Madan Bhandari Memorial College

Binayak Nagar, New Baneshwor

NET CENTRIC COMPUTING (CSC 367)

LAB REPORT

Submitted By:

Utsav Acharya Sharma

Sec: B

TU Roll:24179/076

Submitted To:

Saroj Kumar Mahto

Net Centric Computing Lecturer

Madan Bhandari Memorial College

Lab 1: Date: 2080/01/04

Title: Installation of Visual Studio

Introduction:

1. Net Centric Computing:

   Net-centric computing is a computing architecture that uses networked computers and devices to provide services and applications to end-users. It involves the use of distributed computing technologies to provide scalable, reliable, and flexible computing solutions. Data and applications are stored and processed on distributed computers connected by a network, allowing users to access them from any location and device with an internet connection.

2. Visual Studio:

   Visual Studio is a powerful integrated development environment (IDE) created by Microsoft for building a wide range of software applications, including desktop applications, web applications, mobile applications, and games. It offers a comprehensive suite of tools and services for developers, including code editors, debuggers, compilers, and project management tools. Visual Studio also includes built-in support for many popular programming languages, such as C++, C#, JavaScript, and Python, as well as a variety of frameworks and libraries. With its rich set of features and powerful development tools, Visual Studio is widely regarded as one of the best IDEs for professional software development.

3. C #:

   C# is a modern, object-oriented programming language developed by Microsoft as part of its .NET framework. It was designed to be simple, efficient, and easy to learn, while still being powerful enough to build complex software applications. C# is similar in syntax to other popular programming languages like Java and C++, making it easy for developers to learn and transition between different languages. It is commonly used to develop a wide

range of applications, including desktop applications, web applications, mobile applications, and games. C# supports a variety of programming paradigms, including procedural, functional, and object-oriented programming, and offers features such as automatic memory management and garbage collection. Overall, C# is a versatile and widely-used programming language with a strong emphasis on developer productivity and software quality.

Installation of Visual Studio:

To install Visual Studio, the first step is to download it from the official Microsoft website (https://visualstudio.microsoft.com/vs). There are different versions of Visual Studio available (Community, Enterprise, Professional), but the Community version is available for free and provides a comprehensive set of tools and features for building a wide range of applications. After selecting the Community version, you can download the installation file, which is usually in the form of an executable (.exe) file, and then run it to start the installation process. During the installation process, you may need to select the specific features and components that you want to install, depending on your specific requirements. The latest version of Visual Studio Community is 2022, and it offers a range of new features and improvements to help developers build high-quality software applications. Overall, the installation process for Visual Studio is straightforward and user-friendly, allowing developers to quickly get started with their projects.

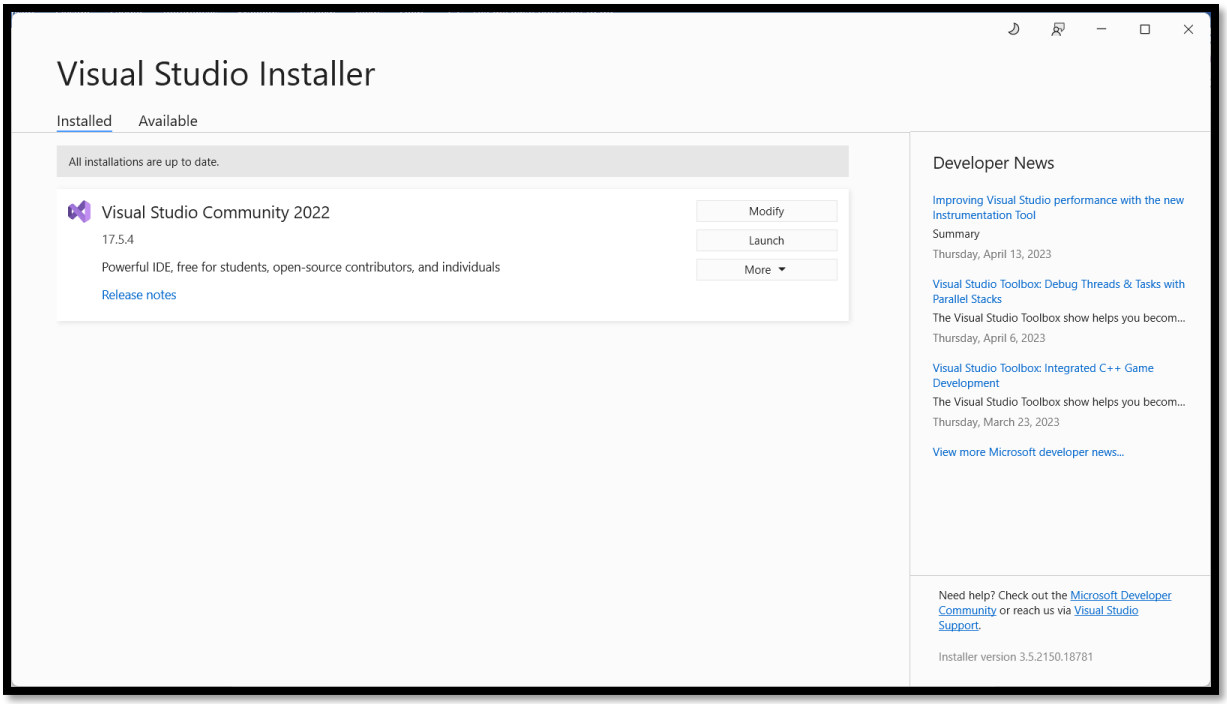


fig: installation window of visual studio 2022 (after the installation of community version)

Opening C# in Visual Studio:

To open the C# in visual studio we follow the steps mention below:

i. Open Visual Studio 2022.
ii. Create a New Project.
iii. Continue without code.
iv. Go to menu file → New → Project → Create a new project → search for templates.
v. Console application for .net framework

Console.App (.net framework)

vi. Click Next.
vii. Configure your new project

Project Name → Lab 1

Framework → .net framework 6.0

viii. Click Next.
ix. Project will be displayed and now modify.

x. To run the program → click on start (press F5).

xi. To display the message

```
Console.ReadLine();
```

xii. Save.

xiii. Start → Result will be shown on console.

Some Snapshots of the above steps

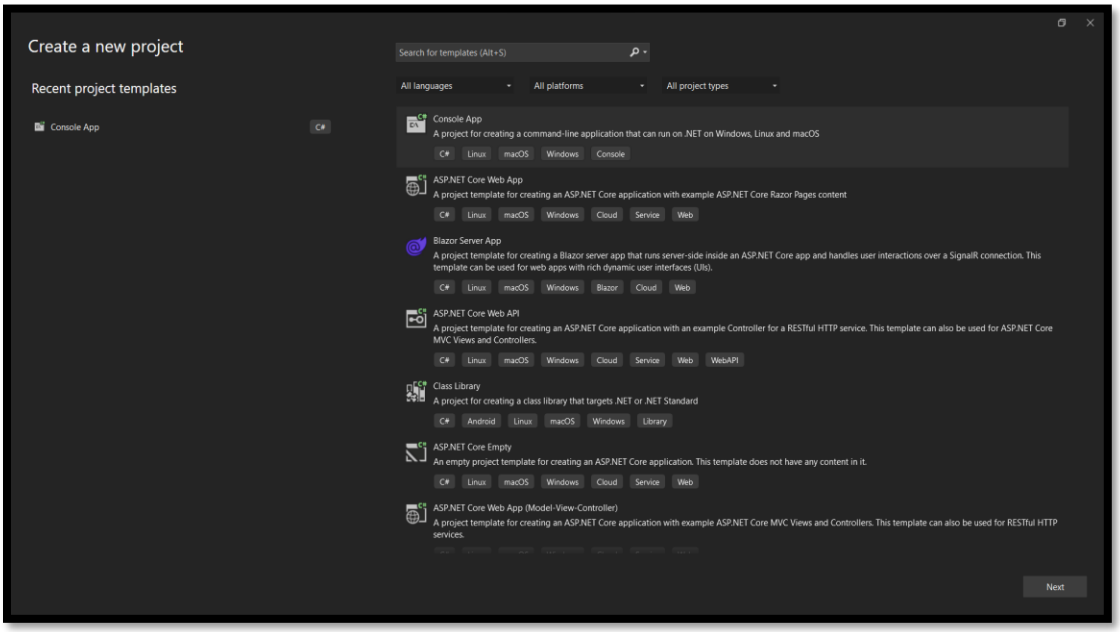


fig: selection of template.

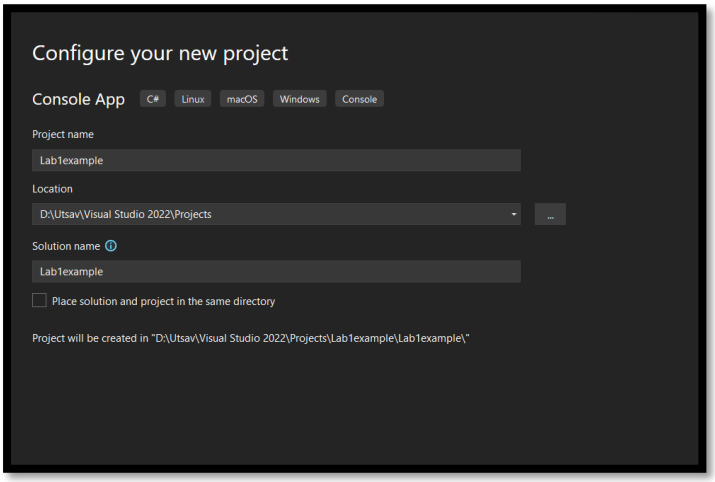


fig: configuration of project.

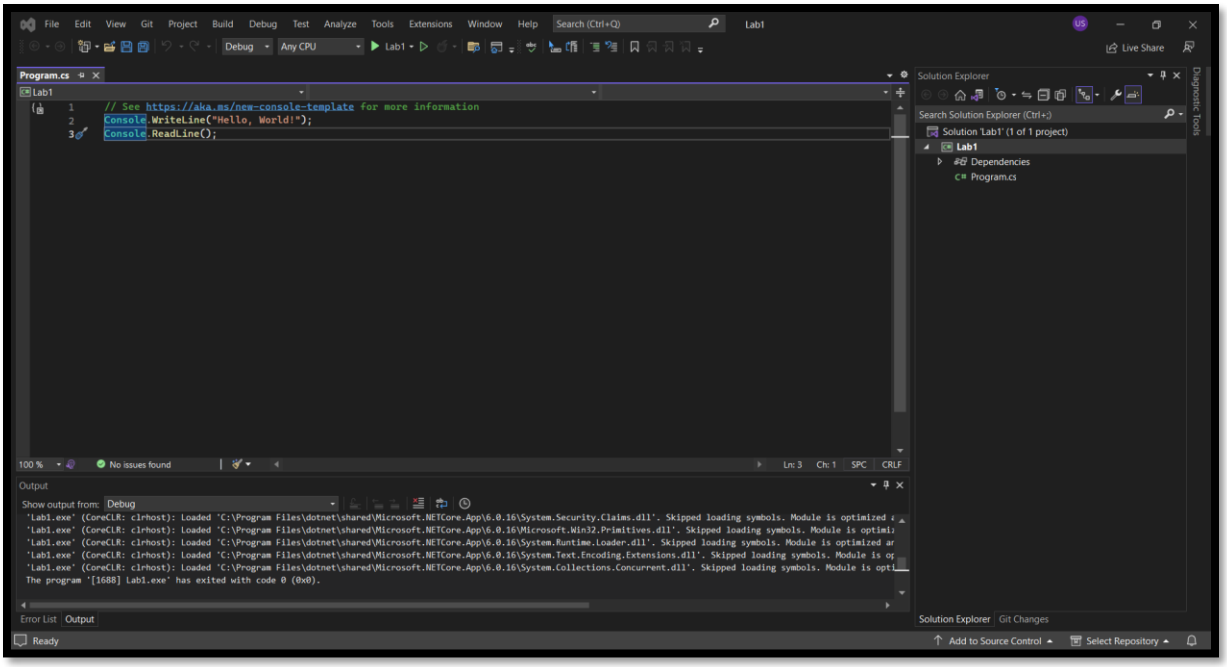


fig: snapshot of the window of visual studio after the creation of project.

fig: output of the code.

Lab 2: Date: 2080/01/05

Title: Basic C# programs

Introduction:

C# is a modern, object-oriented programming language developed by Microsoft as part of its .NET framework. It was designed to be simple, efficient, and easy to learn, while still being powerful enough to build complex software applications. C# is similar in syntax to other popular programming languages like Java and C++, making it easy for developers to learn and transition between different languages. It is commonly used to develop a wide range of applications, including desktop applications, web applications, mobile applications, and games. C# supports a variety of programming paradigms, including procedural, functional, and object-oriented programming, and offers features such as automatic memory management and garbage collection. Overall, C# is a versatile and widely-used programming language with a strong emphasis on developer productivity and software quality.

Syntax Used:

i. using System; → is a directive that indicates that this program will use the System namespace, which contains a variety of useful classes and methods for working with the .NET framework.
ii. using System.Collections.Generic; → is another directive that indicates that this program will use the System.Collections.Generic namespace, which contains a variety of collection classes that can be used to store and manipulate data.
iii. using System.Linq; → is a directive that indicates that this program will use the System.Linq namespace, which contains a set of extension methods that can be used for querying and manipulating data.
iv. using System.Text; → is a directive that indicates that this program will use the System.Text namespace, which contains classes for working with character encodings and string manipulation.

v. using System.Threading.Tasks; → is a directive that indicates that this program will use the System.Threading.Tasks namespace, which contains classes and methods for working with asynchronous programming and multithreading.
vi. namespace Lab2new_Introduction, → defines a namespace called Lab2new_Introduction that contains the program code.
vii. class Program, → defines a class called Program that contains the main method for the program.
viii. static void Main(string[] args), → is the main method of the program, which is the entry point for the program. It takes an array of string arguments as input and returns nothing.
ix. Console.WriteLine("");, → uses the Console class to print the message to the console window.
x. Console.ReadLine();, → waits for the user to press the Enter key before closing the console window.
xi. Console.WriteLine("Hello,"+name); → This line of code uses string concatenation to print the message "Hello," followed by the value of the name variable. The + operator is used to concatenate the string literal "Hello," with the value of the name variable.
xii. Console.WriteLine("Welcome {0}",name); → This line of code uses string formatting to print the message "Welcome " followed by the value of the name variable. The {0} placeholder is used to indicate where the value of the name variable should be inserted in the string. The value of the name variable is then passed as an argument to the Console.WriteLine() method, which replaces the {0} placeholder with the value of the name variable. This approach is more flexible and allows you to format the output in a variety of ways, depending on your specific requirements.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2new_Introduction
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to C#");
            Console.ReadLine();
        }
    }
}
```

Output:

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2_Introduction
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter your name:");
            string name = Console.ReadLine();
            Console.WriteLine("Hello," + name);
             //Concatination Syntax
            Console.WriteLine("Welcome {0}", name);
            Console.ReadLine();
        }
    }
}
```

Output:

Lab 3: Date: 2080/01/07

Title: Entering Multiple String

Introduction:

In C#, a string is a sequence of characters that represents text. It is a reference type, which means that it is a class and is allocated on the heap. String literals are enclosed in double quotes, and can contain any combination of characters, including letters, numbers, and symbols. For example, "Hello, world!" is a string literal.

Syntax Used:

1. Console.WriteLine → to output a message to the console.
2. Console.ReadLine → reads a line of text from the standard input stream.

Code:

```
//Entering Multiple Strings
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab3IntroductionToCsharp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter your first name:");
            String fname = Console.ReadLine();
            Console.WriteLine("Enter your last name:");
            String lname = Console.ReadLine();
            Console.WriteLine("Your name is {0}
             {1}",fname,lname);
            Console.ReadLine();
        }
    }
}
```

Output:

Lab 4: Date: 2080/01/07

Title: Passing Integer Data Type In C# Using Parse

Introduction:

In C#, the int.Parse() method is used to convert a string representation of an integer to an actual integer value. This is useful when you need to convert user input, file input, or other string data into an integer for use in calculations or other operations.

To use int.Parse(), you first read a string input using Console.ReadLine() or some other method, and then pass that string to int.Parse(). If the string can be converted to an integer, the method returns the integer value. If not, it throws an exception.

Syntax Used:

1. Console.WriteLine → to output a message to the console.
2. Console.ReadLine → reads a line of text from the standard input stream.
3. int.Parse() → to convert the string input to an integer value.

Code:

```
//Entering and passing integer data type in c# using parse.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab4IntroductionToCsharp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter first number:");
            int num1 = int.Parse(Console.ReadLine());
            Console.WriteLine("Enter second number:");
            int num2 = int.Parse(Console.ReadLine());
            Console.WriteLine("Enter third number:");
            int num3 = int.Parse(Console.ReadLine());
            Console.WriteLine("Enter fourth number:");
            int num4 = int.Parse(Console.ReadLine());
            Console.WriteLine("Enter fifth number:");
            int num5 = int.Parse(Console.ReadLine());
            int sum = num1 + num2 + num3 + num4 + num5;
            Console.WriteLine("{0} + {1} + {2} + {3} + {4} =
             {5}", num1, num2, num3, num4, num5, sum);
            Console.ReadLine();
        }
    }
}
```

Output:

```
Select Microsoft Visual Studio Debug Console
Enter first number:
5
Enter second number:
10
Enter third number:
15
Enter fourth number:
20
Enter fifth number:
25
5 + 10 + 15 + 20 + 25 = 75


D:\Utsav\Visual Studio 2022\Projects\Lab4\Lab4\bin\Debug\net6.0\Lab4.exe (process 24024) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Lab 5: Date: 2080/01/14

Title: Built In Data Types In C#

Introduction:

In C#, a data type is a classification of data that specifies the type of value that a variable can hold. C# has several built-in data types, which are used to represent different types of values, such as integers, floating-point numbers, characters, and Boolean values. These data types are built into the C# language and are provided by the .NET Framework. They are used to declare variables, parameters, and return types of methods.

Integral Type:

- Signed Integer (which takes negative and positive values)
- Unsigned Integer (which only takes positive values) e.g. age

Integer Types (Based on size):

- SBYTE
- BYTE
- SHORT
- USHORT
- INT
- UINT
- LONG
- ULONG

Methods to see range of particular Datatype:

- MINVALUE() METHOD
- MAXVALUE() METHOD

Memory Management Size:

Types	Range	Size
sbyte	-128 to 127	Signed 8-bit integer
byte	0 to 255	Unsigned 8-bit integer
char	U+0000 to U+ffff	Unicode 16-bit character
short	-32,768 to 32,767	Signed 16-bit integer
ushort	0 to 65,535	Unsigned 16-bit integer
int	-2,147,483,648 to 2,14,483,647	Signed 32-bit integer
uint	0 to 4,294,967,295	Unsigned 32-bit integer
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer

Syntax Used:

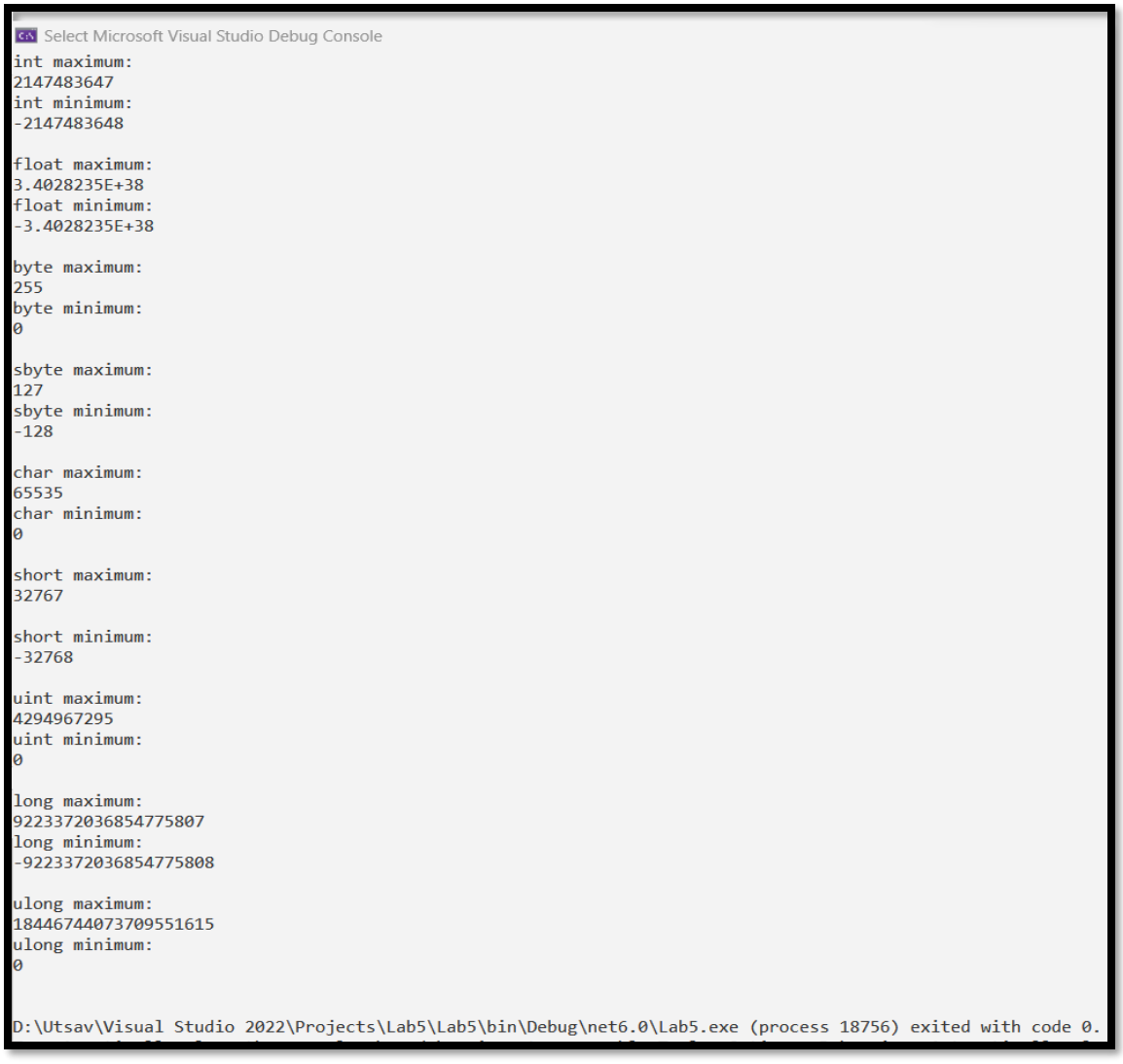
4. Console.WriteLine → to output a message to the console.
5. Console.ReadLine → reads a line of text from the standard input stream.
6. datatype.MaxValue → used to obtain maximum value of datatype.
7. datatype.MinValue → used to obtain minimum value of datatype.
8. "(int)char.MaxValue" and "(int)char.MinValue" → used to convert the char values into their corresponding integer values.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataTypeCSharP
{
    class Program
    {
        static void Main(string[] args)
        {

            //int
            Console.WriteLine("int maximum:");
            Console.WriteLine(int.MaxValue);
            Console.WriteLine("int minimum:");
            Console.WriteLine(int.MinValue);
            //float
            Console.WriteLine("\nfloat maximum:");
            Console.WriteLine(float.MaxValue);
            Console.WriteLine("float minimum:");
            Console.WriteLine(float.MinValue);
            //byte
            Console.WriteLine("\nbyte maximum:");
            Console.WriteLine(byte.MaxValue);
            Console.WriteLine("byte minimum:");
            Console.WriteLine(byte.MinValue);
            //sbyte
            Console.WriteLine("\nsbyte maximum:");
            Console.WriteLine(sbyte.MaxValue);
            Console.WriteLine("sbyte minimum:");
            Console.WriteLine(sbyte.MinValue);
            //char
            Console.WriteLine("\nchar maximum:");
            Console.WriteLine((int)char.MaxValue);
            Console.WriteLine("char minimum:");
            Console.WriteLine((int)char.MinValue);
            //short
            Console.WriteLine("\nshort maximum:");
            Console.WriteLine(short.MaxValue);
            Console.WriteLine("short minimum:");
            Console.WriteLine(short.MinValue);
            //uint
            Console.WriteLine("\nuint maximum:");
            Console.WriteLine(uint.MaxValue);
            Console.WriteLine("uint minimum:");
            Console.WriteLine(uint.MinValue);
            //long
            Console.WriteLine("\nlong maximum:");
            Console.WriteLine(long.MaxValue);
            Console.WriteLine("long minimum:");
            Console.WriteLine(long.MinValue);
```

```
            //ulong
            Console.WriteLine("\nulong maximum:");
            Console.WriteLine(ulong.MaxValue);
            Console.WriteLine("ulong minimum:");
            Console.WriteLine(ulong.MinValue);
            Console.ReadLine();
        }
    }
}
```

Output:



Lab 6: Date: 2080/01/14

Title: Boolean Data Type In C#

Introduction:

In C#, bool is a value type that represents a Boolean value, which can have only one of two possible values: true or false. The bool type is used to represent logical values and is frequently used in conditional expressions and logical operations.

A bool value can be assigned using the true or false keywords, or by the result of a comparison or logical operation.

In C#, bool values are typically used in logical operations, such as if statements, loops, and conditional expressions, to control the flow of the program based on the result of a comparison or a condition.

The bool type has a default value of false. When a bool variable is declared without initialization, it is automatically assigned the value false.

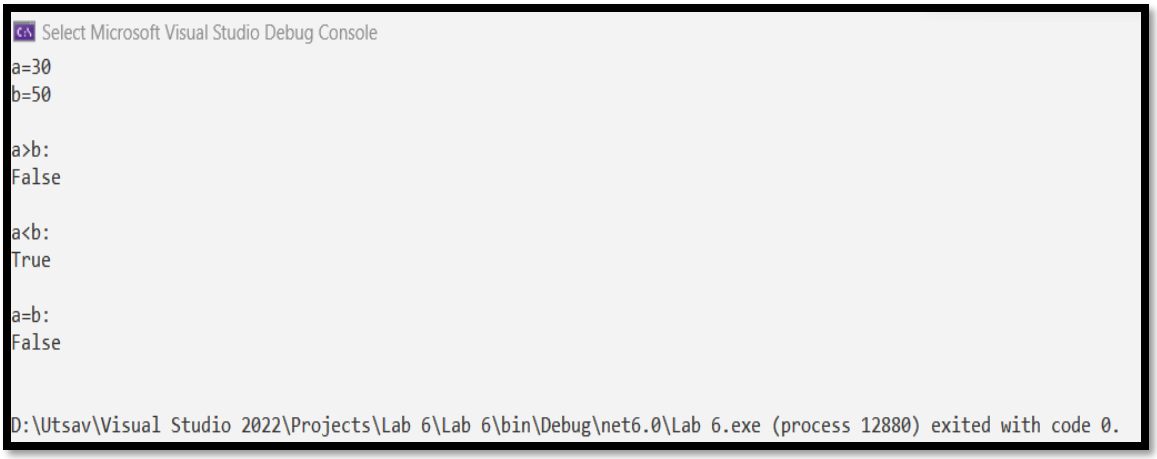
Syntax Used:

1. Console.WriteLine → to output a message to the console.
2. Console.ReadLine → reads a line of text from the standard input stream.
3. bool <variable> → stores true or false value on variable based on the condition.

Code:

```
/*Boolean DataType
 Bool Keyword is used for Boolean Data Type
 which only stored True or False.
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataTypeBoolCSharP
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 30;
            int b = 50;
            //a greater than b
            bool c = a > b;
            Console.WriteLine("a=30\nb=50\n\na>b:");
            Console.WriteLine(c);
            //a smaller than b
            bool d = a < b;
            Console.WriteLine("\na<b:");
            Console.WriteLine(d);
            //a equals to b
            bool e = a == b;
            Console.WriteLine("\na=b:");
            Console.WriteLine(e);
            Console.ReadLine();
        }
    }
}
```

Output:


Select Microsoft Visual Studio Debug Console
a=30
b=50
a>b:
False
a<b:
True
a=b:
False
D:\Utsav\Visual Studio 2022\Projects\Lab 6\Lab 6\bin\Debug\net6.0\Lab 6.exe (process 12880) exited with code 0.

Lab 7: Date: 2080/01/14

Title: Float, Double And Decimal Data Type In C#

Introduction:

1) float:

Float is a 32-bit floating-point type that is used to represent fractional numbers with up to seven significant digits. The float datatype is commonly used in scientific computations, graphics, and engineering applications where precision is important but storage space is limited. To declare a float variable, use the float keyword. f suffix is used to indicate that the literal value should be interpreted as a float.

2) double:

Double is a 64-bit floating-point type that is used to represent fractional numbers with up to 15 or 16 significant digits. The double datatype is commonly used in general-purpose applications where a higher degree of precision is required than what is provided by float. To declare a double variable, use the double keyword.

3) decimal:

Decimal is a 128-bit decimal type that is used to represent fractional numbers with up to 28 or 29 significant digits. The decimal datatype is commonly used in financial and monetary calculations where precision is critical. Unlike float and double, the decimal type provides exact decimal representation and does not suffer from rounding errors. To declare a decimal variable, use the decimal keyword. m suffix is used to indicate that the literal value should be interpreted as a decimal.

Code:

```
//Float Double and Decimal Data Type
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DataTypesFloat
{
    class Program
    {
        static void Main(string[] args)
        {
            //float
            float a = 23.36512345565769f;
            Console.WriteLine("float:{0}", a);
            //double
            double b = 567.6543123125159283471d;
            Console.WriteLine("\ndouble:{0}", b);
            //decimal
            decimal c = 3.14159265358979323846264338328m;
            Console.WriteLine("\ndecimal:{0}",c);
            Console.ReadLine();
        }
    }
}
```

Output:

```
Select Microsoft Visual Studio Debug Console
float:23.365124

double:567.6543123125159

decimal:3.14159265358979323846264338328


D:\Utsav\Visual Studio 2022\Projects\Lab 7\Lab 7\bin\Debug\net6.0\Lab 7.exe (process 10112) exited with code 0.
```

Lab 8: Date: 2080/01/19

Title: Synchronous And Asynchronous Programming

Introduction:

Synchronous Programming:

Synchronous programming is a programming model where operations take place sequentially.

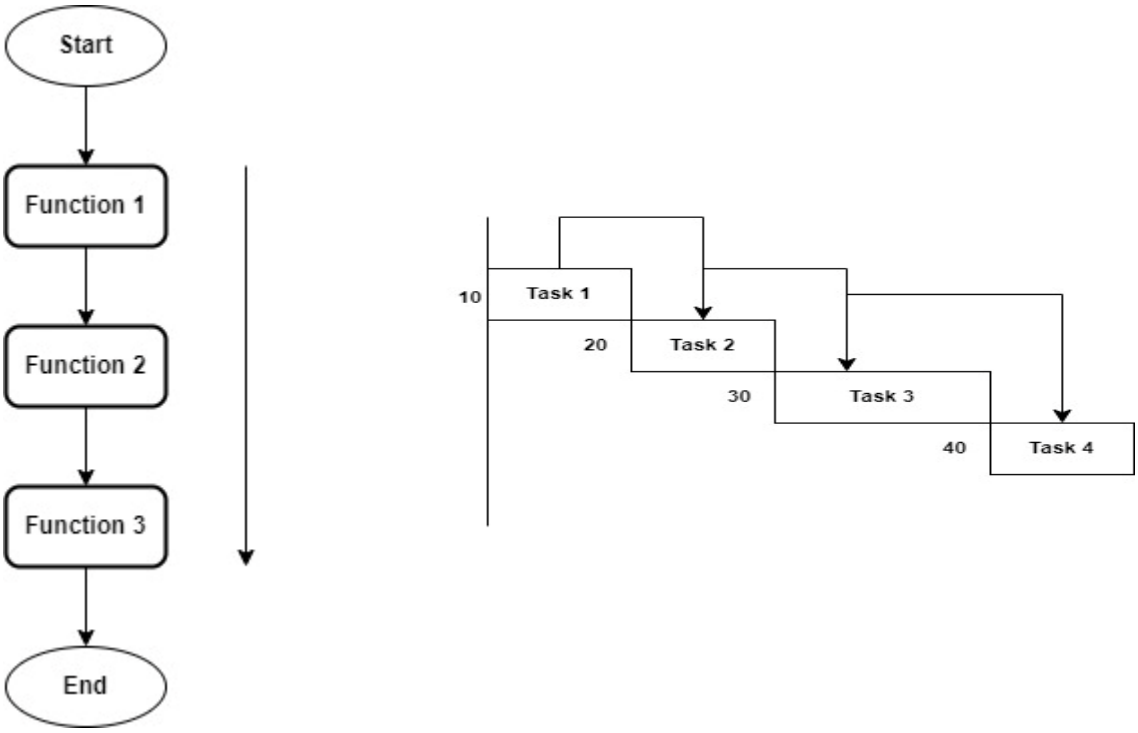
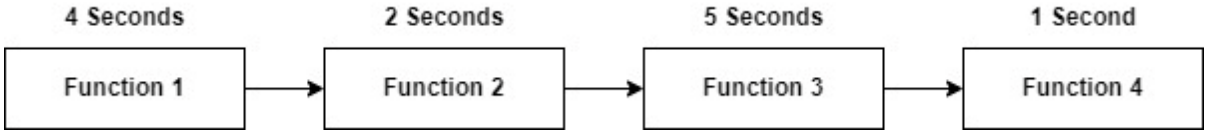


fig: Synchronous Programming



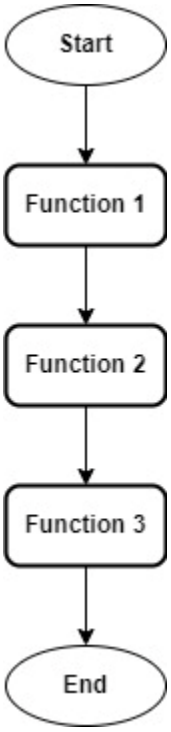
Executed in sequence.

Dependent on each other.

To overcome this delay, we use asynchronous programming model.

Asynchronous Programming:

Asynchronous programming is a programming model where operations does not depend on each other.



In this case function 1 will be executed and function 3 will be executed without waiting for function 2.

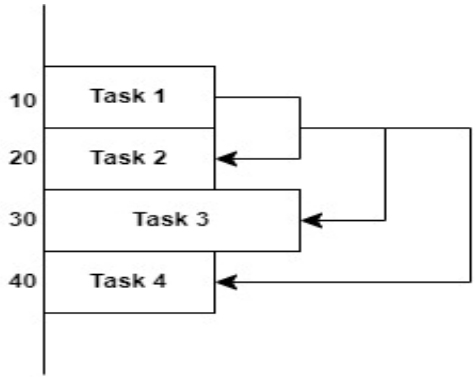
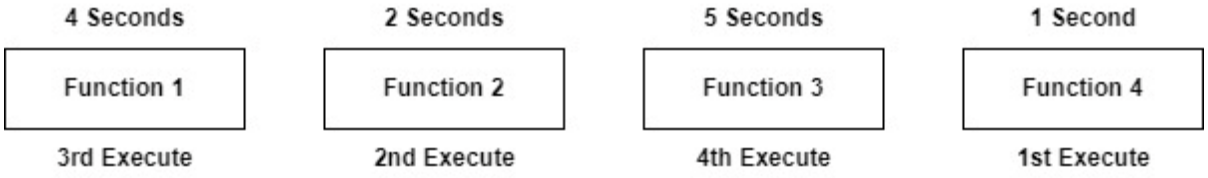


fig: Asynchronous Programming



Not executed in a sequence.

Not dependent on each other.

Syntax Used:

i. using System; → is a directive that indicates that this program will use the System namespace, which contains a variety of useful classes and methods for working with the .NET framework.
ii. using System.Collections.Generic; → is another directive that indicates that this program will use the System.Collections.Generic namespace, which contains a variety of collection classes that can be used to store and manipulate data.
iii. using System.Linq; → is a directive that indicates that this program will use the System.Linq namespace, which contains a set of extension methods that can be used for querying and manipulating data.
iv. using System.Text; → is a directive that indicates that this program will use the System.Text namespace, which contains classes for working with character encodings and string manipulation.
v. using System.Threading; → provides classes and interfaces to support multithreaded programming.
vi. using System.Threading.Tasks; → is a directive that indicates that this program will use the System.Threading.Tasks namespace, which contains classes and methods for working with asynchronous programming and multithreading.
vii. static void Main(string[] args), → is the main method of the program, which is the entry point for the program. It takes an array of string arguments as input and returns nothing.
viii. public static void task_no() → is a method declaration.
ix. public static async void task_no() → is a method declaration but it marks the method as asynchronous method.
x. Task1(), Task2(), Task3(), and Task4() → these methods simulate the execution of some tasks.
xi. The await Task.Run(() => { ... }) → used to asynchronously execute a delegate (a lambda expression) on a background thread and wait for its completion, without blocking the calling thread.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
namespace AsyncAwaitDemo
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Task1();
            Task2();
            Task3();
            Task4();
        }
        public static void Task1()
        {
            Console.WriteLine("Task 1 Starting.....");
            Thread.Sleep(3000);
            Console.WriteLine("Task 1 Completed.");
        }
        public static void Task2()
        {
            Console.WriteLine("Task 2 Starting.....");
            Thread.Sleep(1000);
            Console.WriteLine("Task 2 Completed.");
        }
        public static void Task3()
        {
            Console.WriteLine("Task 3 Starting.....");
            Thread.Sleep(4000);
            Console.WriteLine("Task 3 Completed.");
        }
        public static void Task4()
        {
            Console.WriteLine("Task 4 Starting.....");
            Thread.Sleep(2000);
            Console.WriteLine("Task 4 Completed.");
        }
    }
}
```

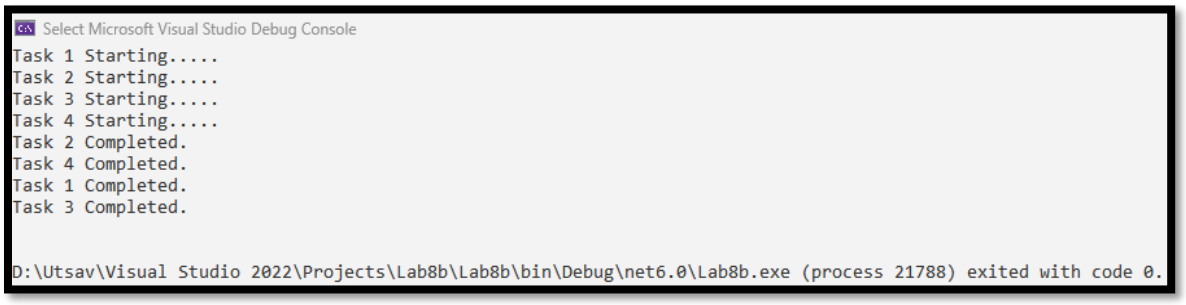
Output:


Select Microsoft Visual Studio Debug Console
Task 1 Starting.....
Task 1 Completed.
Task 2 Starting.....
Task 2 Completed.
Task 3 Starting.....
Task 3 Completed.
Task 4 Starting.....
Task 4 Completed.
D:\Utsav\Visual Studio 2022\Projects\Lab8\Lab8\bin\Debug\net6.0\Lab8.exe (process 17104) exited with code 0.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
namespace AsyncAwaitDemo
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Task1();
            Task2();
            Task3();
            Task4();
            Console.ReadLine();
        }
        public static async void Task1()
        {
            await Task.Run(() =>
            {
                Console.WriteLine("Task 1 Starting.....");
                Thread.Sleep(3000);
                Console.WriteLine("Task 1 Completed.");
            });
        }
        public static async void Task2()
        {
            await Task.Run(() =>
            {
                Console.WriteLine("Task 2 Starting.....");
                Thread.Sleep(1000);
                Console.WriteLine("Task 2 Completed.");
            });
        }
        public static async void Task3()
        {
            await Task.Run(() =>
            {
                Console.WriteLine("Task 3 Starting.....");
                Thread.Sleep(4000);
                Console.WriteLine("Task 3 Completed.");
            });
        }
        public static async void Task4()
        {
            await Task.Run(() =>
            {
                Console.WriteLine("Task 4 Starting.....");
                Thread.Sleep(2000);
                Console.WriteLine("Task 4 Completed.");
            });
        }
    }
}
```

Output:


Select Microsoft Visual Studio Debug Console
Task 1 Starting.....
Task 2 Starting.....
Task 3 Starting.....
Task 4 Starting.....
Task 2 Completed.
Task 4 Completed.
Task 1 Completed.
Task 3 Completed.
D:\Utsav\Visual Studio 2022\Projects\Lab8b\Lab8b\bin\Debug\net6.0\Lab8b.exe (process 21788) exited with code 0.

Lab 9: Date: 2080/01/20

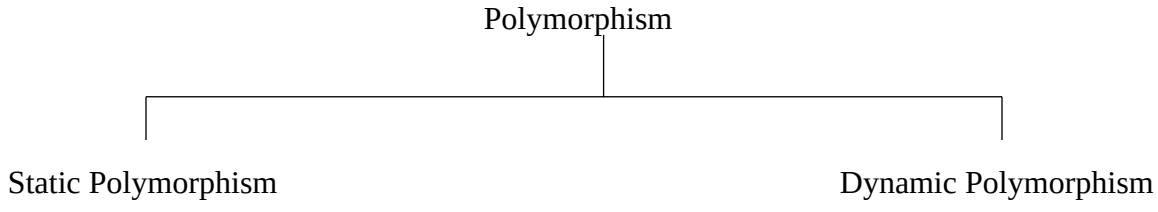
Title: Polymorphism

Introduction:

Polymorphism is a key concept in object-oriented programming (OOP) that allows developers to create code that is flexible and adaptable to a variety of scenarios. Essentially, polymorphism is the idea that a single name or method can be used in different ways depending on the context in which it is being used. This means that a single function or method can take on many different forms, depending on the input it receives or the objects it is working with.

Polymorphism is one of the four fundamental pillars of OOP, along with encapsulation, inheritance, and abstraction. These pillars are key principles that guide developers in creating effective and efficient code that is easy to maintain and update over time. Polymorphism is particularly important because it allows developers to write code that is more modular and reusable, reducing the amount of code they need to write from scratch and making their programs more efficient and effective.

There are two main types of polymorphism: static and dynamic. Static polymorphism, also known as compile-time polymorphism, occurs when the compiler determines which version of a function or method to use at compile time, based on the types of parameters passed in. Dynamic polymorphism, also known as runtime polymorphism, occurs when the correct version of a function or method is determined at runtime, based on the actual type of the object being worked with. Both types of polymorphism have their own advantages and uses, and understanding the differences between them is key to writing effective OOP code.



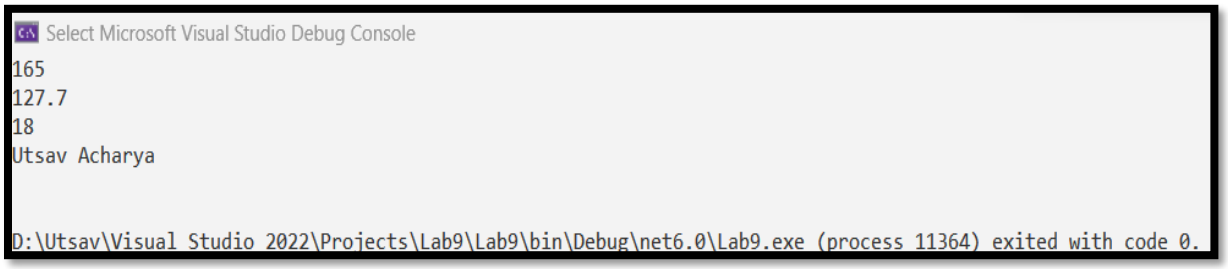
Static polymorphism, also called compile-time polymorphism, is a type of polymorphism where the type of an object is determined at compile time. In this type of polymorphism, the programmer defines multiple methods or operators with the same name but with different parameter types or number of parameters in a class. When a method or operator is called, the compiler determines which version to use based on the arguments passed in. Static polymorphism offers performance benefits over dynamic polymorphism since the method resolution is done at compile time rather than runtime. However, it is less flexible than dynamic polymorphism since the programmer must define all possible methods or operators in advance.

On the other hand, dynamic polymorphism, also known as runtime polymorphism, is a type of polymorphism where the type of an object is determined at runtime. It is achieved through inheritance and method overriding, where a subclass inherits a method from its parent class and provides its own implementation. When the method is called on an object of the subclass, the implementation of the subclass is executed instead of the parent class, allowing the subclass to customize the behavior of the inherited method. Dynamic polymorphism provides flexibility and extensibility to the code, as new subclasses can be created that inherit and override existing methods without modifying the original code. It also enables the use of abstract classes and interfaces, which define a set of methods that must be implemented by any concrete class that implements them. However, dynamic polymorphism incurs a performance overhead since the method resolution is done at runtime and involves an additional lookup step to determine the actual implementation to be called.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace PolymorphismStatic
{
    class Program
    {
        public void Add()
        {
            int a = 120;
            int b = 45;
            int c = a + b;
            Console.WriteLine(c);

        }
        public void Add(int a, int b)
        {
            int c = a + b;
            Console.WriteLine(c);


        }
        public void Add(string a, string b)
        {
            string c = a + " " + b;
            Console.WriteLine(c);
        }
        public void Add(float a, float b)
        {
            float c = a + b;
            Console.WriteLine(c);
        }
        static void Main(string[] args)
        {
            Program P = new Program();
            P.Add();
            P.Add(123f, 4.7f);
            P.Add(10, 8);
            P.Add("Utsav"," ","Acharya");
            Console.ReadLine();
        }

    }

}
```

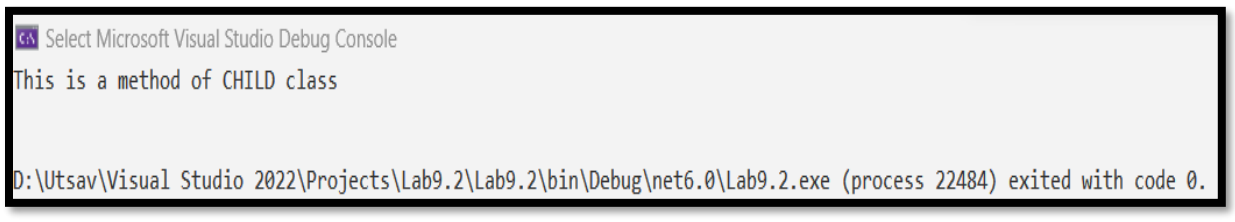
Output:


Select Microsoft Visual Studio Debug Console
165
127.7
18
Utsav Acharya
D:\Utsav\Visual Studio 2022\Projects\Lab9\Lab9\bin\Debug\net6.0\Lab9.exe (process 11364) exited with code 0.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace PlymorphismDynamicHiding
{
    class parent
    {
        public void print()
        {
            Console.WriteLine("This is a method of PARENT
             class.");
        }
    }
    class child : parent
    {
        public void print()
        {
            Console.WriteLine("This is a method of CHILD
             class");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            child c = new child();
            parent p = new child();
            c.print();
            Console.ReadLine();
        }
    }
  }
```

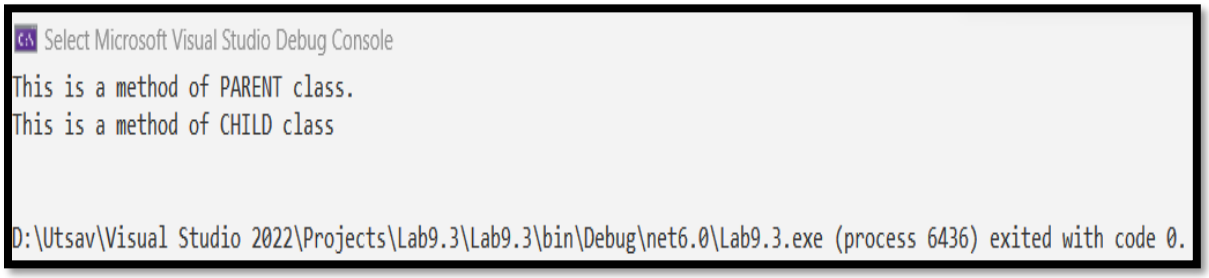
Output:



Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace PlymorphismDynamicOverloading
{
    class parent
    {
        public virtual void print()
        {
            Console.WriteLine("This is a method of PARENT
             class.");
        }
    }
    class child : parent
    {
        public override void print()
        {
            base.print();
            Console.WriteLine("This is a method of CHILD
             class");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            parent p = new child();
            p.print();
            Console.ReadLine();
        }
    }
  }
```

Output:



Lab 10: Date: 2080/01/21

Title: Lambda Expression And Lambda Statement

Introduction:

Lambda Expression:

In C#, a lambda expression is a concise and powerful way to create anonymous functions. It allows you to write code that can be passed as an argument to a method or assigned to a variable, without having to define a separate named method.

A lambda expression consists of three parts: the input parameters, the lambda operator =>, and the function body. The input parameters specify the values that the lambda expression takes as input. The function body specifies the code that the lambda expression executes, and the lambda operator separates the input parameters from the function body.

Here's an example of a lambda expression that adds two numbers:

```
(int a, int b) => a + b
```

In this lambda expression, (int a, int b) specifies the input parameters, which are two integers named a and b. The lambda operator => separates the input parameters from the function body, which is a + b.

Lambda Statement:

A lambda statement is similar to a lambda expression in C#, but it allows you to write more complex functions with multiple statements. Like a lambda expression, a lambda statement is a concise and powerful way to create anonymous functions, which can be passed as arguments to a method or assigned to a variable.

A lambda statement consists of three parts: the input parameters, the lambda operator =>, and the function body. The input parameters specify the values that the lambda statement takes as input. The lambda operator => separates the input parameters from the function body, which is a block of code enclosed in curly braces.

Syntax Used:

1. int[] numbers → declares an array of integers.
2. evenNumbers → a variable declared using the var keyword. The var keyword is used to declare a variable whose type is inferred by the compiler based on the value assigned to it.
3. var evenNumbers = Array.FindAll(numbers, n => n % 2 == 0) → uses a lambda expression to filter the array and create a new array that contains only the even numbers.
4. foreach (int num in evenNumbers) → loops through the evenNumbers array using a foreach loop and prints each even number to the console.
5. var evenNumbers = Array.FindAll(numbers, (int n) => { return n % 2 == 0; }); → uses a lambda statement to filter the array and create a new array that contains only the even numbers.
6. Array.FindAll → method that takes two arguments, the first argument is the array numbers that needs to be filtered and the second argument is a lambda expression/lambda statement.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace lambdaexpression
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create an array of numbers
            int[] numbers = { 10, 23, 43, 24, 65, 77, 80, 3,
             2, 12 };

            // Use a lambda expression to filter the array
            var evenNumbers = Array.FindAll(numbers, n => n
             % 2 == 0);

            // Print the even numbers to the console
            Console.WriteLine("Even numbers in the array:");
            foreach (int num in evenNumbers)
            {
                Console.WriteLine(num);
            }
        }
    }
}
```

Output:

```
Select Microsoft Visual Studio Debug Console
Even numbers in the array:
10
24
80
2
12

D:\Utsav\Visual Studio 2022\Projects\Lab10\Lab10\bin\Debug\net6.0\Lab10.exe (process 11140) exited with code 0.
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace lambdastatement
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create an array of numbers
            int[] numbers = { 10, 23, 43, 24, 65, 77, 80,
             3,2, 12};

            // Use a lambda statement to filter the array
            var evenNumbers = Array.FindAll(numbers, (int n)
             =>
            {
                return n % 2 == 0;
            });

            // Print the even numbers to the console
            Console.WriteLine("Even numbers in the array:");
            foreach (int num in evenNumbers)
            {
                Console.WriteLine(num);
            }
        }
    }
}
```

Output:

```
Select Microsoft Visual Studio Debug Console
Even numbers in the array:
10
24
80
2
12

D:\Utsav\Visual Studio 2022\Projects\Lab10.2\Lab10.2\bin\Debug\net6.0\Lab10.2.exe (process 16040) exited with code 0.
```

Lab 11: Date: 2080/01/21

Title: Exception Handling

Introduction:

Exception Handling:

Exception handling in C# is a mechanism that allows programmers to handle and recover from runtime errors or exceptional situations that may occur while the program is running. These errors may be caused by a variety of reasons, such as unexpected user input, hardware or software malfunctions, or resource constraints.

In C#, exceptions are represented by objects of the Exception class or its derived classes. When an exception is thrown, the runtime searches for an appropriate exception handler that can catch and handle the exception. If no handler is found, the program terminates and the exception is logged.

There are two type of exception handling mechanism: Logical Implementation and Try Catch Implementation.

Try Catch Implementation:

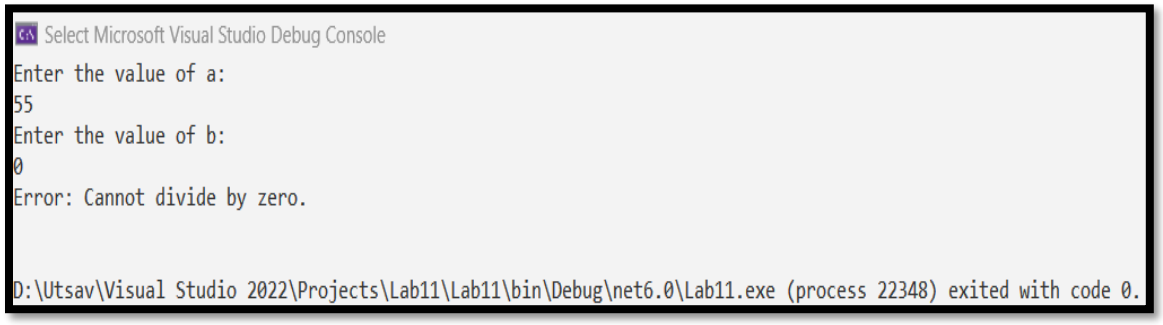
The try-catch implementation of exception handling in C# provides a mechanism for handling runtime errors that may occur during program execution. By enclosing the code that may throw an exception in a try block, you can catch any exception that is thrown and handle it in a catch block. This allows us to gracefully handle runtime errors by displaying appropriate error messages, logging the exception details, or taking other appropriate actions.

In addition, you can use a finally block to perform cleanup operations or release any resources that were allocated in the try block, regardless of whether an exception was thrown or not. The try-catch implementation of exception handling is an important aspect of writing robust and reliable C# code.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ExceptionLogical
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b, c;

            Console.WriteLine("Enter the value of a:");
            a = int.Parse(Console.ReadLine());

            Console.WriteLine("Enter the value of b:");
            b = int.Parse(Console.ReadLine());

            if (b == 0)
            {
                Console.WriteLine("Error: Cannot divide by
                    zero.");
            }
            else
            {
                c = a / b;
                Console.WriteLine("Division of a by b is = "
                    + c);
            }
                Console.ReadLine();
        }
    }
}
```

Output:



Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Exceptiontrycatch
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b, c;

            Console.WriteLine("Enter the value of a:");
            a = int.Parse(Console.ReadLine());

            Console.WriteLine("Enter the value of b:");
            b = int.Parse(Console.ReadLine());

            try
            {
                if (b == 0)
                {
                    throw new DivideByZeroException("Error:
                          Cannot divide by zero.");
                }

                c = a / b;
                Console.WriteLine("Division of a by b is = "
                   + c);
            }
            catch (DivideByZeroException ex)
            {
                Console.WriteLine(ex.Message);
            }

            Console.ReadLine();
        }
    }
}
```

Output:

```
Select Microsoft Visual Studio Debug Console
Enter the value of a:
55
Enter the value of b:
0
Error: Cannot divide by zero.


D:\Utsav\Visual Studio 2022\Projects\Lab11.2\Lab11.2\bin\Debug\net6.0\Lab11.2.exe (process 22468) exited with code 0.
```

Lab 12: Date: 2080/02/04

Title: Constructor

Introduction:

Constructor:

Constructor is a special method of a class which will invoke automatically whenever instance or object of class is created. Constructor name should match with class and constructor does not have any return type.

Constructor are responsible for object initialization and memory allocation of its class. If we create any class without constructor, compiler will automatically create default constructor for that class. There is always at least one constructor in every class.

Types of constructors:

i. Default Constructor: A constructor without having any parameter are called default constructor. In this constructor every instance of the class will be initialized without any parameter value.

ii. Parameterized Constructor: A constructor with at least one parameter is called a parameterized constructor. The advantage of a parameterized constructor is that we can initialize the value at the time of certain of object of class.

iii. Copy Constructor: A parameterized constructor that contains a parameter of same class type is called as copy constructor. The purpose of copy constructor is to initialize new instance to values of an existing instance.

iv. Private Constructor: A constructor with private access modifier is known as private constructor. It is generally used in classes that contain static members only.

v. Static Constructor: A constructor with static keyword is known as static constructor. It is used to initialize static fields of the class and to write the code that needs to be executed only once.

Code:

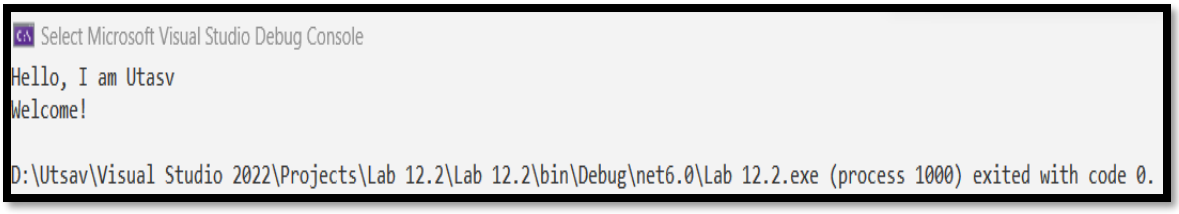
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
    public class MyClass
    {
        private string message;

        // Default constructor
        public MyClass()
        {
            message = "Hello,I am Utsav";
        }

        public void DisplayMessage()
        {
            Console.WriteLine(message);
        }
    }

    public class Program
    {
        public static void Main(string[] args)
        {
            MyClass obj = new MyClass();
            obj.DisplayMessage();
        }
    }
  }
```

Output:

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
    public class MyClass
    {
        private string message;

        // Parameterized constructor
        public MyClass(string msg)
        {
            message = msg;
        }

        public void DisplayMessage()
        {
            Console.WriteLine(message);
        }
    }

    public class Program
    {
        public static void Main(string[] args)
        {
            MyClass obj1 = new MyClass("Hello, I am Utasv");
            obj1.DisplayMessage();

            MyClass obj2 = new MyClass("Welcome!");
            obj2.DisplayMessage();
        }
    }
 }
```

Output:



Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
    public class MyClass
    {
        private string message;

        public MyClass(string msg)
        {
            message = msg;
        }

        // Copy method
        public MyClass Copy()
        {
            MyClass copy = new MyClass(this.message);
            return copy;
        }

        public void DisplayMessage()
        {
            Console.WriteLine(message);
        }
    }

    public class Program
    {
        public static void Main(string[] args)
        {
            MyClass obj1 = new MyClass("Hello, I am Utsav");
            obj1.DisplayMessage();

            MyClass obj2 = obj1.Copy();
            obj2.DisplayMessage();
        }
    }
  }
```

Output:

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
    public class MyClass
    {
        private static int instanceCount;
        private string message;

        // Static constructor
        static MyClass()
        {
            instanceCount = 0;
            Console.WriteLine("Static constructor called.");
        }

        public MyClass(string msg)
        {
            message = msg;
            instanceCount++;
        }

        public void DisplayMessage()
        {
            Console.WriteLine(message);
        }

        public static void DisplayInstanceCount()
        {
            Console.WriteLine("Instance count: " +
instanceCount);
        }
    }

    public class Program
    {
        public static void Main(string[] args)
        {
            MyClass.DisplayInstanceCount();

            MyClass obj1 = new MyClass("Hello");
            obj1.DisplayMessage();
            MyClass.DisplayInstanceCount();

            MyClass obj2 = new MyClass("I");
            obj2.DisplayMessage();
            MyClass.DisplayInstanceCount();

            MyClass obj3 = new MyClass("am");
            obj3.DisplayMessage();
            MyClass.DisplayInstanceCount();

            MyClass obj4 = new MyClass("Utsav");
```

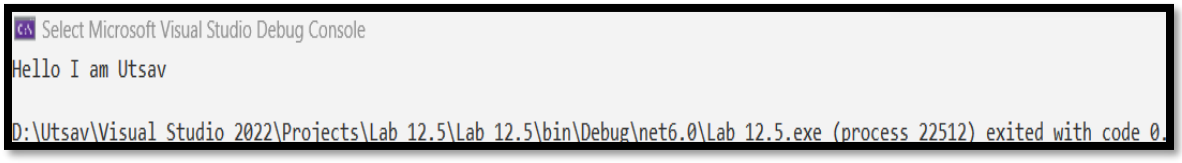
```
            obj4.DisplayMessage();
            MyClass.DisplayInstanceCount();
        }
    }
}
```

Output:

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
    public class MyClass
    {
        private string message;

        // Private constructor
        private MyClass(string msg)
        {
            message = msg;
        }

        public void DisplayMessage()
        {
            Console.WriteLine(message);
        }

        public static MyClass CreateInstance(string msg)
        {
            return new MyClass(msg);
        }
    }

    public class Program
    {
        public static void Main(string[] args)
        {

             /* Trying to create an instance using the
          private constructor will result in an error MyClass
          obj = new MyClass("Hello"); Error: The constructor
          MyClass.MyClass(string) is not accessibleInstead, we
          use the public static method CreateInstance to create
          an instance*/

              MyClass obj = MyClass.CreateInstance("Hello I am
              Utsav");
            obj.DisplayMessage();
        }
    }
}
```

Output:



Lab 13: Date: 2080/02/05

Title: Inheritance

Introduction:

Inheritance:

In C#, inheritance is a fundamental concept in object-oriented programming (OOP) that allows you to define a new class based on an existing class. The new class, called the derived class or subclass, inherits the properties, methods, and behavior of the existing class, known as the base class or superclass. This enables code reuse and promotes a hierarchical structure for organizing and extending classes.

To establish an inheritance relationship between classes in C#, you use the colon (:) symbol followed by the name of the base class after the derived class declaration. The syntax for defining a derived class that inherits from a base class is as follows:

```
class DerivedClass : BaseClass
{
// Additional members and methods specific to the derived class
}
```

Here, DerivedClass is the name of the new class you're creating, and BaseClass is the name of the existing class from which you want to inherit.

The derived class automatically acquires all the public and protected members (fields, properties, and methods) of the base class. It can also add new members or override existing members of the base class to provide specialized behavior. This process is known as extending or overriding the base class functionality.

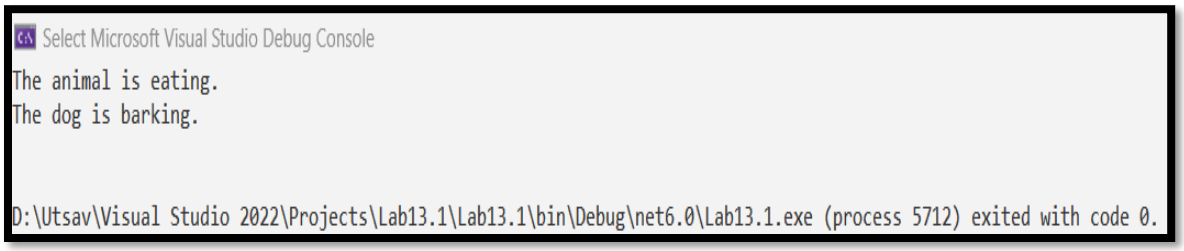
Types of inheritance:

1. Single Inheritance: Single inheritance refers to a derived class inheriting from a single base class. C# supports single inheritance, where a class can have only one direct base class.
2. Multilevel Inheritance: Multilevel inheritance involves creating a chain of derived classes, where each derived class inherits from a base class, and subsequent derived classes inherit from those derived classes. This creates a hierarchical structure of inheritance.
3. Hierarchical Inheritance: Hierarchical inheritance occurs when multiple derived classes inherit from a single base class. Each derived class shares common characteristics and behavior from the base class while adding its own specific features.
4. Multiple Interface Inheritance: C# supports multiple interface inheritance, where a class can implement multiple interfaces. An interface defines a contract specifying a set of methods and properties. By implementing multiple interfaces, a class can exhibit behavior from different sources.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace inheritance
{
    // Base class
    class Animal
    {
        public void Eat()
        {
            Console.WriteLine("The animal is eating.");
        }
    }

    // Derived class inheriting from Animal
    class Dog : Animal
    {
        public void Bark()
        {
            Console.WriteLine("The dog is barking.");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Create an instance of the derived class
            Dog myDog = new Dog();

            // Access base class method
            myDog.Eat();

            // Access derived class method
            myDog.Bark();

            Console.ReadLine();
        }
    }
 }
```

Output:


Select Microsoft Visual Studio Debug Console
The animal is eating.
The dog is barking.
D:\Utsav\Visual Studio 2022\Projects\Lab13.1\Lab13.1\bin\Debug\net6.0\Lab13.1.exe (process 5712) exited with code 0.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace inheritance
{
    // Base class
    class Animal
    {
        public void Eat()
        {
            Console.WriteLine("The animal is eating.");
        }
    }

    // Derived class inheriting from Animal
    class Dog : Animal
    {
        public void Bark()
        {
            Console.WriteLine("The dog is barking.");
        }
    }

    // Derived class inheriting from Dog
    class Labrador : Dog
    {
        public void Swim()
        {
            Console.WriteLine("The Labrador is swimming.");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Create an instance of the derived class
            Labrador myLabrador = new Labrador();

            // Access base class method
            myLabrador.Eat();

            // Access intermediate derived class method
            myLabrador.Bark();

            // Access derived class method
            myLabrador.Swim();

            Console.ReadLine();
        }
    }
}
```

Output:


Select Microsoft Visual Studio Debug Console
The animal is eating.
The dog is barking.
The Labrador is swimming.
D:\Utsav\Visual Studio 2022\Projects\Lab13.2\Lab13.2\bin\Debug\net6.0\Lab13.2.exe (process 23196) exited with code 0.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace inheritance
{
    // Base class
    class Animal
    {
        public void Eat()
        {
            Console.WriteLine("The animal is eating.");
        }
    }
    // Derived class inheriting from Animal
    class Dog : Animal
    {
        public void Bark()
        {
            Console.WriteLine("The dog is barking.");
        }
    }
    // Another derived class inheriting from Animal
    class Cat : Animal
    {
        public void Meow()
        {
            Console.WriteLine("The cat is meowing.");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Create instances of the derived classes
            Dog myDog = new Dog();
            Cat myCat = new Cat();

            // Access base class method from Dog
            myDog.Eat();

            // Access derived class method from Dog
            myDog.Bark();

            // Access base class method from Cat
            myCat.Eat();

            // Access derived class method from Cat
            myCat.Meow();

            Console.ReadLine();
        }
    }
  }
```

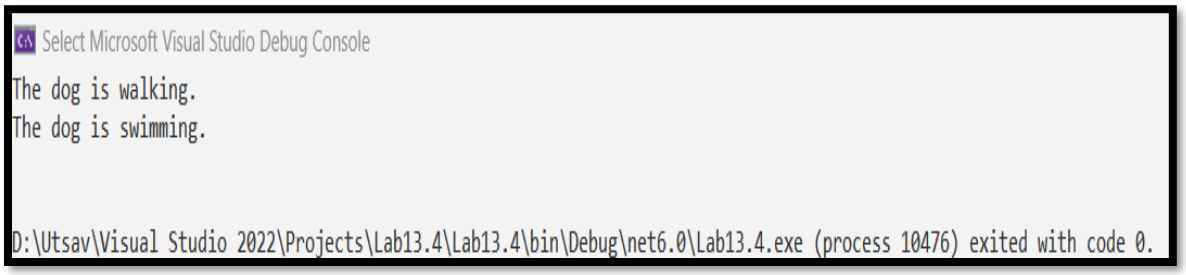
Output:


Select Microsoft Visual Studio Debug Console
The animal is eating.
The dog is barking.
The animal is eating.
The cat is meowing.
D:\Utsav\Visual Studio 2022\Projects\Lab13.3\Lab13.3\bin\Debug\net6.0\Lab13.3.exe (process 4840) exited with code 0.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace inheritance
{
    // First interface
    interface IWalkable
    {
        void Walk();
    }

    // Second interface
    interface ISwimmable
    {
        void Swim();
    }

    // Class implementing both interfaces
    class Dog : IWalkable, ISwimmable
    {
        public void Walk()
        {
            Console.WriteLine("The dog is walking.");
        }

        public void Swim()
        {
            Console.WriteLine("The dog is swimming.");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Create an instance of the class
            Dog myDog = new Dog();

            // Access methods from both interfaces
            myDog.Walk();
            myDog.Swim();

            Console.ReadLine();
        }
    }
  }
```

Output:


Select Microsoft Visual Studio Debug Console
The dog is walking.
The dog is swimming.
D:\Utsav\Visual Studio 2022\Projects\Lab13.4\Lab13.4\bin\Debug\net6.0\Lab13.4.exe (process 10476) exited with code 0.

Lab 14: Date: 2080/02/05

Title: Structs and Enums

Introduction:

Structs:

C# struct also known as C# structure is a simple user-defined type, a lightweight alternative to a class. Similar to classes, structures have behaviors and attributes. C# structs support access modifiers, constructors, indexers, methods, fields, nested types, operators, and properties.

Example:

```
Struct Books{
        public string title;
        public string subject;
        public int id;
};
```

Enums:

An enumeration is a set of named integer constants. An enumerated type is declared using the Enum keyword. C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

Declaring Enum variable syntax:

```
enum <enum_name>{
        enumeration list
};
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace structure
{
    struct Person
    {
        public string Name;
        public int Age;
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Create a new instance of the Person struct
            Person person1;
            person1.Name = "Utsav";
            person1.Age = 22;

            // Access and display the struct members
            Console.WriteLine("Person 1:");
            Console.WriteLine("Name: " + person1.Name);
            Console.WriteLine("Age: " + person1.Age);

            // Create another instance of the Person struct
            Person person2 = new Person();
            person2.Name = "Ram";
            person2.Age = 30;

            Console.WriteLine("\nPerson 2:");
            Console.WriteLine("Name: " + person2.Name);
            Console.WriteLine("Age: " + person2.Age);

            Console.ReadLine();
        }
    }
}
```

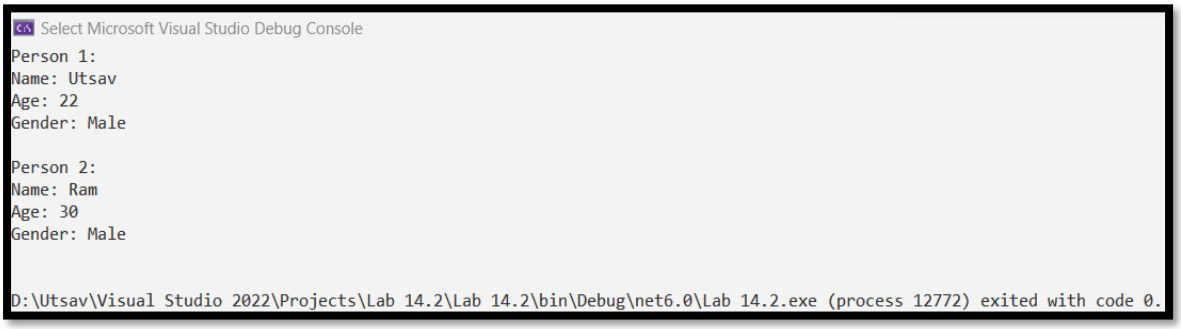
Output:

```
Select Microsoft Visual Studio Debug Console
Person 1:
Name: Utsav
Age: 22

Person 2:
Name: Ram
Age: 30


D:\Utsav\Visual Studio 2022\Projects\Lab14\Lab14\bin\Debug\net6.0\Lab14.exe (process 4644) exited with code 0.
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace enumuration
{
    enum Gender
    {
        Male,
        Female,
        Other
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Create a new instance of the Person struct
            string person1Name = "Utsav";
            int person1Age = 22;
            Gender person1Gender = Gender.Male;
            // Access and display the enum members
            Console.WriteLine("Person 1:");
            Console.WriteLine("Name: " + person1Name);
            Console.WriteLine("Age: " + person1Age);
            Console.WriteLine("Gender: " + person1Gender);
            // Create another instance of the Person struct
            string person2Name = "Ram";
            int person2Age = 30;
            Gender person2Gender = Gender.Male;

            Console.WriteLine("\nPerson 2:");
            Console.WriteLine("Name: " + person2Name);
            Console.WriteLine("Age: " + person2Age);
            Console.WriteLine("Gender: " + person2Gender);

            Console.ReadLine();
        }
    }
}
```

Output:



Lab 15: Date: 2080/02/12

Title: Razor Implementation

Introduction:

Razor:

In the realm of net-centric computing, Razor stands as a prominent player, leveraging its expertise to provide cutting-edge solutions and services. With a focus on connectivity, Razor offers a range of products and technologies that enable seamless integration and efficient communication within networked environments. Their innovative networking solutions empower businesses and individuals to harness the full potential of net-centric computing, facilitating data exchange, collaboration, and streamlined operations. Razor's commitment to delivering reliable, scalable, and secure solutions has earned them a reputation for excellence in the ever-evolving landscape of net-centric computing, driving progress and enabling digital transformation for a connected world.

MVC:

MVC, which stands for Model-View-Controller, is a software architectural pattern widely used in the development of web and desktop applications. The MVC pattern separates an application into three interconnected components: the model, the view, and the controller. The model represents the application's data and business logic, handling data manipulation and storage. The view handles the presentation of data to the user, providing the visual interface and user interaction elements. The controller acts as the intermediary between the model and the view, receiving user input, updating the model, and updating the view accordingly. This separation of concerns promotes modularity, maintainability, and reusability, allowing developers to efficiently manage and evolve complex applications. The MVC pattern has become a cornerstone of modern software development, providing a structured approach to building robust and scalable applications.

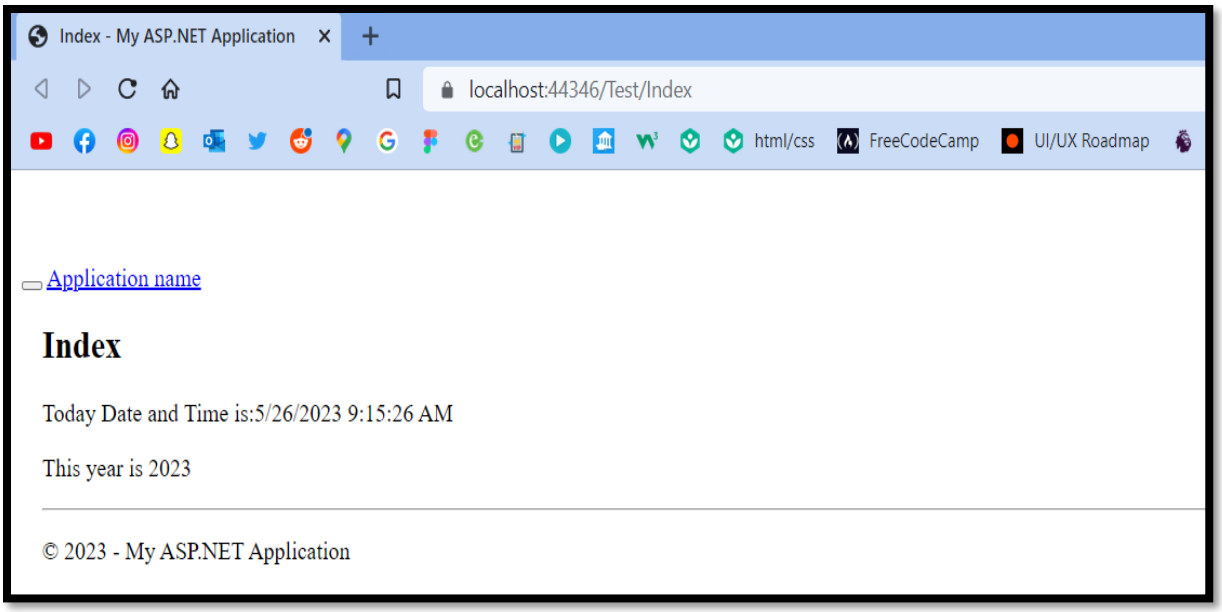
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_15.Controllers
{
    public class TestController : Controller
    {
        // GET: Test
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
<p>Today Date and Time is:@DateTime.Now</p>
<p>This year is @DateTime.Now.Year</p>
```

Output:


Index - My ASP.NET Application
localhost:44346/Test/Index
html/css
FreeCodeCamp
UI/UX Roadmap
Application name
Index
Today Date and Time is:5/26/2023 9:15:26 AM
This year is 2023
© 2023 - My ASP.NET Application

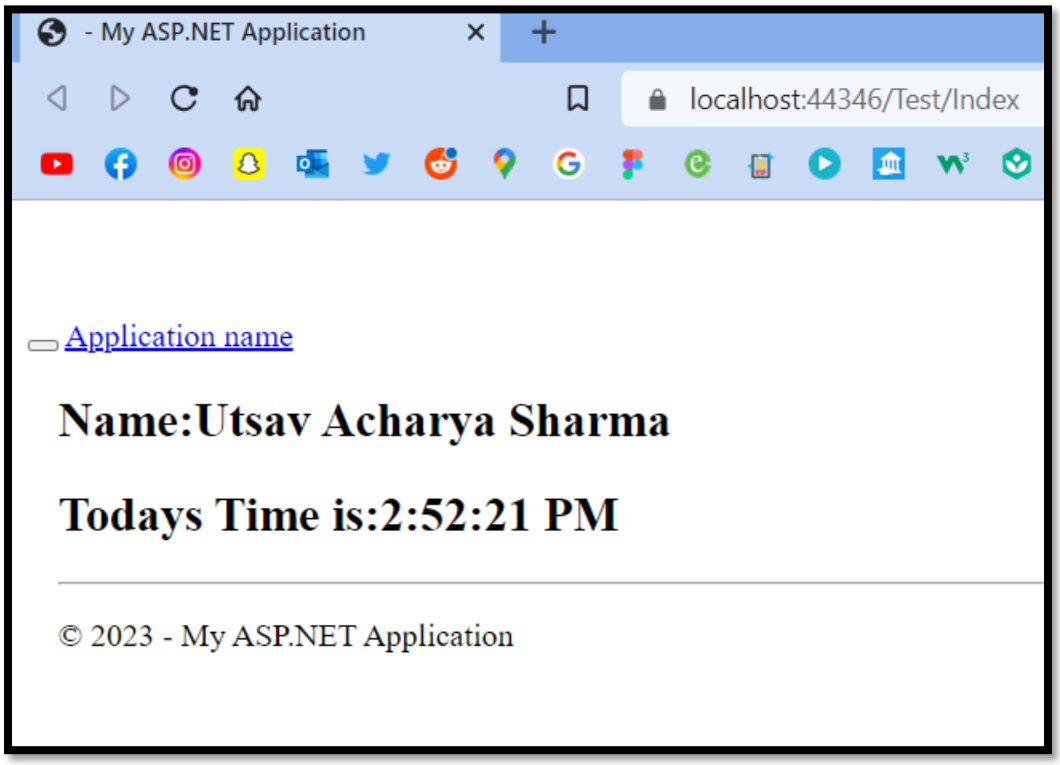
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_15.Controllers
{
    public class TestController : Controller
    {
        // GET: Test
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
<!--Declaring Variables-->
@{
    String name = "Utsav Acharya Sharma";
    var TodaysTime = DateTime.Now.ToLongTimeString();
}
<h2>Name:@name</h2>
<h2>Todays Time is:@TodaysTime</h2>
```

Output:


- My ASP.NET Application
localhost:44346/Test/Index
Application name
Name:Utsav Acharya Sharma
Todays Time is:2:52:21 PM
© 2023 - My ASP.NET Application

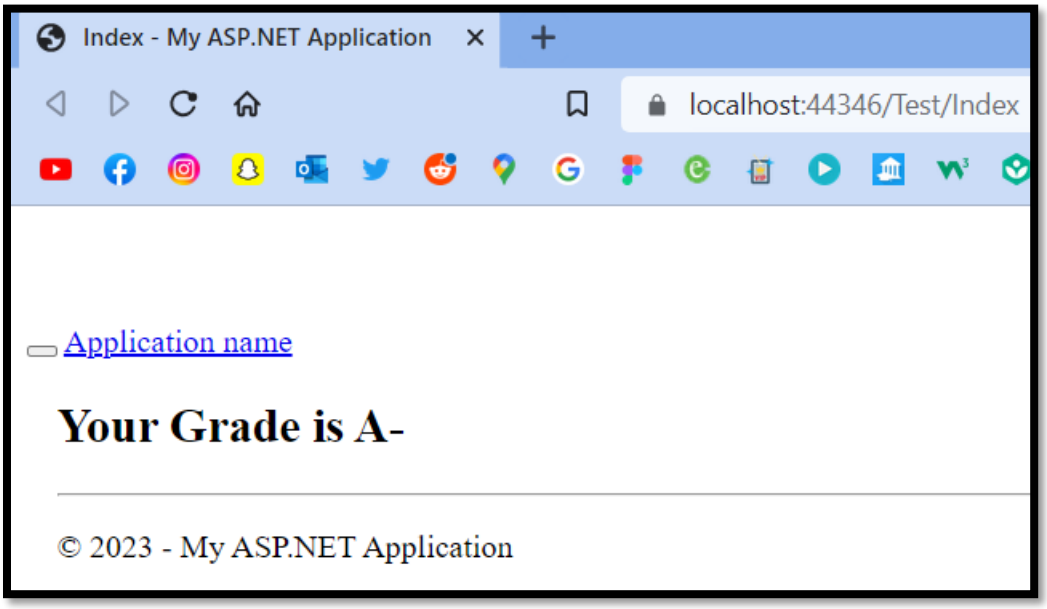
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_15.Controllers
{
    public class TestController : Controller
    {
        // GET: Test
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
@{
    int marks = 80;
    if (marks >= 80)
            {
                <h2>Your Grade is A-</h2>
            }
    else
            {
                <h2>Your Grade is A</h2>
            }
}
```

Output:


Index - My ASP.NET Application
localhost:44346/Test/Index
Application name
Your Grade is A-
© 2023 - My ASP.NET Application

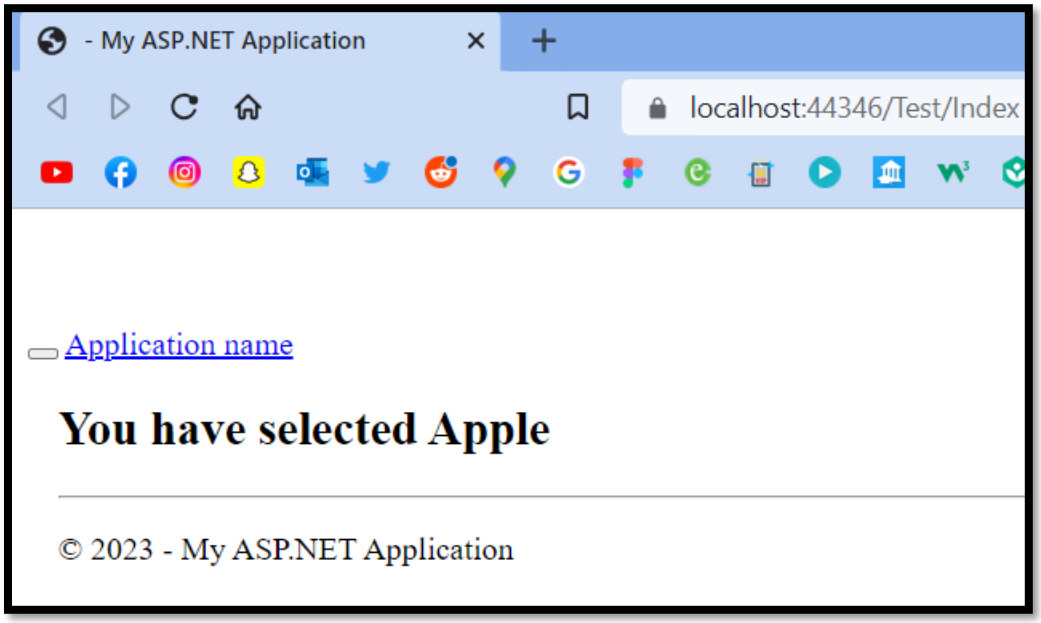
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_15.Controllers
{
    public class TestController : Controller
    {
        // GET: Test
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
@{
    String fruit = "apple";
    switch(fruit)
    {
            case "banana":
             <h2>You have selected Banana</h2>
            break;
            case "strawberry":
             <h2> You have selected Strawberry </h2>
            break;
            case "apple":
             <h2> You have selected Apple </h2>
            break;
            default:
             <h2>Invalid Fruit</h2>
            break;
        }
}
```

Output:


- My ASP.NET Application
localhost:44346/Test/Index
Application name
You have selected Apple
© 2023 - My ASP.NET Application

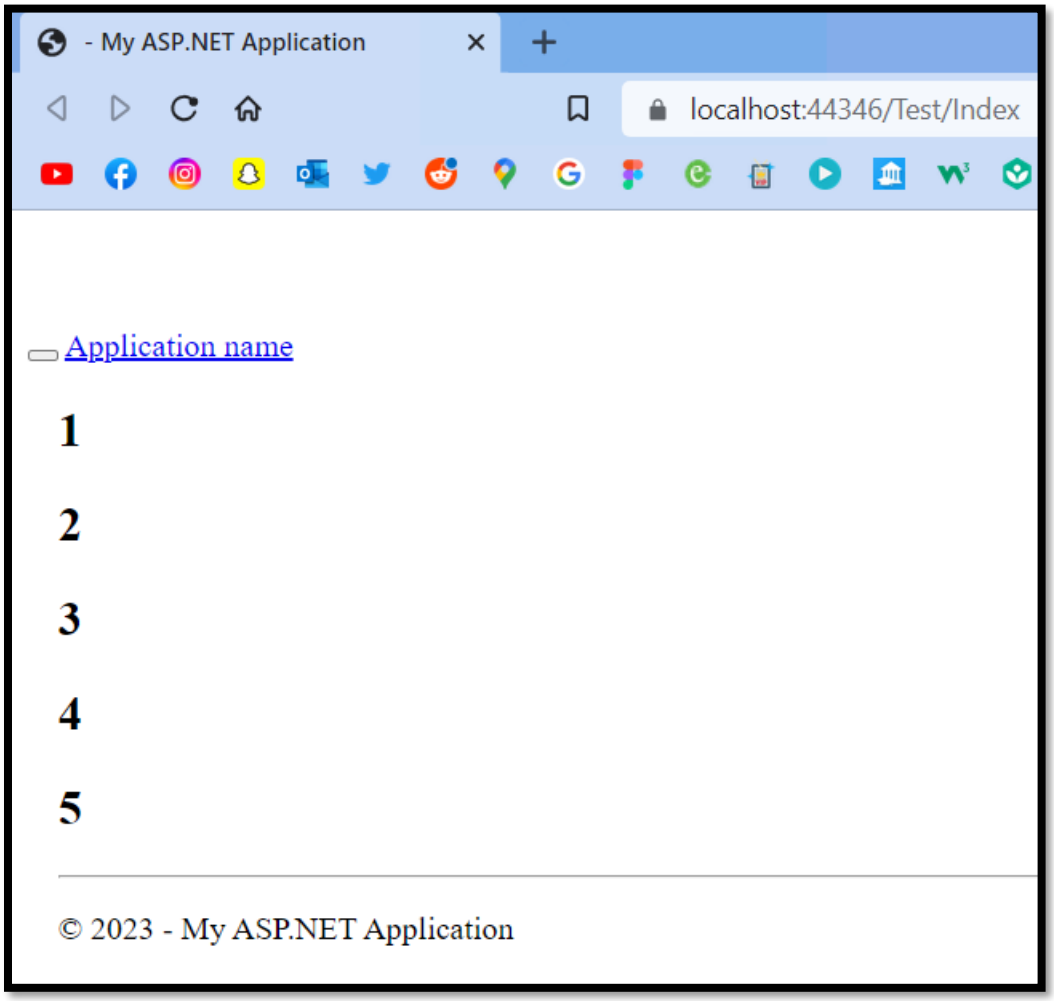
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_15.Controllers
{
    public class TestController : Controller
    {
        // GET: Test
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
<!--Loops-->
@{
    for(int i=1;i<=5;i++)
            {
                <h2>@i</h2>
            }
}
```

Output:


- My ASP.NET Application
localhost:44346/Test/Index
Application name
1
2
3
4
5
© 2023 - My ASP.NET Application

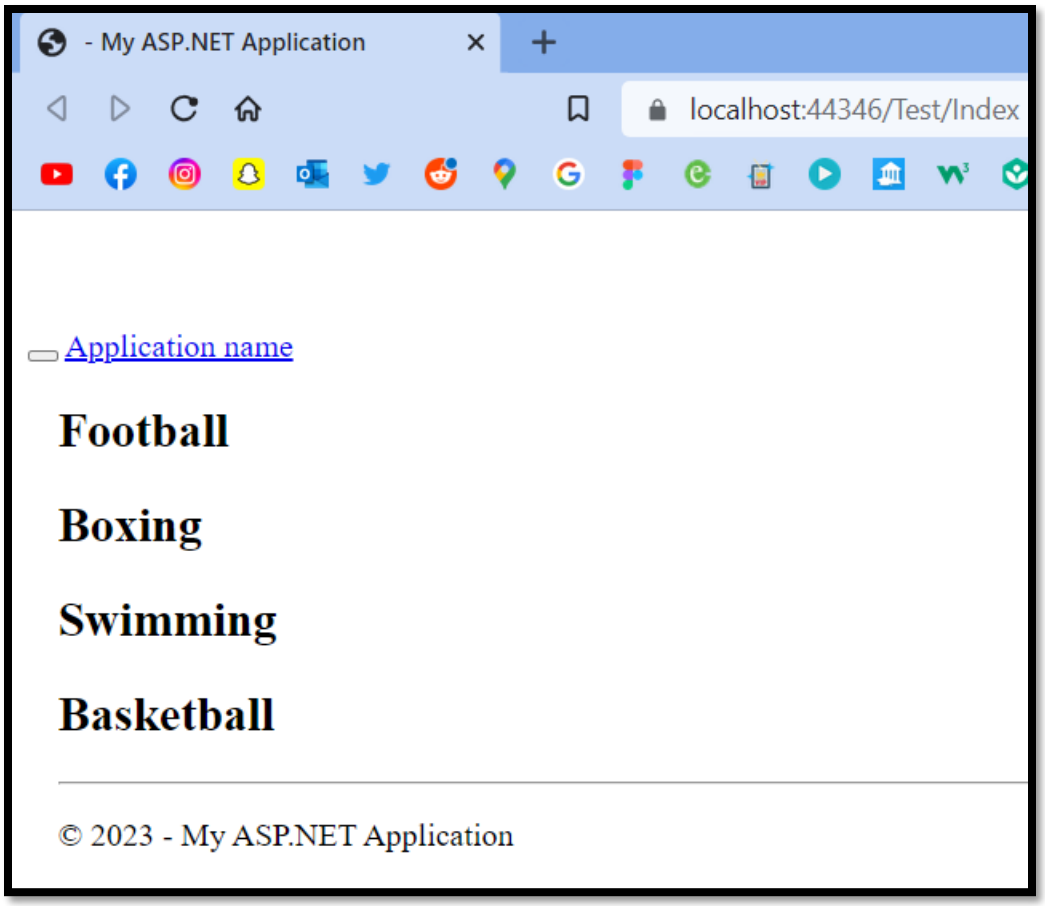
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_15.Controllers
{
    public class TestController : Controller
    {
        // GET: Test
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
<!--Loop String-->
@{
    String[] sports = { "Football", "Boxing", "Swimming",
"Basketball" };
    foreach(String i in sports)
        {
                <h2>@i</h2>
        }
}
```

Output:


- My ASP.NET Application
localhost:44346/Test/Index
Application name
Football
Boxing
Swimming
Basketball
© 2023 - My ASP.NET Application

Lab 16: Date: 2080/02/25

Title: Action Method

Introduction:

The Action method in C# Razor is a helper method used to generate a URL for a specific action within a controller. It is commonly used in conjunction with other Razor helper methods, such as Html.ActionLink or Html.BeginForm, to create links or forms that trigger specific actions in a controller. The Action method takes parameters such as the name of the action method, the name of the controller, and any additional route parameters required by the action method. By utilizing the Action method, developers can dynamically generate URLs that correspond to specific actions in their ASP.NET MVC or Razor Pages applications. This allows for the creation of interactive and navigable user interfaces, enabling users to trigger different actions within the application based on their interactions with links or forms.

Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_16.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
        public String show()
        {
            return "This is a second action method of home
controller";
        }
        public ActionResult aboutus()
        {
            return View();
        }
        public int studentID(int id)
        {
            return id;
        }
    }
}
```

View Code (Index.cshtml):

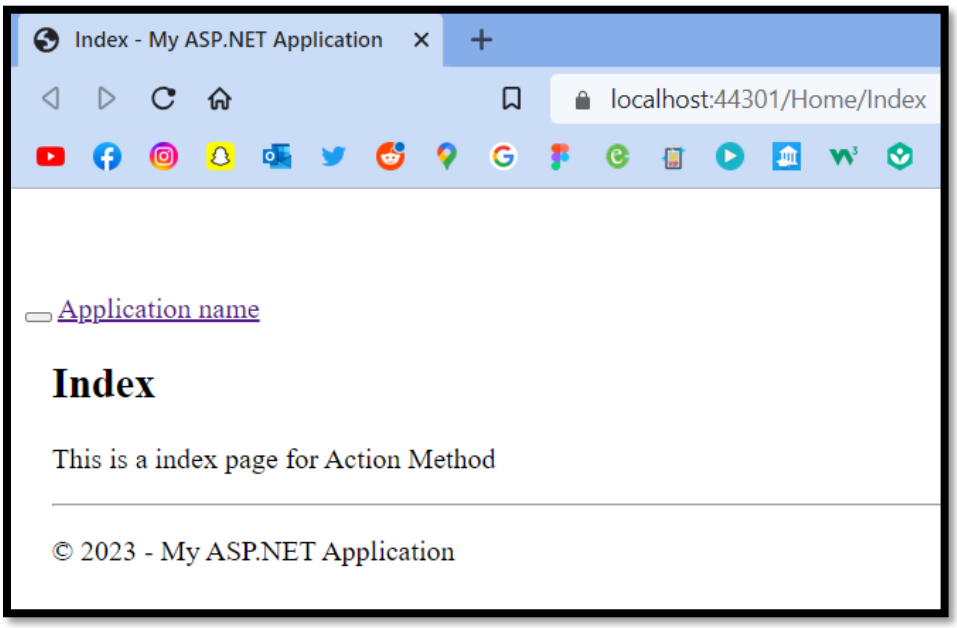
```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
<p>This is a index page for Action Method</p>
```

View Code (aboutus.cshtml):

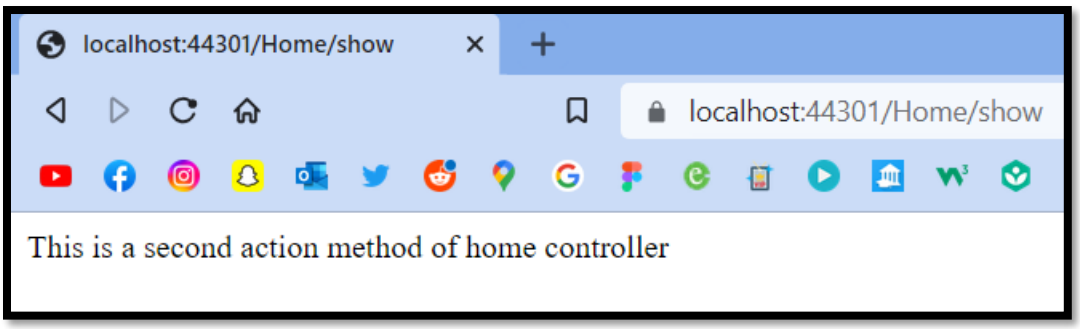
```
@{
    ViewBag.Title = "aboutus";
}

<h2>AboutUs</h2>
<p>This is the AboutUs page for Action Method</p>
```

Output:

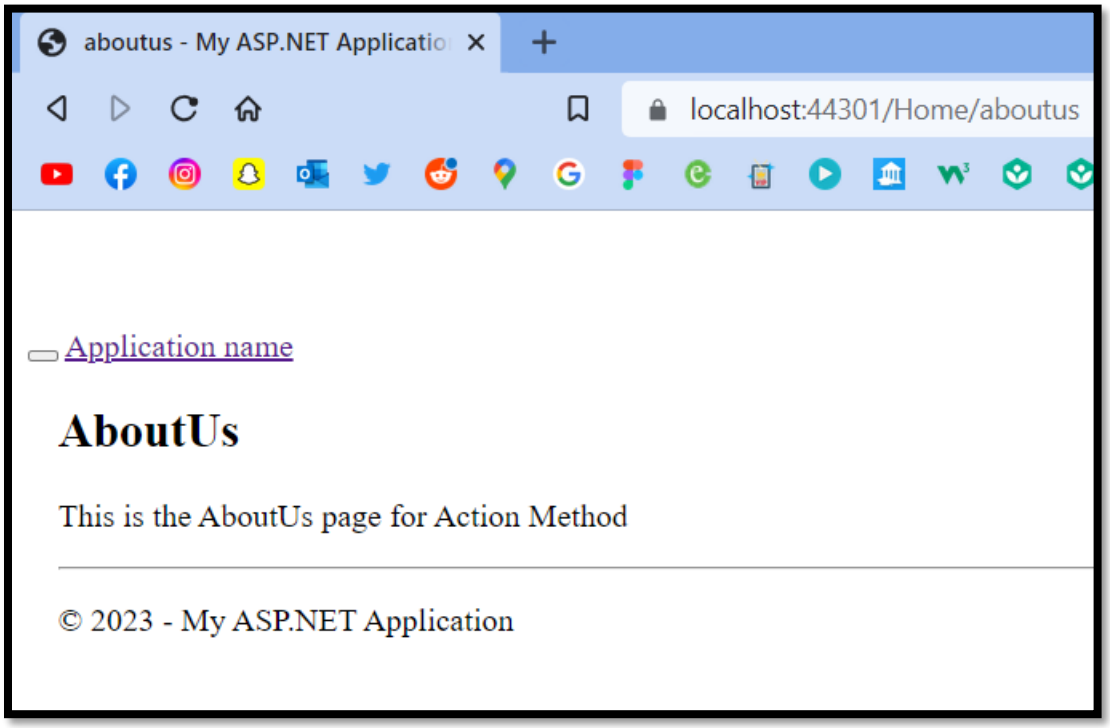
Index:



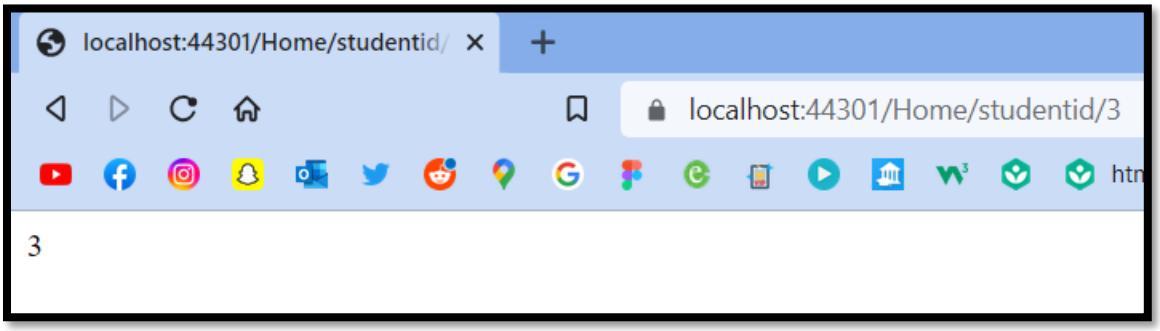
Show:



Aboutus:



StudentID:



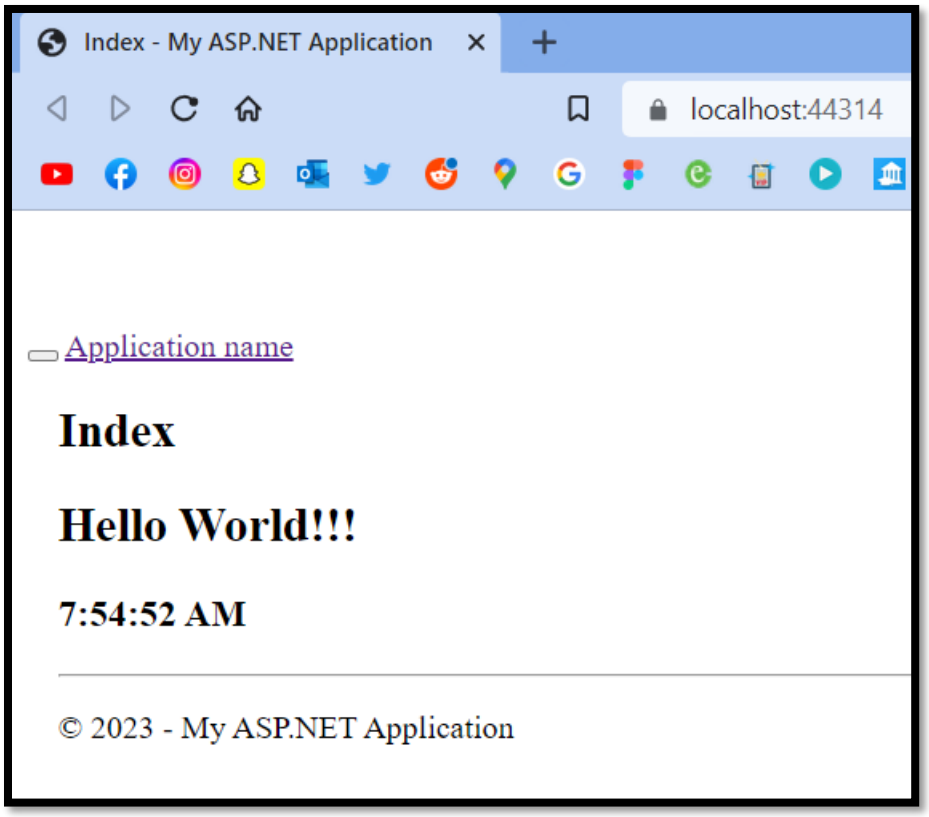
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_17.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            ViewData["Message"] = "Hello World!!!";
            ViewData["CurrentTime"] =
DateTime.Now.ToLongTimeString();
            return View();
        }
    }
}
```

View Code (Index.cshtml):

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
<h2>@ViewData["Message"]</h2>
<h3>@ViewData["CurrentTime"]</h3>
```

Output:


Index - My ASP.NET Application
localhost:44314
Application name
Index
Hello World!!!
7:54:52 AM
© 2023 - My ASP.NET Application

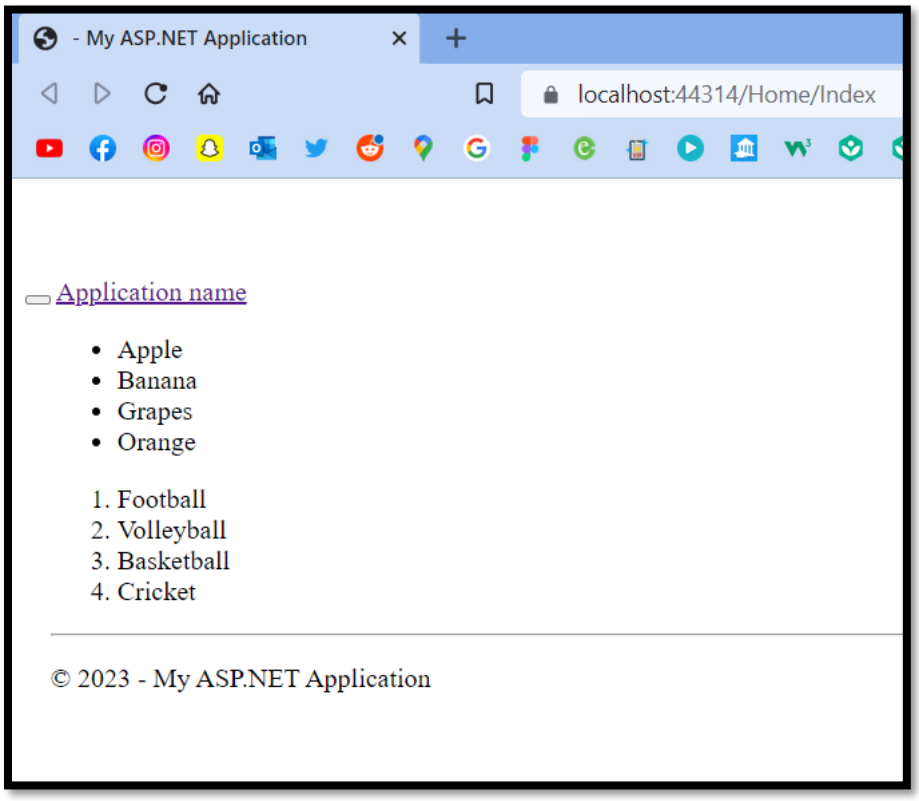
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_17.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            string[] Fruits = { "Apple", "Banana",
"Grapes", "Orange" };
            ViewData["FruitsArray"] = Fruits;
            ViewData["SportsList"] = new List<string>()
            {
                "Football",
                "Volleyball",
                "Basketball",
                "Cricket",
            };
            return View();
        }
    }
}
```

View Code (Index.cshtml):

```
<ul>
    @{
        foreach(string i in
(String[])ViewData["FruitsArray"])
            {
                <li>@i</li>
            }
    }
</ul>
<ol>
    @{
        foreach (string i in
(List<String>)ViewData["SportsList"])
        {
            <li>@i</li>
        }
    }
</ol>
```

Output:


- My ASP.NET Application
localhost:44314/Home/Index
Application name
Apple
Banana
Grapes
Orange
1. Football
2. Volleyball
3. Basketball
4. Cricket
© 2023 - My ASP.NET Application

Lab 17: Date: 2080/02/29

Title: Helper Method

Introduction:

A helper method in software development, particularly in the context of Razor, is a powerful and flexible tool that significantly enhances code reusability and maintainability. With helper methods, you can encapsulate commonly used logic or functionality within a Razor view or a separate code file, enabling you to easily reuse and share that logic across multiple views. By abstracting the code into a helper method, you can avoid duplicating it in various places, which not only reduces code redundancy but also promotes a more modular and organized approach to your application development.

Helper methods offer a concise and efficient way to encapsulate and reuse code. They can be used for a wide range of tasks, such as generating HTML markup, performing calculations, manipulating data, or even implementing complex algorithms. These methods allow you to encapsulate complex or repetitive code into a single, reusable unit that can be invoked whenever needed, making your Razor views cleaner, more readable, and easier to maintain.

Moreover, helper methods help promote the separation of concerns within your application. By extracting common functionality into helper methods, you can keep your Razor views focused on rendering the user interface and delegate other logic to the appropriate helper methods. This separation enhances the overall organization of your codebase and improves its maintainability, as it becomes easier to understand and modify the functionality encapsulated within the helper methods.

In addition to the benefits of code reuse and organization, helper methods also facilitate code sharing within a team or across projects. Since they can be defined in separate code files, they can be easily shared among different views or even different projects. This sharing promotes consistency in implementation and reduces the chances of errors or inconsistencies in how specific functionality is implemented across different parts of your application.

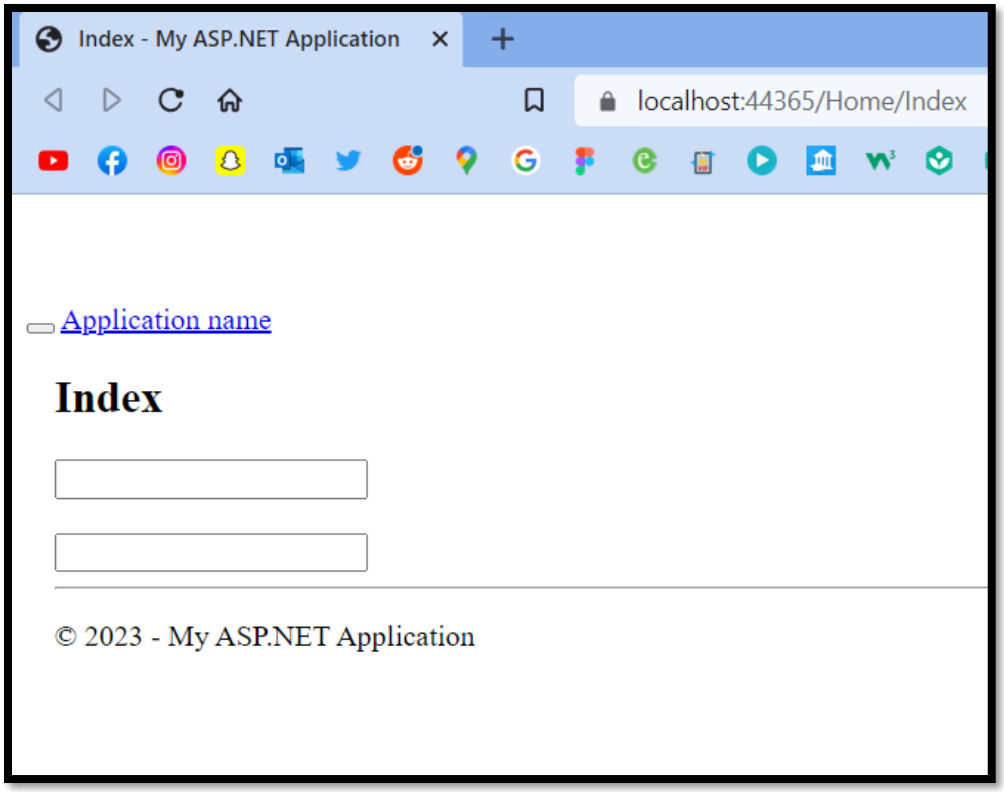
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_17.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code :

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
<input type="text" name="fullname" id="fullname"
value=""/>
<br/>
<br/>
@Html.TextBox("fullname")
```

Output:


Index - My ASP.NET Application
localhost:44365/Home/Index
Application name
Index
© 2023 - My ASP.NET Application

Lab 18: Date: 2080/02/30

Title: Tag Helper (TextBox)

Introduction:

The purpose of this lab report is to introduce the Tag Helper HTML.Textbox, a sophisticated tool designed to revolutionize the creation and functionality of web forms. With its advanced features and intuitive user interface, this tool enables developers to incorporate tagging functionality seamlessly, providing users with a more structured and organized approach to data entry. By implementing the Tag Helper HTML.Textbox, researchers and web developers can enhance the efficiency and effectiveness of their web applications, whether they are utilized in blogging, e-commerce, or collaborative platforms. This lab report will explore the features and benefits of the Tag Helper HTML.Textbox, demonstrating its potential for optimizing data management and user experiences. Through a comprehensive evaluation, we aim to provide insights into the practical applications and advantages of integrating this powerful tool into web development projects.

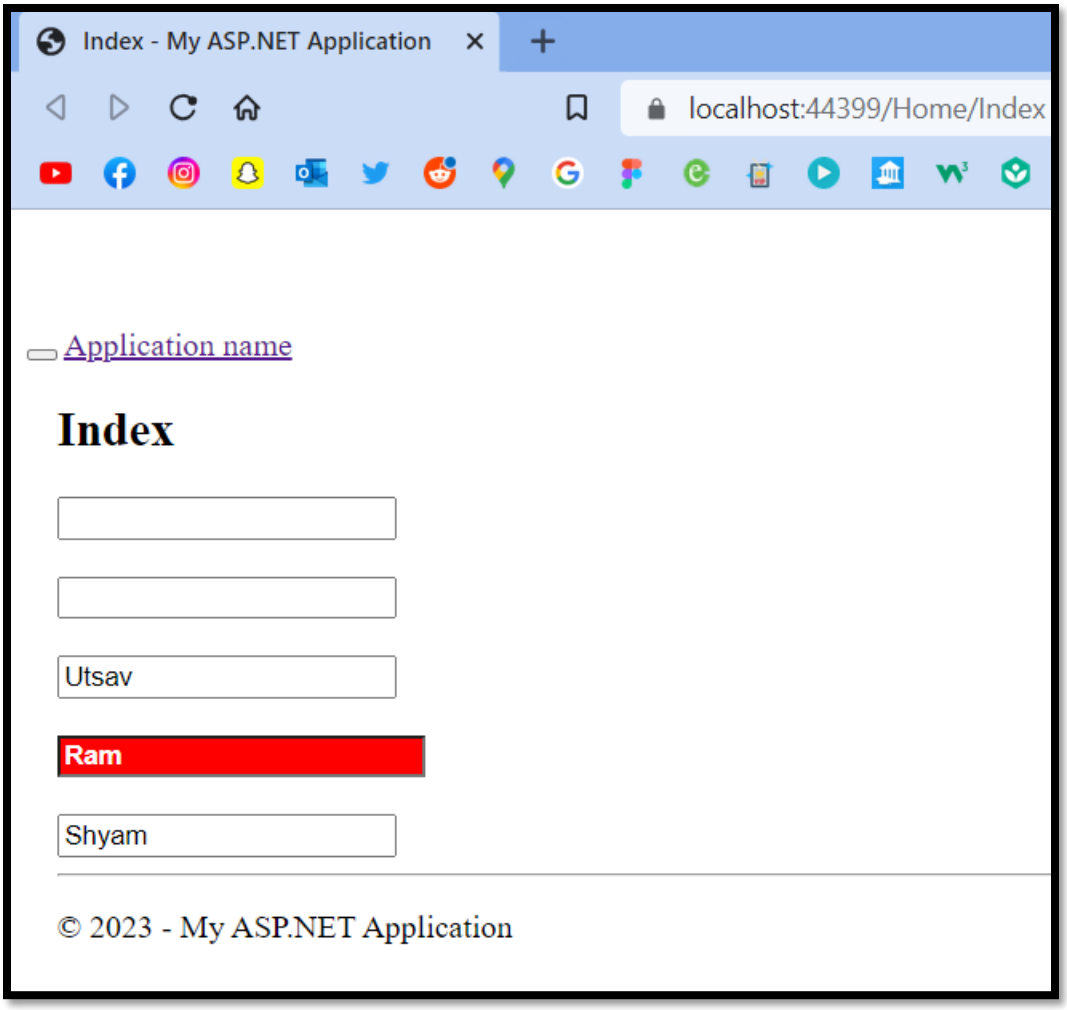
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_18.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
<input type="text" name="fullname" id="fullname" value=""
/>
<br />
<br />
@Html.TextBox("fullname")
<br />
<br />
@Html.TextBox("fullname", "Utsav")
<br />
<br />
@Html.TextBox("fullname", "Ram", new { style =
"background-color:Red; color:White; font-weight:bold" })
<br />
<br />
@Html.TextBox("fullname","Shyam", new {@class = "form-
control", @readonly = true })
```

Output:


Index - My ASP.NET Application
localhost:44399/Home/Index
Application name
Index
Utsav
Ram
Shyam
© 2023 - My ASP.NET Application

Lab 19: Date: 2080/02/30

Title: Tag Helper (HtmlForm)

Introduction:

The objective of this lab report is to introduce HTML.Form in Tag Helper, a powerful tool that simplifies the creation of interactive and dynamic forms in HTML. With its intuitive and user-friendly syntax, HTML.Form in Tag Helper enables developers to generate form elements effortlessly while incorporating essential functionalities such as validation, data binding, and styling. This lab report aims to explore the features and capabilities of HTML.Form in Tag Helper, showcasing its ability to streamline form development and enhance the user experience. By utilizing this robust framework, researchers and web developers can create functional and visually appealing forms, ranging from simple contact forms to complex multi-step registration forms. Through a comprehensive evaluation, we will delve into the practical applications and advantages of integrating HTML.Form in Tag Helper into web development projects, unlocking the full potential of web applications' form functionalities.

Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_19.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
<!--Html.Form-->
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
@{
    Html.BeginForm("Index", "Home");
}
<!--Html.Username-->
@Html.Label("username") @Html.TextBox("Username")
<br />
<br />
<!--Html.password-->
@Html.Label("Password") @Html.Password("Password")
<br />
<br />
<!--Html.RadioButton-->
@Html.Label("Male") @Html.RadioButton("Gender",
"Male")
@Html.Label("Female") @Html.RadioButton("Gender",
"Female")
<br />
<br />
<!--Html.RadioButton-->
@Html.Label("Select
Country") @Html.DropDownList("Country List", new
SelectList(new[] { "Nepal", "India", "Pakistan",
"Srilanka" }, "choose"))
<br />
<br />
<!--Html.TextArea-->
@Html.Label("Comments") @Html.TextArea("CommentArea",
"", 5, 20, new { @class = "form-control" })
```

```
<br />
<br />
<!--Html.TextArea-->
@Html.CheckBox("Acceptterms",true) @Html.Label("I
accept the terms and conditions")
@Html.Hidden("Id")
<br />
<br />
<!--Html.SubmitButton-->
<!--Not in Html Helper-->
<input type="submit" value="Submit"/>
@{
    Html.EndForm();
}
```

Output:

Lab 20: Date: 2080/02/31

Title: Adding Bootstrap to Tag Helper (HtmlForm)

Introduction:

Bootstrap is an immensely popular open-source framework utilized for building responsive and mobile-friendly websites. It offers a vast assortment of CSS and JavaScript components that streamline the development process and ensure consistent and visually appealing user interfaces across various devices and screen sizes. When working with Razor views, developers can harness the power of Bootstrap through the integration of tag helpers. Razor tag helpers simplify the creation of HTML elements with added functionality and styling options. One particularly useful class in Bootstrap is the form-control class, which is designed specifically for form input elements. By leveraging the { @class = "form-control" } tag helper syntax in Razor, developers gain a convenient means to apply the form-control class to input elements. This allows for the effortless application of pre-defined Bootstrap styles, resulting in the creation of professional-looking form controls with a consistent appearance and behavior. With the { @class = "form-control" } tag in Razor, developers can streamline their UI development process, ensuring visually cohesive and user-friendly forms in their web applications.

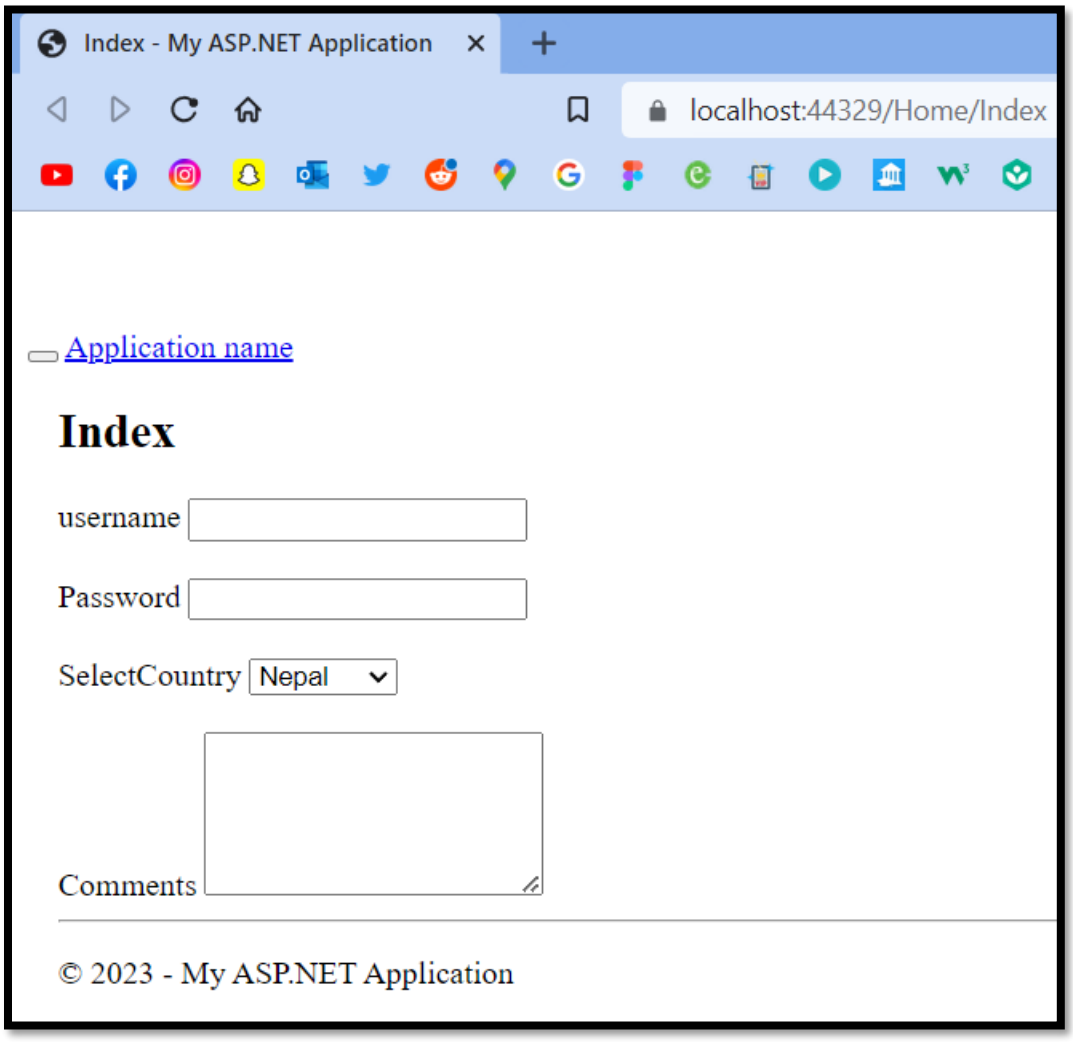
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_20.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
<!--Html.Form-->
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
@{
    Html.BeginForm("Index", "Home");
}
<!--Html.Username-->
@Html.Label("username") @Html.TextBox("Username","",
new { @class = "form-control" })
<br />
<br />
<!--Html.password-->
@Html.Label("Password") @Html.Password("Password",
"", new { @class = "form-control" })
<br />
<br />
<!--Html.DropDown-->
@Html.Label("SelectCountry") @Html.DropDownList("Coun
try List", new SelectList(new[] { "Nepal", "India",
"Pakistan","Srilanka" }, "select"), new { @class = "form-
control" })
<br />
<br />
<!--Html.TextArea-->
@Html.Label("Comments") @Html.TextArea("CommentArea",
"", 5, 20, new { @class = "form-control" })
@*Not applied for radio button and checkbox*@
```

Output:


Index - My ASP.NET Application
localhost:44329/Home/Index
Application name
Index
username
Password
SelectCountry
Nepal
Comments
© 2023 - My ASP.NET Application

Lab 21: Date: 2080/02/31

Title: Validation

Introduction:

This lab report examines the implementation of username, age, and email validation in a Razor MVC application. The objective of the lab was to demonstrate how to validate user input for these specific fields, ensuring that they meet certain criteria. The report outlines the methodology employed to achieve the desired validation using Razor syntax and MVC framework features. It discusses the rationale behind each validation requirement, provides code snippets for implementation, and evaluates the effectiveness of the validation techniques employed. The findings of this lab report contribute to a better understanding of how to implement robust data validation in Razor MVC applications, thereby enhancing the user experience and data integrity.

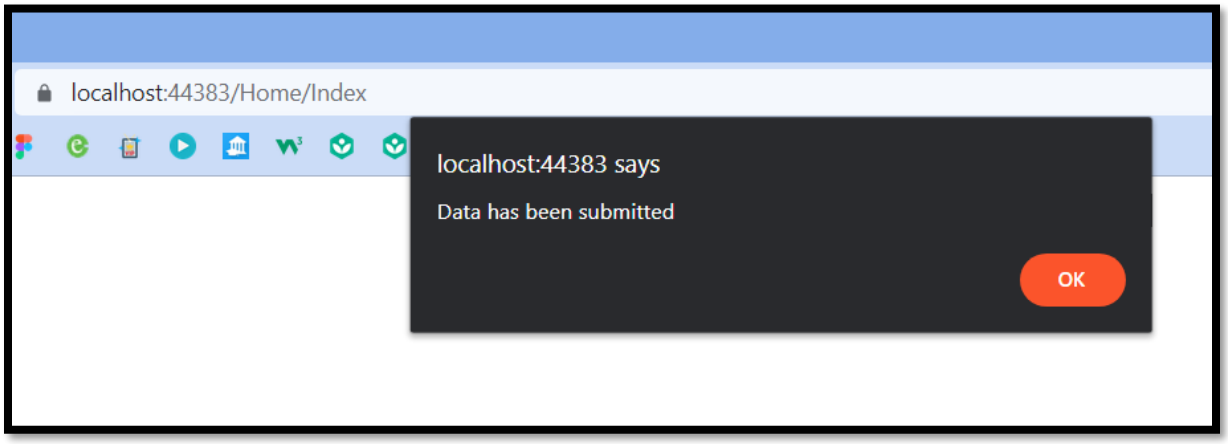
Controller Code:

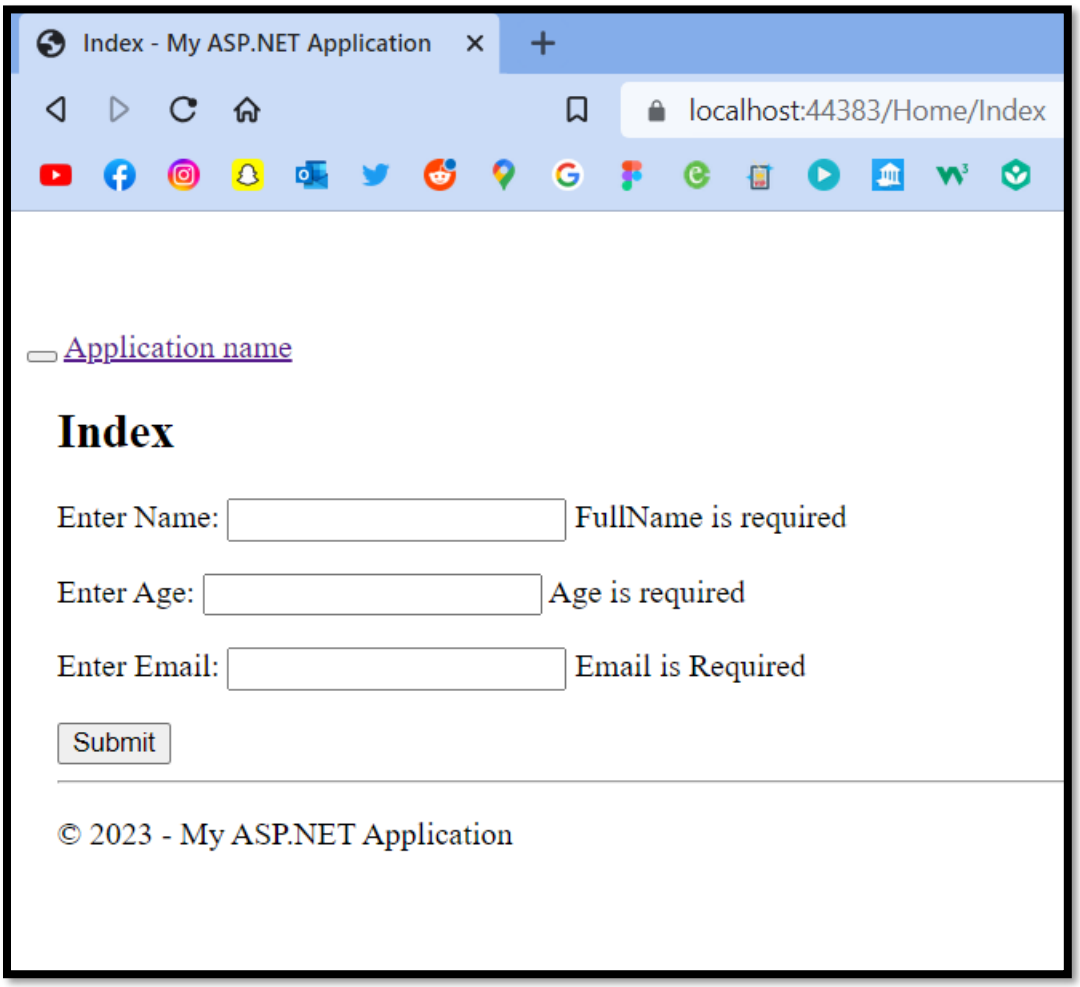
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_21.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
        [HttpPost]
        public ActionResult Index(string FullName,String
Age, String Email)
        {
            if (FullName.Equals("") == true)
            {
                ModelState.AddModelError("fullname",
"FullName is required");
            }
            if(Age.Equals("")==true)
            {
                ModelState.AddModelError("age", "Age is
required");
            }
            if(Email.Equals("")==true)
            {
                ModelState.AddModelError("email", "Email
is Required");
            }
            if(ModelState.IsValid==true)
            {
                ViewData["SuccessMessage"] =
"<script>alert('Data has been submitted')</script>";
                ModelState.Clear();
            }
            return View();
        }
    }
}
```

View Code:

```
<!--Validation-->
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
@using (Html.BeginForm())
            {
                <p>
                    Enter Name: @Html.TextBox("fullname")
                    @Html.ValidationMessage("fullname")
                </p>
                <p>
                    Enter Age: @Html.TextBox("age")
                    @Html.ValidationMessage("age")
                </p>
                <p>
                    Enter Email: @Html.TextBox("email")
                    @Html.ValidationMessage("email")
                </p>
                <input type="Submit" value="Submit"/>
                @Html.Raw(ViewData["SuccessMessage"])
}
```

Output:


localhost:44383/Home/Index
localhost:44383 says
Data has been submitted
OK


Index - My ASP.NET Application
localhost:44383/Home/Index
Application name
Index
Enter Name:
FullName is required
Enter Age:
Age is required
Enter Email:
Email is Required
Submit
© 2023 - My ASP.NET Application

Lab 22: Date: 2080/02/32

Title: Adding error message in Validation

Introduction:

In this lab report, we present the implementation and utilization of two key codes: "@Html.ValidationSummary()" and "<b style="color:red; font-size:12px;"> @ViewData["fullnameerror"]". These codes play a crucial role in enhancing the functionality and visual appeal of web applications developed using HTML and the ASP.NET framework.

The "@Html.ValidationSummary()" code is an essential component of ASP.NET's validation framework. It provides a convenient way to display a summary of validation errors that occur during form submission. By incorporating this code into our application, we can easily consolidate and present validation errors to the user in a concise and user-friendly manner. This aids in improving the overall user experience by ensuring that all necessary error messages are prominently displayed.

On the other hand, the "<b style="color:red; font-size:12px;"> @ViewData["fullnameerror"]" code snippet demonstrates the customization options available for styling error messages. By utilizing HTML markup and CSS styles, we can modify the appearance of error messages to make them more visually impactful. In this specific example, the code applies a red color and adjusts the font size to 12 pixels for the error message associated with the "fullname" field. This helps draw the user's attention to the error and reinforces the importance of rectifying it.

Throughout this lab report, we will showcase how these codes are integrated into a web application to enhance the validation process and improve the overall presentation of error messages. We will discuss the implementation details, demonstrate their functionality, and analyze the impact they have on the user experience. Furthermore, we will highlight any potential considerations and best practices for utilizing these codes effectively.

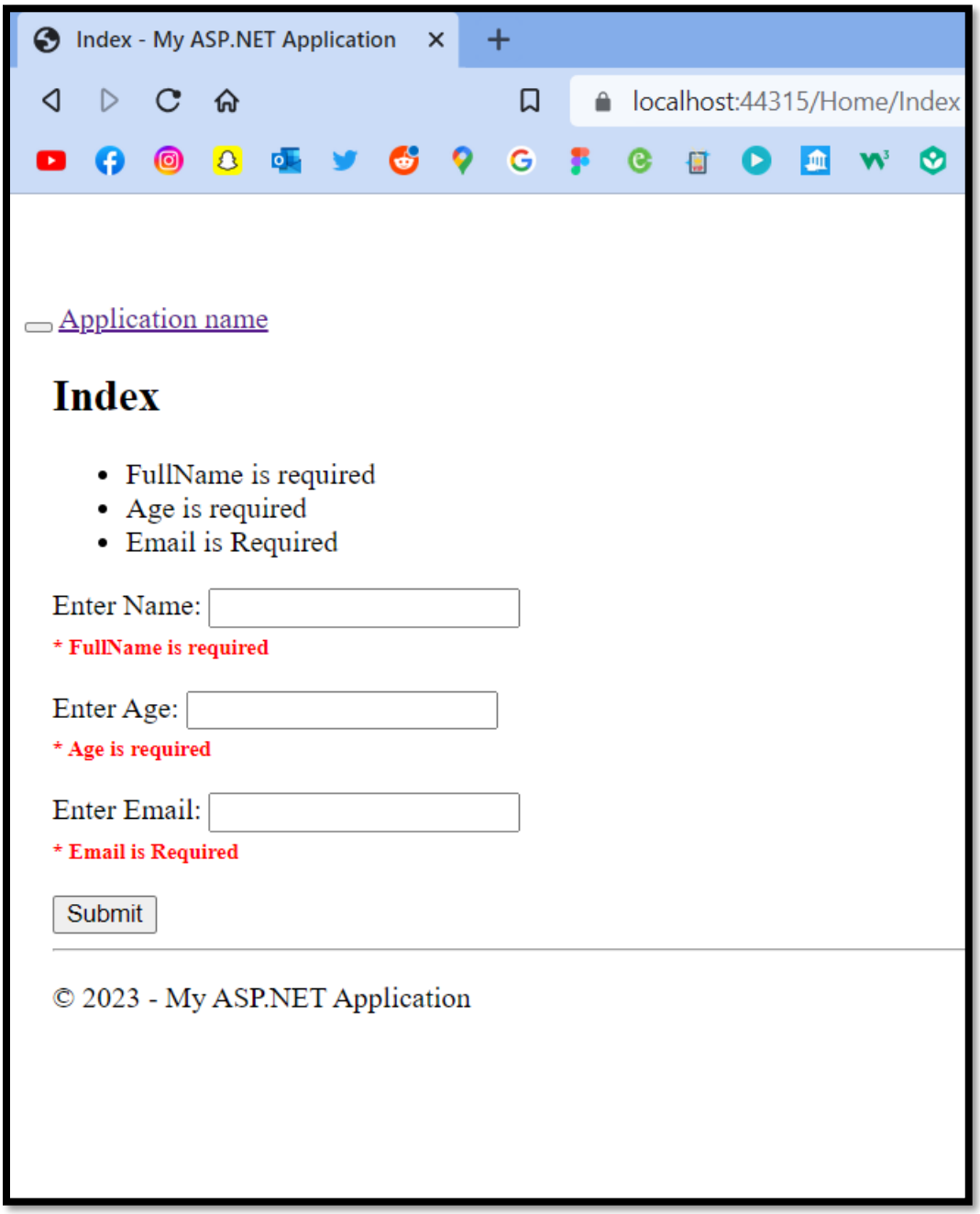
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace Lab_22.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
        [HttpPost]
        public ActionResult Index(string FullName, String
Age, String Email)
        {
            if (FullName.Equals("") == true)
            {

ModelState.AddModelError("fullname","FullName is
required");
                ViewData["fullnameerror"] = "*";
            }
            if (Age.Equals("") == true)
            {
                ModelState.AddModelError("age", "Age is
required");
                ViewData["ageerror"] = "*";
            }
            if (Email.Equals("") == true)
            {
                ModelState.AddModelError("email", "Email
is Required");
                ViewData["emailerror"] = "*";
            }
            if (ModelState.IsValid == true)
            {
                ViewData["SuccessMessage"] =
               "<script>alert('Data has been
submitted')</script>";
                ModelState.Clear();
            }
            return View();
        }
    }
}
```

View Code:

```
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ValidationSummary()
</p>
@using (Html.BeginForm())
{
    <p>
        Enter Name: @Html.TextBox("fullname", "", new
{@class = "form-control"})
        <br/>
        <b style="color:red; font-size:12px;">
            @ViewData["fullnameerror"]
                @Html.ValidationMessage("fullname")
            </b>
    </p>
    <p>
        Enter Age: @Html.TextBox("age", "", new { @class =
"form-control" })
        <br/>
        <b style="color:red; font-size:12px;">
            @ViewData["ageerror"]
                @Html.ValidationMessage("age")
            </b>
    </p>
    <p>
        Enter Email: @Html.TextBox("email", "", new {
@class = "form-control" })
        <br/>
        <b style="color:red; font-size:12px;">
            @ViewData["emailerror"]
                @Html.ValidationMessage("email")
            </b>
    </p>
    <input type="Submit" value="Submit" />
    @Html.Raw(ViewData["SuccessMessage"])
}
```

Output:


Index - My ASP.NET Application
localhost:44315/Home/Index
Application name
Index
FullName is required
Age is required
Email is Required
Enter Name:
* FullName is required
Enter Age:
* Age is required
Enter Email:
* Email is Required
Submit
© 2023 - My ASP.NET Application

Lab 23: Date: 2080/02/32

Title: Use of Regular Expression for Validation

Introduction:

This lab report aims to showcase the implementation of regular expressions (regex) in a Razor MVC application. Regular expressions are powerful tools for pattern matching and validation in various programming languages. In this project, we have utilized two specific regex patterns, one for email validation and another for password validation.

Regular expressions provide a concise and flexible way to define and search for patterns within strings. They consist of a combination of symbols and characters that define a specific pattern to match against. By utilizing regex, developers can enforce specific rules and constraints on user input, ensuring that data entered into a system meets the required format.

The email regex pattern used in this lab report is: "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$". This pattern validates email addresses by checking for the presence of alphanumeric characters, dots, underscores, hyphens, and percentage symbols in the local part of the email. It also ensures the presence of a domain name consisting of alphanumeric characters, dots, and hyphens, followed by a two-letter or more top-level domain.

The password regex pattern used in this lab report is: "^(?=.[a-z])(?=.[A-Z])(?=.*\d).{8,}$". This pattern validates passwords by enforcing the following rules: at least one lowercase letter, at least one uppercase letter, at least one digit, and a minimum length of eight characters. This pattern helps in creating stronger passwords and enhancing the security of user accounts.

In the following sections of this lab report, we will demonstrate the implementation of these regex patterns in a Razor MVC application. We will illustrate how these patterns can be integrated into the application's form validation logic, ensuring that user input adheres to the defined email and password requirements.

Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using System.Web;
using System.Web.Mvc;
namespace Lab_23.Controllers
{
    public class HomeController : Controller
    {
        string EmailPattern = "^[a-zA-Z0-9._%+-]+@[a-zA-
Z0-9.-]+\\.[a-zA-Z]{2,}$";
        string PasswordPattern = "^(?=.*[a-z])(?=.*[A-
Z])(?=.*\\d).{8,}$";
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
        [HttpPost]
        public ActionResult Index(string FullName,string
Password, string Age, string Email)
        {
            if (FullName.Equals("") == true)
            {

                ModelState.AddModelError("fullname",
"FullName is required");

                 ViewData["fullnameerror"] = "*";
            }
            if (Password.Equals("") == true)
            {

                ModelState.AddModelError("password",
"Password is required");

                ViewData["password"] = "*";
            }
            if (Age.Equals("") == true)
            {
                ModelState.AddModelError("age", "Age is
required");

                ViewData["ageerror"] = "*";
            }
            if (Email.Equals("") == true)
            {
                ModelState.AddModelError("email", "Email
is Required");

                ViewData["emailerror"] = "*";
            }
            if (Regex.IsMatch(Email, EmailPattern) ==
false)
```

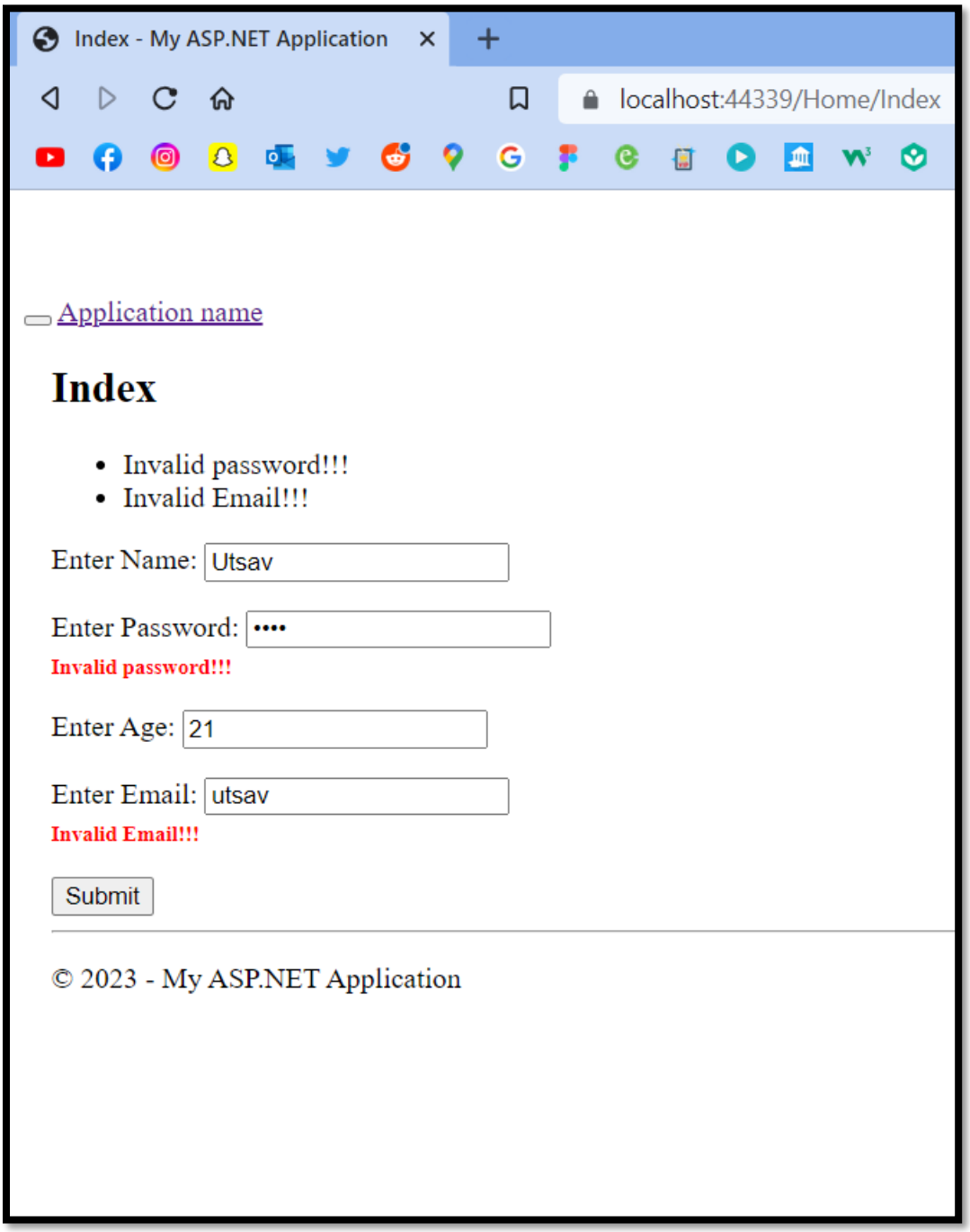
```
    {
                    ModelState.AddModelError("email", "Invalid
    Email!!!");
                }
                if (Regex.IsMatch(Password, PasswordPattern) ==
    false)
                {
                    ModelState.AddModelError("password", "Invalid
    password!!!");
                }
                if (ModelState.IsValid == true)
                {
                    ViewData["SuccessMessage"]
    ="<script>alert('Data has been submitted')</script>";
                ModelState.Clear();
                }
                return View();
            }
        }
    }
```

View Code:

```
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ValidationSummary()
</p>
@using (Html.BeginForm())
{
    <p>
        Enter Name: @Html.TextBox("fullname", "", new { @class
= "form-control" })
        <br />
        <b style="color:red; font-size:12px;">
            @ViewData["fullnameerror"]
            @Html.ValidationMessage("fullname")
        </b>
    </p>
    <p>
        Enter Password: @Html.Password("password", "", new {
@class = "form-control" })
        <br />
        <b style="color:red; font-size:12px;">
            @ViewData["password"]
            @Html.ValidationMessage("password")
        </b>
    </p>
    <p>
        Enter Age: @Html.TextBox("age", "", new { @class =
"form-control" })
```

```
    <br />
            <b style="color:red; font-size:12px;">
                @ViewData["ageerror"]
                @Html.ValidationMessage("age")
            </b>
        </p>
        <p>
            Enter Email: @Html.TextBox("email", "", new { @class
    = "form-control" })
            <br />
            <b style="color:red; font-size:12px;">
                @ViewData["emailerror"]
                @Html.ValidationMessage("email")
            </b>
        </p>
        <input type="Submit" value="Submit" />
        @Html.Raw(ViewData["SuccessMessage"])
    }
```

Output:


Index - My ASP.NET Application
localhost:44339/Home/Index
Application name
Index
Invalid password!!!
Invalid Email!!!
Enter Name: Utsav
Enter Password: ••••
Invalid password!!!
Enter Age: 21
Enter Email: utsav
Invalid Email!!!
Submit
© 2023 - My ASP.NET Application

Lab 24: Date: 2080/03/01

Title: Data Annotation

Introduction:

Data annotation is a critical component of data processing and analysis, and its significance extends to various programming languages, including C#. In the realm of C# development, data annotation plays a crucial role in ensuring data integrity, validation, and overall quality. By leveraging data annotation techniques, developers can enhance the reliability and usability of their C# applications.

One of the primary uses of data annotation in C# is in validating user input. By applying annotations to properties or fields within classes, developers can specify constraints and rules that the data must adhere to. For example, using attributes such as Required, StringLength, or RegularExpression, developers can enforce that certain fields must be filled, set maximum or minimum length limits, or ensure that data matches a specific pattern. These annotations help prevent invalid or inconsistent data from entering the system, improving data quality and minimizing errors.

Furthermore, data annotation in C# facilitates seamless integration with databases and APIs. Many popular data access frameworks and ORMs (Object-Relational Mapping) in C# rely on data annotations to map object properties to database columns or API parameters. By using annotations such as Column, Key, or JsonProperty, developers can specify how data should be serialized, deserialized, or mapped to database tables. This simplifies the process of persisting data or communicating with external services, making data operations more efficient and reducing the amount of manual mapping code.

Data annotation in C# also extends to supporting documentation and code readability. By providing meaningful annotations and comments within the codebase, developers can communicate the intended use, constraints, and expectations of data to fellow developers or future maintainers. This improves code comprehension, reduces ambiguity, and facilitates collaboration within development teams.

Controller Code:

```
using Lab_24.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_24.Controllers
{

    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
        [HttpPost]
        public ActionResult Index(Employee e)
        {
            if(ModelState.IsValid==true)
            {
                ViewData["SuccessMessage"] =
"<script>alert('Data has been submitted!!!')</script>";
                ModelState.Clear();
            }
            return View();
        }
    }
}
```

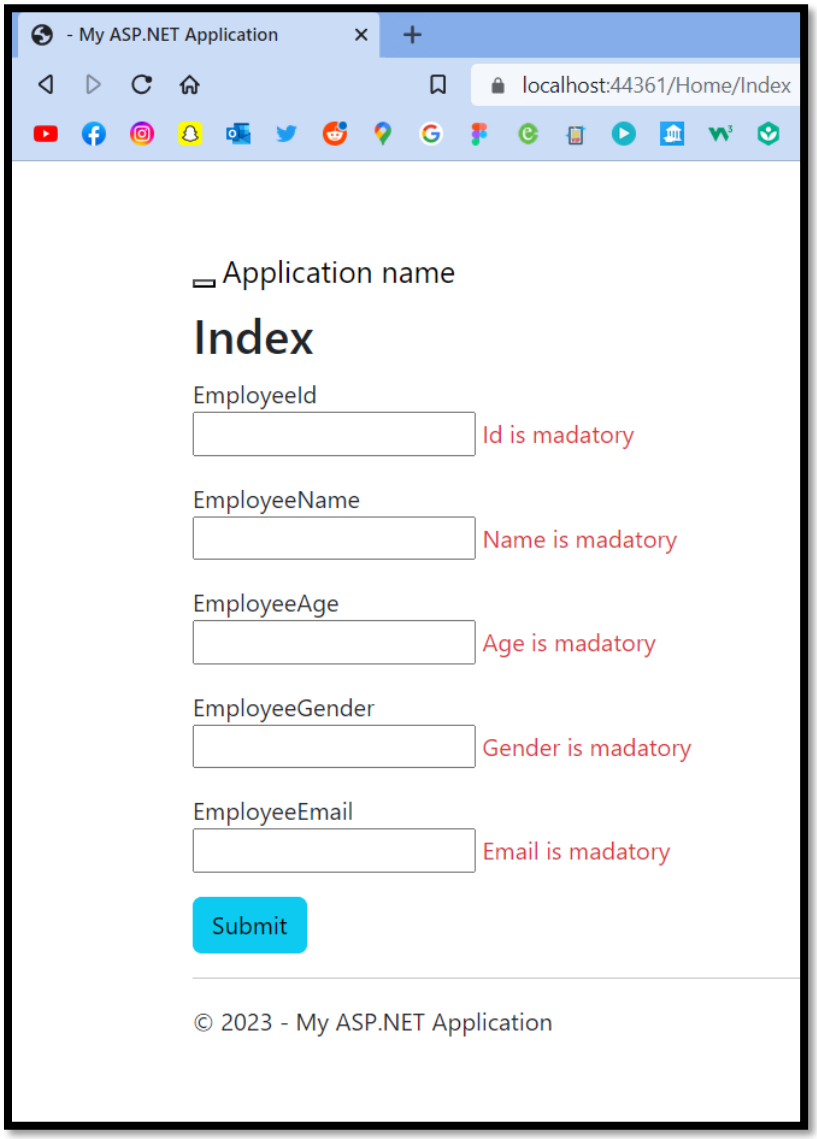
Model Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;
namespace Lab_24.Models
{
    public class Employee{
        [Required(ErrorMessage ="Id is madatory")]
        public int EmployeeId { get; set; }
        [Required(ErrorMessage = "Name is madatory")]
        public string EmployeeName { get; set; }
        [Required(ErrorMessage = "Age is madatory")]
        public int EmployeeAge { get; set; }
        [Required(ErrorMessage = "Gender is madatory")]
        public string EmployeeGender { get; set; }
        [Required(ErrorMessage = "Email is madatory")]
        public string EmployeeEmail { get; set; }
    }
}
```

View Code:

```
@model Lab_24.Models.Employee
@{
    ViewBag.Tilte = "Index";
}
<h2>Index</h2>
@using (Html.BeginForm())
{
    <p>
        @Html.LabelFor(model => model.EmployeeId) <br
/>@Html.TextBoxFor(model => model.EmployeeId, "", new
{@class="format-control"})
        @Html.ValidationMessageFor(Model =>
Model.EmployeeId, "", new { @class = "text-danger" })
    </p>

    <p>
        @Html.LabelFor(model =>
model.EmployeeName) <br />@Html.TextBoxFor(model =>
model.EmployeeName, "", new { @class = "format-control" })
        @Html.ValidationMessageFor(Model =>
Model.EmployeeName, "", new { @class = "text-danger" })
    </p>
    <p>
        @Html.LabelFor(model =>
model.EmployeeAge) <br />@Html.TextBoxFor(model =>
model.EmployeeAge, "", new { @class = "format-control" })
        @Html.ValidationMessageFor(Model =>
Model.EmployeeAge, "", new { @class = "text-danger" })
    </p>
    <p>
        @Html.LabelFor(model =>
model.EmployeeGender) <br />@Html.TextBoxFor(model =>
model.EmployeeGender, "", new { @class = "format-control"
})
        @Html.ValidationMessageFor(Model =>
Model.EmployeeGender, "", new{@class = "text-danger"})
    </p>
    <p>
        @Html.LabelFor(model =>
model.EmployeeEmail) <br />@Html.TextBoxFor(model =>
model.EmployeeEmail, "", new { @class = "format-control"
})
        @Html.ValidationMessageFor(Model =>
Model.EmployeeEmail,"",new { @class = "text-danger" })
    </p>
    <input type="Submit" value="Submit" class="btn btn-
info"/>
    @Html.Raw(ViewData["SuccessMessage"])
}
```

Output:


- My ASP.NET Application
localhost:44361/Home/Index
Application name
Index
EmployeeId
Id is madatory
EmployeeName
Name is madatory
EmployeeAge
Age is madatory
EmployeeGender
Gender is madatory
EmployeeEmail
Email is madatory
Submit
© 2023 - My ASP.NET Application

Lab 25: Date: 2080/03/04

Title: URL Routing

Introduction:

URL routing is a critical aspect of web development that ensures seamless navigation and functionality within web applications. When a user enters a URL or clicks on a link, the URL routing mechanism directs the request to the appropriate destination within the application. This destination can be a specific web page, a resource such as an image or file, or a handler that executes a particular action. To achieve this, developers establish a set of rules and patterns that map incoming URLs to their corresponding endpoints.

URL routing offers numerous benefits in web application development. Firstly, it enables developers to create dynamic and user-friendly applications by allowing users to access different pages or perform specific actions with ease. Instead of relying solely on complex query strings or hard-coded URLs, URL routing provides a more intuitive and readable way to interact with the application. Additionally, by organizing the application's functionality into distinct routes, developers can efficiently manage and maintain the codebase, enhancing the application's scalability and ease of future updates.

Moreover, URL routing contributes to the overall architecture of modern web applications. It promotes the adoption of a modular and decoupled design, where each route or endpoint can be developed and maintained independently. This modular approach allows developers to divide the application's logic into manageable units, facilitating collaboration among team members and ensuring code reusability. Furthermore, URL routing facilitates the implementation of advanced features such as URL parameters, query strings, and route parameters, enabling developers to create flexible and customizable user experiences.

Home Controller:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_25.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
        public ActionResult Student()
        {
            return View();
        }
    }
}
```

Index.cshtml:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        <h1>Student Index Action for URL Routing</h1>
    </div>
</body>
</html>
```

Student.cshtml:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Student</title>
</head>
<body>
    <div>
        <h1>Hello This is Student action for routing2</h1>
    </div>
</body>
</html>
```

StudentController.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace UrlRouting.Controllers
{
    public class StudentController : Controller
    {
        // GET: Student
        public ActionResult StudentIndex()
        {
            return View();
        }
        public ActionResult Index(int? id)
        {
            return Content(id.ToString());
        }
    }
}
```

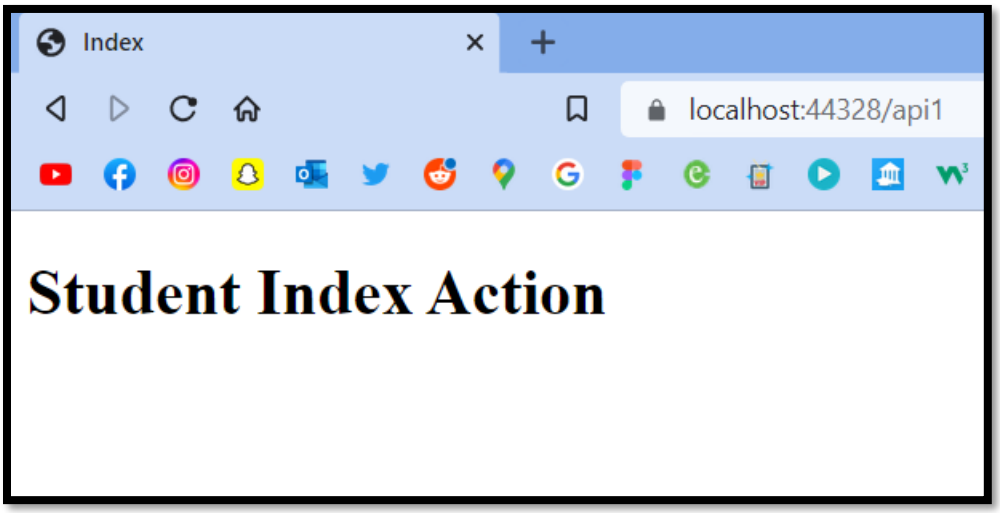
studentindex.cshtml:

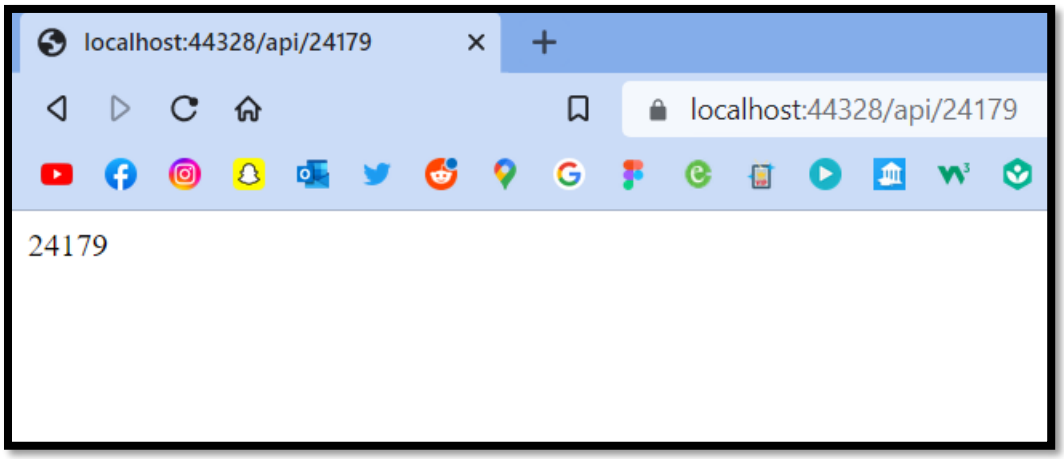
```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        <h1>Student Index Action</h1>
    </div>
</body>
</html>
```

RouteConfig.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace Lab_25
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection
routes)
        {

routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default2",
                url: "api1/",
                defaults: new { controller = "Student",
action = "studentindex", id = UrlParameter.Optional }
            );
            routes.MapRoute(
                name: "Default1",
                url: "api/{id}",
                defaults: new { controller = "Student",
action = "Index", id = UrlParameter.Optional },
constraints: new { id = @"\d+" }
                );
            routes.MapRoute(
               name: "Default",
               url: "{controller}/{action}/{id}",
               defaults: new { controller = "Student",
action = "Index", id = UrlParameter.Optional }
               );
        }
    }
}
```

Output:


Index
localhost:44328/api1
Student Index Action


localhost:44328/api/24179
localhost:44328/api/24179
24179

Lab 26: Date: 2080/03/06

Title: Attribute Based Routing

Introduction:

Attribute-based routing is a powerful and flexible approach to routing and message handling in modern software applications. It enables developers to define routing rules based on specific attributes or properties of incoming messages, allowing for dynamic and granular control over how messages are processed and dispatched. By decoupling routing logic from the underlying infrastructure, attribute-based routing offers increased modularity, extensibility, and adaptability, making it an invaluable tool in building scalable and resilient systems. Whether in the context of microservices architectures, event-driven systems, or distributed applications, attribute-based routing provides a versatile mechanism for efficiently directing messages to the appropriate destinations based on their unique characteristics. This introduction aims to explore the fundamental concepts and benefits of attribute-based routing, shedding light on its practical applications and the advantages it brings to the realm of modern software development.

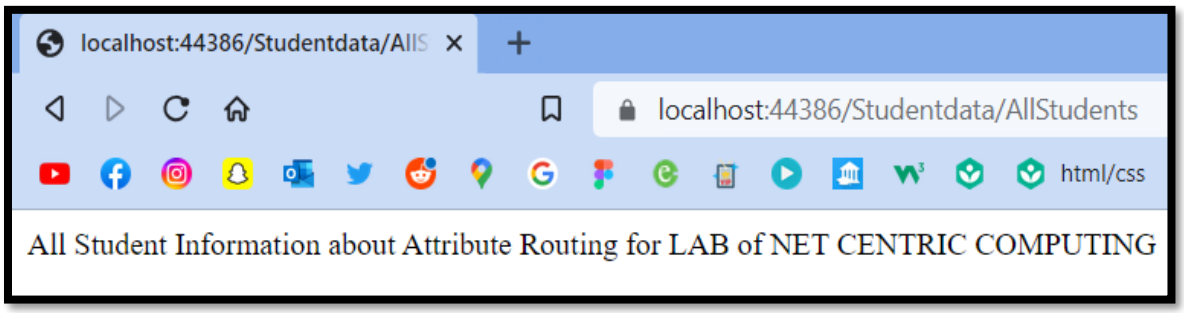
Controller Code:

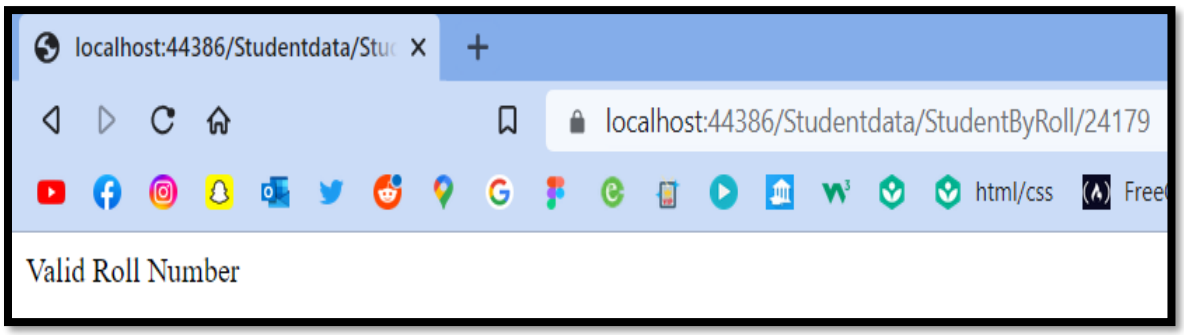
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_26.Controllers
{
    [RoutePrefix("StudentData")]
    public class StudentController : Controller
    {
        // GET: Student
        [Route("AllStudents")]
        public ActionResult Index()
        {
            return Content("All Student Information about
Attribute Routing for LAB of NET CENTRIC COMPUTING");
        }
        [Route("StudentByRoll/{rolln}")]
        public ActionResult studentRollNo(int rolln)
        {
            if(rolln>0)
            {
                return Content("Valid Roll Number");
            }
            else
            {
                return Content("Invalid Roll Number");
            }
        }
    }
}
```

RouteConfig Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace Lab_26
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection
routes)
        {

routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.MapMvcAttributeRoutes();
            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action
= "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

Output:


localhost:44386/Studentdata/AllS
localhost:44386/Studentdata/AllStudents
html/css
All Student Information about Attribute Routing for LAB of NET CENTRIC COMPUTING


localhost:44386/Studentdata/Stu
localhost:44386/Studentdata/StudentByRoll/24179
html/css
Valid Roll Number

Lab 27: Date: 2080/03/07

Title: Creating ASP.NET Web API Using XML & JSON Data Format

Introduction:

In the realm of web development, Application Programming Interfaces (APIs) play a crucial role in facilitating communication and data exchange between different software systems. APIs provide a standardized way for applications to interact, enabling seamless integration and interoperability. As technology continues to advance, developers are constantly seeking efficient and effective methods to build robust APIs that cater to diverse requirements.

This lab report explores the process of developing an ASP.NET Web API using XML and JSON data formats. ASP.NET, a popular web development framework by Microsoft, offers a powerful environment for building scalable and secure web applications. Leveraging the XML (eXtensible Markup Language) and JSON (JavaScript Object Notation) data formats, we aim to create a flexible and versatile API that can handle diverse data structures and satisfy various client needs.

The choice of XML and JSON data formats is motivated by their wide adoption in the industry and their compatibility with a range of programming languages and platforms. XML provides a hierarchical structure for representing data, making it suitable for complex data models. On the other hand, JSON offers a lightweight and human-readable format, making it ideal for web applications that prioritize speed and simplicity.

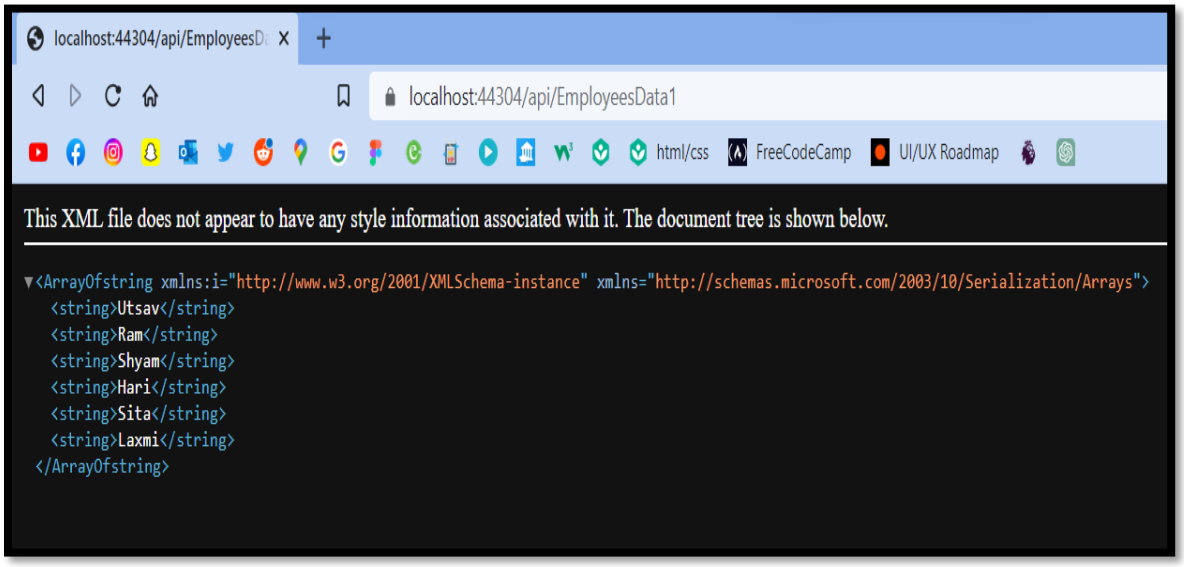
Global.asax Code:

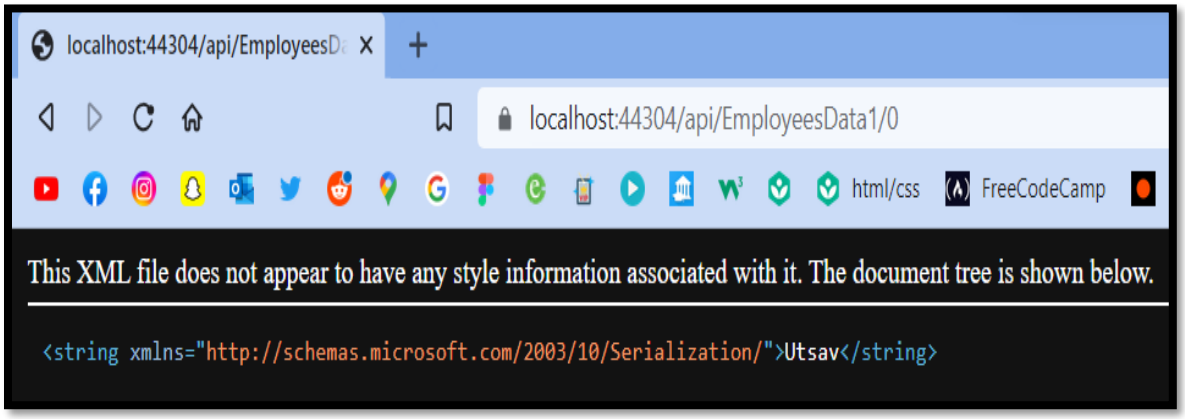
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;
using System.Web.Security;
using System.Web.SessionState;
using System.Web.Http;

namespace Lab_27
{
    public class Global : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            // Code that runs on application startup
            AreaRegistration.RegisterAllAreas();

GlobalConfiguration.Configure(WebApiConfig.Register);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace Lab_27.Controllers
{
    public class EmployeesData1Controller : ApiController
    {
        public string[] myEmployees = {
"Utsav","Ram","Shyam","Hari","Sita","Laxmi" };
        [HttpGet]
        public string[] GetEmployees()
        {
            return myEmployees;
        }
        [HttpGet]
        public string GetEmployeesByIndex(int id)
        {
            return myEmployees[id];
        }
    }
}
```

Output:


localhost:44304/api/EmployeesD
localhost:44304/api/EmployeesData1
html/css
FreeCodeCamp
UI/UX Roadmap
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<ArrayOfstring xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
<string>Utsav</string>
<string>Ram</string>
<string>Shyam</string>
<string>Hari</string>
<string>Sita</string>
<string>Laxmi</string>
</ArrayOfstring>


localhost:44304/api/EmployeesD
localhost:44304/api/EmployeesData1/0
html/css
FreeCodeCamp
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Utsav</string>

Lab 28: Date: 2080/03/11

Title: Entity Framework Code First Approach

Introduction:

The Entity Framework Code First approach has revolutionized the way developers design and interact with databases in modern application development. Traditionally, developers had to start with a preexisting database schema and then build their application models around it. This approach often led to a mismatch between the database design and the application's object-oriented model, resulting in complex mapping configurations and maintenance challenges. However, with Code First, developers can shift their focus to their application's object-oriented models, allowing them to define the domain entities, relationships, and business logic in code without worrying about the underlying database structure.

One of the key advantages of the Code First approach is its ability to seamlessly generate the corresponding database schema from code. Developers can use a set of conventions and configuration options to define how their entities should be mapped to database tables, columns, and relationships. By leveraging these conventions, Code First can automatically generate a database schema that aligns with the defined models. This greatly simplifies the development process and eliminates the need for manual database schema creation and synchronization.

Another significant benefit of Code First is the flexibility it provides in terms of database migrations. As application requirements evolve, developers often need to modify the database schema to accommodate new features or changes in business logic. In traditional database-first approaches, these changes could be time-consuming and error-prone, requiring manual updates to the database schema. However, with Code First, developers can use a concept called "migrations" to manage database schema changes. Migrations allow for easy and automated synchronization between the application models and the database schema, ensuring that the database is always up to date with the latest changes in the codebase.

Connecting Database:
Add the following code in web.config

Syntax:
<connectionStrings>
<add name="[Name_of_connection_string]" connectionString="Data Source = [Database_Server]; Initial Catalog = [Database_Name]; Integrated Security=true;" providerName="System.Data.SqlClient"/>
</connectionStrings>

Package Manager Console Command:

- "Enable-Migration" to enable the migration.
- "Update-Database" to update the database.

Controller Code:

```
using Lab_28.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_28.Controllers
{
    public class HomeController : Controller
    {
        StudentContext db = new StudentContext();
        // GET: Home
        public ActionResult Index()
        {
            var data = db.Student.ToList();
            return View(data);
        }
    }
}
```

View Code:

```
@model IEnumerable<Lab_28.Models.Student>
@{
    ViewBag.Title = "Index";
}

<style>
    table {
        border-collapse: collapse;
        width: 100%;
    }

    th, td {
        border: 1px solid black;
        padding: 8px;
        text-align: left;
    }

    th {
        background-color: #f2f2f2;
    }
</style>

<h2>Index</h2>
<table>
    <tr>
        <th>Id</th>
        <th>Name</th>
        <th>Gender</th>
        <th>Age</th>
    </tr>
    @{
        foreach (var item in Model)
        {
            <tr>
                <td>@item.Id</td>
                <td>@item.Name</td>
                <td>@item.Gender</td>
                <td>@item.Age</td>
            </tr>
        }
    }
</table>
```

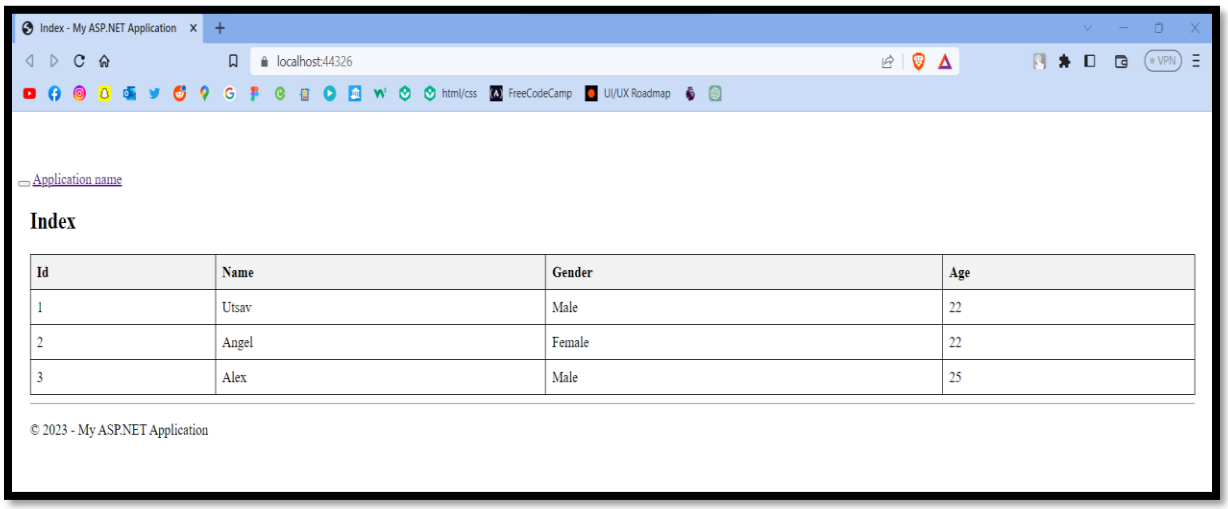
StudentContext.cs Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;
namespace Lab_28.Models
{
    public class StudentContext:DbContext
    {
        public DbSet<Student> Student {  get; set; }
    }
}
```

Student.cs Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Lab_28.Models
{
    public class Student
    {
        [Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        public string Name { get; set; }
        public string Gender { get; set; }
        public int Age { get; set; }
    }
}
```

Output:


Index - My ASP.NET Application
localhost:44326
html/css
FreeCodeCamp
UI/UX Roadmap
Application name
Index
Id
Name
Gender
Age
1
Utsav
Male
22
2
Angel
Female
22
3
Alex
Male
25
© 2023 - My ASP.NET Application

Lab 29: Date: 2080/03/12

Title: Entity Framework Database First Approach

Introduction:

The Database First approach is an alternative method to designing and interacting with databases in modern application development. Unlike the Code First approach, which starts with defining application models and generates the database schema accordingly, the Database First approach begins with an existing database schema and builds application models around it.

Traditionally, developers would first have a preexisting database schema, often created by database administrators or through database design tools. This schema represents the structure of the database, including tables, columns, relationships, and constraints. With the Database First approach, developers then generate or import these database schema definitions into their application development environment.

Once the database schema is available in the development environment, developers can use various tools and frameworks, such as Entity Framework, to generate corresponding application models from the schema. These models represent the entities, relationships, and business logic of the application and are typically defined using object-oriented programming concepts.

To interact with the database using the Database First approach, developers can leverage Object-Relational Mapping (ORM) frameworks like Entity Framework. These frameworks provide features and APIs to map the application models to the database schema. Developers typically configure the mapping manually, specifying how entities, properties, and relationships in the application models correspond to tables, columns, and associations in the database. One of the advantages of the Database First approach is the ability to work directly with an existing database schema.

Home Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_29.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewBag.Title = "Home Page";

            return View();
        }
    }
}
```

Values Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace Lab_29.Controllers
{
    public class ValuesController : ApiController
    {
        // GET api/values
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        public string Get(int id)
        {
            return "value";
        }

        // POST api/values
        public void Post([FromBody] string value)
        {
        }

        // PUT api/values/5
        public void Put(int id, [FromBody] string value)
        {
        }
```

```
        // DELETE api/values/5
        public void Delete(int id)
        {
        }
    }
}
```

Employees Controller Code:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Web;
using System.Web.Mvc;
using Lab_29;

namespace Lab_29.Controllers
{
    public class EmployeesController : Controller
    {
        private utsavEntities db = new utsavEntities();

        // GET: Employees
        public ActionResult Index()
        {
            return View(db.Employees.ToList());
        }

        // GET: Employees/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Employee employee = db.Employees.Find(id);
            if (employee == null)
            {
                return HttpNotFound();
            }
            return View(employee);
        }

        // GET: Employees/Create
        public ActionResult Create()
        {
            return View();
        }
```

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create([Bind(Include =
"Id,Name,Gender,Age,Designation,Salary")] Employee employee)
        {
            if (ModelState.IsValid)
            {
                db.Employees.Add(employee);
                db.SaveChanges();
                return RedirectToAction("Index");
            }

            return View(employee);
        }

        // GET: Employees/Edit/5
        public ActionResult Edit(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Employee employee = db.Employees.Find(id);
            if (employee == null)
            {
                return HttpNotFound();
            }
            return View(employee);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit([Bind(Include =
"Id,Name,Gender,Age,Designation,Salary")] Employee employee)
        {
            if (ModelState.IsValid)
            {
                db.Entry(employee).State =
EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(employee);
        }

        // GET: Employees/Delete/5
        public ActionResult Delete(int? id)
        {

    if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Employee employee = db.Employees.Find(id);
            if (employee == null)
```

```
            {
                return HttpNotFound();
            }
            return View(employee);
        }

        // POST: Employees/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public ActionResult DeleteConfirmed(int id)
        {
            Employee employee = db.Employees.Find(id);
            db.Employees.Remove(employee);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

Employee.cs Code:

```
namespace Lab_29
{
    public class Employee
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Gender { get; set; }
        public int Age { get; set; }
        public string Designation { get; set; }
        public int Salary { get; set; }
    }
}
```

Index.cshtml Code:

```
@model IEnumerable<Lab_29.Employee>

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Gender)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Age)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Designation)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Salary)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Gender)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Age)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Designation)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Salary)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id
}) |
            @Html.ActionLink("Details", "Details", new {
id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new {
id=item.Id })
```

```
        </td>
    </tr>
}

</table>
```

Create.cshtml Code:

```
@model Lab_29.Employee

@{
    ViewBag.Title = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Create</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Employee</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class =
"text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Id, htmlAttributes:
new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Id, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Id,
"", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Name,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Name, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Gender,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Gender, new {
htmlAttributes = new { @class = "form-control" } })
```

```
                @Html.ValidationMessageFor(model =>
model.Gender, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Age,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Age, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Age, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Designation,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Designation,
new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Designation, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Salary,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Salary, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Salary, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create"
class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Delete.cshtml Code:

```
@model Lab_29.Employee

@{
    ViewBag.Title = "Delete";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Employee</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Gender)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Gender)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Age)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Age)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Designation)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Designation)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Salary)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Salary)
        </dd>
```

```
    </dl>

    @using (Html.BeginForm()) {
        @Html.AntiForgeryToken()

        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn
btn-default" /> |
            @Html.ActionLink("Back to List", "Index")
        </div>
    }
</div>
```

Details.cshtml Code:

```
@model Lab_29.Employee

@{
    ViewBag.Title = "Details";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Details</h2>

<div>
    <h4>Employee</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Gender)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Gender)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Age)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Age)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Designation)
        </dt>
```

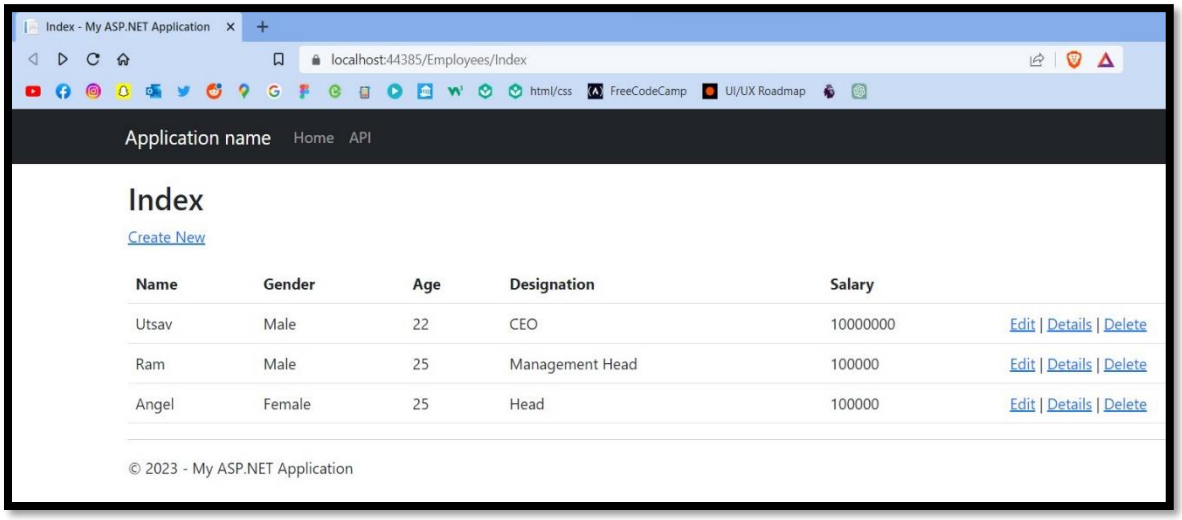
```
        <dd>
            @Html.DisplayFor(model => model.Designation)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Salary)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Salary)
        </dd>

    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "Edit", new { id = Model.Id }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

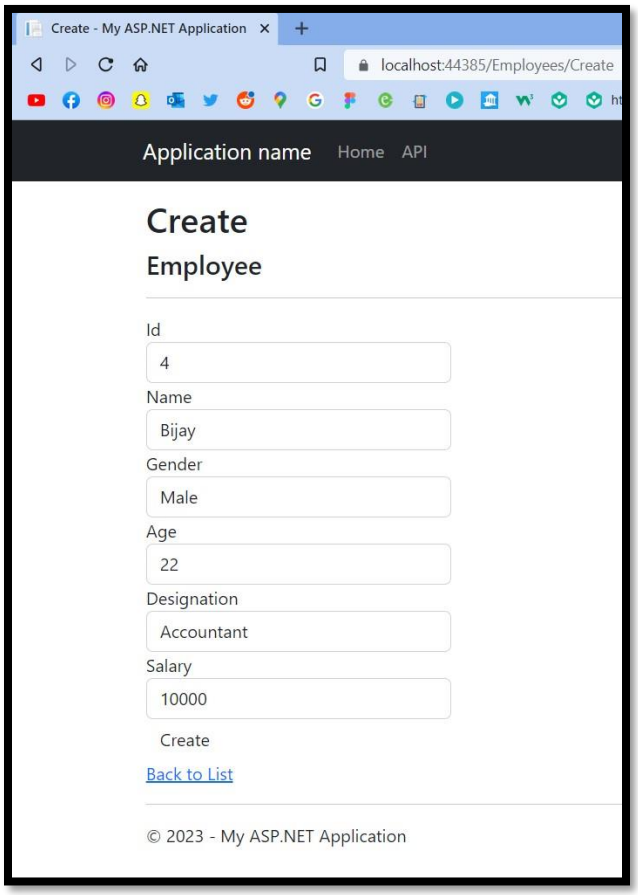
Edit.cshtml Code:

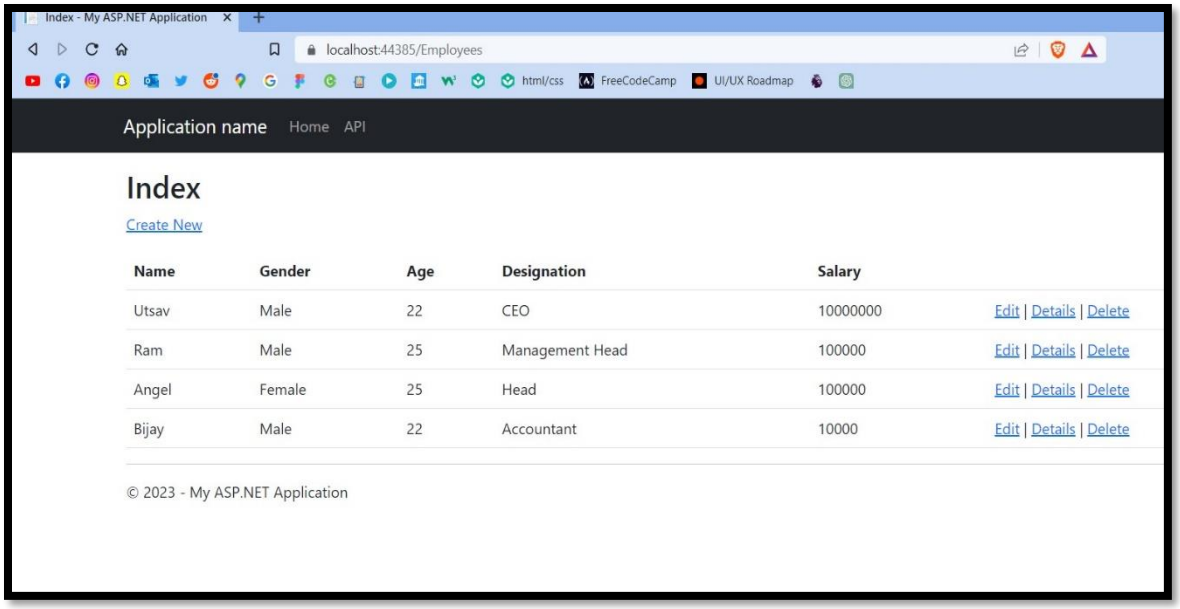
```
@model Lab_29.Employee

@{
    ViewBag.Title = "Edit";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Edit</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Employee</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class =
"text-danger" })
        @Html.HiddenFor(model => model.Id)

        <div class="form-group">
            @Html.LabelFor(model => model.Name,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Name, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Gender,
htmlAttributes: new { @class = "control-label col-md-2" })
```

```
            <div class="col-md-10">
                @Html.EditorFor(model => model.Gender, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Gender, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Age,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Age, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Age, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Designation,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Designation,
new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Designation, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Salary,
htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Salary, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model =>
model.Salary, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn
btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

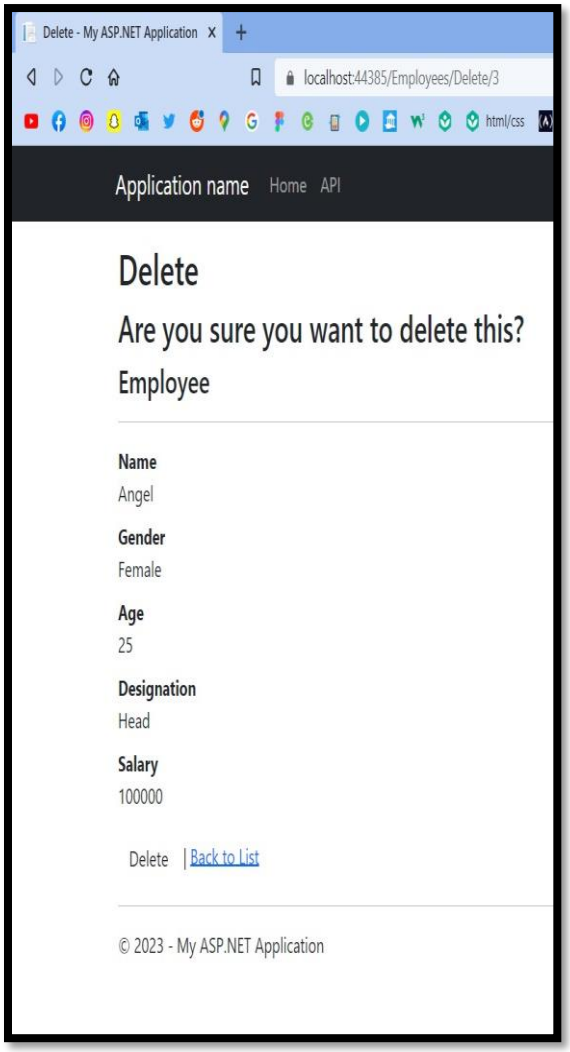
Output:


Index - My ASP.NET Application
localhost:44385/Employees/Index
html/css
FreeCodeCamp
UI/UX Roadmap
Application name
Home
API
Index
Create New
Name
Gender
Age
Designation
Salary
Utsav
Male
22
CEO
10000000
Edit | Details | Delete
Ram
Male
25
Management Head
100000
Edit | Details | Delete
Angel
Female
25
Head
100000
Edit | Details | Delete
© 2023 - My ASP.NET Application


Create - My ASP.NET Application
localhost:44385/Employees/Create
Application name
Home
API
Create
Employee
Id
4
Name
Bijay
Gender
Male
Age
22
Designation
Accountant
Salary
10000
Create
Back to List
© 2023 - My ASP.NET Application


Index - My ASP.NET Application
localhost:44385/Employees
Application name
Home
API
Index
Create New
Name
Gender
Age
Designation
Salary
Utsav
Male
22
CEO
10000000
Edit | Details | Delete
Ram
Male
25
Management Head
100000
Edit | Details | Delete
Angel
Female
25
Head
100000
Edit | Details | Delete
Bijay
Male
22
Accountant
10000
Edit | Details | Delete
© 2023 - My ASP.NET Application


Details - My ASP.NET Application
localhost:44385/Employees/Details/2
Application name
Home
API
Details
Employee
Name
Ram
Gender
Male
Age
25
Designation
Management Head
Salary
100000
Edit | Back to List
© 2023 - My ASP.NET Application


Delete - My ASP.NET Application
localhost:44385/Employees/Delete/3
Application name
Home
API
Delete
Are you sure you want to delete this?
Employee
Name
Angel
Gender
Female
Age
25
Designation
Head
Salary
100000
Delete | Back to List
© 2023 - My ASP.NET Application

Lab 30: Date: 2080/03/12

Title: Temp Data

Introduction:

In modern web development, it is crucial to handle and transfer data effectively between different components of an application. One of the key challenges is maintaining data across multiple requests and actions. ASP.NET MVC provides a powerful mechanism called TempData to address this requirement.

The purpose of this lab report is to showcase the implementation of TempData in an MVC Web API. TempData is a container provided by ASP.NET MVC that allows developers to store and retrieve data between requests. It is particularly useful when we need to pass data from one action to another during a user's session.

In this lab, we will explore the usage of TempData in the context of an MVC Web API. We will examine how TempData can be leveraged to maintain data between actions, redirect requests, and provide a seamless user experience. By utilizing TempData effectively, we can ensure the smooth flow of data and enable efficient communication within our application.

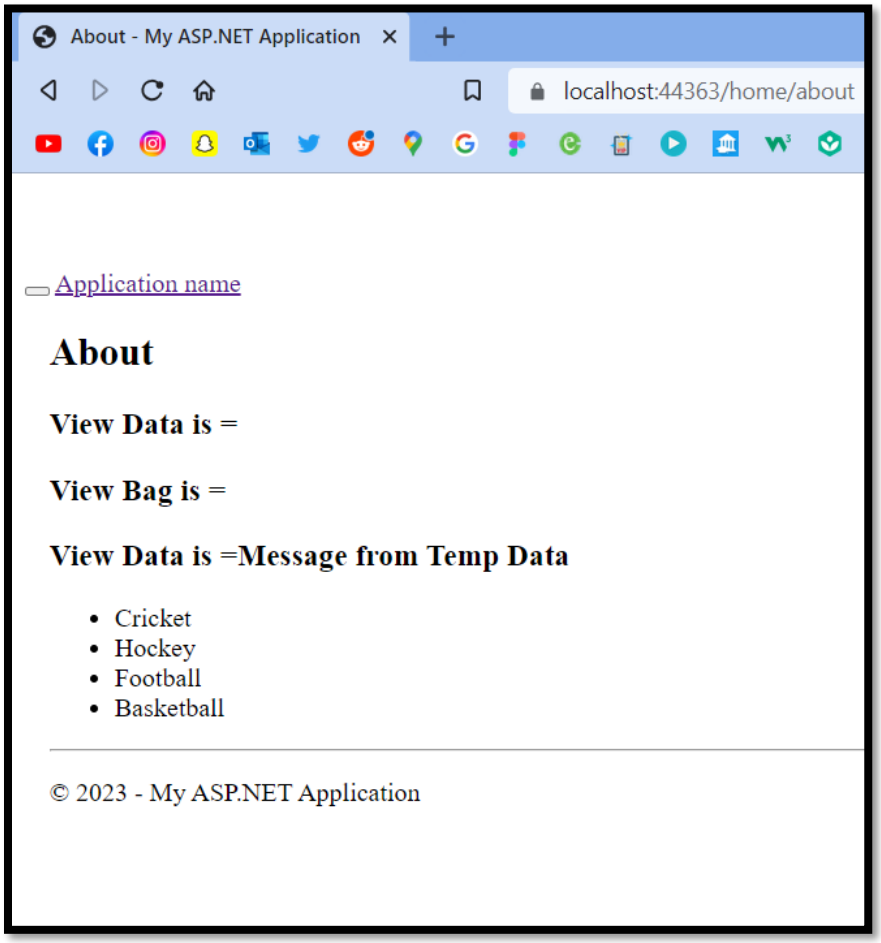
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_30.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            ViewData["Var1"] = "Message from View Data";
            ViewBag.Var2 = "Message from View Bag";
            TempData["Var3"] = "Message from Temp Data";
            string[] games = { "Cricket", "Hockey",
"Football", "Basketball" };
            TempData["GamesArray"] = games;
            return View();
        }
        public ActionResult About()
        {
            return View();
        }
    }
}
```

View Code:

```
@{
    ViewBag.Title = "About";
}

<h2>About</h2>
<h3>View Data is =@ViewData["Var1"]</h3>
<h3>View Bag is =@ViewBag.Var2</h3>
<h3>View Data is =@TempData["Var3"]</h3>
<ul>
    @{
        foreach(string i in (string[])
TempData["GamesArray"])
    {
        <li>@i</li>
    }
    }
</ul>
```

Output:


About - My ASP.NET Application
localhost:44363/home/about
Application name
About
View Data is =
View Bag is =
View Data is =Message from Temp Data
Cricket
Hockey
Football
Basketball
© 2023 - My ASP.NET Application