Name: Arslan Ahmed

Reg#: Fa21-bcs-026

Subject: CC Lab Terminal

Question 6: Write a C# program using Regular Expressions (Regex) to perform the following tasks……

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;

class Program

{

static void Main()

{

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

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

List<UserDetails> validUserDetails = new List<UserDetails>();

// Step 1: Input usernames from user

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

string input = Console.ReadLine();

usernames.AddRange(input.Split(',').Select(s => s.Trim()));

// Process each username

foreach (var username in usernames)

{

var validationResult = ValidateUsername(username);

if (validationResult.isValid)

{

var userDetails = new UserDetails

{

Username = username,

UppercaseCount = validationResult.uppercaseCount,

LowercaseCount = validationResult.lowercaseCount,

DigitCount = validationResult.digitCount,

UnderscoreCount = validationResult.underscoreCount

};

userDetails.Password = GeneratePassword();

userDetails.PasswordStrength = EvaluatePasswordStrength(userDetails.Password);

validUserDetails.Add(userDetails);

}

else

{

invalidUsernames.Add(username);

Console.WriteLine($"Username '{username}' is invalid: {validationResult.errorMessage}");

}

}

// Display validation results

DisplayValidationResults(validUserDetails);

// Display summary and handle retry for invalid usernames

Console.WriteLine($"Summary: Total Usernames: {usernames.Count}, Valid Usernames: {validUserDetails.Count}, Invalid Usernames: {invalidUsernames.Count}");

if (invalidUsernames.Any())

{

Console.Write("Do you want to retry invalid usernames? (y/n): ");

string retryOption = Console.ReadLine();

if (retryOption.ToLower() == "y")

{

Console.Write("Enter invalid usernames: ");

string retryInput = Console.ReadLine();

var retryUsernames = retryInput.Split(',').Select(s => s.Trim()).ToList();

// Retry validation for invalid usernames

foreach (var username in retryUsernames)

{

var validationResult = ValidateUsername(username);

if (validationResult.isValid)

{

var userDetails = new UserDetails

{

Username = username,

UppercaseCount = validationResult.uppercaseCount,

LowercaseCount = validationResult.lowercaseCount,

DigitCount = validationResult.digitCount,

UnderscoreCount = validationResult.underscoreCount

};

userDetails.Password = GeneratePassword();

userDetails.PasswordStrength = EvaluatePasswordStrength(userDetails.Password);

validUserDetails.Add(userDetails);

}

else

{

Console.WriteLine($"Username '{username}' is invalid: {validationResult.errorMessage}");

}

}

}

}

// Save results to file

SaveResultsToFile(validUserDetails, invalidUsernames, usernames.Count);

}

// Validate username using regular expressions

static (bool isValid, string errorMessage, int uppercaseCount, int lowercaseCount, int digitCount, int underscoreCount) ValidateUsername(string username)

{

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

{

return (false, "Username length must be between 5 and 15 characters", 0, 0, 0, 0);

}

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

{

return (false, "Username must start with a letter and can only contain letters, numbers, and underscores", 0, 0, 0, 0);

}

int uppercaseCount = username.Count(c => char.IsUpper(c));

int lowercaseCount = username.Count(c => char.IsLower(c));

int digitCount = username.Count(c => char.IsDigit(c));

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

return (true, "", uppercaseCount, lowercaseCount, digitCount, underscoreCount);

}

// Generate a secure password

static string GeneratePassword()

{

const string upperChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

const string lowerChars = "abcdefghijklmnopqrstuvwxyz";

const string digits = "0123456789";

const string specialChars = "!@#$%^&\*()\_+-=[]{}|;:,.<>?";

Random rand = new Random();

// Ensure at least 2 uppercase, 2 lowercase, 2 digits, 2 special characters

string password = "";

password += upperChars[rand.Next(upperChars.Length)];

password += upperChars[rand.Next(upperChars.Length)];

password += lowerChars[rand.Next(lowerChars.Length)];

password += lowerChars[rand.Next(lowerChars.Length)];

password += digits[rand.Next(digits.Length)];

password += digits[rand.Next(digits.Length)];

password += specialChars[rand.Next(specialChars.Length)];

password += specialChars[rand.Next(specialChars.Length)];

// Fill the remaining 4 characters randomly

string allChars = upperChars + lowerChars + digits + specialChars;

for (int i = 8; i < 12; i++)

{

password += allChars[rand.Next(allChars.Length)];

}

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

}

// Evaluate the strength of the generated password

static string EvaluatePasswordStrength(string password)

{

int strength = 0;

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

if (password.Any(c => char.IsUpper(c))) strength++;

if (password.Any(c => char.IsLower(c))) strength++;

if (password.Any(c => char.IsDigit(c))) strength++;

if (password.Any(c => "!@#$%^&\*()\_+-=[]{}|;:,.<>?".Contains(c))) strength++;

if (strength == 5) return "Strong";

if (strength >= 3) return "Medium";

return "Weak";

}

// Display validation results

static void DisplayValidationResults(List<UserDetails> validUserDetails)

{

foreach (var user in validUserDetails)

{

Console.WriteLine($"1. {user.Username} - Valid");

Console.WriteLine($" Letters: {user.UppercaseCount + user.LowercaseCount} (Uppercase: {user.UppercaseCount}, Lowercase: {user.LowercaseCount}), Digits: {user.DigitCount}, Underscores: {user.UnderscoreCount}");

Console.WriteLine($" Generated Password: {user.Password} (Strength: {user.PasswordStrength})");

}

}

// Save results to file

static void SaveResultsToFile(List<UserDetails> validUserDetails, List<string> invalidUsernames, int totalUsernames)

{

using (StreamWriter writer = new StreamWriter("UserDetails.txt"))

{

// Write validation results

int validCount = validUserDetails.Count;

int invalidCount = invalidUsernames.Count;

foreach (var user in validUserDetails)

{

writer.WriteLine($"1. {user.Username} - Valid");

writer.WriteLine($" Letters: {user.UppercaseCount + user.LowercaseCount} (Uppercase: {user.UppercaseCount}, Lowercase: {user.LowercaseCount}), Digits: {user.DigitCount}, Underscores: {user.UnderscoreCount}");

writer.WriteLine($" Generated Password: {user.Password} (Strength: {user.PasswordStrength})");

}

// Write summary

writer.WriteLine($"Summary:");

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

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

writer.WriteLine($"- Invalid Usernames: {invalidCount}");

// Write invalid usernames

if (invalidUsernames.Any())

{

writer.WriteLine($"Invalid Usernames:");

foreach (var username in invalidUsernames)

{

writer.WriteLine($"- {username}");

}

}

}

Console.WriteLine("Results saved to 'UserDetails.txt'.");

}

}

class UserDetails

{

public string Username { get; set; }

public int UppercaseCount { get; set; }

public int LowercaseCount { get; set; }

public int DigitCount { get; set; }

public int UnderscoreCount { get; set; }

public string Password { get; set; }

public string PasswordStrength { get; set; }

}