# Section 1: Theoretical Questions

## ASP.NET Basics

* + 1. **Difference between ASP.NET Web Forms and ASP.NET MVC:**
       - Explain the differences between ASP.NET Web Forms and ASP.NET MVC. When would you use one over the other?

Answer

|  |  |
| --- | --- |
| ASP.NET Web Forms | ASP.NET MVC |
| Event-Driven Model | Model-View-Controller (MVC) Pattern |
| ViewState | No ViewState |
| Less Control Over HTML and JavaScript | Full Control Over HTML and JavaScript |

ASP.NET MVC is ideal for larger, more complex web applications where clean, maintainable code and high testability are priorities. It’s also preferred for applications requiring fine control over HTML, CSS, and JavaScript for a more customized UI.

## ASP.NET Core vs ASP.NET Framework:

* + - * Describe the main differences between ASP.NET Core and ASP.NET Framework. Why would you choose ASP.NET Core for new projects?

Answer

Cross-Platform support, modern development practices, improved performance and lightweight deployment. Also not to forget there were active development and community support.

## Dependency Injection in ASP.NET Core:

* + - * Explain how Dependency Injection is implemented in ASP.NET Core and its benefits.

Answer

Using extension methods to register groups of services.  
  
Benefits:

* Loosely Coupled Code: DI promotes a loosely coupled design, as dependencies are injected instead of being instantiated within a class. This leads to more flexible and modular code.
* Enhanced Testability: By injecting dependencies, it becomes easy to substitute mocks or stubs in place of actual services, making unit testing straightforward.
* Improved Maintainability: Since dependencies are defined in one central location (Startup.cs), it’s easier to modify or replace services without changing multiple files.
* Separation of Concerns: DI encourages following the Single Responsibility Principle (SRP), where each class has a clear responsibility and does not manage its dependencies.
* Lifecycle Management: ASP.NET Core’s DI container efficiently manages the lifecycle of services, ensuring efficient resource use by disposing of resources when they are no longer needed.

## C# Fundamentals

1. **Asynchronous Programming:**
   * Describe how async and await work in C#. Provide an example of their use.

Answer

The async modifier specifies that a method, lambda expression, or anonymous method is asynchronous. Async methods run synchronously until they reach their first await expression, at which point the method is suspended until the awaited task is complete.

Examples:

using System;

using System.Net.Http;

using System.Threading.Tasks;

public class Example

{

// Asynchronous method to fetch data from a URL

public async Task<string> FetchDataAsync(string url)

{

using (HttpClient client = new HttpClient())

{

// The await keyword pauses execution until GetStringAsync completes

string result = await client.GetStringAsync(url);

return result;

}

}

}

// Using the asynchronous method

public class Program

{

public static async Task Main(string[] args)

{

Example example = new Example();

Console.WriteLine("Fetching data...");

// Await the async method to get the result without blocking the main thread

string data = await example.FetchDataAsync("https://api.example.com/data");

Console.WriteLine("Data received:");

Console.WriteLine(data);

}

}

## Entity Framework Core:

* + What is Entity Framework Core, and how does it differ from Entity Framework 6? Describe its primary features and advantages.

Answer

Entity Framework Core (EF Core) is a newer, more lightweight, and extensible version of Entity Framework 6 (EF6).

EF Core is built to support different databases, such as SQL Server, PostgreSQL, SQLite, MySQL, and others, which can be used across different platforms. This cross-platform compatibility is one of the main advantages of EF Core over its predecessor, Entity Framework, which was primarily designed for Windows and SQL Server.

## Exception Handling:

* + Explain best practices for handling exceptions in ASP.NET applications.

Answer

Use Try-Catch to handle exceptions and avoid crashing the applications.

## Basic Coding

* + 1. **Reverse String Method:**
       - Write a C# method ReverseString(string input) that returns the reversed version of the input string.

Answer

using System;

public class StringManipulation

{

public static string ReverseString(string input)

{

// Check for null or empty input to avoid errors

if (string.IsNullOrEmpty(input))

{

return input;

}

// Convert string to a character array, reverse it, and then create a new string

char[] charArray = input.ToCharArray();

Array.Reverse(charArray);

return new string(charArray);

}

public static void Main()

{

string input = "Hello, World!";

string reversed = ReverseString(input);

Console.WriteLine($"Original: {input}");

Console.WriteLine($"Reversed: {reversed}");

}

}

## Fibonacci Sequence Method:

* + - * Implement a C# method Fibonacci(int n) to return the nth Fibonacci number using recursion.

Answer

using System;

public class FibonacciExample

{

public static int Fibonacci(int n)

{

// Base case: the first and second Fibonacci numbers are both 1

if (n <= 1)

{

return n;

}

// Recursive case: Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)

return Fibonacci(n - 1) + Fibonacci(n - 2);

}

public static void Main()

{

int n = 10; // Example: Get the 10th Fibonacci number

int result = Fibonacci(n);

Console.WriteLine($"The {n}th Fibonacci number is: {result}");

}

}

## ASP.NET Core MVC

1. **Create a Simple ASP.NET Core MVC Application:**
   * Develop a simple ASP.NET Core MVC application with a model, a controller, and a view. The application should display a list of items with basic CRUD operations (Create, Read, Update, Delete).

## Entity Framework Core:

* + Create a simple model and a DbContext for an ASP.NET Core application. Implement a method to add a new record to the database and retrieve all records.

# Section 3: Practical Task

## Build a RESTful API

* + 1. **Task:**
       - Create a RESTful API using ASP.NET Core Web API that performs CRUD operations on a Product entity. The Product entity should have properties such as Id, Name, Price, and Description. Include appropriate routes, and implement methods for creating, reading, updating, and deleting products.

## Debugging

* + 1. **Debugging Code Issues:**

o Given the following code snippet, identify and fix any issues:

public class ProductController : Controller

{

private readonly ApplicationDbContext \_context;

public ProductController(ApplicationDbContext context)

{

\_context = context;

}

public IActionResult Index()

{

var products = \_context.Products.ToList(); return View(products);

}

[HttpPost]

public IActionResult Create(Product product)

{

if (ModelState.IsValid)

{

\_context.Products.Add(product);

\_context.SaveChanges();

return RedirectToAction("Index");

}

return View(product);

}

}

## Optimization

* + 1. **Optimize Data Retrieval:**
       - Analyze and optimize the following ASP.NET Core code snippet to improve performance:

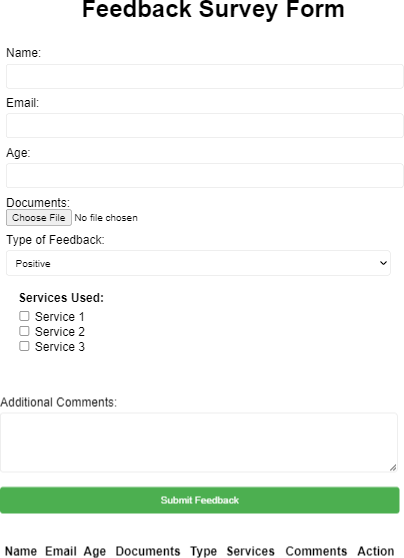
public async Task<IActionResult> GetAllProducts()

{

var products = await \_context.Products.ToListAsync(); return View(products);

}

1. Creating a simple survey form.
2. Include CRUD operation – add, update and delete function.
3. Database – Postgres / MySQL
4. Language – ASP.NET MVC



# Section 6: Submission Requirement

1. Create a public or private GitHub repository for this project.
2. Commit all code, scripts and documentation to the repository.
3. Share the repository link as part of your submission.