Assignment 1

1. A Bank Application

In this unit you've learned the basics of a Distributed system. Now you're going to build one from scratch by yourself! Specifically, you're going to be building a banking application. The system will allow for users to be added, for accounts to be created and assigned, and for transactions to be created and committed.

The system is going to consist of three parts:

- A Data tier, which connects to the BankDB object in the BankDB.dll available with this assignment.
- A Business tier, which allows for filtration of certain fields for user views.
- A Windows desktop application, with three windows, one for user management, one for account management, and one for transaction management.

2. The Data Tier

The data tier for your application will be making the BankDB object (which you must reference in your C# project) available to the rest of the system. This is not a trivial task! In this particular case, the bank is paranoid about hackers stealing customer money, so the BankDB object is itself an interface to the bank system. The system itself being hidden away in the DLL where you can't get to it.

In order to communicate with the Bank system itself, you must create a single, static BankDB.BankDB object. This will let you create UserInterfaces, AccountInterfaces, and TransactionInterfaces, each of which allow the data tier to communicate via the BankDB interface to the bank itself.

These interfaces act almost like virtual butlers. In order to use them properly, you need to select a user, account, or transaction before doing any work on them. You can also create a user, account, or transaction, but you will need to then select it before you can do anything. This will mean that simply creating data tier functions to directly call access interface methods is not enough! You will need to think about how you implement this carefully.

The BankDB object also provides two management functions. The first one is simple, SaveToDisk() will save all users and accounts currently in the system to disk. You will need to create a folder on your C:\ drive, called C:\WebStuff\ for this to work properly. The second management function is ProcessAllTransactions(). This will commit all transactions between accounts. You may have noticed that SaveToDisk() does not save transactions, this is because (at least to the bank) transactions should be ephemeral, and processed by the end of the day. This means that the correct save process for closing the application is to first

process all transactions, and then save to disk. You will need to devise a process to make sure this always happens at regular intervals!

The BankDB processes can return exceptions, however due to the mysterious methods used by the BankDB system, you don't have access to the custom exceptions thrown by it. This means that you will have to determine which exception has been thrown based on the information in the exception (message, and which function you were calling) and then re-raise your own exceptions!

The documentation of the BankDB library is as follows:

2.1 BankDB

Governs access to the Bank system.
• void SaveToDisk(): saves the current state of the Bank System to disk.

- void ProcessAllTransactions(): Processes all transactions currently pending in the system. Will return an exception if the transaction is invalid for any reason.
- TransactionAccessInterface GetTransactionInterface():provisions and returns a Transaction Access object for access to the Transaction subsystem.
- UserAccessInterface GetUserAccess(): provisions and returns a User Access object for access to the User subsystem.
- AccountAccessInterface GetAccountInterface():provisions and returns an Account Access object for access to the Account subsystem.

2.2 AccountAccessInterface

Governs access to the Account subsystem.

- void SelectAccount(uint accountID): Selects an account.
- List<uint> GetAccountIDsByUser(uint userID): Lists all accounts associated with a given User.
- uint CreateAccount(uint userID): Creates an account for the specified user, and returns it's ID.
- void Deposit(uint amount): Deposits money in a selected account.
- void Withdraw(uint amount): Withdraws money from a selected account.
- uint GetBalance(): Gets the balance from a given account.

- uint GetOwner(): Gets the User ID of a given account.

- ## 2.3 UserAccessInterface
  Governs access to the User subsystem.

- void SelectUser(uint userID): Selects a user.

- List<uint> GetUsers(): Gets a list of all users.

- uint CreateUser(): Creates a new user, and returns that user's ID.

- void GetUserName(out string fname, out string lname):Gets the selected User's name.

- void SetUserName(string fname, string lname):Sets the selected User'sname.

- ## 2.4 TransactionAccessInterface
  Governs access to the Transaction subsystem.

- void SelectTransaction(uint TransactionID): Selects a transaction.

- List<uint> GetTransactions(): Gets a list of all transactions.

- uint CreateTransaction(): Creates a transaction and returns it's ID.

- uint GetAmount(): Gets the amount that will be transferred from the selected transaction.

- uint GetSendrAcct():Gets the account ID of the sender of the selected transaction.

- uint GetRecvrAcct():Gets the account ID of the receiver of the selected transaction.

- void SetAmount(uint amount):Sets the amount that will be transferred from the selected transaction.

- void SetSendr(uint acctID): Sets the account ID of the sender of the selected transaction.

- void SetRecvr(uint acctID): Sets the account ID of the receiver of the selected transaction.

3. The Business Tier

The business tier needs to connect to the data tier via .NET Remoting and implement functionality that users should be able to access. Clearly being able to view every User ID, or every transaction regardless of owner is a bad idea. Users should not be able to see any details about transactions or accounts they do not own. The business tier should implement, amongst other features:

- Searching for transactions per account.

- Checking transactions and with drawls to make sure that an account has enough money before trying to do it.

4. The Windows Client

The Windows client needs to allow a User to be created, or be selected, and for all aspects of the banking application to be accessible. This means that Users should be able to:

- Create and manage accounts.
- Create and manage transactions.
- Create and manage their own User record.

They should not be able to save or process transactions, the business and data tiers are responsible for determining when those actions should occur. The windows client should, use separate windows for the User, Account, and Transaction subsystems. It should also use asynchronous communications for the search methods (at the very least) to improve user experience.

5. XSS Application

You will also need to submit an XSS application for the assignment. This is in addition to the rest of the tutorial work. You need to make a web application that has all three kinds of XSS attack vulnerability present.

So, for each of the three attack types:

- Reflective XSS
- Persistent XSS
- DOM XSS

You need to have
- A page that is vulnerable to the attack
- An explanation of how to cause the attack, along with an example.

• An explanation of how to prevent the attack.
• A page that is identical to the vulnerable page, but that you've altered to prevent the attack.

This should follow the way you've described in your explanation.

6. Documentation

Your programs should be well documented. This means that each function should be fully commented, and that you must write a document that outlines the objects, functions, and what they do. Essentially, this should be aimed at someone who would want to maintain or develop on your platform

You will also need to produce three UML Diagrams, one for each tier of the solution (Data, Business, and Presentation). These UML diagrams should only include your objects, not objects managed by the system, and not objects included from the BankDB system.

You should also write a document outlining your design choices. This should be 1 - 3 pages long, and include how you managed to get around the design difficulties associated with the BankDB system.