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.
* url : string – server url.
* messages : SortedSet<IMessage> - contains all sent / retrieved messages
  + Sorted by timestamp.
  + Synched with persistent layer messages.
* users : List<User> - contains all registered users.
  + Synched with persistent layer users.
* request : Request – active request class.
* userHandler : UserHandler – active userHandler class
* messagesHandler : MessagesHandler – active messageHandler 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

#### 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.
* messagesToDisplay: int
* Represents the amount of messages we'd like to display each time the user requests to display messages.
* messagesToRetrieve: int
* Represents the amount of messages we'd like to retrieve from the server.

#### Methods

* playInput(): void
* Manages the user's decisions in which action he'd like to preform (login, register, send, logout etc).
* displayMessageList(ICollection<Message> listToDisplay): void
* displays to the user all the elements (messages) in a list
* 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.

**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
* \_main: ObservableModelMainWindow – Binds and updates the GUI components data with the actual Chatroom updated data.
* messagesToDisplay: int
* Represents the amount of messages we'd like to display each time the user requests to display messages.
* messagesToRetrieve: int
* Represents the amount of messages we'd like to retrieve from the server.

#### Methods

* playInput(): void
* Manages the user's decisions in which action he'd like to preform (login, register, send, logout etc).
* displayMessageList(ICollection<Message> listToDisplay): void
* displays to the user all the elements (messages) in a list
* 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.

# Persistent Layer

#### Functionality

Maintains persistent data regarding the client in local files.

#### Files

Stored in a local folder. The path is static.

## **Log4net package**

#### Functionality

Creates and manages logging for the entire client

## **Handler<T> interface**

#### Functionality

Saves, edits and retrieves data from a given type in a database.

#### Functions

* save(T data): bool
  + Saves data in the files system
  + Returns true if the data was saved successfully
* edit(T data): void
  + Edits data in the files system
  + Returns true if the data was edited successfully
* retrieveAll(): List<T>
  + Retrieves all the data from type T in the files system

## **MassegeHandler<CommunicationMessage>**

#### Functionality

Implements Handler. Manages persistency for *CommunicationMessage*.

#### Fields

* path: final static string
  + The path to the local folder containing the persistent data

## **UserHandler<User>**

#### Functionality

Implements Handler. Manages persistency for *User*.

#### Fields

* path: final static string
  + The path to the local folder containing the persistent data

# Communication Layer

#### Functionality

Performs all communication with server. Given as an outsourced layer.

#### Methods

* Send(string url, string groupID, string nickName, string messageContent) : IMessage
* GetTenMessages(string url): List<IMessage>

## **IMessage Interface**

#### Methods

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

# MileStone\_2\_UnitTests

#### Functionality

Tests project designed to tests multiple functionalities of the chatroom project

## **ChatRoomTests class**

#### Functionality

Tests main functionality of ChatRoom.cs

#### Fields

* dirPath : string – path of persistent data directory
* cr : Chatroom – a chatroom instance to test
* cr2 : Chatroom – a 2nd chatroom instance to test
* user : User – a user instance to test
* user2 : User – a 2nd user instance to test.

#### Methods

* Initialize
  + performed before each test.
  + initializes all fields.
* loginTest\_before\_login\_LoggedInUser\_is\_null
  + verifies that before logging in the loggedinuser is null.
* loginTest\_not\_registered\_user\_login\_fails
  + verifies that an attempt to login an unregistered user fails.
* loginTest\_registered\_user\_login\_succeeds
  + verifies that an attempt to login a registered user succeeds.
* loginTest\_Multiple\_failed\_logins\_shouldnt\_affect\_current\_user
  + verifies that failed login attempts don’t affect current logged in user.
* loginTest\_null\_nickname\_login\_fails
  + verifies that un expected null nickname argument throws a proper exception.
* logoutTest\_successful\_logout\_after\_logout\_loggedinuser\_is\_null
  + verifies that logout after being logged in changes the current logged in user to null as its supposed to do.
* logoutTest\_failed\_logout\_logout\_without\_login\_first\_throws\_exception
  + verifies that logout without initially logging in throws a proper exception.
* registerTest\_successfull\_register
  + verifies registering valid user succeeds.
* registerTest\_register\_already\_registered\_user\_should\_fail
  + verifies that an attempt to register an already registered user throws an exception.
* registerTest\_registeredUser\_should\_be\_persistant
* registerTest\_null\_nickname\_should\_fail
  + verifies that un expected null nickname argument throws a proper exception.
* sendTest\_sentMessage\_should\_be\_persistant
  + verifies that a successful sent message is persistent.
* sendTest\_send\_without\_loggingin\_should\_throw\_exception
  + verifies that before logging in the send method throws a proper exception.
* sendTest\_send\_null\_should\_throw\_exception
  + verifies that an attempt to send null messages throws proper exception.
* sendTest\_message\_with\_over\_150\_chars\_should\_throw\_exception
  + verifies that an attempt to send a message with over 150 chars throws a proper exception
* sendTest\_empty\_msg\_send\_should\_do\_nothing
  + verifies that sending empty message does nothing.
* sendTest\_multypul\_messages\_should\_all\_be\_persistant
  + verifies that un multiple sent messages are all persistent.
* displayNMessagesTest\_with\_invalid\_input\_should\_throw\_exception
  + verifies that displayNMessages with negative integer throw a proper exception.
* displayNMessagesTest\_without\_login\_should\_throw\_exception
  + verifies that displayNMessages without initially logging in throws a proper exception.
* displayNMessagesTest\_should\_throw\_exception\_if\_there\_are\_no\_messages
  + verifies that displayNMessages throws a proper exception when there is nothing to display.
* displayNMessagesTest\_success\_test
  + verifies that an displayNMessages works properly when used properly.
* retrieveUserMessagesTest\_witout\_initailly\_login\_should\_throw\_exception
  + verifies that retrieveUserMessages throws a proper exception when used before initially logging in.
* retrieveUserMessagesTest\_for\_null\_nickname\_should\_throw\_exception
  + verifies that retrieveUserMessages throws proper exception when used on null nickname.
* retrieveUserMessagesTest\_for\_user\_that\_didnt\_send\_yet\_should\_throw\_exception
  + verifies that retrieveUserMessages throws an exception when trying to retrieve messages from a user that didn’t send messages yet.
* retrieveUserMessagesTest\_success\_test
  + verifies that retrieveUserMessages works properly when used properly.
* Cleanup
  + cleans up all persistent data so each test will be unaffected by other tests.
  + performed after each test.

## **UserTests class**

#### Functionality

Tests main functionality of ChatRoom.cs

#### Fields

* tempUser1 : User – user to be used in tests.
* tempUser2 : User – user to be used in tests.
* tempUser3 : User – user to be used in tests.
* tempUser4 : User – user to be used in tests.
* tempUser5 : User – user to be used in tests.

#### Methods

* EqualsTest\_with\_null\_returns\_false
  + verifies that Equals method with null argument returns false.
* EqualsTest\_with\_not\_User\_object\_returns\_false
  + verifies that Equals method with non-user argument returns false.
* EqualsTest\_equal\_user\_test\_succsess
  + verifies that Equals method with logically equal users return true.
* EqualsTest\_different\_username\_returns\_false
  + verifies that Equals method with diff username returns false.
* EqualsTest\_different\_groupID\_returns\_false
  + verifies that Equals method with diff group id returns false.
* EqualsTest\_different\_IDandNick\_returns\_false
  + verifies that Equals method with diff group id and username returns false.
* CompareToTest\_same\_user\_returns\_0
  + verifies that CompareTo method returns 0 on logically equal users.
* CompareToTest\_same\_groupID\_diff\_nick\_sorts\_by\_nick
  + verifies that CompareTo with same group id sorts by nickname.
* CompareToTest\_same\_nick\_diff\_groupID\_sorts\_by\_group
  + verifies that CompareTo with same nick different group sorts by group id.
* CompareToTest\_altogether\_diff\_users
  + verifies that CompareTo with different users altogether sorts properly
* CompareToTest\_null\_compare
  + verifies that CompareTo with null argument sorts null as bigger.
* CompareToTest\_notUser\_compare\_throws\_exception
  + verifies that CompareTo with non user argument throws a proper exception.