**LAB terminal**

**Submitted By:**

Alveena Ayesha

FA21-BCS-014

**Course Instructor:**

Mr. Syed Bilal Bukhari

**Course:**

CC

**Date:**

3ND January, 2025

**Q6**

using System;

using System.Text.RegularExpressions;

using System.IO;

using System.Linq;

using System.Security.Cryptography;

class Program

{

static void Main()

{

Console.Write("Enter usernames (separated by commas): ");

string[] usernames = Console.ReadLine()?.Split(',');

int totalUsernames = 0;

int validUsernames = 0;

int invalidUsernames = 0;

string filePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.UserProfile), "Downloads", "UserDetails.txt");

using (StreamWriter writer = new StreamWriter(filePath))

{

writer.WriteLine("Validation Results:");

foreach (string username in usernames)

{

string trimmedUsername = username.Trim();

totalUsernames++;

if (IsValidUsername(trimmedUsername))

{

validUsernames++;

var details = GetUsernameDetails(trimmedUsername);

string password = GenerateSecurePassword();

string strength = CheckPasswordStrength(password);

writer.WriteLine($"{totalUsernames}. {trimmedUsername} - Valid");

writer.WriteLine($" Letters: {details.Letters} (Uppercase: {details.Uppercase}, Lowercase: {details.Lowercase}), Digits: {details.Digits}, Underscores: {details.Underscores}");

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

}

else

{

invalidUsernames++;

string reason = GetInvalidUsernameReason(trimmedUsername);

writer.WriteLine($"{totalUsernames}. {trimmedUsername} - Invalid ({reason})\n");

}

}

writer.WriteLine("\nSummary:");

writer.WriteLine($"- Total Usernames: {totalUsernames}");

writer.WriteLine($"- Valid Usernames: {validUsernames}");

writer.WriteLine($"- Invalid Usernames: {invalidUsernames}\n");

}

Console.WriteLine("Processing complete. Results saved to Downloads/UserDetails.txt");

}

static bool IsValidUsername(string username)

{

return Regex.IsMatch(username, "^[a-zA-Z][a-zA-Z0-9\_]{4,14}$");

}

static string GetInvalidUsernameReason(string username)

{

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

return "Username length must be between 5 and 15 characters";

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

return "Username must start with a letter";

if (Regex.IsMatch(username, "[^a-zA-Z0-9\_]") || username.Contains(" "))

return "Username contains invalid characters";

return "Unknown error";

}

static (int Letters, int Uppercase, int Lowercase, int Digits, int Underscores) GetUsernameDetails(string username)

{

int letters = username.Count(char.IsLetter);

int uppercase = username.Count(char.IsUpper);

int lowercase = username.Count(char.IsLower);

int digits = username.Count(char.IsDigit);

int underscores = username.Count(c => c == '\_');

return (letters, uppercase, lowercase, digits, underscores);

}

static string GenerateSecurePassword()

{

const string upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

const string lower = "abcdefghijklmnopqrstuvwxyz";

const string digits = "0123456789";

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

Random random = new Random();

string password = new string(Enumerable.Repeat(upper, 2).Select(s => s[random.Next(s.Length)]).ToArray()) +

new string(Enumerable.Repeat(lower, 2).Select(s => s[random.Next(s.Length)]).ToArray()) +

new string(Enumerable.Repeat(digits, 2).Select(s => s[random.Next(s.Length)]).ToArray()) +

new string(Enumerable.Repeat(special, 2).Select(s => s[random.Next(s.Length)]).ToArray());

string allChars = upper + lower + digits + special;

password += new string(Enumerable.Repeat(allChars, 4).Select(s => s[random.Next(s.Length)]).ToArray());

return new string(password.OrderBy(c => random.Next()).ToArray());

}

static string CheckPasswordStrength(string password)

{

int lengthScore = password.Length >= 12 ? 1 : 0;

int varietyScore = (password.Any(char.IsUpper) ? 1 : 0) + (password.Any(char.IsLower) ? 1 : 0) + (password.Any(char.IsDigit) ? 1 : 0) + (password.Any(c => "!@#$%^&\*".Contains(c)) ? 1 : 0);

if (lengthScore == 1 && varietyScore >= 3)

return "Strong";

else if (lengthScore == 1 && varietyScore == 2)

return "Medium";

else

return "Weak";

}

}

