Slip 1

Q.1) AngularJS Script for Addition of Two Numbers

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
   <meta charset="UTF-8">
   <title>AngularJS Addition</title>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-controller="additionController" ng-init="a=0; b=0; sum=0; showSum=false">
   <h2>AngularJS Addition Example</h2>
   <!-- Input Fields -->
   <label>Number 1:</label>
   <input type="number" ng-model="a" /><br>
   <label>Number 2:</label>
   <input type="number" ng-model="b" /><br>
   <!--Button -->
   <button ng-click="calculate()" ng-disabled="!a || !b">Calculate</button>
   <button ng-show="showSum" ng-click="clear()">Clear</button>
   <!-- Result -->
   <p ng-show="showSum">The sum is: <span ng-bind="sum"></span></p>
   <script>
      var app = angular.module('myApp', []);
      app.controller('additionController', function($scope) {
         $scope.calculate = function () {
            $scope.sum = parseInt($scope.a) + parseInt($scope.b);
            $scope.showSum = true;
         };
         $scope.clear = function () {
            $scope.a = 0;
            $scope.b = 0;
            $scope.sum = 0;
            $scope.showSum = false;
         };
      });
   </script>
</body>
</html>
```

Q.2) Node.js Application: Reading Data from Multiple Files Asynchronously const

```
fs = require('fs').promises;
async function readFiles() {
   try {
        // Read files asynchronously
        const [data1, data2, data3] = await Promise.all([
            fs.readFile('file1.txt', 'utf8'),
fs.readFile('file2.txt', 'utf8'),           fs.readFile('file3.txt',
'utf8')
 ]);
        console.log('Contents of file1:', data1);
console.log('Contents of file2:', data2);        console.log('Contents
of file3:', data3);
    } catch (err) {
        console.error('Error reading files:', err);
    }
}
readFiles();
run :- node app.js
```

slip 2

Q.1) AngularJS Script to Print Bank Details in Tabular Form Using `ng-repeat`

```
<!DOCTYPE html>
<html lang="en" ng-app="bankApp">
<head>
    <meta charset="UTF-8">
    <title>Bank Details</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
    <style>
table {
width: 50%;
            border-collapse: collapse;
margin: 20px 0;
        }
        th, td {
            border: 1px solid #ddd;
padding: 8px;
            text-align: left;
        }
        th {
            background-color: #f4f4f4;
        }
    </style>
</head>
<body 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.micr}}</td>
        <td>{{bank.ifsc}}</td>
        <td>{{bank.address}}</td>
      </tr>
    </tbody>
  </table>

  <script>
    var app = angular.module('bankApp', []);
    app.controller('bankController', function ($scope) {
      $scope.banks = [
        { name: 'State Bank of India', micr: '123456', ifsc: 'SBIN0000123', address: 'Mumbai, India' },
        { name: 'Punjab National Bank', micr: '654321', ifsc: 'PUNB0001234', address: 'Delhi, India' },
{ name: 'ICICI Bank', micr: '112233', ifsc: 'ICIC0000111', address: 'Bangalore, India' }
      ];
    });
  </script>
</body>
</html>
```

Q.2) Simple Angular Application to Fetch Data from an API Using `HttpClient`

```
ng new api-fetch-app cd api-fetch-app ng generate service api ng generate
component bank-details
```

api.service.ts

```
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) {}

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

bank-details.component.ts

```
import { Component, OnInit } from '@angular/core'; import
{ ApiService } from '../api.service'; @Component({
  selector: 'app-bank-details',   templateUrl:
'./bank-details.component.html',   styleUrls:
['./bank-details.component.css']
})
export class BankDetailsComponent implements OnInit {
data: any[] = [];

  constructor(private apiService: ApiService) {}

  ngOnInit(): void {
    this.apiService.fetchData().subscribe({
next: (response) => {         this.data =
response;
      },
      error: (error) => {
        console.error('Error fetching data:', error);
      }
    });
  }
}
```

bank-details.component.html

```
<h2>API Data</h2>
<table>
  <thead>
    <tr>
      <th>ID</th>
      <th>Title</th>
      <th>Body</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let item of data">
      <td>{{ item.id }}</td>
      <td>{{ item.title }}</td>
      <td>{{ item.body }}</td>
    </tr>
  </tbody>
</table>
```
Add `HttpClientModule` in `app.module.ts` import {
NgModule } from '@angular/core'; import { BrowserModule }
from '@angular/platform-browser'; import {
AppRoutingModule } from './app-routing.module'; import {
AppComponent } from './app.component'; import {
HttpClientModule } from '@angular/common/http';
import { BankDetailsComponent } from './bank-details/bank-details.component'; @NgModule({
```
  declarations: [
    AppComponent,
    BankDetailsComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

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. [15]

```
<!DOCTYPE html>
<html lang="en" ng-app="gameApp">
<head>
    <meta charset="UTF-8">
    <title>Game List</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script> </head>
<body ng-controller="gameController" ng-init="games=[]">
    <h2>Game List</h2>
    <!-- Button to Load Games -->
    <button ng-click="loadGames()">Show Games</button>
    <!-- Display Games -->
    <ul>
        <li ng-repeat="game in games" ng-bind="game"></li>
</ul>
    <script>
```

```
        var app = angular.module('gameApp', []);
        app.controller('gameController', function($scope) {
            $scope.loadGames = function () {
                $scope.games = ['Chess', 'Football', 'Tennis', 'Cricket', 'Hockey'];
            };
        });
    </script>
</body>
</html>
```

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

```
    { id: 1, name: 'TechCorp', workforce: 25 },
    { id: 2, name: 'Innovate Ltd', workforce: 45 },
    { id: 3, name: 'BuildIt Inc', workforce: 15 },
    { id: 4, name: 'MegaWorks', workforce: 50 }
];
// Find the company with a workforce greater than 30
const largeCompany = companies.find(company => company.workforce > 30); if
(largeCompany) {
    console.log('Company with workforce greater than 30:', largeCompany);
} else {
    console.log('No company found with workforce greater than 30.');
}
```

Slip 4

Q.1) Fetch Details Using ng-repeat in AngularJS

```
<!DOCTYPE html>
<html lang="en" ng-app="detailsApp">
<head>
    <meta charset="UTF-8">
    <title>Fetch Details</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-controller="detailsController">
    <h2>Details List</h2>
    <table border="1">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Age</th>
                <th>Profession</th>
            </tr>
        </thead>
        <tbody>
            <tr ng-repeat="detail in details">
                <td>{{detail.id}}</td>
                <td>{{detail.name}}</td>
                <td>{{detail.age}}</td>
                <td>{{detail.profession}}</td>
            </tr>
        </tbody>
    </table>
<script>
        var app = angular.module('detailsApp', []);
        app.controller('detailsController', function ($scope) {
            $scope.details = [
                { id: 1, name: 'Alice', age: 25, profession: 'Engineer' },
                { id: 2, name: 'Bob', age: 30, profession: 'Doctor' },
                { id: 3, name: 'Charlie', age: 28, profession: 'Teacher' }
            ];
        });
    </script>
</body>
</html>
```

Q.2) Express.js Application with Middleware for Parsing Request Bodies and Validating Input Data
Initialize a New Node.js Project: mkdir express-app cd express-app npm init -y npm install express body-parser joi **Create `app.js`**:

```
const express = require('express'); const
bodyParser = require('body-parser');
const Joi = require('joi');
const app = express(); const
port = 3000;
// Middleware for parsing JSON and form data app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
// Validation schema using Joi const userSchema
= Joi.object({     name:
Joi.string().min(3).required(),     age:
Joi.number().integer().min(18).required(),
    email: Joi.string().email().required()
});
// Route to handle POST requests app.post('/users',
(req, res) => {
    const { error, value } = userSchema.validate(req.body);
if (error) {
        return res.status(400).send({ error: error.details[0].message });
    }
    res.send({
        message: 'User data is valid!',
        data: value
    });
});
// Start the server app.listen(port,
() => {
    console.log(`Server is running on http://localhost:${port}`);
});
```

Slip 5

Q.1) Simple Angular Component to Take Input and Display Data

```
ng new simple-angular-app cd simple-angular-app
ng generate component display-input display-input.component.html
<div class="container">
    <h2>Input Data</h2>
    <label for="inputData">Enter Data:</label>
    <input id="inputData" [(ngModel)]="inputData" placeholder="Type something..." />
    <p><strong>Output:</strong> {{ inputData }}</p>
</div>
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 {
  inputData: string = ''; // Variable to hold the input data
}

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 // Include FormsModule here
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { } app.component.html
<app-display-input></app-display-input> ng
serve
```

Q.2) Implement a Simple Server Using Node.js
mkdir simple-server cd simple-server npm init
-y npm install express
Create `server.js`:

```
const express = require('express');
const app = express(); const port
= 3000; // Middleware to parse
JSON app.use(express.json()); //
Simple GET Route app.get('/',
(req, res) => {
   res.send('Welcome to the Node.js Server!');
});
// Simple POST Route
app.post('/submit', (req, res) => {
const data = req.body;
   res.send({
       message: 'Data received successfully!',
receivedData: data
   });
});
// Start the Server app.listen(port,
() => {
   console.log(`Server is running on http://localhost:${port}`);
});
```

slip 6

Q.1) Express.js Application for Create and Read Operations on Products
mkdir express-crud cd express-crud npm init -y npm install express
body-parser **Create `server.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
database for products let products = [];
// Create Product (POST route)
app.post('/products', (req, res) => {
const { id, name, price } = req.body;     if
(!id || !name || !price) {
        return res.status(400).send({ error: 'ID, Name, and Price are required' });
    }
    const newProduct = { id, name, price };
products.push(newProduct);
    res.status(201).send({ message: 'Product created successfully', product: newProduct });
});

// Read Products (GET route) app.get('/products',
(req, res) => {     res.send(products);
});
// Start the server app.listen(port,
() => {
    console.log(`Server running on http://localhost:${port}`);
});
```

Q.2) Find a Company with Workforce Greater than 30 const

```
companies = [
    { id: 1, name: 'TechCorp', workforce: 50 },
    { id: 2, name: 'Innovate Ltd', workforce: 25 },
    { id: 3, name: 'FutureTech', workforce: 75 }
];
// Find a company with workforce greater than 30
const company = companies.find(company => company.workforce > 30); if
(company) {
    console.log('Company with workforce greater than 30:', company);
} else {
    console.log('No company found with workforce greater than 30.');
}
```

Slip 7

Q.1) Node.js Application to Read Data from Multiple Files Asynchronously

```
mkdir node-async-read cd node-async-read npm init -y readFiles.js const
fs = require('fs').promises; // Function to read a single file
async function readFile(fileName) {
    try {
        const data = await fs.readFile(fileName, 'utf-8');
return data;     } catch (err) {
        throw new Error(`Error reading ${fileName}: ${err.message}`);
    }
}
// Function to read multiple files async
function readMultipleFiles(fileNames) {     try
{
        const promises = fileNames.map(readFile);
const results = await Promise.all(promises);
        return results;
} catch (err) {
console.error(err.m
essage);
    }
}
// Main execution (async
() => {
    const fileNames = ['file1.txt', 'file2.txt'];     const
fileContents = await readMultipleFiles(fileNames);
    console.log('File Contents:');
fileContents.forEach((content, index) => {
        console.log(`File ${index + 1}: ${content}`);
    });
})();
node readFiles.js
```

Q.2) Express.js Application for Create and Read Operations on Users

```
mkdir express-user-crud cd express-user-crud npm init -y
npm install express body-parser server.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
database for users let users = [];
// Create User (POST route)
app.post('/users', (req, res) => {
const { id, name, email } = req.body;
if (!id || !name || !email) {
        return res.status(400).send({ error: 'ID, Name, and Email are required' });
    }
    const newUser = { id, name, email };
users.push(newUser);
    res.status(201).send({ message: 'User created successfully', user: newUser });
});
// Read Users (GET route) app.get('/users',
(req, res) => {     res.send(users);
});
// Start the server app.listen(port,
() => {
   console.log(`Server is running on http://localhost:${port}`);
});
```

Slip 8

Q.1) Simple Angular Application to Fetch Data from an API Using HttpClient and Observables

```
ng new angular-httpclient-demo cd angular-httpclient-demo
npm install @angular/common @angular/core rxjs
ng generate component employee-list app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core'; import {
HttpClientModule } from '@angular/common/http'; import {
AppComponent } from './app.component';
import { EmployeeListComponent } from './employee-list/employee-list.component';
@NgModule({
  declarations: [
    AppComponent,
    EmployeeListComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { } ng
generate service employee
employee.service.ts
import { Injectable } from '@angular/core'; import {
HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  private apiUrl = 'https://jsonplaceholder.typicode.com/users'; // Example API
constructor(private http: HttpClient) {}   getEmployees(): Observable<any[]> {
    return this.http.get<any[]>(this.apiUrl);
  }
}
```
employee-list.component.ts
```
import { Component, OnInit } from '@angular/core';
import { EmployeeService } from '../employee.service';
@Component({
  selector: 'app-employee-list',   templateUrl:
'./employee-list.component.html',
  styleUrls: ['./employee-list.component.css']
})
export class EmployeeListComponent implements OnInit {
employees: any[] = [];
  constructor(private employeeService: EmployeeService) {}
ngOnInit(): void {
    this.employeeService.getEmployees().subscribe(data => {
this.employees = data;
    });
  }
}
```
employee-list.component.html
```
<h2>Employee List</h2>
<ul>
  <li *ngFor="let employee of employees">
    {{ employee.name }} - {{ employee.email }}
  </li>
</ul> ng
serve
```

Q.2) Express.js Application for Create and Update Operations on Employees
mkdir express-crud-employee cd express-crud-employee npm init -y npm install express body-parser **Code for `server.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
database for employees let employees = [];
// Create Employee (POST route)
app.post('/employees', (req, res) => {
const { id, name, position } = req.body;     if
(!id || !name || !position) {
        return res.status(400).send({ error: 'ID, Name, and Position are required' });
    }
```

```
   const newEmployee = { id, name, position };
employees.push(newEmployee);
   res.status(201).send({ message: 'Employee created successfully', employee: newEmployee });
});
// Update Employee (PUT route)
app.put('/employees/:id', (req, res) => {
const { id } = req.params;     const {
name, position } = req.body;
   const employee = employees.find(emp => emp.id === parseInt(id));
if (!employee) {
       return res.status(404).send({ error: 'Employee not found' });
   }
   if (name) employee.name = name;     if
(position) employee.position = position;
   res.send({ message: 'Employee updated successfully', employee });
});
// Start the server app.listen(port,
() => {
   console.log(`Server running on http://localhost:${port}`);
});
```

Slip 9

Q.1) Find a Company with a Workforce Greater than 30 const

```
companies = [
   { id: 1, name: "TechCorp", workforce: 25 },
   { id: 2, name: "InnovateInc", workforce: 45 },
   { id: 3, name: "BuildIt", workforce: 15 },
];
const company = companies.find(company => company.workforce > 30); if
(company) {
   console.log(`Company Found: ${company.name} with a workforce of ${company.workforce}`);
} else {
   console.log("No company with a workforce greater than 30 was found.");
}
```

Q.2) Express.js Application with Middleware for Parsing Request Bodies and Validating Input Data

```
mkdir express-middleware-app cd express-middleware-app npm init -y
npm install express body-parser
```

Code for `server.js`

```
const express = require('express'); const
bodyParser = require('body-parser'); const
app = express(); const port = 3000;
// Middleware to parse JSON and form data app.use(bodyParser.json());
```

```
app.use(bodyParser.urlencoded({ extended: true }));
// In-memory database const
users = [];
// Middleware for input validation const
validateUserInput = (req, res, next) => {
const { name, email, age } = req.body;     if
(!name || !email || !age) {
        return res.status(400).json({ error: "Name, Email, and Age are required fields." });
    }

    if (typeof age !== 'number' || age <= 0) {
        return res.status(400).json({ error: "Age must be a positive number." });
    }
    next();
};
// Create User (POST route)
app.post('/users', validateUserInput, (req, res) => {
const { name, email, age } = req.body;
    const newUser = { id: users.length + 1, name, email, age };
users.push(newUser);
    res.status(201).json({ message: "User created successfully", user: newUser });
});
// Get All Users (GET route) app.get('/users',
(req, res) => {     res.json(users);
});
// Start the server app.listen(port,
() => {
    console.log(`Server running on http://localhost:${port}`);
});
```

Slip 10

Q.1) Implement a Simple Server Using Node.js `server.js`:

```
const http = require('http');
// Define the port const
PORT = 3000; // Create the
server
const server = http.createServer((req, res) => {
    // Set response header
    res.writeHead(200, { 'Content-Type': 'text/plain' });

    // Send a response based on the request URL
    if (req.url === '/') {
        res.end('Welcome to the Node.js server!');
    } else if (req.url === '/about') {
        res.end('This is the About page.');
    } else {
        res.end('Page not found.');
    }
});
// Start the server server.listen(PORT,
() => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

Q.2) Extend Express.js Application with Middleware for Parsing and Validation npm install express body-parser

app.js:

```
const express = require('express'); const
bodyParser = require('body-parser'); const
app = express(); const PORT = 4000;
// Middleware for parsing JSON and form data app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
// In-memory database const
employees = [];
// Middleware for input validation const
validateEmployeeInput = (req, res, next) => {
const { name, position, salary } = req.body;     if
(!name || !position || !salary) {
        return res.status(400).json({ error: 'Name, Position, and Salary are required fields.' });
    }
    if (typeof salary !== 'number' || salary <= 0) {
        return res.status(400).json({ error: 'Salary must be a positive number.' });
    }
    next();
};
// Create Employee (POST route)
app.post('/employees', validateEmployeeInput, (req, res) => {
    const { name, position, salary } = req.body;
    const newEmployee = { id: employees.length + 1, name, position, salary };
employees.push(newEmployee);
    res.status(201).json({ message: 'Employee created successfully', employee: newEmployee });
});
// Update Employee (PUT route)
app.put('/employees/:id', validateEmployeeInput, (req, res) => {
const { id } = req.params;
    const { name, position, salary } = req.body;
    const employee = employees.find(emp => emp.id === parseInt(id));
if (!employee) {
        return res.status(404).json({ error: 'Employee not found.' });
    }
    employee.name = name;
employee.position = position;
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 11

Q.1) Develop an Express.js Application That Defines Routes for Create Operations on a Resource (Movie)

```
npm install express
```

app.js

```
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 to store movies
const movies = [];
// POST route to create a movie app.post('/movies',
(req, res) => {     const { title, director, releaseYear, genre
} = req.body;     if (!title || !director || !releaseYear ||
!genre) {
        return res.status(400).json({ error: 'All fields (title, director, releaseYear, genre) are required.' });
    }
    const newMovie = {
        id: movies.length + 1,
        title,
director,
releaseYear,
genre
    };
    movies.push(newMovie);
    res.status(201).json({ message: 'Movie created successfully', movie: newMovie });
});
// Start the server app.listen(PORT,
() => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

Q.2) Create an Angular Application That Prints the Name of Students Who Play Basketball Using `filter` and `map` Methods ng new basketball-app cd basketball-app
ng generate component student-list

`student-list.component.ts`:
```
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', playsBasketball: true },
    { name: 'Jane Smith', playsBasketball: false },
    { name: 'Alice Johnson', playsBasketball: true },
    { name: 'Bob Brown', playsBasketball: false }
  ];
  basketballPlayers: string[] = [];
ngOnInit(): void {
    this.getBasketballPlayers();
  }
  getBasketballPlayers(): void {
    // Filter students who play basketball and map their names
    this.basketballPlayers = this.students
.filter(student => student.playsBasketball)
      .map(student => student.name);
  }
}
```

student-list.component.html
```
<h2>Students Who Play Basketball</h2>
<ul>
  <li *ngFor="let player of basketballPlayers">{{ player }}</li>
</ul>
```

Slip 12

Q.1) AngularJS Script to Print Employee Details in Tabular Form Using `ng-repeat`
```
<!DOCTYPE html>
<html lang="en" ng-app="employeeApp">
<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>
   <style>        table
{           width:
100%;
          border-collapse: collapse;
       }
       th, td {
          border: 1px solid #ddd;
padding: 8px;
          text-align: left;
       }
   </style>
</head>
<body ng-controller="EmployeeController">
   <h2>Employee Details</h2>
   <table>
       <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>
<script>
       var app = angular.module('employeeApp', []);
       app.controller('EmployeeController', function($scope) {
          $scope.employees = [
             { name: 'John Doe', id: 'E001', pinCode: '12345', address: '123 Main St' },
             { name: 'Jane Smith', id: 'E002', pinCode: '67890', address: '456 Elm St' },
             { name: 'Sam Brown', id: 'E003', pinCode: '54321', address: '789 Oak St' },
{ name: 'Lisa Green', id: 'E004', pinCode: '98765', address: '321 Pine St' }
          ];
       });
   </script>
</body>
</html>
```

Q.2) Develop an Express.js Application That Defines Routes for Create Operations on a Resource (User)
npm install express body-parser app.js

```
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 to store users let
users = [];
// POST route to create a new user
app.post('/users', (req, res) => {     const { name,
email, age, address } = req.body;     if (!name ||
!email || !age || !address) {
       return res.status(400).json({ error: 'All fields (name, email, age, address) are required.' });
   }
   const newUser = {
id: users.length + 1,
name,        email,
age,
       address
   };
   users.push(newUser);
   res.status(201).json({ message: 'User created successfully', user: newUser });
});
// Start the server app.listen(PORT,
() => {
   console.log(`Server is running on http://localhost:${PORT}`);
});
```

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 npm install express body-parser app.js

```
const express = require('express'); const
bodyParser = require('body-parser'); const
app = express(); const PORT = 3000;
// Middleware to parse JSON bodies app.use(bodyParser.json());
// Middleware to parse form data (application/x-www-form-urlencoded)
app.use(bodyParser.urlencoded({ extended: true }));
// In-memory database to store users let
users = [];
```

```
// POST route to create a new user with validation app.post('/users',
(req, res) => {
   const { name, email, age, address } = req.body;
   // Validate input data
   if (!name || !email || !age || !address) {
      return res.status(400).json({ error: 'All fields (name, email, age, address) are required.' });
   }
   // Additional email validation (simple example)
   const emailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;
if (!emailPattern.test(email)) {
      return res.status(400).json({ error: 'Invalid email format.' });
   }
   // Additional age validation (must be a positive integer)
if (isNaN(age) || age <= 0) {
      return res.status(400).json({ error: 'Age must be a positive number.' });
   }
   const newUser = {
id: users.length + 1,
name,      email,
      age,
address
   };
   users.push(newUser);
   res.status(201).json({ message: 'User created successfully', user: newUser });
});
// Get all users (for testing purposes) app.get('/users',
(req, res) => {
   res.status(200).json({ users });
});
// Start the server app.listen(PORT,
() => {
   console.log(`Server is running on http://localhost:${PORT}`);
});
```

Q.2) Create a Simple Angular Component that Takes Input Data and Displays It
ng new input-display-app cd input-display-app ng generate component input-display **input-display.component.ts**: import { Component } from '@angular/core';

```
@Component({
  selector: 'app-input-display',   templateUrl:
'./input-display.component.html',   styleUrls:
['./input-display.component.css']
})
export class InputDisplayComponent {
 userName: string = '';
userAge: number = 0;
userAddress: string = '';
submittedData: any = null;
onSubmit(): void {
this.submittedData = {
name: this.userName,      age:
this.userAge,      address:
this.userAddress
  };
 }
}
```

input-display.component.html:

```
<div>
  <h2>User Information</h2>
  <form (ngSubmit)="onSubmit()">
    <label for="name">Name:</label>
    <input type="text" id="name" [(ngModel)]="userName" name="name" required>
    <label for="age">Age:</label>
    <input type="number" id="age" [(ngModel)]="userAge" name="age" required>
<label for="address">Address:</label>
    <input type="text" id="address" [(ngModel)]="userAddress" name="address"
required>
    <button type="submit">Submit</button>
  </form>
  <div *ngIf="submittedData">
    <h3>Submitted Data:</h3>
    <p>Name: {{ submittedData.name }}</p>
    <p>Age: {{ submittedData.age }}</p>
    <p>Address: {{ submittedData.address }}</p>
  </div>
</div>
```

input-display.component.css (Optional, for styling):

```
form {
  margin-bottom: 20px;
} label {
display: block;
margin-top: 10px;
} input
{
  margin-bottom: 10px;
} button
{
  margin-top: 10px;
} ng
serve
slip 14
```

Q.1) Create an Angular Application that Prints the Name of Students Who Got 85% Using `filter` and `map` Method

```
ng new student-app cd student-app
ng generate component student-list
```

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 {
students = [
    { name: 'John Doe', marks: 92 },
    { name: 'Jane Smith', marks: 85 },
    { name: 'Alice Brown', marks: 88 },
    { name: 'Bob White', marks: 74 },
    { name: 'Charlie Green', marks: 90 }
  ];
  // Filtering and mapping the students who got 85% and above
highScoringStudents = this.students
    .filter(student => student.marks >= 85)
    .map(student => student.name);
}
```

student-list.component.html

```
<div>
  <h2>Students Who Got 85% and Above</h2>
  <ul>
    <li *ngFor="let student of highScoringStudents">{{ student }}</li>
  </ul>
</div> ng
serve
```

Q.2) Develop an Express.js Application that Defines Routes for Create, Update Operations on a Resource (Employee)

```
npm init -y
npm install express body-parser
```

app.js

```
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 to store employees let
employees = [];
// POST route to create a new employee app.post('/employees',
(req, res) => {
    const { id, name, position, salary } = req.body;

    // Validate input data
    if (!id || !name || !position || !salary) {
        return res.status(400).json({ error: 'All fields (id, name, position,
salary) are required.' });
    }
    const newEmployee = { id, name, position, salary };
employees.push(newEmployee);
```

```
    res.status(201).json({ message: 'Employee created successfully',
employee: newEmployee });
});
// PUT route to update an existing employee by ID
app.put('/employees/:id', (req, res) => {
const { id } = req.params;
    const { name, position, salary } = req.body;

    // Find employee by ID
    const employee = employees.find(emp => emp.id === parseInt(id));
if (!employee) {
        return res.status(404).json({ error: 'Employee not found.' });
}
    // Update employee data
    employee.name = name || employee.name;
employee.position = position || employee.position;
employee.salary = salary || employee.salary;

    res.status(200).json({ message: 'Employee updated successfully', employee
});
});

// GET route to fetch all employees (for testing) app.get('/employees',
(req, res) => {     res.status(200).json({ employees });
});
// Start the server app.listen(PORT,
() => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

Slip 15

Q.1) Find an Employee with a Salary Greater Than 25000 in the Array (Using `find` by ID Method)

```
const employees = [
    { id: 1, name: 'John Doe', salary: 30000 },
    { id: 2, name: 'Jane Smith', salary: 20000 },
    { id: 3, name: 'Alice Brown', salary: 35000 },
    { id: 4, name: 'Bob White', salary: 24000 },
    { id: 5, name: 'Charlie Green', salary: 27000 }
];
// Find employee with salary greater than 25000
const employeeWithHighSalary = employees.find(employee => employee.salary >
25000);
console.log(employeeWithHighSalary);
```

Q.2) Create an Angular Application That Prints the Name of Students Who Got 85% Using `filter` and `map` Method `ng new student-app cd student-app`

```
ng generate component student-list
```

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 {
students = [
    { name: 'John Doe', percentage: 92 },
    { name: 'Jane Smith', percentage: 85 },
    { name: 'Alice Brown', percentage: 88 },
    { name: 'Bob White', percentage: 74 },
    { name: 'Charlie Green', percentage: 90 }
  ];
  // Filter students who got 85% or more and map to their names
highScoringStudents = this.students
    .filter(student => student.percentage >= 85)
    .map(student => student.name);
}
```

student-list.component.html

```
<div>
  <h2>Students Who Got 85% and Above</h2>
  <ul>
    <li *ngFor="let student of highScoringStudents">{{ student }}</li>
</ul>
</div>
```