Implementing a GUI-Based Networked Communication System

Adnan T
Computer Science Student
University of Greenwich
-snip--
-snip--

Daniel B
Computer Science Student
University of Greenwich
-snip--
-snip--

Gabriel N
Computer Science Student
University of Greenwich
-snip--
-snip--

William P
Computer Science Student
University of Greenwich
-snip--
--snip--

COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College, Greenwich
London, SE10 9LS, United Kingdom

Abstract - Throughout the following paper, the development of a client-server network communication system will be analysed, including the comment of implementation techniques to most efficiently handle both group and direct messaging between users. The software redundancy and fault tolerance are reviewed to ensure software stability is sustained during unexpected events, in conjunction with examining the stringent testing carried out through computerised JUnit tests and manual end-user methods. After the successful undertaking of the project, during which work was conducted as part of a four-person team to create both a front-end graphical interface and backend engine, the scheme concluded with a final piece of working software featuring clean, commented and easily adaptable code.

Keywords - Communication, Reliability, Redundancy, Versatility

I. Introduction

This academic paper has been crafted based upon the IEEE styling guidelines to detail the procedure and techniques deployed when developing a network-based group communication platform. In addition to meeting the product owner's project specification, the software will be designed to be both functionally sound with built-in fail-safes and feature a somewhat aesthetically pleasing modern interface design, enabling the application to be both easy to use and easy on the eyes.

The task assigned to the team by the client's requirements, as part of the coursework activity, mandated a set of traits within the code that must be present for it to be deemed a success. In some areas, the wording or description stated are unclear or are not accurately defined; in such instances, executive decisions have been made on behalf of the client to allow for a final piece of software to be produced. As a minimum, a client interface must exist that enables the connection to a server (available locally or remotely) by specifying the IP address and port. After a connection has been established, should the user be the first to connect, they are allocated to act as a coordinator for all subsequently connected clients. Any user connected to the server is deemed part of a 'group' and should be able to communicate with any other user in the list; due to the lack of clarity surrounding this feature, the decision was made by the dev team to integrate both a single global group chat, in addition to direct private messaging between other users in the group. It is expected that the server will act as a relay for messages and instructions between the users and that the only processing it will carry out is the reassignment of a coordinator should the existing be disconnected for any reason. With this exception, the coordinator should perform all processing for group management, with each client storing a local copy of the active client list.

Contained within this report is a detailed overview of the design idiosyncrasies and implementation characteristics that our team members have deployed throughout the project, including a critical analysis of the development process and the technologies chosen. Such an overview will also include an evaluation of how, as a group, we utilised numerous developer tools and platforms to ensure that we worked together succinctly, kept the project on time and worked together in tandem with other projects being carried out.

II. Design / Implementation

The implementation of the core requisites stated in the project's technical specification has been meticulously planned and formulated into a production-ready platform, and the core components of which will be traversed in detail forthwith.

As a team of developers working together on a particular project, communication and proactive messaging are crucial components for completing a piece of software to any degree of success. To initiate the project, the group spent the afternoon planning out the project's scope and delegating specific jobs to each member to allow for work to be carried out simultaneously. At this stage, due to the diminutive size of the team, it is also vital that each member recognises the roles of their teammates and the overall project requirements so that, in the event of a disruption, developers can be flexible to assume additional responsibilities as warranted. Throughout the project and a group chat channel and weekly virtual stand-up meetings, jobs and issues were tracked using the GitHub platform to validate the project was on-time and that there were no impediments to the progress of the application. The concluded structure of the project can be viewed within the UML diagram attached to this submission in Appendix B.

It was decided that Java 8 would be used as the application's environment and that the software would be thoroughly tested to be compatible when executed on Windows 10 and Windows 10 Pro at build version 2004 or higher. Each collaborator opted to use their preferred IDE, which included Visual Studio, NetBeans and IntelliJ IDEA.

Firstly, the integration and justification for the technological methods used in creating our server-sided component will be elucidated; it was decided that the server and client should be incorporated into the same application and be simply accessible via a different menu tree option to enable users to set up their group should they not have a server to join. It was understood that when the software operated in server mode, the only processing that this element would perform is selecting a new coordinator if the first user connected or an existing coordinator disconnected; with this exception, it would act as an instruction forwarder.

Reviewing the most significant part of the project, the networked communication, is where the team focused on first, and it was determined that the use of java sockets (a technology to enable two-way communication bound by a port across the TCP layer between applications) would be the selected transfer medium for data. Embedded into both the ClientManager and ServerManager class to enable elements to connect, the technology was relatively simple to integrate and leverage. Overlooking the fact that Java sockets are commonly used and widely accepted as the long-term de facto standard for Java software communication, the package's popularity brings along with it a plethora of advantages. Such benefits include substantial development time designated to just this one component, enabling it to undergo considerably more testing than possible for a team our size in the allocated time and increasing overall stability by reducing the likelihood of network errors across different platforms and doing so as efficiently as possible. Web sockets could be used in such a situation, although they do not provide the needed functionality for the software to be completed in a state that would comply with the product owner's specification.

After concluding our research into the appropriate technology that our team would utilise for data delivery, the attention turned to the data itself. Per the specification, there were two types of data being passed throughout the network that were identified: plain text messages and instructions. During the software's planning phase, it was identified that an abstract factory pattern could be implemented to handle the variations of data being transferred that would inherit similar properties and methods. During the development stage, the software's modularity, simplicity, and reusability were prioritised over the design pattern being show-horned in, resulting in the pattern being dropped from the application. Executable instructions are encoded into messages in the Message class for sending before being detected, decoded, and executed by the recipient through a ServerChannel object; this allows for an extensive range of different commands to be passed without the need for alterations to the class structure. Inheritance from a Transmittable object has also been implemented, making future expandability possible to integrate a transmit protocol pattern.

The instructions packaged up into the Message class are undeniably the core to the application's functional operation, so particular care was spent during their design and planning stage to ensure that they remained modular, flexible, and easily able to adapt to changing environments if needed. The set of instructions are standardised throughout the platform, allowing for the client to both send and receive instructions to other users throughout the network; primarily used for communication between users and the coordinator, the commands cover everything from instructing clients to update their list, adding users to the list or removing users from the group. The server also has the power to delegate a new coordinator should an existing one drop, and the instruction set allows for any user to assume the role and act with correct functionality. Due to these commands' scope, attention to detail was spent documenting new commands making new functionality both quick and straightforward to complete. Such modularity also makes testing significantly more manageable and efficient.

The server-side processing for request handling and forwarding of commands it would receive was initially designed to be executed within a thread, utilising a Singleton design to share values between subclasses to the top-level processing function. The limitation in Java to pass values by reference was the motivating factor for this design decision, but it was noted that this pattern is not looked upon fondly by developers and is uncommonly used in production systems due to maintenance difficulty. This pattern was removed when the aforementioned limitations in maintainability immediately showed themselves during the debugging of a fault detected, resulting in data collisions between threads. A subsequent solution by nesting the instruction handler class was completed; by leveraging the Java programming language's built-in functionality, although passing by reference is not available, the transmission and access of variables of a top-level class in the nested class resolved all outstanding problems experienced.

The client-side implementation for message and instruction processing is handled similarly to that of the server-side, in a FIFO (first-in, first-out) based arrangement. This type of application requires messages to be delivered and specified in the order they were received; such execution makes the most logical sense to ensure that instructions do not get out of sync - stored in an ArrayDeque allows for this to work as expected. The client that is assigned to act as the group coordinator is responsible for the execution of the majority of the command sending, as while users will send a notification when they join, it is the responsibility of the admin to keep all users updated by informing them to update their locally cached copies to keep track of new or disconnected users. The only instruction received from the server remains the trigger for a client to be escalated from a user to the coordinator and begin processing group-related commands.

Reaching the topic of user management and user lists. The coordinator's job is to handle the instruction to trigger clients connected to the system to update their list should a new user join or disconnect unexpectedly. The user list must be both effortlessly maintainable and modular to allow for simple integration with other classes throughout the program. To achieve this, the architecture of the software was adapted around the HashMap storage medium; the locally cached copy of the list was stored in this style as not only did it introduce the ability for constant-time access, but also vastly improved code readability - not to mention the logical sense it makes as each client ID is unique and can be mapped. An alternative solution noted was to cache the list as an array, although the requirement to constantly query for index values was deemed unnecessary due to the additional processing overhead.

Compared to the rest of the project, message sending can be perceived as one of the simplistic aspects of this project. Due to the lack of specification clarity on group-based communication, it was interpreted as both requiring a group channel shared by all users, and for members connected to the group to be able to message each other directly. Implemented within the message object developed previously to handle instruction transfer; content can be packaged up with the respective properties set, allowing the server to forward a direct message straight to the recipient or a group message to all active members.

All of the foregoing features have been built into the software alongside a plethora of redundancy, validation, and verification measures to ensure that the software's stability is unwavering regardless of any unexpected inputs or network related matters. Try-catch statements, input validations, and checks have been distributed throughout the system to ensure that the application in its current state can function, but should modules be re-used on other projects, the core functionality will remain the same. By utilising socket technologies, unexpected disconnects can be handled - informing the coordinator of the network infraction, or if the coordinator is the one that has disconnected, prompting the server to assign a new administrator to prevent the system from crashing or causing inconvenience to the users by leveraging fault tolerance measures.

The group decided that it would be best to build a GUI application rather than one based solely on a command-line interface. The choice resulted in a steep learning curve for the team, having never utilised Java before, and the assistance of both AWT and SWING elements to build up both a modern and usable interface. Various elements within the interface required override methods to modify their settings or attach events to allow them to interact with the user to perform an array of the requested functionality. An example of where inheritance and overrides were explicitly used was creating rounded text boxes - a small detail requested by the interface designer - that lead to a considerable amount of research being needed on the topic. The interface also played an important role in displaying the content, responses and notifications provided by the backend in response to receiving instructions or message content from users.

We handled the unit testing using the JUnit 5 framework, which allows for automated testing of all the project's main components. Integration testing has been primarily handled through the use of manual exploratory checks. All documented tests are available in Appendix A. When designing unit tests, the main concern was that a test would often be considered an integration test rather than a unit test. Most unit tests often required the initialisation of core components due to this networking design's interconnected nature. Most of the main functionalities in the coursework specifications are related to the interaction between client and server. This meant that testing each element individually was challenging, as most functions required various other pieces of the software to work as intended. Even with the described challenges, the unit tests managed to ensure that the system's main components are working as intended, as shown by the Test Case Report found in Appendix A.

III. Analysis and Critical Discussion

Contained within this section of the report will be an in-depth analysis of the finalised software, including reviewing how the program functions as a whole, how redundancy has been built into the application and how the code has been modularised to improve versatility. Any observed weaknesses in implementing a feature will also be noted and will be accompanied by a brief statement as to potential resolutions to it in future revisions of the application.

At the inception of the project, the user interface was carefully designed using a graphics design tool called Figma. The initial designs consisted of seven pages which, after approval from each group member, we would continue to implement into the program using Swing, a GUI widget toolkit for Java applications.

Swing was a clear choice for implementing the user interface; this is primarily due to a limited number of Java GUI toolkits that can create a user-friendly experience to the same standard without compromising our design features. We decided to prioritise Swing over AWT (Abstract Window Toolkit) due to its more versatile components and more up to date toolkit, although some AWT elements were still utilised. Although Swing offered an extensive range of components to build the design, some elements had to be adapted to suit our design requirements. The main two elements that needed to be reworked were the JTextFields and the JButtons; instead of the plain text fields included with the toolkit, we opted for fields with rounded edges, a complete overhaul of the initial design. We used a separate class using the AWT graphics component to draw the text field to achieve this. The JButtons was as simple as using a background image created in Photoshop to transform the design to fit our needs.

The interface consists of multiple JFrames bound together to create separate pages. This method could be improved in future releases by switching between pages in a single frame resulting in a more fluid transition - the transfer of variable data between frames could also have been made a little more succinct in the documentation. Our desired layout for each frame was achieved using JPanels, a container that can store groups of components to build the design. The most predominant frame on the GUI is the client messaging page; JTabbedPanes are utilised to separate group messaging from direct messaging. In addition to the tabs, JScrollPanes are used to enable the user to scroll through messages within the message tabs. Long usernames are currently not supported, but future alterations to the interface would also enable this.

The compilation and execution of the project code allow for the software to be launched successfully. The socket technology utilised throughout the code, enabling clients to establish a connection to the server, is considered a great implementation. It leverages industry standards in communication and doesn't require duplicate code to duplicate existing functionality. The fault tolerance built into this means that should unexpected events transpire, or if the user fails to configure the application correctly, the software will remain stable and handle any such exception gracefully by informing the user of such occurrence. While this tech is extremely powerful and adaptable, data sent is raw, requiring additional processing to interpret the commands and potential security limitations perceived as a cause for concern should this system be deployed to the public without further research.

The code's design, including inheritance, overrides and design patterns, contributed to our focus to ensure that the project was modular and versatile. Throughout the project, there were various instances that design patterns could be seen as a possibility to be implemented; an example of this would be within the Message class where an Abstract Factory Pattern could have been added. This was thoroughly reviewed at the time, and although this would still be a good idea to implement in future revisions, it was decided against to ensure that the processing required instruction sending could be made as simple as possible. Both an observer pattern and transmit protocol pattern have been noticed as feasible with the server class, interacting with the instruction handler. The limiting aspect of this implementation is that it would be more challenging to interact with the processing, which is vital for modularity. Although the application would function as resiliently as possible, these patterns were decided against in various instances in favour of clean code.

The system's main components that required extensive testing were the ServerChannel and ClientManager, as those components are the home of the main requirements of the coursework specification, as ServerChannel handles the server connections to each client and the maintenance of the coordinator status. In contrast, ClientManager takes care of sending messages between a User and the server. The other software components were also properly tested, such as the ClientInstruction class, which handles formatting the user's instructions to the server and the user interface components, with both automated JUnit tests and exploratory tests.

The main weaknesses of the testing approach are the interdependence of most components, as highlighted earlier. Developing individualised testing for absolutely every functionality proved impossible - unit tests should only be for publicly accessible function calls. We followed an approach that it is better to test things, even if not following a "pure" unit test approach, by doing functionality tests observed in a mix of unit and integration tests being present in Appendix A. However, even when considering the sacrifices made to test those components, we believe that it was the correct approach to be taken regarding testing for this project, as it allowed for more robust test cases instead of asserting things that would not necessarily recreate a real use-case for the program.

IV. Conclusions

As a conclusion to both this report and this project as a whole, this section will briefly reflect upon the actions taken throughout the assignment and how such an undertaking has been completed.

The final deliverable that has been produced in response to the specification laid out by the product owner is comprised of code that is deemed to be of excellent quality; a claim substantiated through the evidence of clear and constructive commenting, extensive software modularity, flexibility, and reusability in addition to the various measures implemented to provide superior fault tolerance that has been built into each component. The specification has been followed closely, and where additional clarification has been needed, executive decisions have been made on how best to achieve the interpreted feature to the best of our abilities - such example is the wording of "group-based" that lead to the integration of both a single group chat channel as well as permitting direct messaging for any member connected to the server and deemed part of the 'group'. The technologies used and interface created considered to be modern and utilise the latest techniques for implementation.

As identified in the critical analysis, nothing is perfect, and the codebase to support this submission is no different. It has been identified that the team members assigned to integrate the core functionality potentially let real-world development experience influence decisions, resulting in code cleanliness and reusability to be prioritised over to the coursework requirement that mandated code be forced into design patterns. The program's message and instruction section are an excellent example of this; the code for these components is unambiguous and easy to understand through transmittable use, making it easily expandable when new functionality is required. Abstraction and a considerable number of overrides and inheritance are used throughout the project to make up for what could be seen as a lower frequency of patterns.

Upon reflection, the code for these sections could conform to the abstract factory pattern with some alteration and the server manager as a whole potentially being altered to match an observer pattern. Although these patterns could be applied, given the opportunity to repeat the process, it is not evident if they *should* be applied - patterns are great for providing structure where a solution cannot be envisaged or documented but could potentially lead to unnecessary complexity added. This decision would have to be reserved until such time and would be based on current technologies and exact specification requirements. A design pattern initially implemented into the program and later removed, a Singleton, is unanimously agreed by group members that it should be left out as it should rarely make its way into a production system. The improvement in code maintainability is considered to be more valuable than it being left in solely to satisfy the requirement of utilising it as a pattern.

Overall, it was a gratifying and informative project that allowed all our members to understand the Java language better and enabled the learning of new technologies to be conducted that will inevitably benefit our future careers as software engineers. The deliverable matches the spec and is of high quality, and with the criticisms from the analysis aside, regarded as a success.

Professional Acknowledgements

Our team are deeply grateful to both Dr. Muhammad Taimoor Khan and Dr. Markus Wolf for supporting the group's learning over the past academic term, both through the taught content delivered in university lectures but also by providing us with an interesting challenge through the form of this coursework that was fun to complete and enabled the ability for continued learning to take place using a language not previously covered.

We would also like to extend our thanks to Naureen K for the understanding and cooperation concerning the moderation and review of our group communication, which we conducted via an external platform - Discord - rather than the designated Teams platform offered for use by the university. The flexibility afforded to us was tremendously appreciated and enabled our team to communicate throughout the project more efficiently.

References

DevMedia, (2013). *JUnit Tutorial.* [Online] Available at: https://bit.ly/3tSdOio [Accessed 26 February 2021].

Joy, H., (2011). *Stack Overflow.* [Online] Available at: https://bit.ly/2NTsHlj [Accessed 1 February 2021].

Mattoo, P., (2012). *Stack Overflow.* [Online] Available at: https://bit.ly/2NTsIpn [Accessed 1 March 2021].

Michael, (2011). *Stack Overflow.* [Online] Available at: https://bit.ly/3w2Q50G [Accessed 3 February 2021].

mKorbel, (2013). *Stack Overflow.* [Online] Available at: https://bit.ly/31gVdAt [Accessed 3 February 2021].

NetBeans, (2017). *Writing JUnit Tests in NetBeans IDE.* [Online] Available at: https://bit.ly/2OZIShD [Accessed 4 March 2021].

Paul, H., (2008). *Stack Overflow.* [Online] Available at: https://bit.ly/3f9I2JI [Accessed 6 March 2021].

Peters, M., (2011). *Stack Overflow.* [Online] Available at: https://bit.ly/3ckS2hd [Accessed 4 March 2021].

TiyebM, (2016). *Stack Overflow.* [Online] Available at: https://bit.ly/3lYOHaX [Accessed 3 February 2021].

Javadoc.Io. (2021). *Mockito - Mockito-Core 3.8.0 Javadoc.* [Online] Available at: https://bit.ly/3u09D4i [Accessed 25 March 2021]

Appendix

Appendix A - Testing

A.1 - JUnit Automated Tests (Unit & Integration)

No.	Test	Input	Expected Output	Actual Output	Status	Comment
			MessageTest.Java			
1	testToString() (Direct Message)	"ClientA", "ClientB", "Example Message"	"ClientA::ClientB::Example Message::TIMESTAMP::false"	"ClientA::ClientB::Example Message::TIMESTAMP::false"	Pass	As Expected
2	testToString() (Group Message)	"ClientA", "Group Chat", "Example Message"	"ClientA::Group Chat::Example Message::TIMESTAMP::true"	"ClientA::Group Chat::Example Message::TIMESTAMP::true"	Pass	As Expected
3	testToString() (Direct Message)	"ClientA::ClientB::Example Message::TIMESTAMP::false"	Object Matches "ClientA", "ClientB", "Example Message"	Object Matches "ClientA", "ClientB", "Example Message"	Pass	As Expected
4	testToString() (Group Message)	"ClientA::Group Chat::Example Message::TIMESTAMP::true"	Object Matches "Group Chat", "ClientB", "Example Message"	Object Matches "ClientA", " Group Chat ", "Example Message"	Pass	As Expected
			ClientInfoTest.java			
5	testToString() (Working)	"TestID", "192.168.0.1", 11	TestID", "192.168.0.1", 11	TestID", "192.168.0.1", 11	Pass	As Expected
6	testToString() (Missing Value)	"TestID", "", 11	"TestID", "", 11	"TestID", "", 11	Pass	As Expected
			ClientInstructionTest.java			
7	testConvertInstructionToString()	2<SEPERATOR>BECOME COORD INATOR	2<SEPERATOR>BECOME COORDIN ATOR	2<SEPERATOR>BECOME COORD INATOR	Pass	As Expected
8	testCreateSendMessageInstructionString()	1<SEPERATOR>CLIENTB::Exampl e Message::false	1<SEPERATOR>CLIENTB::Example M essage::false	1<SEPERATOR>CLIENTB::Exampl e Message::false	Pass	As Expected
9	testCreateBecomeCoordinatorInstructionStrin g()	2<SEPERATOR>BECOME COORD INATOR	2<SEPERATOR>BECOME COORDIN ATOR	2<SEPERATOR>BECOME COORD INATOR	Pass	As Expected

10	testCreateRevokeCoordinatorInstructionString()	3<SEPERATOR>REVOKE COORDINATOR	3<SEPERATOR>REVOKE COORDINATOR	3<SEPERATOR>REVOKE COORDINATOR	Pass	As Expected
11	testCreateEstablishConnectionInstructionString()	4<SEPERATOR>CLIENTA	4<SEPERATOR>CLIENTA	4<SEPERATOR>CLIENTA	Pass	As Expected
12	testCreateReviewJoinRequestInstructionString()	5<SEPERATOR>TEMPID::CLIENTA::127.0.0.1::9091	5<SEPERATOR>TEMPID::CLIENTA::127.0.0.1::9091	5<SEPERATOR>TEMPID::CLIENTA::127.0.0.1::9091	Pass	As Expected
13	createRejectJoinRequestInstructionString()	6<SEPERATOR>TEMPID	6<SEPERATOR>TEMPID	6<SEPERATOR>TEMPID	Pass	As Expected
14	testCreateAcceptClientConnectionInstructionString()	7<SEPERATOR>TEMPID::CLIENTA	7<SEPERATOR>TEMPID::CLIENTA	7<SEPERATOR>TEMPID::CLIENTA	Pass	As Expected
15	testCreateUpdateClientInfosServerCacheInstructionString()	8<SEPERATOR>STRINGHERE	8<SEPERATOR>STRINGHERE	8<SEPERATOR>STRINGHERE	Pass	As Expected
16	testCreateAddClientInfoToLocalListInstructionString()	9<SEPERATOR>CLIENTA::127.0.0.1::9091	9<SEPERATOR>CLIENTA::127.0.0.1::9091	9<SEPERATOR>CLIENTA::127.0.0.1::9091	Pass	As Expected
17	testCreateNotifyClientDisconnectedInstructionString()	10<SEPERATOR>CLIENTA	10<SEPERATOR>CLIENTA	10<SEPERATOR>CLIENTA	Pass	As Expected
18	testCreateClientDisconnectedInstructionString()	11<SEPERATOR>CLIENTA	11<SEPERATOR>CLIENTA	11<SEPERATOR>CLIENTA	Pass	As Expected
19	testCreateGetUpdatedClientInfoListInstructionString()	12<SEPERATOR>CLIENTB	12<SEPERATOR>CLIENTB	12<SEPERATOR>CLIENTB	Pass	As Expected
20	testCreateClientAcceptedInstructionString()	13<SEPERATOR>COORDINATOR	13<SEPERATOR>COORDINATOR	13<SEPERATOR>COORDINATOR	Pass	As Expected
21	testCreateSetLocalClientInfoListString()	14<SEPERATOR>STRINGHERE	14<SEPERATOR>STRINGHERE	14<SEPERATOR>STRINGHERE	Pass	As Expected
22	testCreateConnectionRejectedByCoordinatorInstructionString()	15<SEPERATOR>Example Message	15<SEPERATOR>Example Message	15<SEPERATOR>Example Message	Pass	As Expected
23	testCreateNotifyOthersOfNewCoordinatorInstructionString()	16<SEPERATOR>COORDINATOR	16<SEPERATOR>COORDINATOR	16<SEPERATOR>COORDINATOR	Pass	As Expected
			ServerChannelTest.java			
24	testAddMessageToChannel()	"userA", "userB", "testmessage"	"userA", "userB", "testmessage"	"userA", "userB", "testmessage"	Pass	As Expected

25	testNullAddMessageToChannel()	Null	Null	Null	Pass	As Expected
26	testCheckClientConnectionExists()	False	False	False	Pass	As Expected
			ClientManagerTest.java			
27	testAddClientInfoToLocalList()	("ClientC", new ClientInfo("ClientC", new Socket(localIP, 9090)	Null	Null	Passed	As Expected
28	testGetAllClientIDsFromLocalList()	Null	Set<String> = [“Client C”]	Set<String> = [“Client C”]	Passed	As Expected
29	testGetAllClientsInfoFromLocalList()	result.get("ClientC").clientID	ClientC	ClientC	Passed	As Expected
30	testGetClientStatus()	null	False	False	Pass	As Expected
31	testGetAllClientsInfoFromLocalListAsFormattedString()	37	37	37	Pass	As Expected

A.2 - Automated Test Results

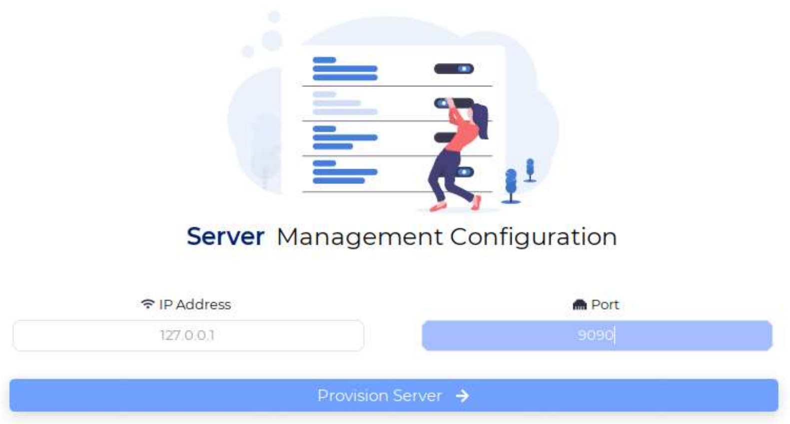
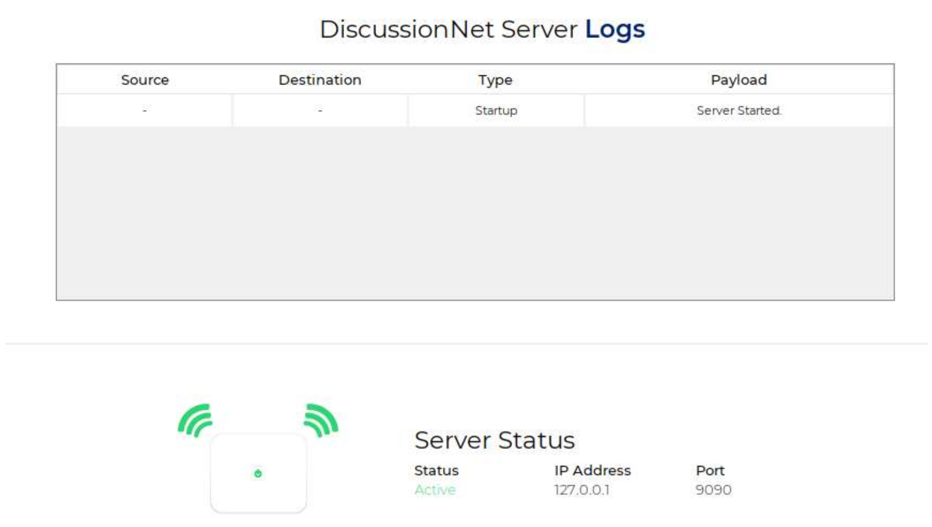
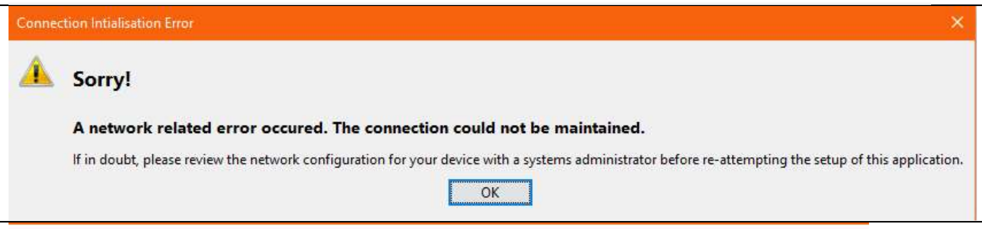
Test File	Test Results
ClientInfoTest.java	ClientInfoTest testToString testToStringError
ClientInstructionTest.java	ClientInstructionTest createRejectJoinRequestInstructionString testConvertInstructionToString testCreateAcceptClientConnectionInstructionString testCreateAddClientInfoToLocalListInstructionString testCreateBecomeCoordinatorInstructionString testCreateClientAcceptedInstructionString testCreateClientDisconnectedInstructionString testCreateConnectionRejectedByCoordinatorInstructionString testCreateEstablishConnectionInstructionString testCreateGetUpdatedClientInfoListInstructionString testCreateNotifyClientDisconnectedInstructionString testCreateNotifyOthersOfNewCoordinatorInstructionString testCreateReviewJoinRequestInstructionString testCreateRevokeCoordinatorInstructionString testCreateSendMessageInstructionString testCreateSetLocalClientInfoListString testCreateUpdateClientInfosServerCacheInstructionString
ServerChannelTest.java	ServerChannelTest testAddMessageToChannel testCheckClientConnectionExists testNullAddMessageToChannel
MessageTest.java	MessageTest testDirectMessageFromString testDirectMessageToString testGroupMessageFromString testGroupMessageToString
ClientManagerTest.java	ClientManagerTest testAddClientInfoToLocalList testGetAllClientIDsFromLocalList testGetAllClientsInfoFromLocalList testGetAllClientsInfoFromLocalListAsFormattedString testGetClientInfoFromLocalList testGetClientStatus

A.3 - Manual Testing

No.	Test	Test Type	Condition	Input	Expected Output	Actual Output	Status	Comment
32	**Start Server**	Valid	No existing server is initialized in the chosen IP / Port	Media 1	UI Log Interface	Media 2	Pass	Server is initialized
33	**Start Server**	Erroneous	There is another server initialized on the chosen ip/port	Media 1	UI Error Message	Media 3	Pass	New server is not initialized, old server continues running as normal.
34	**Client Connecting**	Valid	Server is initialized. There are no other clients connected	Media 4 Media 5	UI Client Interface Coordinator Pop-Up	Media 6	Pass	Client is connected to the server and made coordinator
35	**Client Connecting**	Valid	Server is initialized. There is at least one other client connected. No other clients are using the same ClientID or Port	Media 4, Media 7	UI Client Interface	Media 8	Pass	Client is connected to the server
36	**Client Connecting**	Erroneous	Server is initialized. There is at least one other client connected. At least one client is using the same ClientID or Port	Media 4, Media 9	Network Error Message	Media 10	Pass	Client will not be connected to the server
37	**Coordinator Changes**	Valid	Server is initialized. There are two clients connected. Coordinator (Client A) Disconnects		Non-coordinator client gets the coordinator pop-up message	Media 11	Pass	Client B is made coordinator
38	**Coordinator Changes**	Valid	Server is initialized. There is one client connected. Coordinator (Client B) Disconnects. Client C connects	Media 4, Media 13	UI Client Interface Coordinator Pop-Up	Media 13	Pass	Client C is connected to the server and made the new Coordinator
39	**Unexpected Server Shutdown**	Erroneous	Server is initialized. Client A is connected. Server unexpectedly shuts down		Connected Clients get a disconnected from server pop up	Media 14	Pass	As expected.

40	**Test Messaging**	Valid	Server is initialized. Client A and B are connected. Server-Wide Chat option is chosen.	Media 15	Client A and B see the message in the server-wide chat	Media 16,17	Pass	Message is added to the server-wide channel
41	**Test Messaging**	Valid	Server is initialized. Client A and B are connected. Direct Message Chat is chosen	Media 18	Client A and B see the message on their private chat	Media 19,20	Pass	Message is added to the private channel between Client A and B
42	**Test User List**	Valid	Server is initialized. Client A and B are connected	Media 21	User List pop-up	Media 22	Pass	As expected.
43	**Test Server Log**	Valid	Client A connects.	Media 4,5	Log instruction with Client A Connected	Media 23	Pass	As expected.
44	**Test Server Log**	Valid	Client A becomes coordinator	null	Log instruction with server instruction BecomeCoordinator	Media 24	Pass	As expected.
45	**Test Server Log**	Valid	Client A messages B server-wide.	Media 15	Log message Client A -> server Wide	Media 25	Pass	As expected.
46	**Test Server Log**	Valid	Client A messages B private	Media 18	Log Message Client A -> ClientB	Media 26	Pass	As expected.
47	**Test Server Log**	Valid	Client A disconnects	null	Log instruction Notify client B that client A disconnected	Media 27	Pass	As expected.
48	**Test Server Log**	Valid	Client C connects	Media 28	Log instruction Set new Local Client List	Media 29	Pass	As expected.

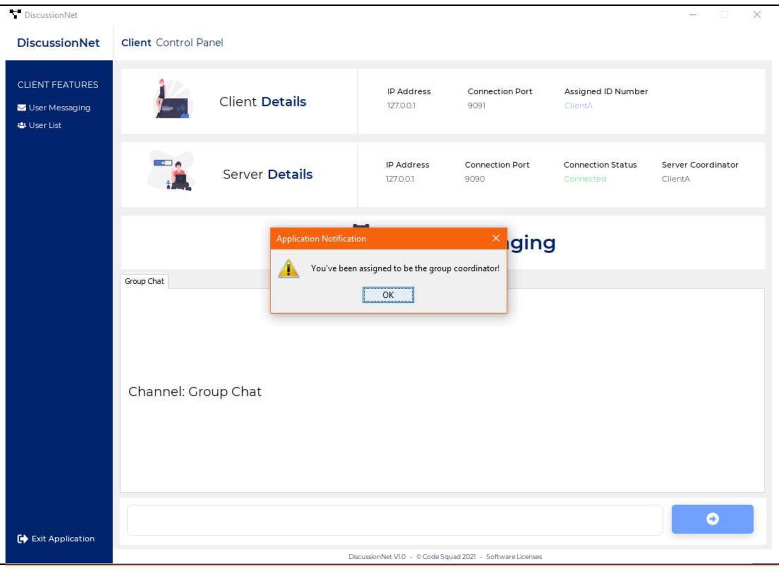
A.4 - Manual Test Screenshots

<table>
<tr><th>ID</th><th>Media</th></tr>
<tr><td>1</td><td>
</td></tr>
<tr><td>2</td><td>
</td></tr>
<tr><td>3</td><td>
</td></tr>
<tr><td>4</td><td>
</td></tr>
</table>

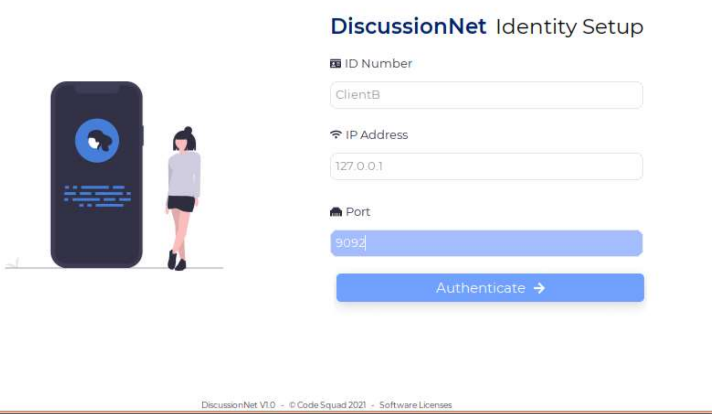
5


DiscussionNet Identity Setup
ID Number
ClientA
IP Address
127.0.0.1
Port
9091
Authenticate

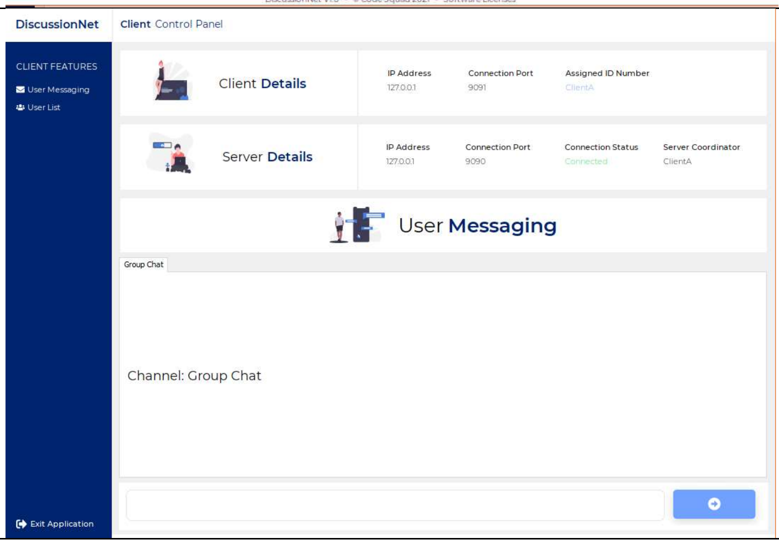
6


DiscussionNet
DiscussionNet
Client Control Panel
CLIENT FEATURES
User Messaging
User List
Client Details
IP Address
127.0.0.1
Connection Port
9091
Assigned ID Number
ClientA
Server Details
IP Address
127.0.0.1
Connection Port
9090
Connection Status
Connected
Server Coordinator
ClientA
Application Notification
You've been assigned to be the group coordinator!
OK
Group Chat
Channel: Group Chat
Exit Application
DiscussionNet V1.0 - © Code Squad 2021 - Software Licenses

7


DiscussionNet Identity Setup
ID Number
ClientB
IP Address
127.0.0.1
Port
9092
Authenticate
DiscussionNet V1.0 - © Code Squad 2021 - Software Licenses

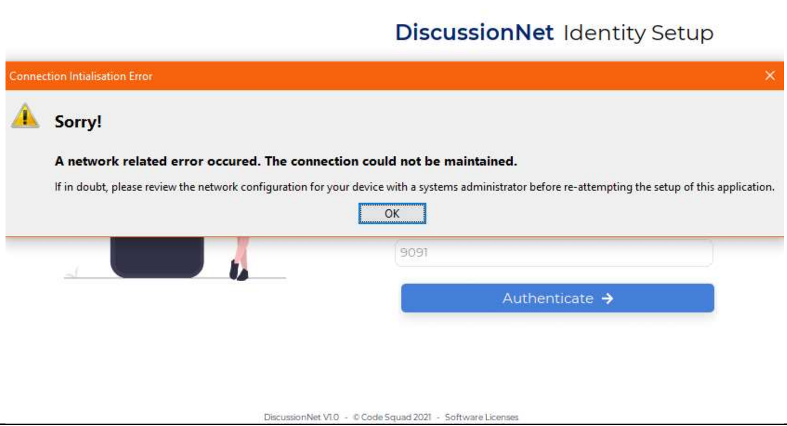
8


DiscussionNet
Client Control Panel
CLIENT FEATURES
User Messaging
User List
Client Details
IP Address
127.0.0.1
Connection Port
9091
Assigned ID Number
ClientA
Server Details
IP Address
127.0.0.1
Connection Port
9090
Connection Status
Connected
Server Coordinator
ClientA
User Messaging
Group Chat
Channel: Group Chat
Exit Application

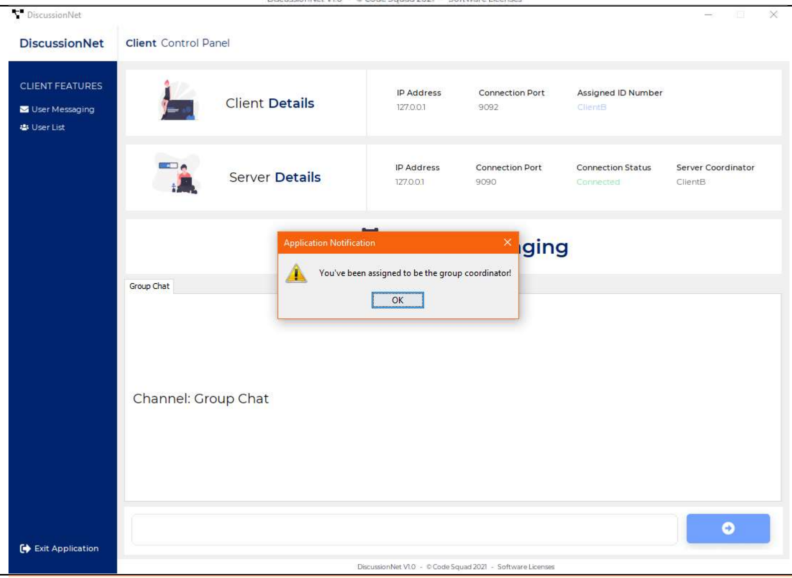
9


DiscussionNet Identity Setup
ID Number
ClientC
IP Address
127.0.0.1
Port
9091
Authenticate

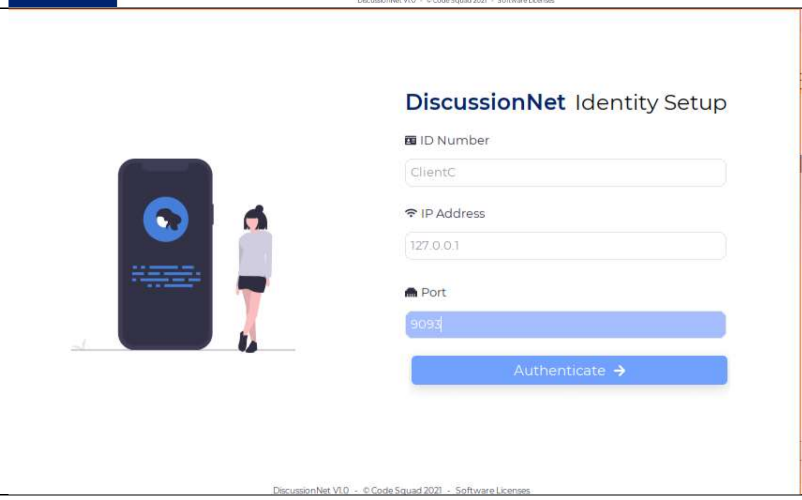
10

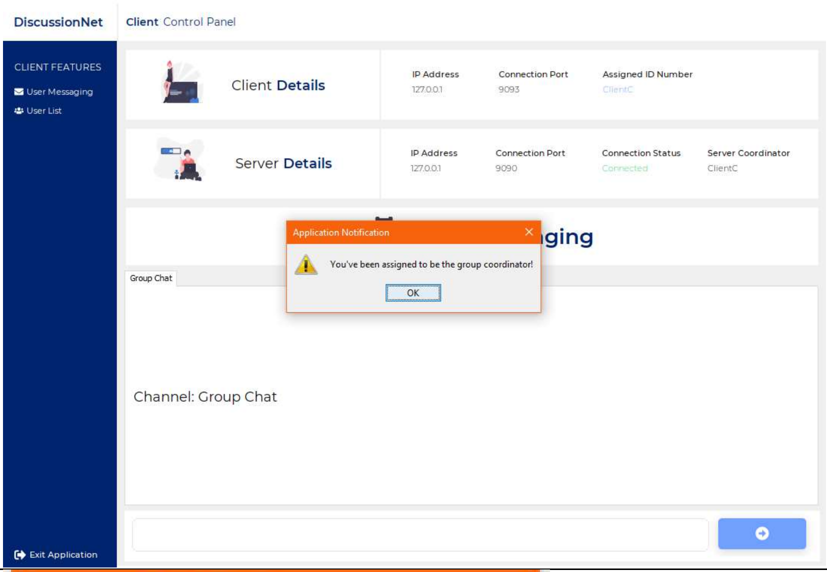
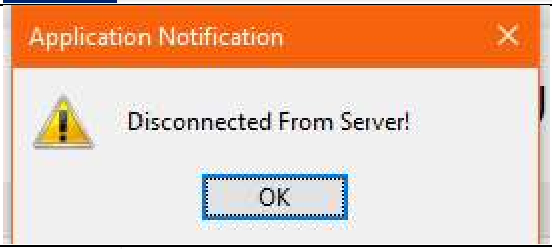
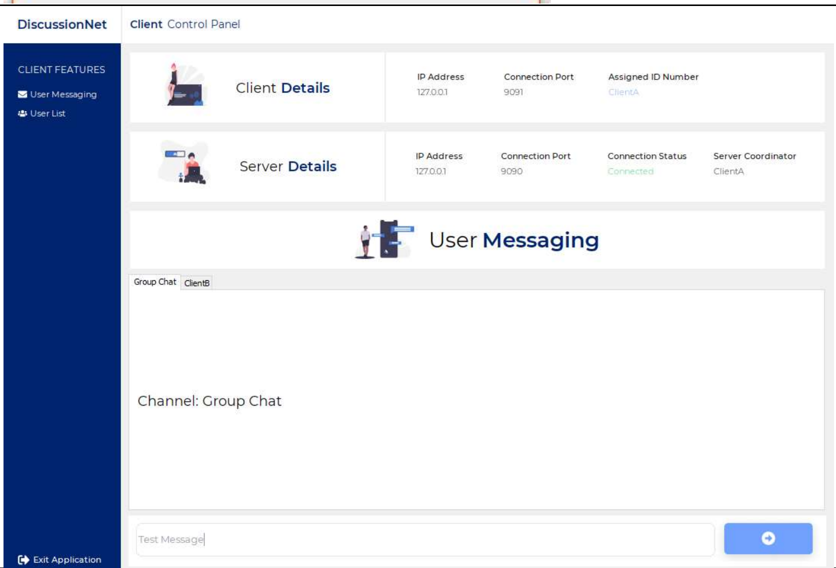
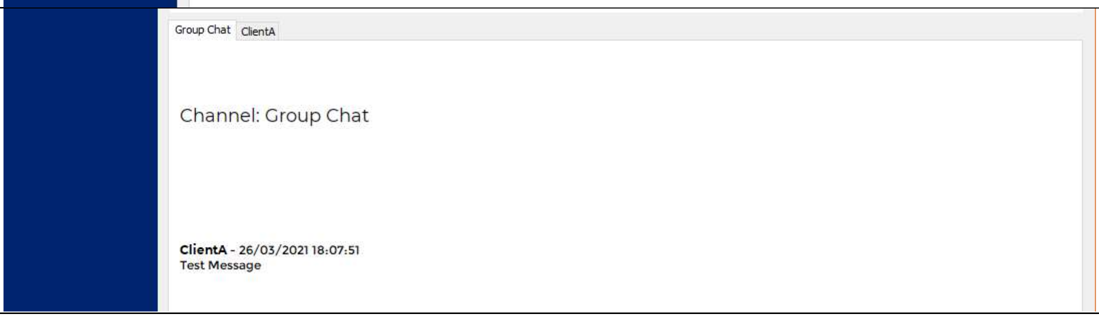

DiscussionNet Identity Setup
Connection Intialisation Error
Sorry!
A network related error occured. The connection could not be maintained.
If in doubt, please review the network configuration for your device with a systems administrator before re-attempting the setup of this application.
OK
9091
Authenticate
DiscussionNet V1.0 - © Code Squad 2021 - Software Licenses

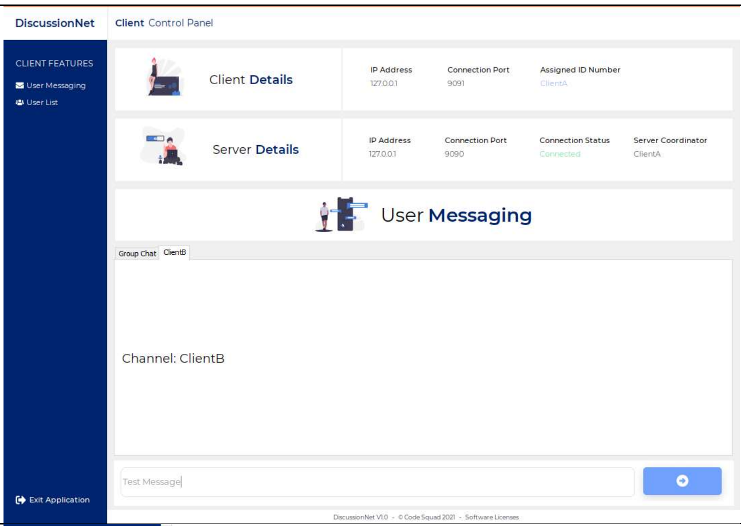
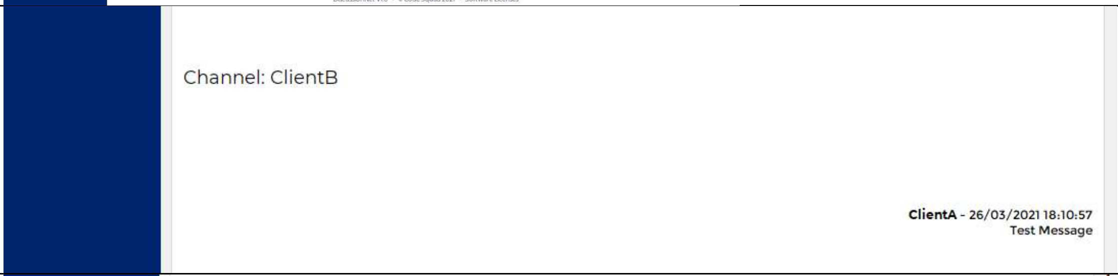
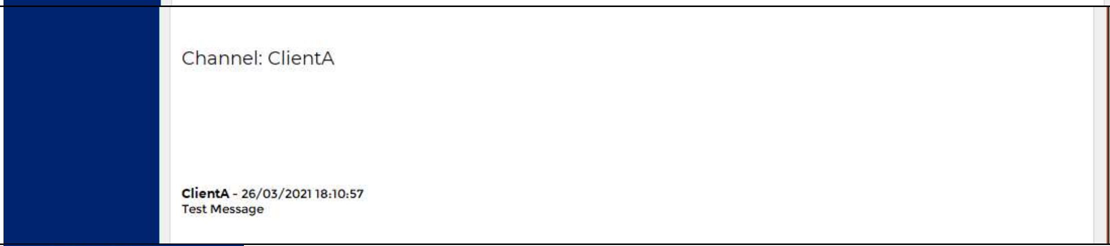
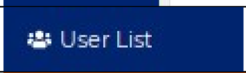
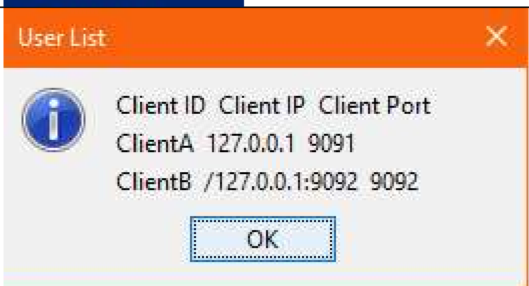
11

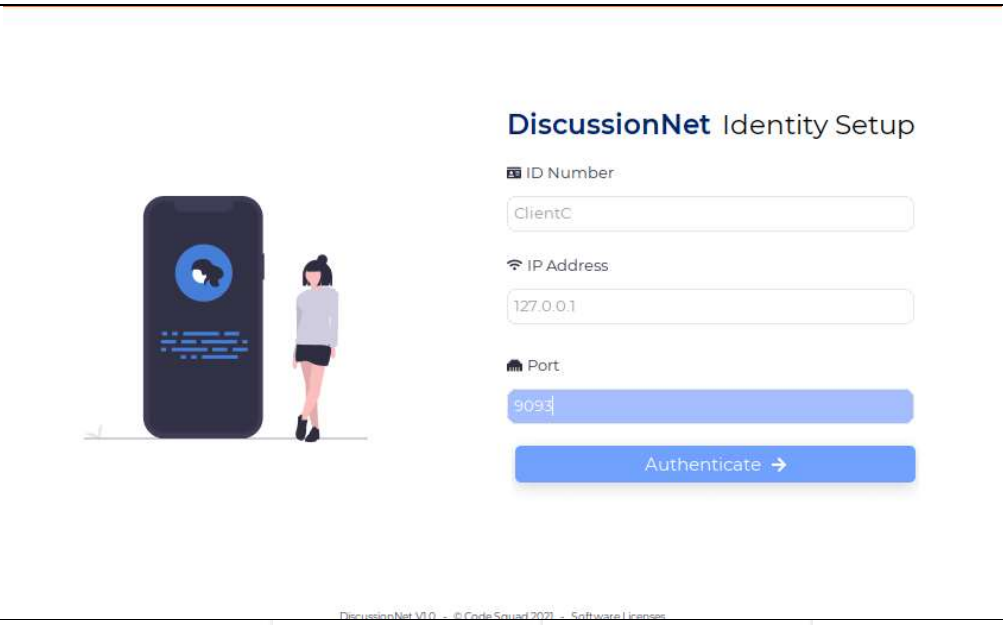

DiscussionNet
DiscussionNet
Client Control Panel
CLIENT FEATURES
User Messaging
User List
Client Details
IP Address
127.0.0.1
Connection Port
9092
Assigned ID Number
ClientB
Server Details
IP Address
127.0.0.1
Connection Port
9090
Connection Status
Connected
Server Coordinator
ClientB
Application Notification
You've been assigned to be the group coordinator!
OK
Group Chat
Channel: Group Chat
Exit Application
DiscussionNet V1.0 - © Code Squad 2021 - Software Licenses

12


DiscussionNet Identity Setup
ID Number
ClientC
IP Address
127.0.0.1
Port
9093
Authenticate
DiscussionNet V1.0 - © Code Squad 2021 - Software Licenses

13	DiscussionNet Client Control Panel CLIENT FEATURES User Messaging User List Client Details IP Address 127.0.0.1 Connection Port 9093 Assigned ID Number ClientC Server Details IP Address 127.0.0.1 Connection Port 9090 Connection Status Connected Server Coordinator ClientC Application Notification You've been assigned to be the group coordinator! OK Group Chat Channel: Group Chat Exit Application
14	Application Notification Disconnected From Server! OK
15	DiscussionNet Client Control Panel CLIENT FEATURES User Messaging User List Client Details IP Address 127.0.0.1 Connection Port 9091 Assigned ID Number ClientA Server Details IP Address 127.0.0.1 Connection Port 9090 Connection Status Connected Server Coordinator ClientA User Messaging Group Chat ClientB Channel: Group Chat Test Message Exit Application
16	Group Chat ClientB Channel: Group Chat ClientA - 26/03/2021 18:07:51 Test Message
17	Group Chat ClientA Channel: Group Chat ClientA - 26/03/2021 18:07:51 Test Message

<table>
<tr><td>18</td><td colspan="4">
</td></tr>
<tr><td>19</td><td colspan="4">
</td></tr>
<tr><td>20</td><td colspan="4">
</td></tr>
<tr><td>21</td><td colspan="4">
</td></tr>
<tr><td>22</td><td colspan="4">
</td></tr>
<tr><td>23</td><td>-</td><td>-</td><td>Connection</td><td>New Client Connected: 3375c5fa-bf1d-4d86-b57b-a...</td></tr>
<tr><td>24</td><td>SERVER</td><td>ClientA</td><td>Instruction</td><td>BECOME COORDINATOR</td></tr>
<tr><td>25</td><td>ClientA</td><td>ClientB</td><td>Message</td><td>Group Chat Message: test message FROM: ClientA</td></tr>
<tr><td>26</td><td>ClientA</td><td>ClientB</td><td>Message</td><td>test message</td></tr>
<tr><td>27</td><td>SERVER</td><td>ClientB</td><td>Instruction</td><td>NOTIFY CLIENT HAS DISCONNECTED</td></tr>
</table>

28	
29	ClientC · ClientB · Instruction · GET UPDATED CLIENT INFO LIST ClientB · ClientC · Instruction · SET LOCAL CLIENT INFO LIST

Appendix B - UML Diagram

Please see attached "UML.pdf" file for non-compressed file.