What is AngularJS?

- **AngularJS** is a JavaScript framework developed by Google.
- It is used to build **dynamic, single-page web applications (SPAs)**.
- It extends **HTML with additional features** and enables **two-way data binding**.

First Release: 2010
Latest Stable Version: AngularJS 1.x (Angular 2+ is a complete rewrite)
Used For: Web applications, dashboards, CRUD operations, and more.

Key Features of AngularJS

- **Two-Way Data Binding** – Automatically syncs data between model and view.
- **Directives** – Extends HTML with custom attributes (ng-model, ng-repeat, etc.).
- **MVC Architecture** – Follows the Model-View-Controller approach.
- **Dependency Injection (DI)** – Injects services where needed.
- **Single-Page Application (SPA) Support** – Loads content dynamically without page refresh.
- **Filters** – Format data for display (uppercase, currency, etc.).

How to Use AngularJS?

A. Adding AngularJS to Your Project

The easiest way to include AngularJS is by using the **CDN link**:

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

You can also **download and include** the AngularJS file manually.

Basic Syntax of AngularJS

A. AngularJS Application Setup

An AngularJS app starts with the ng-app directive in the <html> or <body> tag.

- **Example: Simple AngularJS App**

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

    <div ng-controller="myController">
```

```
        <h1>Hello, {{ name }}!</h1>
        <input type="text" ng-model="name" placeholder="Enter your name">
    </div>

    <script>
        var app = angular.module("myApp", []);
        app.controller("myController", function($scope) {
            $scope.name = "AngularJS";
        });
    </script>

</body>
</html>
```

How It Works?

✓ **ng-app="myApp"** – Defines the AngularJS application.

✓ **ng-controller="myController"** – Assigns a controller to this section.

✓ **{{ name }}** – Displays the variable value dynamically.

✓ **ng-model="name"** – Binds input value to the name variable.

B. Directives (Extend HTML Functionality)
AngularJS has many built-in **directives** that enhance HTML.

Directive	**Description**
ng-app	Declares the AngularJS application.
ng-model	Binds input fields to variables.
ng-bind	Displays dynamic values.
ng-repeat	Loops over an array.
ng-show/ng-hide	Show/hide elements based on conditions.
ng-click	Handles click events.

C. Two-Way Data Binding (Auto-Sync Between Model & View)

AngularJS automatically updates **HTML when JavaScript data changes**.

Example: Real-Time Data Binding

```
<div ng-controller="myController">
  <input type="text" ng-model="message">
  <p>You typed: {{ message }}</p>
</div>
```

As you type in the input box, the text updates instantly!

D. Controllers (Handle Application Logic)

Controllers **manage data and business logic** in AngularJS.

Example: Creating a Controller

```
<div ng-controller="mathController">
   <p>2 + 3 = {{ addNumbers(2, 3) }}</p>
</div>

<script>
   var app = angular.module("myApp", []);
   app.controller("mathController", function($scope) {
      $scope.addNumbers = function(a, b) {
         return a + b;
      };
   });
</script>
```

E. Filters (Modify Displayed Data)

Filters modify how **data appears in the view** (uppercase, currency, date formatting, etc.).

Example: Applying Filters

```
html
CopyEdit
<p>Original: {{ message }}</p>
<p>Uppercase: {{ message | uppercase }}</p>
<p>Lowercase: {{ message | lowercase }}</p>
<p>Currency: {{ price | currency }}</p>

<script>
   var app = angular.module("myApp", []);
   app.controller("myController", function($scope) {
      $scope.message = "Hello AngularJS";
      $scope.price = 199.99;
   });
</script>
```

Filters **format text and numbers dynamically**.

<u>Features of AngularJS</u>

1. **Two-Way Data Binding**
    - Automatically synchronizes data between the model and view, ensuring that any changes in the model are instantly reflected in the view and vice versa.

- **Example:** When a user types something into an input field, it automatically updates the associated data in the model without additional code.

2. **MVC Architecture (Model-View-Controller)**
   - AngularJS uses a simplified **MVC** pattern, which separates application logic (Model), user interface (View), and the controller that connects them.
   - **Example:** The ng-controller directive manages the data and logic for a section of your app.
3. **Templates**
   - AngularJS uses the **combination of HTML and Angular expressions** (like {{ expression }}) to render dynamic content. The browser parses the template into the DOM, and Angular binds it with the model.

4. **Modules**
   - AngularJS organizes code into **modules**. This enables developers to separate different parts of an application logically and load them when necessary.

5. **Dependency Injection (DI)**
   - AngularJS makes use of DI to **inject services** like $http, $scope, or custom services into controllers and components. This simplifies testing and code organization.

6. **Routing**
   - AngularJS enables the creation of **Single Page Applications (SPA)** by routing URLs to specific templates and controllers without page reloads. The ngRoute module is used for routing.

7. **Services**
   - Built-in and custom **services** (like $http for AJAX calls or $timeout for time-based events) offer reusable logic across your application.

8. **Filters**
   - Filters allow you to **format data before displaying it** in the view. Common filters include uppercase, currency, and date.

9**. Directives**
   - AngularJS extends HTML with **custom attributes and elements**. These directives add dynamic behavior to static HTML, such as ng-repeat, ng-model, and ng-click.
   - **Example:** Use ng-repeat to loop over an array and dynamically create HTML content for each item.

<u>Advantages of AngularJS</u>

1. **Two-Way Data Binding**
   - Reduces the amount of code needed to keep the UI in sync with the data model. You don't need to write complex DOM manipulation code for updating the view.

2. **Modular Development**
   - AngularJS encourages modular development using its built-in components like modules, controllers, and services. This **enhances code reusability** and allows for better maintainability.

3. **Simplifies Development**

- With built-in features like two-way data binding, dependency injection, and directives, AngularJS **reduces the need for boilerplate code**.

4. **Single-Page Application (SPA) Support**
   - AngularJS is ideal for creating SPAs where content dynamically updates without reloading the entire page. This enhances **user experience and performance**.

5. **Separation of Concerns (MVC)**
   - By separating logic (Model), UI (View), and control (Controller), AngularJS makes it easier to **maintain large applications**. Developers can work on different components without affecting the others.

6. **Unit Testing Ready**
   - AngularJS is designed with testing in mind, particularly with the dependency injection mechanism, which allows easy mocking of services and controllers during testing. It works well with frameworks like Jasmine and Karma.

7. **Extensibility**
   - AngularJS is highly **customizable and extensible**. You can build custom directives, services, filters, and more. It can also be integrated with other libraries and frameworks.

8. **Community Support and Documentation**
   - Backed by **Google**, AngularJS has a large and active community, offering plenty of resources, tutorials, and support. Its extensive documentation is well-maintained.

9. **Cross-Browser Compatibility**
   - AngularJS handles the discrepancies between different browsers, ensuring a **consistent user experience** across all modern browsers.

10. **Reduced Development Time**
    - Features like two-way data binding and built-in directives allow **faster development**, which is especially useful for building complex, dynamic web applications.

AngularJS Application Structure:

```
/my-angular-app
|— /app
|   |— /controllers
|   |   ├— mainController.js
|   |— /services
|   |   ├— dataService.js
|   |— /directives
|   |   ├— customDirective.js
|   |— /views
|   |   ├— home.html
|   |   ├— about.html
|   |— app.js
|— /assets
|   |— /css
|   |   ├— styles.css
|   |— /js
|   |   ├— script.js
|— index.html
```

Each part of the AngularJS application serves a specific purpose.

index.html (Main Entry Point)
The index.html file is the main page of the application. It includes AngularJS, external scripts, and initializes the app.

Example:
html
CopyEdit

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>AngularJS App</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script src="app/app.js"></script>
  <script src="app/controllers/mainController.js"></script>
</head>
<body ng-app="myApp">

  <div ng-controller="mainController">
    <h1>{{ message }}</h1>
  </div>

</body>
```

```
</html>
```
✓ **Defines AngularJS app with ng-app="myApp"**

✓ **Includes necessary JavaScript files**

---

app.js (Main Module)
This file **initializes the AngularJS application** by defining a module.
Example:
```
js
CopyEdit
var app = angular.module("myApp", []);
```
✓ **Defines a module named myApp**
✓ **The empty array [] represents dependencies (other modules if needed)**

---

controllers/ (Manages Application Logic)
Controllers handle **data and business logic** in AngularJS.

Example: mainController.js
```
app.controller("mainController", function($scope) {
  $scope.message = "Welcome to AngularJS!";
});
```
✓ **Defines mainController**
✓ **Uses $scope to bind data to the view**

---

views/ (Manages HTML Templates)
Views contain **HTML templates** that are dynamically loaded.

Example: home.html
```
<div>
  <h2>Home Page</h2>
  <p>{{ message }}</p>
</div>
```
✓ **Displays dynamic content using {{ message }}**

---

services/ (Handles Data & API Calls)
Services are **reusable functions** that handle data manipulation and API requests.

Example: dataService.js
```
app.service("dataService", function() {
  this.getMessage = function() {
    return "Hello from Data Service!";
  };
});
```
✓ **Defines a reusable dataService**

---

directives/ (Creates Custom HTML Components)
Directives extend HTML by adding **custom behavior**.

Example: customDirective.js
```
app.directive("customMessage", function() {
  return {
```

```
    template: "<h3>This is a custom directive!</h3>"
  };
});
```

✓ Creates a directive <custom-message></custom-message>

---

assets/ (Stores Static Files)

- /css/ → Stylesheets
- /js/ → Additional JavaScript
- /images/ → Images

Complete Example

Below is a **simple AngularJS application** following the structure above.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>AngularJS Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script src="app/app.js"></script>
  <script src="app/controllers/mainController.js"></script>
  <script src="app/services/dataService.js"></script>
</head>
<body ng-app="myApp">

  <div ng-controller="mainController">
    <h1>{{ message }}</h1>
    <p>{{ dataMessage }}</p>
  </div>

</body>
</html>
```

app/app.js

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

app/controllers/mainController.js

```
app.controller("mainController", function($scope, dataService) {
  $scope.message = "Welcome to AngularJS!";
  $scope.dataMessage = dataService.getMessage();
});
```

app/services/dataService.js

```
app.service("dataService", function() {
  this.getMessage = function() {
    return "Hello from Data Service!";
  };
});
```

Component	Description
index.html	Main entry point with AngularJS integration.
app.js	Defines the AngularJS application module.
controllers/	Handles application logic (e.g., mainController.js).
views/	Contains HTML templates for different pages.
services/	Provides reusable functions for data handling (e.g., API calls).
directives/	Creates custom HTML elements.
assets/	Stores static files like CSS, images, and scripts.

AngularJS Routing & Navigation

Routing in **AngularJS** allows you to create **Single Page Applications (SPA)** where the content changes dynamically **without reloading the entire page**. This makes applications **faster, smoother, and more efficient**.

What is Routing in AngularJS?

- ✔ Routing helps in **navigating between different views (pages) dynamically**.
- ✔ It allows users to go to **different sections of a website without refreshing the page**.
- ✔ Uses the **ngRoute module** to handle routing.

Setting Up Routing in AngularJ

Step 1: Include angular-route.js
Before using routing, include the **AngularJS routing library**.

Add this to your index.html

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

Configuring Routes

Routing is defined inside the **AngularJS module** using the $routeProvider.

Example: Basic Routing Setup (app.js)

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

app.config(function($routeProvider) {
  $routeProvider
  .when("/", {
    templateUrl: "views/home.html"
  })
  .when("/about", {
```

```
    templateUrl: "views/about.html"
  })
  .otherwise({
    redirectTo: "/"
  });
});
```

- .when() → Defines different **routes (URLs)** and their corresponding **views (HTML pages)**.
- .otherwise() → Redirects to a default page if an **undefined route** is accessed.

Creating Views (HTML Pages)
Each route loads a different **view (HTML file)** inside the main page.

views/home.html

```
<h2>Welcome to Home Page</h2>
<p>This is the home section of our AngularJS app.</p>
```

views/about.html

```
<h2>About Us</h2>
<p>This page contains information about our application.</p>
```

Navigating Between Pages
To navigate between routes, use **<a href="#/route-name">**.

Example: Add Navigation Links to index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>AngularJS Routing</title>
  <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 src="app.js"></script>
</head>
<body ng-app="myApp">

  <nav>
    <a href="#/">Home</a> |
    <a href="#/about">About</a>
  </nav>

  <div ng-view></div> <!-- The content of the views will load here -->

</body>
</html>
```

- <a href="#/about"> → Loads the **About page** dynamically.
- <div ng-view> → Placeholder where the **page content will change dynamically**.

Adding a Controller to a Route
You can add a **controller** to handle logic inside each route.

Modify app.js to include controllers

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

app.config(function($routeProvider) {
  $routeProvider
  .when("/", {
    templateUrl: "views/home.html",
    controller: "homeController"
  })
  .when("/about", {
    templateUrl: "views/about.html",
    controller: "aboutController"
  })
  .otherwise({
    redirectTo: "/"
  });
});

app.controller("homeController", function($scope) {
  $scope.message = "Welcome to the Home Page!";
});

app.controller("aboutController", function($scope) {
  $scope.message = "This is the About Page.";
});
```

Update views/home.html

```
<h2>Home Page</h2>
<p>{{ message }}</p> <!-- Message from controller -->
```

Update views/about.html

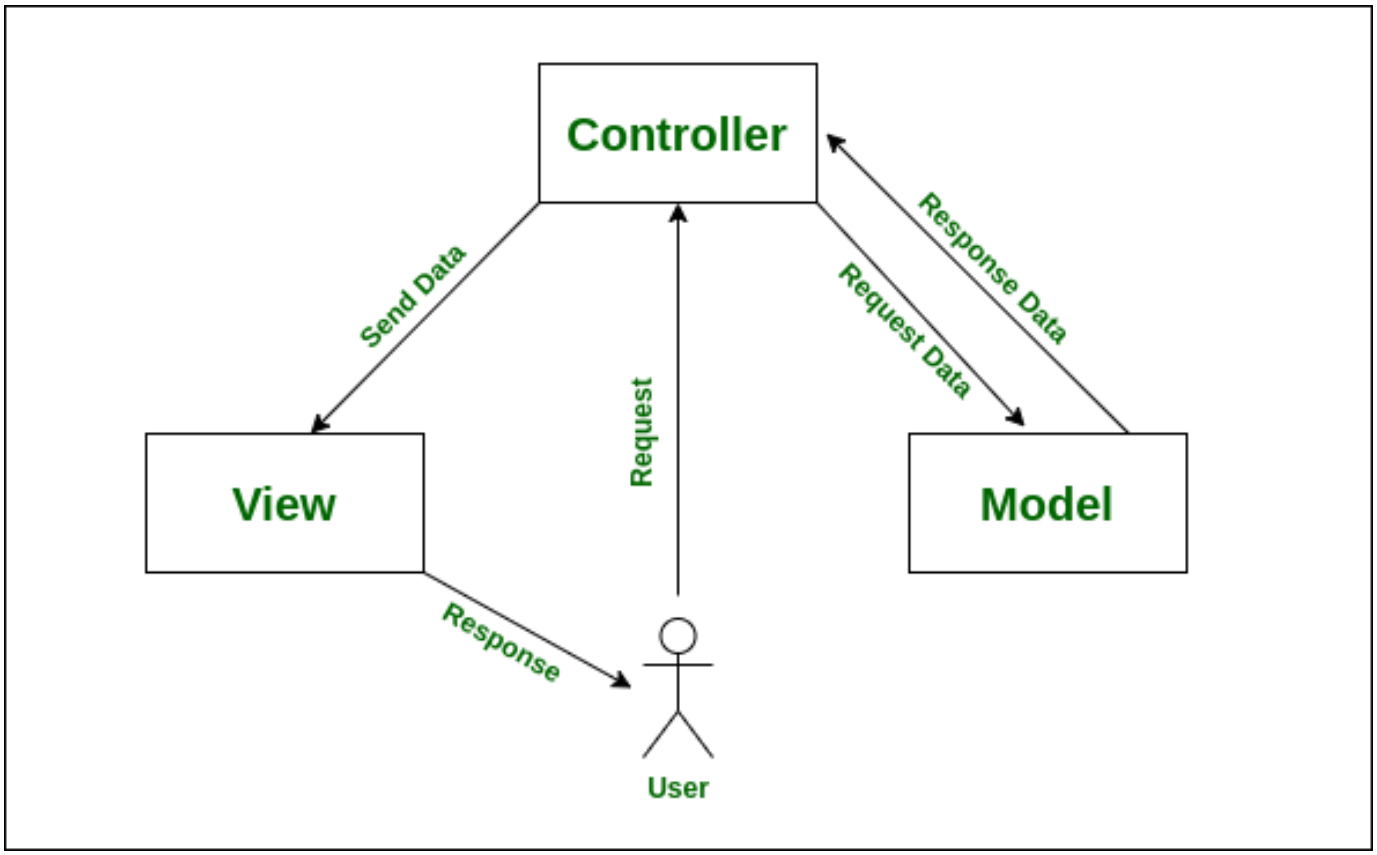
```
<h2>About Page</h2>
<p>{{ message }}</p> <!-- Message from controller -->
```

Now, when you navigate to **/#/about**, the message will change dynamically based on the controller.

Redirecting to a 404 Page (Optional)
If a user enters an **undefined route**, you can redirect them to a **custom 404 page**.
Modify app.js

```
$routeProvider
.otherwise({
    template: "<h2>404 - Page Not Found</h2><p>The page you are looking for
does not exist.</p>"
});
```

MVC ARCHITECTURE:



MVC (Model-View-Controller) is a design pattern used for structuring web applications.

Component	Description
Model	Manages data and business logic.
View	Displays data (UI - HTML pages).
Controller	Handles user interactions and updates the Model & View.

AngularJS Services

In AngularJS you can make your own service, or use one of the many built-in services.

What is a Service?
In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.
AngularJS has about 30 built-in services. One of them is the $location service.
The $location service has methods which return information about the location of the current web page:

Example
Use the $location service in a controller:

```
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $location) {
    $scope.myUrl = $location.absUrl();
});
```

The $http Service

The $http service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

Example
Use the $http service to request data from the server:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.html").then(function (response) {
    $scope.myWelcome = response.data;
  });
})
```

The $timeout Service
The $timeout service is AngularJS' version of the window.setTimeout function.

Example
Display a new message after two seconds:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $timeout) {
  $scope.myHeader = "Hello World!";
  $timeout(function () {
    $scope.myHeader = "How are you today?";
  }, 2000);
});
```

The $interval Service
The $interval service is AngularJS' version of the window.setInterval function.

Example
Display the time every second:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $interval) {
  $scope.theTime = new Date().toLocaleTimeString();
  $interval(function () {
    $scope.theTime = new Date().toLocaleTimeString();
  }, 1000);
});
```

Create Your Own Service
To create your own service, connect your service to the module:

```
Create a service named hexafy:
app.service('hexafy', function(){
  this.myFunc = function (x){
    return x.toString(16);
  }
});
```

To use your custom made service, add it as a dependency when defining the controller:

Example
Use the custom made service named hexafy to convert a number into a hexadecimal number:

```
app.controller('myCtrl', function($scope, hexafy){
  $scope.hex = hexafy.myFunc(255);
});
```

Filters in AngularJS

Filters in AngularJS are used to format data before displaying it in the view. They allow modification of data in expressions and directives like ng-repeat without changing the actual data in the model. Filters can be applied using the pipe (|) symbol in expressions.

Types of Filters in AngularJS

1. **Uppercase Filter (uppercase)**
   Converts a string to uppercase letters.
   Example:
   {{ "angularjs" | uppercase }} <!-- Output: ANGULARJS -->

2. **Lowercase Filter (lowercase)**
   Converts a string to lowercase letters.
   Example:

{{ "ANGULARJS" | lowercase }} <!-- Output: angularjs -->

3. **Currency Filter (currency)**
   Formats a number as a currency value. The default currency symbol is $.
   Example:
   {{ 2500 | currency }} <!-- Output: $2,500.00 -->

4. **Number Filter (number)**
   Formats a number with a specified number of decimal places.
   Example:
   {{ 1234.5678 | number:2 }} <!-- Output: 1,234.57 -->

5. **Date Filter (date)**
   Formats a date value in various formats.
   Example:
   {{ '2024-03-08' | date:'fullDate' }} <!-- Output: Friday, March 8, 2024 -->

6. **JSON Filter (json)**
   Converts an object into a JSON string format.
   Example:
   {{ {name: 'John', age: 30} | json }}
   Output:
   { "name": "John", "age": 30 }

7. **LimitTo Filter (limitTo)**
   Limits the number of items displayed in an array or the number of characters in a string.
   Example:
   {{ "AngularJS Filters" | limitTo:8 }} <!-- Output: AngularJ -->

8. **OrderBy Filter (orderBy)**
   Sorts an array based on a specific field.
   Example:<li ng-repeat="student in students | orderBy:'name'">{{ student.name }}
   This will display the student names in alphabetical order.

Directives in AngularJS

Directives in AngularJS are special attributes or elements that extend the functionality of HTML. They allow developers to create dynamic and reusable components, enhancing the behavior of HTML elements. AngularJS directives are prefixed with ng- and are used to bind data, manipulate the DOM, handle events, and implement reusable components.

Types of Directives in AngularJS

1. **Attribute Directives**
   These directives modify the behavior or appearance of an element.
   - **Example:** ng-class, ng-style, ng-show, ng-hide
   - **Example Usage:** This is a paragraph

2. **Structural Directives**
   These directives modify the structure of the DOM by adding or removing elements.
   - **Example:** ng-if, ng-repeat, ng-switch
   - **Example Usage:**
     ```
     <div ng-if="isLoggedIn">Welcome, User!</div>
     <ul>
       <li ng-repeat="item in items">{{ item }}</li>
     </ul>
     ```

3. **Component Directives**
   These directives are used to create custom components. They are defined using the directive function.
   - **Example Usage:**
     ```
     app.directive("customDirective", function() {
       return {
         template: "<h2>This is a custom directive</h2>"
       };
     });
     ```

4. **Event Directives**
   These directives handle user interactions like clicks and other events.
   - **Example:** ng-click, ng-change, ng-submit
   - **Example Usage:**
     ```
     <button ng-click="showMessage()">Click Me</button>
     ```

Data Binding in AngularJS

Data binding in AngularJS is the process of synchronizing data between the **Model (JavaScript variables)** and the **View (HTML template)**. This eliminates the need for manual DOM manipulation and makes the application dynamic and interactive.

Types of Data Binding in AngularJS

1. **One-Way Data Binding**
   - In one-way data binding, data flows only in one direction: from the model to the view or from the view to the model.
   - AngularJS supports **interpolation ({{ }})** and the ng-bind directive for displaying data.

   Example (Model to View):
   ```
   <p>{{ message }}</p>
   <p ng-bind="message"></p>
   ```
   - Here, if message = "Hello, AngularJS!" in the controller, both elements will display "Hello, AngularJS!".

   Example (View to Model):
   ```
   <input type="text" ng-model="username">
   <p>You entered: {{ username }}</p>
   ```
   - When the user types in the input box, the value of username updates, and the paragraph reflects the change.

2. **Two-Way Data Binding**
    - In two-way data binding, changes in the model update the view, and changes in the view update the model automatically.
    - The ng-model directive is used to achieve two-way binding.

    Example:

    ```
    <input type="text" ng-model="name">
    <p>Hello, {{ name }}!</p>
    ```
    - If the user types "John", the paragraph updates dynamically to "Hello, John!", and name in the model also updates.

Advantages of Data Binding in AngularJS

- Reduces the need for manual DOM manipulation.
- Enhances performance by automatically updating the UI when data changes.
- Improves code efficiency and readability.
- Provides a seamless user experience with real-time updates.

Controller in AngularJS

A **controller** in AngularJS is a JavaScript function that is responsible for handling application logic and controlling the flow of data between the **Model** (data) and the **View** (UI). It acts as a bridge between the data and the user interface, ensuring that changes in the model are reflected in the view.
Controllers are defined inside an AngularJS module and are associated with an HTML element using the ng-controller directive.

Creating a Controller in AngularJS

1. **Define an AngularJS Module and Controller**
2. **Bind the Controller to an HTML Element**

Example: AngularJS Controller

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

    <div ng-controller="myController">
        <h2>AngularJS Controller Example</h2>
        <p>Name: {{ student.name }}</p>
        <p>Age: {{ student.age }}</p>
        <button ng-click="updateAge()">Increase Age</button>
    </div>

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

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

```
            $scope.student = {
                name: "John Doe",
                age: 20
            };

            $scope.updateAge = function() {
                $scope.student.age += 1;
            };
        });
    </script>

</body>
</html>
```

Explanation

1. The **ng-app="myApp"** directive initializes an AngularJS application named "myApp".
2. The **ng-controller="myController"** directive binds the myController function to the element.
3. Inside the controller, the $scope object is used to store and manage data.
4. The student object contains name and age properties, which are displayed using {{ student.name }} and {{ student.age }}.
5. The updateAge function increments the age when the button is clicked, demonstrating how the controller updates the model dynamically.

Role of Controllers in AngularJS

- Manage application data and business logic.
- Bind data to the view using the $scope object.
- Handle user interactions and update the model accordingly.
- Maintain separation between the model and the view for better organization and reusability.

AngularJS Directives: ng-app, ng-model, and ng-bind

Directives in AngularJS are special attributes that extend the functionality of HTML by adding dynamic behavior to web applications. Below are the definitions and explanations of three key AngularJS directives:

1. ng-app Directive

- The ng-app directive is used to **define an AngularJS application** and link it to an HTML document.
- It tells AngularJS that this element (and its children) will be controlled by an AngularJS module.
- It is usually placed in the <html> or <body> tag.

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"></sc
ript>
</head>
<body>
    <h2>AngularJS App Example</h2>
</body>
</html>
```

Explanation:

- Here, ng-app="myApp" initializes an AngularJS application named "myApp".
- Without ng-app, AngularJS will not work on this page.

2. ng-model Directive

- The ng-model directive is used to **bind form inputs (such as text fields, checkboxes, and dropdowns) to variables in the controller's scope**.
- It enables **two-way data binding**, meaning any changes in the input field update the model, and vice versa.

Example:

```
<div ng-app="">
    <label>Enter Name:</label>
    <input type="text" ng-model="username">
    <p>Hello, {{ username }}!</p>
</div>
```

Explanation:

- The input field is bound to the username variable using ng-model.
- As the user types in the input field, the paragraph dynamically updates to reflect the entered text.

3. ng-bind Directive

- The ng-bind directive is used to **bind expressions to HTML elements**.
- It works similarly to {{ }} (interpolation), but it is preferred when dealing with dynamic content, as it avoids flickering during page load.

Example:

```
<div ng-app="" ng-init="message='Welcome to AngularJS'">
    <p ng-bind="message"></p>
</div>
```

Explanation:

- The ng-init directive initializes the variable message with the value "Welcome to AngularJS".
- The ng-bind="message" binds this value to the <p> tag, displaying "Welcome to AngularJS".

Error Handling in JavaScript

Error handling in JavaScript is essential to ensure smooth execution of programs and prevent unexpected crashes. JavaScript provides mechanisms to **catch, handle, and debug errors** using the try...catch statement, throw statement, and finally block.

Types of Errors in JavaScript

1. **Syntax Errors:** Occur when there is a mistake in the syntax of the code.
2. **Reference Errors:** Happen when a variable is used without being declared.
3. **Type Errors:** Occur when a value is not of the expected type.
4. **Range Errors:** Arise when a number is outside its allowed range.

Error Handling Mechanisms in JavaScript

1. Using try...catch

- The try block contains the code that may cause an error.
- The catch block executes if an error occurs, preventing the program from crashing.

Example:

```
try {
    let result = x + 10; // x is not defined
} catch (error) {
    console.log("An error occurred: " + error.message);
}
```

Explanation:

- Since x is not defined, a ReferenceError occurs.
- The catch block catches the error and logs an appropriate message.

---

2. Using throw Statement

- The throw statement allows you to create custom error messages and throw exceptions manually.

Example:

```
function divide(a, b) {
    if (b === 0) {
        throw new Error("Division by zero is not allowed.");
    }
    return a / b;
}

try {
    console.log(divide(10, 0));
} catch (error) {
    console.log("Error: " + error.message);
}
```

Explanation:

- If b is 0, the function throws an error.
- The try...catch block handles the error and displays a custom message.

---

3. Using finally Block

- The finally block executes **whether an error occurs or not**.
- It is useful for cleanup operations like closing files or clearing memory.

Example:

```
try {
    console.log("Trying to execute code...");
    let x = 5 / 0; // Infinity, but no error in JavaScript
} catch (error) {
    console.log("Error: " + error.message);
} finally {
    console.log("Finally block executed.");
}
```

Explanation:

- The finally block always runs, regardless of whether an error occurred or not.