SAVITRIBAI PHULE PUNE UNIVERSITY

M.Sc. (Computer Science) Sem-III

Practical Examination (From 2024-2025)

SUBJECT: CS-611-MJP:

Lab Course on CS-610-MJ (Full Stack Development- II

Practical Slips Programs : Full Stack Developement- II

Slip 1 :

Q.1) Write an AngularJS script for addition of two numbers using ng-init, ng-model & ng-bind. And also demonstrate ng-show, ng-disabled, ng-click directives on button component.

```
<!DOCTYPE html>

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

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

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

</head>

<body ng-controller="myCtrl">


    <!-- Step 1: Initialize numbers using ng-init -->

    <div ng-init="num1 = 0; num2 = 0; sum = 0;">


        <!-- Step 2: Input fields for numbers -->

        <label for="num1">Enter first number:</label>

        <input type="number" id="num1" ng-model="num1">

        <br><br>


        <label for="num2">Enter second number:</label>

        <input type="number" id="num2" ng-model="num2">

        <br><br>


        <!-- Step 3: Show sum using ng-bind -->

        <h3>The sum is: <span ng-bind="sum"></span></h3>


        <!-- Step 4: Button to calculate sum -->

        <button ng-click="calculateSum()" ng-disabled="num1 == 0 || num2 ==
0">Add</button>
```

```
        <!-- Step 5: Display a message when sum is greater than 0 using ng-show -->
        <div ng-show="sum > 0">
            <p>Calculation completed successfully!</p>
        </div>

    </div>

    <script>
        // AngularJS Application and Controller
        var app = angular.module('myApp', []);

        app.controller('myCtrl', function($scope) {
            // Function to calculate the sum of two numbers
            $scope.calculateSum = function() {
                $scope.sum = parseFloat($scope.num1) + parseFloat($scope.num2);
            };
        });
    </script>

</body>
</html>
```

Q.2) Create a Node.js application that reads data from multiple files asynchronously using promises and async/await

```
const fs = require('fs').promises;  // Using fs.promises API for file operations


// Function to read a file asynchronously using Promises

function readFile(fileName) {

    return fs.readFile(fileName, 'utf8')

        .then(data => {

            console.log(`Successfully read ${fileName}:`);

            return data;  // Return file content

        })

        .catch(err => {

            console.error(`Error reading file ${fileName}:`, err);

            throw err;  // Propagate error

        });

}


// Function to read multiple files asynchronously using async/await

async function readFiles() {
    try {
        // Wait for all file readings to complete
        const data1 = await readFile('file1.txt');
        const data2 = await readFile('file2.txt');
        const data3 = await readFile('file3.txt');

        // Log the file contents
        console.log(`File 1 content: ${data1}`);
        console.log(`File 2 content: ${data2}`);
        console.log(`File 3 content: ${data3}`);

    } catch (err) {
        console.error('Error during file read operations:', err);
    }
}

// Start reading the files
readFiles();
```

Slip 2 :

Q.1) Write an AngularJS script to print details of bank (bank name, MICR code, IFC code, address etc.) in tabular form using ng-repeat

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Bank Details</title>

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

    <style>

        table {

            width: 100%;

            border-collapse: collapse;

            margin: 20px 0;

        }

        th, td {

            padding: 8px 12px;

            text-align: left;

            border: 1px solid #ddd;

        }

        th {

            background-color: #f4f4f4;

        }
    </style>

</head>

<body ng-app="bankApp" ng-controller="bankController">


    <h2>Bank Details</h2>


    <table>

        <thead>

            <tr>

                <th>Bank Name</th>

                <th>MICR Code</th>

                <th>IFC Code</th>

                <th>Address</th>

            </tr>

        </thead>

        <tbody>

            <tr ng-repeat="bank in banks">

                <td>{{ bank.name }}</td>

                <td>{{ bank.micrCode }}</td>

                <td>{{ bank.ifcCode }}</td>

                <td>{{ bank.address }}</td>

            </tr>
        </tbody>
    </table>

    <script>
        // AngularJS application module
        var app = angular.module('bankApp', []);

        // Controller for the bank details
        app.controller('bankController', function($scope) {
            $scope.banks = [
                {
                    name: 'State Bank of India',
                    micrCode: '123456789',
                    ifcCode: 'SBI123456',
                    address: 'Mumbai, Maharashtra'
                },
                {
                    name: 'HDFC Bank',
                    micrCode: '987654321',
                    ifcCode: 'HDFC987654',
                    address: 'New Delhi, Delhi'
                },
                {
                    name: 'ICICI Bank',
                    micrCode: '112233445',
                    ifcCode: 'ICICI112233',
                    address: 'Chennai, Tamil Nadu'
                },
                {
                    name: 'Axis Bank',
                    micrCode: '556677889',
                    ifcCode: 'AXIS556677',
                    address: 'Bangalore, Karnataka'
                }
            ];
        });
    </script>

</body>
</html>
```

Q.2) Create a simple Angular application that fetches data from an API using HttpClient. Implement an Observable to fetch data from an API endpoint.

Steps:

1. **Set up Angular Project**: You need to create an Angular project if you don't have one.
2. **Install HttpClientModule**: Make sure the `HttpClientModule` is imported in your application module.
3. **Create Service**: Create a service that uses `HttpClient` to fetch data from an API.
4. **Use Observable**: Implement an Observable to handle asynchronous data fetching.
5. **Bind Data to Component**: Use Angular's data binding to display the data in the component.

Step-by-step Guide

Step 1: Set up a new Angular project (if you haven't already)

First, make sure you have Angular CLI installed. If not, install it using the following command:

```
bash
Copy code
npm install -g @angular/cli
```

Then, create a new Angular project:

```
bash
Copy code
ng new fetch-data-app
cd fetch-data-app
```

Step 2: Install HttpClientModule

In your Angular project, you need to import `HttpClientModule` in your main application module.

Open `src/app/app.module.ts` and update the code as follows:

```
typescript
Copy code
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';  // Import
HttpClientModule
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule  // Add HttpClientModule to the imports array
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step 3: Create a Service to Fetch Data

Next, generate a service that will handle the API call. Open the terminal and use Angular CLI to generate a service:

```
bash
Copy code
ng generate service data
```

This will create a file `data.service.ts`. In that service, we'll use `HttpClient` to fetch data from an API endpoint. Here's how you can set it up:

```
typescript
Copy code
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';  // Import Observable
import { catchError } from 'rxjs/operators';  // To handle errors

@Injectable({
  providedIn: 'root'
})
export class DataService {
  private apiUrl = 'https://jsonplaceholder.typicode.com/posts';  // Example
API endpoint

  constructor(private http: HttpClient) { }

  // Method to get data from the API
  getData(): Observable<any> {
    return this.http.get<any>(this.apiUrl).pipe(
      catchError(error => {
        console.error('Error occurred:', error);
        throw error;
      })
    );
  }
}
```

Step 4: Use the Service in the Component

Now, let’s use the `DataService` to fetch the data in the component and display it.

Open `src/app/app.component.ts` and update it as follows:

```
typescript
Copy code
import { Component, OnInit } from '@angular/core';
import { DataService } from './data.service';  // Import DataService

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  data: any = [];  // Variable to hold the fetched data

  constructor(private dataService: DataService) { }

  // ngOnInit lifecycle hook to call the API when the component loads
  ngOnInit(): void {
    this.dataService.getData().subscribe(
      (response) => {
        this.data = response;  // Store the response in the 'data' variable
      },
      (error) => {
        console.error('Error:', error);  // Handle errors if any
      }
    );
  }
}
```

Step 5: Bind Data in the Template

Now, bind the `data` in the component template (`src/app/app.component.html`) to display it. You can display the fetched data in a list or table format.

```
html
Copy code
<div style="text-align:center; padding: 20px;">
  <h1>Fetched Data from API</h1>

  <div *ngIf="data.length > 0; else noData">
    <ul>
      <li *ngFor="let item of data">
        <h4>{{ item.title }}</h4>
        <p>{{ item.body }}</p>
      </li>
    </ul>
  </div>

  <ng-template #noData>
    <p>No data available.</p>
  </ng-template>
```

```
</div>
```

Step 6: Run the Application

Now that everything is set up, run the Angular application using the following command:

```
bash
Copy code
ng serve
```

Visit `http://localhost:4200` in your browser to see the results. The application will display the data fetched from the API.

Explanation of the Code:

- **HttpClient**: Used for making HTTP requests in Angular. We use the `get` method to send a GET request to the API.
- **Observable**: Angular's `HttpClient.get()` method returns an Observable, which allows you to handle asynchronous data fetching.
- **Subscribe**: We use the `subscribe()` method to get the response once the data is fetched.
- **Error Handling**: We use `catchError` from `rxjs` to handle any errors that occur during the HTTP request.
- ***ngIf and ngFor*: We use `*ngIf` to conditionally render the data and `*ngFor` to loop through the array and display each item.

Sample Output:

The application will display the title and body of posts from the `jsonplaceholder` API in a list. The data will look like:

```
sql
Copy code
Fetched Data from API
Post 1: Title and body of the first post.
Post 2: Title and body of the second post.
...
```

Slip 3 :

Q.1) Write an AngularJS script to display list of games stored in an array on click of button using ng-click and also demonstrate ng-init, ng-bind directive of AngularJS.

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>AngularJS Game List</title>

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

</head>

<body ng-app="gameApp" ng-init="games = ['Soccer', 'Basketball', 'Tennis', 'Cricket',
'Football']">



  <div ng-controller="gameController">



    <!-- Displaying the List of Games using ng-bind -->

    <h3 ng-bind="'Games List'"></h3>



    <!-- Button to trigger the display of games -->

    <button ng-click="showGames = !showGames">

      Click to Display Games
    </button>

    <!-- Display the games list when the button is clicked -->
    <ul ng-show="showGames">
      <li ng-repeat="game in games">{{ game }}</li>
    </ul>

  </div>

  <script>
    // Define AngularJS Application and Controller
    var app = angular.module('gameApp', []);
    app.controller('gameController', function($scope) {
      // ng-init is used in the HTML itself to initialize the games array
      // ng-click logic is used to toggle the showGames boolean value
      $scope.showGames = false; // This will hide the list initially
    });
  </script>

</body>
</html>
```

Q.2) Find a company with a workforce greater than 30 in the array (use find by id method)

```
// Array of companies with id and workforce properties
const companies = [
   { id: 1, name: 'Company A', workforce: 25 },
   { id: 2, name: 'Company B', workforce: 50 },
   { id: 3, name: 'Company C', workforce: 100 },
   { id: 4, name: 'Company D', workforce: 10 },
];

// Function to find the company by id and workforce greater than 30
function findCompanyByIdAndWorkforce(id) {
   // Find the company with a specific id and workforce greater than 30
   const company = companies.find(company => company.id === id &&
company.workforce > 30);
   return company;
}

// Example: Find company with id 2 and workforce greater than 30
const company = findCompanyByIdAndWorkforce(2);

// Display the result
if (company) {
   console.log(`Found company: ${company.name} with workforce:
${company.workforce}`);
} else {
   console.log('No company found with the given conditions');
}
```

Slip 4 :

Q.1) Fetch the details using ng-repeat in AngularJS [15]

```
<!DOCTYPE html>

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

```
<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>AngularJS ng-repeat Example</title>

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

</head>

<body>



    <div ng-controller="BankController">

        <h2>Bank Details</h2>



        <!-- Table to display bank details using ng-repeat -->

        <table border="1">

            <thead>

                <tr>

                    <th>Bank Name</th>

                    <th>MICR Code</th>

                    <th>IFSC Code</th>

                    <th>Address</th>

                </tr>

            </thead>
            <tbody>

                <!-- Using ng-repeat to iterate over the 'banks' array -->

                <tr ng-repeat="bank in banks">

                    <td>{{bank.name}}</td>

                    <td>{{bank.micrCode}}</td>

                    <td>{{bank.ifscCode}}</td>

                    <td>{{bank.address}}</td>

                </tr>

            </tbody>

        </table>

    </div>



    <script>

        // Define AngularJS module and controller

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



        app.controller('BankController', function($scope) {

            // Data for banks

            $scope.banks = [

                { name: 'ABC Bank', micrCode: '123456789', ifscCode: 'ABC123', address:
'123 Main St, City A' },

                { name: 'XYZ Bank', micrCode: '987654321', ifscCode: 'XYZ987', address:
'456 Second St, City B' },
                { name: 'LMN Bank', micrCode: '112233445', ifscCode: 'LMN456', address: 
'789 Third St, City C' },

                { name: 'PQR Bank', micrCode: '998877665', ifscCode: 'PQR321', address: 
'101 First St, City D' }

            ];

        });

    </script>



</body>

</html>
```

Q.2) Express.js application to include middleware for parsing request bodies (e.g., JSON, form data) and validating input data.

```
const express = require('express');
const { body, validationResult } = require('express-validator');
const bodyParser = require('body-parser');

const app = express();
const port = 3000;

// Middleware to parse JSON and form data
app.use(bodyParser.json());  // For JSON payloads
app.use(bodyParser.urlencoded({ extended: true }));  // For form data (x-www-form-
urlencoded)

// Route with input validation
app.post('/submit', [
    // Validate and sanitize input data using express-validator
```

```
   body('name').isString().withMessage('Name must be a
string').notEmpty().withMessage('Name is required'),
   body('email').isEmail().withMessage('Valid email is required').normalizeEmail(),
   body('age').isInt({ min: 18 }).withMessage('Age must be a number and at least 18'),
], (req, res) => {
   // Check for validation errors
   const errors = validationResult(req);
   if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
   }

   // If validation is successful, process the request data
   const { name, email, age } = req.body;
   res.status(200).json({
      message: 'Data received successfully',
      data: {
         name,
         email,
         age
      }
   });
});

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

Slip 5 :

Q.1) Create a simple Angular component that takes input data and displays it. [15]

```
import { Component } from '@angular/core';



@Component({
  selector: 'app-input-display',
  templateUrl: './input-display.component.html',
  styleUrls: ['./input-display.component.css']
})
export class InputDisplayComponent {
  // Define a property to hold the input data
  inputData: string = '';

  // Method to update the inputData (optional)
  updateData(value: string): void {
    this.inputData = value;
  }
}
```

Q.2) Implement a simple server using Node.js. [15]

```
// Import the http module to create an HTTP server
const http = require('http');

// Set the port for the server to listen on
const port = 3000;
```

```
// Create the server

const server = http.createServer((req, res) => {

  // Set the response HTTP header to specify content type

  res.writeHead(200, {'Content-Type': 'text/plain'});



  // Send a response to the client

  res.end('Hello, this is a simple Node.js server!');

});



// Make the server listen on the specified port

server.listen(port, () => {

  console.log(`Server is running at http://localhost:${port}`);

});
```

Slip 6 :

Q.1) Develop an Express.js application that defines routes for Create and Read operations on a resource (products).

```
// Importing required modules
```

```
const express = require('express');

const app = express();

const port = 3000;


// Middleware to parse JSON data from the body of requests

app.use(express.json());


// In-memory product data (as a substitute for a database)

let products = [];


// Route to Create a new product (POST)

app.post('/products', (req, res) => {

  const { name, price } = req.body;


  // Simple validation

  if (!name || !price) {

    return res.status(400).json({ message: 'Name and price are required.' });

  }


  // Creating a new product object

  const newProduct = {

    id: products.length + 1, // simple id generation
    name,
    price
  };

  // Adding the new product to the in-memory database
  products.push(newProduct);

  // Sending back a response with the new product
  res.status(201).json({ message: 'Product created successfully', product: newProduct
});
});

// Route to Read all products (GET)
app.get('/products', (req, res) => {
  res.status(200).json(products);
});

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

Q.2) Find a company with a workforce greater than 30 in the array. (Using find by id method)

```
// Sample data: Array of company objects

const companies = [

  { id: 1, name: "Company A", workforce: 25 },

  { id: 2, name: "Company B", workforce: 50 },

  { id: 3, name: "Company C", workforce: 20 },

  { id: 4, name: "Company D", workforce: 35 },

];


// Use the find method to find the first company with a workforce greater than 30

const companyWithLargeWorkforce = companies.find(company =>
company.workforce > 30);


if (companyWithLargeWorkforce) {

  console.log(`Found a company with a workforce greater than 30:`);

  console.log(`Company ID: ${companyWithLargeWorkforce.id}`);

  console.log(`Company Name: ${companyWithLargeWorkforce.name}`);

  console.log(`Workforce: ${companyWithLargeWorkforce.workforce}`);

} else {

  console.log("No company found with a workforce greater than 30.");

}
```

Slip 7 :

Q.1) Create a Node.js application that reads data from multiple files asynchronously using promises and async/await

```
const fs = require('fs').promises;  // Using fs.promises for promise-based file operations

// Function to read a file asynchronously
const readFile = async (fileName) => {
  try {
    const data = await fs.readFile(fileName, 'utf8');  // Read the file as a string
    console.log(`Data from ${fileName}:`);
    console.log(data);
  } catch (err) {
    console.error(`Error reading file ${fileName}:`, err);
  }
};

// Function to read multiple files asynchronously using async/await
```

```
const readMultipleFiles = async () => {

  try {

    // Using Promise.all to read all files in parallel

    await Promise.all([

      readFile('file1.txt'),

      readFile('file2.txt'),

      readFile('file3.txt')

    ]);

    console.log('All files read successfully.');

  } catch (err) {

    console.error('Error reading files:', err);

  }

};



// Call the function to read the files

readMultipleFiles();
```

Q.2) Develop an Express.js application that defines routes for Create and Read operations on a resource (User)

```
const express = require('express');
const app = express();
const port = 3000;

// Middleware to parse JSON bodies
app.use(express.json());
```

```
// In-memory data store (this could be replaced by a database)
let users = [];

// Route to Create a new user
app.post('/users', (req, res) => {
  const { name, email } = req.body;

  // Basic validation
  if (!name || !email) {
    return res.status(400).json({ error: 'Name and email are required' });
  }

  // Create a new user
  const newUser = { id: users.length + 1, name, email };
  users.push(newUser);

  // Respond with the created user
  res.status(201).json(newUser);
});

// Route to Read (get all users)
app.get('/users', (req, res) => {
  res.status(200).json(users);
});

// Route to Read (get a user by ID)
app.get('/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);
  const user = users.find(u => u.id === userId);

  if (!user) {
    return res.status(404).json({ error: 'User not found' });
  }

  res.status(200).json(user);
});

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

Slip 8 :

Q.1) Create a simple Angular application that fetches data from an API using HttpClient. Implement an Observable to fetch data from an API endpoint

Steps to Create the Application:

1. **Set up the Angular Application**: First, ensure you have Angular CLI installed. If not, install it globally using the following command:

   ```
   bash
   Copy code
   npm install -g @angular/cli
   ```

   Then, create a new Angular project:

   ```
   bash
   Copy code
   ng new fetch-data-app
   cd fetch-data-app
   ```

2. **Install Angular HTTP Client Module**: The `HttpClient` module is required to make HTTP requests. Angular CLI includes this module, but you need to import it into your app.

   Open `app.module.ts` and add the `HttpClientModule` to the imports:

   ```
   typescript
   Copy code
   import { BrowserModule } from '@angular/platform-browser';
   import { NgModule } from '@angular/core';
   import { HttpClientModule } from '@angular/common/http'; // Import
   HttpClientModule

   import { AppComponent } from './app.component';

   @NgModule({
     declarations: [
       AppComponent
   ```

```
  ],
  imports: [
    BrowserModule,
    HttpClientModule  // Add HttpClientModule here
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3. **Create a Service to Fetch Data**: Create a service to handle the HTTP requests. You can generate a service using Angular CLI.

```
bash
Copy code
ng generate service data
```

In the `data.service.ts` file, write the code to fetch data from an API. For this example, we'll use a dummy JSON API (e.g., `https://jsonplaceholder.typicode.com/posts`).

```
typescript
Copy code
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class DataService {

  private apiUrl = 'https://jsonplaceholder.typicode.com/posts'; //
Replace with your API URL

  constructor(private http: HttpClient) { }

  // Method to fetch data using HTTP GET request
  getPosts(): Observable<any> {
    return this.http.get<any>(this.apiUrl);
  }
}
```

In the `getPosts()` method, we're using `http.get()` to make an HTTP GET request. This method returns an `Observable`, which is a common way to handle asynchronous operations in Angular.

4. **Use the Service in the Component**: Open `app.component.ts` and modify it to call the `DataService` and display the data.

```
typescript
```

```
Copy code
import { Component, OnInit } from '@angular/core';
import { DataService } from './data.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  posts: any[] = []; // Array to store fetched posts

  constructor(private dataService: DataService) {}

  ngOnInit(): void {
    // Fetch the data when the component initializes
    this.dataService.getPosts().subscribe(
      (data) => {
        this.posts = data; // Store the fetched data in posts
      },
      (error) => {
        console.error('Error fetching data', error);
      }
    );
  }
}
```

Here, in the `ngOnInit()` lifecycle hook, we're calling the `getPosts()` method from the `DataService` and subscribing to the returned `Observable`. When the data is successfully fetched, it is stored in the `posts` array.

5. **Display the Data in the Template**: Open `app.component.html` and use `ngFor` to loop through the `posts` array and display the data.

```
html
Copy code
<div class="container">
  <h1>Posts</h1>

  <div *ngIf="posts.length === 0">
    <p>Loading...</p>
  </div>

  <div *ngFor="let post of posts">
    <h3>{{ post.title }}</h3>
    <p>{{ post.body }}</p>
  </div>
</div>
```

The `*ngFor` directive is used to loop over each post and display its title and body. The `*ngIf` directive checks if the `posts` array is empty (meaning the data is still loading), and shows a "Loading..." message until the data is available.

6. **Run the Application**: After completing the above steps, start the Angular development server:

   ```
   bash
   Copy code
   ng serve
   ```

   Open `http://localhost:4200/` in your browser. You should see a list of posts fetched from the API.

Full Code Summary:

app.module.ts:

```
typescript
Copy code
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

data.service.ts:

```
typescript
Copy code
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class DataService {
  private apiUrl = 'https://jsonplaceholder.typicode.com/posts';

  constructor(private http: HttpClient) { }

  getPosts(): Observable<any> {
    return this.http.get<any>(this.apiUrl);
```

```
  }
}
```

app.component.ts:

```
typescript
Copy code
import { Component, OnInit } from '@angular/core';
import { DataService } from './data.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  posts: any[] = [];

  constructor(private dataService: DataService) {}

  ngOnInit(): void {
    this.dataService.getPosts().subscribe(
      (data) => {
        this.posts = data;
      },
      (error) => {
        console.error('Error fetching data', error);
      }
    );
  }
}
```

app.component.html:

```
html
Copy code
<div class="container">
  <h1>Posts</h1>

  <div *ngIf="posts.length === 0">
    <p>Loading...</p>
  </div>

  <div *ngFor="let post of posts">
    <h3>{{ post.title }}</h3>
    <p>{{ post.body }}</p>
  </div>
</div>
```

Q.2) Develop an Express.js application that defines routes for Create, Update operations
on a resource (Employee)

```
const express = require('express');
const bodyParser = require('body-parser');

// Initialize the Express app
const app = express();
const port = 3000;

// Middleware to parse JSON data
app.use(bodyParser.json());

// In-memory employee data (this will act as our database for this example)
let employees = [
  { id: 1, name: 'John Doe', position: 'Developer', salary: 50000 },
  { id: 2, name: 'Jane Smith', position: 'Manager', salary: 60000 }
];

// Route to get all employees
app.get('/employees', (req, res) => {
  res.json(employees);
});

// Route to create a new employee
app.post('/employees', (req, res) => {
  const newEmployee = req.body;

  // Validate input data
  if (!newEmployee.name || !newEmployee.position || !newEmployee.salary) {
    return res.status(400).json({ message: 'All fields are required' });
  }

  // Generate a new ID
  const newId = employees.length ? employees[employees.length - 1].id + 1 :
1;
  newEmployee.id = newId;

  // Add the new employee to the list
  employees.push(newEmployee);

  res.status(201).json({ message: 'Employee created', employee: newEmployee
});
});

// Route to update an existing employee
app.put('/employees/:id', (req, res) => {
  const employeeId = parseInt(req.params.id);
  const updatedEmployee = req.body;

  // Find the employee by ID
  const employee = employees.find(emp => emp.id === employeeId);

  if (!employee) {
    return res.status(404).json({ message: 'Employee not found' });
  }

  // Update the employee details
  employee.name = updatedEmployee.name || employee.name;
  employee.position = updatedEmployee.position || employee.position;
  employee.salary = updatedEmployee.salary || employee.salary;
  res.json({ message: 'Employee updated', employee });
});

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

Slip 9

:

Q.1) Find a company with a workforce greater than 30 in the array. (Using find by id method)

```
// Sample array of companies with id and workforce properties

const companies = [

    { id: 1, name: "Company A", workforce: 25 },

    { id: 2, name: "Company B", workforce: 40 },

    { id: 3, name: "Company C", workforce: 50 },

    { id: 4, name: "Company D", workforce: 15 }

];



// Function to find company with workforce greater than 30

function findCompanyById(workforceThreshold) {

    // Using the find() method to search for a company by workforce

    const company = companies.find(company => company.workforce > 
workforceThreshold);
```

```
    return company;

}



// Find a company with workforce greater than 30

const companyWithLargeWorkforce = findCompanyById(30);



// Check the result

if (companyWithLargeWorkforce) {

    console.log(`Company with workforce greater than 30:`);

    console.log(`ID: ${companyWithLargeWorkforce.id}`);

    console.log(`Name: ${companyWithLargeWorkforce.name}`);

    console.log(`Workforce: ${companyWithLargeWorkforce.workforce}`);

} else {

    console.log('No company found with workforce greater than 30.');

}
```

Q.2) Create Express.js application to include middleware for parsing request bodies (e.g., JSON, form data) and validating input data. Send appropriate JSON responses for success and error cases.

```
const express = require('express');
const app = express();
const port = 3000;
```

```
// Middleware to parse incoming request bodies
app.use(express.json()); // For parsing application/json
app.use(express.urlencoded({ extended: true })); // For parsing application/x-www-
form-urlencoded

// Sample middleware to validate input data
const validateInput = (req, res, next) => {
    const { name, age } = req.body;

    if (!name || !age) {
        return res.status(400).json({
            success: false,
            message: "Name and age are required!"
        });
    }

    if (isNaN(age) || age < 18) {
        return res.status(400).json({
            success: false,
            message: "Age must be a number and at least 18!"
        });
    }

    // If validation passes, move to the next middleware or route handler
    next();
};

// POST route to create a new user (with validation)
app.post('/create-user', validateInput, (req, res) => {
    const { name, age } = req.body;

    // Simulate saving the user data
    const user = {
        id: Math.floor(Math.random() * 1000),
        name,
        age
    };

    res.status(201).json({
        success: true,
        message: 'User created successfully',
        user
    });
});

// GET route to fetch user details (for demonstration)
app.get('/user/:id', (req, res) => {
    const userId = req.params.id;

    // Simulate fetching a user (you could use a real database here)
    const user = {
        id: userId,
        name: "John Doe",
        age: 25
    };

    res.json({
        success: true,
        user
    });
});

// Default route for handling unknown requests
app.use((req, res) => {
    res.status(404).json({
        success: false,
        message: "Route not found"
    });
});

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

Slip 10 :

Q.1) Implement a simple server using Node.js. [15]

```
// Importing the http module
const http = require('http');

// Define the hostname and port for the server
const hostname = '127.0.0.1';
const port = 3000;

// Create an HTTP server
const server = http.createServer((req, res) => {
  // Set the response header with a status code and content type
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html');

  // Handle different routes
  if (req.url === '/') {
    res.end('<h1>Welcome to the Home Page!</h1>');
  } else if (req.url === '/about') {
    res.end('<h1>About Us</h1><p>This is a simple Node.js server.</p>');
  } else if (req.url === '/contact') {
    res.end('<h1>Contact Us</h1><p>You can contact us at
example@domain.com.</p>');

  } else {

    res.statusCode = 404;

    res.end('<h1>404 Not Found</h1><p>The page you requested does not
exist.</p>');

  }

});



// Start the server and listen for requests

server.listen(port, hostname, () => {

  console.log(`Server running at http://${hostname}:${port}/`);

});
```

Q.2) Extend the previous Express.js application to include middleware for parsing request bodies (e.g., JSON, form data) and validating input data. Send appropriate JSON responses for success and error cases

```
// Import required libraries

const express = require('express');

const bodyParser = require('body-parser');
const { check, validationResult } = require('express-validator');

// Initialize the Express app
const app = express();

// Middleware to parse JSON and URL-encoded form data
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// Sample route to create a product (POST request)
app.post('/product', [
  // Validation rules
  check('name').isLength({ min: 1 }).withMessage('Name is required'),
  check('price').isFloat({ min: 0 }).withMessage('Price must be a positive number'),
  check('category').isLength({ min: 1 }).withMessage('Category is required'),
], (req, res) => {
  // Validate the incoming request data
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    // If there are validation errors, return a 400 response with errors
    return res.status(400).json({
      success: false,
      message: 'Validation failed',
      errors: errors.array(),
    });
  }

  // If validation passed, process the data (for simplicity, just echo the data)
  const { name, price, category } = req.body;
  return res.status(201).json({
    success: true,
    message: 'Product created successfully',
    data: {
      name,
      price,
      category,
    },
  });
});

// Sample route to get a product (GET request)
app.get('/product/:id', (req, res) => {
  const { id } = req.params;
  if (!id) {
    return res.status(400).json({
      success: false,
      message: 'Product ID is required',
    });
  }

  // Simulate fetching a product (for example, from a database)
  const product = {
    id,
    name: 'Sample Product',
    price: 100,
    category: 'Electronics',
  };

  return res.status(200).json({
    success: true,
    message: 'Product fetched successfully',
    data: product,
  });
});
```

```
// Define a simple error handler for undefined routes
app.use((req, res) => {
  return res.status(404).json({
    success: false,
    message: 'Route not found',
  });
});

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

Slip 11 :

Q.1) Develop an Express.js application that defines routes for Create operations on a resource (Movie)

```
// Import required libraries

const express = require('express');

const bodyParser = require('body-parser');


// Initialize the Express app

const app = express();


// Middleware to parse JSON data in request body

app.use(bodyParser.json());


// In-memory storage for movies (acting as a database)

let movies = [];


// Route to create a movie (POST request)

app.post('/movie', (req, res) => {

  // Extract movie details from the request body

  const { title, director, releaseYear, genre } = req.body;


  // Validate required fields
```

```
if (!title || !director || !releaseYear || !genre) {
  return res.status(400).json({
    success: false,
    message: 'All fields (title, director, releaseYear, genre) are required.'
  });
}

// Create a new movie object
const newMovie = {
  id: movies.length + 1, // Generate a unique ID (just for demonstration)
  title,
  director,
  releaseYear,
  genre
};

// Save the new movie to the "database"
movies.push(newMovie);

// Send success response
return res.status(201).json({
  success: true,
    message: 'Movie created successfully!',
    data: newMovie
  });
});

// Route to get all movies (GET request for testing)
app.get('/movies', (req, res) => {
  return res.status(200).json({
    success: true,
    data: movies
  });
});

// Error handler for undefined routes
app.use((req, res) => {
  return res.status(404).json({
    success: false,
    message: 'Route not found'
  });
});

// Start the server
```

```
const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

  console.log(`Server running at http://localhost:${PORT}`);

});
```

Q.2) Create Angular application that print the name of students who play basketball using filter and map method.

```
import { Component } from '@angular/core';


@Component({

  selector: 'app-student-list',

  templateUrl: './student-list.component.html',

  styleUrls: ['./student-list.component.css']

})

export class StudentListComponent {

  // Array of students with their names and the sports they play

  students = [

    { name: 'Alice', sports: ['Basketball', 'Football'] },

    { name: 'Bob', sports: ['Football', 'Cricket'] },

    { name: 'Charlie', sports: ['Basketball', 'Tennis'] },
    { name: 'David', sports: ['Cricket'] },

    { name: 'Eva', sports: ['Basketball', 'Hockey'] }

  ];



  // Method to get the names of students who play Basketball

  getBasketballPlayers() {

    return this.students

      .filter(student => student.sports.includes('Basketball'))  // Filter students who play
basketball

      .map(student => student.name);  // Map the result to just the student names

  }

}
```

Slip 12 :

Q.1) Write an AngularJS script to print details of Employee (employee name, employee Id,Pin code, address etc.) in tabular form using ng-repeat.

```
<!DOCTYPE html>

<html lang="en">

<head>
```

```
    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Employee Details</title>

    <!-- Include AngularJS from CDN -->

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

    <style>

        table, th, td {

            border: 1px solid black;

            border-collapse: collapse;

            padding: 10px;

        }

        th {

            text-align: left;

        }

    </style>

</head>

<body ng-app="employeeApp" ng-controller="employeeController">



    <h2>Employee Details</h2>



    <!-- Table to display employee details -->
```

```
<table>
    <thead>
        <tr>
            <th>Employee ID</th>
            <th>Employee Name</th>
            <th>Pin Code</th>
            <th>Address</th>
        </tr>
    </thead>
    <tbody>
        <tr ng-repeat="employee in employees">
            <td>{{ employee.id }}</td>
            <td>{{ employee.name }}</td>
            <td>{{ employee.pinCode }}</td>
            <td>{{ employee.address }}</td>
        </tr>
    </tbody>
</table>

<script>
    // AngularJS application
    var app = angular.module('employeeApp', []);
```

```
        // AngularJS Controller

        app.controller('employeeController', function($scope) {

            // Array of employee details

            $scope.employees = [

                {id: 'E001', name: 'Alice', pinCode: '12345', address: '123 Elm Street'},

                {id: 'E002', name: 'Bob', pinCode: '23456', address: '456 Oak Avenue'},

                {id: 'E003', name: 'Charlie', pinCode: '34567', address: '789 Pine Road'},

                {id: 'E004', name: 'David', pinCode: '45678', address: '101 Maple Blvd'},

                {id: 'E005', name: 'Eva', pinCode: '56789', address: '202 Birch Lane'}

            ];

        });

    </script>


</body>

</html>
```

Q.2) Develop an Express.js application that defines routes for Create operations on a resource (User).

```
// app.js

const express = require('express');
const bodyParser = require('body-parser');
const app = express();

// Middleware to parse incoming JSON data
app.use(bodyParser.json());

// In-memory database to store users (For demo purposes)
let users = [];

// POST route to create a new user
app.post('/users', (req, res) => {
   const { name, email, age } = req.body;

   // Check if all required fields are provided
   if (!name || !email || !age) {
      return res.status(400).json({ message: "Name, email, and age are required." });
   }

   // Create a new user object
   const newUser = {
      id: users.length + 1,  // Auto-incremented ID
      name,
      email,
      age
   };

   // Add the new user to the users array
   users.push(newUser);

   // Send response back to the client
   res.status(201).json({
      message: "User created successfully",
      user: newUser
   });
});

// Route to get all users (optional, for testing purposes)
app.get('/users', (req, res) => {
   res.status(200).json(users);
});

// Set the server to listen on port 3000
```

```
app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Slip 13:

Q.1) Extend the previous Express.js application to include middleware for parsing request bodies (e.g., JSON, form data) and validating input data. Send appropriate JSON responses for success and error cases

```
// app.js

const express = require('express');

const bodyParser = require('body-parser');

const app = express();

// Middleware to parse incoming request bodies

app.use(bodyParser.json());  // For JSON data

app.use(bodyParser.urlencoded({ extended: true }));  // For form-data (application/x-www-form-urlencoded)

// In-memory database to store users (For demo purposes)

let users = [];
```

```
// Middleware to validate input data
function validateUserData(req, res, next) {
    const { name, email, age } = req.body;

    // Check if name, email, and age are provided
    if (!name || !email || !age) {
        return res.status(400).json({ message: "Name, email, and age are required." });
    }

    // Validate the format of email (basic check)
    const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
    if (!emailRegex.test(email)) {
        return res.status(400).json({ message: "Invalid email format." });
    }

    // Check if age is a valid number
    if (isNaN(age)) {
        return res.status(400).json({ message: "Age must be a number." });
    }

    // Proceed to the next middleware/route handler if validation passes
    next();
}

// POST route to create a new user
app.post('/users', validateUserData, (req, res) => {
    const { name, email, age } = req.body;

    // Create a new user object
    const newUser = {
        id: users.length + 1,  // Auto-incremented ID
        name,
        email,
        age
    };

    // Add the new user to the users array
    users.push(newUser);

    // Send a success response back to the client
    res.status(201).json({
        message: "User created successfully",
        user: newUser
    });

});


// Route to get all users (optional, for testing purposes)

app.get('/users', (req, res) => {

    res.status(200).json(users);

});


// Set the server to listen on port 3000

app.listen(3000, () => {

    console.log('Server running on port 3000');

});
```

Q.2) Create a simple Angular component that takes input data and displays it. [15]

1. **app.component.ts -** The TypeScript file where we define the component's logic.
2. **app.component.html -** The HTML file for the component's template.

Step 1: Set up the Angular Component

If you haven't already, create a new Angular component using the Angular CLI:

```
bash
Copy code
ng generate component displayInput
```

This command will create the component files needed, including `display-input.component.ts` and `display-input.component.html`.

Step 2: Implement the Component

1. display-input.component.ts (TypeScript Logic)

Define the input property and the function to handle the display of input data.

```
typescript
Copy code
// display-input.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-display-input',
  templateUrl: './display-input.component.html',
  styleUrls: ['./display-input.component.css']
})
export class DisplayInputComponent {
  userInput: string = ''; // Property to store user input
  displayText: string = ''; // Property to display the text

  // Function to update the display text
  updateDisplay() {
    this.displayText = this.userInput;
  }
}
```

2. display-input.component.html (HTML Template)

Add an input field, a button, and an area to display the input text.

```
html
Copy code
<!-- display-input.component.html -->
<div style="text-align: center; margin-top: 20px;">
  <h2>Enter Some Text</h2>
  <input [(ngModel)]="userInput" placeholder="Type something..." />
  <button (click)="updateDisplay()">Display Text</button>

  <h3 *ngIf="displayText">You entered: {{ displayText }}</h3>
</div>
```

Step 3: Update App Module to Use ngModel

To use `[(ngModel)]` in the component, ensure that `FormsModule` is imported in your `app.module.ts`:

```
typescript
Copy code
```

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms'; // Import FormsModule

import { AppComponent } from './app.component';
import { DisplayInputComponent } from './display-input/display-
input.component';

@NgModule({
  declarations: [
    AppComponent,
    DisplayInputComponent
  ],
  imports: [
    BrowserModule,
    FormsModule // Add FormsModule here
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step 4: Add the Component to the App Template

In `app.component.html`, add the `<app-display-input>` selector to display your new component.

```
html
Copy code
<!-- app.component.html -->
<app-display-input></app-display-input>
```

How it Works

1. The user types into the input field, which binds the text to the `userInput` property.
2. When the user clicks the "Display Text" button, it triggers the `updateDisplay()` method, which sets `displayText` to the value of `userInput`.
3. The text entered by the user is displayed below the input field in real-time.

Run the Application

Run the Angular application to see the component in action:

```
bash
Copy code
ng serve
```

Slip 14 :

Q.1) Create Angular application that print the name of students who got 85% using filter and map method.

Step 1: Generate the Component

If you haven't already, create a new Angular component for displaying student data.

```
bash
Copy code
ng generate component student-list
```

Step 2: Define the Component Logic

In `student-list.component.ts`, define an array of students with names and scores, and then use JavaScript's `filter` and `map` methods to display names of students who scored 85% or higher.

student-list.component.ts

```
typescript
Copy code
import { Component } from '@angular/core';

@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  styleUrls: ['./student-list.component.css']
})
export class StudentListComponent {
  // Define an array of students with name and score
  students = [
    { name: 'Alice', score: 90 },
    { name: 'Bob', score: 76 },
    { name: 'Charlie', score: 85 },
    { name: 'David', score: 88 },
    { name: 'Eve', score: 82 }
  ];

  // Use filter and map to get names of students with 85% or higher
  highScorers = this.students
    .filter(student => student.score >= 85)
    .map(student => student.name);
}
```

Step 3: Create the HTML Template

In `student-list.component.html`, display the names of students who scored 85% or higher.

student-list.component.html

```
html
Copy code
<div style="text-align: center; margin-top: 20px;">
  <h2>Students with 85% or Higher</h2>
  <ul>
    <li *ngFor="let studentName of highScorers">{{ studentName }}</li>
  </ul>
</div>
```

Step 4: Add the Component to App Module

Make sure the `StudentListComponent` is declared in the `app.module.ts` file:

```
typescript
Copy code
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { StudentListComponent } from './student-list/student-list.component';

@NgModule({
  declarations: [
    AppComponent,
    StudentListComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step 5: Add the Component to the App Template

In `app.component.html`, include the `<app-student-list>` selector:

```
html
Copy code
<!-- app.component.html -->
<app-student-list></app-student-list>
```

Step 6: Run the Application

Run the application to see the list of students with scores of 85% or higher:

```
bash
Copy code
ng serve
```

Open a browser and navigate to `http://localhost:4200` to see the results.

Explanation

1. **Filter**: Filters the array to include only students with a score of 85 or higher.
2. **Map**: Maps the filtered array to get only the names of those students.

Q.2) Develop an Express.js application that defines routes for Create, Update operations on a resource (Employee)

```
const express = require('express');

const bodyParser = require('body-parser');

const app = express();

const PORT = 3000;



// Middleware to parse JSON request bodies

app.use(bodyParser.json());



// Mock database

let employees = [];
```

```
// Route to create a new employee

app.post('/employees', (req, res) => {

  const { id, name, position, salary } = req.body;



  // Validate required fields

  if (!id || !name || !position || !salary) {

    return res.status(400).json({ error: 'All fields are required (id, name, position,
salary)' });

  }



  // Check if employee with the same ID already exists

  const existingEmployee = employees.find(emp => emp.id === id);

  if (existingEmployee) {

    return res.status(409).json({ error: 'Employee with the same ID already exists' });

  }



  // Add new employee to the database

  const newEmployee = { id, name, position, salary };

  employees.push(newEmployee);

  res.status(201).json({ message: 'Employee created successfully', employee:
newEmployee });

});
```

```
// Route to update an existing employee by ID
app.put('/employees/:id', (req, res) => {
  const employeeId = req.params.id;
  const { name, position, salary } = req.body;

  // Find the employee by ID
  const employee = employees.find(emp => emp.id === employeeId);
  if (!employee) {
    return res.status(404).json({ error: 'Employee not found' });
  }

  // Update employee details
  if (name) employee.name = name;
  if (position) employee.position = position;
  if (salary) employee.salary = salary;

  res.json({ message: 'Employee updated successfully', employee });
});

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

Slip 15 :

Q.1) Find an emp with a Salary greater than 25000 in the array. (Using find by id method)

```
// Sample employee array

const employees = [

    { id: 1, name: 'Alice', position: 'Developer', salary: 20000 },

    { id: 2, name: 'Bob', position: 'Designer', salary: 30000 },

    { id: 3, name: 'Charlie', position: 'Manager', salary: 40000 },

];


// Find an employee with a salary greater than 25,000

const highSalaryEmployee = employees.find(employee => employee.salary > 25000);


if (highSalaryEmployee) {

    console.log('Employee with salary greater than 25000:', highSalaryEmployee);

} else {
```

    console.log('No employee found with salary greater than 25000');

}

Q.2) Create Angular application that print the name of students who got 85% using filter and map method

Step 1: Set up the Angular component

First, create an Angular component named `StudentListComponent`. This component will contain an array of student data and will display only those students who scored 85% or above.

```
bash
Copy code
ng generate component StudentList
```

Step 2: Define Student Data and Filtering Logic

In the `student-list.component.ts` file, define the student data array and use `filter` and `map` to get the names of students who scored 85% or more.

```
typescript
Copy code
// student-list.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  styleUrls: ['./student-list.component.css']
})
export class StudentListComponent {
  // Sample student data
  students = [
    { id: 1, name: 'Alice', score: 90 },
    { id: 2, name: 'Bob', score: 70 },
    { id: 3, name: 'Charlie', score: 85 },
    { id: 4, name: 'David', score: 88 }
  ];

  // Filter and map to get the names of students who scored 85 or more
  topStudents = this.students
```

```
    .filter(student => student.score >= 85)
    .map(student => student.name);
}
```

Step 3: Display the Filtered Data in the Template

In `student-list.component.html`, iterate over the `topStudents` array and display each name.

```
html
Copy code
<!-- student-list.component.html -->
<div>
  <h3>Students who scored 85% or higher:</h3>
  <ul>
    <li *ngFor="let studentName of topStudents">{{ studentName }}</li>
  </ul>
</div>
```

Explanation

- **`filter`**: This method filters out students who have a score less than 85.
- **`map`**: After filtering, `map` extracts only the `name` property of each student who passed the filter criteria.

Step 4: Add the Component to the Main Application Template

Include the `app-student-list` selector in your main `app.component.html` to display the component.

```
html
Copy code
<!-- app.component.html -->
<app-student-list></app-student-list>
```

Expected Output

The application will display the names of students who scored 85% or above:

```
diff
Copy code
Students who scored 85% or higher:
- Alice
- Charlie
```

- David

End