



**Module Title**            **Designing and Developing Object-Oriented Program**

**Assignment Title**      **Fitness Tracker**

**Examination Cycle**    **Autumn 2024**

**Candidate Name**        **Min Khant Kyaw**

**Candidate No**           **207262**

**Centre Name**           **KMD Computer Centre (Yangon)**

**Submission Date:**      **July 10, 2024**

**Important Notes:**

- ❖ Please refer to the Assignment Presentation Requirements for advice on how to set out your assignment. These can be found on the NCC Education *Campus*. Scroll down the left hand side of the screen until you reach Personal Support. Click on this, and then on Policies and Advice. You will find the Assignment Presentation Requirements under the Advice section.
- ❖ You must familiarise yourself with the NCC Education Academic Dishonesty and Plagiarism Policy and ensure that you acknowledge all the sources which you use in your work. The policy is available on *Campus*. Follow the instructions above, but click on Policies rather than Advice.
- ❖ You must complete the 'Statement and Confirmation of Own Work'. The form is available on the Policies section of *Campus*. Scroll down the left hand side until you reach Personal Support. Click on this and then click on Policies and Advice.
- ❖ Please make a note of the recommended word count. You could lose marks if you write 10% more or less than this.
- ❖ You must submit a paper copy and digital copy (on disk or similarly acceptable medium). Media containing viruses, or media which cannot be run directly, will result in a fail grade being awarded for this module.
- ❖ All electronic media will be checked for plagiarism.

**Marker's comments:**

**Moderator's comments:**

**Mark:**

**Moderated  
Mark:**

**Final  
Mark:**

# Statement and Confirmation of Own Work

***A signed copy of this form must be submitted with every assignment.  
If the statement is missing your work may not be marked.  
For Level 7 assignments, please use the separate Level 7  
Candidate Statement of Own Work form instead of this one.***

## Student Declaration

I confirm the following details:

<b>Candidate Name:</b>	<b>Min Khant Kyaw</b>
<b>Candidate ID Number:</b>	<b>207262</b>
<b>Qualification:</b>	<b>Level 4 Diploma in Computing</b>
<b>Unit:</b>	<b>Designing and Developing Object Oriented Programming</b>
<b>Centre:</b>	<b>KMD Computer Centre (Yangon)</b>
<b>Word Count:</b>	<b>1483</b>
<p>I have read and understood both NCC Education's <i>Academic Misconduct Policy</i> and the <i>Referencing and Bibliographies</i> document. To the best of my knowledge my work has been accurately referenced and all sources cited correctly.</p> <p>I confirm that I have not exceeded the stipulated word limit by more than 10%.</p> <p>I confirm that this is my own work and that I have not colluded or plagiarised any part of it.</p>	
<b>Candidate Signature:</b>	<i>mkhant</i>
<b>Date:</b>	<b>July 10, 2024</b>

## Table of Contents

Introduction .....	4
1. Task – 1.....	4
1.1 Analysis .....	4
1.2 Assumption .....	5
1.3 Coding .....	5
2. Task – 1b.....	71
2.1 Program Structure .....	68
2.2 Quality of Algorithms .....	69
2.3 Readability .....	70
3. Task – 2.....	71
3.1 Types of Testing.....	71
3.2 Testing Plan .....	71
3.2.1 Black Box Test Plan.....	72
3.2.2 White Box Test Plan .....	78
3.2.3 Integration Test Plan.....	78
3.3 Test Script.....	79
Unit Testing .....	79
Integration Testing .....	97
4. Task – 3.....	99
Description of Class Diagram .....	99
References.....	106

## **Introduction**

This program mainly includes two user interfaces for admin and normal users. It offers users to track their activities and this program includes a total of 6 activities that users can keep track of their progress in one place. I added 12 pages for both admins and users with a navigation bar so that software users can easily navigate to other pages and search and do for what they want. I make sure to include test cases and testing plan as well and I fixed the errors to ensure users can have good user experience while using the app. Summing up, this software will be of great help for users to track their fitness activities in daily life.

## **Task – 1**

### **Analysis**

Designing and developing a program involves many steps which are required to do many and plenty research and they have to be based on critical thinking from every possible aspect before building it. This helps to ensure that the program can be successfully completed in the end. I tried to make sure to put a lot of planning and effort went into designing this program before it was built.

First, it is needed to decide the expected outcome of the software like what the program will be used for and who will be using it. The fitness tracker program is focused on helping people monitor and improve their physical and mental health. It also helps keep them in good shape. The program offers a good user experience software with the help of a clear and minimalist graphical user interface for users to interact with it comfortably in their everyday life.

Before writing the program, I researched other available similar programs and especially looked for ideas and noticed what their programs were missing that could be useful for customers. After researching other competitors out there, I started deciding the best ways to build the program. This includes how to sketch and draw the basic design, the prototype, how many pages should be included, what features should be added, and how each part and page should be displayed. To do this, I looked at what design and prototyping tools to use and recommended by most people. Figma seems to be the most popular tool for this program. Designing with Figma offers several advantages as it is easy to use and I can make a prototype using Figma in few hours effectively. I can fully focus on creating visually appealing designs for the program thanks to Figma and start developing it after the designing stage.

## **Assumption**

I added some more activities and also an activity list page for the users to easily visualize and remember the available activities in the program. I also implemented a main page for users to be able to decide whether they want to login as a normal user or as an admin. I finalized with adding a page where users can search their activities with the activity name so that they can see their progress and their activities report.

## **a) Fitness Tracker App Overview**

The Fitness Tracker app is designed to help users stay on track with their fitness goals by providing respective features to monitor their workout routines. The app allows users

to set specific targets and track their progress to ensure they are meeting their objectives at their designated timeline. It consists of two main sections:

1. Admin Panel
2. Customer Panel

## **Admin Panel**

The Admin Panel is where administrators can manage the app's functionalities. Admins can create their own accounts and log in to perform one key feature which is the ability to add new activities to the app's database. These activities are then made available for customers to choose from when setting their fitness goals.

## **Customer Panel**

The Customer Panel is where users can access the app's features. Customers can create their own accounts and log in separately from admins, making the interface easy for them to use. After logging in, customers go to the main page, where they can explore different sections of the app. A key feature is the ability to choose from a list of activities created by admins. Customers can pick up to six activities that fit their fitness goals. Once they select their activities, customers can set personal goals for each one. They can specify how many calories they want to burn in a certain time. The app calculates the target calorie burn based on the activity, time spent, and the user's weight, ensuring that the goals are realistic. When the time is up, customers can check if they reached their calorie-burning goals. The app compares the actual calories burned to the target and shows relevant status on their progress. This helps customers stay motivated and adjust their routines if needed. The Fitness Tracker app is designed to be simple and easy to use, helping customers track their fitness goals and improve their health.


## **All Designs and Code for Fitness Tracker program**

### **1. Admin Register (AdminRegister.cs)**

#### **Admin Register GUI Design**

AdminRegister

### Get Started



Already have an account?

[Login](#)

### Admin Registration

Admin ID	<input type="text" value="Admin-003"/>
Admin Name	<input type="text"/>
Admin Email	<input type="text"/>
Admin Password	<input type="password"/>
Admin Username	<input type="text"/>

## **AdminRegister.cs File Code**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace FitnessTrackerApp
{
    public partial class AdminRegister : Form
    {
        FitnessDataSetTableAdapters.AdminsTableAdapter adminTb = new
        FitnessDataSetTableAdapters.AdminsTableAdapter();

        DataTable dtaadmin = new DataTable();

        public AdminRegister()
        {
            InitializeComponent();
        }

        public void ClearAdminForm()
        {
            adminNameTextBox.Text = "";
            adminEmailTextBox.Text = "";
            adminPasswordTextBox.Text = "";
            adminUserNameTextBox.Text = "";
        }
    }
}
```



```

}

public void AdminAutoID()
{
    DataTable trackdta = new DataTable();
    trackdta = adminTb.GetData();

    if (trackdta.Rows.Count == 0)
    {
        adminIDTextBox.Text = "Admin-001";
    }
    else
    {
        int size = trackdta.Rows.Count - 1;
        string oldid = trackdta.Rows[size][0].ToString();
        int newid = Convert.ToInt16(oldid.Substring(6, 3));

        if (newid >= 1 && newid < 9)
        {
            adminIDTextBox.Text = "Admin-00" + (newid + 1);
        }
        else if (newid >= 9 && newid < 99)
        {
            adminIDTextBox.Text = "Admin-0" + (newid + 1);
        }
        else if (newid >= 99 && newid < 999)
        {
            adminIDTextBox.Text = "Admin-" + (newid + 1);
        }
    }
}

```

```

private void button1_Click(object sender, EventArgs e)
{
    AdminLogin aLogin =new AdminLogin();
    this.Hide();
    aLogin.ShowDialog();
}

private void label1_Click(object sender, EventArgs e)
{
    AdminRegister register = new AdminRegister();
    this.Hide();
    register.ShowDialog();
}

private void AdminRegister_Load(object sender, EventArgs e) { }

private void textBox2_TextChanged(object sender, EventArgs e) { }

private void adminLoginBtn_Click(object sender, EventArgs e)
{
    AdminLogin AdminLoginForm = new AdminLogin();
    this.Hide();
    AdminLoginForm.ShowDialog();
}

private void adminLogoutBtn_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private bool is_validUsername(string username)

```

```

{
    //Regular expression to match only letters and numbers
    string pattern = "^[a-zA-Z0-9]+$";
    Regex regex = new Regex(pattern);
    return regex.IsMatch(username);
}

private void registerBtn_Click(object sender, EventArgs e)
{
    string AdminPassword = adminPasswordTextBox.Text;
    string UserName = adminUserNameTextBox.Text;
    bool isUsernameValid = is_validUsername(adminUserNameTextBox.Text);
    try
    {
        if (adminNameTextBox.Text == "") {
            MessageBox.Show("Admin name must not be empty");
            adminNameTextBox.Focus();
        }
        else if (adminEmailTextBox.Text == "") {
            MessageBox.Show("Admin email must not be empty");
            adminEmailTextBox.Focus();
        }
        else if (adminPasswordTextBox.Text == "")
        {
            MessageBox.Show("Admin password must not be empty");
            adminPasswordTextBox.Focus();
        }

        else if (adminPasswordTextBox.Text.Length < 12)
        {

```

```

        MessageBox.Show("Password length must be at least 12 characters", "Warning",
        MessageBoxButtons.OKCancel, MessageBoxIcon.Error);

        adminPasswordTextBox.Focus();
    }
    else if (!AdminPassword.Any(char.IsUpper))
    {
        MessageBox.Show("Please should include Upper Character");
        adminPasswordTextBox.Focus();
    }
    else if (!AdminPassword.Any(char.IsLower))
    {
        MessageBox.Show("Please should include Lower Character");
        adminPasswordTextBox.Focus();
    }
    else if (!AdminPassword.Any(char.IsDigit))
    {
        MessageBox.Show("Please should include digit Character");
        adminPasswordTextBox.Focus();
    }
    else if (adminUserNameTextBox.Text == "") {
        MessageBox.Show("Admin username must not be empty");
        adminUserNameTextBox.Focus();
    }
    else if (isUsernameValid == false)
    {
        MessageBox.Show("Username is not valid, it must contain only letters and numbers", "Warning",
        MessageBoxButtons.OKCancel, MessageBoxIcon.Error);

        adminUserNameTextBox.Focus();
    }
    else
    {
        // Encapsulation admin

```

```

        ClsAdmin adminObj = new ClsAdmin();
        adminObj.AID = adminIDTextBox.Text;
        adminObj.AName = adminNameTextBox.Text;
        adminObj.AEmail = adminEmailTextBox.Text;
        adminObj.APassword = adminPasswordTextBox.Text;
        adminObj.AUserName = adminUserNameTextBox.Text;

        int insertAdmin = adminTb.AdminRegister(adminObj.AID, adminObj.AName, adminObj.AEmail,
        adminObj.APassword, adminObj.AUserName);

        if (insertAdmin > 0)
        {
            MessageBox.Show("Admin registration completed successfully.");
            ClearAdminForm();
            AdminAutoID();

            AdminLogin AdminLoginPage = new AdminLogin();
            this.Hide();
            AdminLoginPage.ShowDialog();
        }
    }
}

catch (Exception err) {
    MessageBox.Show("Please try
again",err.Message,MessageBoxButtons.OKCancel,MessageBoxIcon.Error);
}
}

private void adminPasswordLabel_Click(object sender, EventArgs e) { }

private void adminUserNameLabel_Click(object sender, EventArgs e) { }

private void AdminRegister_Load_1(object sender, EventArgs e)

```

```

{
    AdminAutoID();
}

private void adminClearBtn_Click(object sender, EventArgs e)
{
    ClearAdminForm();
}

private void panel1_Paint(object sender, PaintEventArgs e) {}

private void panel2_Paint(object sender, PaintEventArgs e) { }

private void adminIDTextBox_TextChanged(object sender, EventArgs e) {}

private void adminNameLabel_Click(object sender, EventArgs e) {}

private void adminEmailLabel_Click(object sender, EventArgs e) {}

private void adminIDLabel_Click(object sender, EventArgs e) {}

private void adminRegisterLabel_Click(object sender, EventArgs e) {}

private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    AdminLogin adminLogin = new AdminLogin();
    this.Hide();
    adminLogin.ShowDialog();
}

private void pictureBox1_Click(object sender, EventArgs e) {}

```

```
private void panel3_Paint(object sender, PaintEventArgs e) {}  
}  
}
```

## 2. Admin Login (AdminLogin.cs)

### Admin Login GUI Design

The screenshot shows a Windows application window titled "AdminLogin". Inside the window, the text "Admin Login" is centered at the top. Below it is a circular icon representing a user profile. Under the icon are two text input fields: "Username" and "Password". Below these fields are two buttons: "Login" and "Clear". At the bottom, there is a text label "Don't have an account?" followed by a blue hyperlink "Register Here".

## **AdminLogin.cs File Code**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class AdminLogin : Form
    {
        FitnessDataSetTableAdapters.AdminsTableAdapter adminTb = new
        FitnessDataSetTableAdapters.AdminsTableAdapter();

        DataTable dtAdmin = new DataTable();
        public int loginCount = 0;
        public static string loginUserName, loginUserID;
        public AdminLogin()
        {
            InitializeComponent();
        }

        public void ClearAdminLoginForm()
        {
            adminUserNameTextBox.Text = "";
            adminPasswordTextBox.Text = "";
        }
    }
}
```



```

private void AdminLogin_Load(object sender, EventArgs e) {}

private void registerBtn_Click(object sender, EventArgs e)
{
    if (loginCount == 3)
    {
        MessageBox.Show("Your attempt has failed 3 times");
        Application.Exit();
    }
    else {
        if (adminUserNameTextBox.Text == "") {
            MessageBox.Show("User name can't be empty");
            adminUserNameTextBox.Focus();
        }
        else if (adminPasswordTextBox.Text == "")
        {
            MessageBox.Show("Password can't be empty");
            adminPasswordTextBox.Focus();
        }
        else {
            ClsAdmin adminObj = new ClsAdmin();
            adminObj.AUserName = adminUserNameTextBox.Text;
            adminObj.APassword = adminPasswordTextBox.Text; ;
            dtAdmin = adminTb.AdminLogin(adminObj.AUserName, adminObj.APassword);
            if (dtAdmin.Rows.Count > 0)
            {
                MessageBox.Show("Admin login success");
                loginUserID = dtAdmin.Rows[0][0].ToString();
                loginUserName = dtAdmin.Rows[0][1].ToString();

                AdminDashboard home = new AdminDashboard();
            }
        }
    }
}

```

```

        this.Hide();
        home.ShowDialog();
    }
}
}

private void adminLoginBtn_Click(object sender, EventArgs e) { }

private void linkLblRegister_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    AdminRegister AdminRegisterForm = new AdminRegister();
    this.Hide();
    AdminRegisterForm.ShowDialog();
}

private void adminUserNameLabel_Click(object sender, EventArgs e) { }

private void adminPasswordLabel_Click(object sender, EventArgs e) { }

private void clearFormBtn_Click(object sender, EventArgs e) {
    ClearAdminLoginForm();
}

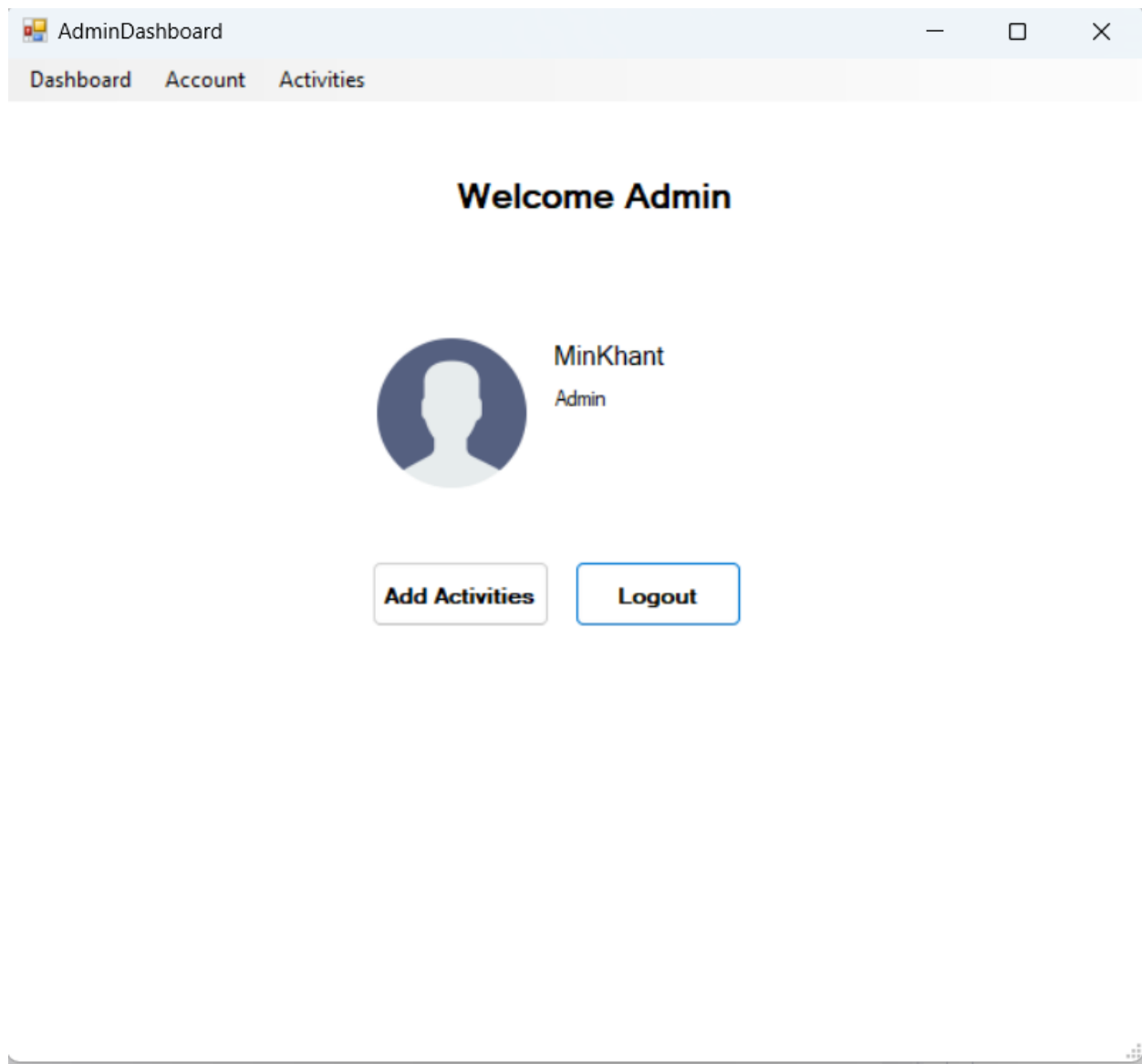
private void labelGoToRegister_Click(object sender, EventArgs e) { }

private void adminPasswordTextBox_TextChanged(object sender, EventArgs e) { }
}
}

```

### 3. Admin Dashboard (AdminDashboard.cs)

#### Admin Dashboard GUI Design



## **AdminDashboard.cs File Code**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class AdminDashboard : Form
    {
        public AdminDashboard()
        {
            InitializeComponent();
        }

        private void AdminDashboard_Load(object sender, EventArgs e)
        {
            lblAdminName.Text = AdminLogin.loginUserName;
        }

        private void btnAddActivity_Click(object sender, EventArgs e) {
            Activity AddActivityForm = new Activity();
            this.Hide();
            AddActivityForm.ShowDialog();
        }
    }
}
```

```

private void lblAddActivities_Click(object sender, EventArgs e)
{
    Activity activityForm = new Activity();
    this.Hide();
    activityForm.ShowDialog();
}

private void label1_Click(object sender, EventArgs e) { }

private void btnLogout_Click(object sender, EventArgs e)
{
    FitnessTrackerLoad mainPage = new FitnessTrackerLoad();
    this.Hide();
    mainPage.ShowDialog();
}

private void pictureBox1_Click(object sender, EventArgs e) { }

private void lblAdminName_Click(object sender, EventArgs e) { }

private void label2_Click_1(object sender, EventArgs e) { }

private void activitiesToolStripMenuitem_Click(object sender, EventArgs e)
{
    Activity act = new Activity();
    this.Hide();
    act.ShowDialog();
}

private void accountToolStripMenuitem_Click(object sender, EventArgs e) { }

```

```

private void logoutToolStripMenuItem_Click(object sender, EventArgs e)
{
    FitnessTrackerLoad mainPage = new FitnessTrackerLoad();
    this.Hide();
    mainPage.ShowDialog();
}
}
}

```

## 4. Activity (Activity.cs)

### Activity GUI Design

The screenshot shows a web application window titled "Activity" with a navigation bar containing "Dashboard", "Account", and "Activities". The main content area is divided into two sections:

**Add new activity**

On the left, there are three buttons: "Clear", "Update", and "Delete". To the right, there are input fields for "Activity ID" (containing "Activity-007"), "Activity Name", "Metric One", "Metric Two", and "Metric Three". An "Add Activity" button is positioned below the "Activity Name" field.

**Activity List**

Below the form is a table with the following data:

	ActivityID	ActivityName	Metric1	Metric2	Metric3
▶	Activity-001	Walking	steps	time taken	distance
	Activity-002	Swimming	number of laps	time taken	average heart rate
	Activity-003	Jumping Ropes	jumps per minute	time taken	total jumps
	Activity-004	Running	steps	time taken	distance
	Activity-005	Cycling	average riding s...	time taken	distance
	Activity-006	Pushups	pushups per min...	time taken	total pushups
*					

## **Activity.cs File Code**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class Activity : Form
    {
        FitnessDataSetTableAdapters.ActivitiesTableAdapter activity = new
        FitnessDataSetTableAdapters.ActivitiesTableAdapter();

        DataTable actdt = new DataTable();

        public Activity()
        {
            InitializeComponent();
        }

        public void ActivityAutoID()
        {
            DataTable trackdta = new DataTable();
            trackdta = activity.GetData();

            if (trackdta.Rows.Count == 0)
            {
```

```

        textBoxActivityID.Text = "Activity-001";
    }
    else
    {
        int size = trackdta.Rows.Count - 1;
        int newid;
        string oldid = trackdta.Rows[size][0].ToString();
        if(oldid.Length == 12 ){
            newid = Convert.ToInt16(oldid.Substring(9, 3));
        }else {
            newid = 081;
        }

        if (newid >= 1 && newid < 9)
        {
            textBoxActivityID.Text = "Activity-00" + (newid + 1);
        }
        else if (newid >= 9 && newid < 99)
        {

            textBoxActivityID.Text = "Activity-0" + (newid + 1);
        }
        else if (newid >= 99 && newid < 999)
        {
            textBoxActivityID.Text = "Activity-" + (newid + 1);
        }
    }
}

public void ClearAll()
{

```



```

        textBoxActivityName.Text = "";
        textBoxMetricOne.Text = "";
        textBoxMetricTwo.Text = "";
        textBoxMetricThree.Text = "";
    }

    private void label2_Click(object sender, EventArgs e) { }

    private void label1_Click(object sender, EventArgs e) { }

    private void label3_Click(object sender, EventArgs e) { }

    private void textBox3_TextChanged(object sender, EventArgs e) { }

    private void textBox2_TextChanged(object sender, EventArgs e) { }

    private void textBox1_TextChanged(object sender, EventArgs e) { }

    private void Activity_Load(object sender, EventArgs e)
    {
        // TODO: This line of code loads data into the 'fitnessDataSet.Activities' table. You can move, or
        // remove it, as needed.
        this.activitiesTableAdapter.Fill(this.fitnessDataSet.Activities);
        ActivityAutoID();
    }

    private void btnClear_Click(object sender, EventArgs e)
    {
        ClearAll();
    }

    private void btnAdd_Click(object sender, EventArgs e)

```

```

{
    if (textBoxActivityName.Text == "") {
        MessageBox.Show("Activity name must not be empty");
        textBoxActivityName.Focus();
    }
    else if (textBoxMetricOne.Text == "") {
        MessageBox.Show("Metric one must not be empty");
        textBoxMetricOne.Focus();
    }
    else if (textBoxMetricTwo.Text == "")
    {
        MessageBox.Show("Metric two must not be empty");
        textBoxMetricTwo.Focus();
    }
    else if (textBoxMetricThree.Text == "")
    {
        MessageBox.Show("Metric three must not be empty");
        textBoxMetricThree.Focus();
    }
    else {
        ClsActivity activityObj = new ClsActivity();
        activityObj.ActID = textBoxActivityID.Text;
        activityObj.ActName = textBoxActivityName.Text;
        activityObj.ActMetricOne = textBoxMetricOne.Text;
        activityObj.ActMetricTwo = textBoxMetricTwo.Text;
        activityObj.ActMetricThree = textBoxMetricThree.Text;

        int insertActivity = activity.InsertActivity(activityObj.ActID, activityObj.ActName,
        activityObj.ActMetricOne, activityObj.ActMetricTwo, activityObj.ActMetricThree);
        if (insertActivity > 0)
        {

```

```

        MessageBox.Show("Activity has been created successfully.");
        dgvActivityList.DataSource = activity.GetData();
        dgvActivityList.Refresh();
        ClearAll();
        // Clear form here
        ActivityAutoID();
    }
}

private void btnAdminRegister_Click(object sender, EventArgs e) { }

private void textBoxActivityID_TextChanged(object sender, EventArgs e) { }

private void btnDelete_Click(object sender, EventArgs e)
{
    int delRow = dgvActivityList.CurrentCell.RowIndex;
    dgvActivityList.Rows.RemoveAt(delRow);
    activity.DeleteActivity(textBoxActivityID.Text);
    MessageBox.Show("Activity Successfully Deleted");
    ClearAll();
    ActivityAutoID();
}

private void btnUpdate_Click(object sender, EventArgs e)
{
    try
    {
        if (textBoxActivityName.Text == "")
        {

```

```

        MessageBox.Show("Fill Activity Name!", "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);

        textBoxActivityName.Focus();
    }

    else if (textBoxMetricOne.Text == "")
    {
        MessageBox.Show("Fill Metric One!", "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);

        textBoxMetricOne.Focus();
    }

    else if (textBoxMetricTwo.Text == "")
    {
        MessageBox.Show("Fill Metric Two!", "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);

        textBoxMetricTwo.Focus();
    }

    else if (textBoxMetricThree.Text == "")
    {
        MessageBox.Show("Fill Admin Password!", "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);

        textBoxMetricThree.Focus();
    }

    else
    {
        ClsActivity activities = new ClsActivity();
        activities.ActID = textBoxActivityID.Text;
        activities.ActName = textBoxActivityName.Text;
        activities.ActMetricOne = textBoxMetricOne.Text;
        activities.ActMetricTwo = textBoxMetricTwo.Text;
        activities.ActMetricThree = textBoxMetricThree.Text;

        activity.UpdateActivity(activities.ActName, activities.ActMetricOne, activities.ActMetricTwo,
        activities.ActMetricThree, activities.ActID);
    }

```

```

        MessageBox.Show("Update Successfully");
        dgvActivityList.DataSource = activity.GetData();
        dgvActivityList.Refresh();
        ClearAll();
        ActivityAutoID();
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

private void dgvActivityList_CellContentClick(object sender, DataGridViewCellEventArgs e) {}

private void dashboardToolStripMenuItem_Click(object sender, EventArgs e) {
    AdminDashboard dashboard = new AdminDashboard();
    this.Hide();
    dashboard.ShowDialog();
}

private void dgvActivityList_CellClick(object sender, DataGridViewCellEventArgs e) {
    int row = dgvActivityList.CurrentRow.Index;
    textBoxActivityID.Text = dgvActivityList[0, row].Value.ToString();
    textBoxActivityName.Text = dgvActivityList[1, row].Value.ToString();
    textBoxMetricOne.Text = dgvActivityList[2, row].Value.ToString();
    textBoxMetricTwo.Text = dgvActivityList[3, row].Value.ToString();
    textBoxMetricThree.Text = dgvActivityList[4, row].Value.ToString();
}

private void logoutToolStripMenuItem_Click(object sender, EventArgs e)

```

```

{
    FitnessTrackerLoad mainPage = new FitnessTrackerLoad();
    this.Hide();
    mainPage.ShowDialog();
}
}
}

```

## 5. Customer Register (CustomerRegister.cs)

### Customer Register GUI Design

The screenshot shows a Windows application window titled "CustomerRegister". The window is divided into two main sections by a vertical dashed line.

**Left Section:**

- At the top is a circular logo for "FITNESS CLUB" featuring a muscular man holding a dumbbell.
- Below the logo, the text "Already have an account?" is displayed, followed by a blue, underlined "Login" link.
- Below the link, four lines of text specify registration requirements:
  - Only allow letters and numbers
  - Must be at least 12 characters
  - At least 1 uppercase character
  - At least 1 lowercase character

**Right Section:**

- The title "User Registration" is centered at the top.
- Below the title are five input fields, each with a label to its left:
  - User ID
  - Full Name
  - Email
  - Password
  - Username
- At the bottom right of the form are two buttons: a "Register" button and a "Clear" button (text is red).

## **CustomerRegister.cs File Code**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace FitnessTrackerApp
{
    public partial class CustomerRegister : Form
    {
        FitnessDataSetTableAdapters.UsersTableAdapter CusTA = new
        FitnessDataSetTableAdapters.UsersTableAdapter();

        DataTable cusdt = new DataTable();

        public CustomerRegister()
        {
            InitializeComponent();
        }

        public void ClearAdminForm()
        {
            customerNameTextBox.Text = "";
            customerEmailTextBox.Text = "";
            customerPasswordTextBox.Text = "";
            customerUserNameTextBox.Text = "";
        }
    }
}
```

```

customerEmailTextBox.Text = "";
}

public void CustomerAutoID()
{
    DataTable trackdta = new DataTable();
    trackdta = CusTA.GetData();

    if (trackdta.Rows.Count == 0)
    {
        customerIDTextBox.Text = "Customer-001";
    }
    else
    {
        int size = trackdta.Rows.Count - 1;
        string oldid = trackdta.Rows[size][0].ToString();
        int newid = Convert.ToInt16(oldid.Substring(9, 3));

        if (newid >= 1 && newid < 9)
        {
            customerIDTextBox.Text = "Customer-00" + (newid + 1);
        }
        else if (newid >= 9 && newid < 99)
        {
            customerIDTextBox.Text = "Customer-0" + (newid + 1);
        }
        else if (newid >= 99 && newid < 999)
        {
            customerIDTextBox.Text = "Customer-" + (newid + 1);
        }
    }
}

```



```

}

private void adminNameLabel_Click(object sender, EventArgs e) { }

private void customerNameTextBox_TextChanged(object sender, EventArgs e) { }

private void label3_Click(object sender, EventArgs e) { }

private void numberLabel_Click(object sender, EventArgs e) { }

private void lowercaseLabel_Click(object sender, EventArgs e) { }

private bool is_validUsername(string username)
{
    //Regular expression to match only letters and numbers
    string pattern = "^[a-zA-Z0-9]+$";
    Regex regex = new Regex(pattern);
    return regex.IsMatch(username);
}

private bool is_validPassword(string password) {
    bool is_valid = true;
    Regex has_Number = new Regex(@"[0-9]");
    Regex has_UpperChar = new Regex(@"[A-Z]");
    Regex has_LowerChar = new Regex(@"[a-z]");
    Regex has_MinMaxChar = new Regex(@".{12,}");
    if (!has_UpperChar.IsMatch(password))
    {
        upperCaseLabel.Text = "Must have at least 1 uppercase character";
        upperCaseLabel.ForeColor = Color.Red;
        customerPasswordTextBox.Focus();
    }
}

```

```

        is_valid = false;
    }
    else {
        upperCaseLabel.Text = "Uppercase character requirement has been fulfilled.";
        upperCaseLabel.ForeColor = Color.Green;
    }
    if (!has_LowerChar.IsMatch(password))
    {
        lowercaseLabel.Text = "Must have at least one lowercase character";
        lowercaseLabel.ForeColor = Color.Red;
        customerPasswordTextBox.Focus();
        is_valid = false;
    }
    else {
        lowercaseLabel.Text = "Lowercase character requirement has been fulfilled.";
        lowercaseLabel.ForeColor = Color.Green;
    }
    if (!has_Number.IsMatch(password))
    {
        numberLabel.Text = "Must have at least one number";
        numberLabel.ForeColor = Color.Red;
        customerPasswordTextBox.Focus();
        is_valid = false;
    }
    else
    {
        numberLabel.Text = "Number requirement has been fulfilled.";
        numberLabel.ForeColor = Color.Green;
    }
    if (!has_MinMaxChar.IsMatch(password))
    {

```

```

        charLimitLabel.Text = "Password must be at least 12 characters";
        charLimitLabel.ForeColor = Color.Red;
        customerPasswordTextBox.Focus();
        is_valid = false;
    }
    else
    {
        numberLabel.Text = "12 characters password has been fulfilled.";
        charLimitLabel.ForeColor = Color.Green;
    }
    return is_valid;
}

private void registerBtn_Click(object sender, EventArgs e)
{
    bool isPasswordValid = is_validPassword(customerPasswordTextBox.Text);
    bool isUsernameValid = is_validUsername(customerUserNameTextBox.Text);

    if (customerNameTextBox.Text == "") {
        MessageBox.Show("Customer name must not be empty");
        customerNameTextBox.Focus();
    }

    else if (isPasswordValid == false) {
        MessageBox.Show("Customer password is not valid", "Warning",
        MessageBoxButtons.OKCancel, MessageBoxIcon.Error);
        customerPasswordTextBox.Focus();
    }

    else if (customerUserNameTextBox.Text == "")
    {
        MessageBox.Show("Admin username must not be empty");
    }
}

```

```

        customerUserNameTextBox.Focus();
    }
    else if (isUsernameValid == false)
    {
        MessageBox.Show("Customer username is not valid, it must contain only letters and numbers",
"Warning", MessageBoxButtons.OKCancel, MessageBoxIcon.Error);
        customerUserNameTextBox.Focus();
    }
    else{
        // Encapsulation customer
        ClsCustomer customerObj = new ClsCustomer();
        customerObj.CusID = customerIDTextBox.Text;
        customerObj.CusName = customerNameTextBox.Text;
        customerObj.CusPassword =customerPasswordTextBox.Text;
        customerObj.CusUName =customerUserNameTextBox.Text;
        customerObj.CusEmail =customerEmailTextBox.Text;

        int insertCustomer = CusTA.CustomerRegister(customerObj.CusID, customerObj.CusName,
customerObj.CusEmail,customerObj.CusPassword, customerObj.CusUName );

        if (insertCustomer > 0)
        {
            MessageBox.Show("Customer account registration completed successfully.");
            ClearAdminForm();
            CustomerAutoID();
            this.Hide();
            CustomerLogin loginForm = new CustomerLogin();
            loginForm.ShowDialog();
        };
    }
}

```

```

private void adminClearBtn_Click(object sender, EventArgs e)
{
    ClearAdminForm();
}

private void customerPasswordTextBox_TextChanged(object sender, EventArgs e)
{
    bool isPasswordValid = is_validPassword(customerPasswordTextBox.Text);
    charLimitLabel.Visible = true;
    lowercaseLabel.Visible = true;
    uppercaseLabel.Visible = true;
    numberLabel.Visible = true;
}

private void CustomerRegister_Load(object sender, EventArgs e)
{
    CustomerAutoID();
}

private void pictureBox1_Click(object sender, EventArgs e) { }

private void charLimitLabel_Click(object sender, EventArgs e) { }

private void customerRegisterLabel_Click(object sender, EventArgs e) { }

private void adminIDLabel_Click(object sender, EventArgs e) { }

private void customerIDTextBox_TextChanged(object sender, EventArgs e) { }

private void customerEmailTextBox_TextChanged(object sender, EventArgs e) { }

```

```
private void adminPasswordLabel_Click(object sender, EventArgs e) { }

private void adminUserNameLabel_Click(object sender, EventArgs e) { }

private void customerUserNameTextBox_TextChanged(object sender, EventArgs e) { }

private void adminEmailLabel_Click(object sender, EventArgs e) { }

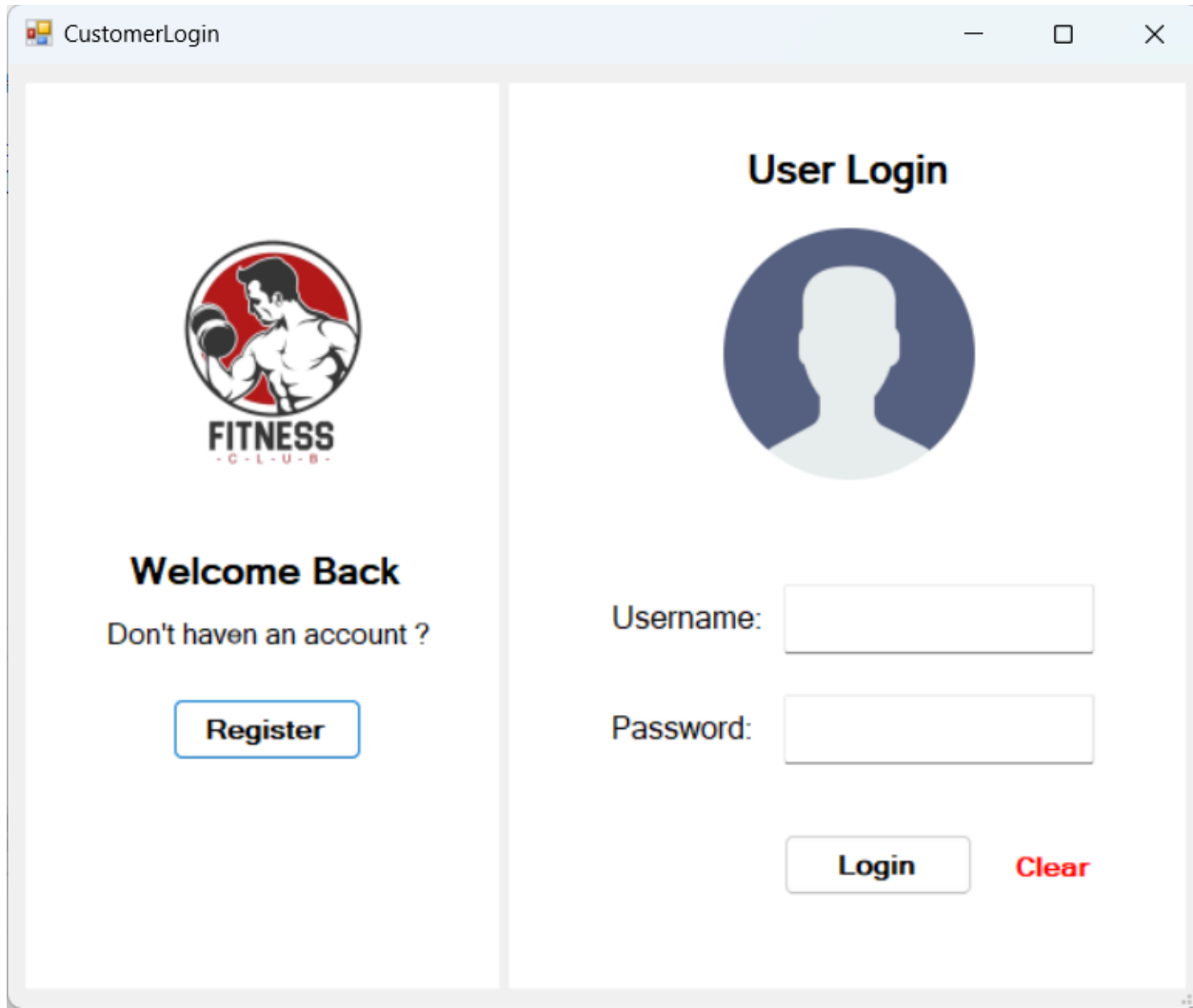
private void adminClearBtn_Click_1(object sender, EventArgs e)
{
    ClearAdminForm();
}

private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    CustomerLogin customerLogin = new CustomerLogin();
    this.Hide();
    customerLogin.ShowDialog();
}

private void label2_Click(object sender, EventArgs e) { }
}
```

## 6. Customer Login (CustomerLogin.cs)


### Customer Login GUI Design



The image shows a Windows application window titled "CustomerLogin". The window is divided into two main sections by a vertical line. The left section features a circular logo with a muscular man holding a dumbbell, with the text "FITNESS -C-L-U-B-" below it. Below the logo, the text "Welcome Back" is displayed, followed by "Don't haven an account ?" and a "Register" button. The right section is titled "User Login" and features a circular placeholder for a user profile picture. Below this, there are input fields for "Username:" and "Password:". At the bottom of the right section, there is a "Login" button and a "Clear" link.

CustomerLogin

**User Login**



**Welcome Back**

Don't haven an account ?

**Register**

Username:

Password:

**Login** **Clear**

## CustomerLogin.cs File Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class CustomerLogin : Form
    {
        // Table adapter goes here.

        // CusTA
        // custd

        FitnessDataSetTableAdapters.UsersTableAdapter CustomerTable = new
        FitnessDataSetTableAdapters.UsersTableAdapter();

        DataTable cusdt = new DataTable();

        public static string CID, CName;

        int loginCount = 0;

        public CustomerLogin()
        {
            InitializeComponent();
        }

        public void ResetLoginForm()
```



```

{
    customerPasswordTextBox.Text = "";
    customerUserNameTextBox.Text = "";
}

private void label3_Click(object sender, EventArgs e) {}

private void panel1_Paint(object sender, PaintEventArgs e) {}

private void customerLoginBtn_Click(object sender, EventArgs e)
{
    if (loginCount == 3)
    {
        MessageBox.Show("Your attempt has failed 3 times");
        Application.Exit();
    }
    else {
        if (customerUserNameTextBox.Text == "")
        {
            MessageBox.Show("User name can't be empty");
            customerUserNameTextBox.Focus();
        }
        else if (customerPasswordTextBox.Text == "")
        {
            MessageBox.Show("Password can't be empty");
            customerPasswordTextBox.Focus();
        }
        else
        {
            ClsCustomer customerObj = new ClsCustomer();
            customerObj.CusUName = customerUserNameTextBox.Text;

```

```

customerObj.CusPassword = customerPasswordTextBox.Text; ;

cusdt = CustomerTable.CustomerLogin(customerObj.CusUName, customerObj.CusPassword);
if (cusdt.Rows.Count > 0)
{
    MessageBox.Show("Customer login success");
    CID = cusdt.Rows[0][0].ToString();
    CName = cusdt.Rows[0][1].ToString();

    ActivitiesList list = new ActivitiesList();
    this.Hide();
    list.ShowDialog();
}
else
{
    loginCount += 1;
    MessageBox.Show("Incorrect credentials");
}
}
}

private void pictureBox1_Click(object sender, EventArgs e) { }

private void pictureBox1_Click_1(object sender, EventArgs e) { }

private void customerUserNameLabel_Click(object sender, EventArgs e) { }

private void customerUserNameTextBox_TextChanged(object sender, EventArgs e) { }

private void customerPasswordTextBox_TextChanged(object sender, EventArgs e) { }

```

```

private void customerPasswordLabel_Click(object sender, EventArgs e) {}

private void clearBtn_Click(object sender, EventArgs e)
{
    ResetLoginForm();
}

private void registerBtn_Click(object sender, EventArgs e)
{
    CustomerRegister customerRegisterForm = new CustomerRegister();
    this.Hide();
    customerRegisterForm.ShowDialog();
}

private void CustomerLogin_Load(object sender, EventArgs e) {}

private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    CustomerRegister customerRegisterForm = new CustomerRegister();
    this.Hide();
    customerRegisterForm.ShowDialog();
}

private void adminRegisterLabel_Click(object sender, EventArgs e) {}

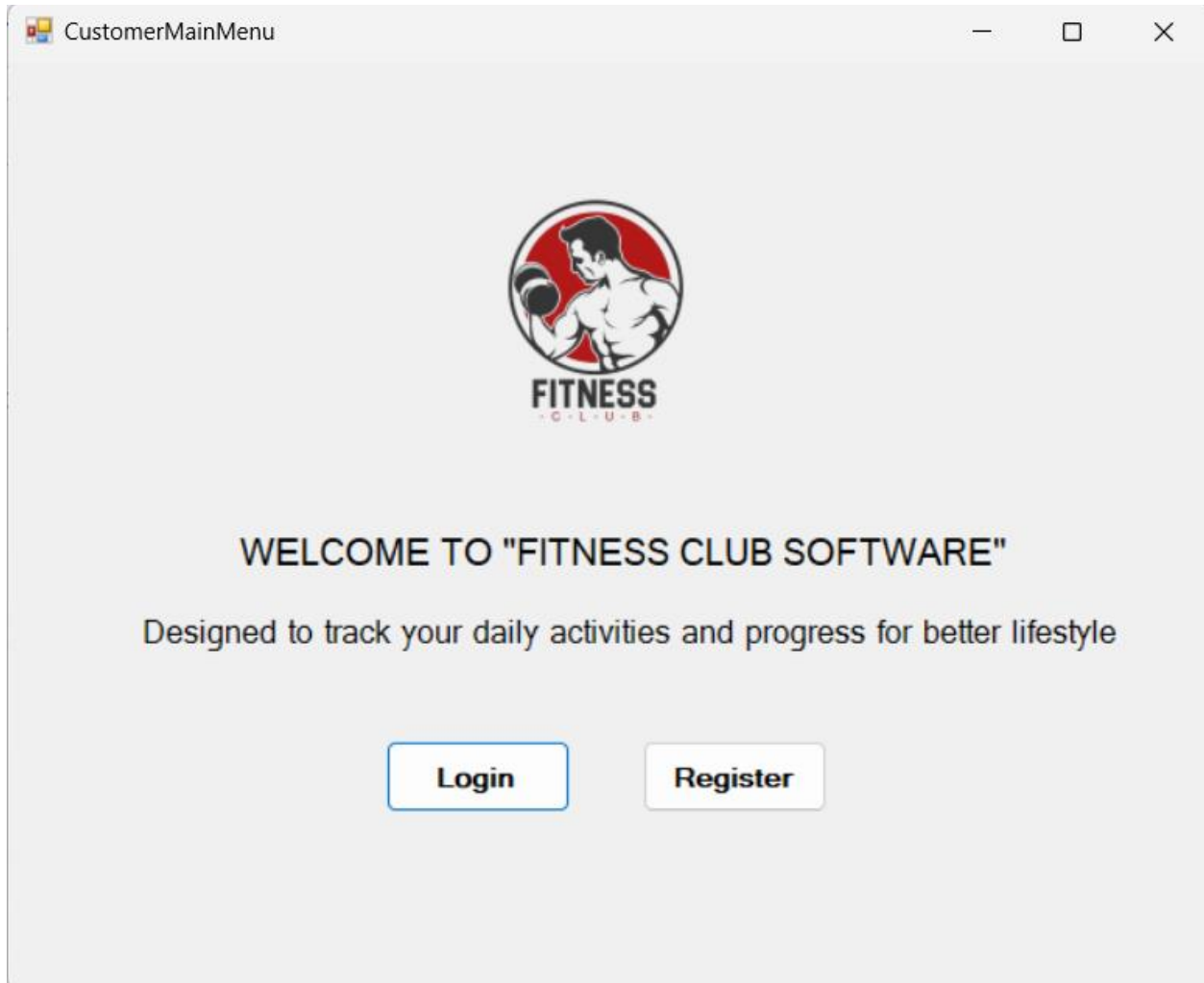
private void adminNameLabel_Click(object sender, EventArgs e) {}

private void linkLabel2_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e) {}
}
}

```

## 7. Customer Main Menu (CustomerMainMenu.cs)

### Customer Main Menu GUI Design



## **CustomerMainMenu Code File**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class CustomerMainMenu : Form
    {
        public CustomerMainMenu()
        {
            InitializeComponent();
        }

        private void NavigateToActivitiesList(){
            ActivitiesList ActivitiesListForm = new ActivitiesList();
            this.Hide();
            ActivitiesListForm.ShowDialog();
        }

        private void label4_Click(object sender, EventArgs e) { }

        private void lblRegister_Click(object sender, EventArgs e)
        {
            CustomerRegister customerRegister= new CustomerRegister();
```

```

        this.Hide();
        customerRegister.ShowDialog();
    }

    private void lblLogin_Click(object sender, EventArgs e)
    {
        CustomerLogin customerLogin = new CustomerLogin();
        this.Hide();
        customerLogin.ShowDialog();
    }

    private void label2_Click(object sender, EventArgs e) { }

    private void loginButton_Click(object sender, EventArgs e)
    {
        CustomerLogin customerLogin = new CustomerLogin();
        this.Hide();
        customerLogin.ShowDialog();
    }

    private void lblActivities_Click(object sender, EventArgs e)
    {
        NavigateToActivitiesList();
    }

    private void lblInformation_Click(object sender, EventArgs e) { }

    private void lblHome_Click(object sender, EventArgs e) { }

    private void CustomerMainMenu_Load(object sender, EventArgs e) { }

```

```
private void EXcellent_Click(object sender, EventArgs e) { }

private void pictureBox1_Click(object sender, EventArgs e) { }

private void pictureBox2_Click(object sender, EventArgs e) { }

private void rectangleShape1_Click(object sender, EventArgs e) { }

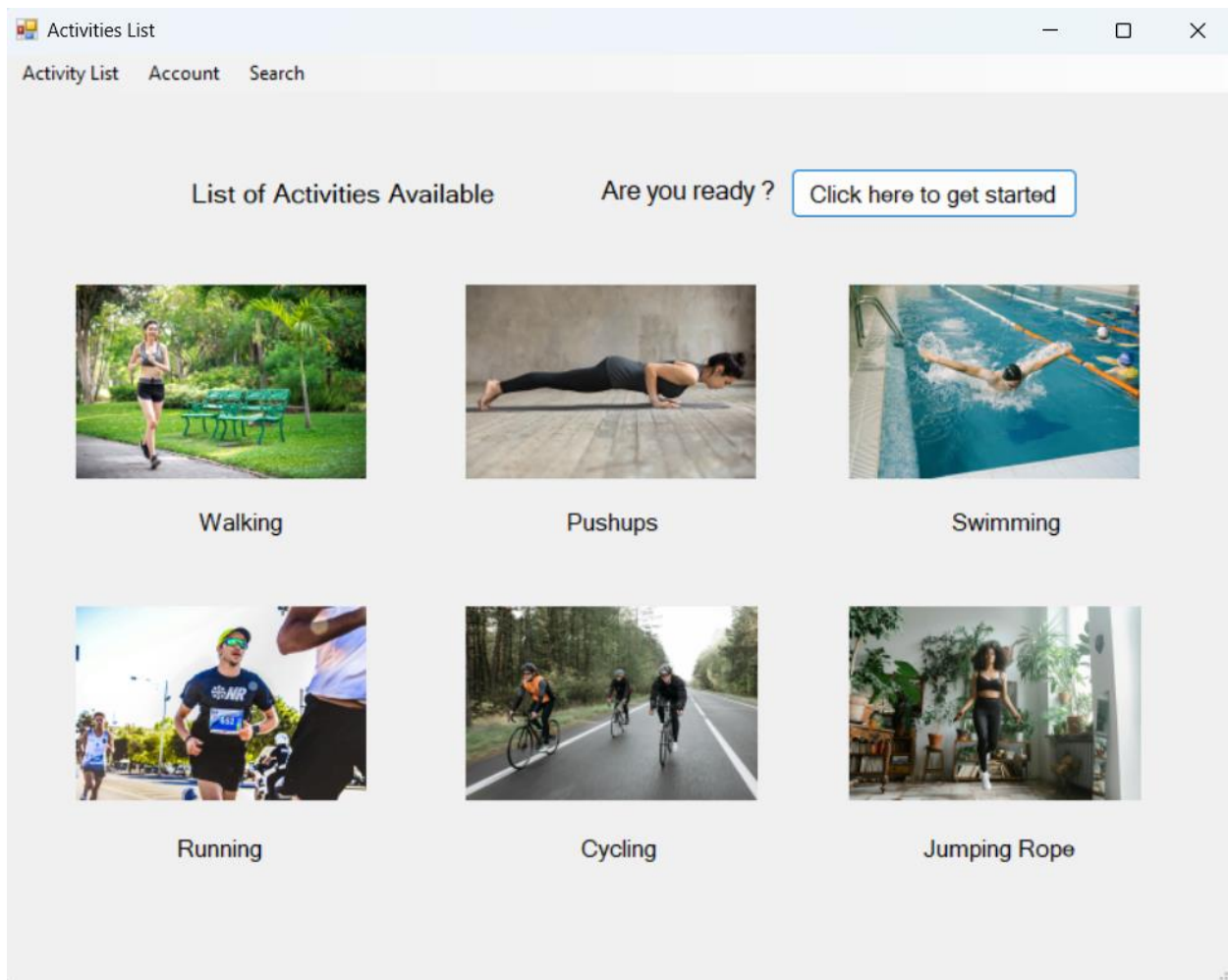
private void pictureBox1_Click_1(object sender, EventArgs e) { }

private void label1_Click(object sender, EventArgs e) { }

private void registerButton_Click(object sender, EventArgs e)
{
    CustomerRegister customerRegister = new CustomerRegister();
    this.Hide();
    customerRegister.ShowDialog();
}
}
```

## 8. Activities List (ActivitiesList.cs)

### Activities List GUI Design



### Activities List.cs Code File

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;
```



```

using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class ActivitiesList : Form
    {
        public ActivitiesList()
        {
            InitializeComponent();
        }

        private void ActivitiesList_Load(object sender, EventArgs e) {}

        private void lblActivities_Click(object sender, EventArgs e)
        {
            ActivitiesList ActivitiesListForm = new ActivitiesList();
            this.Hide();
            ActivitiesListForm.ShowDialog();
        }

        private void lblHome_Click(object sender, EventArgs e)
        {
            CustomerMainMenu HomePage = new CustomerMainMenu();
            this.Hide();
            HomePage.ShowDialog();
        }

        private void lblInformation_Click(object sender, EventArgs e) {}

        private void label2_Click(object sender, EventArgs e) {}
    }
}

```

```

private void pictureBox2_Click(object sender, EventArgs e) {}

private void label3_Click(object sender, EventArgs e) {}

private void pictureBox3_Click(object sender, EventArgs e) {}

private void pictureBox6_Click(object sender, EventArgs e) {}

private void pictureBox5_Click(object sender, EventArgs e) {}

private void pictureBox4_Click(object sender, EventArgs e) {}

private void button1_Click(object sender, EventArgs e)
{
    SetGoal goal = new SetGoal();
    this.Hide();
    goal.ShowDialog();
}

private void logoutToolStripMenuItem_Click(object sender, EventArgs e)
{
    FitnessTrackerLoad mainPage = new FitnessTrackerLoad();
    this.Hide();
    mainPage.ShowDialog();
}

private void activityListToolStripMenuItem_Click(object sender, EventArgs e) {

}

private void searchToolStripMenuItem_Click(object sender, EventArgs e)

```

```

{
    CustomerSearch searchList = new CustomerSearch();
    this.Hide();
    searchList.ShowDialog();
}
}
}

```

## 9. Set Goal (SetGoal.cs)

### SetGoal GUI Design

SetGoal

Activity List Account Search

Welcome **Min Khant Kyaw**

Let's define and achieve your goal

Choose activity and define your goal

Choose Activity

Pushups

Activity ID Activity-006 Track ID Track-010

Metric One pushups per minute

Metric Two time taken

Metric Three total pushups

Your Goal

Deadline Monday, July 15, 2024

Save

## **SetGoal.cs File Code**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class SetGoal : Form
    {
        FitnessDataSetTableAdapters.UsersTableAdapter cusTA = new
        FitnessDataSetTableAdapters.UsersTableAdapter();

        FitnessDataSetTableAdapters.ActivitiesTableAdapter ActivityTA = new
        FitnessDataSetTableAdapters.ActivitiesTableAdapter();

        FitnessDataSetTableAdapters.TrackTableAdapter TrackTA = new
        FitnessDataSetTableAdapters.TrackTableAdapter();

        DataTable Activitydta = new DataTable();

        public SetGoal()
        {
            InitializeComponent();
        }

        public void TrackAutoID()
        {
            DataTable trackdta = new DataTable();
```

```

trackdta = TrackTA.GetData();

if (trackdta.Rows.Count == 0)
{
    textBoxTrackID.Text = "Track-001";
}
else
{
    int size = trackdta.Rows.Count - 1;
    string oldid = trackdta.Rows[size][0].ToString();
    int newid = Convert.ToInt16(oldid.Substring(6, 3));

    if (newid >= 1 && newid < 9)
    {
        textBoxTrackID.Text = "Track-00" + (newid + 1);
    }
    else if (newid >= 9 && newid < 99)
    {
        textBoxTrackID.Text = "Track-0" + (newid + 1);
    }
    else if (newid >= 99 && newid < 999)
    {
        textBoxTrackID.Text = "Track-" + (newid + 1);
    }
}

private void groupBox1_Enter(object sender, EventArgs e) { }

private void lblGoal_Click(object sender, EventArgs e) { }

```

```

private void label1_Click(object sender, EventArgs e) { }

public void SelectActivity() {
    Activitydta = ActivityTA.GetData();
    if (Activitydta.Rows.Count > 0)
    {
        DataRow dr = Activitydta.NewRow();
        comboBoxActivity.DataSource = Activitydta;
        comboBoxActivity.DisplayMember = "ActivityName";
        comboBoxActivity.ValueMember = "ActivityID";
    }
}

private void SetGoal_Load(object sender, EventArgs e)
{
    lblUserName.Text = CustomerLogin.CName;
    lblUserName.Visible = true;

    TrackAutoID();
    SelectActivity();
    textBoxActivityID.Text = "";
}

private void groupBox3_Enter(object sender, EventArgs e) { }

private void comboBoxActivity_SelectedIndexChanged(object sender, EventArgs e)
{
    string id = comboBoxActivity.SelectedValue.ToString();
    textBoxActivityID.Text = id;

    Activitydta = ActivityTA.SelectedActivity(textBoxActivityID.Text);
}

```

```

        if (Activitydta.Rows.Count > 0) {
            textBoxMetricOne.Text = Activitydta.Rows[0]["Metric1"].ToString();
            textBoxMetricTwo.Text = Activitydta.Rows[0]["Metric2"].ToString();
            textBoxMetricThree.Text = Activitydta.Rows[0]["Metric3"].ToString();
        }
    }

    private void buttonSave_Click(object sender, EventArgs e)
    {
        try {
            int recordedCount = TrackTA.RecordTrack(textBoxTrackID.Text, comboBoxActivity.Text,
            "Incomplete", CustomerLogin.CID, textBoxActivityID.Text, goalDate.Text,
            Convert.ToInt32(textBoxGoal.Text));

            if (recordedCount > 0)
            {
                MessageBox.Show("Your goal has been defined successfully.");

                Track t = new Track();

                this.Hide();

                t.ShowDialog();
            }
        }
        catch (Exception ex) {
            MessageBox.Show("Error :" + ex);
        }
    }

    private void textBoxMemberName_TextChanged(object sender, EventArgs e) { }

    private void lblActivities_Click(object sender, EventArgs e)
    {
        ActivitiesList ActivitiesListForm = new ActivitiesList();

        this.Hide();
    }

```

```

        ActivitiesListForm.ShowDialog();
    }

    private void lblMetricOne_Click(object sender, EventArgs e) { }

    private void groupBox1_Enter_1(object sender, EventArgs e) { }

    private void homeToolStripMenuItem_Click(object sender, EventArgs e)
    {
        ActivitiesList activityList = new ActivitiesList();
        this.Hide();
        activityList.ShowDialog();
    }

    private void accountToolStripMenuItem_Click(object sender, EventArgs e) { }

    private void logoutToolStripMenuItem1_Click(object sender, EventArgs e)
    {
        FitnessTrackerLoad mainPage = new FitnessTrackerLoad();
        this.Hide();
        mainPage.ShowDialog();
    }

    private void activityListToolStripMenuItem_Click(object sender, EventArgs e) { }

    private void logoutToolStripMenuItem_Click(object sender, EventArgs e) { }

    private void searchToolStripMenuItem_Click(object sender, EventArgs e)
    {
        CustomerSearch searchList = new CustomerSearch();
        this.Hide();
    }

```




```
        searchList.ShowDialog();
    }
}
}
```

10. Track Goal (Track.cs)

Track GUI Design

Track

Activity ListAccountSearch



Tracking Goal

Track ID

Primary Goal

Total Calories

MET

Body Weight

Time Taken

Save

Calculate

	TrackID	ActivityName	TotalCalBurn	TrackStatus	UserID	ActivityID	TrackDate	Goal
▶	Track-001	Jumping Ropes	1080	Complete	Customer-001	Activity-003	Wednesday, July...	1000
	Track-002	Walking	3600	Complete	Customer-001	Activity-001	Thursday, July 25...	3000
	Track-003	Jumping Ropes		Incomplete	Customer-001	Activity-003	Thursday, July 18...	700
	Track-004	Swimming	720	Complete	Customer-001	Activity-002	Wednesday, July...	300
	Track-005	Swimming		Incomplete	Customer-001	Activity-002	Wednesday, July...	800
	Track-006	Pushups	3200	Complete	Customer-001	Activity-006	Thursday, July 25...	2000
	Track-007	Swimming		Incomplete	Customer-001	Activity-002	Monday, July 15, ...	3000
	Track-010	Pushups		Incomplete	Customer-001	Activity-006	Thursday, July 18...	4000
•								

Track.cs Code File

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```

using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class Track : Form
    {
        FitnessDataSetTableAdapters.TrackTableAdapter track = new
        FitnessDataSetTableAdapters.TrackTableAdapter();

        FitnessDataSetTableAdapters.ActivitiesTableAdapter activity = new
        FitnessDataSetTableAdapters.ActivitiesTableAdapter();

        DataTable trackdta = new DataTable();

        public void ClearTrackForm(){
            textBoxBodyWeight.Text = "";
            textBoxBodyWeight.Text = "";
            textBoxTimeTaken.Text = "";
        }

        public Track()
        {
            InitializeComponent();

            private void Track_Load(object sender, EventArgs e)
            {
                // TODO: This line of code loads data into the 'fitness_tracker.Track' table. You can move, or remove
                it, as needed.

```

```

        this.trackTableAdapter.Fill(this.fitnessDataSet.Track);

        dgvTrack.DataSource = track.GetDataByUserID(CustomerLogin.CID);

        dgvTrack.Refresh();
    }

    private void label5_Click(object sender, EventArgs e) { }

    private void textBoxMet_TextChanged(object sender, EventArgs e) { }

    private void textBoxTrackID_TextChanged(object sender, EventArgs e) { }

    private void groupBox1_Enter(object sender, EventArgs e) { }

    private void dgvTrack_CellClick(object sender, DataGridViewCellEventArgs e)
    {
        int row = dgvTrack.CurrentRow.Index;

        textBoxTrackID.Text = dgvTrack[0, row].Value.ToString();

        textBoxPrimaryGoal.Text = dgvTrack[7, row].Value.ToString();

        dgvTrack.Refresh();
    }

    private void btnCalculate_Click(object sender, EventArgs e)
    {
        int met = Convert.ToInt32(textBoxMet.Text);

        int weight = Convert.ToInt32(textBoxBodyWeight.Text) ;

        int hour = Convert.ToInt32(textBoxTimeTaken.Text) ;

        int result = met * weight * hour;

        textBoxTotalCalories.Text = Convert.ToString(result);
    }

    private void btnSave_Click(object sender, EventArgs e)

```

```

{
    if (textBoxPrimaryGoal.Text == "")
    {
        MessageBox.Show("Goal field must not be empty");
        textBoxPrimaryGoal.Focus();
    }
    else if (textBoxBodyWeight.Text == "") {
        MessageBox.Show("Body weight must not be empty");
        textBoxBodyWeight.Focus();
    }
    else if (textBoxTimeTaken.Text == "") {
        MessageBox.Show("Time taken must not be empty");
        textBoxTimeTaken.Focus();
    } else
    {
        int Goal = Convert.ToInt32(textBoxPrimaryGoal.Text);
        if (Convert.ToInt32(textBoxTotalCalories.Text) >= Goal)
        {
            track.UpdateTrackInformation("Complete", Convert.ToInt32(textBoxTotalCalories.Text),
            textBoxTrackID.Text);
            MessageBox.Show("Your calories burning goal is complete!");
            dgvTrack.DataSource = track.GetDataByUserID(CustomerLogin.CID);
            dgvTrack.Refresh();
            ClearTrackForm();
        }
        else
        {
            MessageBox.Show("Sorry! You cannot get your Goal! Try Again");
        }
    }
}
}

```

```

private void dgvTrack_CellContentClick(object sender, DataGridViewCellEventArgs e) { }

private void logoutToolStripMenuItem_Click(object sender, EventArgs e)
{
    FitnessTrackerLoad mainPage = new FitnessTrackerLoad();
    this.Hide();
    mainPage.ShowDialog();
}

private void activityListToolStripMenuItem_Click(object sender, EventArgs e)
{
    ActivitiesList activityList = new ActivitiesList();
    this.Hide();
    activityList.ShowDialog();
}

private void searchToolStripMenuItem_Click(object sender, EventArgs e)
{
    CustomerSearch searchList = new CustomerSearch();
    this.Hide();
    searchList.ShowDialog();
}
}
}

```

## 11. Activity Name Search (CustomerSearch.cs)

### Customer Search GUI Design

The screenshot shows a Windows application window titled "CustomerSearch". It has a menu bar with "Activity List", "Account", and "Search". The main content area is titled "Search By Activity Name". Below the title is a search input field and a "Search" button. Below the search area is a table with 8 columns: TrackID, ActivityName, TotalCalBurn, TrackStatus, UserID, ActivityID, and TrackDate. The table contains 8 rows of data. The first row is highlighted in blue. Below the table is a scroll bar.

	TrackID	ActivityName	TotalCalBurn	TrackStatus	UserID	ActivityID	TrackDate
▶	Track-001	Jumping Ropes	1080	Complete	Customer-001	Activity-003	Wednesday, July
	Track-002	Walking	3600	Complete	Customer-001	Activity-001	Thursday, July 25
	Track-003	Jumping Ropes		Incomplete	Customer-001	Activity-003	Thursday, July 18
	Track-004	Swimming	720	Complete	Customer-001	Activity-002	Wednesday, July
	Track-005	Swimming		Incomplete	Customer-001	Activity-002	Wednesday, July
	Track-006	Pushups	3200	Complete	Customer-001	Activity-006	Thursday, July 25
	Track-007	Swimming		Incomplete	Customer-001	Activity-002	Monday, July 15,
*							

### CustomerSearch.cs File Code

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;
```

```

using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class CustomerSearch : Form
    {
        FitnessDataSetTableAdapters.TrackTableAdapter track = new
        FitnessDataSetTableAdapters.TrackTableAdapter();

        DataTable dtaTrack = new DataTable();

        public static string trackid, activityName, uid;

        public CustomerSearch()
        {
            InitializeComponent();
        }

        private void label1_Click(object sender, EventArgs e) { }

        private void CustomerSearch_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the 'fitnessDataSet.Track' table. You can move, or remove
            it, as needed.

            this.trackTableAdapter.Fill(this.fitnessDataSet.Track);

            dgvSearchDisplay.DataSource = track.GetDataByUserID(CustomerLogin.CID);

            dgvSearchDisplay.Refresh();
        }

        private void btnSearch_Click(object sender, EventArgs e)
        {
            try {

```

```

        dgvSearchDisplay.DataSource = track.SearchByActivityName(CustomerLogin.CID,
textBoxActivitySearch.Text);

    }

    catch (Exception ex) {
        MessageBox.Show("Something went wrong.", ex.Message);
    }
}

private void textBoxActivitySearch_TextChanged(object sender, EventArgs e) { }

private void activityListToolStripMenuItem_Click(object sender, EventArgs e)
{
    ActivitiesList list = new ActivitiesList();
    this.Hide();
    list.ShowDialog();
}

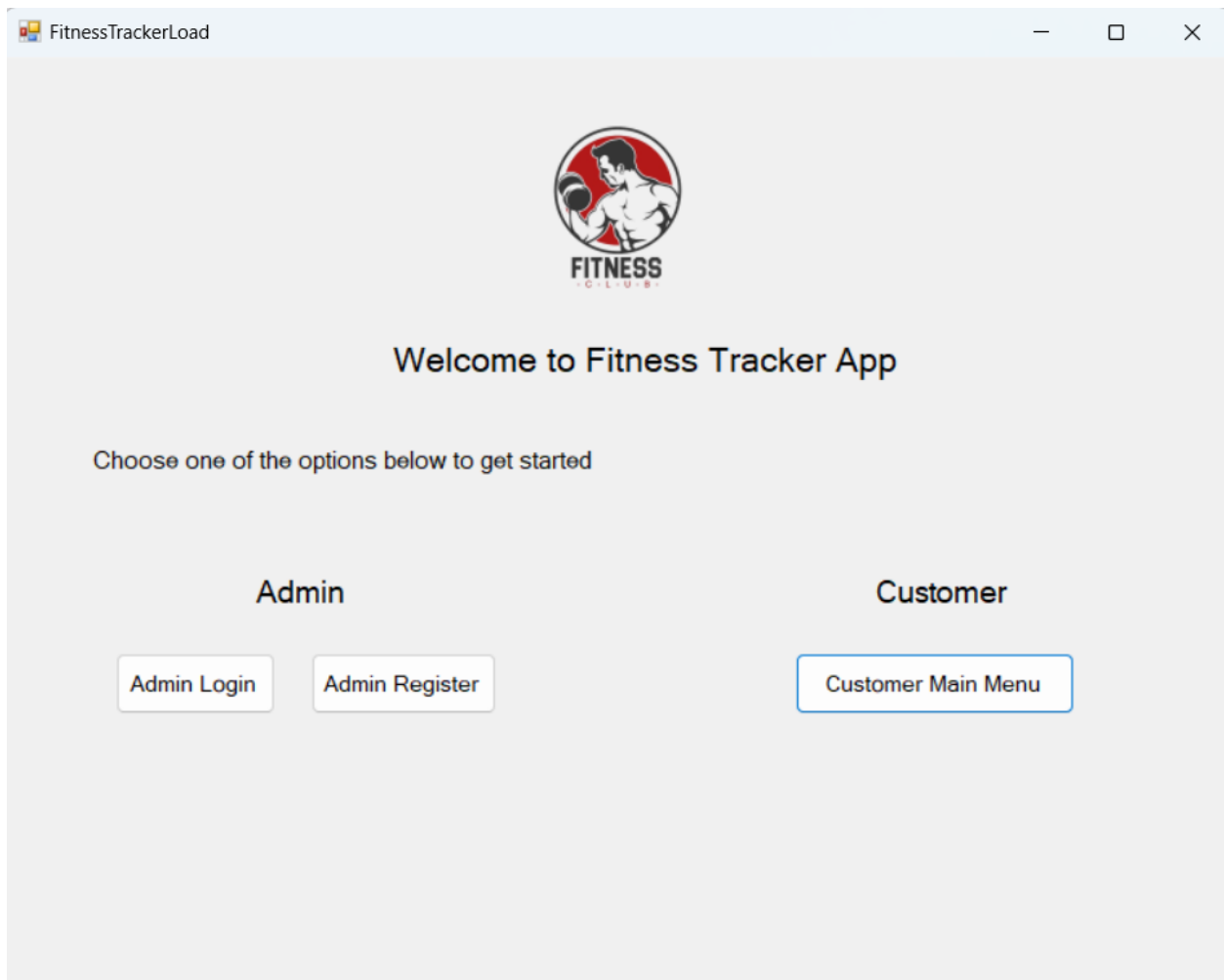
private void logoutToolStripMenuItem_Click(object sender, EventArgs e)
{
    FitnessTrackerLoad mainPage = new FitnessTrackerLoad();
    this.Hide();
    mainPage.ShowDialog();
}
}
}

```

## 12. Main Page for Admin and Customer (FitnessTrackerLoad.cs)



## Fitness Tracker Load Page GUI Design



## FitnessTrackerLoad.cs Code File

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FitnessTrackerApp
{
    public partial class FitnessTrackerLoad : Form
    {
        public FitnessTrackerLoad()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            AdminLogin AdminLoginForm = new AdminLogin();
            this.Hide();
            AdminLoginForm.ShowDialog();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            AdminRegister AdminRegisterForm = new AdminRegister();
            this.Hide();
            AdminRegisterForm.ShowDialog();
        }
    }
}

```

```

    }

    private void button3_Click(object sender, EventArgs e)
    {
        CustomerLogin CustomerLoginForm = new CustomerLogin();
        this.Hide();
        CustomerLoginForm.ShowDialog();
    }

    private void button4_Click(object sender, EventArgs e)
    {
        CustomerRegister CustomerRegisterForm = new CustomerRegister();
        this.Hide();
        CustomerRegisterForm.ShowDialog();
    }

    private void button5_Click(object sender, EventArgs e)
    {
        CustomerMainMenu CustomerMainMenuPage = new CustomerMainMenu();
        this.Hide();
        CustomerMainMenuPage.ShowDialog();
    }

    private void FitnessTrackerLoad_Load(object sender, EventArgs e) {

    }

}

```

## b. Quality of the program

### 1.Program Structure

The program is organized to be easy to maintain and expand. We use short, clear names for folders, making them easy to understand. Most functions are designed to achieve specific goals, which makes them reusable. By writing functions, we can reuse them in different parts of the program without repeating the same code.

### OOP and Encapsulation

#### Encapsulation

Encapsulation means grouping data and the methods that work with that data, while keeping some parts private. This helps protect the data and ensures it stays accurate. Below is an example of a class called `ClsActivity` that uses encapsulation:

#### Code

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

namespace FitnessTrackerApp
{
    class ClsActivity
    {
        String ActivityID, ActivityName, MetricOne, MetricTwo, MetricThree;

        public String ActID
        {
            get { return ActivityID; }
            set { ActivityID = value; }
        }
        public String ActName
        {
            get { return ActivityName; }
            set { ActivityName = value; }
        }
        public String ActMetricOne
        {
            get { return MetricOne; }
            set { MetricOne = value; }
        }
        public String ActMetricTwo
        {
            get { return MetricTwo; }
            set { MetricTwo = value; }
        }
        public String ActMetricThree
        {
```

```
        get { return MetricThree; }  
        set { MetricThree = value; }  
    }  
}
```

## Function

A function is a set of code that can perform a task and return a result. Functions can be called multiple times with different inputs to get different outputs. They help keep the code organized and easy to read. Here's an example of a function that takes users to different pages:

### Code

```
private void NavigateToActivitiesList(){  
    ActivitiesList ActivitiesListForm = new ActivitiesList();  
    this.Hide();  
    ActivitiesListForm.ShowDialog();  
}
```

## Selection

Selection allows the program to make decisions based on certain conditions. For example, we check if a user input is empty. If it is, we show an error message; if not, we continue. Here's how it looks:

### Code

```
if (adminUserNameTextBox.Text == "") {  
    MessageBox.Show("User name can't be empty");  
    adminUserNameTextBox.Focus();  
}  
else if (adminPasswordTextBox.Text == "")  
{  
    MessageBox.Show("Password can't be empty");  
    adminPasswordTextBox.Focus();  
}
```

## Object

Objects are collections of key-value pairs used to store different types of data. They can hold various data types, such as numbers or text. Objects are useful for keeping related information together. Here's an example of creating a customer object:

### Code

```
ClsCustomer customerObj = new ClsCustomer();  
customerObj.CusID = customerIDTextBox.Text;  
customerObj.CusName = customerNameTextBox.Text;  
customerObj.CusPassword =customerPasswordTextBox.Text;  
customerObj.CusUName =customerUserNameTextBox.Text;  
customerObj.CusEmail =customerEmailTextBox.Text
```

## Readability

We use clear and descriptive names for all variables so that developers can easily understand what they do. Comments are added to complex functions to explain their purpose. For instance, the function `NavigateToActivitiesList` clearly indicates its function:

```
private void NavigateToActivitiesList(){  
    ActivitiesList ActivitiesListForm = new ActivitiesList();  
    this.Hide();  
    ActivitiesListForm.ShowDialog();  
}
```

## Usability

The program is designed to be user-friendly, making it easy for users to navigate and find what they need. The layout and colors are consistent, helping users understand how to use the program without needing a manual. Overall, the design is intuitive and enhances the user experience.

## Task – 2

### Testing Plan

#### Purpose of Testing

Testing is an essential aspect of software development, serving multiple purposes that ensure the application functions correctly and meets user expectations. The primary goals of testing include:

- **Defect Identification:** Detecting and addressing bugs before the software is deployed to production.
- **Functionality Verification:** Ensuring that the software performs its intended tasks accurately.
- **Input Validation:** Confirming that the application can handle both expected and unexpected inputs without failure.

#### Types of Testing

To achieve comprehensive coverage, various testing methodologies were employed:

- **Black Box Testing:** This method focuses on testing the software's functionality without knowledge of the internal code structure. It emphasizes user experience by validating inputs and outputs against specified requirements.
- **White Box Testing:** This approach involves testing the internal logic of the code. It requires familiarity with the codebase to ensure that the software's logic is sound, error handling is effective, and the code is optimized.

#### Test Data Selection and Execution

The test data was carefully selected to cover a wide range of scenarios, including valid inputs, invalid inputs, boundary conditions, and exceptional cases. For example, when testing username and password fields, null values, invalid characters, and incorrect combinations were used to ensure proper validation and error handling.

The tests were executed systematically, following the defined test cases and scripts. Each test case was run multiple times to ensure consistency and reliability of results. Any issues or bugs discovered during testing were documented, analyzed, and fixed before proceeding to the next test case.

The test plan and execution were comprehensive, covering both functional and non-functional requirements of the application. The combination of black box testing, white box testing, and exception handling provided a thorough assessment of the application's behavior and robustness.

## Black Box Testing

### Admin Login Form

No.	Test Case	Form Load	Date	Time	Purpose
1.	Username	AdminLogin	July 1	09:00 – 09:15	Ensures username is not empty and the database does actually have the user registered with the provided name.
2.	Password	AdminLogin	July 1	09:15 – 09:30	Ensure password is not empty and checks the database to make sure the provided password is same as the one used when registering an account.

### Admin Registration Form

No.	Test Case	Form Load	Date		Time	Purpose
1.	AdminID	AdminRegister	July 1		9:30 - 9:45	Check if the software automatically generates the relevant and unique id for admin
2.	Admin Name	Admin Register	July 1		9:45 - 10:00	Ensure that the admin can have its own unique name apart from its username which is unique
3.	Admin Username	AdminRegister	July 1		10:00 – 10:15	Ensures username is not empty and username can only contain letters and numbers according to the assignment criteria.
4.	Admin Password	AdminRegister	July 1		10:30 – 10:45	Ensure password is not empty and make sure to have at least 12 characters including 1 uppercase letter and 1 lowercase letter according to the assignment criteria
5.	Admin Email	AdminRegister	July 1		10:45 – 11:00	Ensures email is present, unique and is actually a valid email and belongs to that user.



### Activity

No.	Test Case	Form Load	Date	Time	Purpose
1.	AdminID				
1.	Activity Name	Activity	July 3	09:00 – 09:15	Ensures activity name is not empty.
2.	Metric One	Activity	July 3	09:15 – 09:30	Ensures metric one is not empty.
3.	Metric Two	Activity	July 3	09:45 – 10:00	Ensures metric two is not empty.
4.	Metric Three	Activity	July 3	10:00 – 10:15	Ensures metric three is not empty.

### Customer Register

No.	Test Case	Form Load	Date	Time	Purpose
1.	AdminID				
1.	UserName	CustomerRegister	July 4	09:00 – 09:15	Ensures username is not empty and contains the characters such as number and alphabets.
2.	Password	CustomerRegister	July 4	09:15 – 09:30	Ensure password is not empty and meets all the criteria such as contains uppercase letter, lowercase letter, one number and others.
3.	Admin Email	CustomerRegister	July 4	09:45 – 10:00	Ensures email is present, unique and is actually a valid email and belongs to that user.

### Customer Login

No.	Test Case	Form Load	Date	Time	Purpose
1.	AdminID				
1.	UserName	CustomerLogin	July 5	09:00 – 09:15	Ensures username is not empty and the database does actually have the user registered with the provided name
2.	Password	CustomerLogin	July 5	09:15 – 09:30	Ensure password is not empty and checks the database to make sure the provided password is same as the one used when registering an account

### SetGoal

No.	Test Case	Form Load	Date	Time	Purpose
1.	AdminID				
1.	Choose Activity	SetGoal	July 5	09:00 – 09:15	Ensures this field is not empty. Checks the option selected is valid and only pre-defined activity created by admin through activity form.
2.	Activity ID	SetGoal	July 5	09:15 – 09:30	No validation, since this is auto filled based on selected activity.
3.	Track ID	SetGoal	July 5	09:30 – 09:45	No validation, since this is auto filled based on selected activity.
4.	Your Goal	SetGoal	July 5	09:45 – 10:00	Ensures the field is not empty
5.	Deadline	SetGoal	July 5	10:00 – 10:15	Ensures this field is not empty and selected value is valid date.

### Track Goal

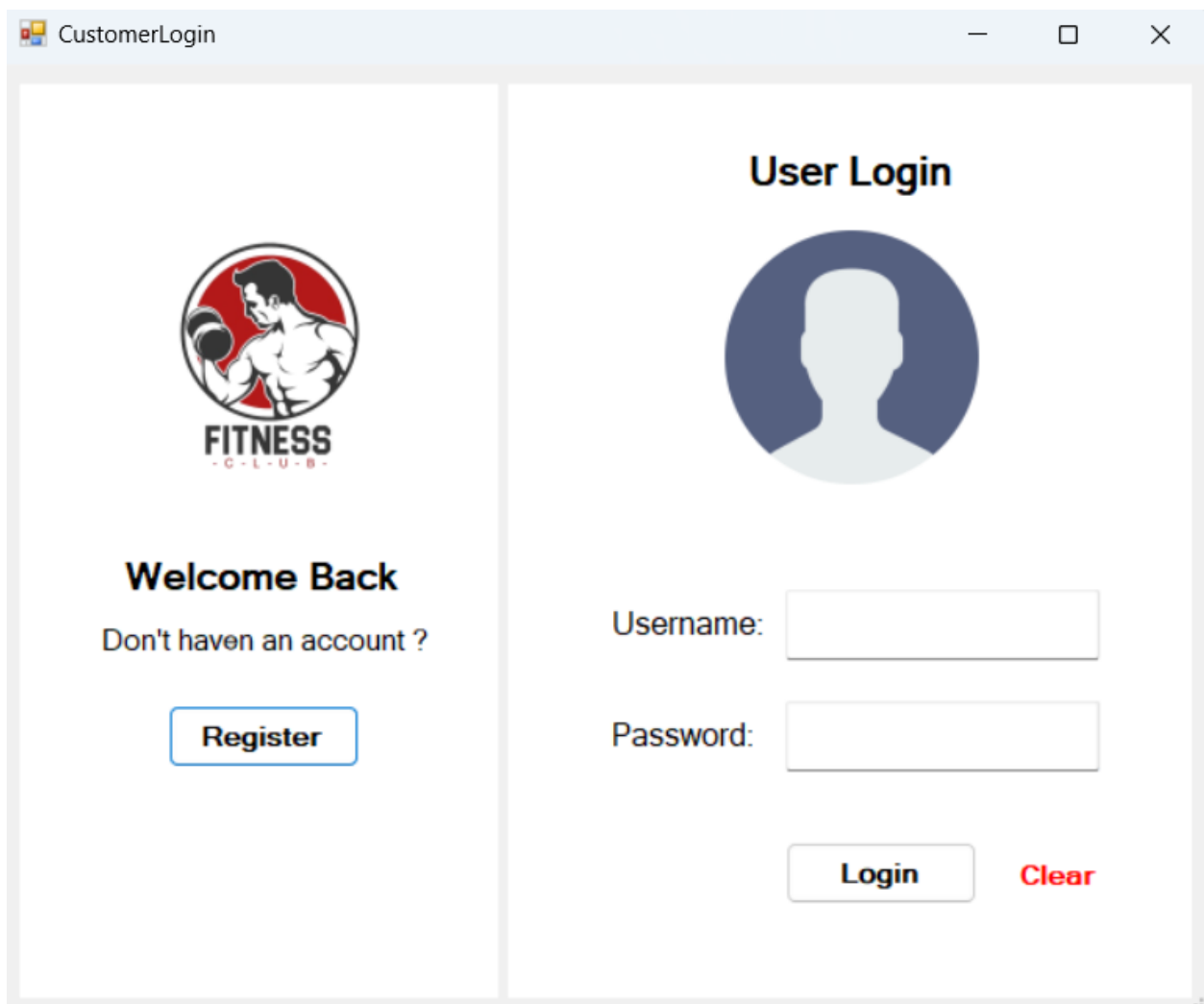
No.	Test Case	Form Load	Date	Time	Purpose
1.	AdminID				
1.	Track ID	TrackGoal	July 5	09:00 – 09:15	Ensures this field is not empty. Checks the option selected is valid and only pre-defined activity created by admin through activity form.
2.	Primary Goal	TrackGoal	July 5	09:15 – 09:30	No validation, since this is auto filled based on selected activity.
3.	Track ID	TrackGoal	July 5	09:30 – 09:45	No validation, since this is auto filled based on selected activity.
4.	Your Goal	TrackGoal	July 5	09:45 – 10:00	Ensures the field is not empty
5.	Deadline	TrackGoal	July 5	10:00 – 10:15	Ensures this field is not empty and selected value is valid date.

### Test Script - Unit Testing

## Admin Login Form

Test Case	Fig 1.1
Test Objective	Username is valid
Procedure	Go to login form, leave the username field empty and submit the form
Test Data	Testing null value
Expected Result	User should get an error dialog saying username is empty.
Actual Result	As shown in Fig 1.1

## Before Testing

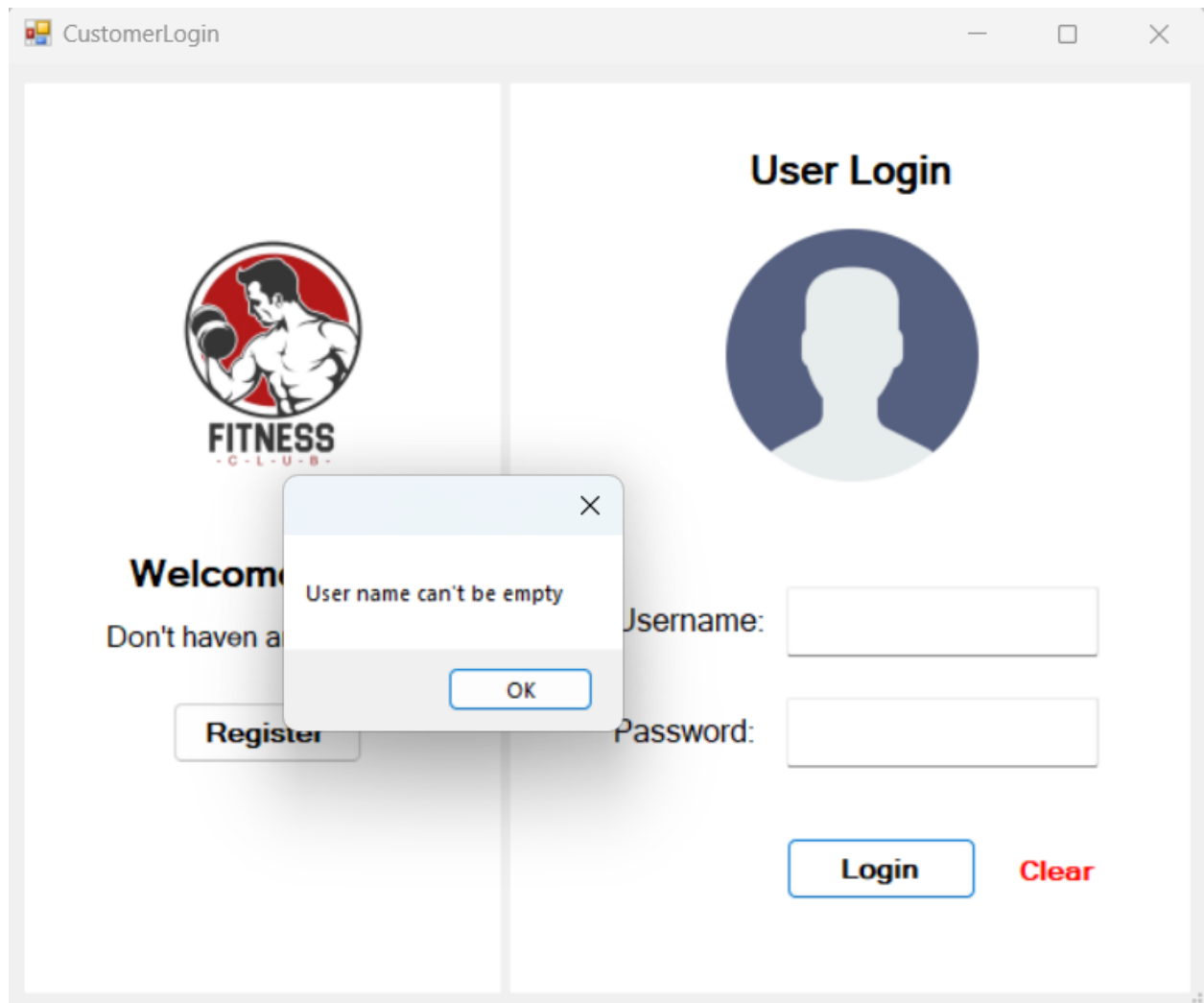


The screenshot shows a web application window titled "CustomerLogin". The interface is split into two main sections. On the left, there is a logo for "FITNESS CLUB" featuring a muscular man holding a dumbbell. Below the logo, it says "Welcome Back" and "Don't haven an account ?" (note the typo). There is a blue "Register" button. On the right, the section is titled "User Login". It features a large blue circular placeholder for a user profile picture. Below this, there are two input fields: "Username:" and "Password:". At the bottom of the login section, there is a "Login" button and a "Clear" link in red text.

Fig 1.1

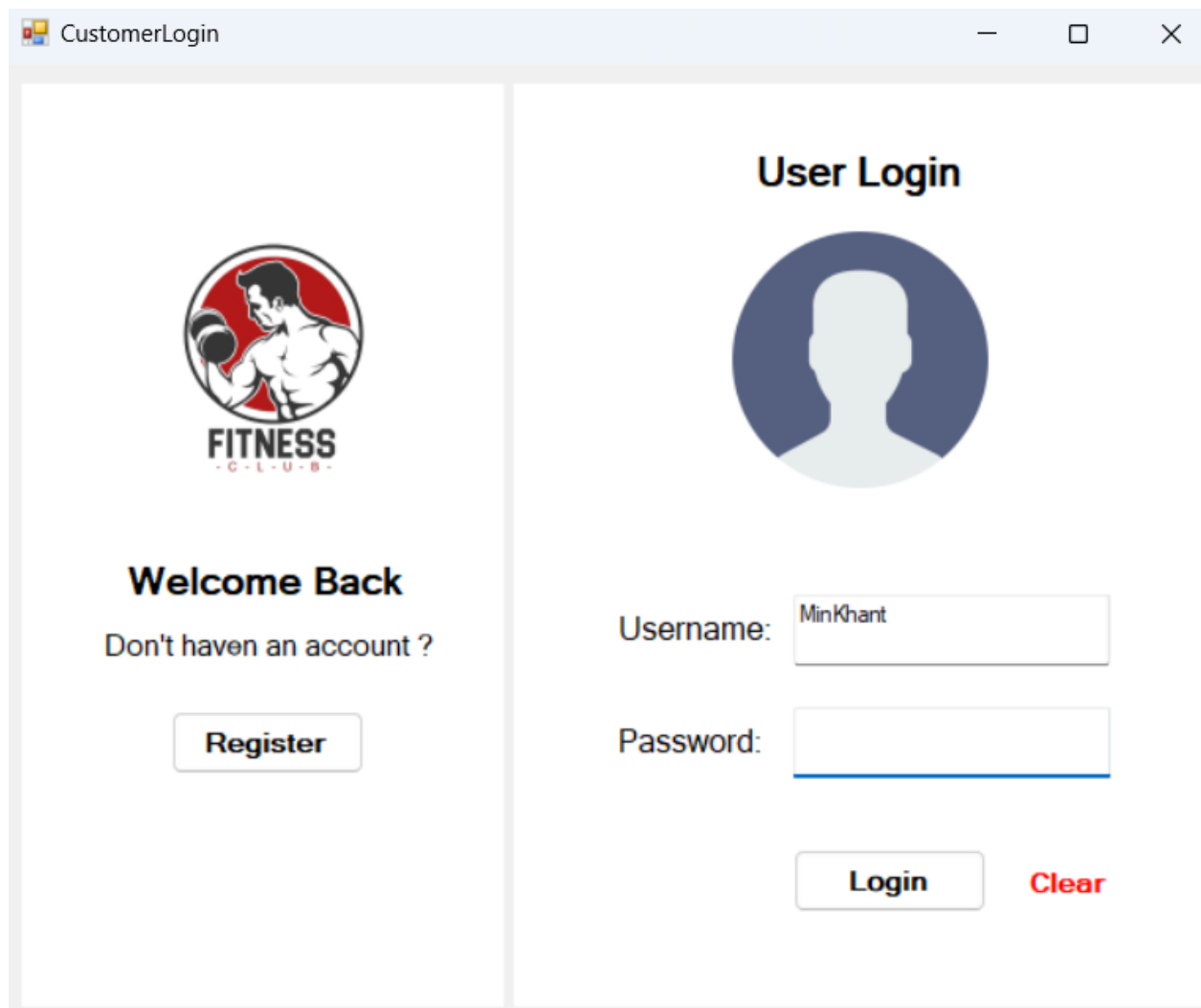
## After Testing

Fig 1.1



Test Case	Fig 1.2
Test Objective	Testing if password is empty
Procedure	Go to login form and submit the form with password field empty
Test Data	Testing null password field
Expected Result	Error dialog should popup saying password is required
Actual Result	As shown in Fig 1.2

Before



The screenshot shows a web application window titled "CustomerLogin". The page is divided into two main sections. On the left, there is a "Welcome Back" message with a "Register" button and a link "Don't have an account?". On the right, there is a "User Login" section with a user profile icon, a "Username:" field containing "MinKhant", a "Password:" field, and "Login" and "Clear" buttons.

**CustomerLogin**

**User Login**

**WELCOME BACK**

Don't have an account ?

**Register**

Username: MinKhant

Password:

**Login** **Clear**

Fig 1.2

After

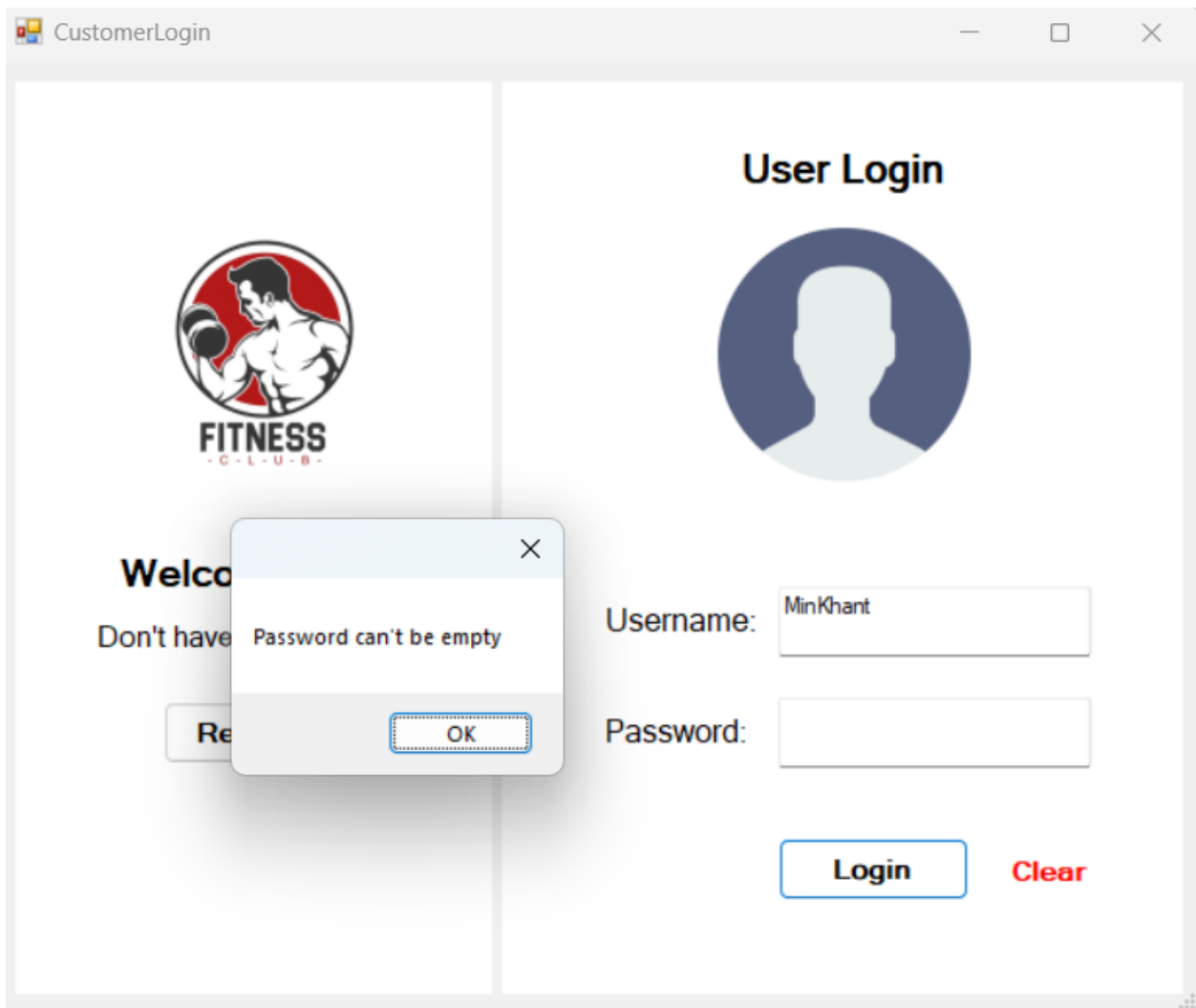
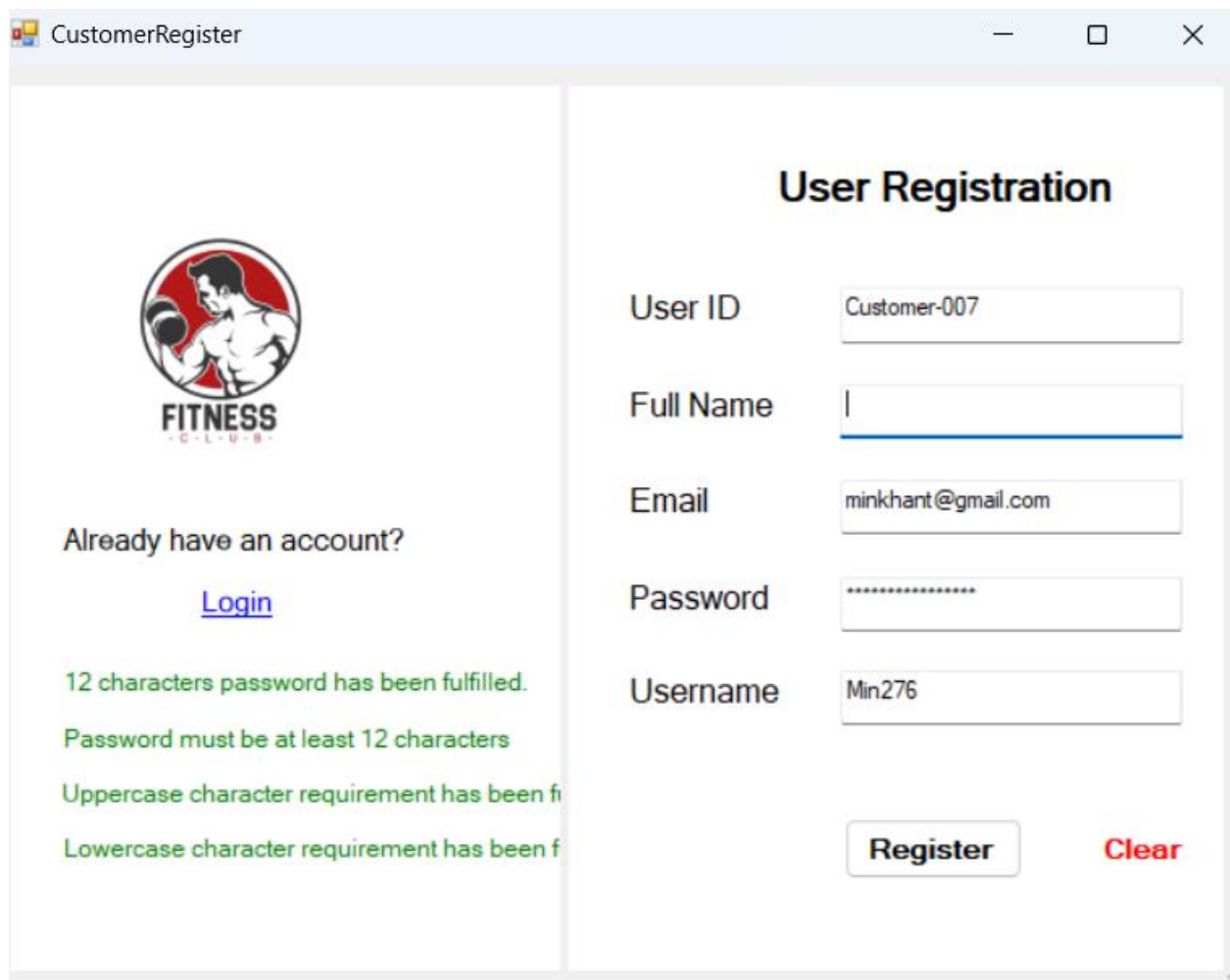


Fig 1.2

## Customer Register

Test Case	Fig 1.4
Test Objective	Customer name should not be empty
Procedure	Go to register form and submit the form without Customer Name
Test Data	Testing null value for Customer Name
Expected Result	Form shouldn't be submitted and should get error dialog
Actual Result	As shown in Fig 1.4

Before Testing (Fig 1.4)



The screenshot shows a web application window titled "CustomerRegister". On the left side, there is a logo for "FITNESS CLUB" featuring a muscular man holding a basketball. Below the logo, the text "Already have an account?" is followed by a blue "Login" link. Further down, there are four green status messages: "12 characters password has been fulfilled.", "Password must be at least 12 characters", "Uppercase character requirement has been fi", and "Lowercase character requirement has been f". On the right side, the heading "User Registration" is displayed. Below it are five input fields: "User ID" with the value "Customer-007", "Full Name" with a single character "I", "Email" with the value "minkhant@gmail.com", "Password" with masked characters "\*\*\*\*\*", and "Username" with the value "Min276". At the bottom right of the form are two buttons: "Register" and "Clear".

After Testing (Fig 1.4)

The screenshot displays a web application window titled "CustomerRegister". On the left side, there is a modal dialog box with a close button (X) in the top right corner. The dialog contains the text "Customer name must not be empty" and an "OK" button at the bottom right. Below the dialog, the text "Already have an account?" is followed by a blue underlined link "Login". Further down, four green messages are listed: "12 characters password has been fulfilled.", "Password must be at least 12 characters", "Uppercase character requirement has been fulfilled", and "Lowercase character requirement has been fulfilled". On the right side, the "User Registration" form is visible. It includes five input fields: "User ID" (containing "Customer-007"), "Full Name" (empty), "Email" (containing "minkhant@gmail.com"), "Password" (containing "\*\*\*\*\*"), and "Username" (containing "Min276"). At the bottom of the form, there are two buttons: a blue "Register" button and a red "Clear" button.

CustomerRegister

### User Registration

User ID

Full Name

Email

Password

Username

Customer name must not be empty

OK

Already have an account?

[Login](#)

12 characters password has been fulfilled.

Password must be at least 12 characters

Uppercase character requirement has been fulfilled

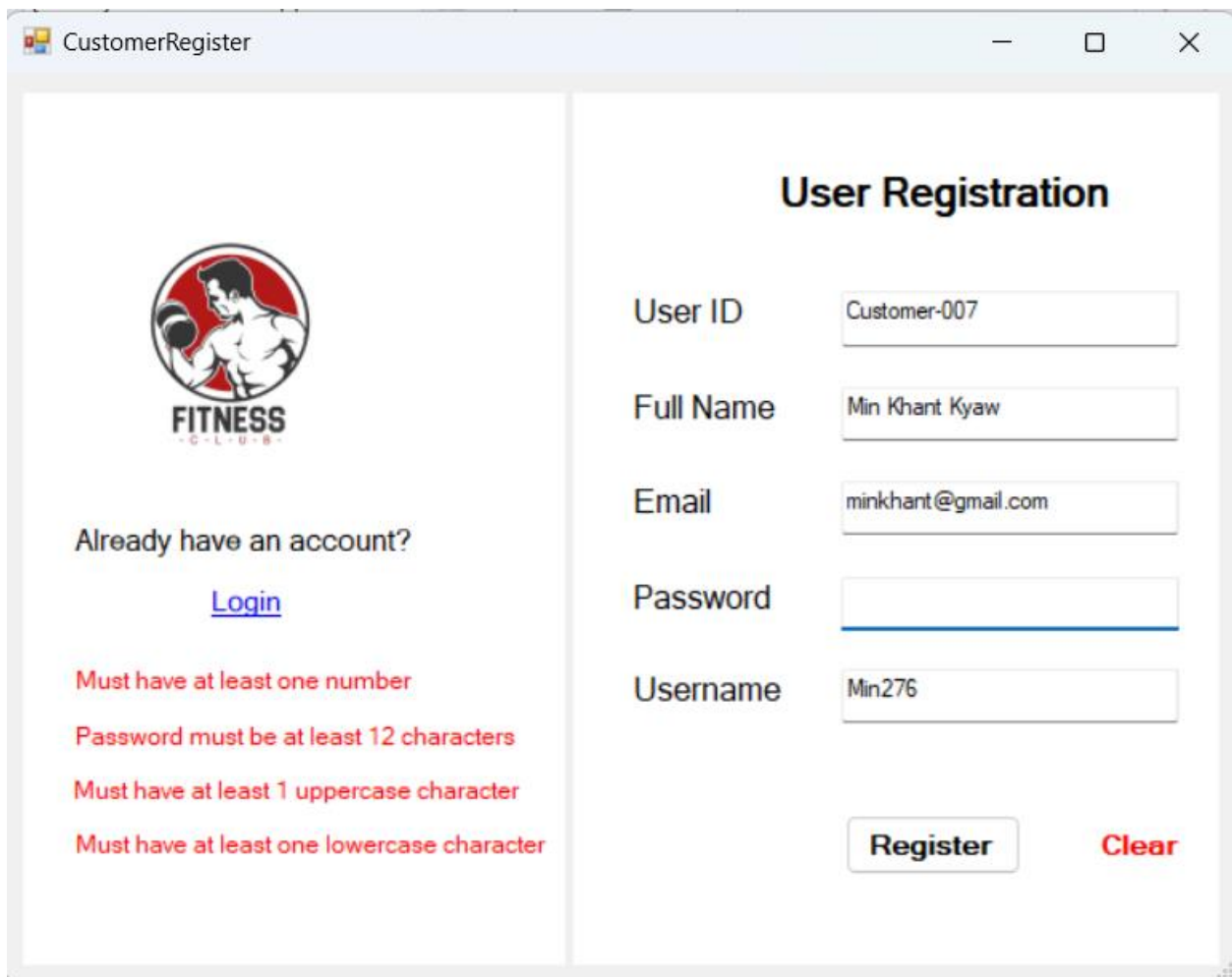
Lowercase character requirement has been fulfilled



## Customer Password

Test Case	Fig 1.5
Test Objective	Testing is password field is empty
Procedure	Go to register form and submit without password
Test Data	Testing null password value
Expected Result	User should get a popup saying password can't be empty
Actual Result	As shown in Fig 1.5

Before Testing (Fig 1.5)



The screenshot shows a web application window titled "CustomerRegister". The page is divided into two main sections. On the left, there is a logo for "FITNESS CLUB" featuring a muscular man holding a dumbbell. Below the logo, the text "Already have an account?" is followed by a blue "Login" link. Further down, four red error messages are listed: "Must have at least one number", "Password must be at least 12 characters", "Must have at least 1 uppercase character", and "Must have at least one lowercase character". On the right, the "User Registration" form is displayed. It contains five input fields: "User ID" (with the value "Customer-007"), "Full Name" (with the value "Min Khant Kyaw"), "Email" (with the value "minkhant@gmail.com"), "Password" (which is empty and has a blue underline), and "Username" (with the value "Min276"). At the bottom right of the form, there are two buttons: "Register" and "Clear".

After Testing (Fig 1.5)

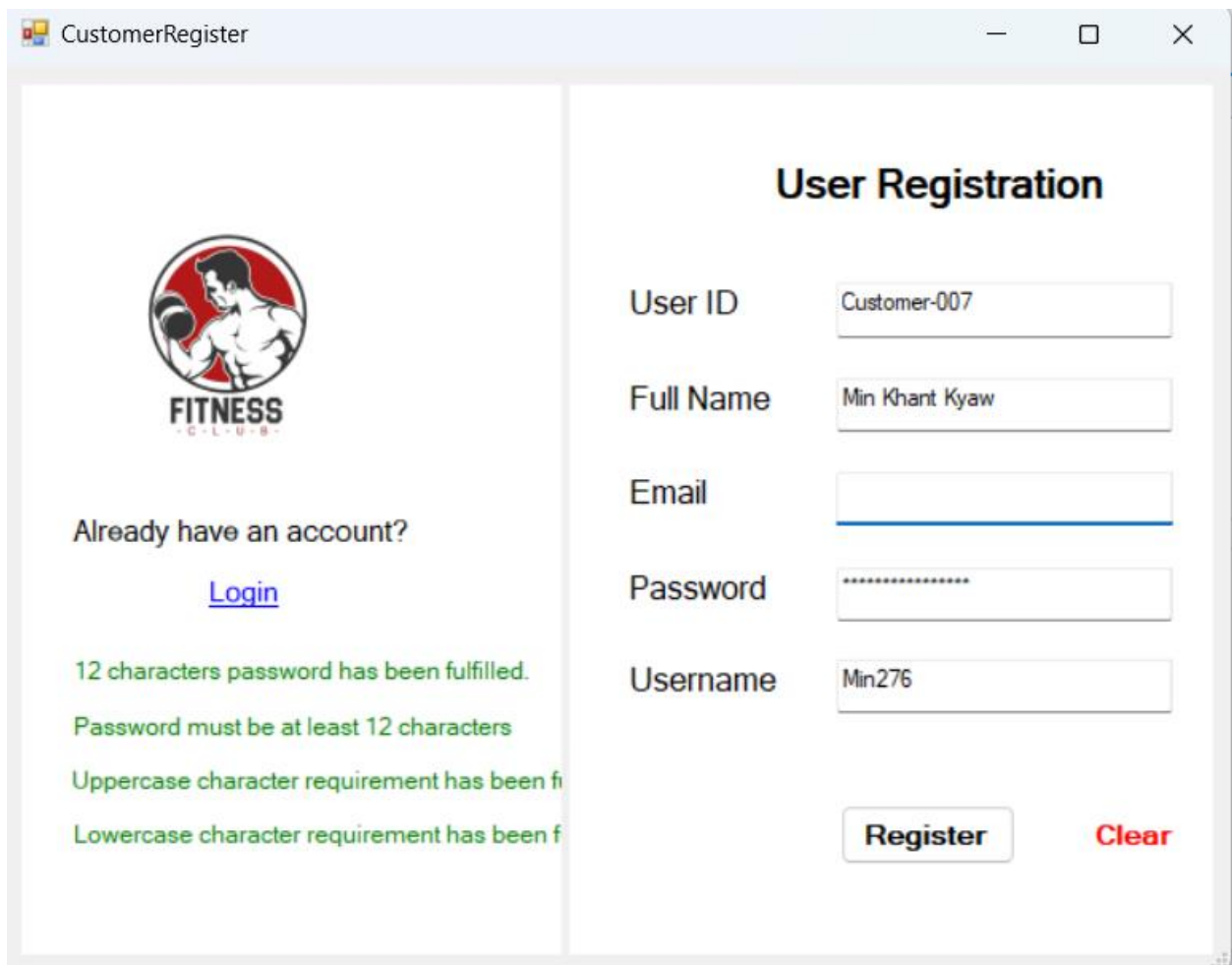
The screenshot shows a web application window titled "CustomerRegister". On the left, there is a "Login" link and a list of password requirements in red text: "Must have at least one number", "Password must be at least 12 characters", "Must have at least 1 uppercase character", and "Must have at least one lowercase character". A "Warning" dialog box is open in the foreground, displaying a red 'X' icon and the message "Customer password is not valid", with "OK" and "Cancel" buttons. On the right, the "User Registration" form contains the following fields and values: "User ID" (Customer-007), "Full Name" (Min Khant Kyaw), "Email" (minkhant@gmail.com), "Password" (empty), and "Username" (Min276). At the bottom right of the form are "Register" and "Clear" buttons.

User Registration	
User ID	Customer-007
Full Name	Min Khant Kyaw
Email	minkhant@gmail.com
Password	
Username	Min276
<b>Register</b> <b>Clear</b>	

## Customer Email

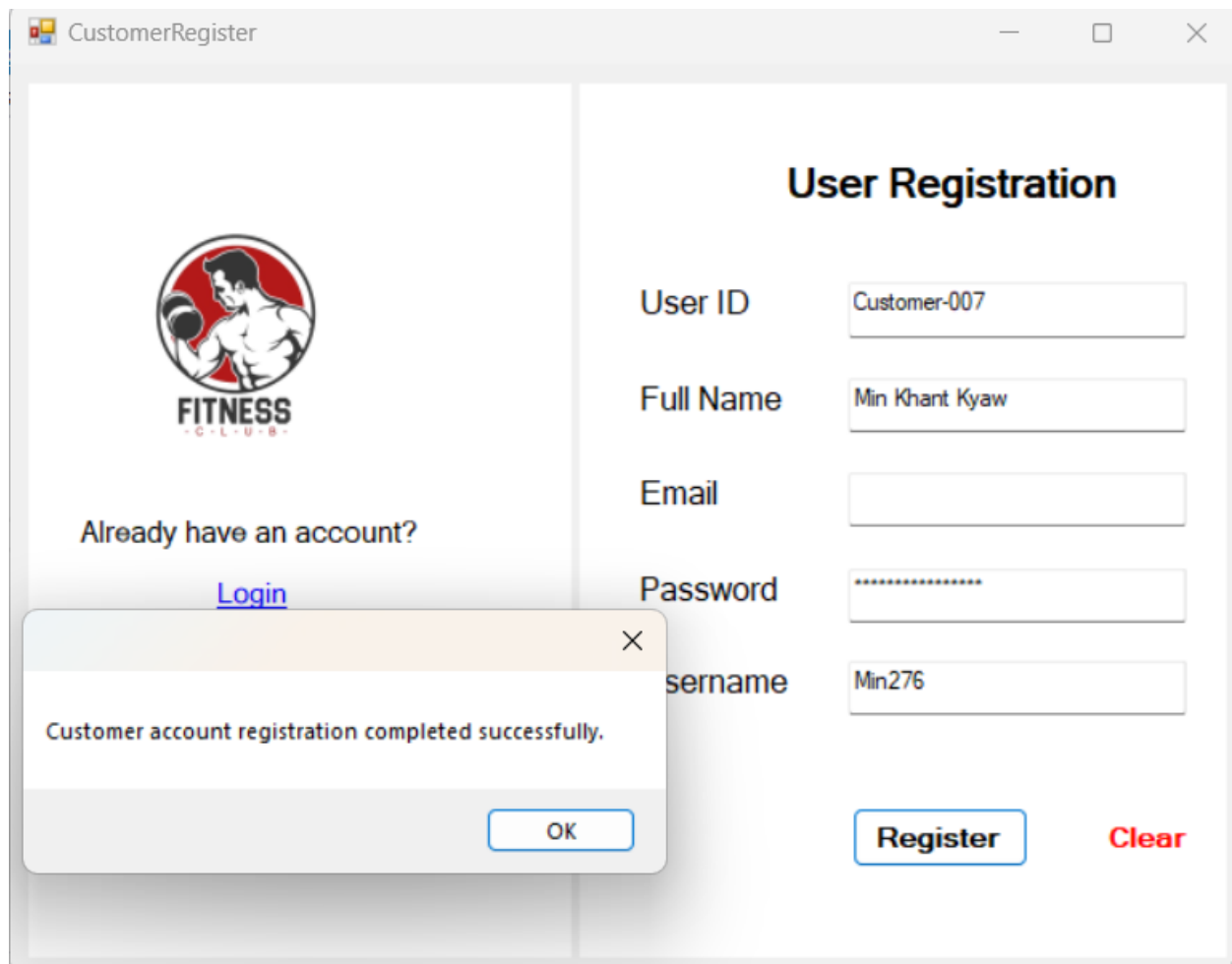
Test Case	Fig 1.6
Test Objective	Testing if email field is empty
Procedure	Go to register form and submit without email
Test Data	Testing null email value
Expected Result	User should get a popup saying email can't be empty
Actual Result	As shown in Fig 1.6

Before Testing (Fig 1.6)



The screenshot shows a web application window titled "CustomerRegister". On the left side, there is a logo for "FITNESS CLUB" featuring a muscular man holding a dumbbell. Below the logo, the text "Already have an account?" is followed by a blue "Login" link. Further down, four green messages indicate password requirements: "12 characters password has been fulfilled.", "Password must be at least 12 characters", "Uppercase character requirement has been fulfilled", and "Lowercase character requirement has been fulfilled". On the right side, the "User Registration" form is displayed. It contains five input fields: "User ID" (filled with "Customer-007"), "Full Name" (filled with "Min Khant Kyaw"), "Email" (empty), "Password" (filled with "\*\*\*\*\*"), and "Username" (filled with "Min276"). At the bottom right of the form are two buttons: "Register" and "Clear".

After Testing (Fig 1.6)



The screenshot displays a web application window titled "CustomerRegister". On the left side, there is a logo for "FITNESS CLUB" featuring a muscular man holding a dumbbell. Below the logo, the text "Already have an account?" is followed by a blue "Login" link. On the right side, the "User Registration" form is visible. It contains the following fields and values:

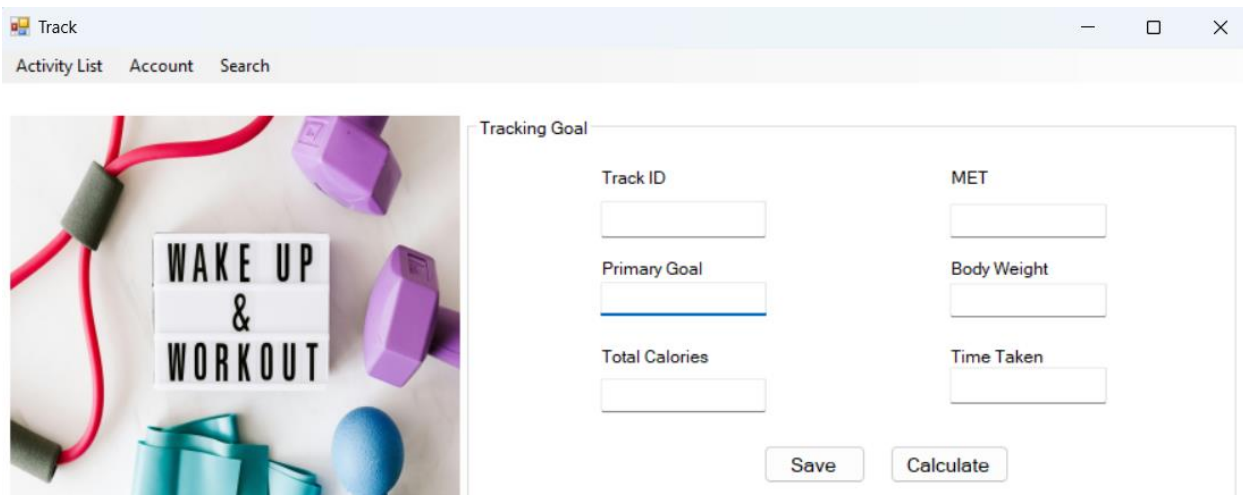
- User ID: Customer-007
- Full Name: Min Khant Kyaw
- Email: (empty)
- Password: (masked with asterisks)
- Username: Min276

At the bottom right of the form are two buttons: "Register" (blue) and "Clear" (red). A modal dialog box is overlaid on the left side of the form, displaying the message "Customer account registration completed successfully." with an "OK" button.

## Track Goal Form Testing

Test Case	Fig 1.7
Test Objective	Testing if primary field is empty
Procedure	Go to SetGoal form and submit without primary goal
Test Data	Testing null primary value
Expected Result	User should get a popup saying primary goal can't be empty
Actual Result	As shown in Fig 1.7

Before Testing (Fig 1.7)



Track

Activity List Account Search

Tracking Goal

Track ID

MET

Primary Goal

Body Weight

Total Calories


Time Taken

Save Calculate

	TrackID	ActivityName	TotalCalBurn	TrackStatus	UserID	ActivityID	TrackDate	Goal
▶	Track-001	Jumping Ropes	1080	Complete	Customer-001	Activity-003	Wednesday, July...	1000
	Track-002	Walking	3600	Complete	Customer-001	Activity-001	Thursday, July 25...	3000
	Track-003	Jumping Ropes		Incomplete	Customer-001	Activity-003	Thursday, July 18...	700
	Track-004	Swimming	720	Complete	Customer-001	Activity-002	Wednesday, July...	300
	Track-005	Swimming		Incomplete	Customer-001	Activity-002	Wednesday, July...	800
	Track-006	Pushups	3200	Complete	Customer-001	Activity-006	Thursday, July 25...	2000
	Track-007	Swimming		Incomplete	Customer-001	Activity-002	Monday, July 15, ...	3000
	Track-010	Pushups		Incomplete	Customer-001	Activity-006	Thursday, July 18...	4000
	Track-011	Swimming		Incomplete	Customer-001	Activity-002	Tuesday, July 16...	300
	Track-012	Pushups		Incomplete	Customer-001	Activity-006	Friday, July 16...	2000

## After Testing (Fig 1.7)

Track
Activity List Account Search



Tracking Goal

Track ID

MET

Primary Goal

Body Weight

Total Calories

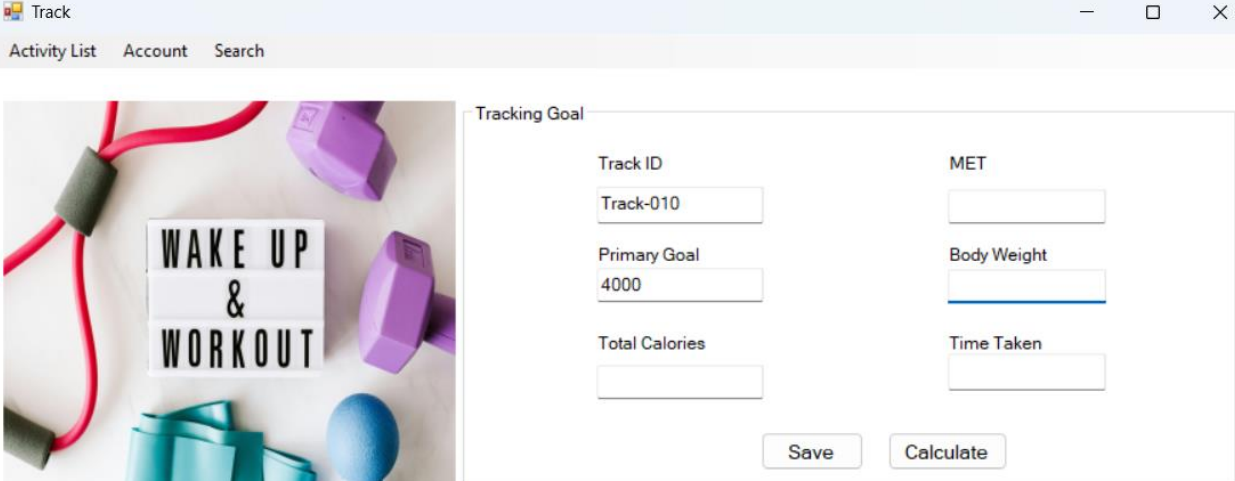
Time Taken

Goal field must not be empty

## Track Goal Form Testing

Test Case	Fig 1.8
Test Objective	Testing if body weight field is empty
Procedure	Go to SetGoal form and submit without body weight
Test Data	Testing null body weight value
Expected Result	User should get a popup saying body weight can't be empty
Actual Result	As shown in Fig 1.8

## Before Testing (Fig 1.7)



Track

Activity List Account Search

Tracking Goal

Track ID: Track-010

Primary Goal: 4000

Total Calories:

MET:

Body Weight:

Time Taken:

Save Calculate

	TrackID	ActivityName	TotalCalBurn	TrackStatus	UserID	ActivityID	TrackDate	Goal
	Track-001	Jumping Ropes	1080	Complete	Customer-001	Activity-003	Wednesday, July...	1000
	Track-002	Walking	3600	Complete	Customer-001	Activity-001	Thursday, July 25...	3000
	Track-003	Jumping Ropes		Incomplete	Customer-001	Activity-003	Thursday, July 18...	700
	Track-004	Swimming	720	Complete	Customer-001	Activity-002	Wednesday, July...	300
	Track-005	Swimming		Incomplete	Customer-001	Activity-002	Wednesday, July...	800
	Track-006	Pushups	3200	Complete	Customer-001	Activity-006	Thursday, July 25...	2000
	Track-007	Swimming		Incomplete	Customer-001	Activity-002	Monday, July 15, ...	3000
▶	Track-010	Pushups		Incomplete	Customer-001	Activity-006	Thursday, July 18...	4000
	Track-011	Swimming		Incomplete	Customer-001	Activity-002	Tuesday, July 16...	300



## After Testing (Fig 1.8)

Track

Activity List Account Search

**Tracking Goal**

Track ID:  MET:

Primary Goal:  Body Weight:

Total Calories:  Time Taken:

Body weight must not be empty

OK


TrackID	ActivityName	TotalCalBurn	TrackStatus	UserID	ActivityID	TrackDate	Goal
Track-001	Jumping Ropes	1080	Complete	Customer-001	Activity-003	Wednesday, July...	1000
Track-002	Walking	3600	Complete	Customer-001	Activity-001	Thursday, July 25...	3000
Track-003	Jumping Ropes		Incomplete	Customer-001	Activity-003	Thursday, July 18...	700
Track-004	Swimming	720	Complete	Customer-001	Activity-002	Wednesday, July...	300
Track-005	Swimming		Incomplete	Customer-001	Activity-002	Wednesday, July...	800
Track-006	Pushups	3200	Complete	Customer-001	Activity-006	Thursday, July 25...	2000
Track-007	Swimming		Incomplete	Customer-001	Activity-002	Monday, July 15, ...	3000
Track-010	Pushups		Incomplete	Customer-001	Activity-006	Thursday, July 18...	4000
Track-011	Swimming		Incomplete	Customer-001	Activity-002	Tuesday, July 16,...	300




## Admin Login

Test Case	Fig 1.9
Test Objective	Testing if username is empty
Procedure	Go to Admin Login form and submit without username
Test Data	Testing null username value
Expected Result	User should get a popup saying username can't be empty
Actual Result	As shown in Fig 1.9

Before Testing (Fig1.9)

AdminLogin

### Admin Login

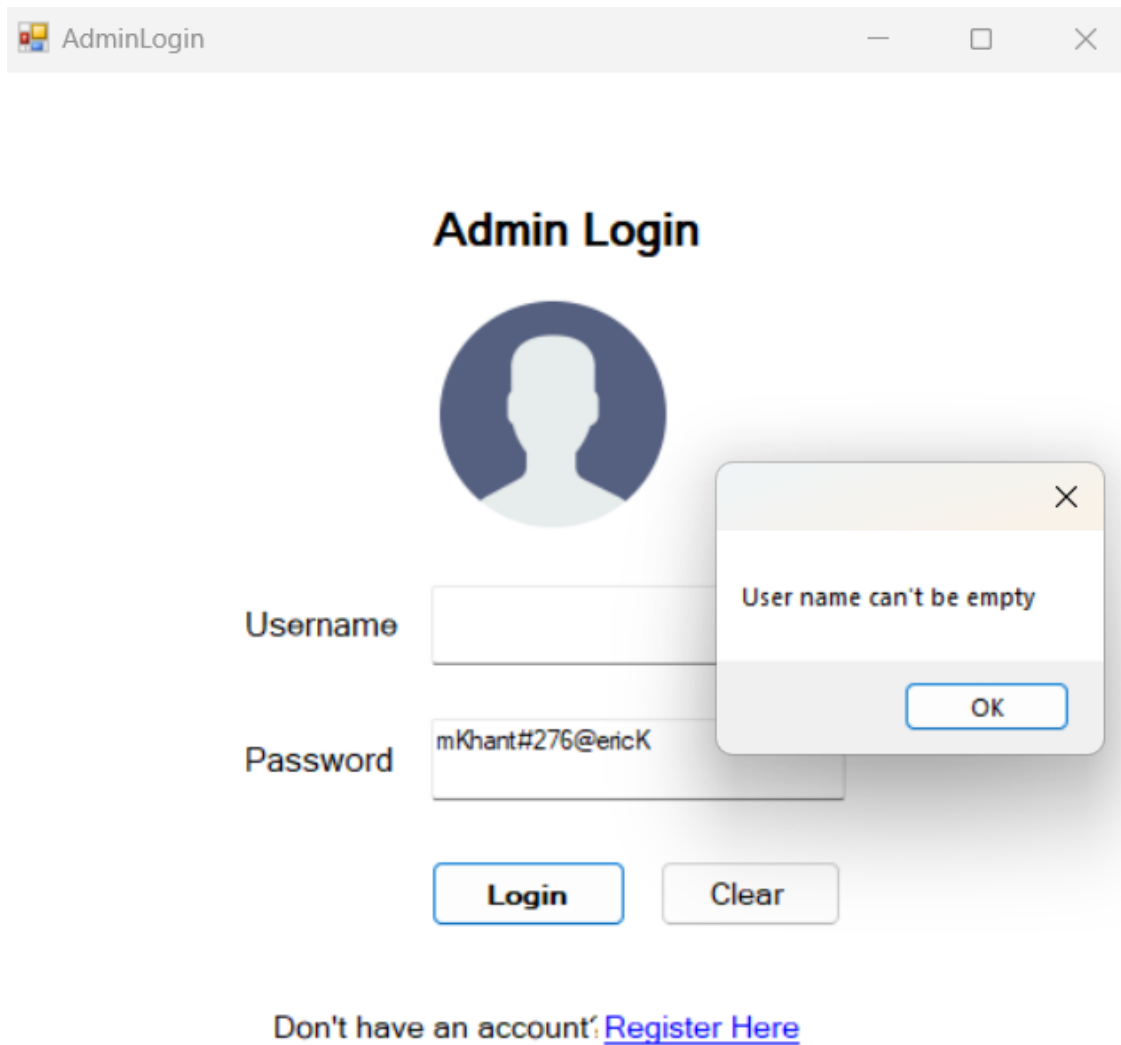


Username

Password

Don't have an account? [Register Here](#)

After Testing (Fig 1.9)



The image shows a web browser window titled "AdminLogin". The page has a heading "Admin Login" and a circular profile icon placeholder. Below the icon are two input fields: "Username" and "Password". The "Username" field is empty, and the "Password" field contains the text "mKhant#276@erick". Below the password field is a "Login" button and a "Clear" button. A modal dialog box is open over the "Username" field, displaying the message "User name can't be empty" and an "OK" button. At the bottom of the page, there is a link that says "Don't have an account? [Register Here](#)".

**Admin Login**

Username

Password

Don't have an account? [Register Here](#)

User name can't be empty

OK

## White Box Testing

### White Box Testing Plan

No	Form	Function	Date	Time
1	AdminRegister	ClearAdminForm() AdminAutoID()	July 01	09:00 – 12:00
2	Customer Register	ClearCustomerForm() CustomerAutoID ()	July 02	09:00 – 12:00
3.	CustomerMainMenu	NavigateToActivitiesList ()	July 03	09:00 – 12:00
4.	SetGoal	SetAutoID() SelectActivity()	July 04	09:00 – 12:00
5.	Activity	ActivityAutoID()	July 05	09:00 – 12:00

## White Box Testing

### Program Code

#### ClearAdminForm ()

```
public void ClearAdminForm()
{
    adminNameTextBox.Text = "";
    adminEmailTextBox.Text = "";
    adminPasswordTextBox.Text = "";
    adminUserNameTextBox.Text = "";
}
```

AdminAutoID()

```
public void AdminAutoID()
{
    DataTable trackdta = new DataTable();
    trackdta = adminTb.GetData();

    if (trackdta.Rows.Count == 0)
    {
        adminIDTextBox.Text = "Admin-001";
    }
    else
    {
        int size = trackdta.Rows.Count - 1;
        string oldid = trackdta.Rows[size][0].ToString();

        int newid = Convert.ToInt16(oldid.Substring(6, 3));

        if (newid >= 1 && newid < 9)
        {
            adminIDTextBox.Text = "Admin-00" + (newid + 1);
        }
        else if (newid >= 9 && newid < 99)
        {
            adminIDTextBox.Text = "Admin-0" + (newid + 1);
        }
        else if (newid >= 99 && newid < 999)
        {
            adminIDTextBox.Text = "Admin-" + (newid + 1);
        }
    }
}
```

CustomerAutoID()

```
public void CustomerAutoID() {  
    cusdt = CusTA.GetData();  
    customerIDTextBox.Text = "ADMIN-0012";  
    // To be continued here  
}
```

ClearCustomerForm()

```
public void ClearAdminForm()  
{  
    customerNameTextBox.Text = "";  
    customerEmailTextBox.Text = "";  
    customerPasswordTextBox.Text = "";  
    customerUserNameTextBox.Text = "";  
    customerEmailTextBox.Text = ""; NavigateToActivitiesList ()  
}
```

NavigateToActivitiesList()

```
private void NavigateToActivitiesList(){  
    ActivitiesList ActivitiesListForm = new ActivitiesList();  
    this.Hide();  
    ActivitiesListForm.ShowDialog();  
}  
}
```

SelectActivity()

```
public void SelectActivity() {  
    Activitydta = ActivityTA.GetData();  
    if (Activitydta.Rows.Count > 0)  
    {  
        DataRow dr = Activitydta.NewRow();  
        comboBoxActivity.DataSource = Activitydta;  
        comboBoxActivity.DisplayMember = "ActivityName";  
        comboBoxActivity.ValueMember = "ActivityID";  
    }  
}
```

## ActivityAutoID()

```
public void ActivityAutoID()
{
    DataTable trackdta = new DataTable();
    trackdta = ActTA.GetData();

    if (trackdta.Rows.Count == 0)
    {
        textBoxActivityID.Text = "Activity-001";
    }
    else
    {
        int size = trackdta.Rows.Count - 1;
        string oldid = trackdta.Rows[size][0].ToString();

        int newid = Convert.ToInt16(oldid.Substring(9, 3));

        if (newid >= 1 && newid < 9)
        {
            textBoxActivityID.Text = "Activity-00" + (newid + 1);
        }
        else if (newid >= 9 && newid < 99)
        {
            textBoxActivityID.Text = "Activity-0" + (newid + 1);
        }
        else if (newid >= 99 && newid < 999)
        {
            textBoxActivityID.Text = "Activity-" + (newid + 1);
        }
    }
}
```

## Exception Handling

Exception handling is crucial for managing unexpected conditions during execution. Various exception scenarios were simulated to ensure the application could handle errors gracefully. Exception Testing

### SetGoal Form

#### Before handling

The screenshot displays the Visual Studio IDE during a debugging session. The main code window shows a C# file with a `FormatException` being thrown at line 12, column 10. The exception message is "Input string was not in a correct format." The `FormatException` is highlighted in the code editor.

The Solution Explorer on the right shows the project structure for `FitnessTrackerApp`. The `Activity.cs` file is selected.

The Exception Details window on the right provides information about the `FormatException`. It includes a **Troubleshooting tips** section with links to help documentation and a **Actions** section with options to view details, copy to clipboard, and open settings.

The Immediate Window at the bottom shows the current state of the application. It lists the `comboBoxActivity` variable, which is of type `System.Windows.Forms.ComboBox` and has a value of `Items.Count: 3`.

```
private void btnSetGoal_Click(object sender, EventArgs e)
{
    Text);
    ToString();
    '].ToString();
    i"].ToString();

    ; comboBoxActivity.Text, "Incomplete", CustomerLogin.CID, textBoxActivityID.Text, goalDate.Text, Convert.ToInt32(textBoxGoal.Text));
    y.");
}

private void btnCancel_Click(object sender, EventArgs e)
{
}
```

## After Handling

The screenshot shows a Windows application window titled "SetGoal". The window has a menu bar with "Activity List", "Account", and "Search". The main content area displays a welcome message "Welcome Min Khant Kyaw" and the instruction "Let's define and achieve your goal". Below this, there is a section titled "Choose activity and define your goal". This section contains a "Choose Activity" dropdown menu with "Walking" selected, two input fields for "Activity ID" and "Track ID" (with "Track-013" entered), a "Your Goal" input field, and a "Deadline" dropdown menu showing "Tuesday, July 16, 2024". A "Save" button is located at the bottom right of the form. An error dialog box is overlaid on the form, displaying the following message: "Error :System.FormatException: Input string was not in a correct format. at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal) at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info) at System.Convert.ToInt32(String value) at FitnessTrackerApp.SetGoal.buttonSave\_Click(Object sender, EventArgs e) in c:\MinKhantKyaw\_P00207262\_DDOOCP\207262\_MIN KHANT KYAW\_DDOOCP\_GA\Program\FitnessTrackerApp\FitnessTrackerApp\SetGoal.cs:line 120". The dialog box has an "OK" button.

## Program Code

```
private void buttonSave_Click(object sender, EventArgs e)
{
    try {
        int recordedCount = TrackTA.RecordTrack(textBoxTrackID.Text, comboBoxActivity.Text,
        "Incomplete", CustomerLogin.CID, textBoxActivityID.Text, goalDate.Text, Convert.ToInt32(textBoxGoal.Text));
        if (recordedCount > 0)
        {
            MessageBox.Show("Your goal has been defined successfully.");
            Track t = new Track();
            this.Hide();
            t.ShowDialog();
        }
    }
    catch (Exception ex) {
        MessageBox.Show("Error :" + ex);
    }
}
```



## **Test Results**

The program underwent rigorous testing using multiple techniques, including manual testing, black box testing, white box testing, and unit testing. Each test was meticulously planned and executed to verify that the software functions as intended. The results indicated that all critical functionalities passed the tests, with minor bugs identified and subsequently fixed.

## **Discussion on Data Selection and Execution**

### **Data Selection**

The selection of test data was driven by the need to cover a comprehensive range of scenarios. Valid inputs were chosen based on the application's specifications, ensuring that they met all defined criteria. Invalid inputs were crafted to challenge the application's robustness, including:

- Empty Fields: To test how the application handles missing required information.
- Invalid Characters: Inputs that include special characters or unexpected formats to validate input sanitization.
- Boundary Conditions: Inputs at the edge of acceptable ranges, such as maximum and minimum lengths for usernames and passwords.

Exceptional cases were also anticipated, such as SQL injection attempts or excessively long input strings, to ensure the application could handle these scenarios without crashing.

### **Execution Strategy**

The execution of tests was systematic and methodical. Each test case was executed in a controlled environment to ensure consistency. The execution followed a predefined schedule, allowing for organized tracking of results and issues. Test cases were run multiple times to ensure reliability, and any anomalies were documented in detail. The results from black box testing provided insights into the user experience, while white box testing offered a deeper understanding of the internal logic and potential vulnerabilities in the code. This dual approach ensured that both functional and non-functional requirements were met.

### **Iterative Testing Process**

The testing process was iterative, allowing for continuous improvement. After each round of testing, results were analyzed, and necessary adjustments were made to both the test cases and the application itself. This feedback loop was crucial in refining the application to meet user needs effectively.

## **Documentation and Reporting**

All test results were meticulously documented, including the specifics of each test case, the data used, and the outcomes. This documentation serves as a valuable resource for future testing and development efforts, providing a historical record of testing activities and outcomes.

## **Conclusion**

In conclusion, the testing plan and execution provided a thorough assessment of the software's functionality and robustness. By carefully selecting test data and following a structured execution strategy, the testing process ensured that the application meets user expectations and performs reliably in production. The combination of black box and white box testing methodologies allowed for a comprehensive evaluation, ultimately contributing to the software's quality and reliability. This revised version meets the requirement for both the testing plan report and the discussion on data selection and execution, ensuring clarity and thoroughness in presenting the testing strategy and its justification.

## Task 3

### Class Diagram Introduction

A class diagram is a simple visual tool used to design systems. It shows the system's structure, highlighting its classes, their attributes (properties), methods (actions), and relationships between classes. These diagrams help in understanding how different parts of the system are connected and work together.

#### Why Class Diagrams are Important:

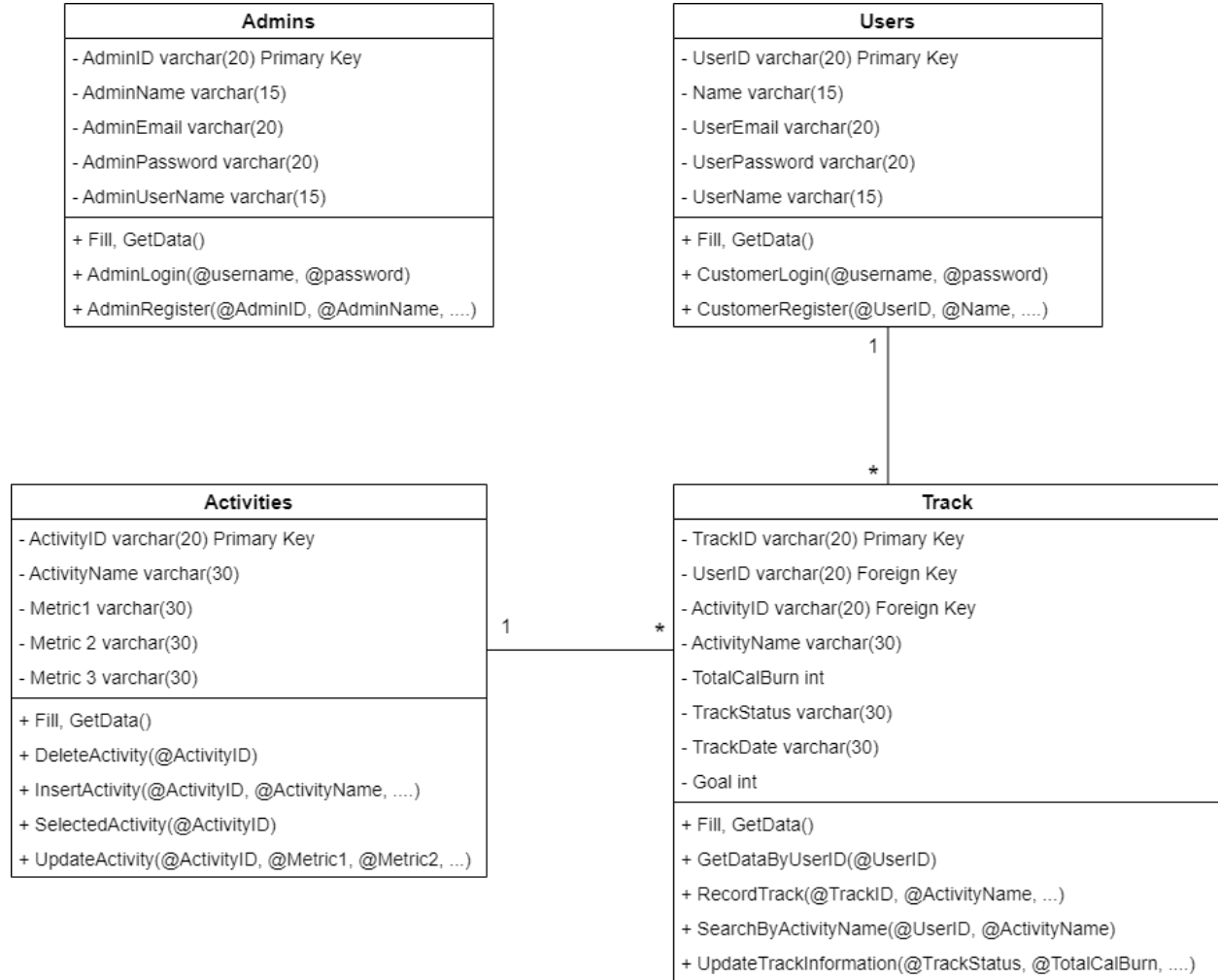
- **Visualize the System:** They offer a clear view of the system's organization.
- **Show Relationships:** They illustrate how classes interact.
- **Guide Development:** They provide a blueprint for developers.

#### Key Components of Class Diagrams:

- **Class:** Represented as a rectangle with three sections:
  - **Top section:** Name of the class (e.g., Cat).
  - **Middle section:** Attributes (e.g., name, age, gender).
  - **Bottom section:** Methods (e.g., getAge, getName).
- **Attributes:** Characteristics or data fields of a class, shown in the middle section (e.g., a Person class may have attributes like name, age, gender).
- **Methods:** Actions that objects of the class can perform, listed in the bottom section (e.g., a Person class may have methods like getAge, walk, eat).
- **Relationships:** Indicate how classes are connected, such as one-to-many or many-to-many relationships. These connections often use a primary key from one class in another, like linking a Person class to a Pet class using the Pet class's primary key.

In summary, class diagrams are crucial for designing systems, providing clarity on how various components interact.

### Class Diagram of Fitness Tracker



## Class Diagrams Explanation

The class diagram for the fitness tracker software consists of four main entities: Admins, Users, Activities, and Track. Users and Activities classes have one to many relationships with Track. Here is a detailed explanation of the design:

### Admins

The Admins entity represents the administrators of the fitness tracker software. It has the following attributes:

**AdminID:** A unique identifier for the admin, used as the primary key.

**AdminName:** The name of the admin.

**AdminEmail:** The email address of the admin.

**AdminPassword:** The password used by the admin to log in.

**AdminUserName:** The username used by the admin to log in.

The Admins entity has the following methods:

**Fill():** A method to fill the Admins table with data.

**GetData():** A method to retrieve data from the Admins table.

**AdminLogin():** A method to authenticate an admin user by checking the provided username and password.

**AdminRegister():** A method to register a new admin user by creating a new record in the Admins table.

### Users

The Users entity represents the customers or users of the fitness tracker software. It has the following attributes:

**UserID:** A unique identifier for the user, used as the primary key.

**Name:** The name of the user.

**UserEmail:** The email address of the user.

**UserPassword:** The password used by the user to log in.

**UserName:** The username used by the user to log in.

The Users entity has the following methods:

**Fill():** A method to fill the Users table with data.

**GetData():** A method to retrieve data from the Users table.

**CustomerLogin():** A method to authenticate a customer user by checking the provided username and password.

**CustomerRegister():** A method to register a new customer user by creating a new record in the Users table.

## Activities

The Activities entity represents the different types of activities that users can track. It has the following attributes:

**ActivityID:** A unique identifier for the activity, used as the primary key.

**ActivityName:** The name of the activity.

**Metric1:** The first metric associated with the activity.

**Metric2:** The second metric associated with the activity.

**Metric3:** The third metric associated with the activity.

The Activities entity has the following methods:

**Fill():** A method to fill the Activities table with data.

**GetData():** A method to retrieve data from the Activities table.

**DeleteActivity():** A method to delete an activity from the Activities table.

**InsertActivity():** A method to insert a new activity into the Activities table.

**SelectedActivity():** A method to retrieve the details of a selected activity.

**UpdateActivity():** A method to update details of an existing activity in Activities table.

## Track

The Track entity represents the user's activity tracking information. It has the following attributes:

**TrackID:** A unique identifier for the tracking record, used as the primary key.

**UserID:** The ID of the user who performed the activity, used as a foreign key to the Users table.

**ActivityID:** The ID of the activity that was performed, used as a foreign key to the Activities table.

**ActivityName:** The name of the activity that was performed.

**TotalCalBurn:** The total number of calories burned during the activity.

**TrackStatus:** The status of the tracking record (e.g., completed, in progress).

**TrackDate:** The date when the activity was performed.

**Goal:** The user's goal for the activity.

The Track entity has the following methods:

**Fill():** A method to fill the Track table with data.

**GetData():** A method to retrieve data from the Track table.

**GetDataByUserID():** A method to retrieve tracking records for a specific user.

**RecordTrack():** A method to create a new tracking record in the Track table.

**SearchByActivityName():** A method to search for tracking records by the activity name.

**UpdateTrackInformation():** A method to update the information for an existing tracking record in the Track table.

## Justification for Class Design

The class diagram for the fitness tracker software follows sound object-oriented design principles and provides a clear separation of concerns between the different entities involved. Let's look at the reason behind the design of each class:

### **Admins Class**

The Admins class is responsible for managing the administrative users of the system. It encapsulates the necessary attributes (AdminID, AdminName, AdminEmail, AdminPassword, AdminUserName) to uniquely identify and authenticate an admin user. The class also provides methods to fill the Admins table with data, retrieve data, authenticate admin login, and register new admin users. This design ensures that admin functionality is isolated and can be easily maintained and extended as needed.

### **Users Class**

The Users class represents the customer or user entity in the system. It has attributes (UserID, Name, UserEmail, UserPassword, UserName) that allow for the unique identification and authentication of users. The class provides methods to fill the Users table, retrieve user data, authenticate customer login, and register new customer users. This design ensures that user-specific functionality is encapsulated and can be easily managed separately from the admin functionality.

### **Activities Class**

The Activities class models the different types of activities that users can track. It has attributes (ActivityID, ActivityName, Metric1, Metric2, Metric3) that describe the activity and its associated metrics. The class provides methods to fill the Activities table, retrieve activity data, delete activities, insert new activities, retrieve details of a selected activity, and update existing activities. This design allows for the easy management and extension of the available activities in the system.

### **Track Class**

The Track class represents the actual tracking of user activities. It has attributes (TrackID, UserID, ActivityID, ActivityName, TotalCalBurn, TrackStatus, TrackDate, Goal) that capture the details of a specific activity tracking instance. The class provides methods to fill the Track table, retrieve tracking data, retrieve data by user ID, record new tracking instances, search by activity name, and update existing tracking records. This design ensures that the tracking functionality is separate from the user and activity management, allowing for a more modular and maintainable system.

### **Relationships and Associations**

The class diagram also establishes relationships between the entities:



- **Users and Track:** A one-to-many relationship, where a user can have multiple tracking records, but each tracking record is associated with a single user. This relationship is established through the UserID foreign key in the Track class.
- **Activities and Track:** A one-to-many relationship, where an activity can be associated with multiple tracking records, but each tracking record is associated with a single activity. This relationship is established through the ActivityID foreign key in the Track class.

These relationships ensure data integrity and allow for the retrieval of related data across the different entities.

## Conclusion

The class diagram for the fitness tracker software demonstrates a well-designed and organized system that follows object-oriented principles. The separation of concerns between the different entities, the encapsulation of functionality within each class, and the establishment of relationships between classes contribute to a maintainable and extensible system. The design choices made in this class diagram provide a solid foundation for implementing the fitness tracker software.

## References

1. Team, P. (2024) *Black box vs white box testing*, *PractiTest*. Available at: <https://www.practitest.com/resource-center/article/black-box-vs-white-box-testing/> (Accessed: 01 July 2024).

2. GeeksforGeeks (2024a) *Class diagram: Unified modeling language (UML)*, *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/> (Accessed: 01 July 2024).

3. Dev, R. (2012) *Black-box testing*, *Black-Box Testing - an overview | ScienceDirect Topics*. Available at: <https://www.sciencedirect.com/topics/computer-science/black-box-testing> (Accessed: 01 July 2024).

4. Ostrand, T. (2002) *Black-box testing - ostrand - major reference works*, *Wiley Online Library*. Available at: <https://onlinelibrary.wiley.com/doi/abs/10.1002/0471028959.sof022> (Accessed: 01 July 2024).

## Candidate checklist

Please use the following checklist to ensure that your work is ready for submission.

Have you read the NCC Education documents 'What is Academic Misconduct? Guidance for Candidates' and 'Avoiding Plagiarism and Collusion: Guidance for



Candidates' and ensured that you have acknowledge all the sources that you have used in your work?

Have you completed the 'Statement and Confirmation of Own Work' form and attached it to your assignment? You must do this.



Have you ensured that your work has not gone over or under the recommended word count by more than 10%?



Have you ensured that your work does not contain viruses and can be run directly?

