# System Requirements Specification Index

For

# C# .Net Core 3.1 Skills Evaluation

Version 4.0

**IIHT Pvt. Ltd.**
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

## 1. Business-Requirement:

### 1.1 USE CASE DESCRIPTION

#### Use Case 1

**Write a program to take input of strings from the user and count the number of uppercase alphabets, lowercase alphabets, digits and special characters. Store these counts in an array of size 4.**

**Description**

1. Take console input of string, in **Main ()** method, from user and store.
2. Pass the string to method '**int[] CategorizeCharactersFromString (string sourceStr)'** and write the logic in that method to count the different types of characters.
3. Return the count from **int[] CategorizeCharactersFromString (string sourceStr)** to Main() and display the values.
4. Template Code of the same is provided at:

   **Project_Directory/NamespaceName/dotnet run**

#### Use Case 2

**Take input of a string from the user and convert it into pig latin.**

**Description**

1. Take console input of a string, in the Main **()** method from user.
2. Pass the string to method '**string ConvertStringToPiglatin(string sourceStr)'** and write the logic in that method to convert the source string piglatin form.
3. Return the result string from **string ConvertStringToPiglatin(string sourceStr)'** to Main() and display the result string.

   If vowels existed in the source string, - returns valid piglatin string

   Example: **source string**: techademy, **piglatin form**: 'echademytay'

   i.e. split the source string up to the 1st vowel index and place that part at the end of the source string. And finally append the 'ay' to the result string

   Else returns "-1"

4 Use Sub Function to check if the source string has Vowels: '**bool IsVowelExisted (string source)'.**

**Return – true, if vowels found**

**False if vowels not found**

5. Template Code of the same is provided at:

   **Project_Directory/NamespaceName/dotnet run**

## Use Case 3

**Take input of a number from user and calculate the number of division operations (divide by  2) performed on the number, until the number becomes 1.**

### Description

1. Take console input of a number, in **Main ()** method from the user.
2. Pass the string to method '**int GetCountOfDivisionOperations(int number)'** and write the logic in that method to get the count of the division (by 2) operations performed, until number becomes 1. While working the logic, have to check the number is odd/even and apply the below logic:

   If number is Odd number – add 1

   If the number is even – divide by 2.
3. Return the result count from '**int GetCountOfDivisionOperations(int number)'** to Main() and display the result value.
4. Template Code of the same is provided at:

   **Project_Directory/NamespaceName/dotnet run**

## 2. Considerations

- Your code will also be evaluated for code quality, naming conventions, readability etc.
- Make sure you do not modify existing class and method names and their signatures, else it would severely affect the final result.
- Make sure you do not add any new class or methods, else it would severely affect the final result.
- Make sure you do not modify any test cases, else it would severely affect final result

## 3. Resources available:

### 3.1 Packages

| Names | Resource | Remarks | Status |
|---|---|---|---|
| Package Structure/Project | | | |
| CategorizeCharacters | Program.cs | This file contains business logic to return characters in category. | Partially Implemented |
| NumberOperations | Program.cs | This class contains all the business logic related to number operations. | Partially Implemented |
| PigLatin | Program.cs | This class contains business logic to returns valid piglatin string. | Partially Implemented |

### 3.2 Package: YakshaEvaluation_Test

**Resources**

**Note: - Under the YakshaEvaluation_Test  contains all test cases for code evaluation, please don't try to alter and edit it.**

## 4. Execution Steps to Follow

1. All actions like build, compile, running application, running test cases will be through Command Terminal.

2. To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top)  Terminal → New Terminal.

3. You need to follow steps (4,5,6,8) for all 3 use cases.
   Project folders: (CategorizeCharacters, NumberOperations, PigLatin)

4. On command prompt, cd into your project folder (cd <Your-Project-folder>).

5. To build your project use command:
   (<Your-Project-folder>/ dotnet build)

6. To launch your application, Run the following command to run the application:
   (<Your-Project-folder> / dotnet run)

7. This editor Auto Saves the code.

8. To run the test cases in CMD, Run the following command to test the application:
   (YakshaEvaluation_Test/dotnet test --logger "console;verbosity=detailed")
   (You can run this command multiple times to identify the test case status,
   and refactor code  to make maximum test cases passed before final submission)

9. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B  - command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.

10. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

11. You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.