PRG 410 – Web Server Programming – Week 1 – Assignment

I – Overview

The purpose of this assignment is to create a Web Server that connect to a database.

From PRG 310, we remember that a typical connection string looks like the following:

```
String s = @"Data Source=localhost;Database=LOGIN;User Id=root;Password="";SSL
Mode=None";
```

So in order to establish a connection string, we need the following parameters:

- Data Source (an IP Address or domain name)
- The name of the database (example: LOGIN)
- User Id (example: root)
- Password (example, none)
- Secure Socket Layer (example: None)

An example of that can be found in the figure below:

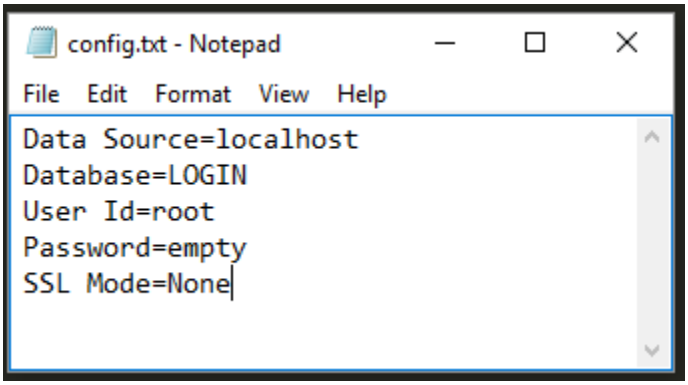

Figure 1 - An Example of Database Config File

I – 1 – Retrieving the data from the file

The following code snippet can be used to access the content of a file, one line at a time

```
using System.IO;

string line = "empty";

System.IO.StreamReader file = new System.IO.StreamReader("config.txt");
while ((line =! null) {
	line = file.ReadLine();
}
```

In this assignment, the parameters used to initialize an SQL connection are coming from a file (we are "streaming content" from a file to the user – space application, as the cool kids like to say 😉).

Now we need to make sure QA is testing our application correctly: Data Marshalling.

If you take a look at figure Figure 1, you will see that each parameter (each line), follows the pattern: **<word><equal sign><word>**, so now, not only we need to retrieve the content from a file, but we also need to make sure that the content within the file has been entered properly. We use Regular Expressions for that.

I.2 – A brief introduction to Regular Expressions

Regular Expressions are a programming concept which allows software developers to describe patterns in text. For example:

- The beginning of a pattern is represented by the "hat/caret" sign: **^**
- The end of a patterns is represented by the "dollars sign: **$**
- A sequence of digits is represented by the "**\d**" sequence (escaped "d")
- If the same the same sequence is supposed to be found between M and N **time (where [M N] is a range)**, then the portion of a regular expression expressing that is going to be: **"'d{M,N}"**. That means if we want to ensure entered between 1 to 3 digits, we would implement that as **"\d{1,3}"**, which has the effect of making sure that part of our pattern contains between 1 up to 3 digits**. Lesson Learned: a digit is represented by "\d"**. If one knows the range of how many digits he/she is expecting, the corresponding RegEx, is going to be "**^\d{1,3}$**".

II – The Assignment (100 Points)

II – 1 – Gathering Configuration Data From a Text File (30 Points)

Using the information in section I.1, write code to gather information the file described in Figure 1.

II – 2 – Write Code to Ensure The Data Is Entered Properly (30 Points)

Using regex101.com and its C# code generator, write code to ensure that the data entered in config.txt is entered correctly. The code developed in class for that would be:

```
            int counter = 0;
            string line;

            // Read the file and display it line by line.
            System.IO.StreamReader file =
                new System.IO.StreamReader(@"config.txt");
            string pattern = @"^[a-zA-Z0-9_ ]*=[a-zA-Z0-9_ ]*$";
            RegexOptions options = RegexOptions.Singleline;
            int regex_count = 0;
            while ((line = file.ReadLine()) != null)
            {

                string input = line;

                MatchCollection collection = Regex.Matches(input, pattern, options);
                if( collection.Count != 1 )
                {
                    Console.WriteLine("QA, please don't mness with Gods.");
                    break;
                } else
                {
```

```
                Match m = collection[0];
                regex_count++;
            }
            counter++;
        }

        if(regex_count == 5 )
        {
            Console.WriteLine("Good Job - QA");
        }
        file.Close();
    }
```

II – 3 – Write Specific Regular Expressions to Check for IP Addresses Or Domain Names for the Data Source

If you take a look at Figure 1, you'll notice that the first attribute (Data Source) is used to specify where our database (from PRG 310) is hosted at (locally or remotely), and that field can accept either:

- An IP v4 address [1 to 3 digits, a period] [1 to 3 digits, a period] [1 to 3 digits, a period] [1 to 3 digits, NO period]
- A URL

You need to build a regular expression for each pattern (IPv4 or URL), and make sure that value to the right of the equal sign (for Data Source) matches either regular expressions you created (one for URLs, another one for IPv4).

II – 4- Write Our Own Exception Class To Handle the Case Where the Config File Contains Error (40 points)

Use the below reference for that
http://www.tutorialsteacher.com/csharp/custom-exception-csharp

Of course, we want to change the error message to "Configuration file missing", or "config file not found" if either the file that we use to establish a connection to a database is missing, or the data within it is not correct.