Chat Room CLI Client LLD

# Terminology

# **Chat Room**

# A virtual environment in which users can post their messages and read the messageswritten by other users.

# **User**

# A person who interacts with the system.

# **Nickname**

# A familiar or humorous name the user uses to identify himself.

# **Registration**

# The act of recording user details.

# **Login**

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

# **Message**

# The text which the user delivers. Message content is limited to 150 characters.

# **Message Frame**

# A written communication sent between the users of the system.

# Buissness Layer

#### Functionality

Performs all logic functions..

## **ChatRoom class**

#### Functionality

Main client class. Maintains and operates all functionality methods.

#### Fields

* loggedInUser : User – current logged in user.
* request : Request – active request class.

#### Methods

* ChatRoom()
  + Initiates all fields
  + Synchronizes users and messages with persistent data.
* login(int g\_id, string nickname) : bool
  + verifies valid user details.
  + Changes logged in user
  + Returns true if successful.
* logout(): void
  + Changes logged in user
* exit() : void
  + logs out.
  + Closes program.
* register(int g\_id , string nickname) : void
  + creates new user.
  + Saves user data to users and to persistent layer
* retrieve Messages(int number): void
  + gets specified number of last messages from sever.
  + Saves messages to messages and to persistent layer.
* send(string message)
  + request.send(message,loggedInUser)
  + saves IMessage to messages and persistent layer.
* displayNMessages(int num) : SortedSet<IMessages>
  + returns a sorted (by time) list of Messages of the last 'num' messages to were retrieved.
* retrieveUserMessages(int g\_id, string nickname) : SortedSet<Message>
  + returns a sorted (by time) list of Messages sent by specified user details.

## **User class**

#### Functionality

User class, contains user details. Serializable.

#### Fields

* nickname : string
* g\_id : int.

## **Request Class**

#### Functionality

In charge of all requests from communication layer. Makes sure not to flood server with more than 20 queries in 10 seconds.

#### Fields

* final MAX\_MESSAGE\_LENGTH : int
* final URL : string
* lastNRequests : Queue
  + contains DateTime values of last N\_ALLOWED queries sent.
  + Used to make sure not to overload server with more than N\_ALLOWED queries in N\_SECS
* Final N\_ALLOWED : int
* Final N\_SECS : int

#### Methods

* send(string msg, User user) : IMessage
  + validates msg.
  + makes a send request to comm' layer.
  + returns IMessage retrieved from comm' layer.
* retrieve Messages(int num) : List<IMessage>
  + makes a retrieve request to comm' layer.
  + returns List<IMessage> of num IMessages retrieved from comm' layer.
* isNotOverloading() : bool
  + returns true if sending another request will not overload the server.
  + If false is returned request shouldn’t be sent.
* private validateMessage(string msg) : bool
  + validates msg.
  + returns true if valid.

## **Message Class**

#### Functionality

Data structure for holding a message – implements IMessage.

#### Fields

* Id : Guid – GUID for message
* UserName : string – nickname of message sender
* Date : DateTime – time message was sent – granted by server
* MessageContent : string – message content
* GroupID : string – senders group id
* IntGroupID – int – senders group id int param.

#### Subclasses

* MessageGUIDComp
  + Implements Comparer<Message>.
  + compares messages by GUID.
* MessageDateComp
  + Implements Comparer<Message>.
  + compares messages by Date.
* MessageUserComp
  + Implements Comparer<Message>.
  + compares messages by sending User.
* MessageMultyComp
  + Implements Comparer<Message>.
  + compares messages by User and Date.

# Presentation layer – CLI

**MainWindow class**

#### Functionality

Displays the main window – the first window the user sees.

Enables the user to use the registration and login features.

Communicates with the Buissness layer for which action the program has to take.

#### Fields

* chtrm - ChatRoom
* A ChatRoom object in order to communicate with the Business layer.
* \_main: ObservableModelMainWindow – Binds and updates the GUI components data with the actual Chatroom updated data.

**ObservableModelMainWindow class**

#### Functionality

Binds and updates the GUI components data with the actual Chatroom updated data.

Communicates with the Business layer for login and registration.

#### Fields

* chtrm - ChatRoom
* The working ChatRoom.
* g\_IdBox – String
* Represents the group's ID number.
* nicknameBox – String
* Represents the users nickname number.
* bkImageLocation – ImageSource
* The Main Window's background image.

#### Methods

* g\_IDToIntAndVerify(String g\_ID): int
* Checks if the received string contains only the characters '0'-'9'.
* Converts the string variable into int and returns it.
* verifyNickname(String nickname): bool
* Checks if a string is not empty.
* Returns true if the string is not empty.

#### Implementing interfaces

* INotifyPropertyChanged

## **ChatRoomWindow class**

#### Functionality

Enables the user to use the ChatRoom features

#### Fields

* chtrm : ChatRoom - The working ChatRoom
* observer : ObservableModelChatroom – Binds and updates the GUI components data with the actual Chatroom updated data
* dispacherTimer : DispacherTimer – Iterates the application data updates. Mainly the messages visible to the user.
* mainWindow: MainWindow – The component that dispatched this ChatRoomWindow and will be displayed once logout occured

## **ObservableModelChatroom class**

#### Functionality

Binds and updates the GUI components data with the actual Chatroom updated data

#### Fields

* Mainly text fields and Boolean values representing sort and filter flags.

#### Implementing interfaces

* INotifyPropertyChanged

# Persistent Layer

#### Functionality

Maintains persistent data regarding the client in DBMS.

#### DataBase

Microsoft SQL server.

## **Log4net package**

#### Functionality

Creates and manages logging for the entire client

## **Handler<T> interface**

#### Functionality

Inserts and retrieves data from a given type T in a database. Handler is associated with a certain table in the DB.

#### Functions

* retrive(Dictionary<string, string> query): List<T>
  + Retrieves data from the DB according to the constraints in the query
  + Returns List<T> containing the query elements that respected the query specification
* insert(): T
  + Inserts T to the DB
* CreateSelectQuery(Dictionary<string, string> query)
* CreateInsertQuery(Dictionary<string, string> query)
  + Both are abstract methods that the extending classes implementing according to the table structure and with respect to the quesy

## **MassegeHandler<IMessage>: Handler<IMessage>**

#### Functionality

Implements Handler. Manages persistency for *IMessage*s.

## **UserHandler<IUser>: Hansler<IUser>**

#### Functionality

Implements Handler. Manages persistency for *IUser*.

# Public Interfaces

## **IMessage Interface**

#### Methods

* Guid Id { get; }
* string UserName { get; }
* DateTime Date { get; }
* string MessageContent { get; }
* string GroupID { get; }
* string ToString();

## **IUser Interface**

#### Methods

* int Id { get; }
* int G\_id { get; }
* string Nickname { get; }
* string HashedPassword { get; }