Name: sharmila Sithravelautham  
Email: sharmisithravelayutham@gmail.com

**Answer Sheet**

**Section 1:**

1. **Role of Angular Services:**

* Angular services are used to hold business logic or data that you want to share across multiple components.
* They are singleton instances, which means the same instance is shared by every component that injects them.
* Services help keep the component code clean by handling data processing, HTTP requests, and other reusable tasks.

1. **Handling HTTP Requests and Responses in Angular:**

* Angular uses the **HttpClient module** to manage HTTP requests and responses.
* This module provides methods like get(), post(), put(), and delete() to communicate with APIs.
* These methods return Observables, making it easy to handle asynchronous data and integrate with RxJS operators for efficient error handling and data transformations.

1. **Benefits of Using RxJS in Angular:**

* RxJS allows Angular apps to handle asynchronous data streams effectively.
* It makes it easy to work with Observables, which represent ongoing events.
* RxJS operators (like map, filter, catchError) let you process data streams, manage complex data flows, and make the code more readable and maintainable.

1. **How Dependency Injection Works in Angular:**

* Dependency Injection (DI) is a design pattern that Angular uses to manage how objects (dependencies) are created and shared.
* When a component needs a service, Angular’s DI system creates an instance of that service (or provides an existing one) and "injects" it into the component.
* This reduces tight coupling between components and services and makes testing easier.

1. **Setting Up a C# Web API Project:** To set up a C# Web API project:
   * Must create a new project and select "ASP.NET Core Web API." Using Visual Studio.
   * Choose the framework version and template options.
   * Visual Studio will generate a sample project with folders for controllers, models, and other configurations.
   * We can then add controllers and define endpoints to handle HTTP requests.

**Section 3:**

**Fixing Issues in the Angular Service Code:**

The throwError function is missing from the rxjs import.

Have to import { throwError } from 'rxjs'; at the top of the component.

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

import { HttpClient } from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';

@Injectable({

providedIn: 'root'

})

export class ItemService {

private apiUrl = 'http://localhost:5000/api/items';

constructor(private http: HttpClient) {}

getItems(): Observable<Item[]> {

return this.http.get<Item[]>(this.apiUrl).pipe(

catchError(this.handleError)

);

}

private handleError(error: any): Observable<never> {

console.error('An error occurred:', error);

return throwError('Something went wrong; please try again later.');

}

}

**Fixing Issues in the C# Web API Controller Code:**

* If Item has required properties, it’s good to check if the model is valid before adding it to the list. You can add a ModelState.IsValid check for better validation.
* Currently, CreatedAtAction is referencing GetItems, which returns a list of items, not a single item. This doesn’t align with CreatedAtAction, which should ideally reference a specific item by id.
* Could add a new GetItemById method and then reference it for CreatedAtAction.

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Linq;

namespace MyApi.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ItemsController : ControllerBase

{

private static List<Item> items = new List<Item>();

[HttpGet]

public ActionResult<IEnumerable<Item>> GetItems()

{

return items;

}

// New method to get item by ID

[HttpGet("{id}")]

public ActionResult<Item> GetItemById(int id) // <-- Added this method

{

var item = items.FirstOrDefault(i => i.Id == id);

if (item == null)

{

return NotFound();

}

return item;

}

[HttpPost]

public ActionResult<Item> PostItem(Item item)

{

if (!ModelState.IsValid) // <-- Added validation check

{

return BadRequest(ModelState);

}

items.Add(item);

return CreatedAtAction(nameof(GetItemById), new { id = item.Id }, item); // <-- Updated to reference GetItemById

}

[HttpPut("{id}")]

public IActionResult PutItem(int id, Item item)

{

var existingItem = items.FirstOrDefault(i => i.Id == id);

if (existingItem == null)

{

return NotFound();

}

// Update all relevant properties of the existing item

existingItem.Name = item.Name;

// Update other properties as needed // <-- Suggest updating other properties if needed

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult DeleteItem(int id)

{

var item = items.FirstOrDefault(i => i.Id == id);

if (item == null)

{

return NotFound();

}

items.Remove(item);

return NoContent();

}

}

}