Distributed Computing Assignment Report

Unit Code: COMP3008

By Aaron Gangemi

Student Number: 19447337

Design Choices:

2.1.1: Three Tier .NET Remoting Application

The three-tier .NET Remoting application comprises of a .NET remoting server, a web server, and the main GUI. This solution was a combination of tutorials 1-3. Once loaded, the GUI presents fields to allow the user to find a user by index or last name. For design purposes, I did not use a random string generator to generate the first name and last name. Instead, I have taken and referenced a list of 25 first and last names and stored them in an array, which are then displayed upon retrieval. Additionally, I have taken an image I found on Pinterest to use as a default image, until the user uploads an image from their own device to represent their profile.

Once the user searches by last name, the progress bar will be used to indicate the searching progress of 100,000 results. A timer has been implemented, that if the progress bar has been running for 150 seconds, the program will effectively timeout the search and assume no results have been found. This has been done as I have calculated that the program will be able to search all results in 2.5 minutes.

An additional feature I have implemented is to allow the user to update their own personal information for their account. If the user searches by name or index and wishes to modify their account, they may click on any of the displayed fields, modify the information then click the "Update User" button. This will then send the updated information to the data tier using the business tier/web service.

Furthermore, I have taken measures to ensure that the Three Tier .NET remoting application will not crash. In the presentation tier (GUI thread), I have caught various exceptions including:

- Format Exceptions: to ensure any data that is being parsed does not crash the program.
- JSON Reader Exceptions: to ensure that if the business tier cannot produce a response due to the URL change, then this will not crash the program.
- Null Reference Exception: to ensure that any response from the business tier does not return null.
- Using regular expressions to validate correct data types

If any of these errors are caught, a message box is displayed to the user stating the error and the error is logged in a file under the directory "WPFApp/LogFiles/log.txt". Success messages are also logged. I have implemented the logging feature by creating a log class which is stored in the Bis-GUI project and connecting it up successfully to the presentation tier. I chose to store the log class at the forefront of the business tier GUI representation as logging data in the presentation tier requires the response from the business tier to go ahead.

2.1.2 Three Tier Web Service Application

The three-tier web service application consists of a data tier and a combined business/presentation hybrid tier. The data tier successfully implements the functions that are stated in the tutorial regarding users, accounts, and transactions. In the business tier, I have chosen to call the “ProcessTransactions()” function on a timer of 30 seconds. Once a user completes a transaction, the transaction ID is displayed in the console, and the user will have 30 seconds to use this ID to retrieve the transaction before it is processed. In the presentation tier, I have made the single page web application entirely green with rounded green buttons which provide all the functions of the business tier on a single web page. I completed this by modifying the site.css file to provide a consistent and colorful layout for the project.

In the presentation tier, I have made my program such that each user will be default have an account. This means that when the user enters their first name and last name to create an account, the business tier will also make a request to create an account for the generated user ID.

In terms of error handling, when the presentation tier submits an ajax request to the business tier, the business tier will return either one of two responses. The first is it will return the desired response, and the second is it will return null. Therefore, to allow for this, in each success function in the presentation tier, I have included a check to determine if the response is “null” or “false”. If either of these responses is null or false, then the presentation tier will display an alert to the user that something has gone wrong. In addition, the business tier utilizes regular expressions to perform checks on valid data input. If the data provided is an invalid data type, then the business tier will return null or false, and the error will be caught and sent back without checking with the data tier. Additionally, I have implemented a log class which is used to either log a success message or an error to file. The business tier successfully utilizes this class and outputs an accurate message of what went wrong. The log file appends data to the file stored in the directory: “Tutorial4LogFiles/log.txt”

I have also added, for aesthetic purposes, a logo which I generated using https://www.freelogodesign.org/.

2.1.3 Peer to Peer Application

For the peer to peer application for tutorial 6, I have constructed the program, such that each client is able to execute a python script as long as it starts with “def main()” and contains a return value. I have allowed the text box, in which the user enters the script to allow for python indentation and new line if they are to press the tab or enter key. I have modified the web browser to display the number of jobs that each client has completed.

Furthermore, I have implemented error handling for both the python code and C# code. If the python script entered is null or empty, then the error is caught, and a message box is displayed back to the user. If the python script cannot be executed, the program catches syntax errors, unbound name exceptions and null reference exceptions to account for anything that could go wrong with the python script.

If anything goes wrong with the communication between clients, such as a client being removed, I have accounted for different exception types to be caught including:

- EndpointNotFoundException which will remove any client that is inactive, and has not been removed
- FaultException which will also remove an inactive client
- CommunicationException which will refresh the client list
- TaskCancelledException which accounts for the dispatcher having to abort.

I have also implemented a logging feature which allows the program to log all errors or success messages to file. These will be appended to the file: “Tutorial6LogFiles/log.txt”.

2.1.4 Blockchain Applications

The smart contract implemented in tutorial 9 utilizes a mining thread and a blockchain thread to implement the smart contract. The blockchain thread connects to the client to the .NET remoting server. If the client attempts to connect to a port that is already in use, an “AddressAlreadyInUseException” will be caught and the client will try again by incrementing the port number.

For the mining thread to process transactions, I have implemented a static transaction queue which stores transaction that are to be processed. A queue seemed appropriate as its first-in-first-out approach to storing transactions proved useful for processing transactions.

To disallow multiple clients to submit transactions to the queue at the same time and to ensure mutual exclusion, I have also implemented a static mutex. By calling Mutex.WaitOne(), the mutex acts as a lock which has been acquired and no other thread can continue execution in the submission function until this mutex calls Mutex.Release().

An additional thread I have chosen to implement was an updating thread. This thread runs infinitely and every 10 seconds, using the Thread.Sleep() method, it will update the submitted jobs list for each client to display which clients have submitted which jobs.

In the GUI thread, I have incorporated a list box to display submitted transactions by the associated client and another list box to display which python scripts have been executed. This job list will be the same across all clients. Using the work of previous tutorials, I have also ensured that all clients blockchains are in sync.

In terms of error handling, I catch the following exceptions:

- EndpointNotFoundException which will remove any client that is inactive, yet has not been removed
- FaultException which will also remove an inactive client
- FormatException: Which clears the queue if a transaction goes wrong
- SyntaxErrorException, UnboundNameException and NullReferenceException to account for any syntax errors with the python code.

If any of the errors are found, then the program displays a message box to alert the user. I utilize the Debug.WriteLine() function to log data to the console. This occurs in any error occurring or success of a submitted block to the chain, and if the chain is required to update a client.

Finally, upon submission of a transaction, the program performs a range of validation to ensure the python program contains a return value, starts with “def main()” and is not null or empty.

Running each program:

I have modified the properties of each solution to run multiple projects in the correct order.

Once the program has been opened, click the start button at the top of the Visual Studio window and wait for the program to run.