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.

#### Methods

* ChatRoom()
  + Initiates all fields
  + Synchronizes users and messages with persistent data.
* login(int g\_id, string nickname) : bool
  + verifys 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 persistant layer
* retrieveMessages(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.
* retrieveMessages(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.

# Presentation layer – CLI

**MainMenu class**

#### Functionality

Manages the I/O with the chat user.

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.

# 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>
  + Retrives 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

# Communiction Layer

#### Functionality

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

#### Methods

* Send(string url, string gourpID, 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();