Distribution and Integration Technologies (TDIN)

Project #1

.NET Remoting

Scenario

A bank needs a distributed application (*Diginote Exchange System*) for allowing its customers to buy, sell and obtain a quote of digital values called *diginotes*. It should be based on *.NET Remoting*. Despite some resemblance with *bitcoins*, this system centralizes the information, taking note of the *diginotes* existence, and who is the current owner of each one. Each *diginote* is represented by an object, has a unique serial number, and a facial value (to simplify we only have *diginotes* of just a facial value of 1). All the *diginotes* are registered in the system (through their serial number), as also the current owner which should be registered on the system. Registered users (with a name, *nick name* and *password*) are the current owners of *diginotes* and anyone interested in acquiring some. The system allows only selling and buying *diginotes*, verifying its existence, and taking note of the new owner, when there is some transaction. *Diginotes* have always a quote (initially its equal to €1.00). When someone pretend to buy or sell *diginotes*, always at the current quote, he should emit a buying or selling order, allowing the system to pair those orders, according to specific rules enumerated in the next section.

Operations

- *Login/logout/register* – All users must identify themselves in the system (or register if it is the first time) with their *nick names* and *passwords* before they are allowed to execute any other operation.

- *Obtain the current quote* – Each user, when *logged in*, and in the client application window, should always see the current quote in real time.

- *Emitting a selling order* – the owners of *diginotes* can emit a selling order specifying the number of *diginotes* to sell. If, at that moment, there were in the system active purchase orders enough for satisfying the number in the selling order, or only to fulfill a part of the order, the operation takes place, at the current quote, in the possible number of *diginotes*. When the purchase orders are not enough to satisfy the selling order, the seller should indicate a price for the *diginotes* not yet transacted in his order, and that **must** be <u>equal or smaller</u> than the current quote. The system takes this price as the new current quote.

- *Emitting a purchase order* – the users that want to buy '*diginotes*' can emit a purchase order indicating the number of *diginotes* to buy. If, at that moment, there is enough offer (pending selling orders), or just a partial fulfillment, the corresponding transaction is made at the current quote. When the offer is not enough to completely fulfill the order, the buyer **must** specify a price <u>equal or greater</u> than the current quote. The system takes this price as the new current quote.

- *Increase or decrease price offers* – At any time the emitters of pending selling or purchase orders can decrease or increase the offered price respectively, determining this way a new current quote.

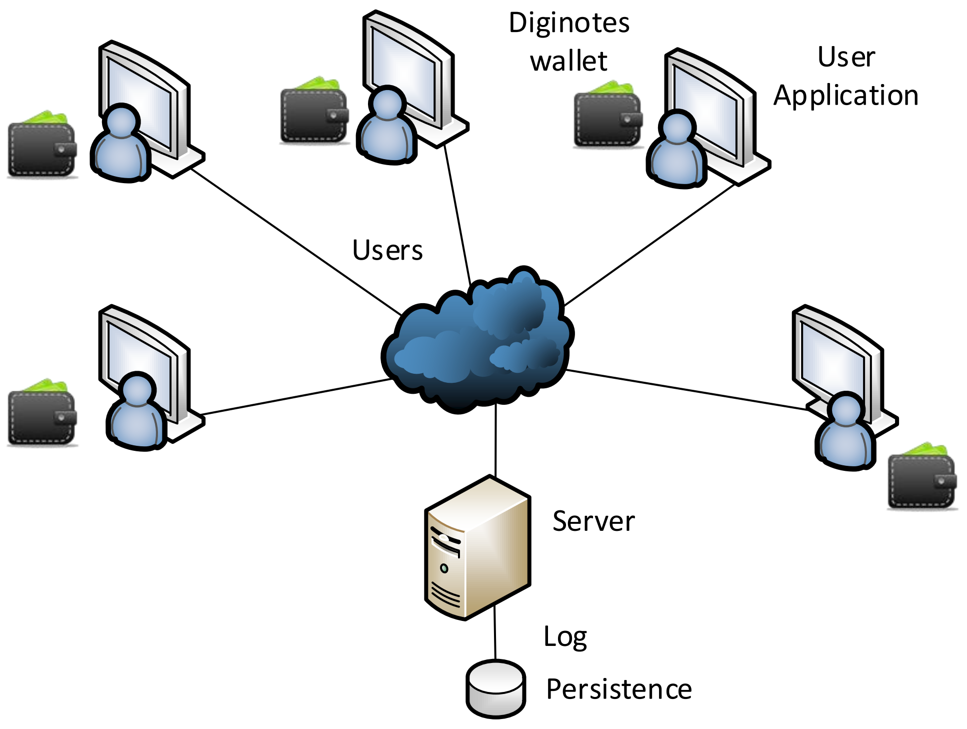
- *Confirm the price of an order* – Whenever the current quote drops and there are other pending selling orders, those are suspended during a time interval (for testing let's use just 1 minute or less). During this suspension time the order emitter can confirm the new quote or withdraw the order. If, after the suspension period, there is no action from the emitter the system assumes a confirmation, making them active again at the current quote. In the same way, whenever the quote rises and there are other pending purchase orders, those are suspended for confirmation of the new price or for withdraw. Again, the system presumes that the orders are confirmed if there is no action from the emitter during the suspension period.

- When the system makes a transaction, pairing pending orders with the one just arrived, it should start with the older pending orders.

- When a selling order is emitted, the indicated *diginotes* serial numbers (each user can maintain a wallet storing his *diginotes*) are verified by the system (their existence and owner). When the transaction is completed, and as soon as possible, the owner is changed in the system, and the transacted serial numbers are sent to both users, allowing them to update their wallets.

- The system should persist a *log* of all transactions. It should also persist all the needed information to recover from a crash or resume normal operation after a shutdown.

- All the relevant changes (for each user) should be shown to the logged in users in their client application interfaces, as soon as possible, after taking place.



Implementation

You should implement a demonstrator of this project using *.NET Remoting*, with the needed GUI application for the users, and the remote objects residing in a central server. All the good practices for .NET Remoting should be included for reliability improvement. The user application should have a simple interface but intuitive, user friendly, and easy to operate. You can add all convenient functionalities, like statistics, graphics, log or past transaction visualization in the user or in some administrative application …
The correct implementation of all stated requirements and rules, correct interface and easiness of operation, set of functionalities, good implementation practices, are relevant factors.

Report

A report should also be elaborated describing the system architecture (applications, modules, objects and their interactions), included functionalities, performed tests, and operation instructions (captures of the main use flows).
Also, instructions to build and initialize the demonstrator should be supplied.