24/8/24

ASSIGNMENT 2

Unit-2 Web Application Development Basics

Q.1 Discuss how C# reduces duplication in using statement with appropriate example.

—4 C# provides a feature called the "using declaration" that reduces code duplication when working with resources that need to be disposed of after use. This feature was introduced in C# 8.0 and simplifies the traditional 'using' statement, allowing you to declare and dispose of resources in a more concise way.

- Traditional 'using' statement :-

It was commonly used to ensure that resources like file handles, database connections or streams were properly disposed of after use.

eg:
```
using (var stream = new FileStream ("example.txt", FileMode.Open)) {
    using (var reader = new StreamReader (stream)) {
        string content = reader.ReadToEnd();
        Console.WriteLine(content);
    }
}
```

- Using Declaration (C# 8.0 and later):-

It simplifies the syntax and reduces duplication, enhances readability and maintains resource safety with less boilerplate code.

eg:
```
// Multiple namespaces in one using statement
using System;
using System.IO;
// using directive aliasing
using Project = MyApp.ProjectNamespace;
// Using declaration (C# 8.0+)
using var file = new StreamReader("example.txt");
```

* How it reduces duplication :-

i) It automatically disposes of the resource when the containing block (like method) ends, without needing 'Σ'.
ii) Eliminates nested blocks, making the code simpler and more readable.
iii) Each resource is declared and disposed of in one line.

Q.2 What is null state analysis? Explain nullable types with example.

→ Null State Analysis is a mechanism in C# that helps developers identify potential 'null' reference errors at compile time. This feature is part of C#'s nullable reference types (introduced in C# 8.0). The compiler uses null state analysis to track whether a variable may be 'null' at various points in the code, thereby helping to prevent runtime exceptions like 'NullReferenceExceptions'.

Also in nullable-enabled contexts, reference types are treated as non-nullable by default. This means you cannot assign 'null' to them unless they are explicitly marked as nullable.

→ Nullable Types :-

1. Nullable Reference Types
   It allows reference types like 'string', 'object' to have 'null' values, indicated by appending a '?' to the type.

2. Nullable Value Types.
   It allows value types like 'int', 'bool', 'DateTime' to have 'null' values, using the '?' syntax, which is particularly useful when dealing with databases or other data sources that may have missing values.

eg:
```
string? name = null;
if (name != null) {
    console.WriteLine(name.Length);
}
int? age = null;
if (age.HasValue) {
    console.WriteLine(age.Value);
}
```

Q.3 Create ASP.NET Core Controller and demonstrate use of null conditional operator and null-coalescing operator.

→ A Controller handles HTTP requests and produces HTTP responses. ~~Let's~~ ☺

eg:

```
using Microsoft.AspNetCore.Mvc;
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase {
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id) {
        Product? product = GetProductById(id);
        // use null conditional
        string? productName = product?.Name;
        // use null-coalescing
        string displayName = productName ?? "Product
                                Not Found";
        return Ok(new {Name = displayName});
    }
    private Product? GetProductById(int id) {
        return id == 1 ? new Product {Name =
                        "laptop"} : null;
    }
}
// Example Product Class
public class Product {
    public string? Name {get; set;}
}
```

- Null Conditional : Safely accesses the "Name" property of 'product' without throwing an exception if 'product' is null.

- Null - coalescing ('??') : Provides a fallback value "Product not found" if 'productName' is null.

Q.4 Explain string interpolation and collection initializer.

-> String Interpolation allows you to embed expressions directly within string literals, making it easier to create formatted strings. It was introduced in C# 6.0 and provides a more readable and concise way to format strings compared to older approaches like 'String.Format'.

To use string interpolation, you prefix a string literal with the '$' symbol and write the expressions within curly braces '{}', inside the string.

eg:
```
string name = "Tony";
int age = 25;
string greeting = $"Hello, my name is {name} and I am {age} years old";
Console.WriteLine(greeting);

// Output : Hello, my name is Tony and I am 25 years old.
```

-> Collection Initializer allows you to create and populate collections like list, arrays, dictionaries, in a concise and readable way at the time of their creation. (introduced in C# 3.0).

A collection initializer uses curly braces '{}' to specify the elements that should be added to the collection.

eg:
```
List<new> num = new List<int> {1, 2, 3, 4, 5};
Dictionary <string, int> ages = new Dictionary
                                <string, int>
{
    {'Tony', 25},
    {'Pepper', 23}
};

// The collection will automatically be populated
  with these elements.
```

Q.5 Discuss pattern matching with appropriate code snippet.

-> Pattern Matching in C# (7.0) allows for more expressive and concise code by checking a value against a pattern, extracting data from it and making decisions based on its type or structure. It is commonly used with 'switch' expressions, ~~'is' expressions~~ and 'if' statements. Basically 'is' keyword is primarily used for pattern matching.

-> Types of Pattern Matching :-

1. Type Pattern : Matches an expression against a specific type.
2. Constant Pattern : Matches an expression against a constant value.
3. Relational Pattern : Matches against a relational expression (eg: x > 5).
4. Logical Pattern : Combines patterns using logical expression (and, or, not).
5. Property Pattern : Matches against a property of an object.
6. Positional Pattern : Matches against the elements of a tuple or array.
7. Switch Expressions : Uses pattern matching within switch statements.

eg:
```
object obj = "Hello, World";
if (obj is string s && s.Length > 5) {
    Console.WriteLine($"Length : {s}");
}
int number = 45;
string description = number switch {
    < 0 => "Negative Number",
      0 => "zero",
    > 0 => "Positive Number",
      _ => "unknown"
};
Console.WriteLine(description);

// Output : Length : Hello, World
            Positive Number.
```

Q.6 Create C# model class and explain use of extension method.

—↳ Creating a C# Model class 'Product'.

```
public class Product {
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public Product (int id, string name, decimal
                                        price) {
        Id = id;
        Name = name;
        Price = price;
    }
}
```

—↳ An Extension Method is a kind of static method that allows you to add new functionality to existing types without modifying their source code or creating a new derived type. They are defined in static classes and can be called as if they were instance methods on the extended type.

eg: Extension Method for 'Product' class

```
public static class ProductExtensions {
    public static decimal ApplyDiscount (this
        Product product, decimal percentage) {
        return product.Price = (product.Price *
                                (percentage / 100));
    }
}
```

—4 Using the Extension Method

```
class Program {
    static void Main(string[] args) {
        Product product = new Product (1, "Laptop", 1000m);
        decimal discountedPrice = product.ApplyDiscount(10);
        Console.WriteLine($"Original Price : {product.Price:C}");
        Console.WriteLine($"Discounted Price : {discountedPrice:C}");
    }
}
```

// Output : Original Price : $1,000.00
Discounted Price : $900.00

Extension methods allows you to add new functionality to existing types without modifying their source code. They improve code readability, promote reuse and encapsulates common functionality, enhancing the flexibility and maintainability of C# code.

Q.7 Develop C# class and explain "async", "await" with appropriate example.

—4 The 'async' and 'await' keywords are used to write asynchronous code, which allows you to perform time-consuming operations like I/O-bound tasks, network requests or

database queries, without blocking the main thread. This leads to more responsive applications, especially in environments like desktop apps, web servers or mobile apps.

eg:
```
using System;
using System.Threading.Tasks;
public class DataFetcher {
    public async Task<String> FetchDataAsync () {
        Console.WriteLine ("Fetching Data ...");
        await Task.Delay (2000);
        return "Data Fetched";
    }
}
class Program {
    static async Task Main (string [] args) {
        var fetcher = new DataFetcher();
        string result = await fetcher.FetchData
                                    Async();
        Console.WriteLine (result);
    }
}

// Output : Fetching data ...
            (after 2 seconds)
            Data Fetched.
```

> 'FetchDataAsync' Method : An async method that simulates a data-fetching operation using 'Task.Delay' to mimic a delay.

> 'await' keyword : used to pause the 'Main' method execution until 'FetchDataAsync' completes, without blocking the thread.

They enhance responsiveness by keeping the UI responsive during long-running tasks, improve scalability by handling more concurrent requests on web servers and simplify writing and reading asynchronous code.

Q.8 write and explain sample HTTP Response status line and at least four headers.

-> When a server responds to an HTTP request, it sends an HTTP response message back to the client. The HTT response consists of several components, including the status line and headers.

eg:

```
HTTP/1.1 200 OK   // Status line
Content-Type : text/html ; Charset = UTF-8
Content-Length : 138
Server : Apache/2.4.1 (Unix)
Date : Tue , 27 Aug 2024 14:38:00 GMT
<html>
    <body>
        <h1> Hello, World! </h1>
    </body>
</html>
```

Headers

1. Status line.

The status line is the first line of the HTTP response and it contains three parts:

- HTTP version: 'HTTP/1.1' indicates the HTTP version used.
- Status Code: '200' is the status code representing a successful ~~co~~ request. (always 3 digits).
- Reason Phrase: 'OK' is a human-readable description of the status code.

2. Headers

i) 'Content-Type': Indicates the media type of the response content, specifying the type of data being returned.

- 'text/html': The media type indicates that the content is HTML.
- 'charset=UTF-8': Specifes character encoding used to encode the content.

ii) 'content-length': Indicates the size of the response body in bytes. This is important for the client to know how much data ~~it~~ to expect.

- '138': The length of response body is 138 bytes.

iii) 'Server': Provides information about the

software used by the server to handle the request.

> 'Apache/2.4.1' : Server is running Apache version 2.4.1.

> 'Unix' : Server is running on a Unix-based operating system.

iv) 'Date' : Contains the date and time at which the server generated the response. It is provided in GMT (Greenwich Mean Time).

> 'Tue, 27 Aug 2024 14:38:00 GMT' : Response was generated on this date and time.

Q.9 Define Cookie. List and explain types of cookies with the use of each type.

—> A cookie is a small piece of data stored on the user's device by a web browser while browsing a website. Cookies are used to remember information about the user, such as their preferences, login status or tracking information across multiple visits to the website.

—> Types of Cookies are :-

1. Session Cookies.
   They are temporary cookies that

are created and stored in the browser's memory during a user's session on a website. They are deleted automatically when the user closes the browser.

It is used to maintain user login status, ~~ten~~ store temporary shopping cart items and track user interactions within a web application.

eg: Session ID = abc123 ; Path = / ; HttpOnly

2. Persistent Cookies

They are also known as permanent or stored cookies. They remain on the user's device for a specified period or until they are deleted by the user. They have an expiration date, after which the browser automatically deletes them.

It is used to remember user preferences (like language or theme), track user behavior over time and implement user authentication by remembering login credentials.

eg: User Preferences = theme = dark ; Expires = Fri, 30 Aug 2024 12:00:00 GMT ; Path = /

★ There are other cookies like First-Party, Third-Party, Secure and HttpOnly Cookies.

Q.10 Develop ASP.NET core controller and demonstrate writing cookie and reading cookie.

—ᵠ In ASP.NET Core, cookies ~~are~~ can be managed using the 'HttpContext' object.

eg:
```
using Microsoft.AspNetCore.Mvc;
[ApiController]
[Route("api/[controller]")]
public class CookieController : ControllerBase {
    //Write a Cookie
    [HttpGet("write")]
    public IActionResult WriteCookie() {
        Response.Cookies.Append("UserPreference",
        "DarkMode", new CookieOptions {
            Expires = DateTime.Now.AddDays(7)
        });
        return OK("Cookie written");
    }
    //Read a Cookie
    [HttpGet("read")]
    public IActionResult ReadCookie() {
        string? cookieValue = Request.Cookies
                              ["UserPreference"];
        if (cookieValue != null)
            return OK($"Cookie Value: {cookieValue}");
        else
            return NotFound("Cookie Not Found");
    }
}
```

- 'writeCookie' Method, creates a cookie named "UserPreference" with the value "DarkMode" and sets the cookie to expire in 7 days using "CookieOptions".

- 'ReadCookie' Method, reads the value of the "UserPreference" cookie and returns the cookie if found or a "Not Found" response if not.

Q.11 Write code snippet having CookieOptions class and explain any four properties of CookieOptions class.

→ The 'CookieOptions' class allows you to configure various options when setting cookies.

eg:
```
using Microsoft.AspNetCore.Mvc;
[ApiController]
[Route("api/[controller]")]
public class CookieController : ControllerBase {
    [HttpGet("write")]
    public IActionResult WriteCookie() {
        var cookieOptions = new CookieOptions {
            Expires = DateTime.Now.AddDays(7),
            HttpOnly = True,
            Secure = True,
            SameSite = SameSiteMode.Strict
        };
        Response.Cookies.Append("UserPreference",
        "DarkMode", cookieOptions);
        return OK("Cookie Written");
    }
}
```

i) 'Expires': Sets the expiration date and time for the cookie. If not set, the cookie becomes a session cookie.

Here, the cookie ~~will~~ is set to expire in 7 days from the current date and time.

This property is used when you want the cookie to persist beyond the current session, such as remembering user preferences across visits.

ii) 'HttpOnly': When set to 'True', it restricts access to the cookie from client-side scripts, enhancing security by preventing cross-site scripting (XSS) attacks.

Here, it is set to 'true' so it ensures that the cookie cannot be accessed via Javascript.

It is used for cookies that contains sensitive information like authentication tokens.

iii) 'Secure': Ensures the cookie is sent to the server only over HTTPS, which protects it from being intercepted in transit.

Here, it is set to 'true' so it ensures that the cookie is only transmitted over HTTPS.

Used for cookies that contain sensitive information, especially in

production environments where HTTPS is used.

iv) 'SameSite' : Controls whether the cookie is sent with cross-site requests. The 'SameSiteMode.Strict' value restricts the cookie to only same-site requests, providing protection against cross-site request forgery (CSRF) attacks.

Here, 'Strict' is used which ensures that the cookie is only sent with requests that originates from the same domain.

Used 'Strict' for cookies that should not be sent with cross-site requests, helping to protect against CSRF attacks.

Q.12 Write steps to use session in ASP.NET core MVC Project. Discuss necessary code snippet whenever required. -4

Sessions are used to store data that is needed across multiple requests.

Step-1: Install Necessary NuGet Package.

- dotnet add package Microsoft.AspNetCore.Session

If you are using ASP.NET Core 2.X or later, the session middleware is included by default.

Step-2: Configure Session in 'Startup.cs' file.

★ Default 'Startup.cs' file contains ConfigureServices and Configure (also Startup) class.

i) Add session services

```
public void ConfigureServices(IServiceCollection
                                services) {
    services.AddControllersWithViews();
    // Add session services
    services.AddSession(options => {
        options.IdleTimeout = TimeSpan.FromMinutes(30);
        options.Cookie.HttpOnly = true;
        options.Cookie.IsEssential = true;
    });
}
```

ii) Enable session Middleware.

```
public void Configure(IApplicationBuilder app,
                      IWebHostEnvironment env) {
    * // Here, it contains the predefined
      default code of this class.
    // use session Middleware
    app.UseSession();
}
```

Step-3: ~~Store Data in Session~~ use session in a controller

* All the below codes will be written in a single controller file.

i) Store data in session

```
public IActionResult SetSession() {
    HttpContext.Session.SetString("Username",
                                  "Tony Mikaelsen");
    HttpContext.Session.SetInt32("UserAge", 25);
    return OK("Session Values Set");
}
```

ii) Retrieve Data from Session.

```
public IActionResult GetSession() {
    String username = HttpContext.Session.GetString
                                    ("Username");
    int? userAge = HttpContext.Session.GetInt32("userAge");
    ViewBag.Username = username;
    ViewBag.UserAge = userAge;
    return View();
}
```

Extra: iii) Remove Data from Session.

```
public IActionResult LogOut() {
    HttpContext.Session.Remove("Username");
    HttpContext.Session.Clear();
    return RedirectToAction("Index");
}
```

Step-4: Use Session Data in Views

```
@{
    ViewBag.Title = "GetSession";
}
<h2> About </h2>
<p> Username : @ViewBag.Username </p>
<p> User Age : @ViewBag.UserAge </p>
```

xxxxxxxxxx