

# SUSHI CODE: A MODERN CHAT APPLICATION

Alessandro Buonerba, Tommaso Bruno,  
Amir Mostafa, Mustafa Sami

COMP1549: Advanced Programming  
University of Greenwich  
Old Royal Naval College  
United Kingdom

**Abstract**—This document will cover the implementation of the application in detail, from the design to the code structure and logic behind it. Rather than follow a theoretical approach too closely, the development team decided to undertake a practical approach, explaining the reasons why and how this application was built. Additionally, the document will graphically show pieces of software, its features and output, its complications, and its analysis.

**Keywords:** *chat, java, client, server, github.*

## I. INTRODUCTION

This coursework's main focus was to teach students how to build an application that connects clients through a network distributed system that conforms to specific requirements. Before connecting to the server, the user must interact with the initial login interface, which allows the server to store the initial information required to start a new thread specifically tailored for the new user. The parameters were a nickname, a server, and a port. The server must be started before any client can try to approach it, and when a user sends this information through the client, it will be passed to the server, which will start a new thread so that it can handle more than one client at the same time. At the time of joining, the new user will be welcomed with a series of messages that show the list of the currently active and online users, their IPs and port, and more importantly, which of them is the assigned coordinator, also known as the administrator.

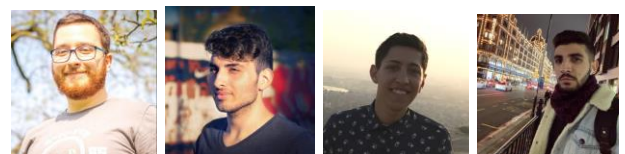
Sushi Code is a centralised server chat application that has two main classes: Client and Server. The main purpose of this application is to visually connect a group of twenty users at most who can then interact with one another through an interface. However, there is more to this application than that. For example, while the main window can be used both to send messages to all users in the

public chat, the development team has also built a private message system that can be accessed through a command. The commands section will be covered in the later sections of this document.

Before starting the development, the team wanted to determine how to manage and track progress, issues, functionalities, bugs fixes, and enhancements. The solution was to use a version control system (VCS) that also contributed to maintaining a consistent, solid backup of all application versions so that they could restore parts that may become compromised. GitHub seemed to be the best choice, and the dev team does not regret it.

The dev team also had to choose which programming language to use, and Java seemed to be the most reasonable choice because it was the one adopted by the university for tutorials and lecture examples. Below is a brief description of the members of the team involved in the creation of this application.

### A. The Dev Team



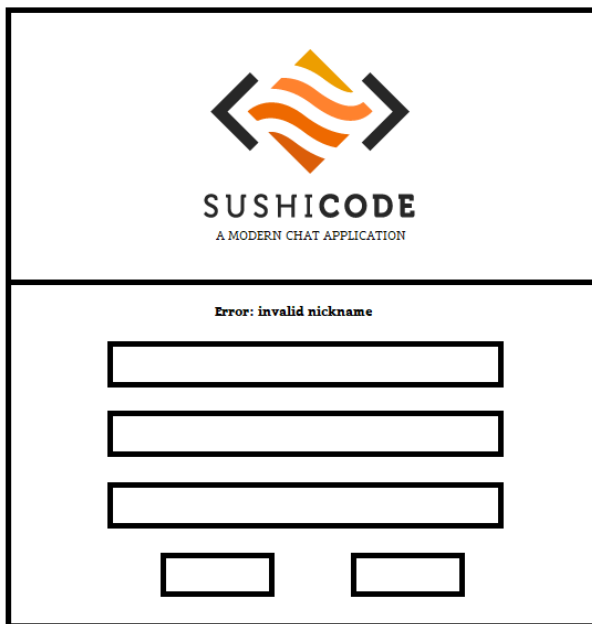
The dev team is composed of four university students based in London. They always strive to make their applications the best they can be, going above and beyond the expectations.

Working as a team can sometimes be difficult, especially when there are deadlines to meet. For their first application, the dev team decided to go all-in and form a group called Sushi Code, a nice combination of their love for the most popular Japanese dish and their main passion. In the conclusion, there will be a link to the repository, to check out not only this but also various other projects made from them.

## II. DESIGN/IMPLEMENTATION

In the early stages of development, the team came up with different solutions to accommodate the various requirements. The first idea was to create a basic command-line interface (CLI), but after an accurate discussion, the team decided to meet all the technical requirements through a better solution that would be more scalable for future enhancements.

### A. Sketches and Logic



This first sketch was used as a reference for building a login screen that met the first requirement of the coursework. The edit-boxes are used to send the first input from the user, such as ID, server, and port. Here, there is the first check; if the username is already taken by any of the current users, then the server will send an error message asking the user to choose a different username. It will also check if the username has any characters that are not alphanumeric through a regex matches method and give an error accordingly. If everything goes positively, the server will accept the connection with a new thread and `socket.accept` if the server is not full.

### B. Implementations and Requirements

When the server accepts new connections, it will update a few internal variables in the `ArrayList`, such as `userName` and counter. The `userName` array contains the ID's of all users that join the server — whether they are currently online or not — until the last time the server updates it with a broadcast. If the user is the first one to enter the server, he or she will be informed and granted administrator status.



When a user joins after the first member — or better said, after the administrator — the app will contact all the present members through the server and send them the new user's information. Each user will then reply — always through the server — with all the members' information, specifying who the current administrator is.

Now that we have all members' information, we can constantly check if anyone has disconnected. To create a list with online members only, the dev team implemented a broadcast function. This function works through a PING-PONG system using a pre-declared `ArrayList`. The server will send a PING to anyone in the `userName` list, which was populated by anyone who joined the server. The online members will then respond through the client with a PONG. The server will then store the username of any clients who replied in the `onlineUsers` `ArrayList` and then compare the array to the `userName` `ArrayList`. If anyone has disconnected, the two arrays will be different in size. Any element in the `userName` array that is not present in the `onlineUsers` array will be removed.

A very important feature was implemented through the `userName` array. The first element of the array is always the administrator. However, if the administrator disconnects — by choice or for any other reason, such as an abnormal disconnection — the administrator status will be passed to a new element to replace the previous one. In this case, the new element is the second user to join the server. This broadcast is launched every five seconds from the start of the server until its end. Additionally, when a user disconnects from the server, everyone online at the time will be informed through a message.

We have so far included and implemented a group formation, connection, and communication system and a group state maintenance system with server logs with timestamps. These features cover almost all the coursework requirements apart from the unit tests, which will be covered in the last part of this chapter.

The following is a list of implemented features and commands that can be executed in order to create a much more functional, scalable, and complete application.

### C. Further Implementations

The team wanted to implement many functionalities during development. A to-do list was created with each command, its description, and its logic. We tried to implement a command each day to achieve our goal of implementing them all. The main command to run is /help because it shows a list of usable commands to the client.

```
this.chatArea.append("-----" + "\n" +
    "/help - to access the various list of command " + "\n" +
    "/clear - to clear the chat client side " + "\n" +
    "/whois - to get information about an user " + "\n" +
    "/quit - to quit the chat " + "\n" +
    "/msg - to send a message to a member " + "\n" +
    "/nickname - to change nickname " + "\n" +
    "/away - to set status away " + "\n" +
    "/online - to come online after being away " + "\n" +
    "/credits - to see application credits and creators " + "\n" +
    "/info - to see personal IP and PORT " + "\n" +
    "/clearlogs - clear logs client side" + "\n" +
    "/motd - set message of the day" + "\n" +
    "-----" + "\n");
```

The above image shows the append used to show all the commands (through /help) to the client in the window chatArea with a small description.

### D. Environment

As stated in the introduction, a JVM seemed to be the most obvious choice because we knew the language very well, and all lecture and lab examples were written in Java. In other circumstances, we would probably have tried out a different language such as Scala, which we find very elegant. However, the main reason why we would want to try it is that it has main, built-in asynchronous threading and non-blocking I/O behaviours. Additionally, it has very high single-line power, which makes it much more appealing than Java because it produces much less code to maintain.

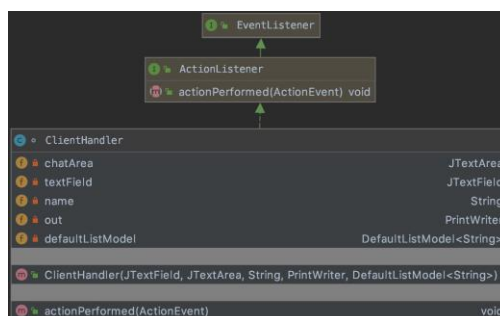
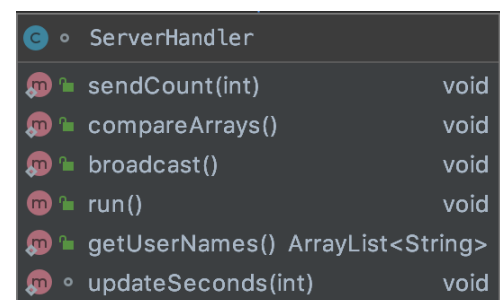
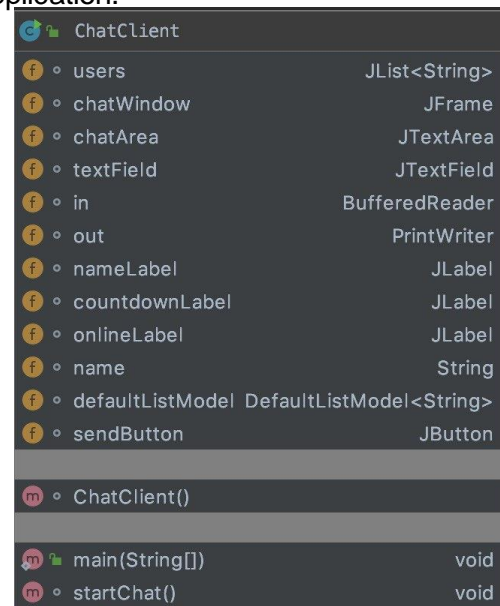
For IDE, we choose IntelliJ Idea Ultimate because we find it to be the undisputed king of the Java world. It has not only a beautiful dark theme (we are looking at you, NetBeans) but also a very clever code completion; it is the most reliable refactoring and one of the best debuggers we have ever seen. We used IntelliJ in conjunction with its new font with enabled ligature, which made coding a much better experience overall with a new and exciting feeling.

We also needed a version control system to see progressive changes and implement the code. Our choice was between GitHub and GitLab. In the end, we choose GitHub simply because we already had some repositories there, and we did not feel the need to migrate to another platform just yet. Another reason we choose GitHub was the perfect integration with Heroku, which made it an even choice, allowing us to have an updated worker Dyno at each push without any manual interaction.

### E. Design and Justifications

The development team intended to make this application scalable and continue its development in the future. To do so, we had a few explicit thinking patterns in mind to apply: write as little code as possible for each functionality to deploy a bug-free experience, reuse as much code as possible when building specific functions with getters and setters and break things into modules so that they can run independently. Also, for security purpose, we designed the code to have as little public variables and methods as possible. This design made the code much more maintainable, and limiting the control the user has over the code is a good practice.

Below are UML diagrams of the main classes of the application.





### III. ANALYSIS AND CRITICAL DISCUSSION

The development team intended to make this application scalable and continue its development in the future. Below are the coding explanations for the requirements with detailed analyses.

#### A. Coding Explanation

In this section, everything related to the requirements for the coursework will be explained in detail, showing what each part of the code does to meet the expectations.

```
out1.println("NAMEACCEPTED" + name);
if (ServerHandler.onlineUsers.size() == 1) {
    out1.println("FIRST");
}
```

The server will handle this part of the code. The first part will be sent to the client if the username meets all the previous checks, such as regex match and a unique name. Also, if the onlineUsers array has only one element, it will send a message to the client to let him or her know what to do next.

```
} else if (str.equals("FIRST")) {
    JOptionPane.showMessageDialog(chatWindow, message: "You are the first user in the chat");
}
```

This is the code on client-side. If the client receives a string that is equal to FIRST, then the server will send a message to the user through a dialogue.

To meet the second requirement, we constructed the following code on the server-side.

```
if (ServerHandler.userNames.size() > 1) {
    for (int i = 0; i < ServerHandler.userNames.size(); i++) {
        if (i == 0)
            out1.println("ID: " + ServerHandler.userNames.get(i) + "/ IP: " +
                sockets.get(0).getInetAddress() + ", PORT: " +
                sockets.get(0).getPort() + " [ADMIN]");
    }
}
```

With a loop based on the array size, the server will send strings to the client to handle. If the user is an admin, the server will also send a string for the client to identify and take the appropriate changes in account.

Next, when a new user joins the server, and everything goes properly, the user's client will then send his or her information to the server for everyone to access. Below is the code implemented.

```
if (!ServerHandler.onlineUsers.contains(name)) {
    ServerHandler.onlineUsers.add(name);
}
```

At this point, there was a new requirement to meet: how does the server know when someone is not responding anymore? We implemented a PING-PONG system. The server sends a PING to all clients connected. If the client is online, the client will send a PONG back to the server. The server will then add all those who replied to another array. onlineUsers. The server will compare the

userName array and the refreshed onlineUsers to determine which users have been disconnected (abnormally or not).

```
public static void compareArrays() {
    broadcast();
    System.out.println("USERNAMES BEFORE: " + ServerHandler.userNames);
    System.out.println("ONLINE BEFORE: " + ServerHandler.onlineUsers);
    ServerHandler.userNames.removeIf(x -> (!ServerHandler.onlineUsers.contains(x)));
    System.out.println("USERNAMES AFTER: " + ServerHandler.userNames);
    System.out.println("ONLINE AFTER: " + ServerHandler.onlineUsers);
}
```

The compareArrays() function will be executed every five seconds in an infinite loop until the server is alive. With that method, we have a broadcast system and are comparing arrays constantly without any user interaction. Below is the broadcast() function.

```
public static void broadcast() {
    if (ServerHandler.userNames.size() > 0) {
        for (PrintWriter out : ServerHandler.writers) {
            out.println("PING");
        }
    }
}
```

Next, if the admin does not respond or disconnects, a new admin will be assigned automatically and systematically.

```
if (ServerHandler.onlineUsers.size() > 0) {
    String disconnected = name;
    String newAdmin;
    if (ServerHandler.userNames.size() > 1) {
        if (ServerHandler.userNames.get(0).equals(name)) {
            System.out.println("Admin disconnected");
            newAdmin = ServerHandler.userNames.get(1);
        }
        for (PrintWriter out : ServerHandler.writers) {
            out.println("NEWADMIN" + disconnected + "/" + newAdmin);
        }
    }
}
```

If the onlineUsers array's size is more than zero, then it will declare two variables used in the inners. If the user who disconnected is equal to the first element of the userNames array, the server will give the administrator role to the next element in the array.

The next requirement is that the app must maintain a counter of active users. Because this app is server centralised, this requirement will be handled by the server. A new method was, therefore, created on the server side.

```
for (PrintWriter out : ServerHandler.writers) {
    out.println("SIZE" + ServerHandler.userNames.size());
}
```

The forEach loop is inside the compareArray() function, which is executed every five seconds. All clients receive the keyword SIZE, which is then handled in the following code.

```
} else if (str.startsWith("SIZE")) {
    onlineLabel.setText("Online: " + str.substring(4) + "/ 20");
}
```

The last requirement was for the application to have a quit shortcut for the client. This was implemented through the KeyListener.

```
@Override
public void keyPressed(KeyEvent e) {
    if (e.isControlDown() && e.getKeyCode() == KeyEvent.VK_C) {
        System.out.println("pressed");
        System.exit(status: 0);
    }
}
```

If there is a keyPressed on a KeyEvent and the CTRL button is down with a VK\_C keycode, the server will execute a System.exit(). The rest of the other class will then handle the logic for disconnections.

### B. Modularity

The development team achieved the modular solution of the project using the object-oriented nature of the language. We used techniques such as encapsulation and loose coupling when possible to achieve less interdependency and information flow. Loose coupling makes code highly exchangeable and makes it much easier to swap pieces of code between code, objects, and components.

### C. Fault Tolerance

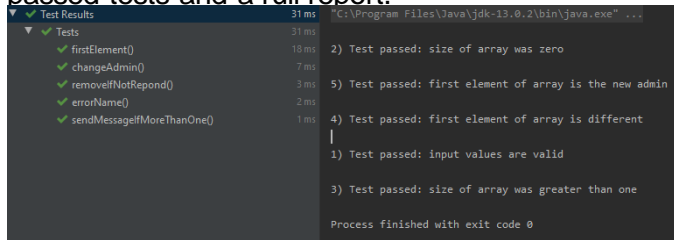
A fault tolerance system was implemented in the deployed application. One of the main implementations was correlated to the requirements assigned. If the administrator disconnects for whatever reason, we have placed a piece of code that automatically takes care of it without any problem — a new administrator will be assigned. Below is the code implemented to take care of this feature.

```
if (ServerHandler.onlineUsers.size() > 0) {
    String disconnected = name;
    String newAdmin;
    if (ServerHandler.userNames.size() > 1) {
        if (ServerHandler.userNames.get(0).equals(name)) {
            System.out.println("Admin disconnected");
            newAdmin = ServerHandler.userNames.get(1);

            for (PrintWriter out : ServerHandler.writers) {
                out.println("NEWADMIN" + disconnected + "/" + newAdmin);
            }
        }
    }
}
```

### D. JUnit Testing

A JUnit framework was used to implement tests on the code that achieves the main requirements of the coursework. In early stages of the application, we wrote a few tests before implementing the code, ensuring that the code would immediately do whatever we intended it to do and helping us a lot in the early development phase. Below are the passed tests and a full report.



The screenshot shows the JUnit test results in an IDE. On the left, a list of test methods is shown with green checkmarks indicating they all passed. On the right, a detailed report lists each test with its duration and a description of what was verified. The tests are: firstElement(), changeAdmin(), removeIfNotRespond(), errorName(), and sendMessageMoreThanOne(). The report also includes a final status: 'Process finished with exit code 0'.

Test Method	Duration	Test Description
firstElement()	18 ms	2) Test passed: size of array was zero
changeAdmin()	7 ms	5) Test passed: first element of array is the new admin
removeIfNotRespond()	3 ms	4) Test passed: first element of array is different
errorName()	2 ms	1) Test passed: input values are valid
sendMessageMoreThanOne()	1 ms	3) Test passed: size of array was greater than one

Process finished with exit code 0

On the right is a list of all test methods, and on the left a System.out.print() that describes how the tests were passed. Full code in the appendix.

### E. Limitations and Weaknesses

One of the main limitations of this chat application is the not yet implemented Nickname server. A Nickname server is an individual and external server that handles user registration and identification so that they can keep their unique IDs. Having such a system is beneficial and removes potential impostors. With the current system, there is no way to know if the user connected is an impostor or not, making the whole system vulnerable to takeovers.

Another limitation is not having a channel server. At the moment, there is only one main room that can be managed by the administrator, and a new coordinator will be chosen systematically when the previous one disconnects, making it very easy for random members to gain power and use it to take over the room.

Additionally, another limitation is not having one or more channel systems with a chan server to manage a list of various rooms. This would expand the application by far, making it possible to create channels with restrictions, such as making them private, by invitation only, and by password only, and most importantly register a channel so that only users with the correct password could declare themselves administrators.

## IV. CONCLUSION

Through this project, we had the chance to test our abilities and push ourselves to the limit both independently and as a group. We learned a lot about more than just networking programming. We wanted to create an application that we would use to communicate with one another, an application that would meet our needs. Even though there is still much to improve and do, we believe that we have achieved what we wanted to and have uploaded something with a discrete level of quality and not just bare bones.

Overall, this project has been a wonderful experience will be highly valuable in the future, especially in our CV. As future improvements, the development team has set a roadmap that involves implementing Nickname and Channel servers. Also, we will make the application even more modular to extends its functionality and scale it to an even larger product to accomplish the creation of different channels with different behaviours and functions.

The GitHub repository used by the development team can be found at the following link:

<https://github.com/Dieman89/chatApp>

## APPENDIX: JUNIT TESTS

```
class Tests {

    @Test
    void errorName() {
        String actual = TestMethods.errorName( user: "Tommaso", ip: "localhost");
        Assertions.assertEquals( expected: "PASSED", actual);

        System.out.print("1) Test passed: input values are valid");
    }

    @Test
    void firstElement() {
        ArrayList<String> users = new ArrayList<>();
        boolean first = TestMethods.firstElementTest(users);

        Assertions.assertTrue(first);

        System.out.print("2) Test passed: size of array was zero");
    }

    @Test
    void sendMessageIfMoreThanOne() {
        ArrayList<String> users = new ArrayList<>();
        users.add("Tommaso");
        users.add("Dieman");
        String first = TestMethods.sendMessageIfSizeIsMoreThanOne(users);

        Assertions.assertEquals( expected: "MESSAGE SENT TO THE OTHERS", first);

        System.out.print("3) Test passed: size of array was greater than one");
    }
}
```

## APPENDIX: JUNIT TESTS

```
@Test
void removeIfNotRepond() {
    ArrayList<String> actualUsers = new ArrayList<>();
    ArrayList<String> online = new ArrayList<>();
    ArrayList<String> finalArray;

    actualUsers.add("Tommaso");
    online.add("Tommaso");
    online.add("Dieman");

    String[] strings = new String[]{"Tommaso"};
    finalArray = TestMethods.testNotRespond(actualUsers, online);

    Assertions.assertArrayEquals(strings, finalArray.toArray());

    System.out.print("4) Test passed: first element of array is different");
}

@Test
void changeAdmin() {
    ArrayList<String> actualUsers = new ArrayList<>();
    ArrayList<String> online = new ArrayList<>();

    actualUsers.add("Tommaso");
    actualUsers.add("Dieman");
    //online.add("Tommaso");
    online.add("Dieman");

    String actual = TestMethods.changeAdmin(actualUsers, online);

    Assertions.assertEquals( expected: "New admin = Dieman", actual);

    System.out.print("5) Test passed: first element of array is the new admin");
}
```

## APPENDIX: JUNIT METHODS

```
public class TestMethods {

    //////////// TEST ////////////

    public static boolean firstElementTest(ArrayList<String> users) {
        boolean first;
        if (users.size() > 0)
            first = false;
        else first = true;
        return first;
    }

    public static String sendMessageIfSizeIsMoreThanOne(ArrayList<String> users) {
        String result;
        if (users.size() > 1)
            result = "MESSAGE SENT TO THE OTHERS";
        else result = "MESSAGE NOT SENT";

        return result;
    }

    public static ArrayList<String> testNotRespond(ArrayList<String> actualUsers, ArrayList<String> onlineUsers) {
        ArrayList<String> finalArray;

        if (actualUsers.equals(onlineUsers)) {
            finalArray = actualUsers;
        } else {
            actualUsers.removeIf(x -> (!onlineUsers.contains(x)));
            finalArray = actualUsers;
        }

        return finalArray;
    }

    public static String changeAdmin(ArrayList<String> actualUsers, ArrayList<String> onlineUsers) {
        String result;

        String adminBefore = actualUsers.get(0);
        actualUsers.removeIf(x -> (!onlineUsers.contains(x)));
        String adminAfter = actualUsers.get(0);

        if (!adminAfter.equals(adminBefore))
            result = "New admin = " + adminAfter;
        else result = "Admin not change = " + adminBefore;
        return result;
    }

    public static String errorName(String user, String ip) {
        String error;
        if (!ip.trim().equals("") || !user.trim().equals("")) {
            if (user.matches( regex: "^[a-zA-Z0-9]+$" ) && (ip.matches( regex: "(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)" )
                || ip.matches( regex: "^[a-zA-Z0-9]+$" )))
                error = "PASSED";
            else error = "FAILED";
        } else error = "FAILED";

        return error;
    }

    //////////// TEST ////////////
}
```



APPENDIX: GITHUB (HOME)

Dieman89 / chatAppPrivate

Unwatch releases0★ Star0🍴 Fork0

<> Code

🔔 Issues1

🔗 Pull requests0

🔧 Actions

📁 Projects0

📖 Wiki

🔒 Security

📊 Insights

⚙️ Settings

Chat application with server and client.

Edit

Manage topics

📄 44 commits

🌿 2 branches

📦 0 packages

🔗 0 releases

👤 2 contributors

📄 MIT

Branch: masterNew pull request

Create new fileUpload filesFind fileClone or download


TommasoBruno99 pusho tuttoLatest commit 17d794a 3 minutes ago

.idea	pusho tutto	3 minutes ago
fonts	final changes	yesterday
images	clearer code	5 minutes ago
src	clearer code	5 minutes ago
.gitignore	READ UPDATED	3 days ago
Chat.iml	READ UPDATED	3 days ago
LICENSE	Create LICENSE	yesterday
README.md	pusho tutto	3 minutes ago

📖 README.md

edit

contributors repo not foundforks repo not foundstars repo not foundissues repo not foundlicense MIT🌐 LinkedIn



SUSHICODE

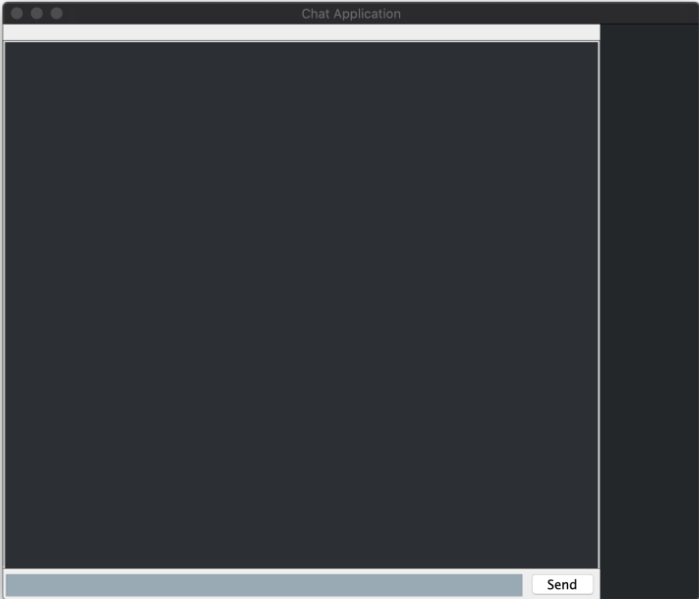
Chat Application

An application realised for an university project.





Explore the docs »


View Demo · Report Bug · Request Feature

About The Project



APPENDIX: GITHUB (ISSUES)

<input type="checkbox"/>	🔔 1 Open	✅ 4 Closed	Author ▾	Label ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	🔴 Quit by Command	enhancement						
	#4 by Dieman89 was closed 2 days ago							
<input type="checkbox"/>	🔴 Existing Members send Info	enhancement						
	#3 by Dieman89 was closed 2 days ago							
<input type="checkbox"/>	🔴 New Member Info	enhancement						
	#2 by Dieman89 was closed 2 days ago							
<input type="checkbox"/>	🔴 Administrator	enhancement						
	#1 by Dieman89 was closed 4 days ago							

<input type="checkbox"/>	🔔 1 Open	✅ 4 Closed	Author ▾	Label ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	🟢 Commands	enhancement						💬 1
	#5 opened 3 days ago by Dieman89							


APPENDIX: GITHUB (LICENSE)

Branch: master ▾

chatApp / LICENSE

Find file

Copy path



Dieman89/chatApp is licensed under the  
**MIT License**

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

✔ Commercial use  
✔ Modification  
✔ Distribution  
✔ Private use


Limitations

✖ Liability  
✖ Warranty

Conditions

🔗 License and copyright notice

This is not legal advice. [Learn more about repository licenses.](#)

 Dieman89 Create LICENSE

0dc1301 yesterday

1 contributor

21 lines (17 sloc) | 1.07 KB

RawBlameHistory📄✎🗑

1 MIT License

2

3 Copyright (c) 2020 Alessandro Buonerba & Tommaso Bruno

4

5 Permission is hereby granted, free of charge, to any person obtaining a copy

6 of this software and associated documentation files (the "Software"), to deal

7 in the Software without restriction, including without limitation the rights

8 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

9 copies of the Software, and to permit persons to whom the Software is

10 furnished to do so, subject to the following conditions:

11

12 The above copyright notice and this permission notice shall be included in all

13 copies or substantial portions of the Software.

14

15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE


18 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

19 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

20 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

21 SOFTWARE.

## APPENDIX: LOGIN



SUSHICODE

No error

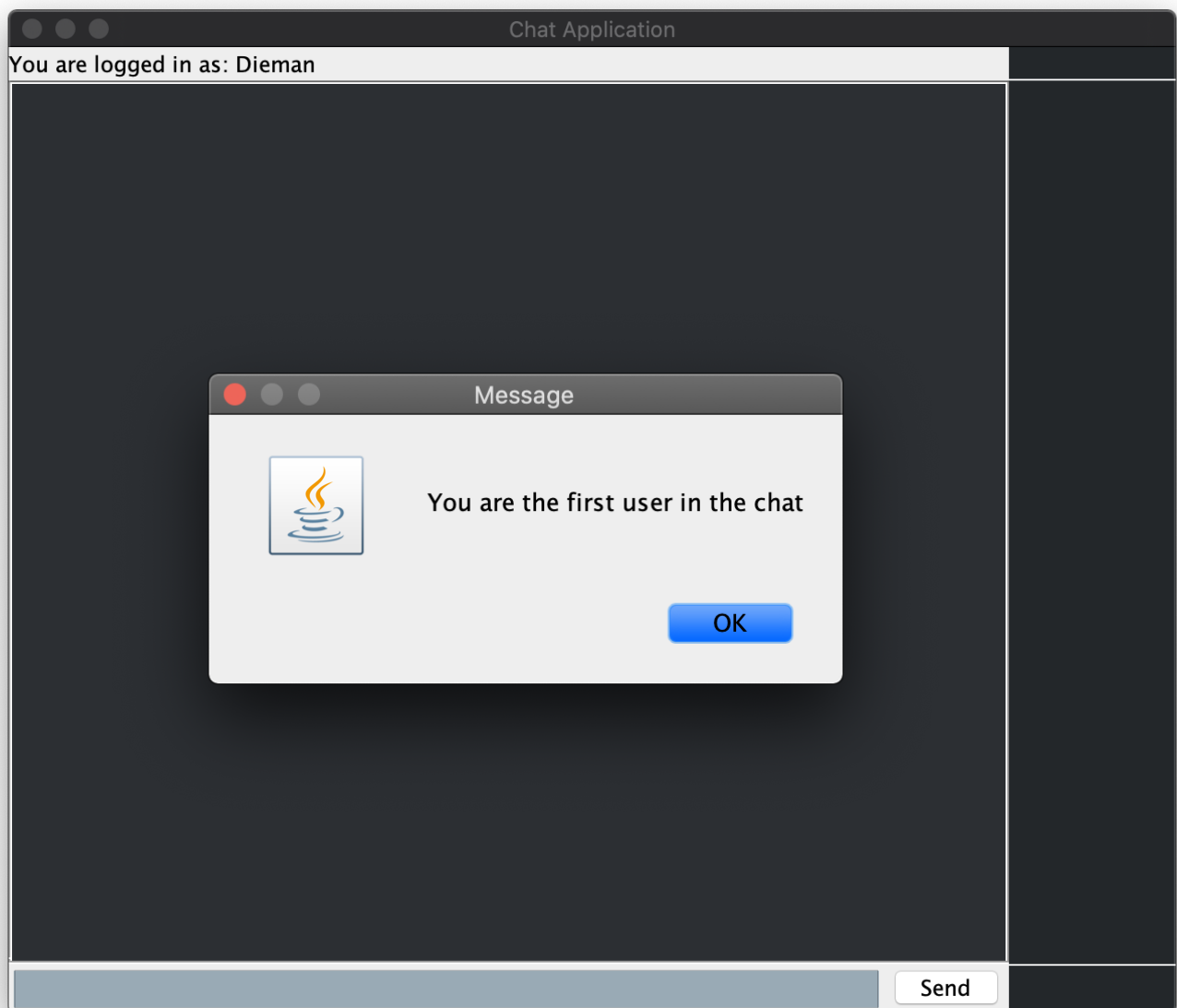
USERNAME

IP ADDRESS

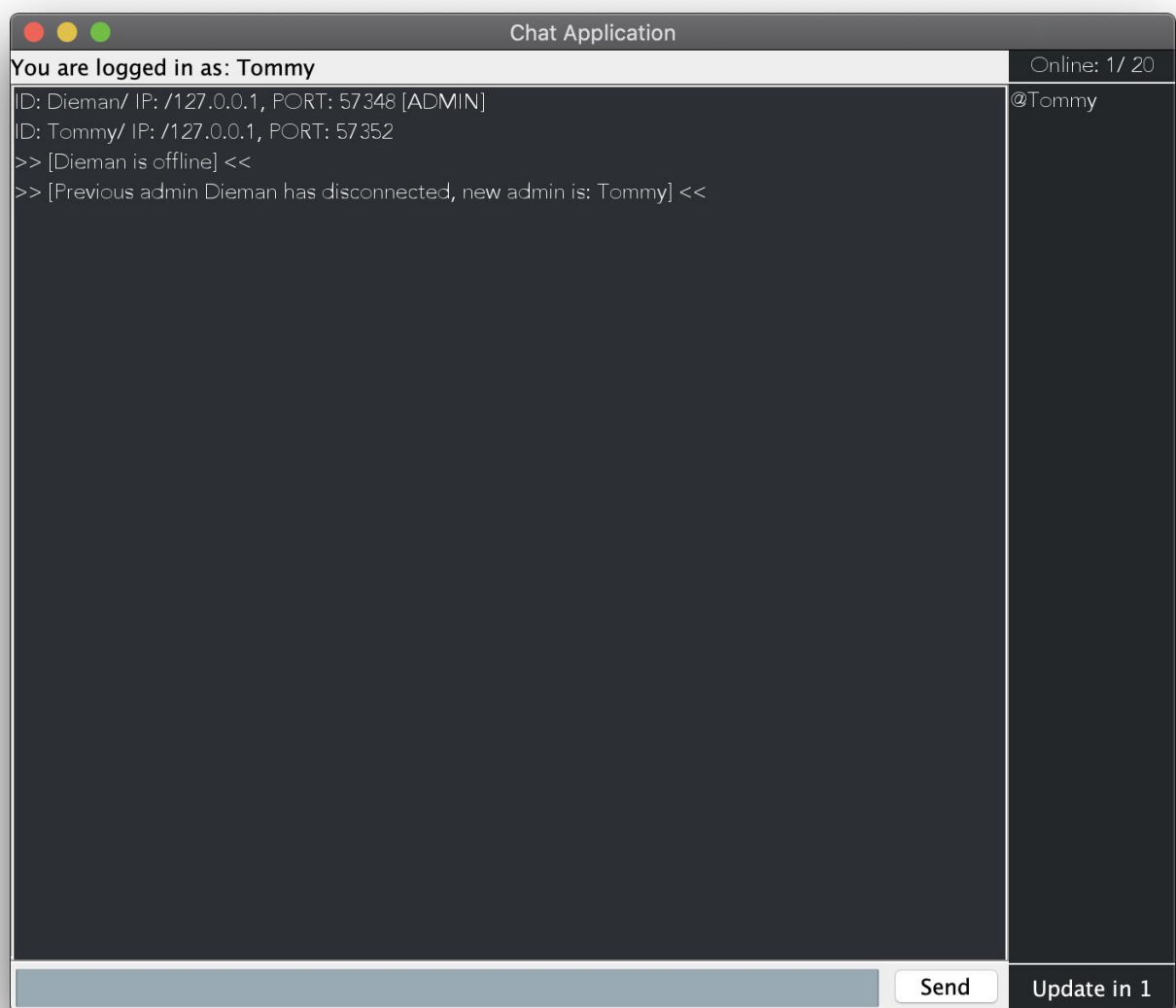
LOGIN

EXIT

## APPENDIX: LIVE (FIRST USER JOIN)

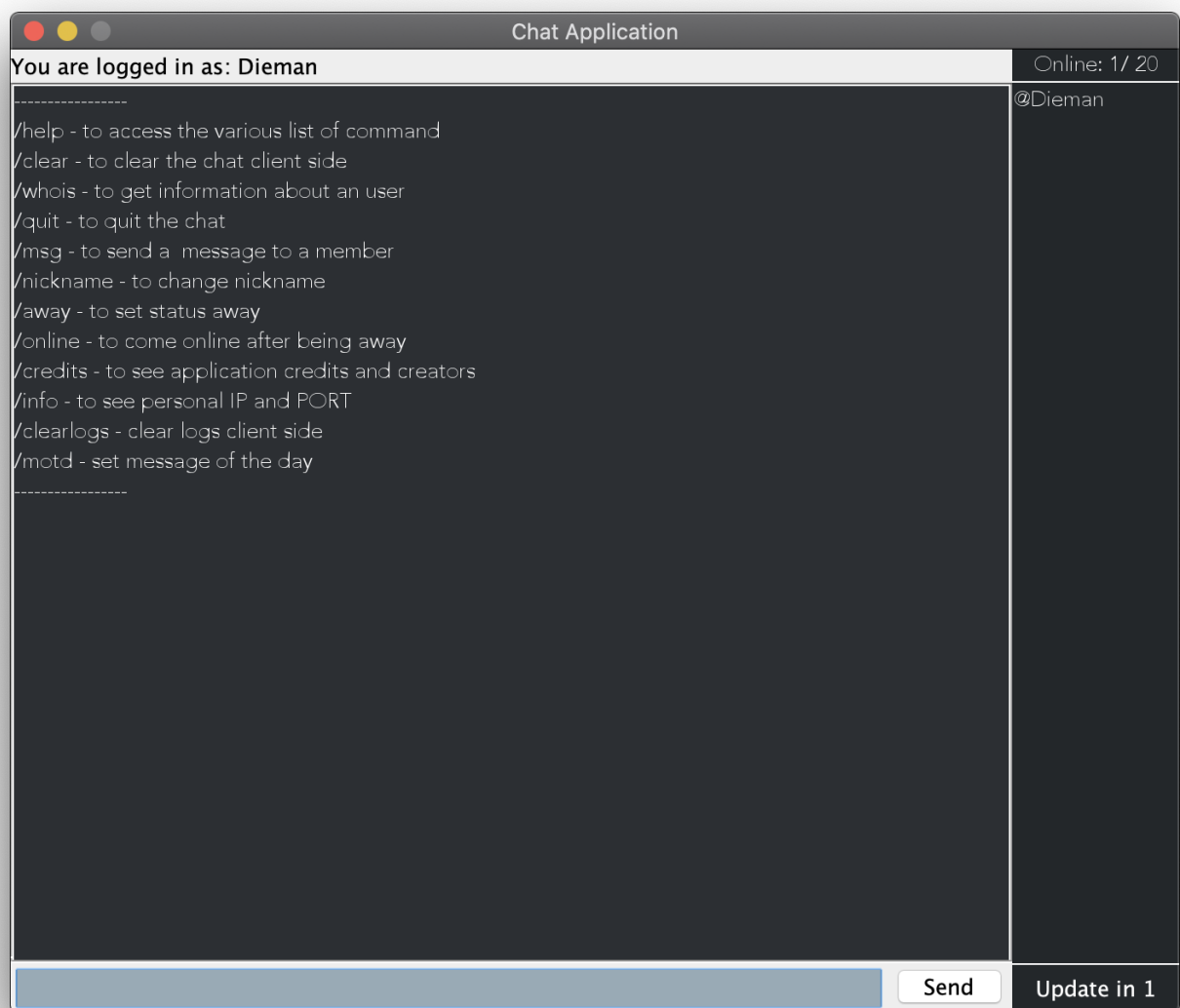


## APPENDIX: LIVE (DISCONNECT)

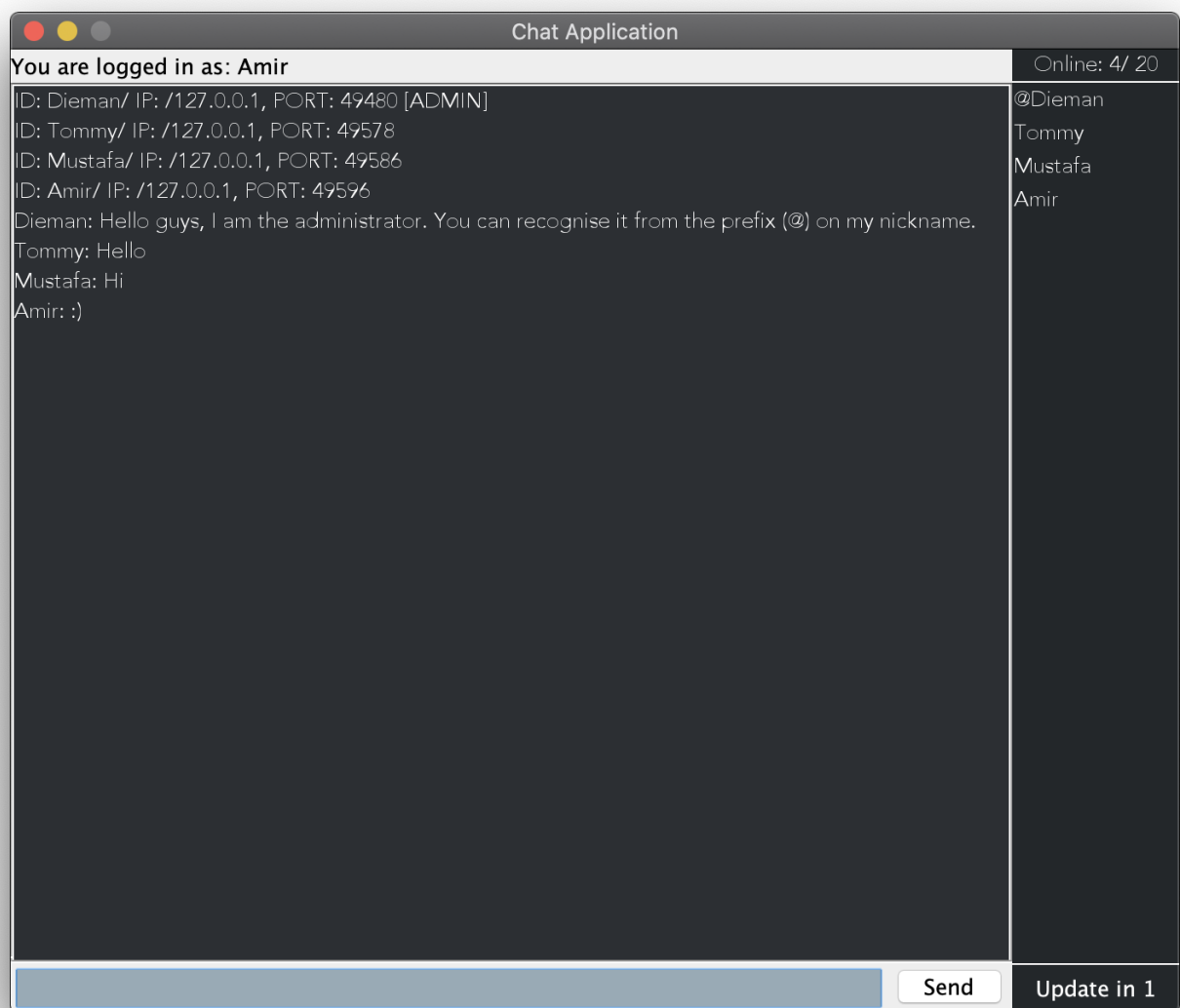




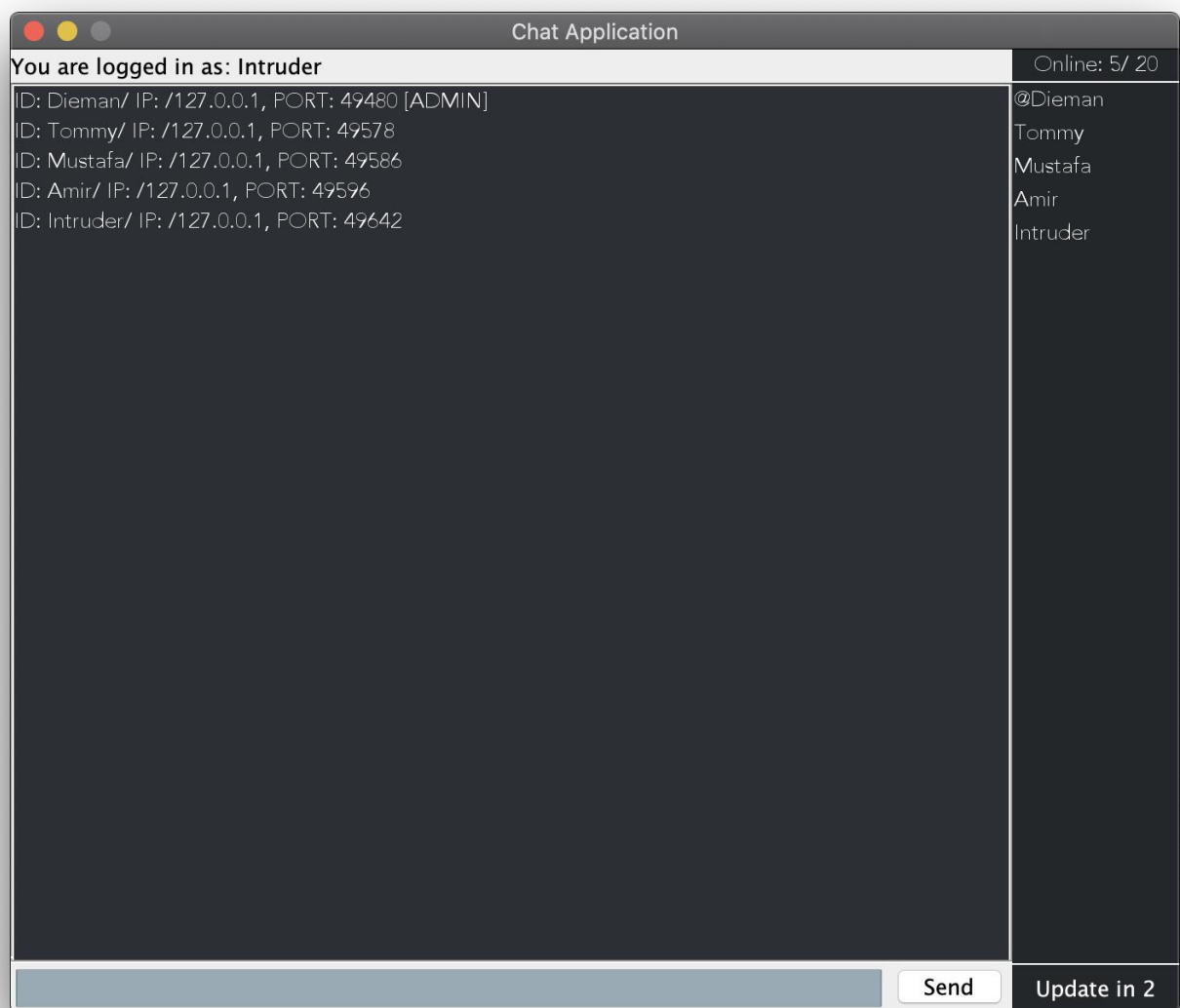
## APPENDIX: LIVE (COMMAND: HELP)



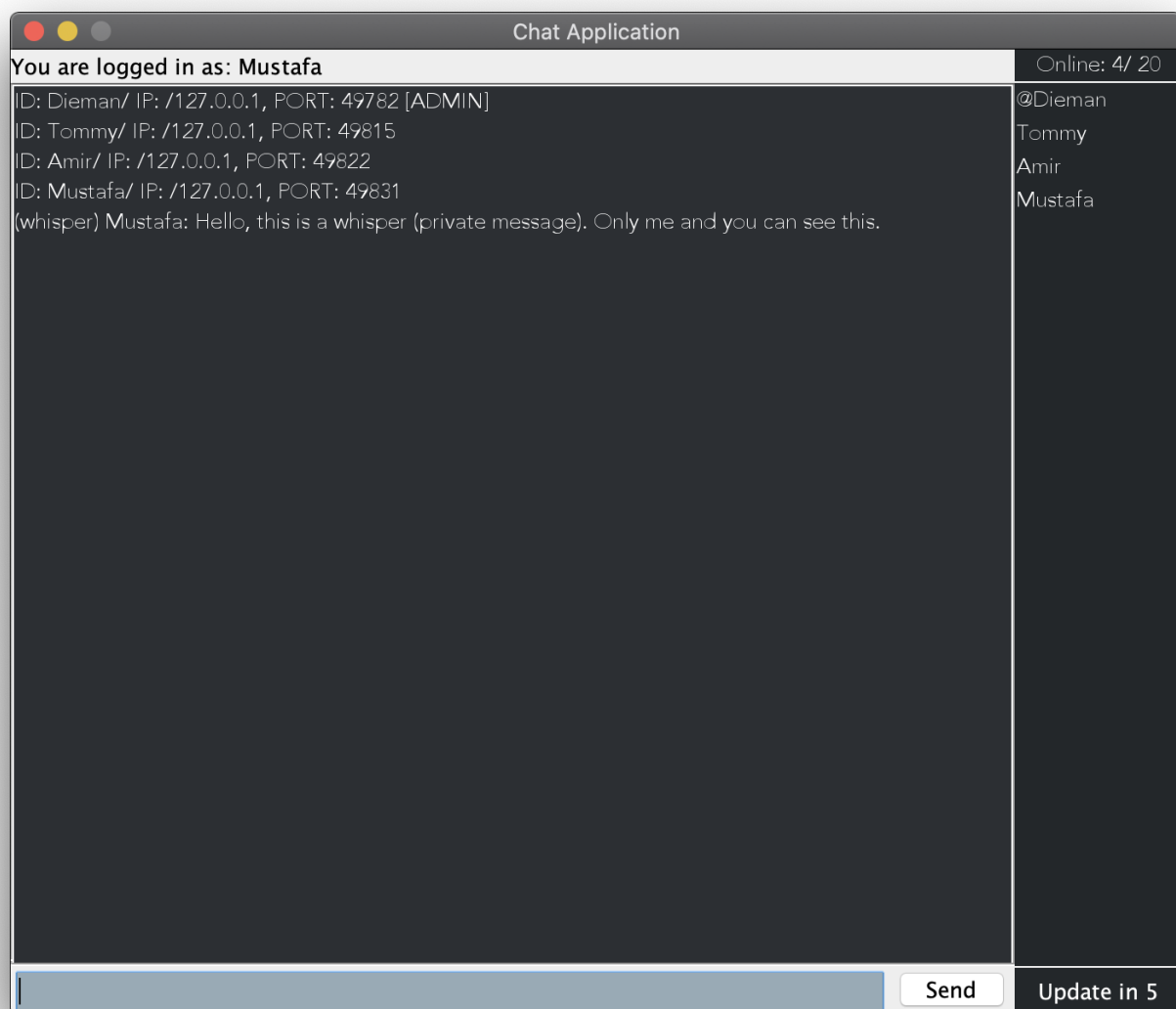
## APPENDIX: LIVE (SIMULATION)



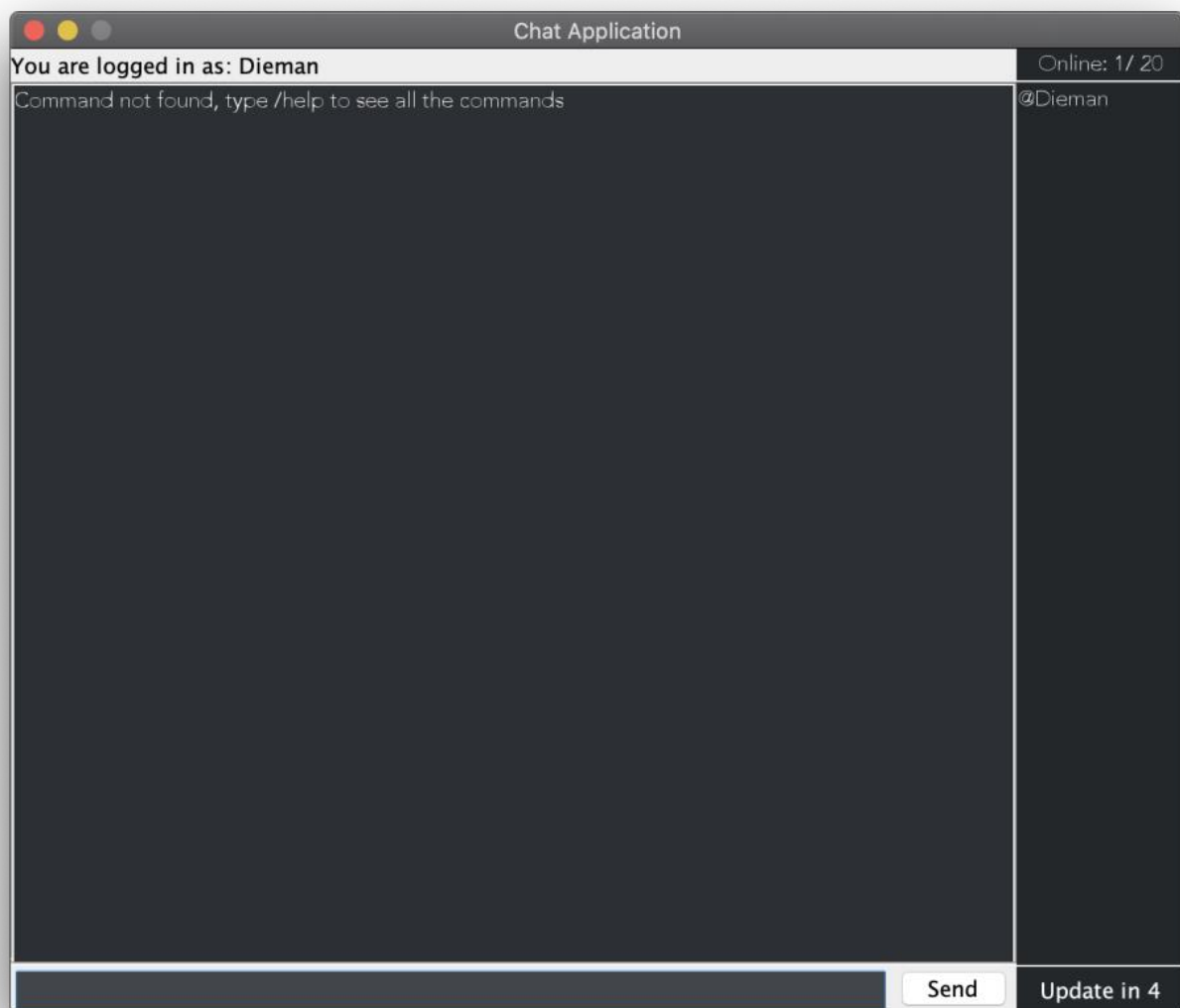
## APPENDIX: LIVE (MESSAGES AT START)



## APPENDIX: LIVE (WHISPER)

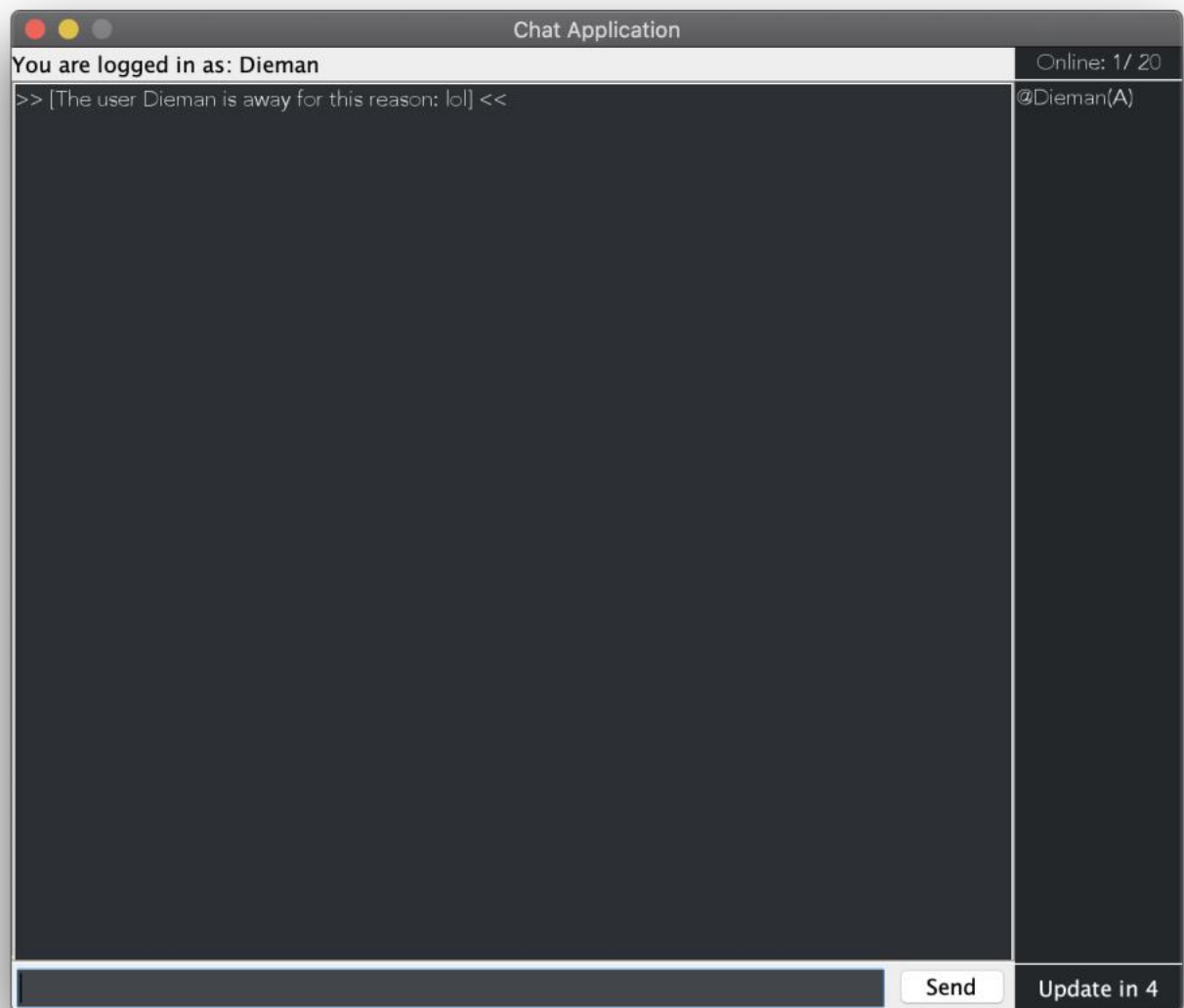


## APPENDIX: LIVE (COMMAND NOT FOUND)





## APPENDIX: LIVE (AWAY)



## APPENDIX: LIVE (ONLINE)

