlight = function() {

$scope.isHighlighted = !$scope.isHighlighted;

};

});

## ****3. Explanation of Directives Used****

### ✅ ****ng-model****

* Binds input field data to $scope.name.
* Any changes in the input field automatically update the displayed text.

### ✅ ****ng-repeat****

* Loops through $scope.fruits and displays each fruit in a <li> element.

### ✅ ****ng-if****

* Displays a paragraph only when $scope.isVisible is true.

### ✅ ****ng-show / ng-hide****

* Toggles visibility of an element based on $scope.isVisible.

### ✅ ****ng-click****

* Increments $scope.counter when the button is clicked.

### ✅ ****ng-class****

* Applies a CSS class (highlight) dynamically based on $scope.isHighlighted.

## ****4. Run the Application****

1. Open index.html in a browser.
2. Try interacting with input fields, buttons, and toggles.
3. Observe how AngularJS directives control behavior dynamically.

**Unit Testing in AngularJS**

**Ans:**

Unit testing in AngularJS is typically done using **Jasmine** as the testing framework and **Karma** as the test runner. Below is a step-by-step guide to writing unit tests in AngularJS.

## ****Step 1: Install Required Dependencies****

If you haven't already, install Karma, Jasmine, and AngularJS mocks:

npm install -g karma-cli

npm install --save-dev karma karma-jasmine jasmine-core karma-chrome-launcher karma-angular-filesort karma-coverage karma-jasmine-html-reporter angular-mocks

npm install angular angular-mocks --save-dev

## ****Step 2: Configure Karma****

Create a karma.conf.js file in your project root with the following configuration:

module.exports = function (config) {

config.set({

basePath: '',

frameworks: ['jasmine'],

files: [

'node\_modules/angular/angular.js',

'node\_modules/angular-mocks/angular-mocks.js',

'app/\*\*/\*.js',

'test/\*\*/\*.spec.js'

],

preprocessors: {

'app/\*\*/\*.js': ['coverage']

},

reporters: ['progress', 'coverage'],

browsers: ['Chrome'],

singleRun: false

});

};

## ****Step 3: Create an AngularJS Service****

Let's create a simple AngularJS service in app/services/myService.js:

angular.module('myApp', [])

.service('myService', function () {

this.add = function (a, b) {

return a + b;

};

this.multiply = function (a, b) {

return a \* b;

};

});

## ****Step 4: Write Unit Tests****

Create a test file test/services/myService.spec.js and write the unit tests for myService.

describe('myService', function () {

var myService;

// Load the module before each test

beforeEach(module('myApp'));

// Inject the service before each test

beforeEach(inject(function (\_myService\_) {

myService = \_myService\_;

}));

// Test add function

it('should add two numbers correctly', function () {

expect(myService.add(2, 3)).toBe(5);

});

// Test multiply function

it('should multiply two numbers correctly', function () {

expect(myService.multiply(2, 3)).toBe(6);

});

});

## ****Step 5: Run the Tests****

Run the following command to execute your tests:

karma start

**Forms, inputs and Services**

**Ans:**

### ****Forms, Inputs, and Services in AngularJS**** (Step-by-Step Guide)

In this guide, we'll cover:  
✅ Creating a **form with inputs** in AngularJS  
✅ Using **ng-model** for two-way data binding  
✅ **Validating** form inputs  
✅ Submitting form data to a **service**

## ****Step 1: Set Up AngularJS****

First, include AngularJS in your project. If you're not using a package manager, you can add it via a CDN in index.html:

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Form Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="FormController">

<form name="userForm" ng-submit="submitForm()" novalidate>

<label>Name:</label>

<input type="text" name="name" ng-model="user.name" required>

<span ng-show="userForm.name.$touched && userForm.name.$invalid">Required</span>

<br>

<label>Email:</label>

<input type="email" name="email" ng-model="user.email" required>

<span ng-show="userForm.email.$touched && userForm.email.$invalid">Invalid Email</span>

<br>

<label>Age:</label>

<input type="number" name="age" ng-model="user.age" min="18" required>

<span ng-show="userForm.age.$touched && userForm.age.$invalid">Age must be 18+</span>

<br>

<button type="submit" ng-disabled="userForm.$invalid">Submit</button>

</form>

<p><strong>Form Data:</strong> {{ user }}</p>

</div>

<script src="app.js"></script>

<script src="userService.js"></script>

</body>

</html>

## ****Step 2: Create an AngularJS Module and Controller (****app.js****)****

javascript

CopyEdit

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

app.controller('FormController', function ($scope, UserService) {

$scope.user = {

name: '',

email: '',

age: ''

};

$scope.submitForm = function () {

if ($scope.userForm.$valid) {

UserService.saveUser($scope.user);

}

};

});

## ****Step 3: Create an AngularJS Service (****userService.js****)****

javascript

CopyEdit

app.service('UserService', function () {

var savedUser = {};

this.saveUser = function (user) {

savedUser = angular.copy(user);

console.log('User Saved:', savedUser);

};

this.getUser = function () {

return savedUser;

};

});

## ****Step 4: Run and Test the Form****

✅ When you **type** into inputs, data is bound using ng-model.  
✅ When you **submit** the form, it **validates** inputs and calls submitForm().  
✅ The service UserService **saves the user data**.

**Leverage Data-Binding and Models**

**Ans:**

AngularJS (version 1.x) is a popular JavaScript framework used for building dynamic single-page applications. One of its core features is **data binding**, which allows you to automatically synchronize data between the model (the JavaScript objects) and the view (the user interface).

Let's go through a simple example step by step that demonstrates how **data-binding** and **models** work in AngularJS.

### Step 1: Set Up Your AngularJS Project

To get started with AngularJS, you need to include the AngularJS library in your HTML file. You can use a CDN link to quickly get started.

#### Create an index.html file:

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS 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">

<div ng-controller="myController">

<h1>{{ message }}</h1>

<input type="text" ng-model="message">

<p>You typed: {{ message }}</p>

</div>

<script src="app.js"></script>

</body>

</html>

In this HTML, we have:

* A div element with the ng-controller directive. This binds the view to a specific controller, which will be defined in JavaScript.
* A model message is displayed in the <h1> and bound to the <input> using the ng-model directive.
* The {{ message }} syntax is used for **two-way data binding**. When the user types in the input field, it automatically updates the value displayed in the <h1> and <p>.

### Step 2: Define AngularJS Module and Controller

Now, you need to write the AngularJS JavaScript code that defines the module and controller for your application. This JavaScript file (app.js) should be included at the bottom of the index.html file.

#### Create an app.js file:

javascript

CopyEdit

// Define the AngularJS module

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

// Define the controller

app.controller('myController', function($scope) {

// Model data (binding it to the view)

$scope.message = 'Hello, AngularJS!';

});

### Explanation:

1. **Module (myApp)**: This creates a new AngularJS module called myApp. In AngularJS, a module is a container for the application. It's where you define controllers, services, and other components.
2. **Controller (myController)**: The controller is used to control the data (model) and the behavior of the view. We inject the $scope object into the controller, which is a special object that allows you to create properties and methods that are accessible in the HTML view.
3. **Model (message)**: Inside the controller, we define a property message on $scope. This is our model, which will be bound to the view. Initially, message is set to 'Hello, AngularJS!'.

### Step 3: Test the Application

1. Open the index.html file in a browser.
2. You should see:
   * A heading: Hello, AngularJS!
   * An input box with the default value Hello, AngularJS!
   * A text output below the input box displaying You typed: Hello, AngularJS!
3. Now, when you start typing something in the input box, you'll notice:
   * The content of the <h1> tag changes in real time to match what you're typing.
   * The <p> tag updates with the new value of message.

### How Data Binding Works in This Example

* **One-way Data Binding**: Initially, the message from the controller is displayed in the <h1> using the {{ message }} syntax. This is an example of **one-way data binding** because data flows from the model (controller) to the view (HTML).
* **Two-way Data Binding**: When you use ng-model="message" on the input field, AngularJS automatically binds the value of the message model to the input field. Any change in the input box updates the model, and the view is automatically updated as well. This is an example of **two-way data binding**.

### Step 4: Modify and Experiment

You can modify the model data in the controller and see how the view updates in real time.

For example, add a button to change the message when clicked:

<button ng-click="changeMessage()">Change Message</button>

Then, in the app.js, define the changeMessage function:

app.controller('myController', function($scope) {

$scope.message = 'Hello, AngularJS!';

// Function to change message

$scope.changeMessage = function() {

$scope.message = 'You clicked the button!';

};

});

### Step 5: Wrap Up

With AngularJS, data-binding allows you to keep your view and model in sync automatically. When you change the data in the model, the view updates without any extra code needed, and vice versa. This makes building dynamic and interactive applications much easier.

#### Summary:

* **Two-way data binding** allows changes in the view (input field) to automatically update the model ($scope.message) and vice versa.
* The ng-model directive is used for two-way data binding.
* The {{ message }} syntax is used for one-way data binding to display the model data in the view.

**Form Validation and States**

**&**

**Error Handling with Forms**

**Ans:**

Form validation in AngularJS allows you to check whether a user’s input is valid before submitting the form. AngularJS provides built-in directives to help with form validation, such as ng-model, ng-required, ng-minlength, ng-pattern, and others. It also automatically tracks the form’s state, like whether the fields are dirty, touched, or valid.

Let's go step by step to create a simple form with validation and see how form states work in AngularJS.

### Step 1: Set Up Your Project

First, let's set up a basic AngularJS project similar to the previous example.

#### Create index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Form Validation Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

<h2>AngularJS Form Validation Example</h2>

<form name="myForm" ng-submit="submitForm()" novalidate>

<!-- Username Field -->

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

<input type="text" id="username" name="username" ng-model="user.username" ng-required="true" ng-minlength="5" />

<span ng-show="myForm.username.$touched && myForm.username.$invalid">

<span ng-show="myForm.username.$error.required">Username is required.</span>

<span ng-show="myForm.username.$error.minlength">Username must be at least 5 characters long.</span>

</span>

<br><br>

<!-- Email Field -->

<label for="email">Email:</label>

<input type="email" id="email" name="email" ng-model="user.email" ng-required="true" />

<span ng-show="myForm.email.$touched && myForm.email.$invalid">

<span ng-show="myForm.email.$error.required">Email is required.</span>

<span ng-show="myForm.email.$error.email">Please enter a valid email address.</span>

</span>

<br><br>

<!-- Submit Button -->

<button type="submit" ng-disabled="myForm.$invalid">Submit</button>

</form>

<div ng-show="formSubmitted">

<h3>Form Data Submitted:</h3>

<p>Username: {{ user.username }}</p>

<p>Email: {{ user.email }}</p>

</div>

</div>

<script src="app.js"></script>

</body>

</html>

### Step 2: Create the AngularJS Application

Now, let's create the AngularJS module and controller to handle the form validation and submission.

#### Create app.js:

// Define the AngularJS module

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

// Define the controller for handling form submission and validation

app.controller('myController', function($scope) {

// Initial data for the user

$scope.user = {

username: '',

email: ''

};

// To track if the form has been submitted

$scope.formSubmitted = false;

// Function to handle form submission

$scope.submitForm = function() {

if ($scope.myForm.$valid) {

// Form is valid, set submitted flag to true and log the user data

$scope.formSubmitted = true;

} else {

// If form is invalid, keep the user on the form

$scope.formSubmitted = false;

}

};

});

### Step 3: Explanation of the Code

#### HTML Form:

* **ng-submit**: This directive triggers the submitForm() function when the user submits the form.
* **ng-required**: This validates whether the field is required. If the user tries to submit the form without filling out the required field, it will show an error.
* **ng-minlength**: This directive enforces a minimum length requirement for the field. For example, the username should be at least 5 characters long.
* **ng-model**: This binds the input fields (username, email) to the AngularJS model (user), which makes the form data available in the controller.

#### Validation Feedback:

* **ng-show**: This directive is used to display validation error messages when the user interacts with the input fields.
* **$touched**: This flag indicates whether the user has clicked on (or interacted with) the input field. The error messages will only show once the user has "touched" the field.
* **$invalid**: This flag indicates whether the form field is invalid based on the validation rules you defined.

#### Submit Button:

* **ng-disabled**: The submit button is disabled if the form is invalid. This ensures the user cannot submit the form unless all required fields are valid.

#### Display Submitted Data:

* After the form is successfully submitted, the data entered by the user (i.e., the username and email) is displayed below the form.

### Step 4: Form States in AngularJS

AngularJS automatically keeps track of the form state, which is helpful for showing error messages and knowing if a field has been interacted with. Below are the key form state properties:

* **$valid**: Returns true if the form is valid (i.e., all fields pass validation).
* **$invalid**: Returns true if the form is invalid (i.e., any field fails validation).
* **$dirty**: Returns true if the form or field has been modified (the user has changed its value).
* **$pristine**: Returns true if the form or field has not been modified (the user has not changed its value).
* **$touched**: Returns true if the form or field has been clicked or interacted with.
* **$untouched**: Returns true if the form or field has not been interacted with.

In this example:

* The error messages are displayed based on the $touched and $invalid states.
* The submit button is disabled using the $invalid state, ensuring that users can't submit the form if it's invalid.

### Step 5: Testing the Form Validation

1. **Open the index.html file in your browser.**
2. You should see a form with "Username" and "Email" input fields and a "Submit" button.
3. Try submitting the form with invalid data:
   * If the username is less than 5 characters, it will show an error: Username must be at least 5 characters long.
   * If the email is empty or invalid, it will show an error: Email is required. or Please enter a valid email address.
4. When the form is filled out correctly, the "Submit" button will become active. Clicking it will display the submitted data below the form.

### Step 6: Customizing the Validation

You can add more validation rules to the form:

1. **Custom validation (using ng-pattern)**: If you want to enforce a specific pattern (e.g., a phone number), you can use ng-pattern:

html

CopyEdit

<label for="phone">Phone:</label>

<input type="text" id="phone" name="phone" ng-model="user.phone" ng-pattern="/^\d{3}-\d{3}-\d{4}$" />

<span ng-show="myForm.phone.$touched && myForm.phone.$invalid">

<span ng-show="myForm.phone.$error.pattern">Phone number must be in the format: 123-456-7890.</span>

</span>

1. **Custom error message**: You can also create custom error messages using the $error object to target specific validation errors, such as required, minlength, or pattern.

**ngModelOptions**

**Ans:**

In **AngularJS**, ngModelOptions is used to customize the behavior of ngModel in forms and input elements. It allows developers to control settings such as debounce time, update events, getter/setter functions, and more.

## ****1. Basic Usage of**** ngModelOptions

The ngModelOptions directive is used along with ngModel to define options like updateOn, debounce, allowInvalid, etc.

### ****Example 1: Using**** debounce ****for Delayed Input Update****

The debounce option is used to delay the model update until a specified time has passed.

#### ****Step-by-Step****

1. Create an input field with ngModelOptions that specifies a debounce time.
2. Observe how the model updates after a delay.

#### ****Code****

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>ngModelOptions - Debounce 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="myCtrl">

<h3>Debounce Example</h3>

<p>Type something in the input field, and observe the delayed update:</p>

<input type="text" ng-model="name" ng-model-options="{ debounce: 1000 }">

<p><strong>Model Value:</strong> {{ name }}</p>

<script>

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

app.controller('myCtrl', function($scope) {

$scope.name = "";

});

</script>

</body>

</html>

### ****Explanation****

* debounce: 1000 means the model updates **1 second after** the user stops typing.
* Without debounce, the model updates instantly on every keystroke.

## ****2. Using**** updateOn ****to Control Update Timing****

By default, ngModel updates on the input event. You can change this using updateOn.

### ****Example 2: Update Only on**** blur ****Event****

#### ****Code****

<input type="text" ng-model="email" ng-model-options="{ updateOn: 'blur' }">

<p><strong>Email:</strong> {{ email }}</p>

### ****Explanation****

* The model (email) is updated **only when the input field loses focus (blur)**, instead of on every keystroke.

## ****3. Combining**** updateOn ****with**** debounce

You can combine multiple options.

### ****Example 3: Update on**** blur ****or with a 2-second delay****

#### ****Code****

html

CopyEdit

<input type="text" ng-model="username" ng-model-options="{ updateOn: 'blur', debounce: { 'blur': 2000 } }">

<p><strong>Username:</strong> {{ username }}</p>

### ****Explanation****

* If the user clicks away (blur), the model updates immediately.
* Otherwise, if they stop typing, the model updates after **2 seconds**.

## ****4. Allow Invalid Values (****allowInvalid****)****

By default, AngularJS prevents invalid values from being assigned to the model. The allowInvalid: true option allows invalid values to be stored.

### ****Example 4: Allow Invalid Input****

html

CopyEdit

<input type="number" ng-model="age" ng-model-options="{ allowInvalid: true }">

<p><strong>Age:</strong> {{ age }}</p>

### ****Explanation****

* If the user types an invalid number (e.g., "abc"), it will still be stored in the model.

## ****5. Using**** getterSetter ****for Custom Model Binding****

This option allows using a function for getting and setting values.

### ****Example 5: Custom Getter/Setter****

#### ****Code****

html

CopyEdit

<input type="text" ng-model="customModel" ng-model-options="{ getterSetter: true }">

<p><strong>Custom Model:</strong> {{ customModel() }}</p>

#### ****Controller:****

js

CopyEdit

app.controller('myCtrl', function($scope) {

var storedValue = "Default Value";

$scope.customModel = function(newValue) {

return arguments.length ? storedValue = newValue : storedValue;

};

});

### ****Explanation****

* customModel is now a **function** instead of a direct variable.
* Calling customModel() retrieves the stored value.
* Assigning a value updates storedValue.

## ****Conclusion****

The ngModelOptions directive gives fine-grained control over ngModel. You can: ✅ Delay updates using debounce  
✅ Change when updates happen with updateOn  
✅ Store invalid values with allowInvalid  
✅ Use custom getter/setter functions

**Nested Forms with ng-form**

**Ans:**

## ****Nested Forms with**** ng-form ****in AngularJS (Step-by-Step Guide)****

In **AngularJS**, ng-form allows us to create **nested forms** inside a parent form. This is useful when dealing with complex forms with grouped sections that need individual validation.

## ****1. Why Use**** ng-form****?****

🔹 Helps manage **nested validations** within sections.  
🔹 Useful for **multi-step forms** or **grouped inputs**.  
🔹 Allows each section to have its **own $valid state** while still being part of the parent form.

## ****2. Basic Example: Using**** ng-form ****for Nested Forms****

### ****Step 1: Create an AngularJS App****

First, set up your AngularJS application and controller.

#### ****Code****

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Nested Forms with ng-form</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-app="myApp" ng-controller="myCtrl">

<h2>Nested Forms Example</h2>

<!-- Parent Form -->

<form name="parentForm" novalidate>

<!-- Personal Details Section (Nested Form) -->

<fieldset>

<legend>Personal Details</legend>

<ng-form name="personalForm">

<label>Name: </label>

<input type="text" name="name" ng-model="user.name" required>

<span ng-show="personalForm.name.$dirty && personalForm.name.$invalid" style="color:red;">

Name is required!

</span>

<br>

<label>Email: </label>

<input type="email" name="email" ng-model="user.email" required>

<span ng-show="personalForm.email.$dirty && personalForm.email.$invalid" style="color:red;">

Valid email is required!

</span>

</ng-form>

</fieldset>

<!-- Address Details Section (Nested Form) -->

<fieldset>

<legend>Address Details</legend>

<ng-form name="addressForm">

<label>Street: </label>

<input type="text" name="street" ng-model="user.address.street" required>

<span ng-show="addressForm.street.$dirty && addressForm.street.$invalid" style="color:red;">

Street is required!

</span>

<br>

<label>City: </label>

<input type="text" name="city" ng-model="user.address.city" required>

<span ng-show="addressForm.city.$dirty && addressForm.city.$invalid" style="color:red;">

City is required!

</span>

</ng-form>

</fieldset>

<!-- Form Status -->

<p><strong>Personal Form Valid:</strong> {{ personalForm.$valid }}</p>

<p><strong>Address Form Valid:</strong> {{ addressForm.$valid }}</p>

<p><strong>Parent Form Valid:</strong> {{ parentForm.$valid }}</p>

<button type="submit" ng-disabled="!parentForm.$valid">Submit</button>

</form>

<script>

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

app.controller('myCtrl', function($scope) {

$scope.user = {

name: '',

email: '',

address: {

street: '',

city: ''

}

};

});

</script>

</body>

</html>

## ****3. Explanation****

🔹 **ng-form name="personalForm"**

* Creates a nested form inside the main form (parentForm).
* Has its own validation and $valid state.

🔹 **Validations**

* If the user types and deletes, the error message appears.
* Each section is validated separately.

🔹 **Form State Check ($valid)**

* personalForm.$valid → Checks validity of "Personal Details".
* addressForm.$valid → Checks validity of "Address Details".
* parentForm.$valid → Checks the entire form, including nested forms.

🔹 **Submit Button Disabled Until Valid**

* The submit button is disabled (ng-disabled="!parentForm.$valid") until all fields are valid.

## ****4. Advantages of**** ng-form

✅ Each section is independently validated.  
✅ Prevents unnecessary revalidation of the entire form.  
✅ Useful for large, complex forms with multiple sections.

**Other Form Controls**

**Ans:**

## ****Form Controls in AngularJS (Step-by-Step Examples)****

AngularJS provides various **form controls** to handle user input effectively. The most common form controls include:

* **Text Inputs (<input type="text">)**
* **Checkboxes (<input type="checkbox">)**
* **Radio Buttons (<input type="radio">)**
* **Select Dropdown (<select>)**
* **Text Area (<textarea>)**
* **Date Picker (<input type="date">)**
* **Number Input (<input type="number">)**

Let's go step by step through each one with examples. 🚀

## ****1. Basic Form Setup in AngularJS****

First, set up an **AngularJS application** and a controller to manage form data.

### ****Step 1: Create AngularJS App****

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Form Controls</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-app="myApp" ng-controller="myCtrl">

<h2>AngularJS Form Controls</h2>

<!-- Start Form -->

<form name="myForm" novalidate>

<!-- Text Input -->

<label>Name:</label>

<input type="text" name="name" ng-model="user.name" required>

<span ng-show="myForm.name.$dirty && myForm.name.$invalid" style="color:red;">

Name is required!

</span>

<br><br>

<!-- Email Input -->

<label>Email:</label>

<input type="email" name="email" ng-model="user.email" required>

<span ng-show="myForm.email.$dirty && myForm.email.$invalid" style="color:red;">

Enter a valid email!

</span>

<br><br>

<!-- Number Input -->

<label>Age:</label>

<input type="number" name="age" ng-model="user.age" min="18" max="60" required>

<span ng-show="myForm.age.$dirty && myForm.age.$invalid" style="color:red;">

Age must be between 18 and 60!

</span>

<br><br>

<!-- Checkbox -->

<label>

<input type="checkbox" ng-model="user.agree"> I agree to terms

</label>

<br><br>

<!-- Radio Buttons -->

<label>Gender:</label>

<input type="radio" ng-model="user.gender" value="Male"> Male

<input type="radio" ng-model="user.gender" value="Female"> Female

<input type="radio" ng-model="user.gender" value="Other"> Other

<br><br>

<!-- Select Dropdown -->

<label>Country:</label>

<select ng-model="user.country">

<option value="">Select</option>

<option value="USA">USA</option>

<option value="UK">UK</option>

<option value="India">India</option>

</select>

<br><br>

<!-- Textarea -->

<label>Comments:</label>

<textarea ng-model="user.comments"></textarea>

<br><br>

<!-- Date Picker -->

<label>Birthdate:</label>

<input type="date" ng-model="user.birthdate">

<br><br>

<!-- Form Validation Status -->

<p><strong>Form Valid:</strong> {{ myForm.$valid }}</p>

<!-- Submit Button (Disabled if Form is Invalid) -->

<button type="submit" ng-disabled="!myForm.$valid">Submit</button>

</form>

<h3>Form Data Preview</h3>

<pre>{{ user | json }}</pre>

<script>

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

app.controller('myCtrl', function($scope) {

$scope.user = {

name: '',

email: '',

age: null,

agree: false,

gender: '',

country: '',

comments: '',

birthdate: ''

};

});

</script>

</body>

</html>

## ****2. Explanation of Form Controls****

### ✅ ****Text Input (****<input type="text">****)****

html

CopyEdit

<input type="text" name="name" ng-model="user.name" required>

* Stores user input in user.name
* required makes it a **mandatory field**
* Displays an error message if empty

### ✅ ****Email Input (****<input type="email">****)****

html

CopyEdit

<input type="email" name="email" ng-model="user.email" required>

* Validates proper email format
* Shows an error if an invalid email is entered

### ✅ ****Number Input (****<input type="number">****)****

html

CopyEdit

<input type="number" name="age" ng-model="user.age" min="18" max="60" required>

* Restricts input to **numeric values**
* Enforces **min/max validation**

### ✅ ****Checkbox (****<input type="checkbox">****)****

html

CopyEdit

<input type="checkbox" ng-model="user.agree"> I agree to terms

* Stores **true/false** values
* Useful for **terms and conditions**

### ✅ ****Radio Buttons (****<input type="radio">****)****

html

CopyEdit

<input type="radio" ng-model="user.gender" value="Male"> Male

<input type="radio" ng-model="user.gender" value="Female"> Female

<input type="radio" ng-model="user.gender" value="Other"> Other

* Allows **only one option** to be selected at a time

### ✅ ****Dropdown (****<select> ****with**** <option>****)****

html

CopyEdit

<select ng-model="user.country">

<option value="">Select</option>

<option value="USA">USA</option>

<option value="UK">UK</option>

<option value="India">India</option>

</select>

* User selects from a **predefined list of options**
* Stores selected value in user.country

### ✅ ****Textarea (****<textarea>****)****

html

CopyEdit

<textarea ng-model="user.comments"></textarea>

* Used for **multi-line input**
* Stores text in user.comments

### ✅ ****Date Input (****<input type="date">****)****

html

CopyEdit

<input type="date" ng-model="user.birthdate">

* Opens a **date picker** for input selection
* Stores value in **ISO format** (e.g., "2025-02-10")

### ✅ ****Validation and Submit Button****

html

CopyEdit

<p><strong>Form Valid:</strong> {{ myForm.$valid }}</p>

<button type="submit" ng-disabled="!myForm.$valid">Submit</button>

* **Displays real-time form validity**
* **Disables the submit button** if the form is invalid

## ****3. Live Data Preview (****<pre>{{ user | json }}</pre>****)****

html

CopyEdit

<h3>Form Data Preview</h3>

<pre>{{ user | json }}</pre>

* Shows live form data updates in **JSON format**
* Useful for debugging **real-time input changes**

## ****4. Conclusion****

🎯 **AngularJS provides powerful form controls** with built-in validation.  
🎯 You can **bind models (ng-model)** to get real-time updates.  
🎯 **Use myForm.$valid** to check overall form validity.  
🎯 **Disable submission (ng-disabled)** until all fields are valid.