Patient

```
export class Patient {
  patientID: number;
  name: string;
  medicalCondition: string;

  constructor(patientID: number, name: string, medicalCondition: string) {
    this.patientID = patientID;
    this.name = name;
    this.medicalCondition = medicalCondition;
  }

  // displayDetails() method to log patient details
  displayDetails(): void {
    console.log(`Patient ID: ${this.patientID}, Name: ${this.name}, Medical Condition:
${this.medicalCondition}`);
  }

  // hasCondition() method to check if the patient has a specific condition
  hasCondition(condition: string): string {
    if (this.medicalCondition === condition) {
      return `This patient has ${this.medicalCondition}.`;
    } else {
      return `This patient does not have ${condition}.`;
    }
  }
}

// Create a new Patient object
const patient1 = new Patient(1, 'John Doe', 'Diabetes');

// Call displayDetails and hasCondition methods
patient1.displayDetails();
console.log(patient1.hasCondition('Diabetes'));  // This patient has Diabetes.
console.log(patient1.hasCondition('Cancer'));  // This patient does not have Cancer.


//All test cases passed code

export class Patient {
  patientID: number;
  name: string;
  medicalCondition: string;
  constructor(patientID: number, name: string, medicalCondition: string) {
    this.patientID = patientID;
    this.name = name;
    this.medicalCondition = medicalCondition;
  }

  // displayDetails() method to log patient details
  displayDetails(): void {
    console.log(`Patient ID: ${this.patientID}, Name: ${this.name}, Medical Condition:
${this.medicalCondition}`);
  }

  // hasCondition() method to check if the patient has a specific condition
  hasCondition(condition: string): boolean {
    if (this.medicalCondition === condition) {
      return true;
    } else {
      return false;
    }
  }
}

// Create a new Patient object
const patient1 = new Patient(1, 'John Doe', 'Diabetes');

// Call displayDetails and hasCondition methods
patient1.displayDetails();
console.log(patient1.hasCondition('Diabetes'));  // This patient has Diabetes.
console.log(patient1.hasCondition('Cancer'));  // This patient does not have Cancer.
```

==

Shopping Cart

```
const products = [
  { id: 1, name: "Product 1", price: 10 },
  { id: 2, name: "Product 2", price: 20 },
  { id: 3, name: "Product 3", price: 30 }
];

const shoppingCart = {
  items: [],
  coupon: null,

  // Function to add products to the cart
  addToCart: function(productId, quantity) {
    const product = products.find(p => p.id === productId);
    if (!product) {
      console.log("Product not found!");
      return;
    }
    const existingItem = this.items.find(item => item.product.id === productId);
    if (existingItem) {
      existingItem.quantity += quantity;
    } else {
      this.items.push({ product, quantity });
    }
    console.log(`${quantity} x ${product.name} added to cart.`);
  },

  // Function to view the current contents of the cart
  viewCart: function() {
    console.log("Cart Contents:");
    if (this.items.length === 0) {
      console.log("Your cart is empty.");
      return;
    }
    this.items.forEach(item => {
      console.log(
        `${item.product.name} - $${item.product.price} x ${item.quantity} =
$${item.product.price * item.quantity}`
      );
    });
  },

  // Function to apply a coupon code
  applyCoupon: function(couponCode) {
    // Let's assume we have only one valid coupon for simplicity
    const validCoupons = {
      "DISCOUNT10": 10, // 10% discount
      "DISCOUNT20": 20  // 20% discount
    };

    if (validCoupons[couponCode]) {
      this.coupon = { code: couponCode, discount: validCoupons[couponCode] };
      console.log(`Coupon ${couponCode} applied. You get ${validCoupons[couponCode]}%
off.`);
    } else {
      console.log("Invalid coupon code.");
    }
  },

  // Function to calculate the total payable amount
  calculateTotalAmount: function() {
    let total = 0;
    this.items.forEach(item => {
      total += item.product.price * item.quantity;
    });

    if (this.coupon) {
      const discountAmount = (total * this.coupon.discount) / 100;
      total -= discountAmount;
      console.log(`Discount of ${this.coupon.discount}% applied: -
$${discountAmount.toFixed(2)}`);
    }

    console.log(`Total Payable Amount: $${total.toFixed(2)}`);
    return total;
  }
};

// Example usage
shoppingCart.addToCart(1, 2);
shoppingCart.addToCart(2, 1);
shoppingCart.viewCart();
shoppingCart.applyCoupon("DISCOUNT10");
shoppingCart.calculateTotalAmount();

module.exports = shoppingCart;
```

==

Average Marks

```
// Define two arrays: one for names and another for marks
const names: string[] = ["A", "B"];
const marks: number[] = [10, 20];
```

```
// Display names and marks using a for loop
console.log("Student Names and Marks");
for (let i = 0; i < names.length; i++) {
  console.log(`${names[i]}: ${marks[i]}`);
}

// Function to calculate average
export function findAvg(marks: number[]): number {
  let tot = 0;
  for (let i = 0; i < marks.length; i++) {
    tot += marks[i];
  }
  const averageMarks = tot / marks.length;
  return averageMarks;
}

// Display the average
console.log(`\nAverage Marks: ${findAvg(marks)}`);
```

==

SpringBoot Booking API Question

Booking.entity

```
package com.hotelbooking.entity;

public class Booking {
    private int id;
    private String guestName;
    private int roomNumber;

    public Booking() {
    }

    public Booking(String guestName, int roomNumber) {
        this.guestName = guestName;
        this.roomNumber = roomNumber;
    }
    public Booking(String guestName, int roomNumber, int id) {
        this.guestName = guestName;
        this.roomNumber = roomNumber;
        this.id = id;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getGuestName() {
        return guestName;
    }

    public void setGuestName(String guestName) {
        this.guestName = guestName;
    }

    public int getRoomNumber() {
        return roomNumber;
    }

    public void setRoomNumber(int roomNumber) {
        this.roomNumber = roomNumber;
    }
}
```

BookingService.java

```
package com.hotelbooking.service;

import com.hotelbooking.entity.Booking;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class BookingService {
    private static int counter = 0;
    private final List<Booking> bookingList = new ArrayList<>();

    public Booking addBooking(Booking booking) {
        booking.setId(++counter);
        bookingList.add(booking);
        return booking;
    }

    public Booking getBookingById(int id) {
        return bookingList.stream()
                .filter(b -> b.getId() == id)
                .findFirst()
                .orElse(null);
    }
}
```

BookingController

```
package com.hotelbooking.controller;
import com.hotelbooking.entity.Booking;
import com.hotelbooking.service.BookingService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/bookings")
public class BookingController {

    private final BookingService bookingService;

    // Constructor-based injection is preferred
    public BookingController(BookingService bookingService) {
        this.bookingService = bookingService;
    }

    @PostMapping
    public ResponseEntity<Booking> addBooking(@RequestBody Booking booking) {
        Booking savedBooking = bookingService.addBooking(booking);
        return new ResponseEntity<>(savedBooking, HttpStatus.CREATED);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Booking> getBookingById(@PathVariable int id) {
        Booking booking = bookingService.getBookingById(id);
        if (booking != null) {
            return new ResponseEntity<>(booking, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
}
```

==

React Question

```
import React, { useState, useEffect, useCallback } from "react";
import { Map, GoogleApiWrapper } from "google-maps-react";
import LocationMarker from "./LocationMarker";
export const App = ({ google }) => {
  const [properties, setProperties] = useState([]);
  const [searchQuery, setSearchQuery] = useState("");
  const [searchResults, setSearchResults] = useState([]);
  const [mapCenter, setMapCenter] = useState({ lat: 31.5497, lng: 74.3436 });
  const [isMounted, setIsMounted] = useState(true);
  const [map, setMap] = useState(null);
  const [markers, setMarkers] = useState([]);



  const handleMapReady = (mapProps, map) => {
    setMap(map);
    setMapCenter(map.center.toJSON());

      };

  const handleSearch = () => {
    if(!google || google.maps) return;
    const service=new google.maps.places.PlaceService(map);
    service.textSearch({query:searchQuery},(results,status)=>{
      if(status==="OK"){
        setSearchResults(results);
      }
    });
  };

  const handleAddLocation = (result) => {
    const location=result.geometry.location;
    const position={
      lat:location.let(),
      lng:location.lng(),
    };

    const marker=new google.maps.Marker({
      position,map,title:result.name,
    });
    setMarkers((prev)=>[...prev,marker]);
    setProperties((prev)=>[...prev,{name:result.name,position}]);
    setSearchResults([]);
    setSearchQuery("");


  };

  const handleRemoveLocation = useCallback(
   (index)=>{
    const newProperties=[...properties];
    newProperties.splice(index,1);
    setProperties(newProperties);
    removeMarker(index);
  },
    [properties]
  );

  const removeMarker = useCallback(


        (index)=>{
          if(index<0 || index>=markers[index]) return;
        const marker=markers[index];
        markers.setMap(null);
        const newMarkers=[...markers];
        newMarkers.splice(index,1);
        setMarkers(newMarkers);
        },
          [markers]
      );


  const handleMapClick = (mapProps, map, clickEvent) => {
    const geocoder = new google.maps.Geocoder();
    const latLng={
      lat: clickEvent.latLng.lat(),
          lng: clickEvent.latLng.lng()
    }
    geocoder.geocode(
      {
        location: {
          lat: clickEvent.latLng.lat(),
          lng: clickEvent.latLng.lng(),
        },
      },
      (results, status) => {
        if (status === "OK") {
          if (results[0]) {
            const marker=new
google.maps.Marker({position:latLng,map,title:results[0].formatted_address,});
            setMarkers((prev)=>[...prev,marker]);
            setProperties((prev)=>[...prev,{name:results[0].formatted_address,position:latLng},]);

        } else {
          console.log("Geocoder failed due to: " + status);
        }
      }
  });
  };

  useEffect(() => {
    if (properties.length > 0 && isMounted) {
     if(properties.length>0){
        setMapCenter(properties[properties.length-1].position);
      }

    }
  }, [properties]);

  return (
    <div style={{ display: "flex" }}>
```

```
      <div style={{ flex: "1 1 50%", position: "relative", height: "500px" }}>
      <label htmlFor="search">Enter location</label>
        <input type="text" id="search" value={searchQuery}
onChange={(e)=>setSearchQuery(e.target.value)} placeholder="Search Location"/>
        <button onClick={handleSearch} >Search</button>

        <ul>
          {searchResults.map((result,index)=>{
            <li key={index}>{result.name}<button onClick={
              ()=>{
                handleAddLocation(result)
              }
            }>Add</button></li>
          })}
        </ul>
        <h3>Saved Location</h3>
        <ul>
          {properties.map((prop,index)=>(

            <li key={index}>{prop.name}<button onClick={
              ()=>{
                handleRemoveLocation(index)
              }}>Remove</button></li>
            ))}
        </ul>
      </div>

      <div>
        <Map
        google={google}
        zoom={5}
        initialCenter={mapCenter}
        onReady={handleMapReady}
        onClick={handleMapClick}
        >
      {properties.map((prop,index)=>(
        <LocationMarker
        key={index}
        position={prop.position}
        map={map}
        marker={markers[index]}
        onRemove={()=>handleRemoveLocation(index)}
        />
      ))}

        </Map>
      </div>
    </div>
  );
};

export default GoogleApiWrapper({
  apiKey: "AIzaSyDh0LyUchQyqlcsHgYRO5w7iUV4ttlNdDI",
})(App);
```

LocationMarker.js

```
import { useEffect } from 'react';

const LocationMarker = ({ position, map, marker, onRemove }) => {
  useEffect(() => {
      if(!marker)return;
      const handleClick=()=>{
        if(onRemove) onRemove();
        marker.setMap(null);
      };
      marker.addListener("click",handleClick);

    return () => {
      window.google.maps.event.clearListeners(marker,"click");
    };
  }, [map, position, marker, onRemove]);

  return null;
};
export default LocationMarker;
```

==

Angular Question

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot, UrlTree }
from '@angular/router';
import { AuthService } from './auth.service';
import { Observable } from 'rxjs';
```

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) {

   }

   canActivate(route:ActivatedRouteSnapshot,state:RouterStateSnapshot):boolean{
    const expectedRole=route.data['role'];
    if(!this.authService.isLoggedIn){
      this.router.navigate(['/login']);
      return false;
    }
    if(expectedRole === 'admin' && !this.authService.isAdmin){
      this.router.navigate(['/unauthorized']);
      return false;
    }

    if(expectedRole === 'user' && this.authService.isAdmin){
      this.router.navigate(['/unauthorized']);
      return false;
    }


    return true;
  }
}



----
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private readonly AUTH_TOKEN_KEY = 'auth_token';
  private readonly IS_ADMIN_KEY = 'is_admin';
  private readonly USERNAME_KEY='username';
  constructor() {
   //complete missing code here

  }

  get isLoggedIn(): boolean {
   //complete missing code here
   return localStorage.getItem(this.AUTH_TOKEN_KEY)==='true';
  }

  set isLoggedIn(value: boolean) {
  //complete missing code here
    localStorage.setItem(this.AUTH_TOKEN_KEY,value.toString());
  }


  get isAdmin(): boolean {
//complete missing code here
return localStorage.getItem(this.IS_ADMIN_KEY)==='true';

  }

  set SetUser(value: string) {
  //complete missing code here
  localStorage.setItem(this.USERNAME_KEY,value);
  localStorage.setItem(this.AUTH_TOKEN_KEY,'true');

  const is_admin=value==='is_admin';
  localStorage.setItem(this.IS_ADMIN_KEY,is_admin.toString());

  // localStorage.setItem(this.USERNAME_KEY,!is_admin.)
  }

  login(username: string, password: string): boolean {
    // Implement authentication logic
   //complete missing code here
   if((username === 'admin' && password === 'admin') || (username === 'user' && password
=== 'user')){
    this.isLoggedIn=true;
    localStorage.setItem(this.IS_ADMIN_KEY,(username ==='admin').toString());
    this.SetUser=username;
    return true;
   }
   return false;
  }

  logout(): void {
//complete missing code here
localStorage.removeItem(this.AUTH_TOKEN_KEY);

localStorage.removeItem(this.IS_ADMIN_KEY);
localStorage.removeItem(this.USERNAME_KEY);
}
}

--
import { CommonModule } from '@angular/common';
import { Component, OnInit } from '@angular/core';
import { AbstractControl, FormBuilder, FormGroup, FormsModule, Validators } from
'@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from '../services/auth.service';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrl: './login.component.scss'
})
export class LoginComponent implements OnInit{
  formModel:any={}
  itemForm: FormGroup;
  loginFailed:boolean=false;
  constructor(private authService: AuthService,private formBuilder: FormBuilder, private
router:Router)
    {
      this.itemForm=this.formBuilder.group({
        username:['',Validators.required],
        password:['',Validators.required]
      });

    }
  ngOnInit(): void {
  }
  onRegister()
  {
    if(this.itemForm.valid)
    {
      const username=this.itemForm.value.username;
      const password=this.itemForm.value.password;
      const success=this.authService.login(username,password);
      if(success){
        this.loginFailed=false;
        if(this.authService.isAdmin){
          this.router.navigate(['/admin']);
        }
        else{
          this.router.navigate(['/user']);
        }
      }
      else{
        this.itemForm.markAllAsTouched();
      }
    }
    else{
      this.itemForm.markAllAsTouched();

    }
  }

}
```

==

HTML Article Publishing

```
<!DOCTYPE html>

<html>

  <head>

    <title>CMS: Article Publishing Interface</title>

    <link

      rel="stylesheet"

      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"

      integrity="sha512-
z3gLpd7yknf1YoNbCzqRKc4qyor8gaKU1qmn+CShxbuBusANI9QpRohGBreCFkKxLhei6
S9CQXFEbbKuqLg0DA=="

      crossorigin="anonymous"
      referrerpolicy="no-referrer"
    />
    <style>
      body {
        background-color: #1e1e2f;
      }

      form {
        display: flex;
        flex-direction: column;
        width: 50%;
        justify-content: center;
        align-items: center;
        border: 1px solid #fff;
        margin: 0 auto;
        padding: 10px;
      }

      div {
        width: 100%;
        display: flex;
        justify-content: center;
        margin: 1rem;
      }

      label {
        width: 20%;
      }
```

```
input[type="text"],
select,
textarea {
  width: 80%;
  padding: 8px;
  border: 1px solid #ccc;
}

input[type="checkbox"] {
  transform: scale(1.2);
}

a:hover {
  color: orange;
}

.publish {
  justify-content: start;
  align-items: center;
  gap: 10px;
}

.summary,
.error {
  color: white;
}

.error {
  background-color: red;
  flex-direction: column;
}

.error p {
  text-align: center;
}

.error p::first-letter {
  text-transform: capitalize;
}

.articles {
  display: flex;
  flex-direction: column;
  width: 51%;
  justify-content: center;
  align-items: center;
  margin: 0 auto;
}

.article {
  display: flex;
  width: 100%;
  flex-direction: column;
  border: 1px solid #fff;
  color: white;
  align-items: center;
  margin-bottom: 10px;
  padding: 10px;
```

```
    }

    .delete-icon {
      color: red;
      padding: 10px;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <form>
    <h2 style="color: white">CMS: Article Publishing Interface</h2>
    <div>
      <label for="title" style="color: white">Article Title</label>
      <input type="text" id="title" />
    </div>

    <div>
      <label for="content" style="color: white">Article Content</label>
      <textarea id="content" rows="4"></textarea>
    </div>

    <div>
      <label for="category" style="color: white">Category</label>
      <select id="category">
        <option value="Technology">Technology</option>
        <option value="Lifestyle">Lifestyle</option>
        <option value="Business">Business</option>
      </select>
  </div>

  <div class="publish">
    <label for="is_published" style="color: white">Publish</label>
    <input type="checkbox" id="is_published" />
  </div>

  <div>
    <button id="add-article">Submit</button>
  </div>

  <div id="error"></div>
</form>

<script type="text/javascript">
  const articles = [];

  const btn = document.getElementById("add-article");
  btn.addEventListener("click", addArticle);

  function addArticle(e) {
    e.preventDefault();
    let errorMessages = [];
    const article = {};

    const title = document.getElementById("title").value.trim();
    const content = document.getElementById("content").value.trim();
    const category = document.getElementById("category").value;
    const isPublished = document.getElementById("is_published").checked;
    const errorDiv = document.getElementById("error");
    errorDiv.innerHTML = "";

    if (!title) errorMessages.push("title is empty ");
    if (!content) errorMessages.push("content is empty ");

    if (errorMessages.length > 0) {
      errorDiv.innerHTML = errorMessages.map(msg => `<p>${msg}</p>`).join("") +
"<p></p>";
    } else {
      article.title = title;
      article.content = content;
      article.category = category;
      article.isPublished = isPublished;

      articles.push(article);
      displayArticles(articles);

      document.getElementById("title").value = "";
      document.getElementById("content").value = "";
      document.getElementById("category").value = "Technology";
      document.getElementById("is_published").checked = false;
    }
  }

  function displayArticles(articles) {
    const existingContainer = document.querySelector(".articles");
    if (existingContainer) {
```

```
  document.body.removeChild(existingContainer);
}
const articlesContainer = document.createElement("div");
articlesContainer.classList.add("articles");

articles.forEach((article, index) => {
  const articleDiv = document.createElement("div");
  articleDiv.classList.add("article");

  const title = document.createElement("h3");
  title.textContent = article.title;

  const category = document.createElement("p");
  category.textContent = "Category: " + article.category;

  const status = document.createElement("p");
  status.textContent = "Status: " + (article.isPublished ? "Published" : "Draft");

  const deleteIcon = document.createElement("i");
  deleteIcon.classList.add("fas", "fa-trash-alt", "delete-icon");
  deleteIcon.addEventListener("click", () => {
    articles.splice(index, 1);
    displayArticles(articles);
  });

  articleDiv.appendChild(title);
  articleDiv.appendChild(category);
  articleDiv.appendChild(status);
  articleDiv.appendChild(deleteIcon);
```

```
          articlesContainer.appendChild(articleDiv);
        });

        document.body.appendChild(articlesContainer);
      }
    </script>
  </body>
</html>
```

==

React Patient Question

```
// PatientForm.js
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

const PatientForm = () => {
  const [formData, setFormData] = useState({
    name: '',
    dob: '',
    medicalHistory: '',
    currentMedications: '',
  });

  const [errors, setErrors] = useState({});
  const navigate = useNavigate();

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
```

```
   setErrors({ ...errors, [name]: '' });
  };

  const validate = (data) => {
   const errors = {};
   if (!data.name.trim()) errors.name = 'Name is required';
   if (!data.dob) errors.dob = 'Date of Birth is required';
   else if (new Date(data.dob) > new Date()) {
     errors.dob = 'Date of Birth cannot be a future date';
   }
   return errors;
  };

  const handleSubmit = (e) => {
   e.preventDefault();
   const validationErrors = validate(formData);
   if (Object.keys(validationErrors).length > 0) {
     setErrors(validationErrors);
     navigate('/error');
   } else {
     setErrors({});
     navigate('/welcome');
   }
  };

  return (
   <div className="form-container">
     <h1>Patient Registration</h1>
     <form onSubmit={handleSubmit}>
```

```
        <div>
          <label htmlFor="name">Name:</label>
          <input type="text" id="name" name="name" value={formData.name}
onChange={handleChange} />
          {errors.name && <p>{errors.name}</p>}
        </div>
        <div>
          <label htmlFor="dob">Date of Birth:</label>
          <input type="date" id="dob" name="dob" value={formData.dob}
onChange={handleChange} />
          {errors.dob && <p>{errors.dob}</p>}
        </div>
        <div>
          <label htmlFor="medicalHistory">Medical History:</label>
          <textarea id="medicalHistory" name="medicalHistory"
value={formData.medicalHistory} onChange={handleChange}></textarea>
        </div>
        <div>
          <label htmlFor="currentMedications">Current Medications:</label>
          <input type="text" id="currentMedications" name="currentMedications"
value={formData.currentMedications} onChange={handleChange} />
        </div>
        <button type="submit">Submit</button>
      </form>
    </div>
  );
};

export default PatientForm;
```

```
import React from 'react';

const ErrorPage = () => {
  return <h1>Error: Invalid form submission</h1>;
};

export default ErrorPage;




// WelcomePage.js

import React from 'react';

const WelcomePage = () => {
  return <h1>Welcome! Patient registered successfully.</h1>;
};

export default WelcomePage;
```

==

Angular Employee Question

employee component

```
import { Component } from '@angular/core';
import { Employee } from '../model/employee.model';

@Component({
  selector: 'app-employee',
  templateUrl: './employee.component.html',
  styleUrls: ['./employee.component.scss']
})
export class EmployeeComponent {
  // Create an employee list here
  employees: Employee[] = [
    { id: 1, name: 'Monisha', position: 'Software Engineer'},
    { id: 2, name: 'Roshna', position: 'Product Manager' },
    { id: 3, name: 'Neha', position: 'UX Designer'},

  ];
}
```

```
//employee html

<table>
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Position</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let employee of employees">
```

```
      <td>{{ employee.id }}</td>
      <td>{{ employee.name }}</td>
      <td>{{ employee.position }}</td>
    </tr>
  </tbody>
</table>
```

vote html

```
<div>
  <!-- Label for the age input field -->
  <label for="age">Enter your Age</label>

  <!-- Input field with two-way data binding to 'userAge' -->
  <input id="age" [(ngModel)]="userAge" type="number" placeholder="Enter your age" />

  <!-- Button to trigger eligibility check -->
  <button (click)="checkEligibility()">Check Eligibility</button>

  <!-- Paragraph to display the eligibility message -->
  <p *ngIf="eligibilityMessage">{{ eligibilityMessage }}</p>
</div>
```