UNIT 5

HTML 5, CSS 3 with Bootstrap, JavaScript ES6. Angular JS

Introduction to HTML 5

CSS 3 with Bootstrap

JavaScript ES6

Getting started with Node.js

Introduction To Angular JS

Angular JS Modules Directives & Building Databases.

Q: Define Angular JS. Explain features of Angular JS. –3

AngularJS is a structural framework for dynamic web applications. It allows you to use HTML as your template language and extend HTML's syntax to express your application's components clearly and succinctly. Data binding and dependency injection eliminate much of the code you would otherwise have to write.

Features of AngularJS

1. **MVC Architecture**: AngularJS is built around the Model-View-Controller (MVC) design pattern. This helps in separating the logic, data, and presentation layer of an application, making it easier to manage and scale.
2. **Data Binding**: Two-way data binding is one of the most notable features of AngularJS. It means that any changes in the model are immediately reflected in the view and vice versa, providing a responsive experience.
3. **Directives**: AngularJS extends HTML with new attributes called directives. Directives are special markers on DOM elements (such as attributes, element names, comments, or CSS classes) that tell the compiler to attach a specified behaviour to that DOM element (or even transform the DOM element and its children).
4. **Dependency Injection**: AngularJS has a built-in dependency injection subsystem that helps manage the dependencies of various components in an application. This allows you to develop, understand, and test the code more easily.
5. **Templates**: AngularJS uses HTML templates to define the user interface of an application. These templates are parsed by the browser and turned into the DOM, which is then rendered as the user interface. The templates are compiled in the browser, providing a dynamic user experience.

6. **Modules**: AngularJS uses modules to separate different parts of an application into reusable components. This modular approach helps in organizing the code better and makes it more maintainable.
7. **Services**: AngularJS provides several built-in services (e.g., $http for making XMLHttpRequests). Services in AngularJS are singleton objects or functions that carry out specific tasks and are typically used for sharing data and functions across an application.
8. **Routing**: The ngRoute module in AngularJS allows developers to create different views for different URLs, enabling single-page applications (SPAs). This helps in managing different states and views in an application.
9. **Filters**: Filters in AngularJS allow you to format data displayed to the user. Filters can be used in views, controllers, or services, and they can perform tasks like formatting dates, filtering arrays, or converting text to uppercase.
10. **Testability**: AngularJS is designed with testability in mind, making it easier to write tests for various components. It supports both unit testing and end-to-end testing.
11. **Form Validation**: AngularJS provides client-side form validation. It monitors the state of the form and input fields (e.g., if they have been modified, if they are valid, etc.) and provides visual feedback to the user.
12. **Built-in Services and Factories**: AngularJS provides several built-in services and factories like $http for AJAX calls, $location for URL management, and more, simplifying common tasks in web development.

Q: Explain the following Angular JS Directives with an example: i. ng-app ii. ng-init.—3

AngularJS directives are special markers or attributes added to HTML elements that extend the capabilities of HTML. They allow you to create reusable components, manage data binding, and handle user interactions more effectively. Here's a comprehensive look at some of the most commonly used AngularJS directives:

1. ng-app

Definition: The ng-app directive is used to define the root element of an AngularJS application. It initializes the AngularJS framework and denotes the scope of the Angular application.

```
<!DOCTYPE html>
<html>
<head>
   <title>AngularJS ng-app Example</title>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-app="myApp">
   <h1>Welcome to AngularJS Application</h1>
   <div>
      <p>This is a simple AngularJS application.</p>
   </div></body> </html>
```

2. ng-init

Definition: The ng-init directive initializes AngularJS application variables and sets their values. It is typically used to initialize data in the scope.

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

   <div ng-init="firstName='John'; lastName='Doe'">
      <p>First Name: {{ firstName }}</p>
      <p>Last Name: {{ lastName }}</p>
   </div>
</body>
</html>
```

1. ng-app

Definition: Defines the root element of an AngularJS application. Purpose: Initializes the AngularJS framework and specifies the scope of the Angular application.

2. ng-model

Definition: Binds the value of HTML controls (like input, select, and textarea) to application data. Purpose: Creates a two-way data binding between the view (HTML) and the model (application data).

3. ng-bind

Definition: Replaces the content of an HTML element with the value of a given expression. Purpose: Provides a more readable and less cluttered way to bind data to the view compared to using double curly braces {{}}.

4. ng-controller

Definition: Attaches a controller class to the view. Purpose: Defines the scope of the controller and binds it to a section of the DOM, enabling data binding and functionality defined in the controller.

5. ng-repeat

Definition: Repeats a set of HTML elements for each item in a collection. Purpose: Allows iteration over an array or object to create multiple elements dynamically based on the data.

6. ng-show / ng-hide

Definition: Conditionally shows or hides an HTML element. Purpose: Provides a way to display or hide elements based on the value of a boolean expression.

7. ng-if

Definition: Removes and recreates the element in the DOM based on the condition. Purpose: More efficient than ng-show/ng-hide because it physically adds or removes the element from the DOM, reducing the performance overhead.

8. ng-class

Definition: Dynamically binds one or more CSS classes to an HTML element. Purpose: Allows conditional styling of elements based on expressions.

9. ng-style

Definition: Dynamically binds one or more CSS styles to an HTML element. Purpose: Provides a way to set the style properties of an element dynamically based on expressions.

10. ng-click

Definition: Defines behavior for the click event. Purpose: Binds a function or expression to be executed when the element is clicked.

11. ng-submit

Definition: Binds an expression to the submit event of a form. Purpose: Provides a way to handle form submissions using AngularJS.

12. ng-include

Definition: Includes an external HTML fragment. Purpose: Allows for the inclusion of external HTML files into the current HTML document, facilitating the reuse of code.

13. ng-options

Definition: Populates a <select> element with options. Purpose: Simplifies the creation of dropdown menus by binding them to an array or object in the model.

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

```
    <h1>Hello, {{name}}!</h1>
    <input type="text" ng-model="name">
    <p ng-bind="name"></p>

    <h2>Item List</h2>
    <ul>
        <li ng-repeat="item in items">{{item.name}}</li>
    </ul>

    <p ng-show="isVisible">This paragraph is visible.</p>
    <p ng-hide="isVisible">This paragraph is hidden.</p>
    <button ng-click="toggleVisibility()">Toggle Visibility</button>

    <h2>Submit Form</h2>
    <form ng-submit="submitForm()">
        <input type="text" ng-model="username" placeholder="Enter your username">
        <button type="submit">Submit</button>
    </form>

    <div ng-include="'header.html'"></div>

    <h2>Select an Item</h2>
    <select ng-model="selectedItem" ng-options="item.name for item in items"></select>
    <p>You selected: {{selectedItem.name}}</p>

    <div ng-class="{'active': isActive, 'disabled': isDisabled}">
        This div has dynamic classes.
    </div>
    <button ng-click="toggleClass()">Toggle Class</button>

    <div ng-style="{'color': color, 'font-size': size + 'px'}">
        This div has dynamic styles.
    </div>
    <button ng-click="changeStyle()">Change Style</button>
</body>
</html>
```

```
angular.module('myApp', [])
.controller('MyController', function($scope) {
    $scope.name = 'John Doe';
    $scope.items = [
        {name: 'Item 1'},
        {name: 'Item 2'},
        {name: 'Item 3'}
    ];
    $scope.isVisible = true;
    $scope.toggleVisibility = function() {
        $scope.isVisible = !$scope.isVisible;
    };
    $scope.submitForm = function() {
        alert('Form submitted with username: ' + $scope.username);
    };
    $scope.selectedItem = $scope.items[0];
```

```
    $scope.isActive = true;
    $scope.isDisabled = false;
    $scope.toggleClass = function() {
        $scope.isActive = !$scope.isActive;
        $scope.isDisabled = !$scope.isDisabled;
    };

    $scope.color = 'blue';
    $scope.size = 14;
    $scope.changeStyle = function() {
        $scope.color = $scope.color === 'blue' ? 'red' : 'blue';
        $scope.size = $scope.size === 14 ? 20 : 14;
    };
});
```

Explain Angular JS Modules —3

AngularJS modules are containers for different parts of an application, such as controllers, services, filters, directives, etc. They help in organizing the code and making it modular and maintainable. Each module is defined using the angular. Module function.

Key Features of AngularJS Modules

1. **Encapsulation**:
    - Modules help encapsulate code into functional units, which makes the application easier to manage and scale.
2. **Dependency Injection**:
    - Modules enable dependency injection, allowing the various components (like controllers and services) to be easily managed and injected where needed.
3. **Reusability**:
    - Modules promote reusability by allowing you to define reusable pieces of code that can be shared across different parts of the application or even across different applications.
4. **Separation of Concerns**:
    - Modules help in separating concerns by grouping related components together. For example, controllers dealing with user management can be grouped into a user module.

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <title>AngularJS Modules Example</title>
```

```
    <script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
    <script src="app.js"></script>
  </head>
  <body>
    <div ng-controller="MainController">
      <h1>{{message}}</h1>
    </div>
    <div ng-controller="SecondaryController">
      <h2>{{secondaryMessage}}</h2>
    </div>
  </body></html>
```

```
// Define the main module
angular.module('myApp', [])
// Define the main controller
.controller('MainController', function($scope) {
  $scope.message = 'Hello from the Main Controller!';
})
// Define another controller
.controller('SecondaryController', function($scope) {
  $scope.secondaryMessage = 'Hello from the Secondary Controller!';
});
```

Differentiate among JavaScript and Node.js.

Feature	*JavaScript (Browser)*	*Node.js*
Definition	High-level, interpreted scripting language for web pages.	Server-side runtime environment for executing JavaScript.
Execution Environment	Runs in web browsers.	Runs on servers or command-line interfaces.
Primary Use	Front-end development, creating interactive web pages.	Backend development, handling server-side logic and APIs.
Execution Context	Client-side, interacts with the DOM and browser APIs.	Server-side, handles server requests and responses.
APIs Available	Browser-specific APIs (e.g., document, window, localStorage).	Server-specific APIs (e.g., fs for file system, http for HTTP server).
Module System	No native module system; uses tools like Webpack or modern ES6+ modules.	CommonJS modules (using require and module.exports) and ES Modules (using import and export).
Concurrency Model	Single-threaded, handled via events and callbacks.	Non-blocking I/O with an event-driven, single-threaded architecture.

Package Management	Not applicable.	Uses npm (Node Package Manager) for managing packages and dependencies.
Typical Applications	Form validation, animations, dynamic content updates.	Building web servers, APIs, real-time applications, interacting with databases.
Standard Library	Limited to browser APIs.	Extensive built-in modules for server-side operations.
I/O Operations	Typically relies on AJAX or Fetch API for asynchronous operations.	Provides non-blocking I/O operations, facilitating high concurrency.
Development Tools	Browser Developer Tools, transpilers like Babel.	Node Package Manager (npm), various CLI tools, build tools like Webpack.

Design the code for simple "Hello World" Angular JS application and showcase the Model, View and Controller part in it ---2

To design a simple "Hello World" AngularJS application that showcases the Model, View, and Controller (MVC) pattern, follow these steps:

1. Setup the Project

Create the following files:

- index.html (for the view)
- app.js (for the AngularJS module and controller)

2. index.html (View)

This file defines the structure of your web page and integrates AngularJS.

```
<!DOCTYPE html>
<html ng-app="helloWorldApp">
<head>
    <title>Hello World AngularJS</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
    <script src="app.js"></script>
</head>
<body ng-controller="HelloController">
    <h1>{{ greeting }}</h1>
    <input type="text" ng-model="userName" placeholder="Enter your name">
    <p>Hello, {{ userName }}!</p>
</body>
</html>
```

3. app.js (Controller and Module)

This file defines the AngularJS module and the controller. The controller will manage the application's data (Model) and behavior.

```
// Define the AngularJS module
angular.module('helloWorldApp', [])
.controller('HelloController', function($scope) {
    // Model: Define data and initialize it
    $scope.greeting = 'Hello, World!'; // Data binding for the view
    // Initialize model
    $scope.userName = ''; // Bind this to the input field});
```

Explanation

1. Model

- **Defined in app.js**:
- **Purpose**: Holds the data used by the controller and the view. In this example, greeting and userName are part of the model.

```
$scope.greeting = 'Hello, World!';
$scope.userName = '';
```

2. View

- **Defined in index.html**:
- **Purpose**: Displays the data from the model and binds user inputs. {{ greeting }} and {{ userName }} are AngularJS expressions that display data from the model.

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

3. Controller

- **Defined in app.js**:
- **Purpose**: Manages the application's logic. It initializes the data and updates it based on user interactions.

```
angular.module('helloWorldApp', [])

.controller('HelloController', function($scope) {

    $scope.greeting = 'Hello, World!';

    $scope.userName = '';

});
```

Model: Contains the application's data ($scope.greeting and $scope.userName).

View: Defines how the data is presented to the user and allows for user input (index.html).

Controller: Manages the interaction between the model and view (app.js).

Write a note on Node.js. ---2

Node.js is a powerful, open-source runtime environment that allows JavaScript to be used for server-side programming. It is built on Chrome's V8 JavaScript engine and provides an event-driven, non-blocking I/O model that makes it efficient and suitable for scalable network applications.

Key Features

1. **JavaScript Runtime**:
    - Node.js extends JavaScript's use from the client-side (browsers) to the server-side. It enables developers to use JavaScript for writing server-side code, leveraging the same language for both front-end and back-end development.
2. **Event-Driven Architecture**:
    - Node.js employs an event-driven architecture, meaning it uses events and callbacks to handle asynchronous operations. This allows it to handle multiple tasks concurrently without blocking the execution of other operations, leading to high performance and responsiveness.
3. **Non-Blocking I/O**:
    - The non-blocking, asynchronous nature of Node.js allows it to perform I/O operations (such as reading files, querying databases) without waiting for them to complete. This contributes to its efficiency in handling numerous connections simultaneously.
4. **Single-Threaded**:
    - Node.js operates on a single-threaded event loop, which can handle many connections efficiently. This design simplifies the development model by avoiding the complexity of multi-threaded programming.

5. **npm (Node Package Manager)**:
    - Node.js comes with npm, the world's largest ecosystem of open-source libraries and modules. Developers can easily install and manage packages and dependencies for their projects using npm, fostering a rich development environment.
6. **Cross-Platform**:
    - Node.js is cross-platform, meaning it can run on various operating systems, including Windows, macOS, and Linux. This makes it versatile and suitable for different development environments.
7. **Scalability**:
    - Node.js is designed to be scalable. Its non-blocking I/O operations and event-driven nature allow it to handle large volumes of concurrent connections, making it well-suited for building scalable network applications.
8. **Real-Time Capabilities**:
    - It excels in real-time applications such as chat applications or live updates due to its event-driven nature and WebSocket support, which allows for persistent connections and real-time communication.

Typical Use Cases

1. **Web Servers**:
    - Node.js can be used to build web servers that handle HTTP requests and serve content. Its non-blocking I/O is particularly effective for handling high-traffic websites and APIs.
2. **APIs**:
    - It is commonly used for creating RESTful APIs and microservices. Its asynchronous capabilities and lightweight nature make it ideal for handling API requests and responses.
3. **Real-Time Applications**:
    - Node.js is well-suited for applications that require real-time data processing, such as online gaming, chat applications, and live data feeds.
4. **Streaming Applications**:
    - Due to its efficient handling of data streams, Node.js is used in applications that require continuous data streaming, such as video or audio streaming services.
5. **Single Page Applications (SPAs)**:

- Node.js can serve as the backend for SPAs built with frameworks like React or Angular, managing API requests and real-time interactions.

```
// Import the HTTP module
const http = require('http');
// Define the hostname and port
const hostname = '127.0.0.1';
const port = 3000;

// Create an HTTP server
const server = http.createServer((req, res) => {
    // Set the response header
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    // Send the response body
    res.end('Hello, World!\n');
});
// Make the server listen on the specified port
server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

Explain how to redirect to a particular section of a page using HTML with an example.

Redirecting to a particular section of a page in HTML is commonly achieved using **anchor links**. This method involves creating a hyperlink that points to an element with a specific id attribute within the same page. When the link is clicked, the browser scrolls to the section of the page where the element with that id is located.

Steps to Redirect to a Particular Section

1. **Create the Target Section**: Add an id attribute to the HTML element where you want to scroll to.
2. **Create the Link**: Add an anchor (<a>) tag that references the id of the target section.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Anchor Link Example</title>
```

```
    <style>
      section {
        height: 600px;
        border: 1px solid #ddd;
        padding: 20px;
        margin-bottom: 20px;
      }
    </style>
  </head>
  <body>

    <!-- Navigation Links -->
    <nav>
      <ul>
        <li><a href="#section1">Go to Section 1</a></li>
        <li><a href="#section2">Go to Section 2</a></li>
        <li><a href="#section3">Go to Section 3</a></li>
      </ul>
    </nav>

    <!-- Sections -->
    <section id="section1">
      <h2>Section 1</h2>
      <p>This is Section 1.</p>
    </section>

    <section id="section2">
      <h2>Section 2</h2>
      <p>This is Section 2.</p>
    </section>

    <section id="section3">
      <h2>Section 3</h2>
      <p>This is Section 3.</p>
    </section>

  </body>
  </html>
```