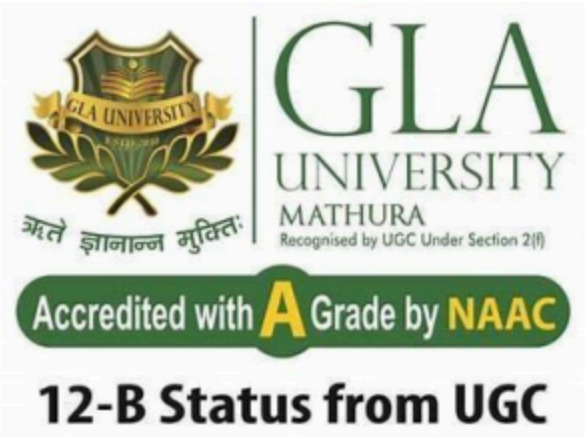
.Net Framework using C# LAB

MCAE0471

MCA 2nd Year



Session:-2025-26

Submitted to:-

Mr. Sachendra Singh Chauhan

Assistant Professor

Dept. CEA

Submitted by:-

Tinku Sharma

Roll No. – 4 0

Uni. Roll No. – 2 4 8 4 2 0 0 1 9 7

Assignment 2 - .NET Framework (C#) Solutions

1. Key differences between Windows services and Web services in C#

Windows Service:
- Runs on Windows as a background process (no UI).
- Typically used for long-running tasks, scheduled jobs, background processing.
- Installed on the machine (Service Control Manager) and controlled via services.msc, sc.exe, or PowerShell.

Web Service (SOAP/REST):
- Exposes functionality over HTTP(S) so remote clients can call it.
- Can be hosted in IIS, Kestrel, or other web hosts.
- Designed for interoperability between systems (SOAP uses WSDL; REST uses HTTP/JSON).

Summary differences:
- Purpose: Windows Service = background local processing; Web Service = remote API.
- Hosting: Windows Service = Service Control Manager; Web Service = Web host/IIS.
- Communication: Windows Service usually does not directly expose HTTP endpoints (unless specifically implemented); Web Service does.

2. Windows service explanation and OnStart / OnStop methods

A Windows Service is a class derived from System.ServiceProcess.ServiceBase. The OS starts and stops it via the Service Control Manager.

OnStart:
- Called by the Service Control Manager when the service is started.
- Should perform initialization and start worker threads/timers but must return quickly.

OnStop:
- Called when the service is stopped.
- Should stop timers/threads and perform cleanup. Also must return in a timely manner.

Other useful methods: OnPause, OnContinue, OnShutdown for other service lifecycle events.

3. Basic Windows service that logs date/time to a text file and installation steps

Sample Windows Service (C#). This example logs current date/time every minute to a text file.

File: TimeLoggerService.cs

```
using System;
using System.IO;
using System.ServiceProcess;
```

```
using System.Timers;

public class TimeLoggerService : ServiceBase
{
    private Timer _timer;
    private readonly string _logPath =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "timelog.txt");

    public TimeLoggerService()
    {
        this.ServiceName = "TimeLoggerService";
    }

    protected override void OnStart(string[] args)
    {
        _timer = new Timer(60000); // 60,000 ms = 1 minute
        _timer.Elapsed += Timer_Elapsed;
        _timer.AutoReset = true;
        _timer.Start();
        Log("Service started.");
    }

    private void Timer_Elapsed(object sender, ElapsedEventArgs e)
    {
        Log($"Current time: {DateTime.Now:yyyy-MM-dd HH:mm:ss}");
    }

    protected override void OnStop()
    {
        _timer?.Stop();
        _timer?.Dispose();
        Log("Service stopped.");
    }

    private void Log(string message)
    {
        try
        {
            File.AppendAllText(_logPath, $"{DateTime.Now:yyyy-MM-dd HH:mm:ss} -
{message}{Environment.NewLine}");
        }
        catch { /* avoid throwing in service */ }
    }

    // Main entry to run as console for debugging:
    public static void Main(string[] args)
    {
        #if DEBUG
        var svc = new TimeLoggerService();
        svc.OnStart(args);
        Console.WriteLine("Service running -- press Enter to stop (debug mode)...");
        Console.ReadLine();
        svc.OnStop();
```

```
        #else
        ServiceBase.Run(new TimeLoggerService());
        #endif
    }
}
```

Installation steps (development/debugging):
1. Build the service project as an EXE.
2. For testing you can run the EXE in DEBUG mode (see Main above which supports console run).

Install as Windows Service (production):
- Using sc.exe (requires admin):

```
sc create TimeLoggerService binPath= "C:\Path\To\Your\Service.exe" start= auto
sc start TimeLoggerService
sc stop TimeLoggerService
sc delete TimeLoggerService
```

- Or use PowerShell's New-Service:

```
New-Service -Name "TimeLoggerService" -BinaryPathName "C:\Path\To\Your\Service.exe" -
StartupType Automatic
Start-Service TimeLoggerService
```

- Older .NET installers: installutil.exe (from Developer Command Prompt):

```
InstallUtil.exe "C:\Path\To\Your\Service.exe"
InstallUtil.exe /u "C:\Path\To\Your\Service.exe"    // to uninstall
```

Note: Running/creating services requires Administrator privileges. For .NET Core/5+ consider using worker services and hosting as Windows service with Microsoft.Extensions.Hosting.

4. Design a website to consume a SOAP-based Web service (example: currency conversion)

High-level steps:
1. Obtain the SOAP service WSDL (e.g., http://example.com/CurrencyService?wsdl).
2. In Visual Studio: Add > Service Reference (or Connected Service). This will generate a client proxy.
3. In an ASP.NET Web Forms or MVC page, call the generated client and display results in a TextBox.

Simple example (ASP.NET Web Forms code-behind) assuming a generated proxy named CurrencyServiceClient:

```
// Default.aspx.cs
using System;
public partial class _Default : System.Web.UI.Page
{
    protected void btnConvert_Click(object sender, EventArgs e)
    {
        var client = new CurrencyServiceReference.CurrencyServiceClient();
        decimal amount = decimal.Parse(txtAmount.Text);
        string from = txtFrom.Text;
        string to = txtTo.Text;
        decimal result = client.ConvertCurrency(amount, from, to); // method provided
by SOAP service
        txtResult.Text = result.ToString("F2");
        client.Close();
    }
}
```

If you can't use SOAP or want a lightweight approach, call a SOAP endpoint via HttpClient by constructing SOAP XML envelopes and posting to the service endpoint with proper SOAPAction headers. For modern services prefer REST/JSON if available.

5. What are attributes in C#? Purpose and example of a custom attribute

Attributes are metadata applied to assemblies, types, members, parameters, etc. They provide additional information that can be read at runtime via reflection or used by the compiler/tools.
Common attributes: [Obsolete], [Serializable], [DataContract], [WebMethod], etc.

Example custom attribute:

```
using System;

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, Inherited = false,
AllowMultiple = false)]
public sealed class ExampleAttribute : Attribute
{
    public string Info { get; }

    public ExampleAttribute(string info)
    {
        Info = info;
    }
}

// Usage:
[Example("This class demonstrates a custom attribute.")]
public class MyClass { }
```

6. Use of the Obsolete attribute and its compilation impact

[Obsolete] marks program elements as outdated. It can be applied as:
[Obsolete] // shows compiler warning

[Obsolete("message")] // shows compiler warning with message
[Obsolete("message", true)] // treats usage as error (compilation fails)

Impact: If the second parameter (error) is true, usage causes a compile-time error; otherwise the compiler emits a warning with the provided message.

```
[Obsolete("Use NewMethod instead", false)]
public void OldMethod() { }

[Obsolete("Removed entirely", true)]
public void RemovedMethod() { }
```

7. Program to create a custom attribute Author (Name, Version)

Attribute definition and usage example:

```
using System;

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple =
false)]
public class AuthorAttribute : Attribute
{
    public string Name { get; }
    public string Version { get; set; }

    public AuthorAttribute(string name)
    {
        Name = name;
    }
}

// Usage:
[Author("Aryansh Chaudhary", Version = "1.0")]
public class SampleClass
{
    [Author("Aryansh", Version = "1.0")]
    public void SampleMethod() { }
}
```

8. What is an assembly? Private vs Shared assemblies

Assembly: the deployable unit of .NET (DLL or EXE) containing compiled IL, metadata, manifest, and resources.

Private assembly:
- Placed in the application's folder and used only by that application.

Shared assembly (strong-named, placed in GAC):
- Installed in the Global Assembly Cache (GAC) and can be shared by multiple apps.
- Requires a strong name (public/private key pair) and versioning.

9. Structure of an assembly; role of manifest and metadata

An assembly contains:
- Manifest: metadata about the assembly (identity: name, version, culture, public key), list of files, referenced assemblies, required permissions.
- Metadata: type/member definitions, attributes, custom attributes (used by reflection).
- Intermediate Language (IL) code and resources (images, strings).

Manifest tells the runtime how to bind the assembly, resolve references, and enforce version policy. Metadata describes types and members so tools and the runtime can work with them.

10. Steps to create a shared assembly and register in GAC

1. Create the class library project and compile to DLL.
2. Strong-name the assembly:
- Generate a key pair: sn -k KeyFile.snk
- Add the key to AssemblyInfo (or <AssemblyKeyFile>) or use [assembly: AssemblyKeyFile("KeyFile.snk")].
3. Sign the assembly and build.
4. Install into GAC:
- Using gacutil (developer tools): gacutil -i YourAssembly.dll

On newer systems, gacutil is not recommended for production; use installers or the 'Global Assembly Cache' tool in older SDKs. The GAC is primarily used by .NET Framework (not .NET Core/5+).

Note: On modern .NET (Core/5+), GAC is not used; prefer NuGet packages or shared frameworks.

11. Create a class library and consume it in a console app (steps & sample)

Steps:
1. In Visual Studio: New Project -> Class Library (.NET Framework or .NET Standard) -> implement classes.
2. Build project to produce MyLibrary.dll.
3. In Console App project: Add Reference -> Projects -> select the class library (or browse to DLL).
4. Use the namespace and call methods.

Example library and console usage:

```
// MyLibrary.cs (Class Library)
namespace MyLibrary
{
    public class Greeter
    {
        public static string SayHello(string name) => $"Hello, {name}!";
    }
```

```
}

// Program.cs (Console App)
using System;
using MyLibrary;

class Program
{
    static void Main()
    {
        Console.WriteLine(Greeter.SayHello("Aryansh"));
    }
}
```

12. Using ADO.NET to fetch employee records and display in a DataGridView

Assume WinForms application with a DataGridView named dataGridView1. Use SqlDataAdapter + DataSet for two-way binding and editing.

Sample code:

```
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

string connString = "Server=SERVERNAME;Database=YourDB;Trusted_Connection=True;";
string sql = "SELECT EmployeeID, Name, Department, Salary FROM Employees";

DataSet ds = new DataSet();
using (SqlConnection conn = new SqlConnection(connString))
{
    SqlDataAdapter adapter = new SqlDataAdapter(sql, conn);
    SqlCommandBuilder builder = new SqlCommandBuilder(adapter); // to enable
update/insert/delete
    adapter.Fill(ds, "Employees");

    dataGridView1.DataSource = ds.Tables["Employees"];
}

// To save changes back to DB:
adapter.Update(ds, "Employees");
```

Notes:
- Use SqlCommandBuilder only for simple scenarios; for complex updates define InsertCommand/UpdateCommand/DeleteCommand with parameters.
- Consider using BindingSource for better navigation/filtering.

13. Using DataSet and DataTable for offline functionality

Approach:
- Fill a DataSet/DataTable via SqlDataAdapter and close the DB connection. The DataSet contains data in-memory and can be used offline.

- Make edits to the DataTable while offline.
- When online, use SqlDataAdapter.Update to push changes back (requires proper Insert/Update/Delete commands with parameters and primary keys).

Benefits: disconnected architecture, less DB locking, ability to work offline and sync later.

14. Parameterized queries in ADO.NET to prevent SQL injection

Always use parameters instead of concatenating user inputs.

Example:

```
using (SqlConnection conn = new SqlConnection(connString))
using (SqlCommand cmd = conn.CreateCommand())
{
    cmd.CommandText = "SELECT EmployeeID, Name FROM Employees WHERE Department = @dept
AND Salary >= @minSalary";
    cmd.Parameters.AddWithValue("@dept", deptTextBox.Text);
    cmd.Parameters.AddWithValue("@minSalary", decimal.Parse(minSalaryTextBox.Text));
    conn.Open();
    using (SqlDataReader rdr = cmd.ExecuteReader())
    {
        // read data
    }
}
```

Note: Prefer using Add and specifying SqlDbType for better control instead of AddWithValue which can infer incorrect types.

15. Inventory system: DataGridView editing, aggregate query, when to use DataSet vs DataReader

Requirements and recommended approaches:

a) Display editable list of products and save changes:
- Use a DataSet/DataTable with SqlDataAdapter. Bind the DataTable to a DataGridView. The SqlDataAdapter (with proper Insert/Update/Delete commands) will persist changed rows back to the database via adapter.Update(ds, "Products").
Reason: DataSet is disconnected and supports editing, relationships, and change tracking which simplifies two-way editing.

b) Retrieve total number of products and combined value quickly without loading all product details:
- Use a direct aggregate SQL query executed with SqlCommand and ExecuteScalar or ExecuteReader: e.g. SELECT COUNT(*) AS TotalCount, SUM(Price * Quantity) AS TotalValue FROM Products;
Reason: Aggregation at the DB avoids transferring many rows and is much faster and memory-efficient.

c) Why DataSet vs DataReader:
- DataSet (via SqlDataAdapter): good for editable, disconnected scenarios where you need to present and modify data in UI and later sync changes.
- DataReader: forward-only, read-only, connected. Use it for fast, low-memory retrieval of large result sets or when you only need to read.

Sample aggregate code:

```
using (SqlConnection conn = new SqlConnection(connString))
using (SqlCommand cmd = new SqlCommand("SELECT COUNT(*) AS TotalCount, SUM(Price *
Quantity) AS TotalValue FROM Products", conn))
{
    conn.Open();
    using (SqlDataReader rdr = cmd.ExecuteReader())
    {
        if (rdr.Read())
        {
            int totalCount = rdr.IsDBNull(0) ? 0 : rdr.GetInt32(0);
            decimal totalValue = rdr.IsDBNull(1) ? 0 : rdr.GetDecimal(1);
        }
    }
}
```

Created by Aryansh Chaudhary

Uni. Roll No:- 2484200038

Class Roll No:- 11