LAB REPORT OF

Net Centric Computing

CSC 367

Submitted To

Taranath Jaishy

Department of BSc. CSIT

BUTWAL MULTIPLE CAMPUS

Butwal, Rupandehi

Submitted By

Santosh Khatri

BSc. CSIT 6th Sem

Roll No: 25164

Reg.No: 5-2-50-317-2019

Lab_Report_1:

Create a Class "Employee" with following specs:

- *Field Members: firstName, lastName, salary*
- *Properties: FirstName, LastName, Salary*
- *Methods: ShowFullName(), IncrementSalary(double s)*
- *Constructor: Employee (……….,……………..,………….)*
  Now create a object of Employee("Ram","Bahadur",20000)
    - *Show Employee Full Name and Salary.*
    - *Change first name to "Hari" and increment salary by 5000.*
    - *Show full name and salary.*

```
using System;
namespace Lab_1
{
    class Employee
    {
        string firstName=" ", lastName=" ";
        double salary;
        //Method----> ShowFullName
        public void ShowFullName()
        {
            Console.WriteLine("Full Name:"+ " "+firstName+"
"+lastName);
        }
        //Method--->showSalary
        public void showSalary()
        {
            Console.WriteLine("Salary: " + salary);
        }
        //Method----> IncrementSalary
        public void IncrementSalary(double s)
        {
            salary=salary+s;
        }
        // property---> FirstName
        public string FirstName
        {
            get { return firstName; }
            set { firstName = value; }
        }
        // property---> LastName
        public string LastName
        {
            get { return lastName; }
            set { lastName = value; }
        }
        // property---> Salary
        public double Salary
        {
            get { return salary; }
            set { salary = value; }
        }
        //Constructors
        public Employee(string firstName, string lastName, double
salary)
        {
            this.FirstName= firstName;
            this.LastName= lastName;
            this.Salary= salary;

        }
    }
    class Program
    {
        public static void Main(string[] args)
        {
            Employee employee_1=new
Employee("Ram","Bahadur",20000);
            Console.WriteLine("Fullname and Salary is: ");
            employee_1.ShowFullName();
            employee_1.showSalary();

            //Changing first Name to "Hari"
            employee_1.FirstName = "Hari";
           //Incrementing salary by 5000
            employee_1.IncrementSalary(5000);
            employee_1.ShowFullName();
            Console.WriteLine("Salary after increment: ");
            employee_1.showSalary();
            Console.ReadKey();
        }
    }
}
```

Lab_Report_2:

Explain the differences between Errors and Exceptions in C#? Also demonstrate with example.

Aspect	Error	Exception
Definition	Errors are unexpected issues that occur during runtime, often causing a program to terminate abruptly.	Exceptions are issues that occur during runtime, representing abnormal situations that can be handled within the code.
Cause	Errors are usually caused by external factors such as hardware failure, memory issues, or bugs in the compiler or runtime environment.	Exceptions are typically caused by logical errors in the code, like invalid input or incorrect assumptions made by the programmer.
Handling	Errors are hard to handle programmatically since they often indicate severe issues in the environment or the program itself.	Exceptions can be caught and handled using try-catch blocks, allowing the program to gracefully handle unexpected situations.
Termination	Errors usually result in program termination without any chance for recovery or cleanup.	Exceptions provide an opportunity to gracefully exit a specific code block while allowing cleanup operations in the catch or finally blocks.
Examples	Segmentation fault, stack overflow, hardware failure.	DivideByZeroException, NullReferenceException, FileNotFoundException.

Example Code is given Below.

```
using System;

class Program
{
    static void Main()
    {
        // Example of an error
        // This will cause a "System.StackOverflowException" error
        InfiniteRecursion();

        // Example of exception handling
        try
        {
            int result = Divide(10, 0);
            Console.WriteLine("Result: " + result);
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine("Divide by zero exception: " +
ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine("An unexpected exception occurred: "
+ ex.Message);
        }
    }

    static void InfiniteRecursion()
    {
        InfiniteRecursion(); // This will lead to a stack overflow
error
    }

    static int Divide(int a, int b)
    {
        return a / b; // This will throw a DivideByZeroException
if b is 0
    }
}
```

Lab_Report_3:

Explain CRUD functionality using Entity Framework Core in ASP.Net MVC. [Use at least 5 attributes scaffolding (Show process and code for each function.)].

CRUD (Create, Read, Update, and Delete) operations using Entity Framework Core in ASP.NET MVC are fundamental tasks for building web applications. In this example, I'll demonstrate how to implement these operations with at least five attributes for a simple "Task" entity.

Let's start with the entity definition:

1. **Create a Task Entity**

Create a Task class with the necessary attributes. In this example, we'll include attributes such as Id, Title, Description, DueDate, and IsDone.

```
        public class Task
        {
          public int Id { get; set; }
          [Required]
          public string Title { get; set; }
          public string Description { get; set; }
          [DataType(DataType.Date)]
          public DateTime DueDate { get; set; }
          public bool IsDone { get; set; }
}
```

2. **Create DbContext**
   Set up the DbContext class for Entity Framework Core.

```
        public class ApplicationDbContext : DbContext
        {
          public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
        options) : base(options)
          {
          }

          public DbSet<Task> Tasks { get; set; }
}
```

3. **Controller Actions**
   Now, let's create a controller to handle CRUD operations. Scaffold the controller using Entity Framework for the Task model. You can use the following command in the Package Manager Console:

```
Scaffold-Controller -DbContext ApplicationDbContext -ControllerName TasksController -
Model Task
```

   This will generate a TasksController with CRUD actions.

4. **Index Action (Read)**
   In the TasksController, you will find the Index action that retrieves a list of tasks.

```
        public async Task<IActionResult> Index()
        {
          return View(await _context.Tasks.ToListAsync());
}
```

5. **Create Action (Create)**
   The Create action is responsible for adding new tasks to the database.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult>
Create([Bind("Id,Title,Description,DueDate,IsDone")] Task task)
{
   if (ModelState.IsValid)
   {
      _context.Add(task);
      await _context.SaveChangesAsync();
      return RedirectToAction(nameof(Index));
   }
   return View(task);
}
```

6. **Edit Action (Update)**
   The Edit action allows you to update existing tasks.

```
public async Task<IActionResult> Edit(int? id)
{
   if (id == null)
   {
      return NotFound();
   }

   var task = await _context.Tasks.FindAsync(id);
   if (task == null)
   {
      return NotFound();
   }
   return View(task);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("Id,Title,Description,DueDate,IsDone")] Task task)
{
   if (id != task.Id)
   {
      return NotFound();
   }
```

```
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(task);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!TaskExists(task.Id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(task);
}
```

7. **Delete Action (Delete)**

The Delete action allows you to remove tasks from the database.

```
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var task = await _context.Tasks.FirstOrDefaultAsync(m => m.Id == id);
            if (task == null)
            {
                return NotFound();
            }

            return View(task);
        }

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var task = await _context.Tasks.FindAsync(id);
            _context.Tasks.Remove(task);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
}
```

Views for each actions in CRUD

Here's how you can create the appropriate views for the ASP.NET MVC application with CRUD operations using Entity Framework Core.

1. **Index.cshtml (Views/Tasks/Index.cshtml)**

```
@model List<Task>
@{
    ViewData["Title"] = "Task List";
}
<h2>@ViewData["Title"]</h2>
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>Title</th>
            <th>Description</th>
            <th>Due Date</th>
            <th>Is Done</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var task in Model)
        {
            <tr>
                <td>@task.Title</td>
                <td>@task.Description</td>
                <td>@task.DueDate.ToShortDateString()</td>
                <td>@(task.IsDone ? "Yes" : "No")</td>
                <td>
                    <a asp-action="Edit" asp-route-id="@task.Id">Edit</a> |
                    <a asp-action="Details" asp-route-id="@task.Id">Details</a> |
                    <a asp-action="Delete" asp-route-id="@task.Id">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

2. **Create.cshtml (Views/Tasks/Create.cshtml)**

```
@model Task
@{
    ViewData["Title"] = "Create Task";
}
<h2>@ViewData["Title"]</h2>
<form asp-action="Create">
    <div class="form-group">
        <label asp-for="Title"></label>
        <input asp-for="Title" class="form-control" />
        <span asp-validation-for="Title" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Description"></label>
        <textarea asp-for="Description" class="form-control"></textarea>
    </div>
    <div class="form-group">
        <label asp-for="DueDate"></label>
        <input asp-for="DueDate" class="form-control" />
        <span asp-validation-for="DueDate" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="IsDone"></label>
        <input asp-for="IsDone" class="form-control" />
    </div>

    <button type="submit" class="btn btn-primary">Create</button>

    <a asp-action="Index">Back to List</a>

</form>
```

3. Edit.cshtml (Views/Tasks/Edit.cshtml)

```
@model Task
@{
    ViewData["Title"] = "Edit Task";
}
<h2>@ViewData["Title"]</h2>
<form asp-action="Edit">
    <input type="hidden" asp-for="Id" />
    <div class="form-group">
        <label asp-for="Title"></label>
        <input asp-for="Title" class="form-control" />
        <span asp-validation-for="Title" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Description"></label>
        <textarea asp-for="Description" class="form-control"></textarea>
    </div>
    <div class="form-group">
        <label asp-for="DueDate"></label>
        <input asp-for="DueDate" class="form-control" />
        <span asp-validation-for="DueDate" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="IsDone"></label>
        <input asp-for="IsDone" class="form-control" />
    </div>
    <button type="submit" class="btn btn-primary">Save</button>
    <a asp-action="Index">Back to List</a>
</form>
```

4. **Delete.cshtml (Views/Tasks/Delete.cshtml)**

```
@model Task
@{
    ViewData["Title"] = "Delete Task";
}
<h2>@ViewData["Title"]</h2>
```

```
<h3>Are you sure you want to delete this?</h3>
<div>
  <h4>Task</h4>
  <hr />
  <dl class="row">
    <dt class="col-sm-2">Title</dt>
    <dd class="col-sm-10">@Model.Title</dd>
    <dt class="col-sm-2">Description</dt>
    <dd class="col-sm-10">@Model.Description</dd>
    <dt class="col-sm-2">Due Date</dt>
    <dd class="col-sm-10">@Model.DueDate.ToShortDateString()</dd>
    <dt class="col-sm-2">Is Done</dt>
    <dd class="col-sm-10">@Model.IsDone</dd>
  </dl>
  <form asp-action="DeleteConfirmed">
    <input type="hidden" asp-for="Id" />
    <button type="submit" class="btn btn-danger">Delete</button>
    <a asp-action="Index">Back to List</a>
  </form>
</div>
```

Startup.cs

```
public class Startup
{
  // ... Other configurations ...

  public void ConfigureServices(IServiceCollection services)
  {
    services.AddDbContext<ApplicationDbContext>(options =>
      options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddControllersWithViews();
  }

  public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
  {
    // ... Other configurations ...

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
      endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Tasks}/{action=Index}/{id?}");
    });
  }
}
```

Finally connection string in Appsetting.json file.

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\mssqllocaldb;Database=YourDatabaseName;Trusted_Connection=True;Multipl
eActiveResultSets=true"
  },

  // ... Other configuration settings ...
}
```

Lab_Report_4:

Explain ADO.Net, its architecture with example. (With Coding.)

ADO.NET is a data access technology from the Microsoft .NET Framework that provides communication between relational and non-relational systems through a common set of components. ADO.NET is a set of computer software components that programmers can use to access data and data services from a database. It is a part of the base class library that is included with the Microsoft .NET Framework. It is commonly used by programmers to access and modify data stored in relational database systems, though it can also access data in non-relational data sources.

ADO.NET is conceptually divided into consumers and data providers. The consumers are the applications that need access to the data, and the providers are the software components that implement the interface and thereby provide the data to the consumer.

There are two main types of data providers in ADO.NET:

Connection-oriented providers: These providers maintain a connection to the data source for the duration of the data access operation. This is the traditional way of accessing data, and it is still the most efficient way to access data that is frequently updated.

Disconnected providers: These providers do not maintain a connection to the data source for the duration of the data access operation. Instead, they cache the data locally and allow the application to work with the data in a disconnected manner. This is a more efficient way to access data that is not frequently updated.

ADO.NET provides a rich set of components for creating distributed, data-sharing applications. These components include:

Connection objects: These objects are used to establish a connection to a data source.

Command objects: These objects are used to execute queries and stored procedures against a data source.

DataReader objects: These objects are used to read data from a data source one row at a time.

DataSet objects: These objects are used to store data in a disconnected manner.

DataAdapter objects: These objects are used to fill and update DataSet objects.

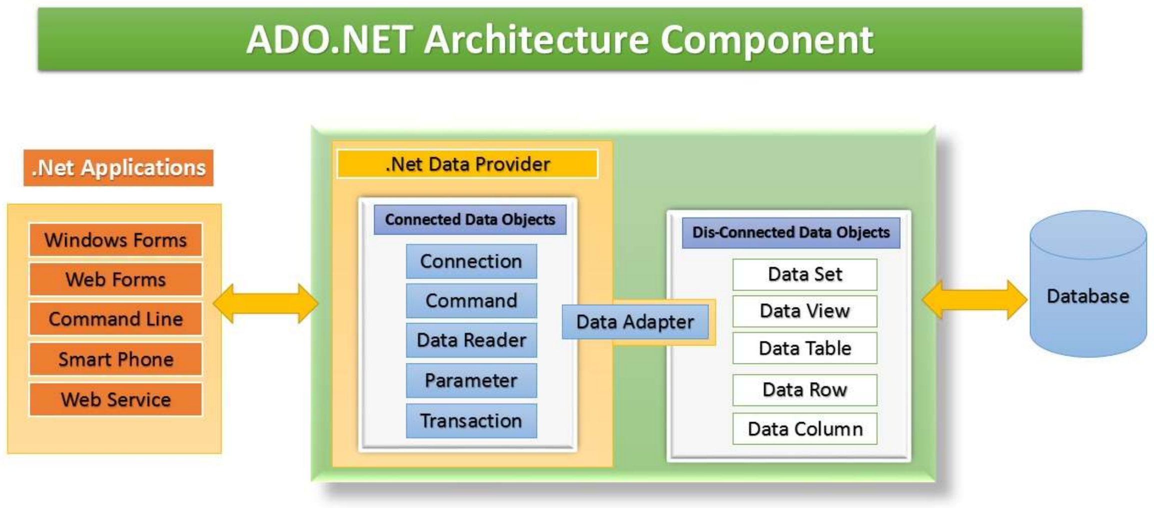
ADO.NET is a powerful and versatile data access technology that can be used to access data from a variety of sources. It is the recommended data access technology for .NET Framework applications.

Here are some of the benefits of using ADO.NET:

Performance: ADO.NET is a high-performance data access technology. It can provide efficient access to data stored in relational database systems.

Flexibility: ADO.NET is a flexible data access technology. It can be used to access data from a variety of sources, including relational database systems, XML files, and Web services.

Ease of use: ADO.NET is an easy-to-use data access technology. It provides a rich set of components that make it easy to connect to data sources, execute queries, and retrieve data.



//Example Code

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace AdoNetCrudExample
{
   class Program
   {
      static void Main(string[] args)
      {
         string connectionString = "";

         using (SqlConnection connection = new SqlConnection(connectionString))
         {
            connection.Open();

            // CREATE - Insert a new product
            InsertProduct(connection, "New Product", 10.99m);

            // READ - Retrieve and display all products
            Console.WriteLine("All Products:");
            DisplayAllProducts(connection);

            // UPDATE - Update a product's price
            UpdateProductPrice(connection, 1, 12.99m);

            // READ - Retrieve and display all products again
            Console.WriteLine("All Products after Update:");
            DisplayAllProducts(connection);

            // DELETE - Delete a product
            DeleteProduct(connection, 2);

            // READ - Retrieve and display all products after deletion
            Console.WriteLine("All Products after Deletion:");
            DisplayAllProducts(connection);
         }
      }

      static void InsertProduct(SqlConnection connection, string productName, decimal price)
      {
         string insertQuery = "INSERT INTO Products (ProductName, Price) VALUES (@ProductName,
@Price)";
         using (SqlCommand command = new SqlCommand(insertQuery, connection))
         {
            command.Parameters.AddWithValue("@ProductName", productName);
            command.Parameters.AddWithValue("@Price", price);
            command.ExecuteNonQuery();
            Console.WriteLine("Product inserted.");
```

```
        }
    }

    static void DisplayAllProducts(SqlConnection connection)
    {
        string selectQuery = "SELECT * FROM Products";
        using (SqlCommand command = new SqlCommand(selectQuery, connection))
        {
            using (SqlDataReader reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    Console.WriteLine($"ID: {reader["ProductID"]}, Name: {reader["ProductName"]}, 
Price: {reader["Price"]}");
                }
            }
        }
    }

    static void UpdateProductPrice(SqlConnection connection, int productId, decimal newPrice)
    {
        string updateQuery = "UPDATE Products SET Price = @Price WHERE ProductID = 
@ProductID";
        using (SqlCommand command = new SqlCommand(updateQuery, connection))
        {
            command.Parameters.AddWithValue("@Price", newPrice);
            command.Parameters.AddWithValue("@ProductID", productId);
            command.ExecuteNonQuery();
            Console.WriteLine("Product updated.");
        }
    }

    static void DeleteProduct(SqlConnection connection, int productId)
    {
        string deleteQuery = "DELETE FROM Products WHERE ProductID = @ProductID";
        using (SqlCommand command = new SqlCommand(deleteQuery, connection))
        {
            command.Parameters.AddWithValue("@ProductID", productId);
            command.ExecuteNonQuery();
            Console.WriteLine("Product deleted.");
        }
    }
  }
}
```

Lab_Report_5:

Validate a contact form using:

a). Javascript

b). Jquery

Contact Form That Will be validated:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Contact Form</title>
</head>
<body style="font-family: Arial, sans-serif; background-color: #f7f7f7; margin: 0; padding: 0;
display: flex; justify-content: center; align-items: center; min-height: 100vh;">

   <div style="background-color: white; padding: 20px; border-radius: 5px; box-shadow: 0 0 10px
rgba(0, 0, 0, 0.1); max-width: 400px; width: 100%;">

      <h1 style="margin-bottom: 20px;">Contact Form</h1>

      <form id="contactForm" action="submit.php" method="post">
         <label for="name">Your Name:</label>
         <input type="text" id="name" name="name" style="width: 100%; padding: 8px; margin-
bottom: 10px; border: 1px solid #ccc; border-radius: 3px;" required>

         <label for="email">Email Address:</label>
         <input type="email" id="email" name="email" style="width: 100%; padding: 8px; margin-
bottom: 10px; border: 1px solid #ccc; border-radius: 3px;" required>

         <label for="phone">Phone Number:</label>
         <input type="tel" id="phone" name="phone" style="width: 100%; padding: 8px; margin-
bottom: 10px; border: 1px solid #ccc; border-radius: 3px;">

         <label for="subject">Email Subject:</label>
         <input type="text" id="subject" name="subject" style="width: 100%; padding: 8px; margin-
bottom: 10px; border: 1px solid #ccc; border-radius: 3px;">

         <label for="message">Your Message:</label>
         <textarea id="message" name="message" rows="4" style="width: 100%; padding: 8px;
margin-bottom: 10px; border: 1px solid #ccc; border-radius: 3px;" required></textarea>

         <button type="submit" style="background-color: #007bff; color: white; padding: 10px
20px; border: none; border-radius: 3px; cursor: pointer;">Submit</button>
      </form>
```

```
    </div>

</body>
</html>
```

Validation Script Using JavaScript.

```
function validateForm() {
    var name = document.getElementById("name").value;
    var email = document.getElementById("email").value;
    var phone = document.getElementById("phone").value;
    var message = document.getElementById("message").value;

    if (name === "" || email === "" || message === "") {
        alert("Please fill in all required fields.");
        return false;
    }

    if (!isValidEmail(email)) {
        alert("Please enter a valid email address.");
        return false;
    }

    return true;
}

function isValidEmail(email) {
    var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return emailPattern.test(email);
}
```

Validation Script Using JQuery.

```
$(document).ready(function() {
    $("#contactForm").submit(function(event) {
        var name = $("#name").val();
        var email = $("#email").val();
        var message = $("#message").val();

        if (name === "" || email === "" || message === "") {
            alert("Please fill in all required fields.");
            event.preventDefault();
        } else if (!isValidEmail(email)) {
            alert("Please enter a valid email address.");
            event.preventDefault();
        }
    });
});

function isValidEmail(email) {
    var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return emailPattern.test(email);
}
```

---End of Report---