High Level Design

Terminology

Chat Room
A virtual environment in which users can post their messages and read the messages
written by other users.

User
A person with a group id, nickname and password who interacts with the system.

Registration
The act of recording user details into data base.

Login
The act of signing into the system by the user.

Message
The text which the user delivers. Message content is limited to 100 characters.

Communication model

Requests

Send message request
A send message request is initiated by the user, the request is sent to the database, which assinges the message with a unique ID (GUID) and the server's timestamp.

Retrieve messages request
A retrieve message request is asked every two seconds and it will ask the database for the new messages that has been sent, it will not retrieve messages twice and no more than 200 messages

Edit message request
An Edit message request is initiated by the user, the edit can be dan only for messages from the current user otherwise it will tell the user he can't edit messages that does not belongs to him

Actors

Users
A person connected to the chatroom using a client software, for sending and receiving messages. A user is identified by her group ID and a nickname that is unique to her group.

Milestone 1 Low Level Design

<u>classes</u>

logic Layer -**the logic layer will be incharge of managing all actions that the users can make and handling all the logical structure of the chatroom**

<u>Chat-room</u>

this class will get the data from the presentation layer make it ordered, validate it and sending it to the persistent layer for saving and restoring

<u>Atribbutes</u>:

(-) logger //will be in charge of logging the data to a txt file
(-) MessageHandler message_handler
(-) UserHandler user_handler
(-) Log log
(-) List<IUsers> users
(-) List<Message> messages
(-) User currentUser
(-) const string url
(-)const string INVALID_NICKNAME
(-) const string INVALID_GROUPID
(-)   const string EMPTY_INPUT
(-) const string INVALID_LOGIN
(-) const string WRONG_PASSWORD
(-) const string ILLEGAL_LENGTH_MESSAGE

<u>Methods</u>:

(+) Start() - preparing the system to start runing
(+) List<IMessage> MessageManager(bool, String, String, String, String, Bool)
(+) String HashedPassword(String password)
(+)void Registration(int groupId, string nickname, string password)
(+) bool Login(int groupId, string nickname)
(+) List<String> MessageManager(bool ascending, string filter,string sort, string groupId,string nickName) - This returns an updated list of messages that are organized according to the user choice of sorting and filter
(-) bool IsValidNickname(int groupId, string nickname)
(-) bool IsValidGroupID(int groupId)
(-) bool CheckIfInputIsEmpty((string str)
(-) bool isValidPassword(string password)
(+) List<IMessage>LegalSizeOfMessage(List<IMessage> list) - this will get a list and return a list with no more then 200 messages

(+) bool CanEdit(IMessage lastmessage)
(+)void editMessage(string newMessage , Imessage lastMessage)
(+) Login()
(+)logOut()
(+) Send(string messageContent)
(+) List<IMessage> SortByNickname(List<IMessage>updatelist, Boolean ascending)
(+) List<IMessage> SortByIdNicknameTimestamp(List<IMessage> updatelist, Boolean ascending)
(+) List<IMessage> SortTimestamp(List<IMessage> updatelist,Boolean ascending)

USER
A class of a person who interacts with the system.
Attributes:
(-)int groupId
(-)string nickname
(-)string password
(-)string SALT
Methods:
(+)ToString()
Message
A class which describes an object which includes:
Attributes:
(-)Guid Id   //is determined by the server
(-)string nickname
(-)DateTime Date //is determined by the server
(-)string messageContent //limited to 100 characters
(-) int MAX_LENGTH = 100
(-)int groupId
methods
(+)CheckValidity(string content)
(+) String toString(String)
Hashing
A class which is in charge of encrypting the passwords

DataAccess-**this layer will connect between the DataBase and the Logic layer.**
Class: MessageHandler
This class will connect between the DataBase and the Logic layer in all that related to the messages part.
fields:
(-) log log

(-) string sql_query;
(-) string server_address;
(-) string database_name;
(-) string user_name;
(-) string password;
(-) string connetion_string;
(-) SqlConnection connection;
(-) SqlCommand command;
(-) SqlDataReader data_reader;
(-) IList<IQueryAction> filters;
(-) DateTime lastDate;
(-) const int MAX_MESSAGES = 200;

<u>**methods:**</u>
(+)List<IMessage> RetrieveMessages(boolean isStart)// this function returns the new messages that were not withdrawn and if there are filters , return the messages filtered.
(+)void ClearFilters()
(+)void AddGroupFilter(int groupId)
(+)void AddNicknameFilter(string nickname)
(+)int GetUserID(Imessage message) : int // return the user id of the sender of the message
(+)void insertNewMessage(Imessage message) //insert the new message to the database
(+) void EditByGuid(String newmessagecontent,String messageguid) // this function gets a message guid and new message content and edit the message with this guid.

<u>**interface**</u>**: IQueryAction**
<u>**methods:**</u>
(-)execute(string query):String // build an sql statement

<u>**ClassName:**</u> **GroupFilter: IQueryAction**
this class implement IQueryAction. this class build an sql statement of filter msg by id.
<u>**Field:**</u>
(-)int id
<u>**Method:**</u>
(+) String execute(String query)

<u>**ClassName: NicknameFilter :IQueryAction**</u>
this class implement IQueryAction. this class build an sql statement of filter msg by nickname.

Field:
(-)string nickname
Method:
(+) String execute(String query)

UserHandler
This class will connect between the DataBase and the Logic layer in all that related to the Users part
Fields:
(-) log log;
(-) bool isStart;
(-) string sql_query;
(-) string server_address;
(-) string database_name;
(-) string user_name;
(-) string password;
(-) string connetion_string;
(-) SqlConnection connection;
(-) SqlCommand command;
(-) SqlDataReader data_reader;
(-) static string SALT = "1337";

methods:
(+) saveToFile(IUser user)
(+) List< IUser > restoreUsers()
(+)List<IUser > retreiveUsers()
(+) void InsertNewUsers(IUser user)
(+) IUser retreiveUser(int groupid,String nickname, String password)
(+) IUser createUserInstance(SQLDataReader datareader)
(+) bool isValidNickname(String groupid, String nickname)
(+) bool isValidPassword(String groupid, String nickname,String password)

Presentation Layer - **will be in charge of all the presentation part to the user and will manage the communication between the user and the logic layer. it will contain all the windows that the users can see**

MainWindow

- this class will be the first class that the client can see from here they will be able to move for the other window and enter as users
<u>Attributes</u>:
(-) logger //will be in charge of logging the data to a txt file
(-)ChatRoom chatRoom
<u>methods:</u>
(-) void Register_Click(Object sender, RoutedEventArgs e)
// Open the registration window //display the client the information of the registration process and give the logic layer the details that the client gave the system for handling
(-) void Login_Click(Object sender, RoutedEventArgs e)-
//Open the Login Window//display the client the information of the login process and give the logic layer the details that the client gave the system for handling.
(-) void Exit_Click(Object sender, RoutedEventArgs e)-Close the program if asked.

<u>Login Window</u>
- this class will ask the client for the login details and send them for the logic layer for validation
<u>Attributes</u>:
(-) logger //will be in charge of logging the data to a txt file
(-)ChatRoom chatRoom
(-) String hashpassword
(+) bool validation
(+) String INVALIDPASSWORD
(-) ObservableOBjectChatRoom _main
<u>methods:</u>
(-) Login - If there are no problems, open the ChatRoom window
(-) void Login_Click(Object sender, RoutedEventArgs e)
(-) void passwordBox_PasswordChanged(Object sender, RoutedEventArgs e)
(-)BackToMain(Object sender, RoutedEventArgs e)- Open the main window and close this one.

<u>Class: Chat Room Window</u>
- this class will be the main chat room here the user can see the messages, write new ones, and organize the messages according to his wish.
<u>Attributes</u>:
(-) logger //will be in charge of logging the data to a txt file
(-)ChatRoom chatRoom
(-)string nickName
(-) string groupId
(-) string currChose
(-) string sort

(-)string filter
(-)bool ascending
(-) DispatcherTimer dispatcherTimer
(-) bool isPressed
(-) ObservableOBjectChatRoom _main

<u>methods:</u>
(-)inisializeFilterandSorter() - inisialize the choose boxeses that will appear on the application with the sorting and filter options
(-) dispatcherTimer_Tick() - this is a timer that responsible for updating the new messages from the server every two seconds
(-)ButtonClickLogOut - inform the logic layer that a user asked to log out and move the user to the main manu.
(-) ButtonClickSend -sends the chatroom the send request and informs the user in case of an invalid message.
(-)ButtonClickFAS -filter and sort - this function occur when the user press the filter and sort buttons and sends the chat room the new order and filter that the user now wants to use.
(-) void RadioButton_checked_name(Object sender, RoutedEventArgs e)
(-) void RadioButton_checked_time(Object sender, RoutedEventArgs e)
(-) void RadioButton_checked_allSort(Object sender, RoutedEventArgs e)
(-) void ListBox_MouseDoubleClicked(Object sender, RoutedEventArgs e)

(-) RadioButton_checked_sorting options() - Three functions - save the user choice of sorting in the class fields until the user want to start the sort .
(-) filterOptions_SelectionChanged() - this will get the users current filtering choice and display him the place to enter the filter details.

ClassName: EditMessage
this class will allow the user to edit messages that he sent in the condition that he is the one who sent the message.
<u>Attributes:</u>
(-) ChatRoom chat;
(-) IMessage lastMessage;
(-) Log log
(-) ObservableOBjectChatRoom _main
<u>Methods:</u>
(+)EditMessage(ChatRoom chat , IMessage lastMessage , ObservableObjectChatRoom _main)
(-) void btnDialogOk_Click(object sender, RoutedEventArgs e)
(-) void btnDialogCENcel_Click(object sender, RoutedEventArgs e)