UNIT-IV

1. Introduction to Angular JS:

Ans: Angular is a framework used for building web applications. It was created to make it easier for developers to build structured and organized applications. Originally called AngularJS, it was released in 2010 and quickly became popular due to its ability to create single-page applications. AngularJS underwent a major update and is now simply called Angular. The latest version introduced new ideas and improved the framework for modern web development. Angular 2 is the next version of the framework, which was created to be better and easier to use than Angular 1. There are some important differences between Angular 1 and 2:

- **Syntax:** Angular 2 uses a new version of the JavaScript language called ES2015, but not all browsers support it yet. So Angular 2 uses TypeScript, helps you write better code.
- **Modules:** In Angular 2, modules are easier to use because they are based on a built-in module system in JavaScript called ES2015.
- **Controllers:** In Angular 1, controllers were the main way of building components, but in Angular 2, components are the basic building block.
- **Scopes:** Angular 2 has a cleaner and more readable component model that does not rely on the famous $scope object from Angular 1.
- **Decorators:** Angular 2 uses a feature called decorators that allows developers to add features or data to classes and members without extending them.
- **Dependency Injection:** Angular 2 has made dependency injection simpler and now supports multiple injectors instead of one.

2. Key concepts of Angular JS

Ans: Some of the key features of Angular JS include:

- **Directives:** Directives are markers on a DOM element that tell AngularJS to attach a specific behavior to that element.
- **Modules:** Modules are containers for related code, such as controllers, services, and directives and they help keep code organized and modular.
- **Controllers:** Controllers are responsible for managing the application's data and logic and also interact with the view through scope objects.
- **Scope:** Scope objects are used to communicate between the controller and the view.
- **Services:** Services are singleton objects that provide functionality across the application.
- **Filters:** Filters are used to format data before it is displayed to the user and they can be used to sort, filter, or transform data in a variety of ways.
- **Dependency Injection:** Dependency injection is a design pattern used to manage object dependencies and it allows objects to be easily tested and decoupled from one another.

3. Angular JS modules

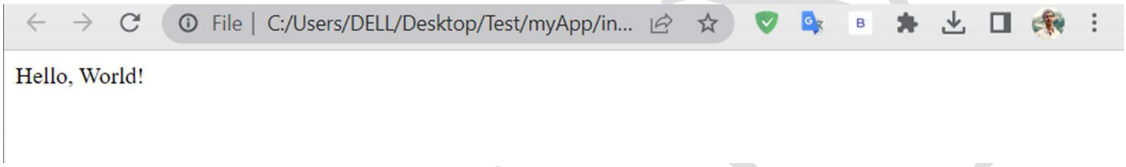
Ans: In AngularJS, a module is a container for different components of an application such as controllers, services, directives, filters, etc. It is essentially a container of related functionality that can be organized into a single, standalone unit. A module can depend on other modules. This means that a module can reuse code from other modules or be reused by other modules. Creating a module in AngularJS is simple and can be done using the **angular.module()** method. This method takes two arguments: the name of the module and an array of dependencies (if any).

Program:

```
<!DOCTYPE html>
<html 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">
        {{ message }}
      </div>
      <script>
        angular.module('myApp', []).controller('myController',
  function($scope) {
          $scope.message = 'Hello, World!';
        });
      </script>
    </body>
  </html>
```

Output:



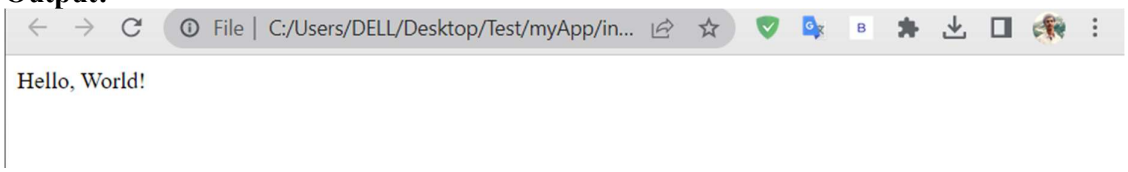
4. Dependency Injection in Angular JS

Ans: Dependency Injection is a design pattern that helps to manage dependencies between objects. It is a built-in feature that helps to create loosely coupled components and make them more testable and maintainable. In AngularJS, a component's dependencies are specified as arguments to its constructor function. AngularJS then resolves these dependencies and passes them to the component instance. This approach allows components to be easily configured and reused, making them more modular.

Program:

```
<!DOCTYPE html>
<html ng-app="myApp">
  <head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.j
s"></script>
  </head>
  <body>
    <div ng-controller="myController">
      {{ message }}
    </div>
    <script>
      angular.module('myApp', []).controller('myController',
['$scope', 'Greet', function($scope, Greet) {
        $scope.message = Greet.sayHello('World');
      }]).service('Greet', function() { this.sayHello = function(name)
{
            return 'Hello, ' + name + '!';
        };
      });
    </script>
  </body>
</html>
```

Output:



5. Bootstrapping an Angular JS application

Ans: Bootstrapping an AngularJS application is the process of initializing and configuring the application. AngularJS provides the ng-app directive, which is used to define the root element of the application. When the page is loaded, AngularJS searches for the ng-app directive and then automatically initializes the application.

Program:

Refer 3.

6. Installing Angular JS

Ans: To install AngularJS, you can use either of the following methods:

- You can include the AngularJS library in your HTML file by adding the following script tag:
  ```
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  ```
- You can download the AngularJS library from the official website at https://angularjs.org/ and include it in your HTML file by adding the following script tag:
  ```
  <script src="path/to/angular.js"></script>
  ```

7. Structuring an Angular JS application

Ans: Structuring an AngularJS application is important to ensure the application is maintainable, scalable, and easy to navigate. There are multiple ways we can structure applications and of course different structures are better suited for different applications.

Ex:

- app/
  - css/
    - style.css
  - js/
    - app.js
    - controllers.js
    - services.js
    - directives.js
  - views/
    - home.html
  - index.html

8. Angular JS MVC entities

Ans: AngularJS uses the Model-View-Controller architecture to organize an application's codebase into three interconnected components:

- **Model:** This represents the data and business logic of the application. In AngularJS, models are often created using JavaScript objects.
- **View:** This represents the user interface of the application. Views are typically created using HTML, CSS, and AngularJS directives, which are special HTML attributes that bind AngularJS expressions to elements in the view.
- **Controller:** This is the glue between the model and the view. Controllers are responsible for setting up the initial state of the model and handling user interactions. In AngularJS, controllers are created using JavaScript functions.

VINY

By using the MVC pattern, developers can organize their code in a modular and maintainable way, which makes it easier to build complex applications. In AngularJS, the $scope object serves as the glue between the model and the view, allowing developers to easily bind data from the model to the view and handle user input in the controller.

9. Angular JS views, controllers and scopes

Ans: In AngularJS, views, controllers, and scopes are the key components of the Model-View-Controller architecture used for building dynamic web applications. Here's a brief overview of each component:

- **Views:** Views in AngularJS are the user interface components of the application, typically written in HTML, that display data and respond to user events. Views can be static, such as a simple HTML file, or dynamic, generated using AngularJS directives and expressions.
- **Controllers:** Controllers in AngularJS are responsible for handling user input, manipulating data models, and updating views. They serve as the glue between the view and the model, providing a way to respond to user events and modify the state of the application.
- **Scopes:** Scopes in AngularJS provide a context for the application data, allowing views and controllers to communicate and interact with the data models. Scopes are objects that contain application data and methods, and can be nested to create a hierarchy of scope objects.

Overall, the Views are responsible for displaying data to the user, controllers handle user input and modify the data model, and scopes provide the context for the data and methods used by both views and controllers.

10. Angular JS routing

Ans: AngularJS routing is a powerful feature that allows you to build Single Page Applications by providing a way to map URLs to different views and controllers. It allows you to define different routes for different URLs and bind controllers and views to them. Here's a simple example of how to define a route in AngularJS:

Program:

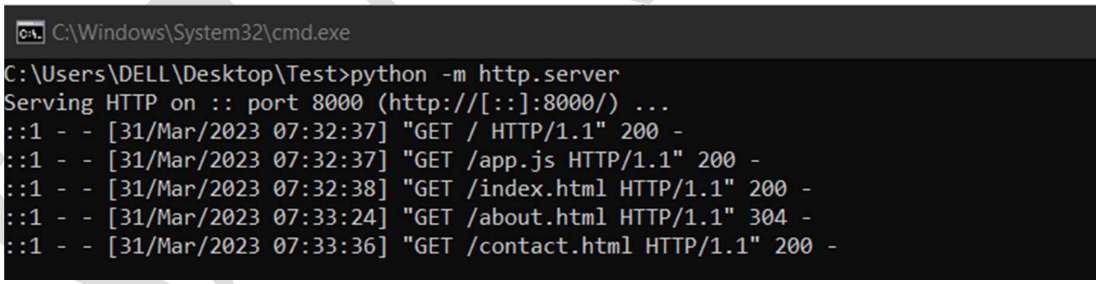
```
//index.html
<!DOCTYPE html>
<html>
<head>
   <title>My AngularJS App</title>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min
.js"></script>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-
route.min.js"></script>
   <script src="app.js"></script>
</head>
<body ng-app="myApp">
   <div ng-view></div>
</body>
</html>

//about.html
<h3>About Us</h3>
<p>We are a company that makes things happen.</p>

//contact.html
<h3>Contact Us</h3>
<p>Phone: 123-456-7890</p>
<p>Email: info@mycompany.com</p>
```

```
//app.js
var app = angular.module('myApp', ['ngRoute']);
app.config(function($routeProvider) {
  $routeProvider
  .when("/", {
    templateUrl: "index.html"
  })
  .when("/about", {
    templateUrl: "about.html"
  })
  .when("/contact", {
    templateUrl: "contact.html"
  });
});
```

Output:



11. Angular JS services

Ans: In AngularJS, services are objects that provide some functionality and can be injected into other components such as controllers, directives, and filters. Services are used to abstract common functionality that needs to be reused across different parts of an application.

Program:

```
//login.html
<div ng-controller="LoginController">
  <form ng-submit="login()">
    <label>Username:</label>
    <input type="text" ng-model="username" required>
    <label>Password:</label>
    <input type="password" ng-model="password" required>
```

```
    <button type="submit">Login</button>
  </form>
  <p ng-show="errorMessage">{{ errorMessage }}</p>
</div>

//app.js
app.controller('DashboardController', function($scope, $http) {
  $http.get('/api/user')
    .then(function(response) {
      $scope.message = 'Welcome ' + response.data.username;
    }, function(error) {
      window.location.href = '/login';
    });

  $scope.logout = function() {
    $http.get('/api/logout')
      .then(function(response) {
        window.location.href = '/login';
      });
  };
});
```

Output:

12. Managing Angular JS authentication

Ans: Managing authentication in an AngularJS application involves implementing a system for registering and authenticating users. This can be done using various techniques, including session management, JSON Web Tokens (JWTs), or OAuth2. To manage authentication in AngularJS, you can follow these basic steps:

- Create a login form: Create a login form in HTML that takes in a username and password.
- Create an API endpoint: Create an API endpoint on your server that accepts a POST request with the username and password from the login form.
- Send a POST request to the API endpoint: In your AngularJS application, use the $http service to send a POST request to the API endpoint with the username and password.
- Store the authentication token: When the API endpoint returns a successful response, it should also return an authentication token. Store this token in a cookie or local storage.]
- Check for the authentication token: When the user navigates to a protected page, check for the authentication token in the cookie or local storage. If the token is present, allow the user to access the page. If the token is not present, redirect the user to the login page.
- Implement logout functionality: Create an API endpoint that the AngularJS application can send a logout request to. When the logout request is sent, clear the authentication token from the cookie or local storage.
- Use authorization: Use the authentication token to authorize access to protected resources on the server.

VNNY