SE Lab 4 Contents (3-Tiers with WebForm)

I. Class Activity 2
Exercise 1: Setting up 3-Tier Architecture with Web Form 2
Step 1: Set Up Your Project 2
Step 2: Create the Data Access Layer (DAL) 2
Step 3: Create the Business Logic Layer (BLL) 4
Step 4: Create the Presentation Layer (PL) with CRUD Operations 6
Step 5: Configure web.config 9
Step 6: Create table Customer with MSSQL 10
Step 7: Run the Application 10
Exercise 2: Login Form with 3-Tier Architecture 12
Exercise 3: Login Form to Main form with 3-Tier Architecture 12
II. Homework 12
III. Lab 3 Report Submission 12

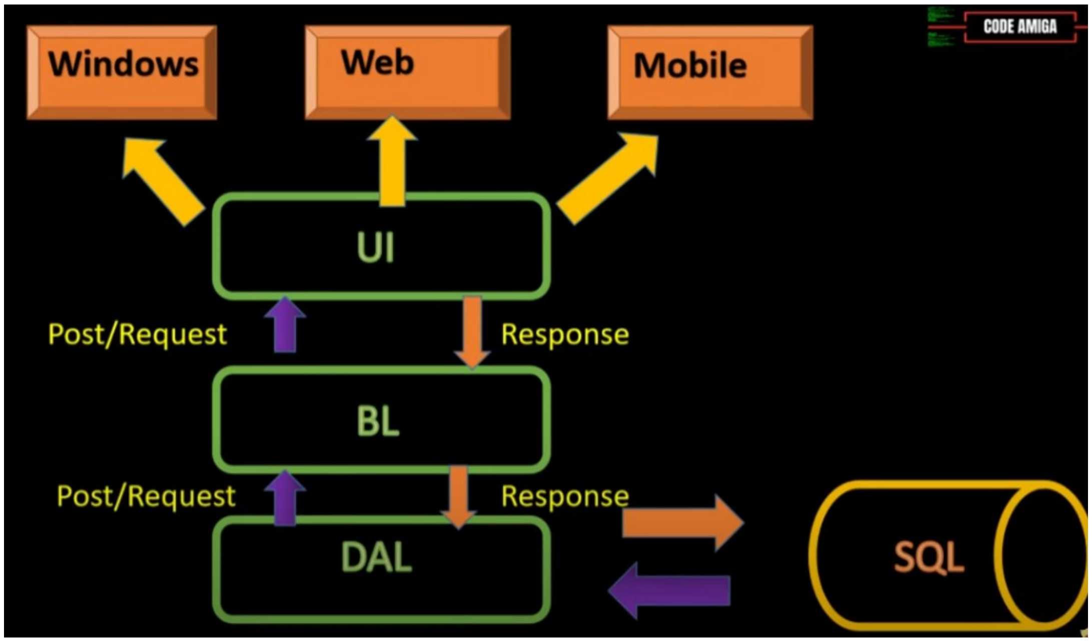


Image Source: https://www.youtube.com/watch?v=P4E9vHYsyAU

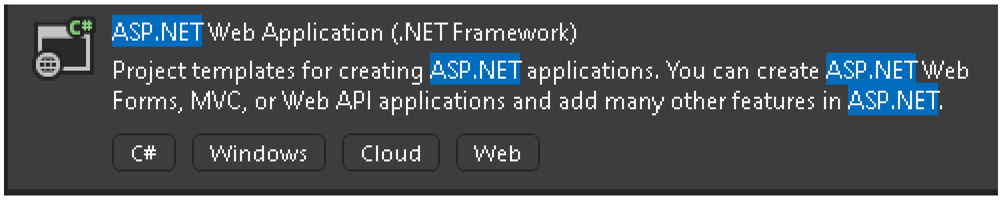
This lab has the same requirements as Lab 3, but while Lab 3 uses WinForms, this lab uses Web Forms.

I. Class Activity

Exercise 1: Setting up 3-Tier Architecture with Web Form

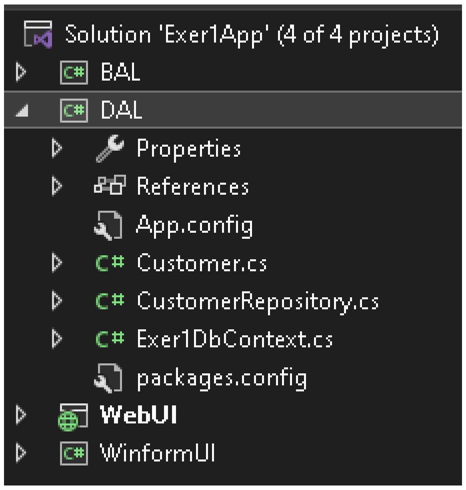
Step 1: Set Up Your Project

1. **Open Visual Studio** and create a new solution.
2. **Add three projects** to your solution:
   - **Data Access Layer (DAL)**: Class Library project.
   - **Business Logic Layer (BLL)**: Class Library project.
   - **Presentation Layer (WebUI)**: ASP.NET Web Application project.



ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Step 2: Create the Data Access Layer (DAL)



```
Solution 'Exer1App' (4 of 4 projects)
  BAL
  DAL
    Properties
    References
    App.config
    Customer.cs
    CustomerRepository.cs
    Exer1DbContext.cs
    packages.config
  WebUI
  WinformUI
```

1. **Install Entity Framework** via NuGet Package Manager.
2. **Add a new class** in the DAL project, e.g., **Customer.cs**, **CustomerRepository**.cs, **DBContext.cs**

   // Customer.cs (Model)

```
namespace DAL
{
    public class Customer
    {
        public int Id { get; set; } //CustomerId
        public string Name { get; set; }
        public string Email { get; set; }
    }
}
```

// CustomerRepository.cs

```
namespace DAL
{
    public class CustomerRepository
    {
        private readonly Exer1DbContext _context;
        public CustomerRepository()
        {
            _context = new Exer1DbContext();
        }
        public void AddCustomer(Customer customer)
        {
            // Code to add customer to the database
            _context.Customers.Add(customer);
            _context.SaveChanges();
        }
        public Customer GetCustomerById(int id)
        {
            // Code to retrieve customer from the database
            //return new Customer(); // Placeholder
            return _context.Customers.FirstOrDefault(c => c.Id == id);
        }
        public List<Customer> GetAllCustomers()
        {
            return _context.Customers.ToList();
        }
        public void UpdateCustomer(Customer customer)
        {
            var existingCustomer = _context.Customers.FirstOrDefault(c
=> c.Id == customer.Id);
            if (existingCustomer != null)
            {
                existingCustomer.Name = customer.Name;
                existingCustomer.Email = customer.Email;
                _context.SaveChanges();
            }
        }
        public void DeleteCustomer(int id)
        {
            var customer = _context.Customers.FirstOrDefault(c => c.Id
== id);
            if (customer != null)
            {
                _context.Customers.Remove(customer);
                _context.SaveChanges();
            }
        }

    }
}
```

Exer1DbContext.cs (DBContext)

```
namespace DAL
{
    public class Exer1DbContext : DbContext
    {
        public DbSet<Customer> Customers { get; set; }
        public Exer1DbContext() : base("name=MyConn")
        { }
    }
}
```

Step 3: Create the Business Logic Layer (BLL)

1. **Add a new class** in the BLL project, e.g., CustomerService.cs.
2. **Reference the DAL project** in the BLL project.

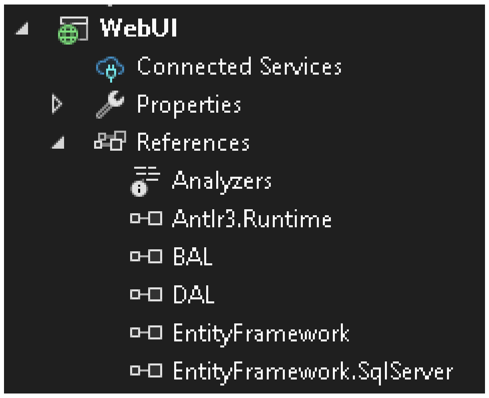
   // CustomerService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DAL;
namespace BAL
{
    public class CustomerService
    {
        private readonly CustomerRepository _customerRepository;
        public CustomerService()
        {
            _customerRepository = new CustomerRepository();
        }
        public void AddCustomer(Customer customer)
        {
            // Business logic before adding customer
            _customerRepository.AddCustomer(customer);
        }
        public Customer GetCustomer(int id)
        {
            // Business logic before retrieving customer
            return _customerRepository.GetCustomerById(id);
        }
        public List<Customer> GetAllCustomers()
        {
            return _customerRepository.GetAllCustomers();
        }
        public void UpdateCustomer(Customer customer)
        {
            // Business logic before updating customer
            _customerRepository.UpdateCustomer(customer);
        }
        public void DeleteCustomer(int id)
        {
            // Business logic before deleting customer
            _customerRepository.DeleteCustomer(id);
        }
    }
}
```

Step 4: Create the Presentation Layer (PL) with CRUD Operations

1. Set Up the Web Application

1. **Create a new ASP.NET Web Forms (.Net Framework) project** in Visual Studio.
2. **Install Entity Framework via NuGet Package Manager.**
3. **Add references** to the BLL and DAL projects.


WebUI
Connected Services
Properties
References
Analyzers
Antlr3.Runtime
BAL
DAL
EntityFramework
EntityFramework.SqlServer

2. Design the Web Page

1. **Open the Default.aspx** page (or create a new Web Form name CustomerWForm.aspx).
2. **Design the UI** for CRUD operations. Add the necessary control and use of Properties for configuration

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="CustomerWForm.aspx.cs" Inherits="WebUI.CustomerWForm" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Customer Management</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2>Customer Management</h2>
            <asp:Label ID="lblId" runat="server" Text="ID:"></asp:Label>
            <asp:TextBox ID="txtId" runat="server"></asp:TextBox><br />
            <asp:Label ID="lblName" runat="server" Text="Name:"></asp:Label>
            <asp:TextBox ID="txtName" runat="server"></asp:TextBox><br />
            <asp:Label ID="lblEmail" runat="server" Text="Email:"></asp:Label>
            <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox><br />
            <asp:Button ID="btnAdd" runat="server" Text="Add" OnClick="btnAdd_Click"
/><br />
            <asp:Button ID="btnGet" runat="server" Text="Get" OnClick="btnGet_Click"
/><br />
```

```
            <asp:Button ID="btnUpdate" runat="server" Text="Update"
OnClick="btnUpdate_Click" /><br />
            <asp:Button ID="btnDelete" runat="server" Text="Delete"
OnClick="btnDelete_Click" /><br />
            <br /><br />
            <asp:GridView ID="gvCustomers" runat="server"
AutoGenerateColumns="true"></asp:GridView>
        </div>
    </form>
</body>
</html>
```

3. Implement Code-Behind Logic

1. **Open the Default.aspx.cs** file (or CustomerWForm.aspx.cs)
2. **Add the necessary using directives**:
3. **Implement the event handlers**:

```
using BAL;
using DAL;
namespace WebUI
{
    public partial class CustomerWForm : System.Web.UI.Page
    {
        private readonly CustomerService _customerService = new
CustomerService();
        /*public CustomerWForm()
        {
            _customerService = new CustomerService();
        }*/
        protected void Page_Load(object sender, EventArgs e)
        {

            if (!IsPostBack)
            {
                BindCustomerGrid();
            }
        }
        private void BindCustomerGrid()
        {
            var customers = _customerService.GetAllCustomers();

            gvCustomers.DataSource = customers;
            gvCustomers.DataBind();
        }
        protected void btnAdd_Click(object sender, EventArgs e)
        {
            var customer = new Customer
            {
                Name = txtName.Text,
                Email = txtEmail.Text
            };
            _customerService.AddCustomer(customer);
            Response.Write("Customer added successfully!");
            //Refresh gvCustomer
            BindCustomerGrid();
        }
        protected void btnGet_Click(object sender, EventArgs e)
        {
            int id = int.Parse(txtId.Text);
            var customer = _customerService.GetCustomer(id);
            if (customer != null)
            {
                txtName.Text = customer.Name;
                txtEmail.Text = customer.Email;
            }
            else
            {
                Response.Write("Customer not found!");
            }
        }
        protected void btnUpdate_Click(object sender, EventArgs e)
        {
            var customer = new Customer
            {
                Id = int.Parse(txtId.Text),
                Name = txtName.Text,
                Email = txtEmail.Text
            };
            _customerService.UpdateCustomer(customer);
```

```
                Response.Write("Customer updated successfully!");
                //Refresh gvCustomer
                BindCustomerGrid();
            }
            protected void btnDelete_Click(object sender, EventArgs e)
            {
                int id = int.Parse(txtId.Text);
                _customerService.DeleteCustomer(id);
                Response.Write("Customer deleted successfully!");
                //Refresh gvCustomer
                BindCustomerGrid();
            }
        }
    }
```

Step 5: Configure web.config

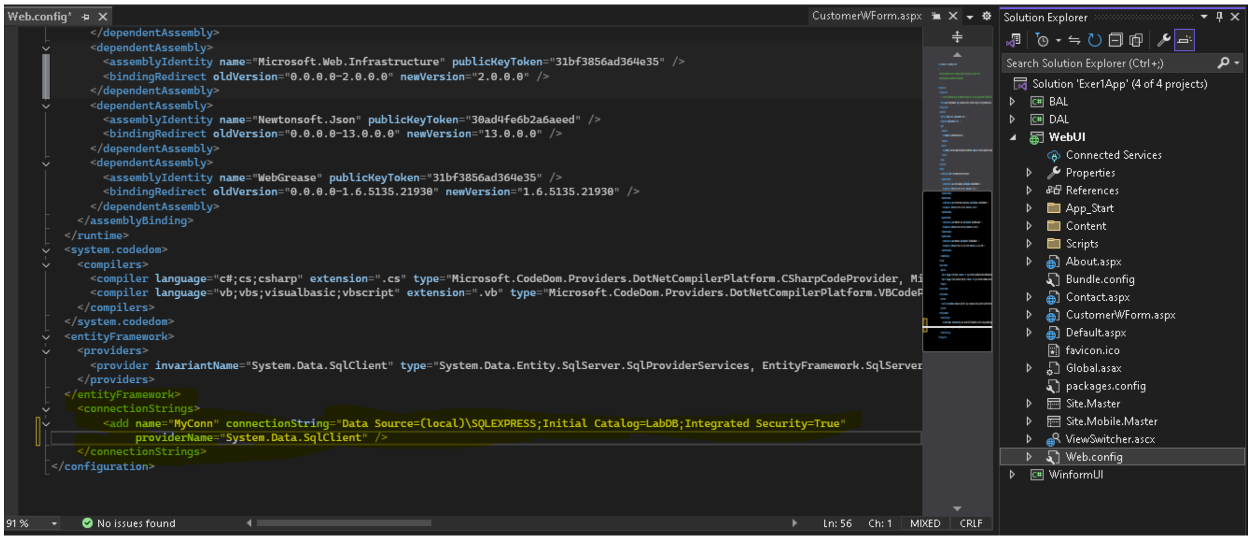
1. **Open web.config** in your ASP.NET project.
2. **Add the connection string** under the <connectionStrings> section.

```
  <connectionStrings>
    <add name="MyConn" connectionString="Data Source=(local)\SQLEXPRESS;Initial
Catalog=LabDB;Integrated Security=True" providerName="System.Data.SqlClient" />
  </connectionStrings>
```

3. **Add the Entity Framework configuration** under the <configSections> and <entityFramework> sections.

   No need if you already installed at Step 4 -2

Example web.config: (just for your reference, you need to be careful). What you need to do just add "connection string" like Step 5.2



```
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.Web.Infrastructure" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="0.0.0.0-2.0.0.0" newVersion="2.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30ad4fe6b2a6aeed" />
        <bindingRedirect oldVersion="0.0.0.0-13.0.0.0" newVersion="13.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="WebGrease" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="0.0.0.0-1.6.5135.21930" newVersion="1.6.5135.21930" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
  <system.codedom>
    <compilers>
      <compiler language="c#;cs;csharp" extension=".cs" type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider, Mi
      <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb" type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeP
    </compilers>
  </system.codedom>
  <entityFramework>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer
    </providers>
  </entityFramework>
    <connectionStrings>
      <add name="MyConn" connectionString="Data Source=(local)\SQLEXPRESS;Initial Catalog=LabDB;Integrated Security=True"
           providerName="System.Data.SqlClient" />
    </connectionStrings>
</configuration>
```

Step 6: Create table Customer with MSSQL

- Create database name : LabDB, create table Customer

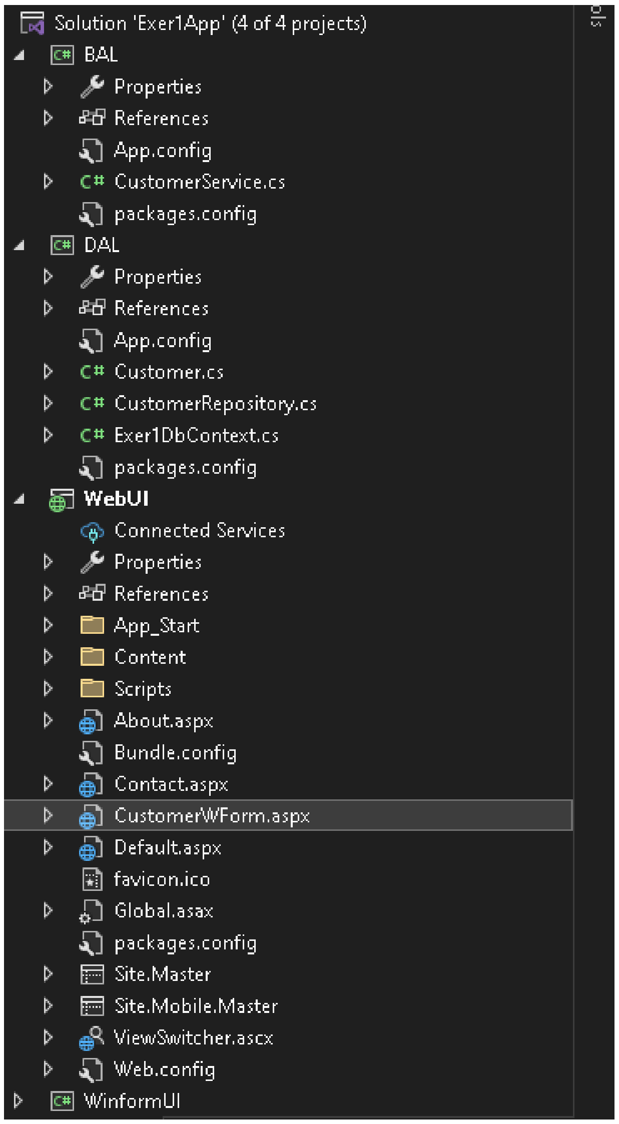
```
CREATE TABLE Customers (
   Id INT PRIMARY KEY IDENTITY(1,1),
   Name NVARCHAR(100),
   Email NVARCHAR(100)
);
```

- Sample data into the Customers table:

```
INSERT INTO Customers (Name, Email) VALUES ('John Doe', 'john.doe@tdtu.com');
INSERT INTO Customers (Name, Email) VALUES ('Jane Smith', 'jane.smith@tdtu.com');
INSERT INTO Customers (Name, Email) VALUES ('Trung Pham', 'trung.pham@tdtu.com');
INSERT INTO Customers (Name, Email) VALUES ('Emily Davis', 'emily.davis@tdtu.com');
INSERT INTO Customers (Name, Email) VALUES ('Thai Pham', 'thai.pham@etdtu.com');
```

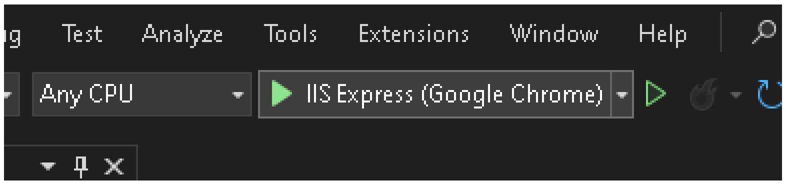
Step 7: Run the Application

1. Project Structure

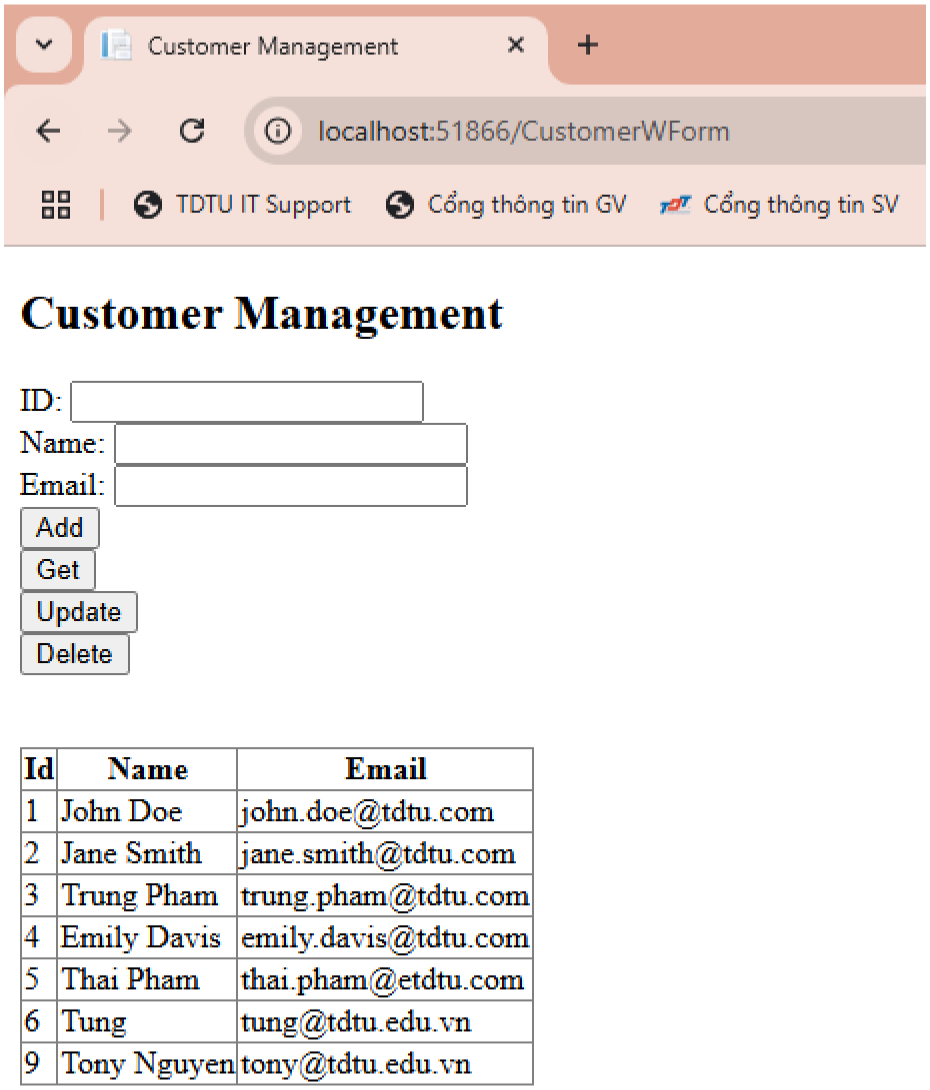


```
Solution 'Exer1App' (4 of 4 projects)
  BAL
    Properties
    References
    App.config
    CustomerService.cs
    packages.config
  DAL
    Properties
    References
    App.config
    Customer.cs
    CustomerRepository.cs
    Exer1DbContext.cs
    packages.config
  WebUI
    Connected Services
    Properties
    References
    App_Start
    Content
    Scripts
    About.aspx
    Bundle.config
    Contact.aspx
    CustomerWForm.aspx
    Default.aspx
    favicon.ico
    Global.asax
    packages.config
    Site.Master
    Site.Mobile.Master
    ViewSwitcher.ascx
    Web.config
  WinformUI
```

2. **Build the solution** to ensure all projects compile successfully:
    - Go to Build > Build Solutio or click on



3. **Run the Windows Forms application** and test the functionality:
    - Set **PresentationLayer (WebUI)** as the startup project.
    - Press F5 to run the application.



Customer Management

localhost:51866/CustomerWForm

TDTU IT Support | Cổng thông tin GV | Cổng thông tin SV

Customer Management

ID:
Name:
Email:

Add
Get
Update
Delete

Id	Name	Email
1	John Doe	john.doe@tdtu.com
2	Jane Smith	jane.smith@tdtu.com
3	Trung Pham	trung.pham@tdtu.com
4	Emily Davis	emily.davis@tdtu.com
5	Thai Pham	thai.pham@etdtu.com
6	Tung	tung@tdtu.edu.vn
9	Tony Nguyen	tony@tdtu.edu.vn

This should give you a basic Web Forms application using a 3-Tier Architecture. You can expand on this by adding more features, implementing dependency injection, and improving error handling.

Exercise 2: Login Form with 3-Tier Architecture

- **Change Login form in Lab2 to 3-Tier Architecture**
- **Should create a new project with setting up from scratch like Exer1**

Exercise 3: Login Form to Main form with 3-Tier Architecture

- **Use Exer2, after logging in successfully then go to Main Form.**
- **On Main form, add menu strip with link to Customer form (reuse Exer 1)**
- **Also add Product Form, Supplier Form**

II. Homework

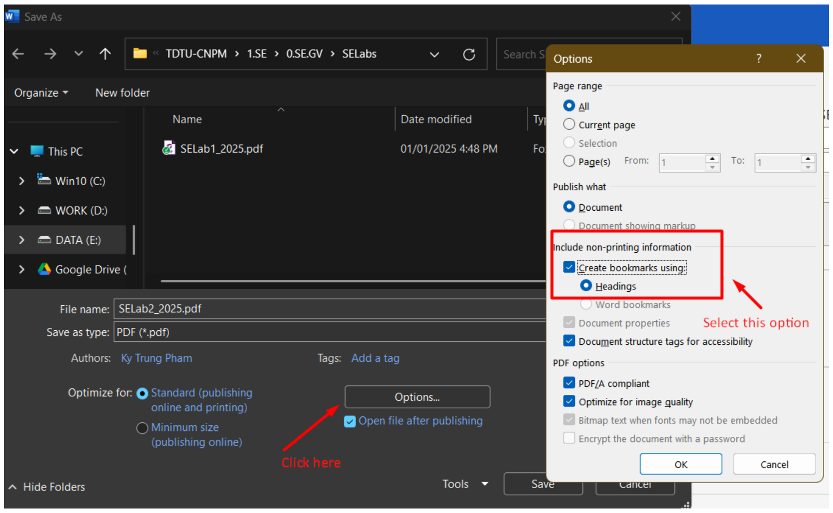
-Use the same requirements from **Lab2 (School Management System)** homework but you should apply 3-Tier Architecture and Web Form.

III. Lab 3 Report Submission

- **Introduction:** Briefly describe the purpose of the exercises.
- **Exercises:** List each exercise with a brief description of what you did and learned for Class Activity included the screenshots of running App output.
- **Homework:** Describe what you do with including the screenshots of the output for homework exercise.
- **Conclusion:** Summarize your overall learning experience and any challenges you faced.

Submit 2 files on eLearning:

- StudentID_StudentName.zip (contain Windows Web Project & .sql file) for both Class Activity and Homework.
- StudentID_StudentName.pdf (require to create bookmarks)



Save As
TDTU-CNPM › 1.SE › 0.SE.GV › SELabs
SELab1_2025.pdf 01/01/2025 4:48 PM
File name: SELab2_2025.pdf
Save as type: PDF (*.pdf)
Authors: Ky Trung Pham Tags: Add a tag
Optimize for: [✓ Standard (publishing online and printing)] Minimum size (publishing online)
[✓ Open file after publishing]

Options
Page range: [✓ All] Current page Selection Page(s) From: 1 To: 1
Publish what: [✓ Document] Document showing markup
Include non-printing information
[✓ Create bookmarks using:] [✓ Headings] Word bookmarks
[✓ Document properties]
[✓ Document structure tags for accessibility]
PDF options
[✓ PDF/A compliant]
[✓ Optimize for image quality]
[✓ Bitmap text when fonts may not be embedded]
Encrypt the document with a password

Notes: There may be a typo/mistake during creating this document. Please correct it by yourself.
Enjoy ☺ Email: tg_phamthaikytrung@tdtu.edu.vn