**Code:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;

using System.Windows.Forms;

namespace UsernameValidationApps

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void validateButton\_Click(object sender, EventArgs e)

{

// Clear previous output

textBox2.Clear();

// Get initial usernames

List<string> usernames = textBox1.Text.Split(',').Select(u => u.Trim()).ToList();

ProcessUsernames(usernames, out var results);

textBox2.Text += results;

// Retry Invalid Names

RetryInvalidUsernames(usernames, out var retryResults);

textBox2.Text += retryResults;

// Save to File

SaveResultsToFile(textBox2.Text);

}

private void ProcessUsernames(List<string> usernames, out string results)

{

results = string.Empty;

List<string> validUsernames = new List<string>();

List<string> invalidUsernames = new List<string>();

foreach (var username in usernames)

{

var (isValid, reason, details) = ValidateUsername(username);

if (isValid)

{

validUsernames.Add(username);

string password = GeneratePassword();

string strength = EvaluatePasswordStrength(password);

results += $"{username} - Valid\n" +

$" Letters: {details["Uppercase"] + details["Lowercase"]} (Uppercase: {details["Uppercase"]}, Lowercase: {details["Lowercase"]}), " +

$"Digits: {details["Digits"]}, Underscores: {details["Underscores"]}\n" +

$" Generated Password: {password} (Strength: {strength})\n\n";

}

else

{

results += $"{username} - Invalid ({reason})\n\n";

invalidUsernames.Add(username);

}

}

results += "Summary:\n";

results += $"- Total Usernames: {usernames.Count}\n";

results += $"- Valid Usernames: {validUsernames.Count}\n";

results += $"- Invalid Usernames: {invalidUsernames.Count}\n\n";

if (invalidUsernames.Count > 0)

{

results += $"Invalid Usernames: {string.Join(", ", invalidUsernames)}\n";

}

}

private (bool isValid, string reason, Dictionary<string, int> details) ValidateUsername(string username)

{

var details = new Dictionary<string, int> { { "Uppercase", 0 }, { "Lowercase", 0 }, { "Digits", 0 }, { "Underscores", 0 } };

if (username.Length < 5 || username.Length > 15)

return (false, "Username length must be between 5 and 15.", details);

if (!Regex.IsMatch(username, @"^[a-zA-Z]"))

return (false, "Username must start with a letter.", details);

if (!Regex.IsMatch(username, @"^[a-zA-Z0-9\_]+$"))

return (false, "Username can only contain letters, numbers, and underscores.", details);

foreach (char c in username)

{

if (char.IsUpper(c)) details["Uppercase"]++;

else if (char.IsLower(c)) details["Lowercase"]++;

else if (char.IsDigit(c)) details["Digits"]++;

else if (c == '\_') details["Underscores"]++;

}

return (true, "Valid", details);

}

private string GeneratePassword()

{

const string upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

const string lower = "abcdefghijklmnopqrstuvwxyz";

const string digits = "0123456789";

const string special = "!@#$%^&\*";

Random rnd = new Random();

string password = string.Empty;

password += new string(Enumerable.Range(0, 2).Select(x => upper[rnd.Next(upper.Length)]).ToArray());

password += new string(Enumerable.Range(0, 2).Select(x => lower[rnd.Next(lower.Length)]).ToArray());

password += new string(Enumerable.Range(0, 2).Select(x => digits[rnd.Next(digits.Length)]).ToArray());

password += new string(Enumerable.Range(0, 2).Select(x => special[rnd.Next(special.Length)]).ToArray());

string allChars = upper + lower + digits + special;

password += new string(Enumerable.Range(0, 4).Select(x => allChars[rnd.Next(allChars.Length)]).ToArray());

return new string(password.OrderBy(x => rnd.Next()).ToArray());

}

private string EvaluatePasswordStrength(string password)

{

int score = 0;

if (password.Length >= 12) score++;

if (Regex.IsMatch(password, @"[A-Z]")) score++;

if (Regex.IsMatch(password, @"[a-z]")) score++;

if (Regex.IsMatch(password, @"\d")) score++;

if (Regex.IsMatch(password, @"[!@#$%^&\*]")) score++;

if (score >= 4)

return "Strong";

else if (score == 3)

return "Medium";

else

return "Weak";

}

private void RetryInvalidUsernames(List<string> usernames, out string retryResults)

{

retryResults = string.Empty;

// Extract invalid usernames from the list and ask to retry

var invalidUsernames = usernames.Where(u => !ValidateUsername(u).isValid).ToList();

if (invalidUsernames.Count > 0)

{

DialogResult result = MessageBox.Show("Do you want to retry invalid usernames?", "Retry Option", MessageBoxButtons.YesNo);

if (result == DialogResult.Yes)

{

ProcessUsernames(invalidUsernames, out retryResults);

}

}

}

private void SaveResultsToFile(string content)

{

SaveFileDialog saveFileDialog = new SaveFileDialog

{

FileName = "UserDetails.txt",

Filter = "Text files (\*.txt)|\*.txt"

};

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

File.WriteAllText(saveFileDialog.FileName, content);

MessageBox.Show("Results saved successfully.");

}

}

}

}

Output:





