Modules in AngularJS

Introduction to AngularJS Modules

A **module** in AngularJS is a container that defines an application and its various components, such as controllers, directives, services, and filters. It provides a way to organize code into reusable and manageable units.

In AngularJS, a module is created using the angular.module function. It acts as the root container for the entire application.

Creating an AngularJS Module

To create a module, use the angular.module function with the following syntax:

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

Explanation:

- "myApp" is the name of the module.
- [] is an array where dependencies (other modules) can be listed. Since it is empty, no dependencies are included.

Using a Module in an HTML File

After defining a module, it must be linked to the HTML file using the ng-app directive.

Example:

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"
></script>
    <script>
        var app = angular.module("myApp", []);
    </script>
</head>
<body>
    <h2>AngularJS Module Example</h2>
</body>
</html>
```

Explanation:

- ng-app="myApp" binds the module to the HTML document.
- The script initializes the module named "myApp".

Adding Components to a Module

A module can include controllers, directives, and services.

Example with a Controller:

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"
></script>
    <script>
        var app = angular.module("myApp", []);

        app.controller("myController", function($scope) {
            $scope.message = "Welcome to AngularJS!";
        });
    </script>
</head>
<body>
    <div ng-controller="myController">
        <h2>{{ message }}</h2>
    </div>
</body>
</html>
```

Explanation:

- The module "myApp" is created.
- The myController controller is defined inside the module.
- The controller assigns a message to $scope.message, which is displayed using {{ message }}.

Advantages of Using Modules in AngularJS

1. **Code Organization** – Helps in structuring code into reusable parts.
2. **Reusability** – Components inside a module can be used across different parts of an application.
3. **Separation of Concerns** – Keeps different functionalities separate for better maintainability.
4. **Dependency Management** – Modules help manage dependencies effectively.

Directives in AngularJS

Introduction to Directives

Directives in AngularJS are special attributes that extend HTML functionality. They allow developers to create dynamic and reusable components by attaching behavior to HTML elements.

AngularJS provides built-in directives, and developers can also create custom directives to enhance their applications.

Types of Directives in AngularJS

1. **Built-in Directives** – Provided by AngularJS, such as ng-app, ng-model, ng-repeat, etc.
2. **Custom Directives** – User-defined directives that add custom behavior to HTML elements.

Commonly Used Built-in Directives

1. ng-app Directive

- Initializes an AngularJS application.
- Usually placed in the <html> or <body> tag.

Example:

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

---

2. ng-model Directive

- Binds form inputs to a variable in the controller’s scope.
- Enables **two-way data binding**.

Example:

```
<div ng-app="">
    <input type="text" ng-model="name">
    <p>Hello, {{ name }}!</p>
</div>
```

- As the user types, the text updates dynamically.

---

3. ng-bind Directive

- Binds data to an HTML element, similar to {{ }} interpolation.

Example:

```
<p ng-bind="message"></p>
```

---

4. ng-repeat Directive

- Loops through an array and displays the data dynamically.

Example:

```
<ul ng-app="" ng-init="students=['John', 'Mike', 'Emma']">
    <li ng-repeat="student in students">{{ student }}</li>
```

```
</ul>
```

---

5. ng-if, ng-show, and ng-hide Directives

- **ng-if**: Conditionally adds or removes an element from the DOM.
- **ng-show**: Shows an element if the condition is true.
- **ng-hide**: Hides an element if the condition is true.

Example:

```
<div ng-app="" ng-init="isVisible=true">
    <button ng-click="isVisible = !isVisible">Toggle</button>
    <p ng-if="isVisible">This text appears if isVisible is true.</p>
</div>
```

Creating a Custom Directive

Developers can create their own directives using the directive function.

Example:

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"
></script>
    <script>
        var app = angular.module("myApp", []);

        app.directive("customMessage", function() {
            return {
                template: "<h2>This is a custom directive!</h2>"
            };
        });
    </script>
</head>
<body>
    <div custom-message></div>
</body>
</html>
```

Explanation:

- customMessage is a custom directive that adds a template to the element where it is used.

Advantages of Directives

1. **Enhances HTML Functionality** – Adds dynamic behavior to static HTML.

2. **Code Reusability** – Reduces code duplication by creating reusable components.
3. **Better Code Organization** – Separates logic from presentation for maintainability.
4. **Improves Readability** – Makes the application more readable and structured.

Routes in AngularJS

Introduction to Routing in AngularJS

Routing in AngularJS enables **Single Page Applications (SPA)** by allowing navigation between different views without reloading the entire webpage. The ngRoute module is commonly used to manage routes, enabling dynamic content updates based on the URL.

Setting Up Routing in AngularJS

1. Include Required Libraries

To use routing, include both the AngularJS core library and the angular-route.js file in your HTML file.

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

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

---

2. Create an AngularJS Module and Configure Routes

Use the $routeProvider service to define different routes in your application.

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"
></script>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-
route.js"></script>
    <script>
        var app = angular.module("myApp", ["ngRoute"]);

        app.config(function($routeProvider) {
            $routeProvider
            .when("/", {
                templateUrl: "home.html"
            })
            .when("/about", {
                templateUrl: "about.html"
            })
            .when("/contact", {
                templateUrl: "contact.html"
            })
            .otherwise({
                template: "<h2>404 Page Not Found</h2>"
            });
        });
    </script>
</head>
<body>
    <h1>AngularJS Routing Example</h1>
    <a href="#!/">Home</a> |
    <a href="#!/about">About</a> |
    <a href="#!/contact">Contact</a>

    <div ng-view></div>
</body>
</html>
```

---

3. Create Separate HTML Files for Each View

home.html:

```
<h2>Welcome to the Home Page</h2>
<p>This is the main page content.</p>
```

about.html:

```
<h2>About Us</h2>
<p>Information about our company.</p>
```

contact.html:

```
<h2>Contact Us</h2>
<p>Reach out via email at contact@example.com.</p>
```

Explanation of the Code

1. **Module Creation:**
    - The "myApp" module is created, and the "ngRoute" module is included.
2. **Route Configuration using $routeProvider:**
    - .when("/", { templateUrl: "home.html" }) loads home.html when the URL is #/.

- .otherwise({ template: "404 Page Not Found" }) handles undefined routes.

3. **Navigation Using Hash URLs (#!)**
   - <a href="#!/about">About</a> changes the content dynamically.
4. **Displaying Content with ng-view:**
   - The <div ng-view></div> is the placeholder where different views load dynamically.

Advantages of Routing in AngularJS

1. **Enables Single Page Applications (SPA)** – No full-page reloads, making navigation faster.
2. **Efficient Content Loading** – Loads only required parts of the page dynamically.
3. **Improved Code Organization** – Separates views into different HTML files for maintainability.
4. **Enhanced User Experience** – Provides seamless transitions between sections.

AngularJS Forms and Validations

Introduction to Forms in AngularJS

Forms in AngularJS provide an easy way to collect user input and validate the data before processing. AngularJS extends HTML forms with built-in validation features and provides **two-way data binding** through the ng-model directive.

---

1. Creating a Basic AngularJS Form

A form in AngularJS is defined using the <form> element, and input fields are bound using ng-model.

Example:

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"
></script>
</head>
<body>
    <div ng-controller="FormController">
        <form name="userForm">
            <label>Name:</label>
            <input type="text" name="userName" ng-model="user.name"
required>
            <p ng-show="userForm.userName.$error.required">Name is
required.</p>
```

```
            <label>Email:</label>
            <input type="email" name="userEmail" ng-model="user.email"
required>
            <p ng-show="userForm.userEmail.$error.required">Email is
required.</p>
            <p ng-show="userForm.userEmail.$error.email">Invalid email
format.</p>

            <label>Age:</label>
            <input type="number" name="userAge" ng-model="user.age"
min="18" required>
            <p ng-show="userForm.userAge.$error.required">Age is
required.</p>
            <p ng-show="userForm.userAge.$error.min">Minimum age should be
18.</p>

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

    <script>
        var app = angular.module("myApp", []);
        app.controller("FormController", function($scope) {
            $scope.user = {};
        });
    </script>
</body>
</html>
```

---

2. Understanding Form Validation in AngularJS

AngularJS provides built-in validation properties that help track the state of form fields:

Property	**Description**
$valid	Returns true if all form fields are valid.
$invalid	Returns true if at least one field is invalid.
$dirty	Returns true if the field value has been modified.
$pristine	Returns true if the field has not been modified.
$touched	Returns true if the field has been focused and then left.

3. Commonly Used Form Validation Directives

Directive	**Description**
required	Ensures the field is not empty.
ng-minlength	Sets a minimum character length.
ng-maxlength	Sets a maximum character length.
ng-pattern	Validates input based on a specified pattern.
type="email"	Ensures input follows a valid email format.
type="number"	Restricts input to numbers only.
ng-disabled	Disables the button if the form is invalid.

---

4. Example of Custom Validation Using ng-pattern

You can use the ng-pattern directive to validate inputs with custom regular expressions.

```
<label>Phone Number:</label>
<input type="text" name="userPhone" ng-model="user.phone" ng-
pattern="/^[0-9]{10}$/" required>
<p ng-show="userForm.userPhone.$error.pattern">Phone number must be 10
digits.</p>
```

---

5. Displaying Error Messages Dynamically

To show error messages only after the user interacts with the field, use $touched.

```
<p ng-show="userForm.userEmail.$touched &&
userForm.userEmail.$error.email">
    Please enter a valid email address.
</p>
```

Advantages of AngularJS Form Validations

1. **Automatic Validation** – AngularJS automatically tracks field validity.
2. **Real-time Feedback** – Users get instant validation messages without submitting the form.
3. **Custom Validations** – Developers can define their own validation rules.
4. **Improves User Experience** – Forms are interactive and prevent incorrect submissions.

Data Binding in AngularJS

Introduction to Data Binding

Data binding in AngularJS is the process of synchronizing the data between the model (JavaScript variables) and the view (HTML). This eliminates the need for manually updating the DOM, making applications more interactive and efficient.

Types of Data Binding in AngularJS

1. **One-Way Data Binding**
2. **Two-Way Data Binding**

---

1. One-Way Data Binding

One-way data binding updates the view when the model changes, but changes in the view do not affect the model.

Example:

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"
></script>
</head>
<body>
    <div ng-controller="MyController">
        <p>Enter Name: <input type="text" ng-model="name"></p>
        <p>Hello, {{ name }}</p>
    </div>

    <script>
        var app = angular.module("myApp", []);
        app.controller("MyController", function($scope) {
            $scope.name = "John";
        });
    </script>
</body>
</html>
```

Data Binding in AngularJS

Introduction to Data Binding

Data binding in AngularJS is the process of synchronizing the data between the model (JavaScript variables) and the view (HTML). This eliminates the need for manually updating the DOM, making applications more interactive and efficient.

Types of Data Binding in AngularJS

1. One-Way Data Binding
2. Two-Way Data Binding

---

1. One-Way Data Binding

One-way data binding updates the view when the model changes, but changes in the view do not affect the model.

Example:

html

CopyEdit

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>
  <div ng-controller="MyController">
    <p>Enter Name: <input type="text" ng-model="name"></p>
    <p>Hello, {{ name }}</p>
  </div>

  <script>
    var app = angular.module("myApp", []);
    app.controller("MyController", function($scope) {
      $scope.name = "John";
    });
  </script>
</body>
</html>
```

Explanation:

- The input field is bound to the name variable using ng-model.

- **The {{ name }} expression displays the value dynamically.**
- **This updates the view whenever the model changes but does not update the model if the view changes outside of ng-model.**

---

2. Two-Way Data Binding

Two-way data binding allows real-time synchronization between the model and the view. Any change in the input field updates the model, and any change in the model updates the view automatically.

Example:

html

CopyEdit

```
<p>Enter Age: <input type="number" ng-model="age"></p>

<p>Your age is: {{ age }}</p>
```

Explanation:

- **The ng-model directive binds the input field with the age variable.**
- **When the user changes the value, both the model and view are updated simultaneously.**

Difference Between One-Way and Two-Way Data Binding

Feature	One-Way Data Binding	Two-Way Data Binding
Updates View	Yes	Yes
Updates Model	No	Yes
Example Syntax	{{ variable }}	ng-model="variable"
Performance	Faster	Slightly Slower

Advantages of Data Binding in AngularJS

1. **Reduces Manual DOM Manipulation – No need for document.getElementById() or innerHTML.**
2. **Simplifies Development – Less code to manage updates between the model and view.**
3. **Enhances User Experience – Real-time updates improve interactivity.**
4. **Automatic Synchronization – Keeps UI and data in sync automatically.**

Explanation:

- The input field is bound to the name variable using ng-model.

- The {{ name }} expression displays the value dynamically.
- This updates the view whenever the model changes but does not update the model if the view changes outside of ng-model.

---

2. Two-Way Data Binding

Two-way data binding allows real-time synchronization between the model and the view. Any change in the input field updates the model, and any change in the model updates the view automatically.

Example:

```
<p>Enter Age: <input type="number" ng-model="age"></p>
<p>Your age is: {{ age }}</p>
```

Explanation:

- The ng-model directive binds the input field with the age variable.
- When the user changes the value, both the model and view are updated simultaneously.

Difference Between One-Way and Two-Way Data Binding

Feature	**One-Way Data Binding**	**Two-Way Data Binding**
Updates View	Yes	Yes
Updates Model	No	Yes
Example Syntax	{{ variable }}	ng-model="variable"
Performance	Faster	Slightly Slower

Advantages of Data Binding in AngularJS

1. **Reduces Manual DOM Manipulation** – No need for document.getElementById() or innerHTML.
2. **Simplifies Development** – Less code to manage updates between the model and view.
3. **Enhances User Experience** – Real-time updates improve interactivity.
4. **Automatic Synchronization** – Keeps UI and data in sync automatically.