

Module-1

1. Creates a simple AngularJS application with a controller

This code creates a simple AngularJS application with a controller (myController). It contains an input field to enter a number and displays the squared value of that number using two-way data binding provided by AngularJS.

To run this code, create two files named index.html and app.js. Place the respective code blocks in each file. Then, open index.html in a web browser. You'll see an input field where you can enter a number, and the squared value of that number will be displayed in real-time below it.

HTML (index.html):

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS MVC 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>AngularJS MVC Example</h1>
  <div>
    <label>Enter a number: </label>
    <input type="number" ng-model="numberInput" ng-change="updateResult()">
  </div>
  <div>
    <p>Number entered: {{ numberInput }}</p>
    <p>Squared value: {{ squaredValue }}</p>
  </div>
</body>
</html>
```

JavaScript (app.js):

```
var myApp = angular.module('myApp', []);
myApp.controller('myController', function($scope) {
  $scope.numberInput = 0;
  $scope.squaredValue = 0;
  $scope.updateResult = function() {
    $scope.squaredValue = $scope.numberInput * $scope.numberInput;
  };
});
```

OUTPUT

File | C:/Users/OPERASPACE/Desktop/atest/index.html

AngularJS MVC Example

Enter a number:

Number entered: 5

Squared value: 25

2. Creates an AngularJS application with a controller (myController) and demonstrates the use of expressions, directives like ng-show, ng-repeat, and a controller function (showMessages) triggered by a button click.

The application displays a welcome message for a user, a hidden message count initially, and an empty list of messages. Clicking the "Show Messages" button populates the messages and displays the count of new messages. The directives like ng-show control the visibility of elements based on conditions, and ng-repeat helps to iterate over the list of messages.

To run this code, create two files named index.html and app.js, and paste the respective code blocks into each file. Then, open index.html in a web browser to see the AngularJS application in action.

HTML (index.html):

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS Expressions, Directives, and Controllers 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>AngularJS Expressions, Directives, and Controllers</h1>
  <div>
    <p>Welcome, {{ username }}!</p>
    <p ng-show="showMessage">You have {{ messages.length }} new messages!</p>
    <ul>
      <li ng-repeat="message in messages">{{ message }}</li>
    </ul>
    <button ng-click="showMessages()">Show Messages</button>
  </div>
</body>
</html>
```

JavaScript (app.js):

```
var myApp = angular.module('myApp', []);
myApp.controller('myController', function($scope) {
  $scope.username = "User123";
  $scope.showMessage = false;
  $scope.messages = [];

  $scope.showMessages = function() {
    $scope.messages = ["Message 1", "Message 2", "Message 3"];
    $scope.showMessage = true;
  };
});
```

← ↻ 📄 File | C:/Users/OPERASPACE/Desktop/atest/index.html

AngularJS Expressions, Directives, and Controllers

Welcome, User123!

Show Messages

← ↻ 📄 File | C:/Users/OPERASPACE/Desktop/atest/index.html

AngularJS Expressions, Directives, and Controllers

Welcome, User123!

You have 3 new messages!

- Message 1
- Message 2
- Message 3

Show Messages

Module-2

3.Create angularjs Forms and perform

1. Form Validation

2. Error Handling with Forms

3. Nested Forms with ng-form

This code showcases two nested forms within a main form. Each nested form (userForm and addressForm) has its own validation rules and error handling. The mainForm wraps these nested forms and is considered invalid if any of the nested forms are invalid. The submit button will be disabled until the entire form is valid..

HTML (index.html):

```
<!DOCTYPE html>
<html ng-app="formApp">
<head>
  <title>AngularJS Nested Forms</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script src="app.js"></script>
  <style>
    .error {
      color: red;
    }
  </style>
</head>
<body>
  <div ng-controller="formController">
    <form name="mainForm" novalidate>
      <h2>Main Form</h2>
      <div ng-form="userForm" name="userForm">
        <h3>User Information</h3>
        <div>
          <label>Name:</label>
          <input type="text" name="name" ng-model="user.name" required>
          <span class="error" ng-show="userForm.name.$touched && userForm.name.$error.required">Name is
required</span>
        </div>
        <div>
          <label>Email:</label>
          <input type="email" name="email" ng-model="user.email" required>
          <span class="error" ng-show="userForm.email.$touched && userForm.email.$error.required">Email is
required</span>
          <span class="error" ng-show="userForm.email.$error.email && !userForm.email.$error.required">Invalid
email</span>
        </div>
      </div>
      <div ng-form="addressForm" name="addressForm">
        <h3>Address Information</h3>
        <div>
          <label>Address Line:</label>
          <input type="text" name="address" ng-model="user.address" required>
          <span class="error" ng-show="addressForm.address.$touched && addressForm.address.$error.required">Address
is required</span>
        </div>
        <div>
          <label>City:</label>
          <input type="text" name="city" ng-model="user.city" required>
          <span class="error" ng-show="addressForm.city.$touched && addressForm.city.$error.required">City is
required</span>
        </div>
      </div>
    </form>
  </div>
</body>
</html>
```

```

    </div>
    <div>
      <button ng-click="submitForm()" ng-disabled="mainForm.$invalid">Submit</button>
    </div>
  </form>
  <div ng-show="submitted">
    <h3>Form Submitted Successfully!</h3>
    <p>Name: {{ user.name }}</p>
    <p>Email: {{ user.email }}</p>
    <p>Address: {{ user.address }}</p>
    <p>City: {{ user.city }}</p>
  </div>
</div>
</body>
</html>

```

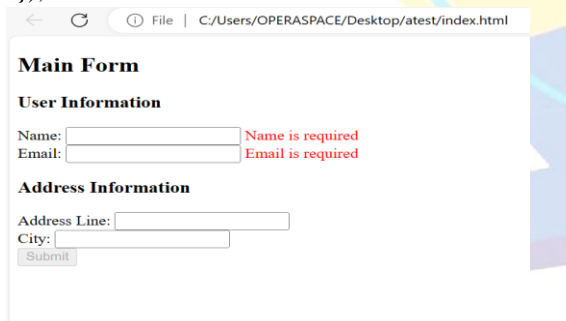
JAVASCRIPT(app.js)

```

angular.module('formApp', [])
.controller('formController', function($scope) {
  $scope.user = {};
  $scope.submitted = false;

  $scope.submitForm = function() {
    if ($scope.mainForm.$valid) {
      // Form is valid, perform submission logic here
      $scope.submitted = true;
    } else {
      // Handle errors or display appropriate messages
      // For example, mark the fields as touched to display errors
      $scope.mainForm.$setSubmitted();
    }
  };
});

```



File | C:/Users/OPERASPACE/Desktop/atest/index.html

Main Form

User Information

Name: Name is required

Email: Email is required

Address Information

Address Line:

City:

File | C:/Users/OPERASPACE/Desktop/atest/index.html

Main Form

User Information

Name:

Email:

Address Information

Address Line:

City:

Form Submitted Successfully!

Name: MMEC

Email: mmec@gmail.com

Address: Belagavi

City: Belagavi

Module-3

4. Demonstrates how to use filters within controllers and services in an AngularJS application:

- An input field for searching names.
- A list generated using ng-repeat that filters the ctrl.items array using the custom nameFilter filter.
- A custom nameFilter filter that filters items based on the name property and a search text.
- A DataService that provides items (simulated data for this example) to the controller.

HTML (index.html):

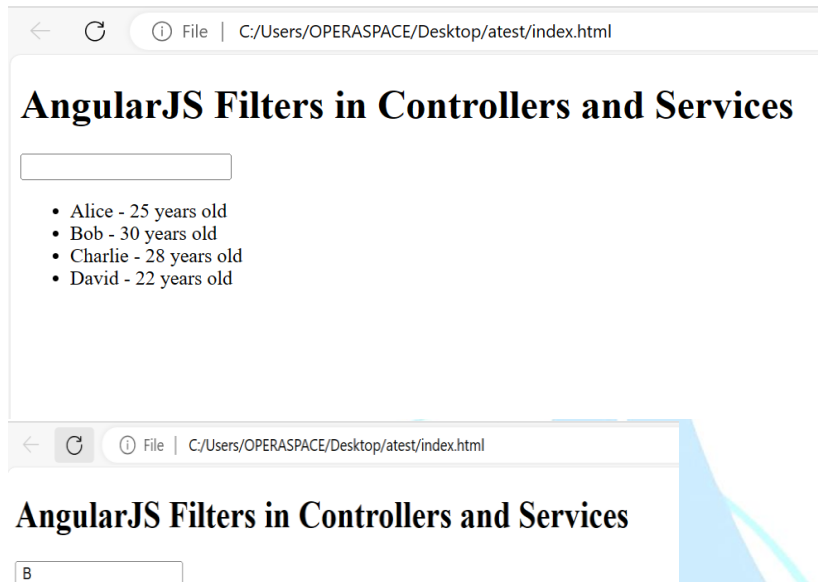
```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS Filters in Controllers and Services</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>

<div ng-controller="MyController as ctrl">
  <h1>AngularJS Filters in Controllers and Services</h1>
  <input type="text" ng-model="ctrl.search" placeholder="Search by name">

  <ul>
    <li ng-repeat="item in ctrl.filteredItems = (ctrl.items | nameFilter:ctrl.search)">
      {{ item.name }} - {{ item.age }} years old
    </li>
  </ul>
</div>

<script>
angular.module('myApp', [])
// Custom Filter
.filter('nameFilter', function() {
  return function(items, searchText) {
    var filtered = [];
    angular.forEach(items, function(item) {
      if (item.name.toLowerCase().includes(searchText.toLowerCase())) {
        filtered.push(item);
      }
    });
    return filtered;
  };
})
// Custom AngularJS Service
.service('DataService', function() {
  this.getItems = function() {
    // Simulated data for demonstration
    return [
      { name: 'Alice', age: 25 },
      { name: 'Bob', age: 30 },
      { name: 'Charlie', age: 28 },
      { name: 'David', age: 22 }
    ];
  };
})
.controller('MyController', ['DataService', function(DataService) {
```

```
var vm = this;  
vm.items = [];  
  
// Using the custom service within the controller  
vm.items = DataService.getItems();  
});  
</script>  
  
</body>  
</html>
```



Module-4

5.Create AngularJS application with a controller (MyController) that uses a custom service (DataService) to fetch data from an API endpoint using Angular's internal \$http service.

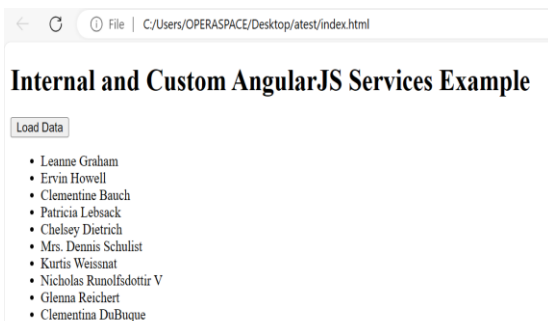
The HTML contains a button that triggers the loadData() function defined in the controller.

The controller uses the DataService to make an HTTP GET request to a sample API (<https://jsonplaceholder.typicode.com/users>) and displays the retrieved data in a list.

HTML (index.html):

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS Services</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>
<div ng-controller="MyController as ctrl">
  <h1>Internal and Custom AngularJS Services Example</h1>

  <button ng-click="ctrl.loadData()">Load Data</button>
  <ul>
    <li ng-repeat="item in ctrl.items">{{ item.name }}</li>
  </ul>
</div>
<script>
angular.module('myApp', [])
  // Custom AngularJS Service
  .service('DataService', ['$http', function($http) {
    this.fetchData = function() {
      return $http.get('https://jsonplaceholder.typicode.com/users'); // Example API endpoint
    };
  }])
  .controller('MyController', ['DataService', function(DataService) {
    var vm = this;
    vm.items = [];
    // Using the custom service within the controller
    vm.loadData = function() {
      DataService.fetchData()
        .then(function(response) {
          vm.items = response.data;
        })
        .catch(function(error) {
          console.error('Error fetching data:', error);
        });
    };
  }]);
</script>
</body>
</html>
```



6 . Create angularjs HTTP Services program for Building Database, Front End and BackEnd

index.html:

This file contains the HTML structure and references to AngularJS and your application script.

app.js:

This file contains the AngularJS application logic, including the main controller.

services/backendService.js:

This file contains the service that simulates backend functionality by providing data to the controller.

Frontend (AngularJS):

HTML:

index.html

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS HTTP Services</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="MainController as main">
  <h1>Books</h1>
  <ul>
    <li ng-repeat="book in main.books">
      {{ book.title }} by {{ book.author }}
    </li>
  </ul>
</body>
</html>
```

JavaScript:

app.js

```
angular.module('myApp', [])
.controller('MainController', ['$http', 'BackendService', function($http, BackendService) {
  var main = this;
  main.books = [];
  // Fetch data from the simulated backend service
  BackendService.getBooks()
  .then(function(response) {
    main.books = response;
  })
  .catch(function(error) {
    console.error('Error fetching data:', error);
  });
}]);
```

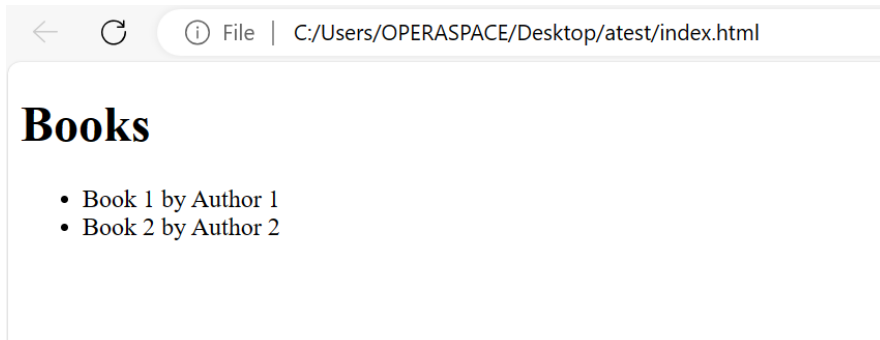
services/backendService.js:

```
angular.module('myApp')
.service('BackendService', ['$q', '$timeout', function($q, $timeout) {
  this.getBooks = function() {
    var deferred = $q.defer();
    // Simulating HTTP request delay
    $timeout(function() {
      var books = [
        { title: 'Book 1', author: 'Author 1' },
        { title: 'Book 2', author: 'Author 2' },
      ];
      deferred.resolve(books);
    }, 1000);
    return deferred.promise;
  };
}]);
```



```
// Add more inbuilt data as needed
];
deferred.resolve(books);
}, 1000);

return deferred.promise;
};
})();
```



Module-5

7. Create program Building a Webserver with Node displaying message "Hello, this is your web server!"

Step 1: Set Up Your Project

First, create a new directory for your project and navigate into it using the terminal:

```
bash Copy code

mkdir my-web-server
cd my-web-server
```

Step 2: Initialize npm

Initialize npm in your project directory to manage dependencies:

```
bash Copy code

npm init -y
```

Step 3: Install Required Packages

Install the http module, which is a part of Node.js, and express, a popular web framework for Node.js:

```
bash Copy code

npm install express
```

Step 4: Create the Server File

Create a file named server.js in your project directory.

```
// Import required modules
const express = require('express');

// Create an Express application
const app = express();

// Define a route to handle GET requests to the root URL
app.get('/', (req, res) => {
  res.send('Hello, this is your web server!');
});

// Start the server on port 3000
const port = 3000;
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

Step 5: Start the Server

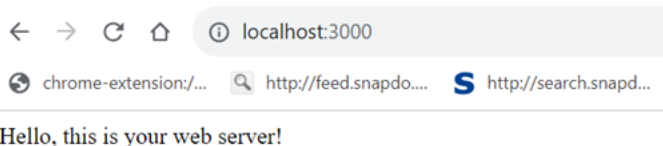
Run your server using Node.js:

```
bash Copy code

node server.js
```

Step 6: Access the Web Server

Open a web browser and visit <http://localhost:3000> to see your server's response.



8.Program to implement Nodejs simple routing to serve different pages with EJS templates and a layout structure

Step 1: Set Up Your Project

Create a new directory for your project and initialize npm:

```
bash Copy code  
  
mkdir node-http-views-layouts  
cd node-http-views-layouts  
npm init -y
```

Step 2: Install Required Packages

Install the http module (built-in) and ejs (templating engine):

```
bash Copy code  
  
npm install ejs
```

Step 3: Create the Server File

Create a file named server.js in your project directory.


```
const http = require('http');  
const fs = require('fs');  
const ejs = require('ejs');  
// Read the layout template  
const layoutTemplate = fs.readFileSync('./views/layout.ejs', 'utf-8');  
  
// Render function to inject content into layout  
function renderContent(content, title = 'Node.js HTTP Server') {  
  return ejs.render(layoutTemplate, { content, title });  
}  
  
// Create an HTTP server  
const server = http.createServer((req, res) => {  
  if (req.url === '/' || req.url === '/home') {  
    // Read the home page content  
    const homePageContent = fs.readFileSync('./views/home.ejs', 'utf-8');  
    const renderedContent = renderContent(homePageContent, 'Home Page');  
    res.writeHead(200, { 'Content-Type': 'text/html' });  
    res.end(renderedContent);  
  } else if (req.url === '/about') {  
    // Read the about page content  
    const aboutPageContent = fs.readFileSync('./views/about.ejs', 'utf-8');  
    const renderedContent = renderContent(aboutPageContent, 'About Page');  
    res.writeHead(200, { 'Content-Type': 'text/html' });  
    res.end(renderedContent);  
  } else {  
    // Handle 404 - Page Not Found  
    res.writeHead(404, { 'Content-Type': 'text/plain' });  
    res.end('404 - Page Not Found');  
  }  
});  
// Set the server to listen on port 3000  
const port = 3000;  
server.listen(port, () => {  
  console.log(`Server is running on http://localhost:${port}`);  
});
```

Step 4: Create Views (EJS Templates)

Create a views directory in your project folder and create home.ejs, about.ejs, and layout.ejs files inside it.

views/home.ejs

html

 Copy code

```
<h1>Welcome to the Home Page</h1>
<p>This is the home page content.</p>
```

views/about.ejs

html

 Copy code

```
<h1>About Us</h1>
<p>This is the about page content.</p>
```


views/layout.ejs

```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
</head>
<body>
  <header>
    <nav>
      <a href="/">Home</a>
      <a href="/about">About</a>
    </nav>
  </header>
  <main>
    <%= content %>
  </main>
  <footer>
    <p>&copy; 2023 Node HTTP Server</p>
  </footer>
</body>
</html>
```

Step 5: Run the Server

Execute the server by running:

bash

 Copy code

```
node server.js
```

Step 6: Access the Web Pages

Open a web browser and visit <http://localhost:3000> to see the home page and <http://localhost:3000/about> for the about page.

