**Distributed System**

**Project Report**

Distributed Chat Application



Table des matières

[I. Project description and requirements 3](#_Toc171290391)

[II. Architecture Design/Theory 4](#_Toc171290392)

[A) Authentication 4](#_Toc171290393)

[B) Messaging 7](#_Toc171290394)

[1. Private Message 7](#_Toc171290395)

[2. Broadcast Message 7](#_Toc171290396)

[C) Fault tolerance 7](#_Toc171290397)

[III. Implementation 8](#_Toc171290398)

[A) GitHub of the project 8](#_Toc171290399)

[B) Send Broadcast Message 8](#_Toc171290400)

[C) Receive Broadcast Message 8](#_Toc171290401)

[D) Send Private Message 8](#_Toc171290402)

[E) Receive Private Message 8](#_Toc171290403)

[F) Fault tolerance 8](#_Toc171290404)

[IV. Test 9](#_Toc171290405)

# Project description and requirements

The application to implement is a peer-to-peer distributed chat application. With this application, the user should be able to send broadcast messages up to 9 different chatters (connected users) and he should be able to send private messages to a specific chatter.

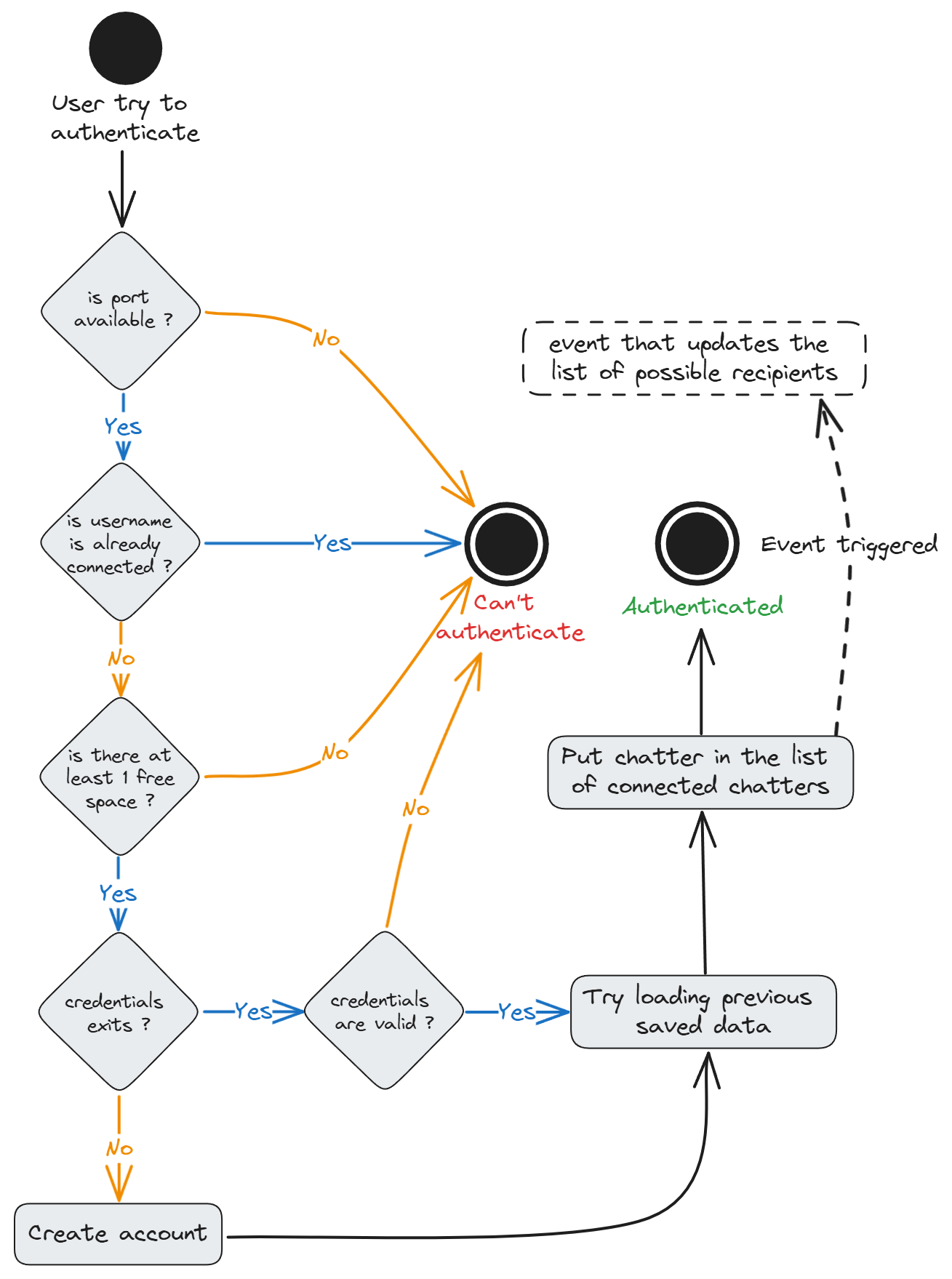
There is few constraints and assumptions made for this project. The maximum of connected chatters at the same time is 10. Communication between chatters should use UDP which means that messages are unreliable and unordered when sent (we must find solutions to address these problems). Broadcast messages should be received in a logical order (find the best algorithm to manage that). Private messages should be received in order (we need define which order).

In the whole report, chatters is define as clients or users.

# Architecture Design/Theory

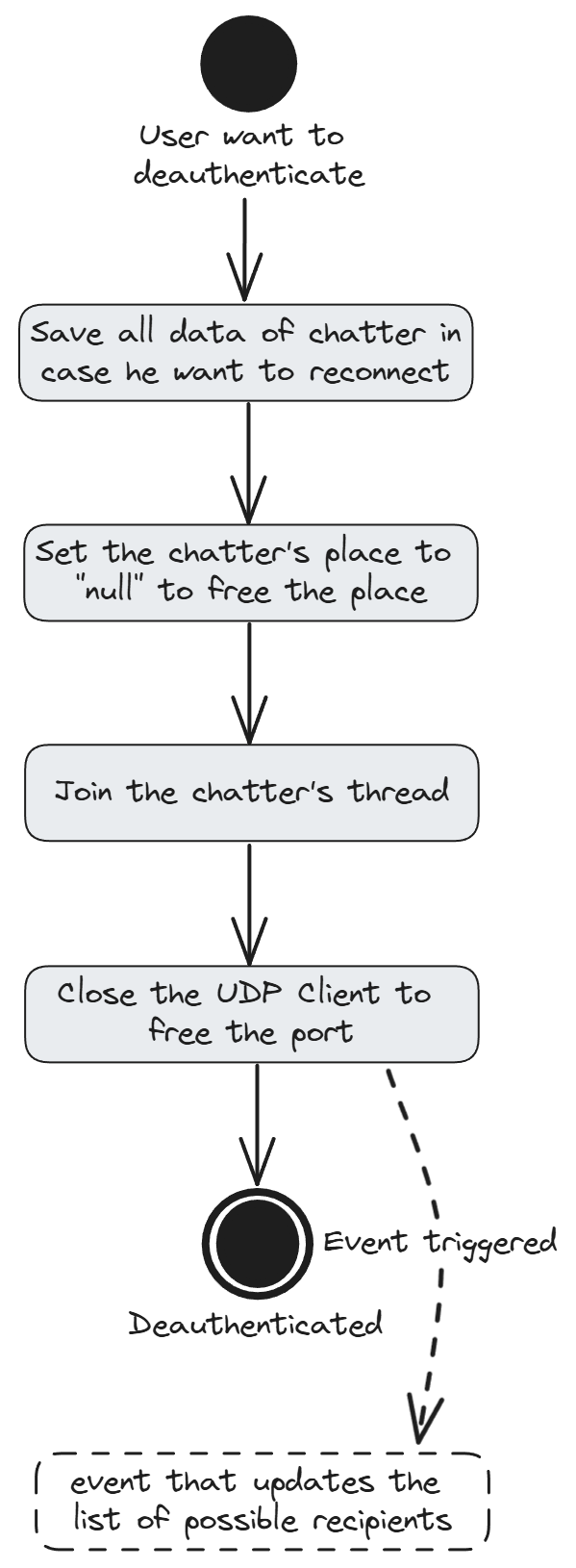
## Authentication

The first step is to authenticate users who wish to use the application. If the user has never logged in, an account must be created. If an account already exists, you need to check the user's login details to make sure they match. However, these are not the only checks to be carried out. Below is an activity diagram showing the algorithmic logic to be applied for server authentication.

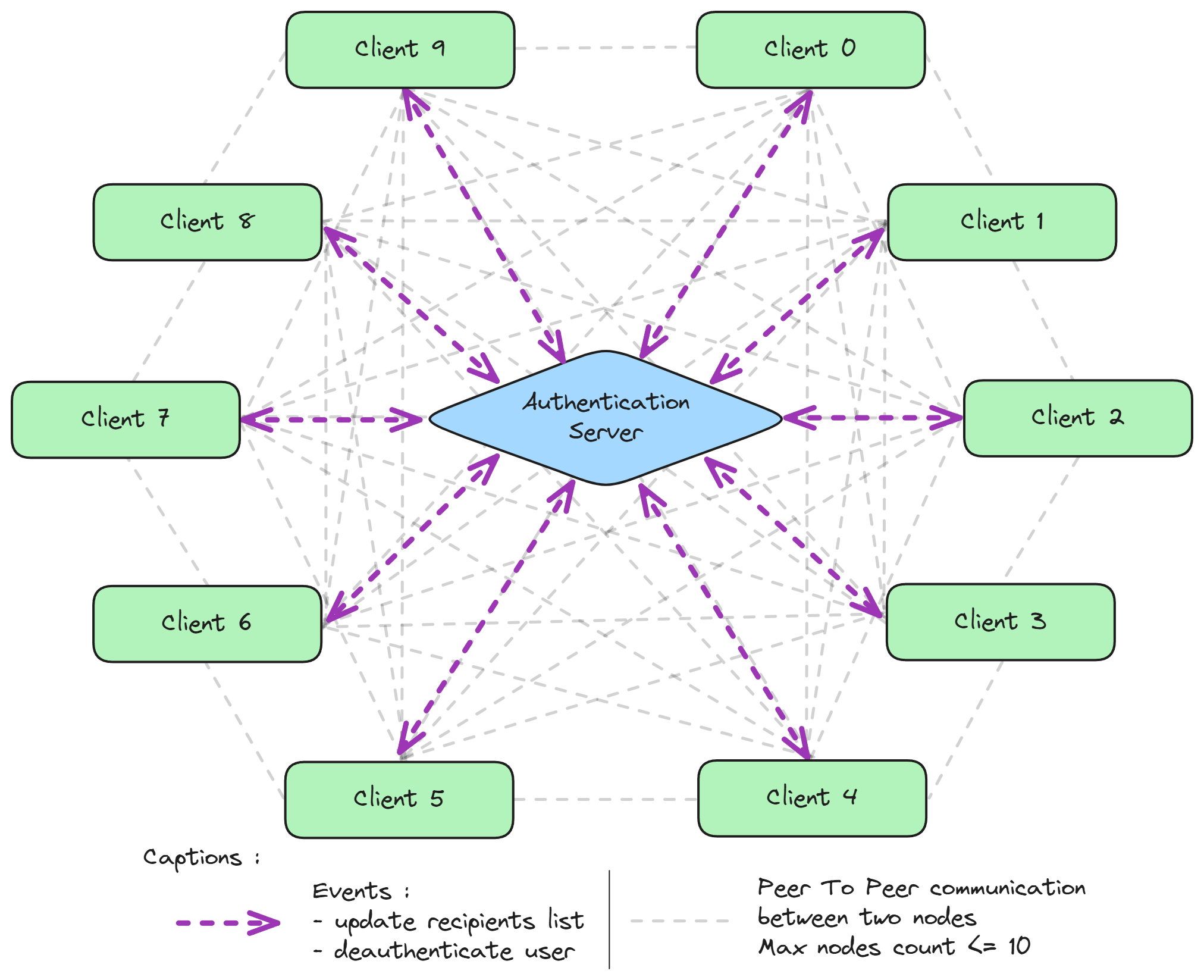


Scheme 1 : Activity Diagram for chatter’s authentication on the application

In fact, if authentication is the most important part of the Authentication Server, the user might also want to log out without causing data loss and/or errors for other users and future users. Below is a diagram of the user logout activity.



Scheme 2 : Activity Diagram for chatter’s deauthentication on the application



Scheme 3 : Diagram focusing on peer to peer with Authentication Server

The events that update the list of possible recipients on clients’ UIs (User Interfaces) allow to each client to select the communication channel he wants to use. All chatters must use the Authentication to instantiate an UDP client that allows for peer-to-peer communication using UDP. Closing the client must automatically call the deauthentication function of the Authentication Server. Since the scope of the project is focusing on private messaging and broadcast messaging, the Authentication is a shared static class that will not communicate with clients using UDP.

## Messaging

### Private Message

### Broadcast Message

## Fault tolerance

# Implementation

## GitHub of the project

## Send Broadcast Message

## Receive Broadcast Message

## Send Private Message

## Receive Private Message

## Fault tolerance

# Test