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">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Addition of Two Numbers in AngularJS</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-app="additionApp" ng-controller="AdditionController" ng-init="num1=0; num2=0; result=0; showResult=false">

  <h2>Addition of Two Numbers</h2>

  <!-- Input fields for the numbers with ng-model -->
  <label for="num1">Enter first number:</label>
  <input type="number" id="num1" ng-model="num1" required>
  <br>

  <label for="num2">Enter second number:</label>
  <input type="number" id="num2" ng-model="num2" required>
  <br><br>

  <!-- Button with ng-click to trigger addition, ng-disabled if inputs are empty -->
  <button ng-click="addNumbers()" ng-disabled="!num1 || !num2">Add Numbers</button>

  <!-- Display result only when showResult is true using ng-show -->
```

```
    <h3 ng-show="showResult">Result: <span ng-bind="result"></span></h3>

    <script>
      // Define AngularJS module and controller
      angular.module('additionApp', [])
        .controller('AdditionController', function($scope) {
          // Function to add numbers and display result
          $scope.addNumbers = function() {
            $scope.result = parseFloat($scope.num1) + parseFloat($scope.num2);
            $scope.showResult = true;
          };
        });
    </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;

async function readFiles(filePaths) {
  try {
    // Read multiple files asynchronously using Promise.all
    const fileReadPromises = filePaths.map(path => fs.readFile(path, 'utf-8'));
    const fileContents = await Promise.all(fileReadPromises);

    // Log each file's content
    fileContents.forEach((content, index) => {
      console.log(`Content of file ${filePaths[index]}:\n${content}\n`);
```

```
  });
 } catch (error) {
  console.error("Error reading files:", error);
 }
}

// Example file paths (replace with actual paths)
const files = ['./file1.txt', './file2.txt', './file3.txt'];

// Call the function to read files
readFiles(files);
```

SLIP2

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

//html file

```
<!DOCTYPE html>
<html ng-app="bankApp">
<head>
 <title>Bank Details</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="BankController">

 <h2>Bank Details</h2>
 <table border="1">
  <thead>
```

```
    <tr>
      <th>Bank Name</th>
      <th>MICR Code</th>
      <th>IFSC Code</th>
      <th>Address</th>
    </tr>
  </thead>
  <tbody>
    <!-- ng-repeat to iterate over each bank object -->
    <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>

</body>
</html>
//angularJs file
// Define AngularJS application
var app = angular.module('bankApp', []);

// Define controller
app.controller('BankController', function($scope) {
 // List of banks
 $scope.banks = [
  {
```

```
    name: 'Bank of America',
    micrCode: '123456789',
    ifscCode: 'BOFA12345',
    address: '123 Main St, New York, NY'
  },
  {
    name: 'Wells Fargo',
    micrCode: '987654321',
    ifscCode: 'WF123456',
    address: '456 Elm St, San Francisco, CA'
  },
  {
    name: 'Chase Bank',
    micrCode: '543216789',
    ifscCode: 'CHAS09876',
    address: '789 Maple Ave, Chicago, IL'
  }
 ];
});
```

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

```
// src/app/data.service.ts
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'; // Sample API

  constructor(private http: HttpClient) { }

  // Method to fetch data from the API
  getData(): Observable<any> {
    return this.http.get(this.apiUrl);
  }
}
```

Slip 3

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 ng-app="gameApp">
<head>
    <title>Game List Display</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-controller="GameController" ng-init="isListVisible=false">

    <h1>Game List</h1>

    <!-- Button to show the list of games -->
    <button ng-click="showGames()">Show Games</button>
```

```
  <!-- Display a message when the list is empty -->
  <p ng-if="!games.length">No games to display</p>

  <!-- Display the list of games -->
  <ul ng-if="isListVisible">
    <li ng-repeat="game in games" ng-bind="game"></li>
  </ul>

  <script>
    // Define the AngularJS application module
    angular.module('gameApp', [])
      .controller('GameController', function($scope) {
        // Initialize the list of games
        $scope.games = ['Chess', 'Monopoly', 'Scrabble', 'Risk'];

        // Function to show the list of games
        $scope.showGames = function() {
          $scope.isListVisible = true;
        };
      });
  </script>
</body>
</html>
```

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

```
type Company = {
  id: number;
```

```
  name: string;
  workforce: number;
};

const companies: Company[] = [
  { id: 1, name: "TechCorp", workforce: 25 },
  { id: 2, name: "InnovateInc", workforce: 40 },
  { id: 3, name: "BuildIt", workforce: 15 },
  { id: 4, name: "DevelopHub", workforce: 35 },
];

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

console.log(companyWithLargeWorkforce);
```

SLIP 4

Fetch the details using ng-repeat in AngularJS

```
<div ng-app="myApp" ng-controller="CompanyController">
  <ul>
    <li ng-repeat="company in companies | filter: filterByWorkforce">
      <strong>ID:</strong> {{ company.id }} <br>
```

```
    <strong>Name:</strong> {{ company.name }} <br>
    <strong>Workforce:</strong> {{ company.workforce }}
    <hr>
   </li>
  </ul>
</div>
var app = angular.module('myApp', []);
app.controller('CompanyController', function($scope) {
  $scope.companies = [
    { id: 1, name: "TechCorp", workforce: 25 },
    { id: 2, name: "InnovateInc", workforce: 40 },
    { id: 3, name: "BuildIt", workforce: 15 },
    { id: 4, name: "DevelopHub", workforce: 35 }
  ];

  $scope.filterByWorkforce = function(company) {
    return company.workforce > 30;
  };
});
```

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 app = express();
const PORT = 3000;

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

// Sample route with validation middleware
app.post(
  '/submit',
  // Validation middleware
  [
    body('name').isString().withMessage('Name must be a string').notEmpty().withMessage('Name is required'),
    body('email').isEmail().withMessage('Invalid email address'),
    body('age').isInt({ min: 1 }).withMessage('Age must be a positive integer'),
  ],
  (req, res) => {
    // Check for validation errors
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
```

```
    return res.status(400).json({ errors: errors.array() });
  }

  // Process the request if validation passes
  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 on http://localhost:${PORT}`);
});
```

SLIP5

Create a simple Angular component that takes input data and displays it.

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

@Component({
 selector: 'app-display-item',
 templateUrl: './display-item.component.html',
 styleUrls: ['./display-item.component.css']
```

```
})
export class DisplayItemComponent {
  @Input() item: string = ''; // Input property to accept data from parent
}
<div *ngIf="item">
  <p>{{ item }}</p>
</div>
<app-display-item [item]="'Hello, Angular!'"></app-display-item>
```

Implement a simple server using Node.js

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

// Define the port the server will listen on
const PORT = 3000;

// Create the server
const server = http.createServer((req, res) => {
  // Set the response header content type
  res.writeHead(200, { 'Content-Type': 'text/plain' });

  // Send a response message
```

```
  res.end('Hello, World!\n');
});

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

SLIP 6

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

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

// Middleware to parse incoming JSON requests
app.use(express.json());

// Sample in-memory data to store products
let products = [
  { id: 1, name: 'Product 1', price: 100 },
  { id: 2, name: 'Product 2', price: 150 },
];
```

```
// Route for creating a new product (POST)
app.post('/products', (req, res) => {
  const { name, price } = req.body;

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

  const newProduct = {
    id: products.length + 1,  // Simple auto-increment logic
    name,
    price,
  };

  products.push(newProduct);
  res.status(201).json(newProduct);  // Send the new product as response
});

// Route for getting all products (GET)
```

```
app.get('/products', (req, res) => {
  res.status(200).json(products);  // Return the list of products
});

// Route for getting a single product by id (GET)
app.get('/products/:id', (req, res) => {
  const productId = parseInt(req.params.id);
  const product = products.find(p => p.id === productId);

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

  res.status(200).json(product);  // Return the found product
});

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

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

```
const companies = [
  { id: 1, name: 'Company A', workforce: 25 },
  { id: 2, name: 'Company B', workforce: 35 },
  { id: 3, name: 'Company C', workforce: 50 },
  { id: 4, name: 'Company D', workforce: 20 },
];
const companyIdToSearch = 2; // The id of the company we're looking for

// Use the `find` method to locate the company with the given id and workforce > 30
const company = companies.find(company => company.id === companyIdToSearch &&
company.workforce > 30);

if (company) {
  console.log(`Found company: ${company.name}, Workforce: ${company.workforce}`);
} else {
  console.log('No company found with the specified id and workforce greater than 30.');
}
```

SLIP7

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

```
const fs = require('fs').promises;  // Using promises-based fs module
const path = require('path');

// List of files to read
const files = [
  path.join(__dirname, 'data', 'file1.txt'),
```

```
  path.join(__dirname, 'data', 'file2.txt'),
  path.join(__dirname, 'data', 'file3.txt')
];

// Function to read files asynchronously using async/await
async function readFiles() {
  try {
    const fileContents = await Promise.all(files.map(async (filePath) => {
      const content = await fs.readFile(filePath, 'utf8'); // Read file content asynchronously
      return content;
    }));

    // Print the contents of the files
    fileContents.forEach((content, index) => {
      console.log(`Content of file${index + 1}:`);
      console.log(content);
      console.log('---');
    });
  } catch (error) {
    console.error('Error reading files:', error);
  }
}

// Call the function to read files
readFiles();
```

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 database (Array of users) - This will be used to store users
let users = [
 { id: 1, name: 'Alice', email: 'alice@example.com' },
 { id: 2, name: 'Bob', email: 'bob@example.com' },
];

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

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

 // Create a new user and add it to the in-memory "database"
 const newUser = {
  id: users.length + 1, // Simple auto-increment logic for ID
  name,
  email,
 };

 users.push(newUser);
```

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

// Route to get all users (GET request)
app.get('/users', (req, res) => {
  res.status(200).json(users); // Return the list of users
});

// Route to get a single user by ID (GET request)
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({ message: 'User not found' });
  }

  res.status(200).json(user); // Return the found user
});

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

SLIP 8

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

```
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 imports
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

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

```
  constructor(private http: HttpClient) {}

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

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 bodies
app.use(bodyParser.json());

// In-memory "database" for storing employees
let employees = [];

// Route to create a new employee
app.post('/employee', (req, res) => {
  const { id, name, position, salary } = req.body;

  if (!id || !name || !position || !salary) {
    return res.status(400).json({ error: 'Missing required fields' });
  }
```

```
  // Check if employee with the given ID already exists
  const existingEmployee = employees.find(emp => emp.id === id);
  if (existingEmployee) {
    return res.status(400).json({ error: 'Employee with this ID already exists' });
  }

  const newEmployee = { id, name, position, salary };
  employees.push(newEmployee);

  res.status(201).json(newEmployee);
});

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

  const employeeIndex = employees.findIndex(emp => emp.id === id);

  if (employeeIndex === -1) {
    return res.status(404).json({ error: 'Employee not found' });
  }

  // Update the employee details
  employees[employeeIndex] = {
    id,
    name: name || employees[employeeIndex].name,
    position: position || employees[employeeIndex].position,
    salary: salary || employees[employeeIndex].salary,
```

```
  };

  res.json(employees[employeeIndex]);
});

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

SLIP 9

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

```
interface Company {
  id: number;
  name: string;
  workforce: number;
}

const companies: Company[] = [
  { id: 1, name: 'Company A', workforce: 50 },
  { id: 2, name: 'Company B', workforce: 20 },
  { id: 3, name: 'Company C', workforce: 60 },
  { id: 4, name: 'Company D', workforce: 10 },
```

```
];

// Finding a company with workforce greater than 30
const companyWithLargeWorkforce = companies.find(company =>
company.workforce > 30);

if (companyWithLargeWorkforce) {
  console.log(`Company with workforce greater than 30:
${companyWithLargeWorkforce.name}`);
} else {
  console.log('No company with workforce greater than 30 was
found.');
}
```

) 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.

```
import express, { Request, Response, NextFunction } from 'express';
import { body, validationResult } from 'express-validator';
import bodyParser from 'body-parser';

const app = express();

// Middleware to parse JSON and URL-encoded data
app.use(bodyParser.json());
```

```
app.use(bodyParser.urlencoded({ extended: true }));


// Route to handle form submission (POST)
app.post(
  '/submit',
  // Input validation middleware
  body('name').isString().withMessage('Name must be a string'),
  body('age').isInt({ min: 18 }).withMessage('Age must be an integer
and at least 18'),
  (req: Request, res: Response, next: NextFunction) => {
    // Check if there are validation errors
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({
        success: false,
        errors: errors.array(),
      });
    }
    next();
  },
  (req: Request, res: Response) => {
    // If no validation errors, process the request data
```

```
    const { name, age } = req.body;
    res.json({
      success: true,
      message: 'Form submitted successfully!',
      data: { name, age },
    });
  }
);

// Middleware to handle 404 errors (in case of invalid routes)
app.use((req: Request, res: Response) => {
  res.status(404).json({
    success: false,
    message: 'Not Found',
  });
});

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

SLIP 10

Implement a simple server using Node.js.

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

// Define the server
const server = http.createServer((req, res) => {
  // Set the response header to indicate content type
  res.writeHead(200, { 'Content-Type': 'text/plain' });

  // Write a response message
  res.end('Hello, World!\n');
});

// Define the port and host for the server
const port = 3000;
const host = 'localhost';

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

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.

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

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

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


const app = express();


// Middleware to parse JSON and URL-encoded data

app.use(bodyParser.json()); // for application/json

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


// Example route with validation

app.post('/user', [

  // Validate and sanitize input

  body('username').isLength({ min: 3 }).withMessage('Username must be at least 3 characters long'),

  body('email').isEmail().withMessage('Please provide a valid email address'),

  body('password').isLength({ min: 6 }).withMessage('Password must be at least 6 characters long')

], (req, res) => {

  // Check for validation errors
```

```
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  // Handle valid data
  const { username, email, password } = req.body;
  res.status(200).json({
    message: 'User created successfully',
    data: { username, email }
  });
});

// Example of an endpoint that would handle a GET request
app.get('/', (req, res) => {
  res.status(200).json({
    message: 'Welcome to the API'
  });
});

// Global error handler (optional)
app.use((err, req, res, next) => {
```

```
  console.error(err.stack);
  res.status(500).json({ message: 'Something went wrong!' });
});

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

SLIP11

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

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

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

// Middleware to parse incoming request bodies
app.use(bodyParser.json());

// Sample in-memory movie database (you can replace it with a real database like MongoDB or MySQL later)
```

```
const movies = [];

// Create movie route (POST)
app.post('/movies', (req, res) => {
  const { title, director, releaseYear, genre } = req.body;

  // Validation (you can extend this as per requirements)
  if (!title || !director || !releaseYear || !genre) {
    return res.status(400).json({ error: 'All fields (title, director, releaseYear, genre) are required.' });
  }

  // Create a new movie object
  const newMovie = { id: movies.length + 1, title, director, releaseYear, genre };

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

  // Respond with the newly created movie
  res.status(201).json(newMovie);
});
```

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

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

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


@Component({
  selector: 'app-basketball',
  templateUrl: './basketball.component.html',
  styleUrls: ['./basketball.component.css']
})
export class BasketballComponent implements OnInit {


  students = [
    { name: 'John', playsBasketball: true },
    { name: 'Jane', playsBasketball: false },
    { name: 'Tom', playsBasketball: true },
    { name: 'Lucy', playsBasketball: false },
    { name: 'Alex', playsBasketball: true }
```

```
  ];

  basketballPlayers: string[] = [];

  constructor() { }

  ngOnInit(): void {
    this.filterAndMapBasketballPlayers();
  }

  filterAndMapBasketballPlayers() {
    this.basketballPlayers = this.students
      .filter(student => student.playsBasketball)  // Filters students who play basketball
      .map(student => student.name);  // Maps to an array of names of those students
  }

}
<div>
  <h2>Students Who Play Basketball</h2>
  <ul>
    <li *ngFor="let player of basketballPlayers">{{ player }}</li>
```

```
  </ul>
</div>
```

SLIP12

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>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-app="employeeApp" ng-controller="EmployeeController">

  <h1>Employee Details</h1>

  <table border="1">
    <thead>
      <tr>
```

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

</body>
</html>
// Define the AngularJS module and controller
angular.module('employeeApp', [])
 .controller('EmployeeController', function($scope) {
  // Define the list of employee details
```

```
  $scope.employees = [
    { name: 'John Doe', id: 'E001', pinCode: '12345', address: '1234 Elm Street' },
    { name: 'Jane Smith', id: 'E002', pinCode: '67890', address: '5678 Oak Avenue' },
    { name: 'Mark Johnson', id: 'E003', pinCode: '11223', address: '9101 Pine Road' },
    { name: 'Sarah Williams', id: 'E004', pinCode: '44556', address: '1112 Maple Lane' }
  ];
 });
```

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();
const port = 3000;

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

// In-memory data storage for users
```

```
let users = [];

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

  // Validate that name and email are provided
  if (!name || !email) {
    return res.status(400).json({ message: 'Name and email are required' });
  }

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

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

  // Return the created user with a 201 status
  return res.status(201).json(newUser);
});
```

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

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

SLIP13

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.

```
const express = require('express');
const Joi = require('joi');
const app = express();

// Middleware to parse JSON and form data
app.use(express.json()); // for parsing application/json
app.use(express.urlencoded({ extended: true })); // for parsing application/x-www-form-urlencoded
```

```
// Define a simple route that expects some input data (e.g., username and email)
app.post('/submit', (req, res) => {
  // Validation schema using Joi
  const schema = Joi.object({
    username: Joi.string().min(3).max(30).required().messages({
      'string.base': 'Username should be a string',
      'string.min': 'Username should be at least 3 characters long',
      'string.max': 'Username should not be longer than 30 characters',
      'any.required': 'Username is required',
    }),
    email: Joi.string().email().required().messages({
      'string.email': 'Please provide a valid email address',
      'any.required': 'Email is required',
    }),
  });

  // Validate request data
  const { error, value } = schema.validate(req.body);

  if (error) {
    // If validation fails, return an error response with details
```

```
    return res.status(400).json({
      success: false,
      message: 'Validation error',
      details: error.details.map(detail => detail.message),
    });
  }

  // If validation succeeds, proceed with the logic
  return res.status(200).json({
    success: true,
    message: 'Data successfully received',
    data: value, // Send the valid input data back
  });
});

// Default route
app.get('/', (req, res) => {
  res.send('Welcome to the Express.js app!');
});

// Start server
const PORT = process.env.PORT || 3000;
```

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

Create a simple Angular component that takes input data and displays it.

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

@Component({
  selector: 'app-display-data',
  templateUrl: './display-data.component.html',
  styleUrls: ['./display-data.component.css']
})
export class DisplayDataComponent {
  @Input() data: string = ''; // Input property to accept data
<div>
  <p>Data received: {{ data }}</p>
</div>}
```

SLIP 14

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

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

```
@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  styleUrls: ['./student-list.component.css']
})
export class StudentListComponent implements OnInit {
  students = [
    { name: 'John', score: 90 },
    { name: 'Jane', score: 78 },
    { name: 'Jake', score: 85 },
    { name: 'Sara', score: 92 },
    { name: 'Tom', score: 80 }
  ];

  filteredStudentNames: string[] = [];

  ngOnInit(): void {
    // Filter students who scored 85% or more and map to their names
    this.filteredStudentNames = this.students
      .filter(student => student.score >= 85)  // Filter students by score
      .map(student => student.name);          // Map the filtered students
to their names
```

```
  }
}
```

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

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

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

// In-memory database (just for demo purposes)
let employees = [
  { id: 1, name: 'Alice', position: 'Developer' },
  { id: 2, name: 'Bob', position: 'Manager' }
];

// Create an employee
app.post('/employees', (req, res) => {
  const { name, position } = req.body;
```

```
  // Validate the request body
  if (!name || !position) {
    return res.status(400).json({ error: 'Name and position are
required.' });
  }

  const newEmployee = {
    id: employees.length + 1,
    name,
    position
  };

  employees.push(newEmployee);
  res.status(201).json(newEmployee);
});

// Update an employee
app.put('/employees/:id', (req, res) => {
  const employeeId = parseInt(req.params.id);
  const { name, position } = 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 the employee's data
  if (name) employee.name = name;
  if (position) employee.position = position;

  res.json(employee);
});

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

SLIP 15

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

```
const employees = [
  { id: 1, name: 'John', salary: 30000 },
```

```
  { id: 2, name: 'Jane', salary: 22000 },
  { id: 3, name: 'Alice', salary: 28000 },
  { id: 4, name: 'Bob', salary: 26000 }
];

// Find employee with salary greater than 25000
const employee = employees.find(emp => emp.salary > 25000);

if (employee) {
  console.log(`Employee found: ${employee.name}, Salary: ${employee.salary}`);
} else {
  console.log('No employee found with salary greater than 25000');
}
```

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

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

@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  styleUrls: ['./student-list.component.css']
})
```

```
export class StudentListComponent implements OnInit {
  students = [
    { name: 'John Doe', score: 90 },
    { name: 'Jane Smith', score: 80 },
    { name: 'Bob Brown', score: 85 },
    { name: 'Alice Johnson', score: 95 },
    { name: 'Charlie Lee', score: 70 }
  ];

  studentsWith85Percent: string[] = [];

  ngOnInit(): void {
    // Filter the students who scored 85% or more, then map to get their names.
    this.studentsWith85Percent = this.students
      .filter(student => student.score >= 85)  // Filters students with score 85% or more
      .map(student => student.name);          // Maps to just the names
  }
}
<div>
  <h2>Students with 85% or more:</h2>
  <ul>
```

```
   <li *ngFor="let student of studentsWith85Percent">{{ student
}}</li>
  </ul>
</div>
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 { }
```