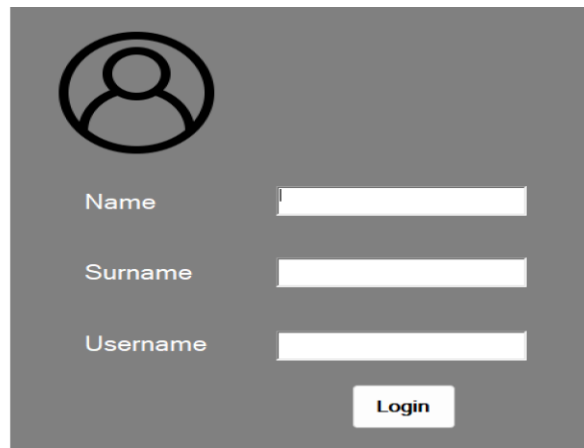


## Project Process and Stages

After determining the purpose of the project, I analyzed the user requirements and the operations to be done. I determined basic functions such as creating an XML file by entering the title and content in a digital diary application, displaying these files in the List and editing or deleting them when necessary. First, I designed the login screen and completed the operations to be done.

A login screen with a dark gray background. At the top left is a circular icon of a person. Below it are three white input fields labeled "Name", "Surname", and "Username". At the bottom right is a white button labeled "Login".

*Login Screen*

While designing the user interface of the project, I aimed to create a user-friendly and simple interface. I designed a structure that allows the display of added files in the list with a date picker (DateTimePicker), title and content fields, buttons for adding, updating and deleting operations. During the software development process, I worked on the management of XML files. In the file adding process, when the user clicks the "Add to Daily" button after entering the title and content, an XML file is created using the title as the file name. The data entered by the user is written to the content of this file and the file name is added to the list. In the file reading process, when the user selects a file from the list, the content of this file is read in XML format and displayed in the user interface. In the file updating process, the user can update the file by pressing the "Update" button after making a change in the content section. This process is done by overwriting the file with new data. In the file deletion process, when the user selects a file from the list and presses the "Delete" button, the relevant XML file is deleted from the file system and removed from the list.

Enter Title

Add To Daily

Update

Hello Gökay

Close App

1 Eylül 2024 Pazar

01-09-2024 - Birthday

Enter Content

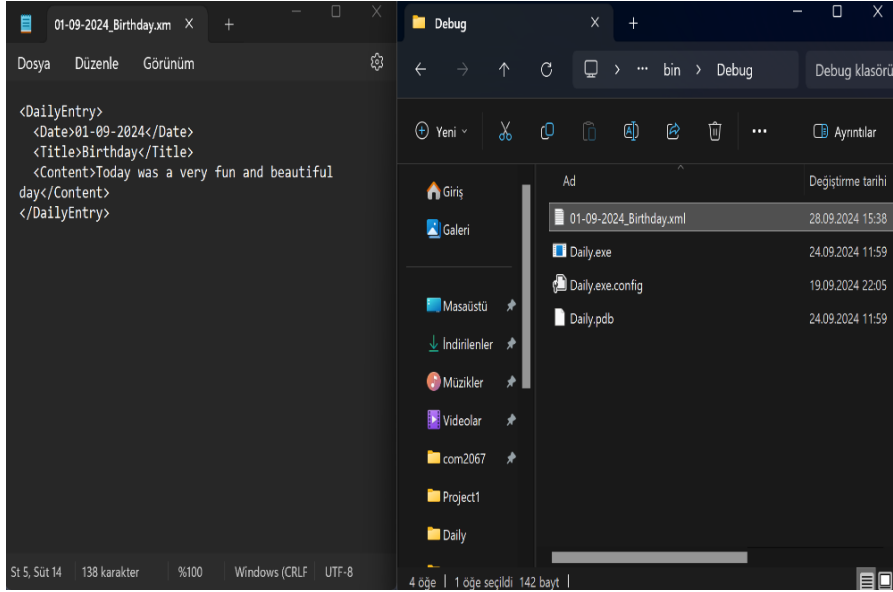
Birthday

Today was a very fun and beautiful day

Gökay Çetinakdoğan gokay12

Delete

Main Screen



The resulting XML file and its content

## Appendix

```
namespace Daily
{
    4 references
    public partial class Form1 : Form
    {
        public UserDefinition _user;

        1 reference
        public Form1(UserDefinition user)
        {
            InitializeComponent();
            _user = user;

            this.Size = new Size(950, 600);
            this.Load += new System.EventHandler(this.Form1_Load);
            this.ListBox1.SelectedIndexChanged += new System.EventHandler(this.ListBox1_SelectedIndexChanged);
        }

        2 references
        private void Form1_Load(object sender, EventArgs e)
        {
            label2.Text = "Hello " + _user.FirstName;
            label3.Text = _user.FirstName + " " + _user.LastName + " " + _user.Username;

            string[] files = Directory.GetFiles(Application.StartupPath, "*.xml");

            foreach (var file in files)
            {
                XmlDocument xml = new XmlDocument();
                xml.Load(file);

                XmlNode date = xml.SelectSingleNode("//Date");
                XmlNode title = xml.SelectSingleNode("//Title");

                if (date != null && title != null)
                {
                    string entry = $"{date.InnerText} - {title.InnerText}";
                    if (!ListBox1.Items.Contains(entry))
                    {
                        ListBox1.Items.Add(entry);
                    }
                }
            }
        }
    }
}
```

### *Entry Codes*

This code in figure 4 displays the user's information when a form is loaded and reads all the XML files in the application folder and adds the date and title information to ListBox1. This process is used to present a listing and selection interface to the user

```

1 reference
private void AddButton_Click(object sender, EventArgs e)
{
    string title = richTextBox1.Text;

    if (!string.IsNullOrEmpty(title))
    {
        string content = richTextBox2.Text;

        DateTime selectedDate = dateTimePicker1.Value;

        DailyDefinition daily = new DailyDefinition
        {
            Date = selectedDate,
            Title = title,
            Content = content
        };

        string fileName = $"{selectedDate:dd-MM-yyyy}_{title}.xml";
        string filePath = Path.Combine(Application.StartupPath, fileName);

        XmlDocument x = new XmlDocument();

        XmlElement root = x.CreateElement("DailyEntry");
        XmlElement dateElement = x.CreateElement("Date");
        dateElement.InnerText = daily.Date.ToString("dd-MM-yyyy");
        XmlElement titleElement = x.CreateElement("Title");
        titleElement.InnerText = daily.Title;
        XmlElement contentElement = x.CreateElement("Content");
        contentElement.InnerText = daily.Content;

        root.AppendChild(dateElement);
        root.AppendChild(titleElement);
        root.AppendChild(contentElement);
        x.AppendChild(root);

        x.Save(filePath);
    }
}

```

#### *Button Codes*

This code in figure 5 stores a log record in XML format when the Add button is used and provides access to the user via the list with date and title information. The process steps are as follows: Title, content and date information are received from the user. The information is saved in the XML file. Log information is displayed in ListBox1. Input fields are cleared.

```

private void UpdateDaily_Click(object sender, EventArgs e)
{
    if (ListBox1.SelectedItem != null)
    {
        string selectedItem = ListBox1.SelectedItem.ToString();
        string[] p = selectedItem.Split(new[] { " - " }, StringSplitOptions.None);

        if (p.Length == 2)
        {
            string selectedDate = p[0];
            string selectedTitle = p[1];

            string fileName = $"{selectedDate}_{selectedTitle}.xml";
            string filePath = Path.Combine(Application.StartupPath, fileName);

            if (File.Exists(filePath))
            {
                XmlDocument xml = new XmlDocument();
                xml.Load(filePath);

                string newContent = richTextBox2.Text;

                XmlNode content = xml.SelectSingleNode("//Content");

                if (content != null)
                {
                    content.InnerText = newContent;
                }
                else
                {
                    XmlElement content0 = xml.CreateElement("Content");
                    content0.InnerText = newContent;
                    xml.DocumentElement.AppendChild(content0);
                }

                xml.Save(filePath);
                MessageBox.Show("XML file has been updated");
            }
            else
            {
                MessageBox.Show("File not found.");
            }
        }
    }
}

```

#### Button Codes

This section is designed to update the XML file in the selected index from the ListBox. It is checked whether the user has selected an item in the ListBox; if not, the process is stopped. The selected item is parsed according to the " - " character to obtain date (p[0]) and title (p[1]) information. Using this information, an XML file name is created and the existence of this file in the application's initial directory is checked. If the file exists, the file is loaded with the XmlDocument object and the new content entered by the user into richTextBox2 is retrieved. The <Content> tag in the XML is checked: if so, the content of this tag is updated with the new value, if not, a new <Content> tag is created and added. The updated XML file is saved and a successful process message is displayed to the user. If the file does not exist, the user is given a "File not found" message and the process is terminated.

```

private void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (ListBox1.SelectedItem != null)
    {
        string selectedItem = ListBox1.SelectedItem.ToString();

        string[] p = selectedItem.Split(new[] { " - " }, StringSplitOptions.None);

        if (p.Length == 2)
        {
            TextBox.Text = p[1];

            string selectedDate = p[0];
            string selectedTitle = p[1];

            string fileName = $"{selectedDate}_{selectedTitle}.xml";
            string filePath = Path.Combine(Application.StartupPath, fileName);

            if (File.Exists(filePath))
            {
                XmlDocument x = new XmlDocument();
                x.Load(filePath);

                XmlNode _content = x.SelectSingleNode("//Content");
                if (_content != null)
                {
                    listBox2.Items.Clear();
                    listBox2.Items.Add(_content.InnerText);
                }
            }
            else
            {
                MessageBox.Show("File not found");
            }
        }
    }
}

```

#### *List Codes*

The purpose of this code in figure 7 is to load the XML file and display the <Content> information on the screen when the user selects an item:

When the user makes a selection from the ListBox, the corresponding title and date information are retrieved. Using this information, the name of the XML file is generated, and its existence is checked. If the file exists, its content is read, the data within the <Content> tag is parsed, and this content is displayed in another ListBox (listBox2) in a list format for the user to view. However, if the file cannot be found, the system detects an error condition and displays a warning message to inform the user that the file does not exist.

```

using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml;

namespace Daily.Data
{
    4 references
    public class DailyDefinition
    {
        int _id;
        string _title;
        string _content;
        DateTime _date;
        UserDefinition _user;

        1 reference
        public DailyDefinition()
        {
        }

        0 references
        public DailyDefinition(int id, string title, string content, DateTime date, UserDefinition user)
        {
            Id = id;
            Title = title;
            Content = content;
            Date = date;
            User = user;
        }

        1 reference
        public int Id { get => _id; set => _id = value; }
        4 references
        public string Title { get => _title; set => _title = value; }
        3 references
        public string Content { get => _content; set => _content = value; }
        4 references
        public DateTime Date { get => _date; set => _date = value; }
        1 reference
        public UserDefinition User { get => _user; set => _user = value; }
    }
}

```

*Class Codes*

The code in Figure 8 defines a `DailyDefinition` class. This class holds the properties of a daily record and associated user information. It is used in the application's log-based data management. It is managed by applying encapsulation. This provides a controlled get and set mechanism by restricting direct access to fields (`_id`, `_title`, etc.).