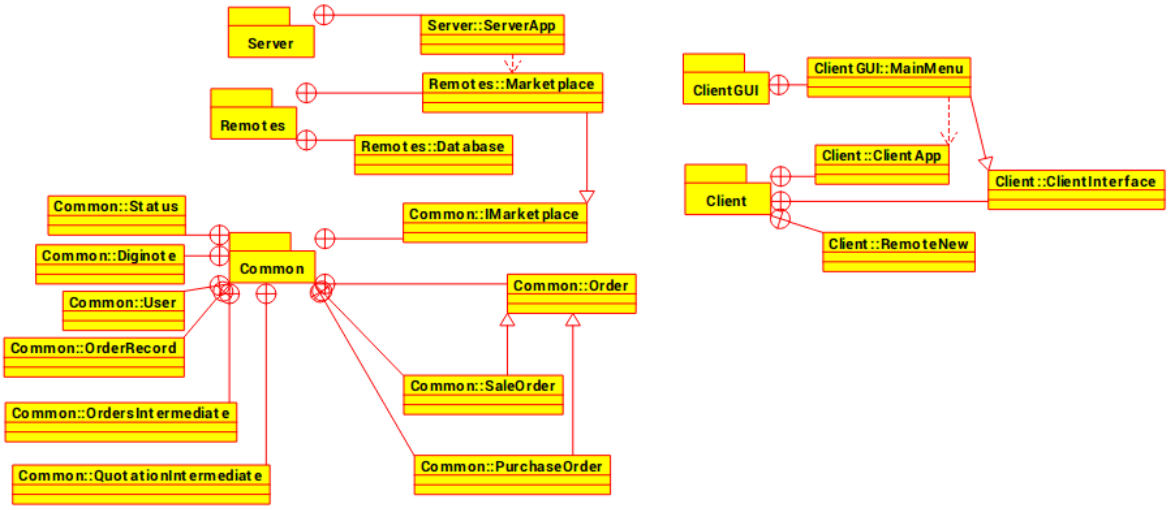
TDIN 1st Project

Diginotes Marketplace

Architecture



(does not contain many of the dependencies as the image would be unreadable)

The solution is subdivided in 5 projects:

- **Common** (library)
  Provides all the content that has to be accessible both to the user and the server applications. It's intent is to be distributed alongside the application to the clients. This includes the interface for the *Marketplace*, the classes for *User*, *Diginote*, *Order*, as well as classes responsible for interconnecting the event triggering from the server to the client (*OrdersIntermediate* for orders dispatch notifications and *QuotationIntermediate* for quotation's update notifications).

- **Remotes** (library)
  Requires *Common*.
  Contains the implementation of *Marketplace*, the shared object in singleton mode, as well as the *Database*, stored persistently through Serialization, which contains all the information about users, pending orders and execution history

- **Server** (executable)

Requires *Common* and *Remotes*.
Runs the server application, defining the shared object *Marketplace* as a singleton object.

- **Client** (library)
  Requires *Common*.
  Provides the API to be used by the final client application, be it based on command line or graphical user interface. Is responsible for interacting directly with the shared object.

- **ClientGUI** (executable)
  Requires Client.
  Provides the interface for the client, using the *Client* API.

Functionality

The application provides all the functionality required, implemented according to their desired behaviour, namely

- register of new user;
- login and logout;
- creation of purchase and sale orders
- edition of pending purchase and sale orders
- logging of the transfers
- persistent storage of the database
- updated quotation in all users
- dispatching of orders

Alongside those, we keep track of the balance of each user. This is a metric that enables him/her to measure his/her success (positive represents profit, and negative represents loss). It is updated every time there is a purchase or a sell, being decremented or incremented respectively by the amount of diginotes involved times the current quotation.

As this is a quotation market simulation, plots are very useful to analyse changes with time. As such, we've added two plots, one for the balance of the user, and another for the global quotation, updated each time a relevant action occurs.

Tests Executed

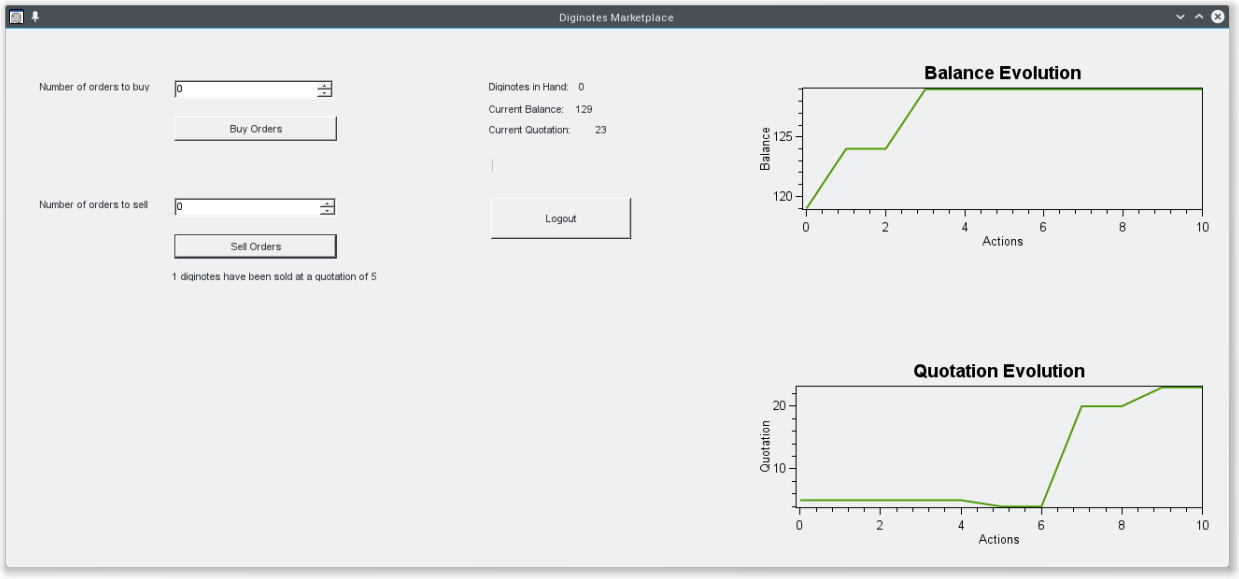
The system was tested globally, passing all the criteria specified. Among other tests, the main were:

- decrement of the quotation, to check if the orders are only dispatched after one minute;
- update of orders values;
- registration of new users;

- login with an user already logged-in, expecting a warning;
- produce orders that can be dispatched immediately, to check if it is possible to update the quotation, expecting otherwise;
- change the quotation to invalid values (lower or higher than the current being the order a purchase of a sale respectively), expecting a warning;
- login com utilizador invalido;

The application was also tested with instances of clients and the server on different machines on the same local network.

Flow of execution



A new user to the platform performs the following steps:

1. Register, providing his name, username and password
2. Login
3. Perform one of the following, as many times as desired:
    - Add purchase order
        - May involve update of the current quotation
    - Add purchase order
        - May involve update of the current quotation
    - Update purchase order
    - Update sale order
4. Logout

While this process executes, the user may be warned about dispatched orders, as well as check his balance, current quotation and current number of diginotes he possesses, as well as analysing the updates on his/her balance and on global quotation.

Instructions for use

The solution should be loaded into Visual Studio, and compiled. After this, it is only needed to run the output artifact of the Server project to instantiate a server, and the output artifact of ClientGUI for a client instance.