# PROJECT : KORAKAAGAZ

## CONTENT MODULE

(UNDER INFRASTRUCTURE TEAM)

## DESIGN SPECIFICATIONS

- written by BADAL KUMAR (111701008)

## INTRODUCTION

In this project, there is a requirement of a chat window where any client can type in some messages and it should be displayed to all other active client's machine. Further, it is also required that the profile image of the client is displayed whoever is sending the message. This is where the role of content module arrives. In our application, the UI module will receive the input username of the user along with the profile image when any new user joins. This username and corresponding profile image will be passed to the content module within the application. Content module will ensure that the aforementioned user details reach to the content module of every active client by sending it to server through the networking module. Similarly, any message written in the chat window will be sent to content module by UI module and the content module transfers the message to server after doing some processing (converting to json format) via networking module. The server broadcasts it to all active client via networking module and content module of each client receives the formatted message by its networking module, decodes it and sends it to UI module of respective client. Then, UI module displays the message in the chat window.

# INTERFACE

Our module is going to cooperate with the UI module and networking module. The interface provided by networking module to us are

```
public interface INotificationHandler {
    void onMessageReceived (String message);
}
```

and

```
public interface ICommunicator {
    void start ();
    void stop ();
    void send (String destination, String message, String identifier);
    void subscribeForNotifications (String identifier,
INotificationHandler handler);
}
```

The interface that content module is providing to UI module are

```
package infrastructure.content;

public interface IContentCommunicator {
	/*
	 * UI will call this method with one json formatted string argument
	 * @param : userDetails - It is json string which will contain server
ipAddress, username, image
	 */
	void initialiseUser(String userDetails);
	/*
	 * UI will call this to paas the message of client to everyone else
	 * @param : message - it is the actual message as a json string, more fields
can be accomodated in future
	 */
	void sendMessageToContent(String message);
	/*
	 * UI will call this to notify that the current user is quitting the session
to all other users
	 */
	void notifyUserExit();
	/*
	 * UI will subscribe to us in order to receive any updates coming from other
users
	 * @param : identifier - This will be a string and unique too
	 * @param : handler - By using this handler, methods of
IContentNotificationHandler will be called
	 */
	void subscribeForNotifications(String identifier, IContentNotificationHandler
handler);

}
```

and

```
package infrastructure.content;

public interface IContentNotificationHandler {
    void onNewUserJoined (String username);
    void onMessageReceived(String messageDetails);
    void onUserExit (String username);
}
```

Our module is going to implement the *onMessageReceived* method in the *INotificationHandler* and all the methods mentioned in the *contentICommunicator* interface. All the methods mentioned in *contentNotificationHandler* will be implemented by UI module.

Since we need the port value of board server, we have an interface with processing module too which is given below.

```
package infrastructure.content;

public interface IServerPort {
	/*
	 * the processing module calls this method to send port of board server
to each new client
	 * @param : port, which is the port of board server in int datatype
	 */
	void sendPort(int port);

}
```

The final contentFactory class will be as follows.

```
package infrastructure.content;

public final class ContentFactory {

	private static IContentCommunicator instance1;
	private static IServerPort instance2;

	private ContentFactory() {}

	public static IContentCommunicator getContentCommunicator() {
		if (instance1 == null) {
			instance1 = new ContentCommunicator();
		}
		return instance1;
	}
	public static IServerPort getServerPort() {
		if (instance2 == null) {
			instance2 = new ServerPort();
		}
		return instance2;
	}
}
```

# DIVISION OF WORK

This module consists of two team members, Badal Kumar (111701008) who is me and Talha Yaseen (111701019). Based on suggestions from team lead Rahul R, I will handle all the things that UI module sends to content module, do some processing and then forward those to networking module whereas Talha will handle anything coming from networking module, does some processing and pass it to the UI module. So, this design specification document along with Talha's design specification document will describe the whole functionality of the content module.

# My Work Agenda

I will define a class named *contentCommunicator* which will implement the *contentICommunicator* interface where all the methods will be defined.

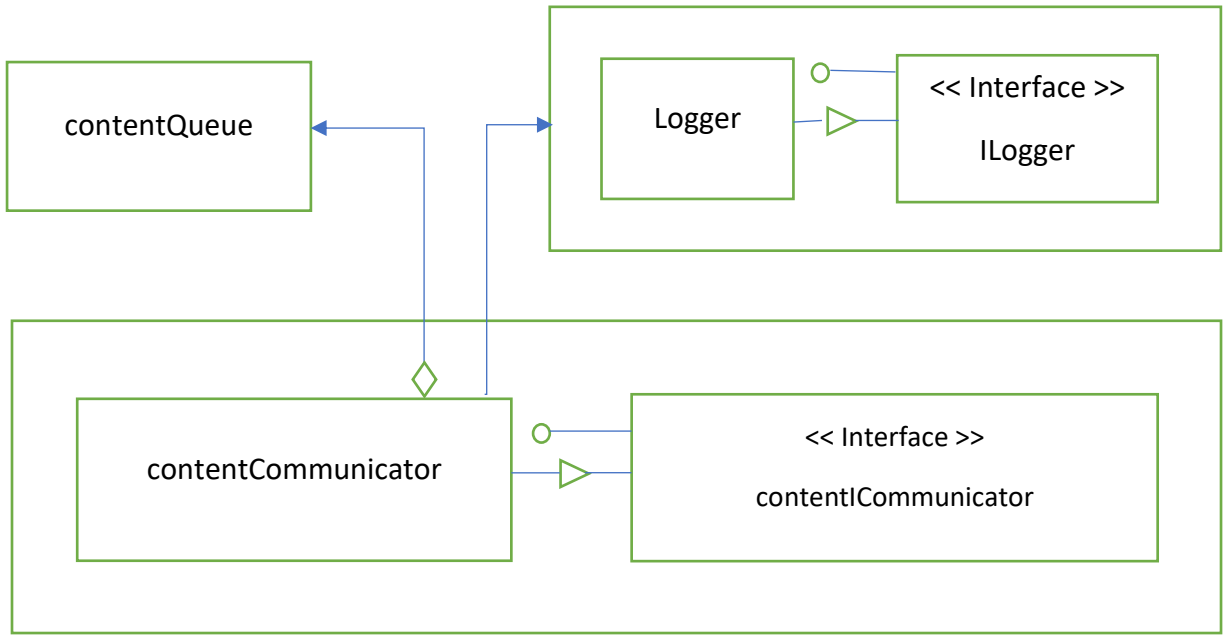
I would initialise one hashmap for storing *username* and *image*. The UI module will call the method *initialiseUser* when any new user starts the application. By this, I will get the *username* of the user and profile *image* of the user alongwith the board server's ip_address. The hashmap mentioned above will be filled with proper data and then I will call the *send* method provided by *ICommunicator* interface with arguments "server's ip_address" as destination, "content" as identifier and a json formatted string with fields "newuser", "username" and "profile_image of new user" as message. Note that *message*, *identifier* and *destination* are required arguments for the *send* method. I will also remember the username of the client locally in a variable for future needs. Further, it should also be noted that this *initialiseUser* will be called only once. After the server receives these messages, it will update its own map of username and images and broadcast to all other client and the map will be received by every other client. Refer to Talha's spec for more details on what happens next.

When the UI module calls the *notifyUserExit* method, then all the hashmaps and queues on that client which has been initialised by content module will be cleared off. I will call the *send* function with *destination* as "*server's ip_address along with port*", *identifier* as "*content*" and json formatted string which will contain fields *"userExit" and "username"* as *message*. The server will then receive the information, delete the user_details of that user from its map and then broadcasts it to all other clients. All clients will then delete the corresponding details of that user and a corresponding message will be displayed in the chat window. Refer Talha's spec for the later part.

For implementing the *sendMessageToContent* method, we need queues. I will create a new class *contentQueue* and define the various methods necessary for functioning of the queue. Then I will create an object of queue class in *contentCommunicator* class. Talha will also need a different queue. So, he will also create an object of queue class by different name. When UI module calls the *sendMessageToContent* method, I will receive the message from the argument. This will be executed in one thread. Then, I will convert it into json formatted string with fields *"message", "time" and "actual_message_passed"*. Then I will put this json formatted string in the queue object. From the queue, a separate thread will be calling the *send* method of the *ICommunicator* interface with arguments destination as "server's ip_address", identifier as "content" and message as "dequeue_output_of_queue". Further, this message will be broadcasted to all clients after the server receives the message through networking module. Refer to Talha's spec to know what happens next.

I will be using the logger class too. At various stages, I will need to log different kind of informations. I will use the logError, logWarning, logSuccess and logInfo methods provided by the Logger class to log relevant kind of informations and finally these will be displayed on the console developed by UI module.

## CLASS Diagram




## Some other Design Decisions

- The UI module has agreed to convert the profile image into some encoded string and then use that as an argument in the initialiseUser method. They will also decode the string formatted image back into image and display it on chat window when displaying messages. They will be given encoded image from arguments of onMessageReceived method in the contentNotificationHandler interface.
- The encoded profile image will be stored on each client for each active user. The reason being that it reduces the overhead of sending encoded image each time with the message over the networking module. This was finalised after discussions with architect and team-lead.
- In the discussions with architect, he mentioned that server is going to be implemented by processing module but I'm unsure on how the content handling on the server side is going to be done. I will eagerly contribute with the content handling part if needed.