**Introduction**

The project supposed to serve users to communicate and send messages through the server. The client connects and communicates with a server and performs activities such as login, registration, send, receives and display messages.

**Presentation Layer-Graphical User Interface**

**Main Window**

The start-up component which responsible to initialize the program , show the entrance window for the user and navigate to the suit window by the user selection.

**Attributes**

Private logger: Logger the singletone that responsible for the action of log writing.

Private chatroom: Chatroom the initialized instance of the current user.

**Functionality**

Private button\_exit\_Click(object sender, RoutedEventArgs e); exit the program when “exit” button is clicked.

Private button\_register\_Click(object sender, RoutedEventArgs e); opens the LoginRegister window with the enum attribute type of register when “register” button is clicked.

Private button\_login\_Click(object sender, RoutedEventArgs e); opens the LoginRegister window with the enum attribute type of login when “Login” button is clicked.

**LoginRegister Window**

This component responsible to handle the user inserted username and group ID details and perform validation tests for the input against the data saved in the file system.

All of the above performed accordingly to the clicked button (Login/Register).

**Attributes**

Private \_securedPassword: string holds the hashed password.

Private \_passwordIsValid: bool flag that tells if inserted password is valid.

Private \_enterTypeStatus: int holds the window appearance and act status (login/registration).

Private chatroom: Chatroom the initialized instance of the current user.

Private logger: Logger the initialized received singletone that responsible for the action of log writing.

Private userLoggedIn: bool a flag which tells if the current user is logged in to the system.

Private loggedInNickname:string holds the nickname of the logged in user.

lastRegistered: string[] the group ID and nickname of the last registered user.

Private \_main: ObrModels an instance of the observable class which responsible to connect between the xaml file to the ‘code behind’.

**Functionality**

Private pswrd\_hasChanged(object sender, RoutedEventArgs eventArgs);this event handles the changes in the password passwordbox, hashing it and checks for validation of inserted password. Then transfers the secured hashed password.

Private user\_name\_textBoxHasChanged(object sender, RoutedEventArgs eventArgs); refresh the user name text box.

Private loginRegisterProcess(object sender , RoutedEventArgs eventArgs); navigate the user according to its inserted inputs.

private checkValidity (string str, int type, Logger logger); returns true of false whether the user's input has only valid characters.

Private button\_logReg\_Click(object sender, RoutedEventArgs e); calls for loginRegisterProcess when the button register or Login is clicked.

Private goToMainWindow(); opens a ain window window and closes current window.

Private button\_Back\_Click(object sender, RoutedEventArgs e); this method navigate the user back to the main menu when “back” button is clicked.

Private comboBox\_g\_id\_Init(); this method responsible for initializing the Group ID combobox item values in group ID’s.

Private goToMainWindow (); an assistance method which is called from the back button click action.

checkValidityPass(string str,Logger logger); checks the validity of inserted password – English and numbers characters only in the required length (4<=length<=16).

**Chatroom Window**

This component presents the functionalities inside the Chat-room.

In addition it displays the selected actions on screen accordingly.

**Attributes**

Private chatroom: Chatroom the initialized instance of the current user.

Private logger: Logger the initialized received singletone that responsible for the action of log writing.

Private \_isEdit:bool a flag tells if current sent message is for update or sending. True for update and false for send.

Private \_pointedMessage:MessageItem holds the current chosen message to edit.

Private sortby: IFilterSort holds the chosen user sort criteria.

Private filterby: IFilterSort holds the chosen user filter criteria.

Private filterAndSort:List<IFilterSort> list of chosen filters and sorts criteria.

Private \_main: ObrModels an instance of the observable class which responsible to connect between the xaml file to the ‘code behind’.

Private loggedInNickname:string holds the nickname of the logged in user.

Private clearChatroom:bool flag that tells if the chat-room messages needs to be clear.

Private usersList:string a list which holds the users nickname which are saved in the system.

dispatcherTimer: DispatcherTimer an instance of DispatcherTimer. It is the trigger for the refresh action.

**Functionality**

Private dispatcherTimer\_Tick(object sender,EventArgs e); this method is called to refresh the messages screen using refresh function

Private chatRefresh();refreshes the chatroom messages according to user's sorting and filtering choices.

Public comboBoxFilteByUserOptions(); this sends the users list to the combo box which contains the users list

Private FILTER\_SORT\_Click(object sender, RoutedEventArgs e); this apply the changes were made by the user in the chosen filter and sort criteria.

Private listBoxLoad(object sender,RoutedEventArgs e); scroll down the listBox automatically.

Private button\_Send\_Click(object sender, RoutedEventArgs e);this method navigates to the method send or edit accordingly the \_isEdit flag, when the “send” button is clicked.

Private edit();performs a message update action in aim to edit exist message

Private send();transfer a send request with the message content to the Business Layer.

Private indicatorColor(string colour); this method responsible to color the indicator update messages.

Private Button\_Logout(object sender, RoutedEventArgs e); this event log out the user and navigate to the main window of the program.

Private textBox\_NewMessage\_TextChanged(object sender, TextChangedEventArgs e); this method updates the textbox content binding property when the new message text box is changed by the user

Private send\_keyDown(object sender, KeyEventArgs e); connects between the “Enter” key pressing to the send click event.

Private textBox\_NewMessage\_gotFocus(object sender, RoutedEventArgs e); when user clicks on textbox it gets focus.

Private textBox\_NewMessage\_LostFocus(object sender, RoutedEventArgs e); when user clicks out of textbox it lost the focus.

Private edit\_Click(object sender, RoutedEventArgs e); set the properties of the edit action process in their fit state according to the user choose.

**MessageItem Class**

This class holds the properties with the messages with their edit buttons and responsible for data binding them.

This class implements IEquatable and INotifyPropertyChanged.

**Attributes**

Private \_message: ReadOnlyMessage holds a certain message in type of ReadOnlyMessage.

Private \_buttonVisibility:string holds the visibility state of an edit button.

Private \_isSelected:bool flag that tells if an edit button is selected.

**Functionality**

Public changeIcon();select the source path of the edit button image;

**Obr Model Class**

This class responsible to connect between the XAML GUI to the ‘code behind’ .cs files.

This class implements the interface “INotifyPropertyChanged” and includes the properties in aim to do so (the connection).

**Attributes**

public Messages: ObservableCollection<MessageItem> -messages list

public FilterByGidList: ObservableCollection<string>-holds set of values for the group id combo box.

public FilterByGidNickname: ObservableCollection<string>- holds set of values for the users combo box.

public g\_id\_selection: ObservableCollection<string>-holds set of values for the group id combo box in the LoginRegister window.

public FilterByItems: public ObservableCollection<string>- holds set of values for the filterby combo box.

public SortByItems: ObservableCollection<string>- holds set of values for the filterby combo box.

Private usersList: List<KeyValuePair<string, string>> - holds pairs of group id and nickname for each user in the users list.

Private FilterByCombobox:string-holds the selection filter criteria to show.

Private \_g\_id\_selected:string- holds the selected grop id from the combobox.

Private \_FilterGidView:string- holds the visibility status of groupID combobox in the chatroom window.

Private \_ \_FilterNicknameView:string- holds the visibility status of nickname combobox in the chatroom window.

Private \_filterByGid:string – holds the selected value of group ID in the group id combobox inside filterby options in chatroom window.

Private \_filterByNickname:string – holds the selected value of nickname in the nickname combobox inside filterby options in chatroom window.

Private \_logRegStatus:string-holds the appearance status of “LoginRegister” window.

Private \_logRegBtnStatus:string – holds the string should be displayed on the button that sends the user inserted details in LoginRegister window.

Private \_\_ascending:bool- for ascending messages order selection, this should hold true .

Private\_\_descending:bool- for descending messages order selection, this should hold true .

Private \_SortByCombobox:string-holds the current sort-by criteria.

Private \_messageContent:string – holds the current message to display inside the send messages textbox.

Private \_loggedInUser:string – holds the current logged in user nickname to show in logged in label which is located in the chatroom window.

Private \_serverTime:string- holds the time received from the server in aim to display it in the chatroom window.

Private \_indicator:string- holds the system update messages to show as “Last update” in Chatroom window.

Private \_indicatorColor:string- holds the color we should paint the indicator label with.

Private \_user\_name:string- holds the inserted nickname inside the user name text box in LoginRegister window

**Functionality**

Private setItemsFilterBy(); this method set the options of filter-by inside it’s combo box in the chatroom window.

Private setItemsGid (); inserts the group IDs into FilterByGidList property list.

private setItemsNickname();inserts the users nicknames that are saved in the system into FilterByGidNickname property list.

public setGidItemsInComboBox(string s); sets the group IDs to the g\_id\_selection property which responsible to update the group id combo box in the loginRegister window.

Private setItemSortBy; this method set the options of sort-by inside it’s combo box in the chatroom window.

private filterByMenuChanged(); sets the visibility of sub combo box for filter-by criteria according to the user selectionin the Chatroom window.

Private Messages CollectionChanged (object sender, NotifyCollectionChangedEventArgs e); this method updates the messages list box property.

Public OnPropertyChanged([CallerMemberName] string propertyName = null); this method Occurs when a property value changes, and updates the properties which are binds.

**ListBox class:**

Customize list box that extends the list box of WPF with auto scroll option.

**Business Layer**

**Chatroom Class**

This class constitutes as a “User Manager” and as a “Message Manager”.

**Attributes**

private loggedInUser: User holds the link to the current logged in user. null by default.

private url: string server's address.

private messages: List<Message> list of last messages, ordered by timestamp.

private users: List<User> list of users.

private logger: Logger a logger which keeps and traces all errors and messages onto a log file.

private msgHandler: MessageHandler retrieves messages from file-system and keeps new ones for backup.

private usersHandler: UserHandler retrieves users from file-system and keeps new ones for backup.

private MAX\_MESSAGE\_LENGTH: const int the max input length for message.

Private messagesToPrint: List<ReadOnlyMessage> list of ReadOnlyMessage type messages.

private \_lastUpdateTime:DateTime holds the last time there was a retrieve.

Private \_dateTime:DateTime holds current date time.

Private \_ascending:bool holds the sort state for messages display.

**Functionality**

public Chatroom(); default constructor.

public registration(string nickname, int gID); generates a new username and adds to file-system.

public login(String nickname, int gID); asks for username, group id and password entered by user, and activates if matches.

public logout(); disconnects from chatroom.

public retrieveMessages(bool useTime); retrieves last messages from server data base.

public displayLastMsg(int numOfMessages); displays last amount of messages requested by user which are already retrieved.

public displayAllByUser (string nickname, int numOfMessages); displays specified amount of messages sent by a certain user.

public send(string message); collects a message from the user and calls *send* function in user's functionality.

Public refresh(string[] filters, bool checkTime, ISort sorter, bool ascending); takes user's choices of sorting and filters and updates the messages screen accordingly.

Private updateMessages(List<IMessage> msgs, bool clear); enforce messages list update.

Private removeMessage();remove the message in the right place in list according to the sort type (ascending or not) in case there are more than 200 messages in list.

Private addMessage(Message tMsg); add a message to the list of messages.

Public update(ReadOnlyMessage oldMessage, string updatedBody); performs an exist message update with new message body and message send time.

Private messageCheck(string str); checks the validity of the message length.

Private englishCheck(string str); checks the validity of characters in a message.

**Hashing Class**

The class received from the course staff.

This class responsible for encrypting the password string that received from the user by the class methods.

**User Class**

User is an object which defines the user and holds the behavior of send message.

This class implements IEquatable.

**Attributes**

private nickname: string describes user's name.

private charLimitPerMessage: int the limit of characters for a message.

private G\_id: int describes user's group ID number.

private numOfMessages: int counts the number of messages sent by user.

**Functionality**

public User(string nickname, int gID); constructs a new user and then saves user details in the file-system.

public send(string message); wraps a received string into an IMessage object and then creates a Message object.

**Message Class**

Message is an object that wraps an entered string by user with details such as user's nickname, time, etc.

This class implements IEquatable.

**Attributes**

private body: string the message delivered by user.

private nickname: string nickname of the user who sent the message.

Private \_g\_id: string holds the group id of the message’s sender.

private GUID: Guid a unique message ID generated by the server.

private date: DateTime the date the message was published.

**Functionality**

public Message(IMassage message); gets an IMassage object with GUID, timestamp and the message content and finally constructs a new message and stores in the file-system.

public static factory(string message, string url, string g\_id, string nickname, int charLimitPerMsg); the first step in the message's creation factory happens here, this method gets all details about a requested message, if it passes all validity checks, factory asks the constructor to create a new message.

public static checkValidity(string message); validates if the message has maximum of 100 characters.

public toString() prints message with its details.

Public updateBody(string updateBody); update this message with a new content (body).

Public updateDate(DateTime newDateTime); update this message with a new date time.

**MessageConvertor Class:**

This class gather collection of methods which responsible for convert between messages types.

It supports the following convertions:

* IMessage to Message.
* Message to ReadOnlyMessage.
* Message to IMessage.
* ReadOnlyMessage to Message.

**ReadOnlyMessage Class:**

This class main purpose is to be able to transfer data of messages from the Business layer to other layers without reveal the behaviors of the Message class.

**Sort:**

Includes all sorters.

**SortByTimestamp Class**

Implements ISort. This sorter is responsible for sorting by timestamp. In addition, it holds a boolean attribute which specifies whether the sorting is ascending or descending.

**SortByNickname Class**

Implements ISort. This sorter is responsible for sorting by nickname. In addition, it holds a boolean attribute which specifies whether the sorting is ascending or descending.

**SortByAll Class**

Implements ISort. This sorter is responsible for sorting by in the following order: at first by GID, then by nickname and at last by timestamp. In addition, it holds a boolean attribute which specifies whether the sorting is ascending or descending.

**Data Access Layer**

**IQueryAction Interface**

IQueryAction represents the method which a filter class must implement.

This method returns a string represents SQL query.

**GroupFilter class**

Defines a filter type by SQL adding query on the group ID received in the constructor.

Implements IQueryAction.

**NicknameFilter class**

Defines a filter type by SQL adding query on the nickname received in the constructor.

Implements IQueryAction.

**Query Class**

This class defines a general SQL quary which configured in advance and has methods that gives it the flexibility to be updated and extends for user demand.

**IMessage class**

This class defines a message type that includes the required properties for message to be stored in the server data base.

The properties are:

Public \_Id : GUID a unique id for the message.

Public \_UserName : string user nickname.

public \_Date : DateTime send time.

public \_messageContent : string the body of the message.

Public \_GroupID : string the group id of the sender.

**MessageHandler Class**

MessageHandler class's responsibly is to send and retrieve Messages to and from the server.

**Attributes**

private \_filters: List<IQueryAction> a list of SQL filters for query.

Private \_logger:Logger logger

Private connection\_string:string the string which holds the connection attempt command with the server.

Private server\_address:string server address

Private \_lastDateQuery:const string select the last message by time query.

Private database\_name:string db name.

Private user\_name:string user name registered to db

Private password:string password for authentication

**Functionality**

public retrieveMessages(DateTime lastDate); retrieve messages as IMessages list from the server with a date filter

public retrieveMessages():retrieve messages as IMessages list from the server.

Public communicate(string query); responsible to open and close a connection with the server DB and perform a query request from the data base.

Public sendMessage(IMessage msg); this method get IMessage type message and send it to the server.

Public updateMessage(IMessage msg); this method get IMessage type message, then it finds the specified message in the server database and updates it.

Private findUserId(string G\_id, string nickName);returns the ID of a received user from the users table in the data base.

Public getLastDate(); returns the date of the last updated message that is stored in DB.

Private createQuery(); create a new query from the filters that stores in \_filters list.

Private clearFilters(); clear list of query filters (\_filters).

Public addGroupFilter(string gid); add a group filter with the received gid to \_filters.

Public addNicknameFilter(string nickname); add a nickname filter with the received nickname to \_filters.

**UserHandler Class**

UserHandler class's responsibly is to get and set Users to and from the server data base.

**Attributes**

Note: there are the same attributes for data base connection. See MessageHandler attributes.

**Functionality**

Public login(string nickName, string G\_id, string password); checks if there is a match between a registerd user in the data base to the received user details.

public register(string nickName, string G\_id, string password); first checks if there is a user with these details in the data base,if there is no such user, insert the received details into users table (register as a user).

Public getUsers(); returns a list of users with their nickname and group id for each.

**ILogger**

**Logger Class**

The Logger class writes to a text file and documents errors, messages about processes, bad logins/registrations.

**Attributes**

private static readonly log: log4net.ILog the origin logger

private static instance : Logger the log object we use to save messages, which uses the original logger.

public static Instance; returns the *instance logger* object.

**Functionality**

public logInfoMessage(string message); events that happen following the system process.

public logWarnMessage(string message); warnings that occur due to user misuse.

public logErrorMessage(string message); errors that appear when system is not working properly.

public logFatalMessage(string message); errors that crash the system.

**Files**

* Messages.bin ; This file includes the history of messages which sent over the CLI to the server.
* Users.bin ; This file holds information about users which registered into the CLI.

**ClientTests**

**Chatroom\_tests class:**

The Chatroom\_tests class, tests the main functionalities of the class Chatroom.

This class contains positive and negative tests for methods which belongs to the class Chatroom.

**Attributes**

Private chatroom: Chatroom the initialized instance of the current user with its functionalities.

Private user : User an instance of a user.

private url:string; holds the address of the server.

**Functionality**

public TestChatroomConstruction();tests the construction of Chatroom instance.

public TestRegistrationMethod();perform negative and positive tests for registration functionality.

public login\_and\_logout\_Tests(); tests the functionalities of logging in and logging out of the user.

Public retrieveMessagesFailingTest();negative test for the retrieve messages functionality, here we try to connect to a not exist server address;

Pubic retrieveMessagesTest();tests the ability to get new Imessages from the server.

Public sendMessageTest(); tests the functionality of sending message by the user.

Public sendMessageFailedTest(); negative test for sending message by the user.

Public deleteFile(); perform the precondition for the chatroom tests. This delete the users file from the test project.

**User\_tests class:**

The User\_tests class, tests the main functionalities of the class User.

This class contains positive and negative tests for methods which belongs to the class Chatroom.

**Attributes**

private url:string; holds the address of the server.

**Functionality**

public TestUserSend(); perform negative and positive tests for the functionality of send message by the user.

Message Stub class:

This class simulates the method in Messages class which return a message or null back to the send method of User class.