Object Oriented Design

Chat System Project

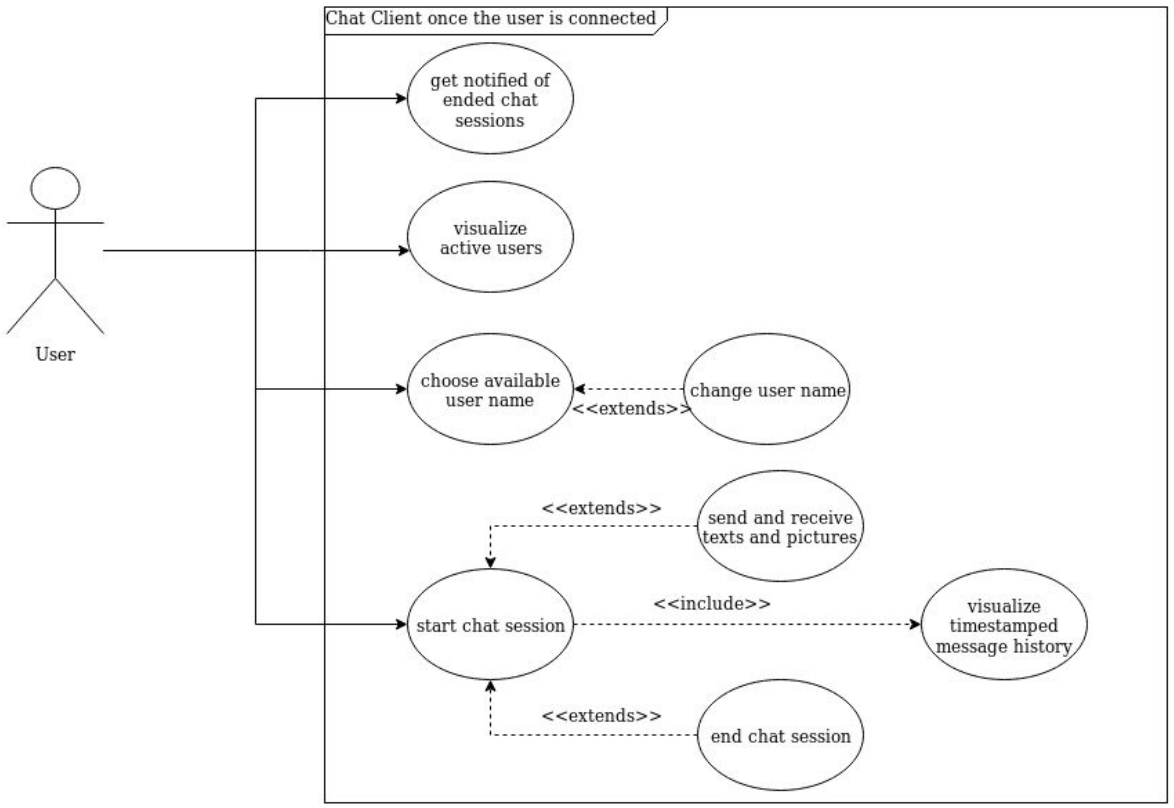
Aurélien Michaud - Laura Burlon--Roux
4 IR A1 | 2019/20

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

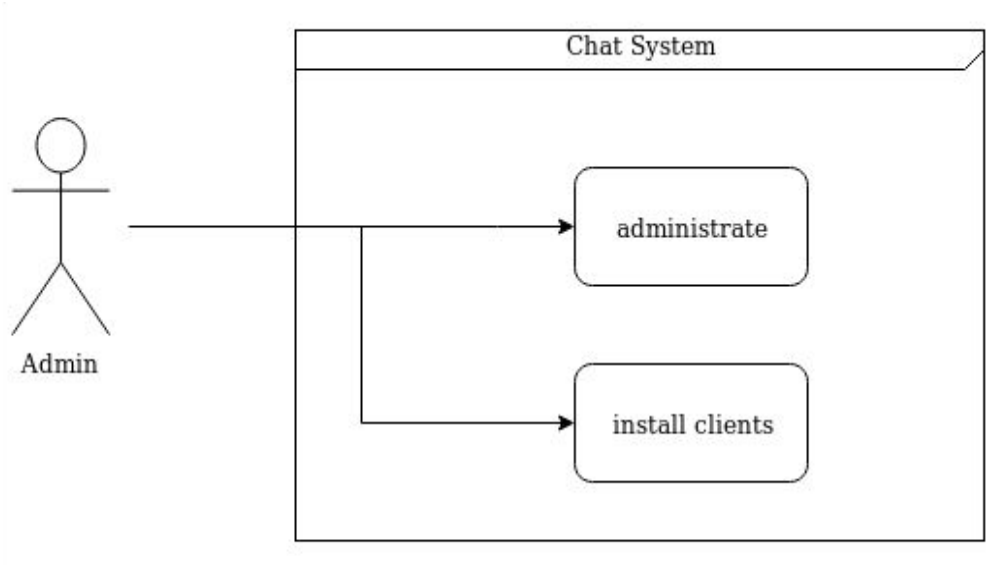
Sommaire

Use Case Diagrams

Client

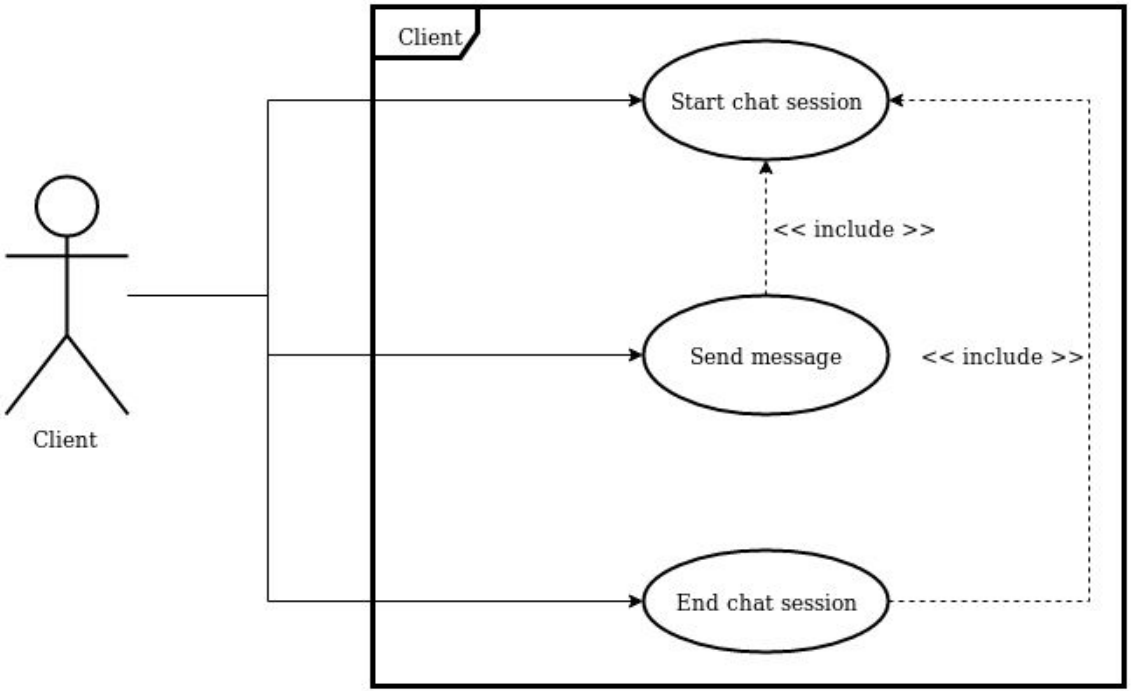


Administrator

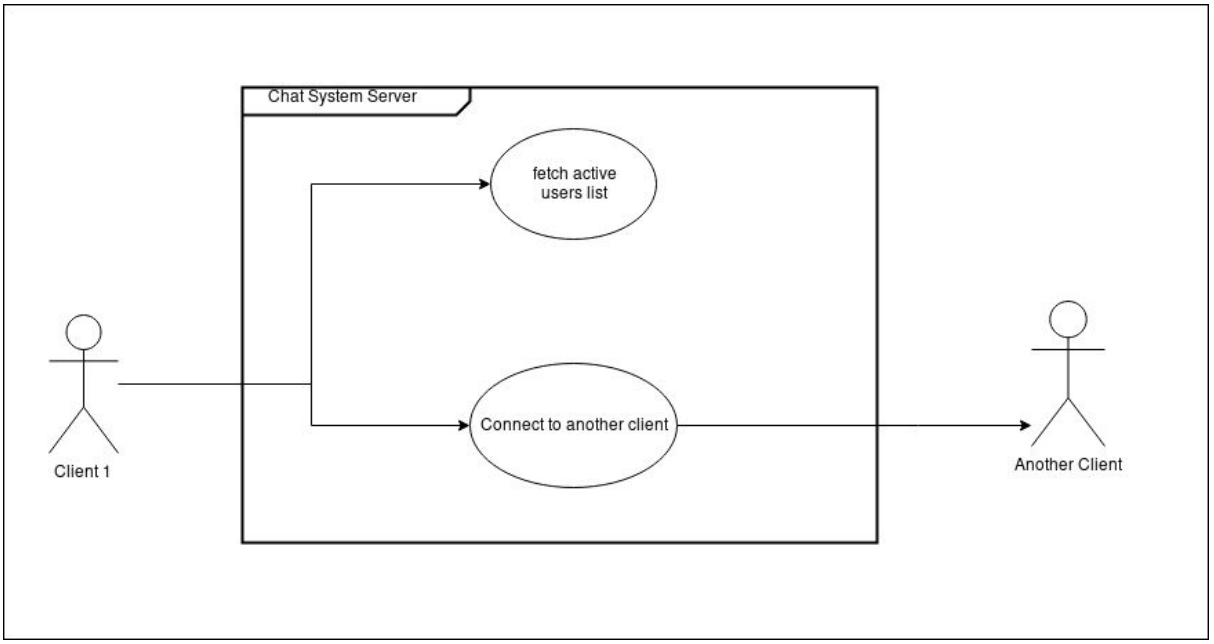


Extrait du cahier des charges : *"En effet, un administrateur de l'organisme qui souhaite utiliser le système réalise l'installation des agents sur les postes des personnes amenées à interagir et, après une configuration minimale du poste de travail, le système est fonctionnel."* (section 1.1.1)

Client to another client



Client remotely connected to another client



Sequence Diagrams

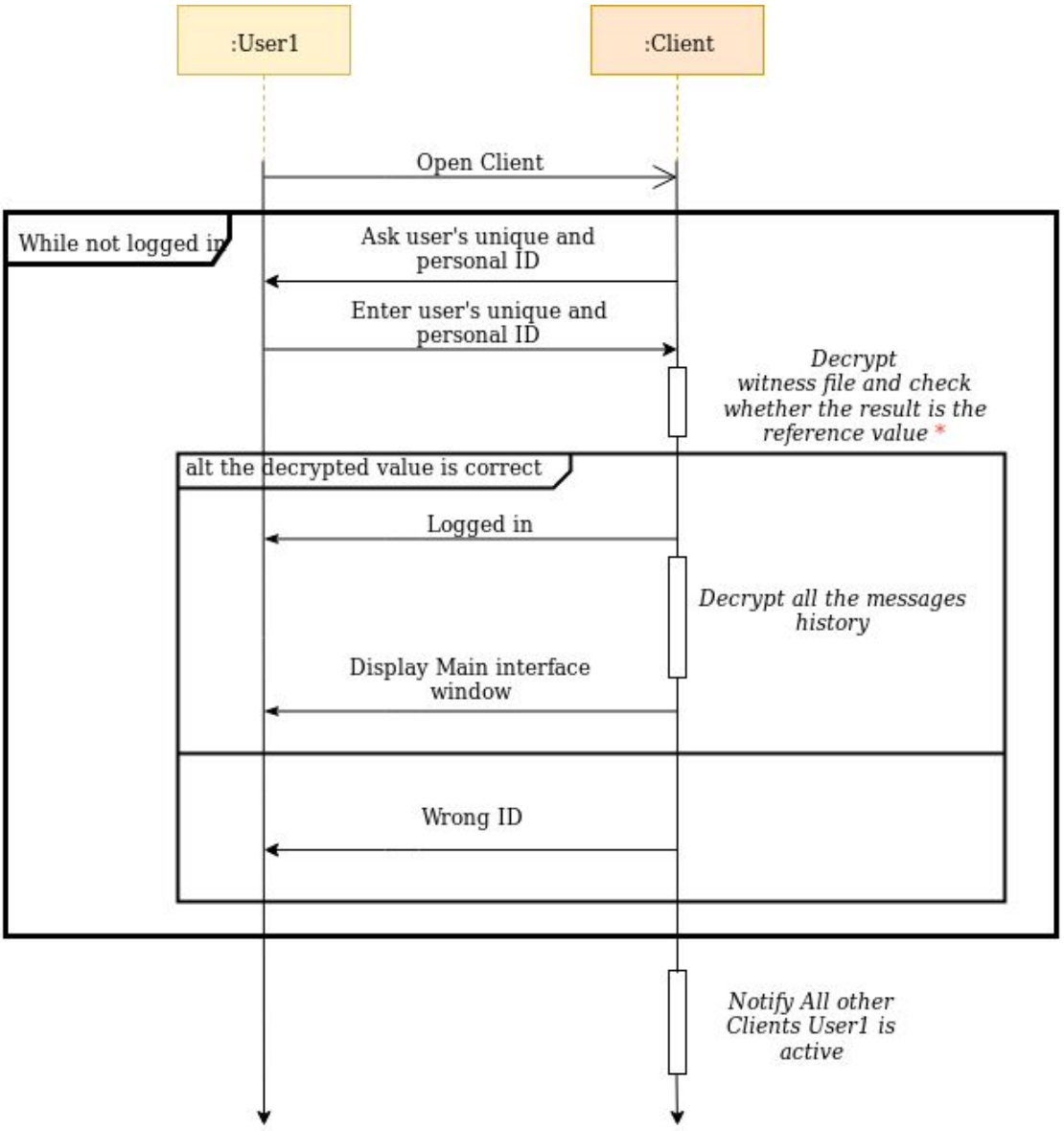
En local, le degré d'acceptation de l'utilisation d'une architecture 3-tiers pour certaines fonctionnalités ayant changé au cours des TD, pour le moment nous préférons vous présenter nos deux travaux de conception.

Pour cela, nous vous exposons en premier lieu le cas d'une "architecture 3-tiers faible", c'est à dire le cas où l'implémentation ne comprend pas d'accès à un serveur distant (même pour accéder à la liste des utilisateurs actifs en local, ou encore l'historique des messages par exemple). Nous avons cru comprendre qu'il s'agissait de la version attendue.

Par la suite vous disposerez de notre réflexion autour de la conception d'une "architecture 3-tiers forte", qui comme nous l'avons compris, ne sera pas la version retenue pour l'implémentation.

Diagramme de séquences architecture 3-tiers faible

User Log In



*Each Client has a witness file : Basically a file containing the characters sequence (e.g. "WITNESS"). This file is encrypted once the client is installed by the administrator with the user's unique and personal ID.
At connection time, the user is asked to provide its ID. This latter one is used to decrypt this same witness file. If the resulting character sequence is the same as the initial one ("WITNESS" in our example), the connection succeed. Otherwise the user is invited to provide the correct ID again.

Chaque utilisateur se verra donner un clé d'authentification. Cette clé sera une suite de symboles hexadécimaux qui fera office de clé de chiffrement pour l'historique des messages.

Chaque historique des conversations est associé à la signature de la clé d'authentification de l'utilisateur avec qui la conversation a eu lieu. Ce système résiste à la très grande variabilité du pseudo par exemple, ou encore de l'adresse IP. Il en est de même au sein de la base de donnée du serveur distant, qui lui ne stocke que la liste des utilisateurs actifs. Ainsi, chaque utilisateur a accès aux signatures des clés de tous les autres utilisateurs (qui font office de certificat) mais cela n'implique aucun problème de sécurité.

Notify that a remotely connected client is active

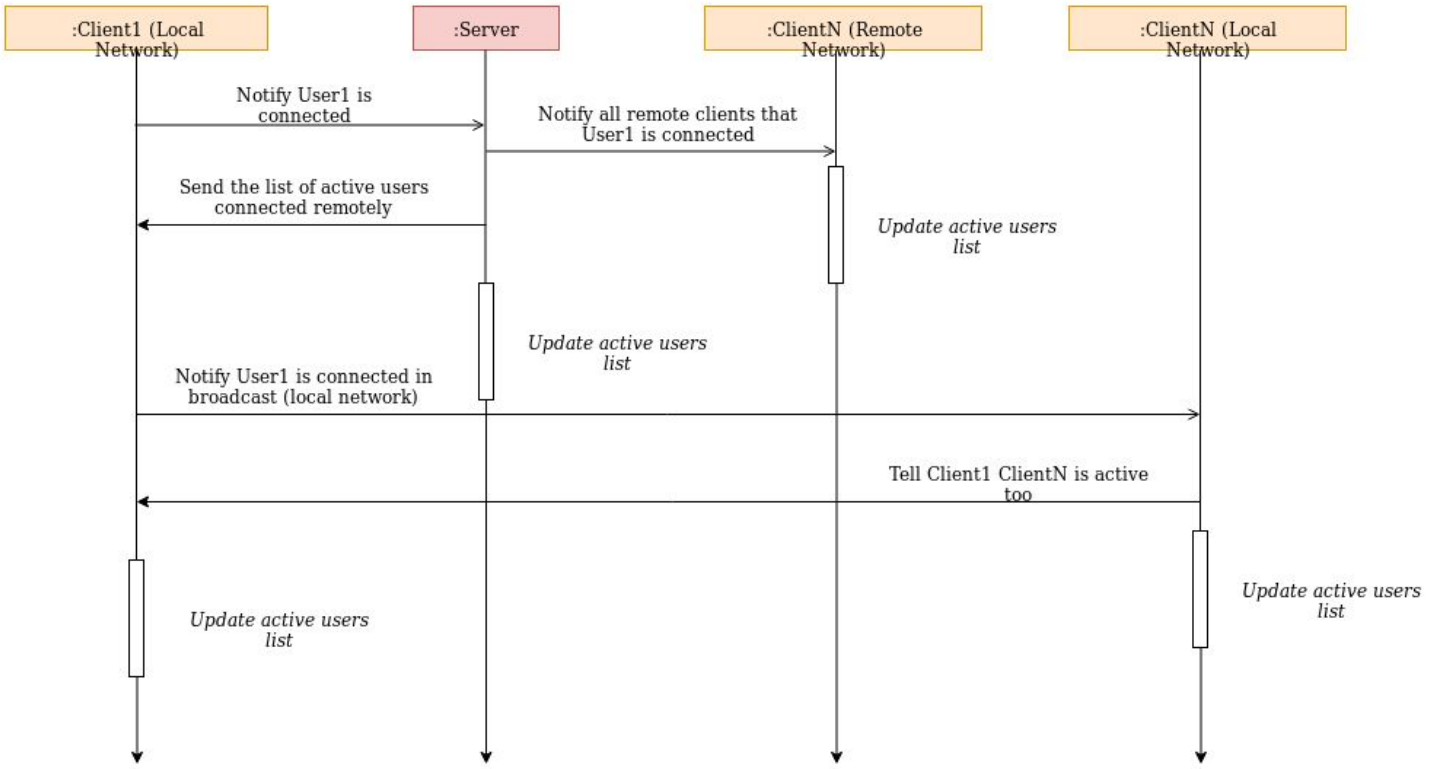
Dans le cas d'un agent (client) connecté à l'extérieur du réseau local, il prévient instantanément le serveur distant pour qu'il puisse à son tour prévenir tous les agents, qu'ils soient en local ou à distance, que ce nouvel utilisateur est désormais actif.

Notify that a locally connected client is active

Lors de la connexion, l'agent (client) en local se charge de prévenir tous les autres agents, sur le même réseau, que l'utilisateur associé est actif (en broadcast). Dans le même temps, il prévient aussi le serveur pour que celui-ci puisse informer les autres agents connectés à distance.

Par le biais de cette notification, l'agent fournira le pseudo de l'utilisateur (une fois que sa liste des utilisateurs actifs, et donc la liste des pseudos, est mise à jour) et sa signature de clé permettant de vérifier sa réelle identité.

Ainsi, dans les deux cas de figure, tous les utilisateurs connectés seront informés en temps réel de la connexion d'un autre utilisateur.



Extrait du cahier des charges correspondant :

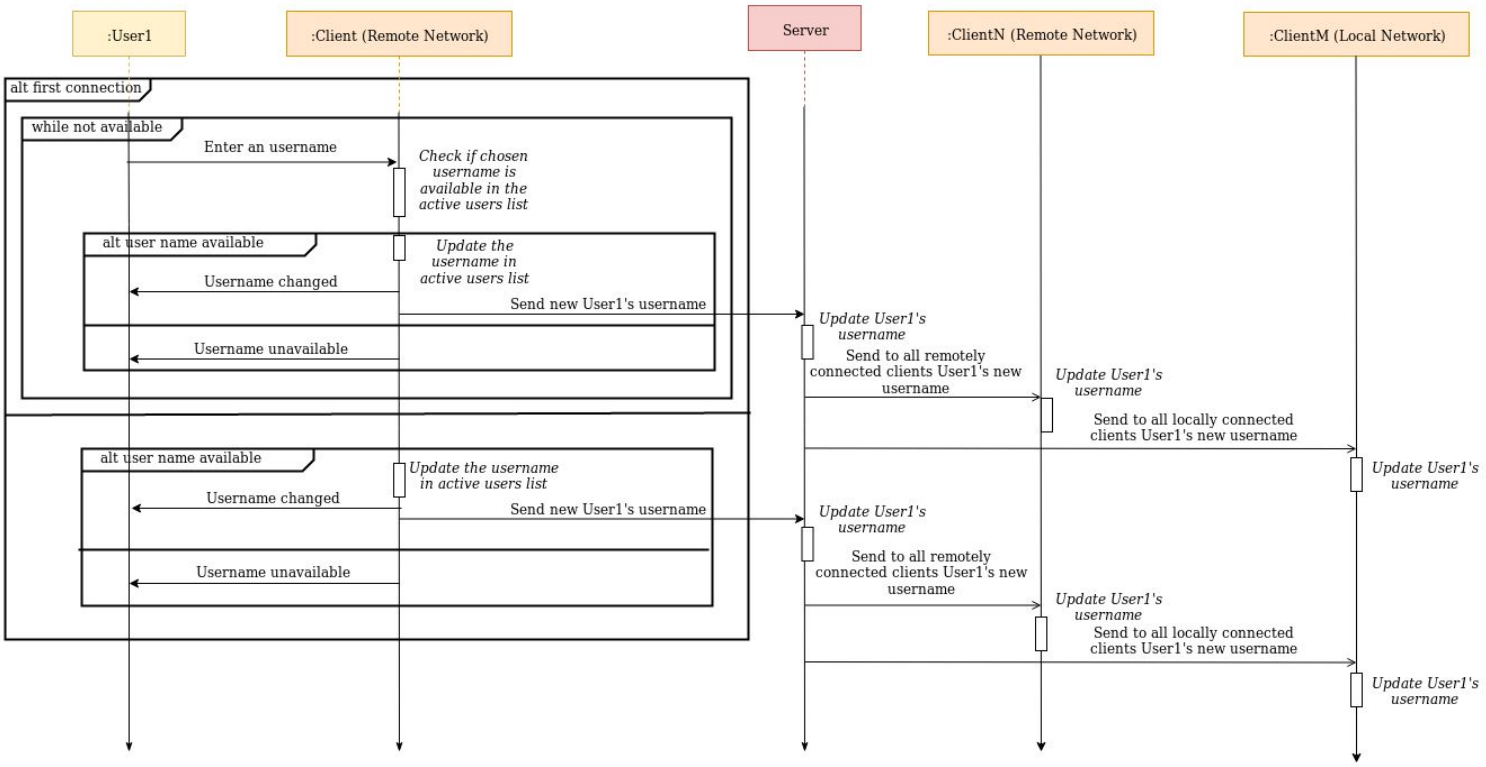
[CdC-Bs-25] " *Le temps d'apparition des utilisateurs au sein de la liste des utilisateurs pour lesquels l'agent est actif ne doit pas excéder 5 secondes* "

[CdC-Bs-8] " *Le système doit permettre à l'utilisateur d'identifier simplement l'ensemble des utilisateurs pour lesquels l'agent est actif sur le réseau* "

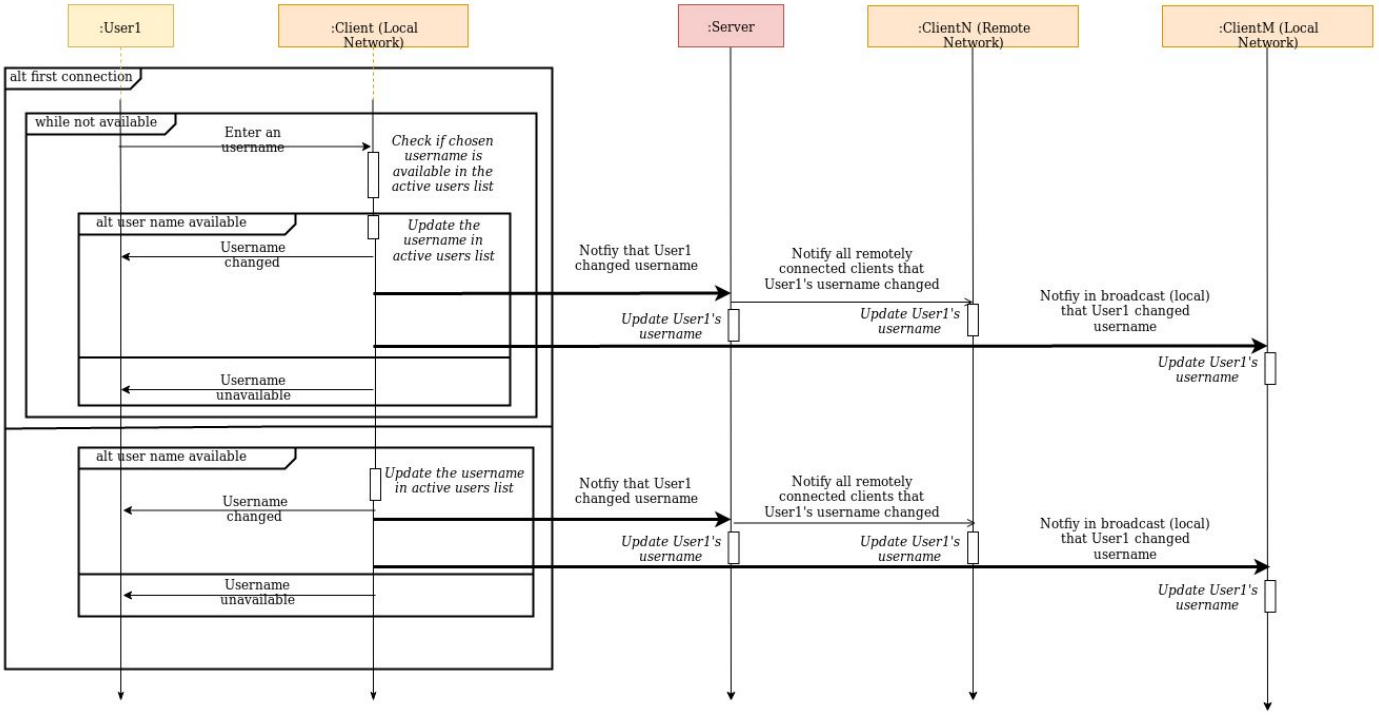
Set Username for a remotely connected client

Puisque la liste des utilisateurs actifs est à jour en temps réel, lors du changement de pseudo, l'agent (client) peut directement se référer à cette même liste pour vérifier l'unicité du dit-pseudo entré par l'utilisateur.

Puis, de la même manière que pour notifier la connexion d'un utilisateur, l'agent à distance prévient (seulement) le serveur pour qu'il puisse à son tour tout d'abord modifier sa liste des utilisateurs actifs en changeant le pseudo pour ensuite en notifier tous les autres agents.



Set Username for a locally connected client



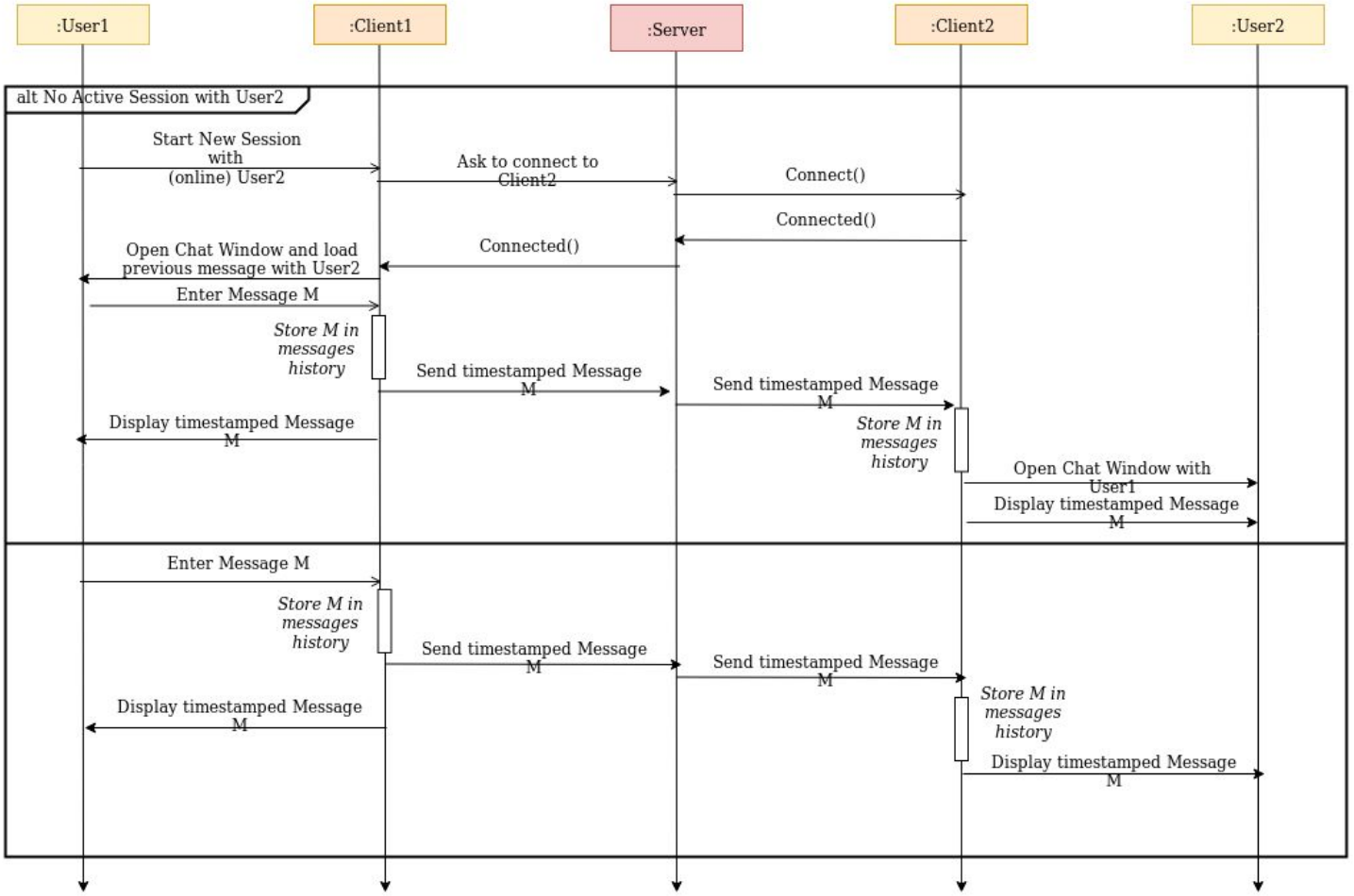
Dans le cas d'un agent connecté sur le réseau local, en plus d'informer le serveur (qui ne préviendra que les agents distants), celui-ci enverra en broadcast la même information pour tenir à jour les agents du réseau local.

Extrait du cahier des charges :
[CdC-Bs-7] "*Le système doit permettre à l'utilisateur de choisir un pseudonyme avec lequel il sera reconnu dans ses interactions avec le système"*
[CdC-Bs-10] *"Le système doit garantir l'unicité du pseudonyme des utilisateurs pour lesquels l'agent est actif sur le réseau"*
[CdC-Bs-16] *"Le système doit permettre à l'utilisateur de changer le pseudonyme qu'il utilise ausein du système de clavardage à tout moment"*
[CdC-Bs-17] *"Lorsqu'un utilisateur change de pseudonyme, l'ensemble des autres utilisateurs du système en sont informés"*
[CdC-Bs-18] "*Le changement de pseudonyme par un utilisateur ne doit pas entraîner la fin des sessions de clavardage en cours au moment du changement de pseudonyme"*
[CdC-Bs-24] *"Lorsque la vérification de l'unicité du pseudonyme de l'utilisateur échoue, l'utilisateur doit en être informé dans une période ne dépassant pas 3 secondes"*
[CdC-Bs-20] *"Le changement de pseudonyme d'un utilisateur doit être visible de l'ensemble des autres utilisateurs dans un temps inférieur à 20 secondes"*

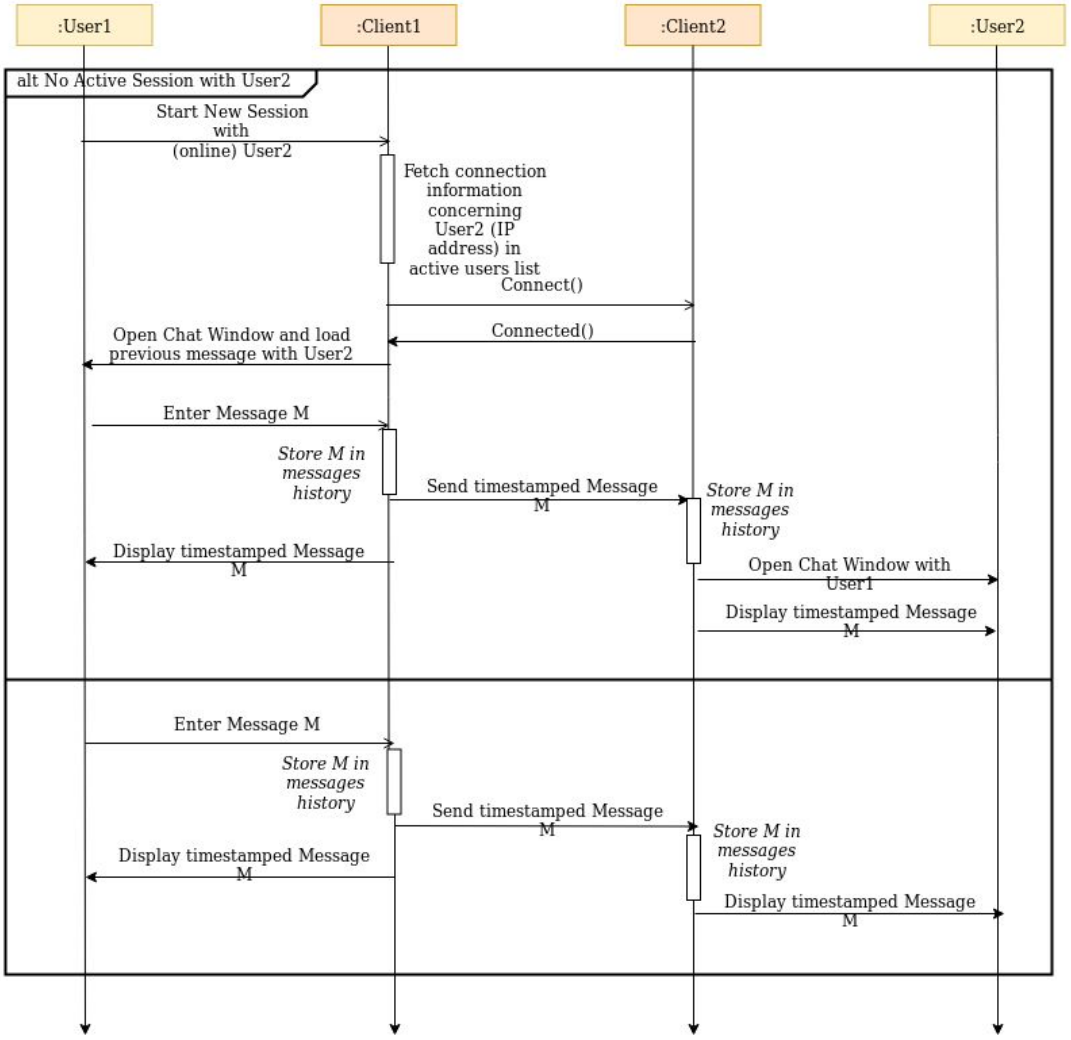
Send a message from a remotely connected client

Concernant les sessions de clavardage d'agent à agent, à partir du moment où un des deux agent (au moins) est distant au réseau local, l'ouverture d'une telle session et donc la connexion de ces deux agents devient complexe. En effet, tous les agents n'ayant que des adresses IP non routables (privées), il est impossible, dans la liste des utilisateurs actifs par exemple, de stocker des informations permettant à un agent de se connecter directement à un autre agent. Pour cela, nous proposons la méthode suivante :

1. Quand un agent se connecte, il ouvre une connexion directe avec le serveur distant (adresse IP publique), peu importe qu'il soit dans le réseau local ou non.
2. Lorsqu'un agent souhaite se connecter à un autre agent dans le cas où l'un des deux (au moins) est à distance, il demande au serveur d'établir une session de clavardage.
3. Le serveur fera l'intermédiaire entre tous les messages du premier l'agent vers le deuxième.



Send a message from a locally connected client



Lorsque l'agent est sur le réseau local, et qu'il souhaite se connecter à un agent sur ce même réseau, il pourrait, lui, se référer à l'adresse IP privée de son destinataire puisqu'ils appartiennent au même réseau.

Extrait du cahier des charges :

[CdC-Bs-12] "*L'horodatage de chacun des messages reçus par un utilisateur sera accessible à celui-ci de façon simple.*"

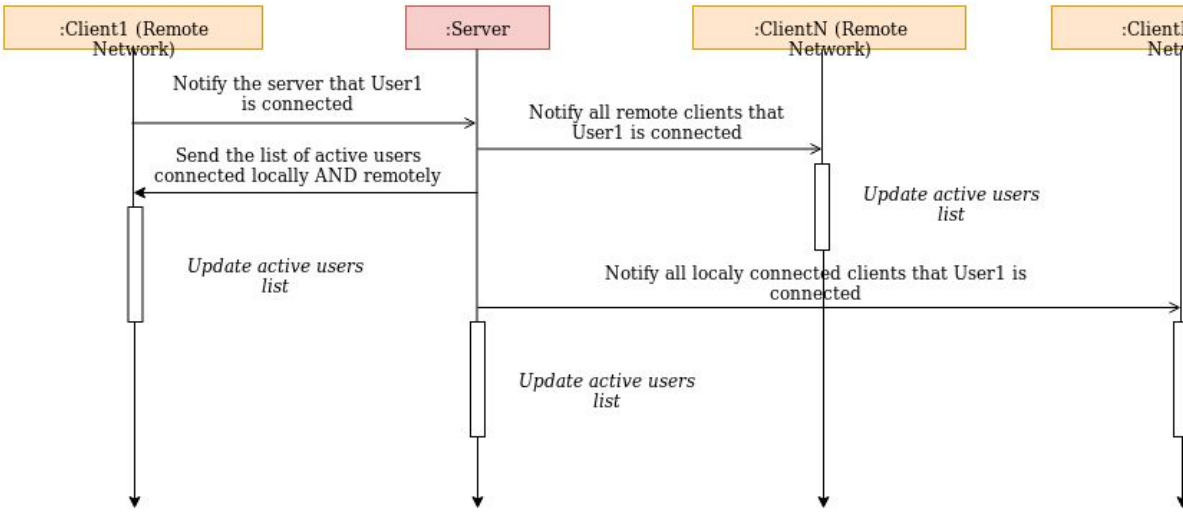
[CdC-Bs-14] "*Lorsqu'un utilisateur démarre un nouvelle session de clavardage avec un utilisateur avec lequel il a préalablement échangé des données par l'intermédiaire du système, l'historique des messages s'affiche."*

[CdC-Bs-21] *"Le temps écoulé entre l'envoi d'un message par un utilisateur et la réception par un autre utilisateur ne doit pas excéder 1 seconde."*

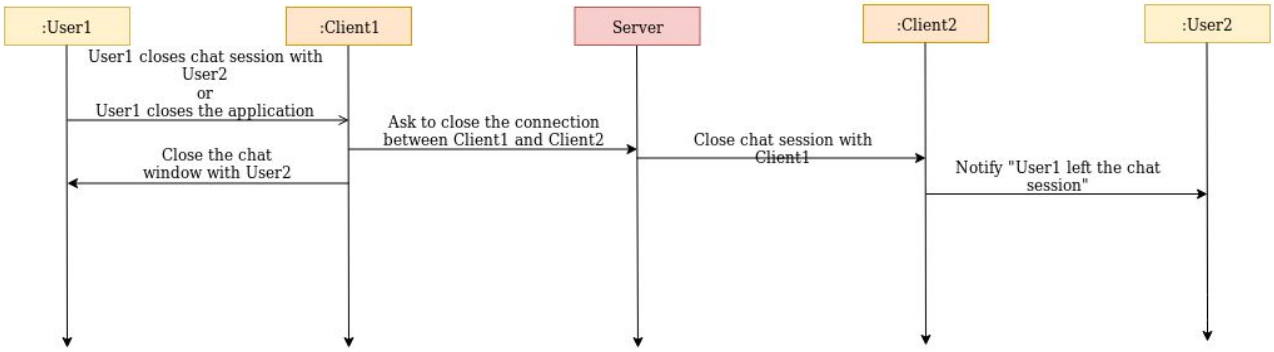
[CdC-Bs-9] *"Le système doit permettre à l'utilisateur de démarrer une session de clavardage avec un utilisateur du système qu'il choisira dans la liste des utilisateurs pour lesquels l'agent est actif sur le réseau."*

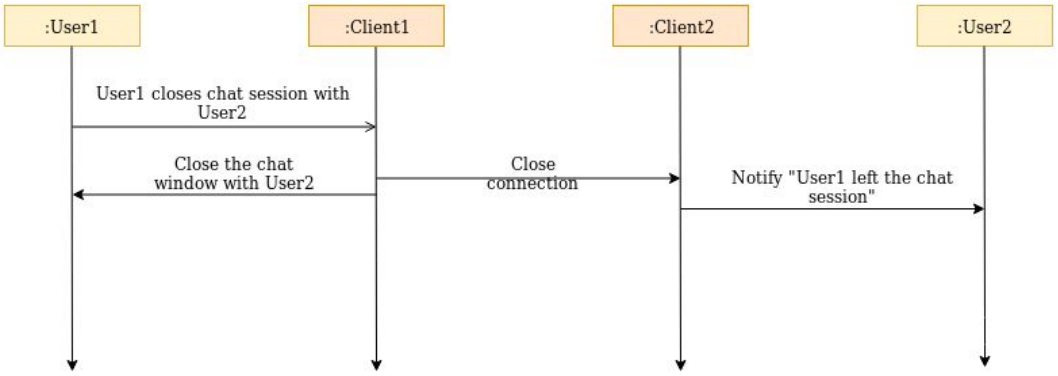
[CdC-Bs-11] *"Tous les messages échangés au sein d'une session de clavardage seront horodatés"*

End a session from a remotely connected client



End a session from a locally connected client

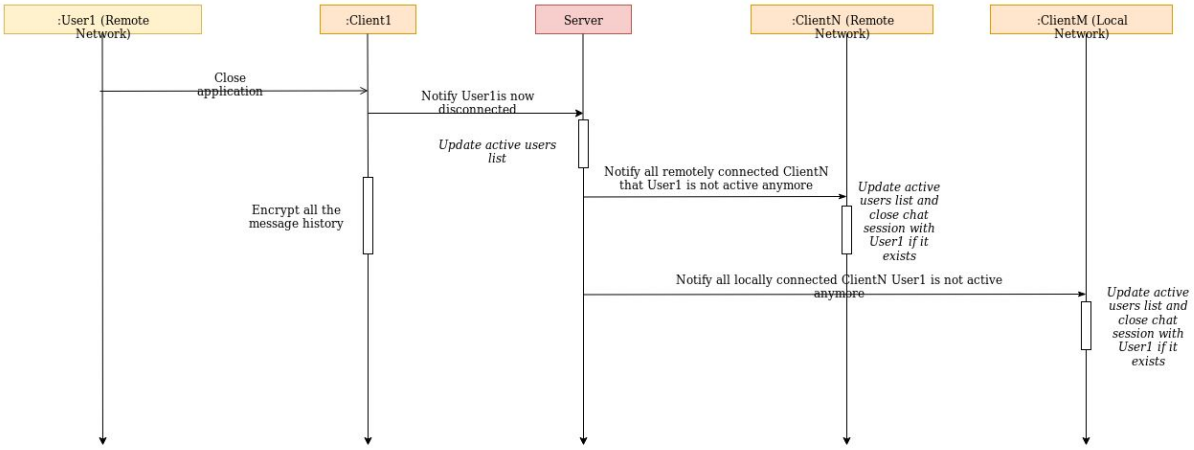




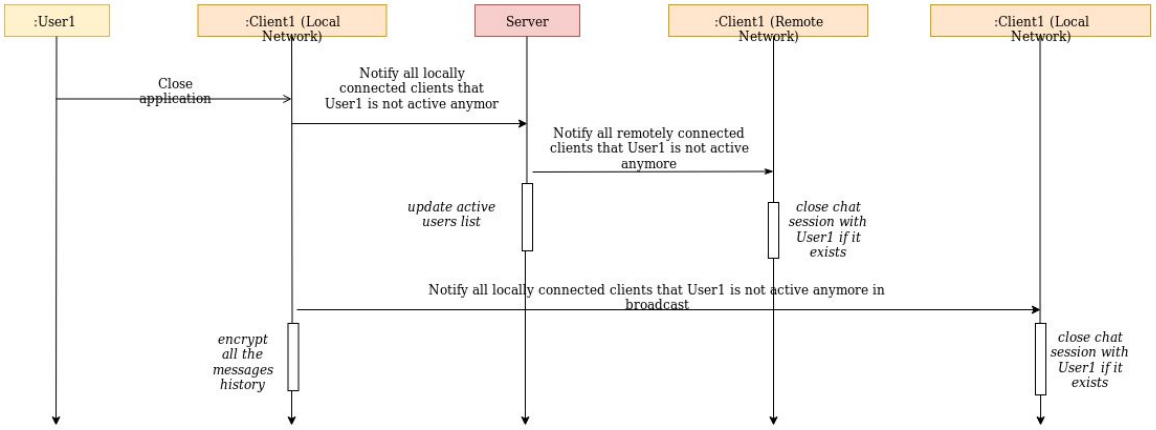
Extrait du cahier des charges :

[CdC-Bs-13] *"Un utilisateur peut mettre unilatéralement fin à une session de clavardage"*

Quit a remotely connected client



Quit a locally connected client



Conclusion des structures de données

Liste des utilisateurs actifs :

Les agents et le serveur distant conserveront une liste des utilisateurs actifs comportant les informations suivantes : *Signature de clé, Pseudo actuel, Adresse IP*.

La signature de clé d'authentification de chaque utilisateur est unique, invariante et publique et permet de certifier l'identité des utilisateurs (et donc de ne pas se baser sur leur pseudo ou leur adresse IP qui eux peuvent être variants). Le champs Adresse IP pourra être instancié à une certaine valeur "None" par exemple, pour préciser que la connexion directe à cet agent est impossible (car distant du réseau local) et qu'elle doit se faire par l'intermédiaire du serveur.

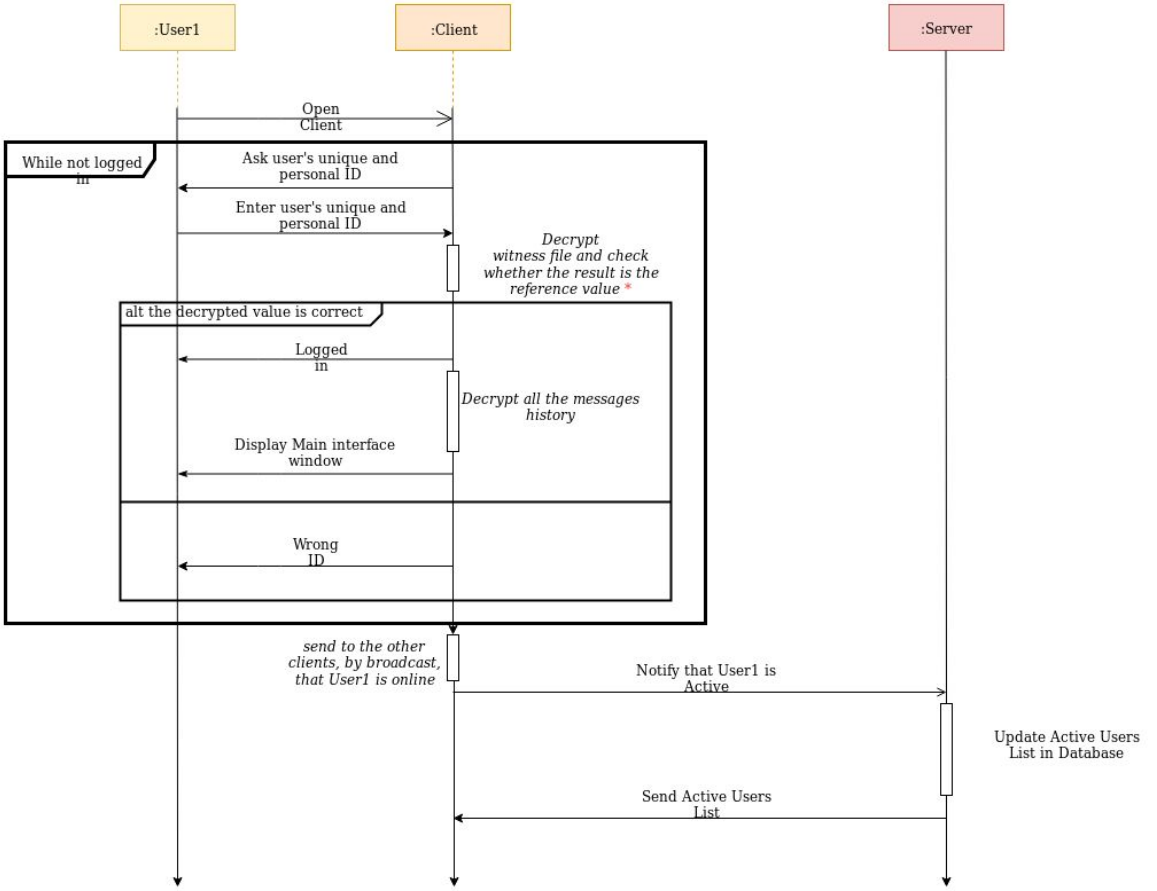
Historique des messages :

Les agents maintiendront un historique des messages horodatés, local à la machine sur laquelle ils s'executeront. Cet historique pourra se traduire par exemple par un format XML. Chaque conversation avec un utilisateur sera alors sauvegardée en associant

tous les messages à la signature de la clé de chiffrement de l'utilisateur destinataire. Il sera chiffré (lors de la déconnexion) et déchiffré (lors de la connexion) par la clé d'authentification de l'utilisateur associé à cet agent.

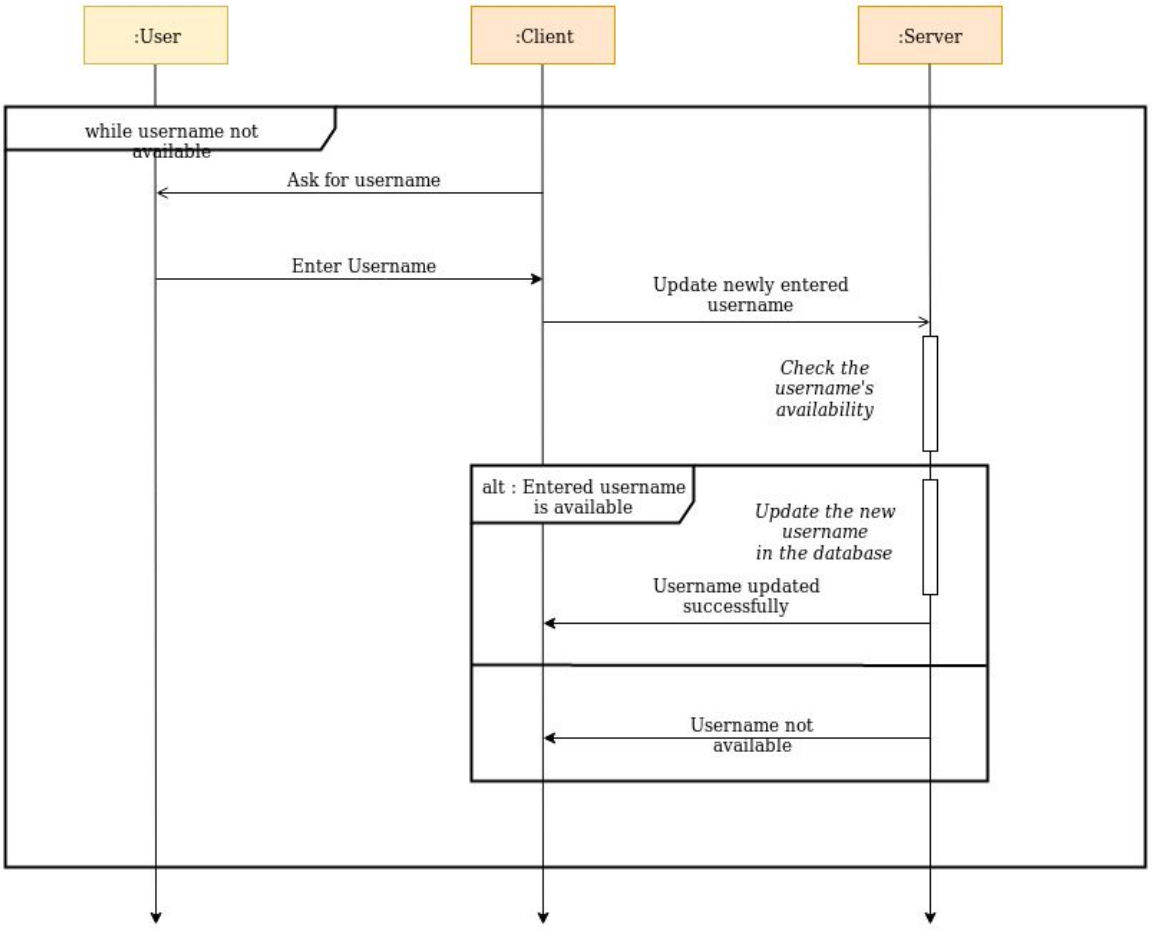
Diagramme de séquences architecture 3-tiers forte

User login

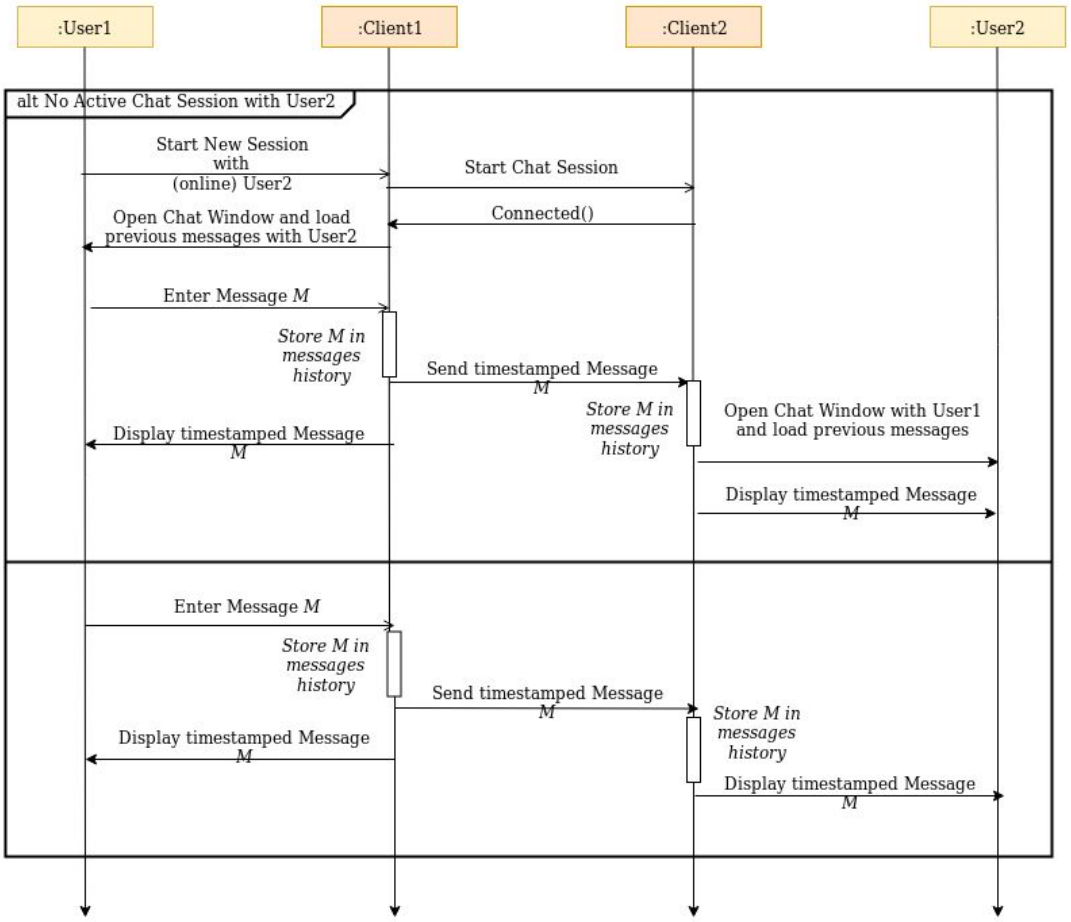


*Each Client has a witness file : Basically a file containing the characters sequence (e.g. "WITNESS"). This file is encrypted once the client is installed by the administrator with the user's unique and personal ID.
At connection time, the user is asked to provide its ID. This latter one is used to decrypt this same witness file. If the resulting character sequence is the same as the initial one ("WITNESS" in our example), the connection succeed. Otherwise the user is invited to provide the correct ID again.

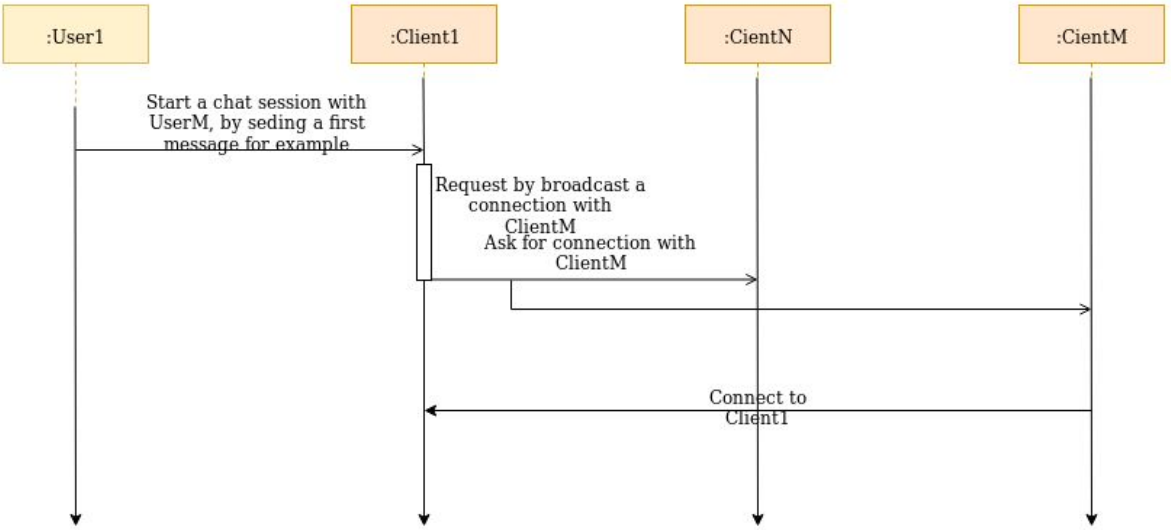
Set a new username



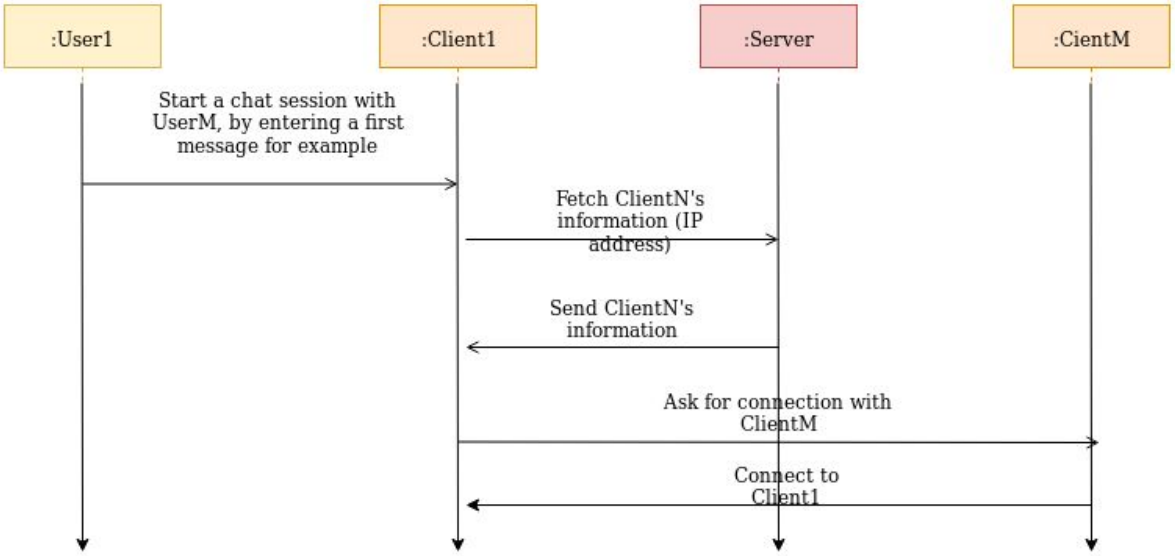
Open a chat session



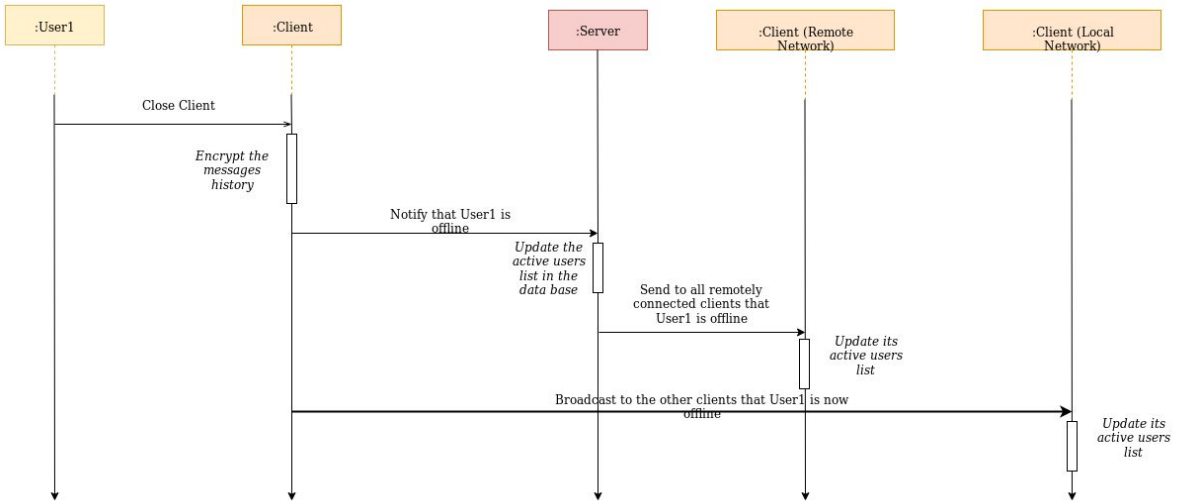
Start a chat session from a locally connected client



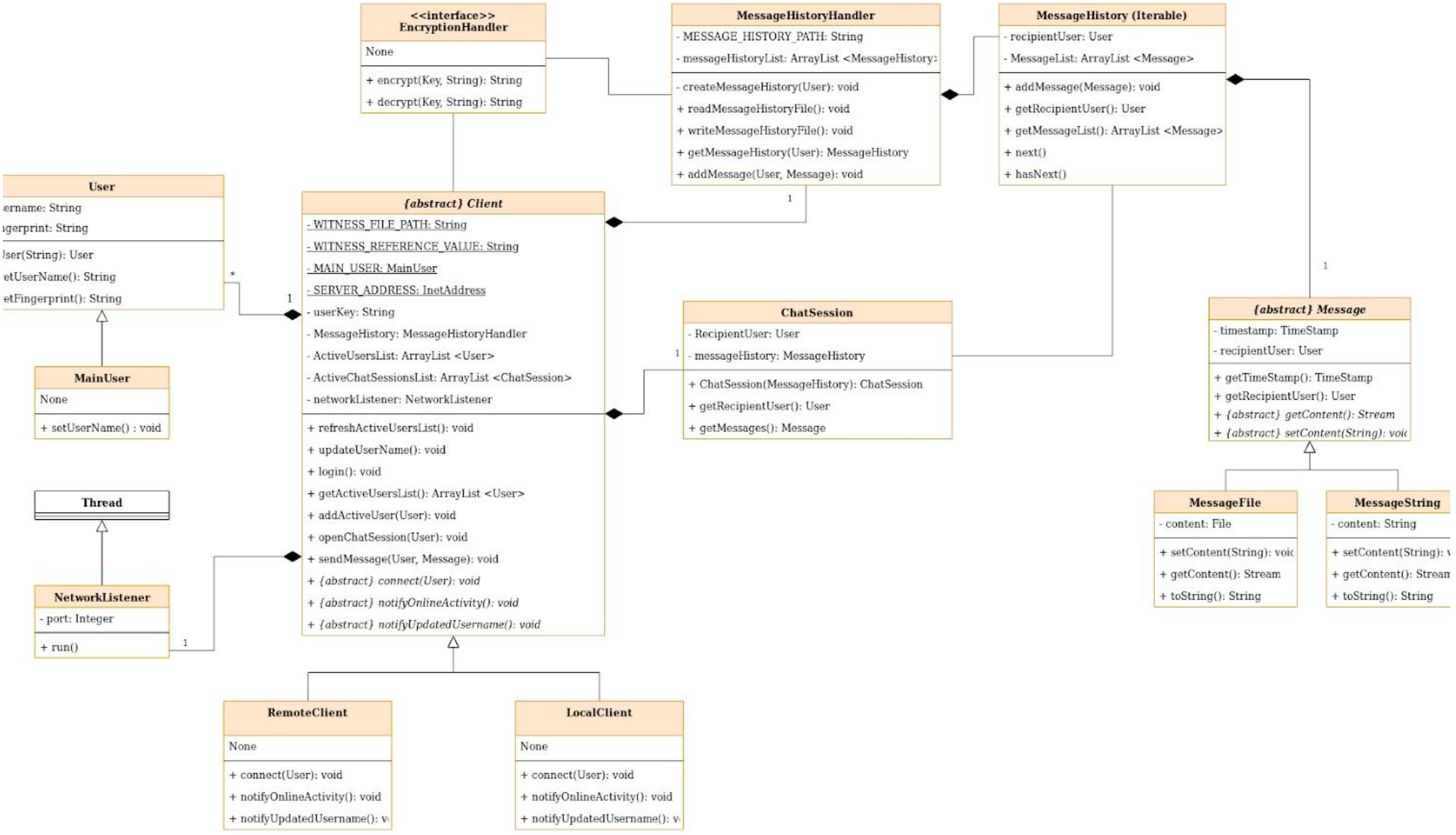
Start a chat session from a network connected client



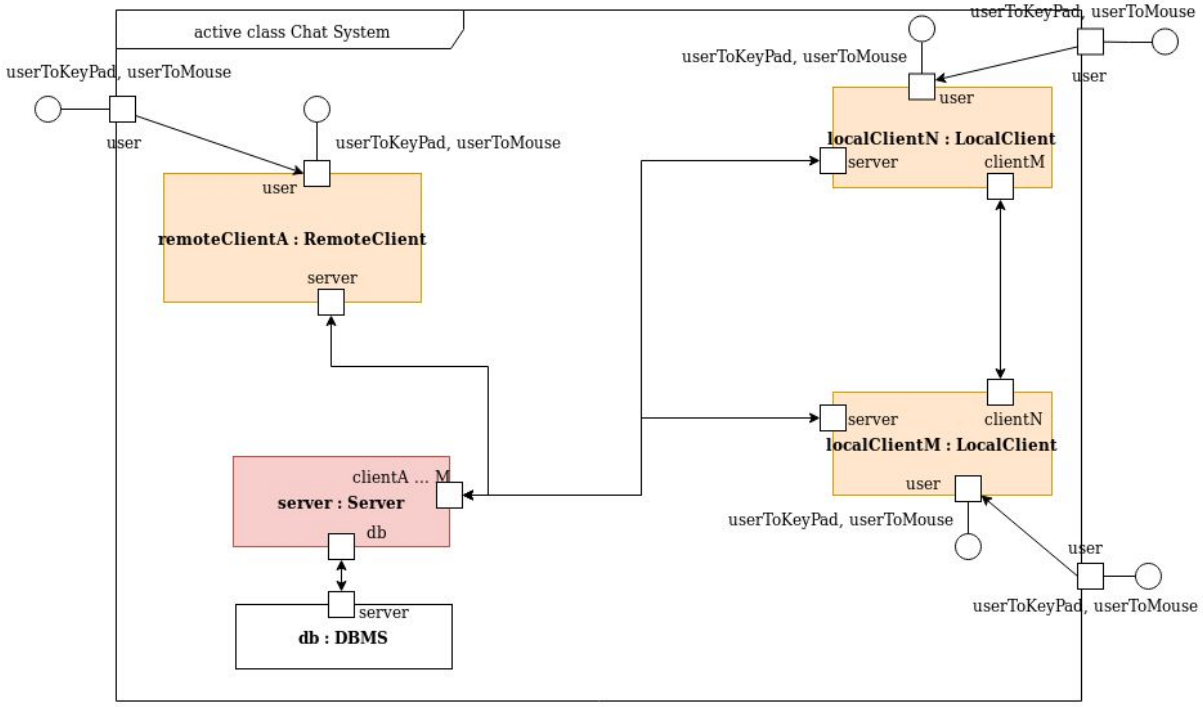
Disconnection

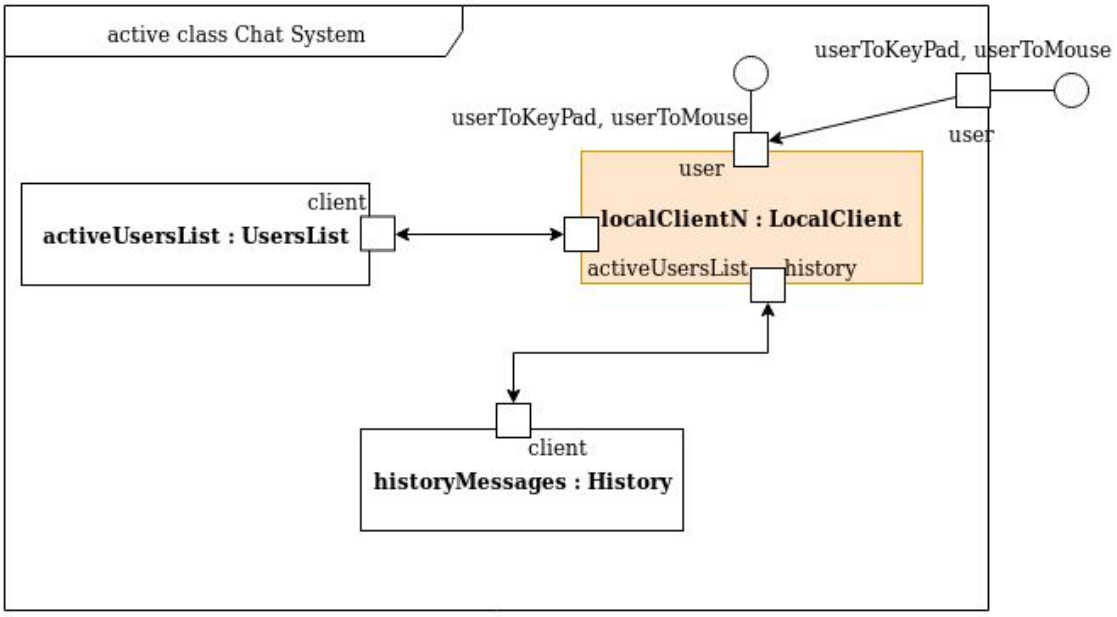


Class Diagram

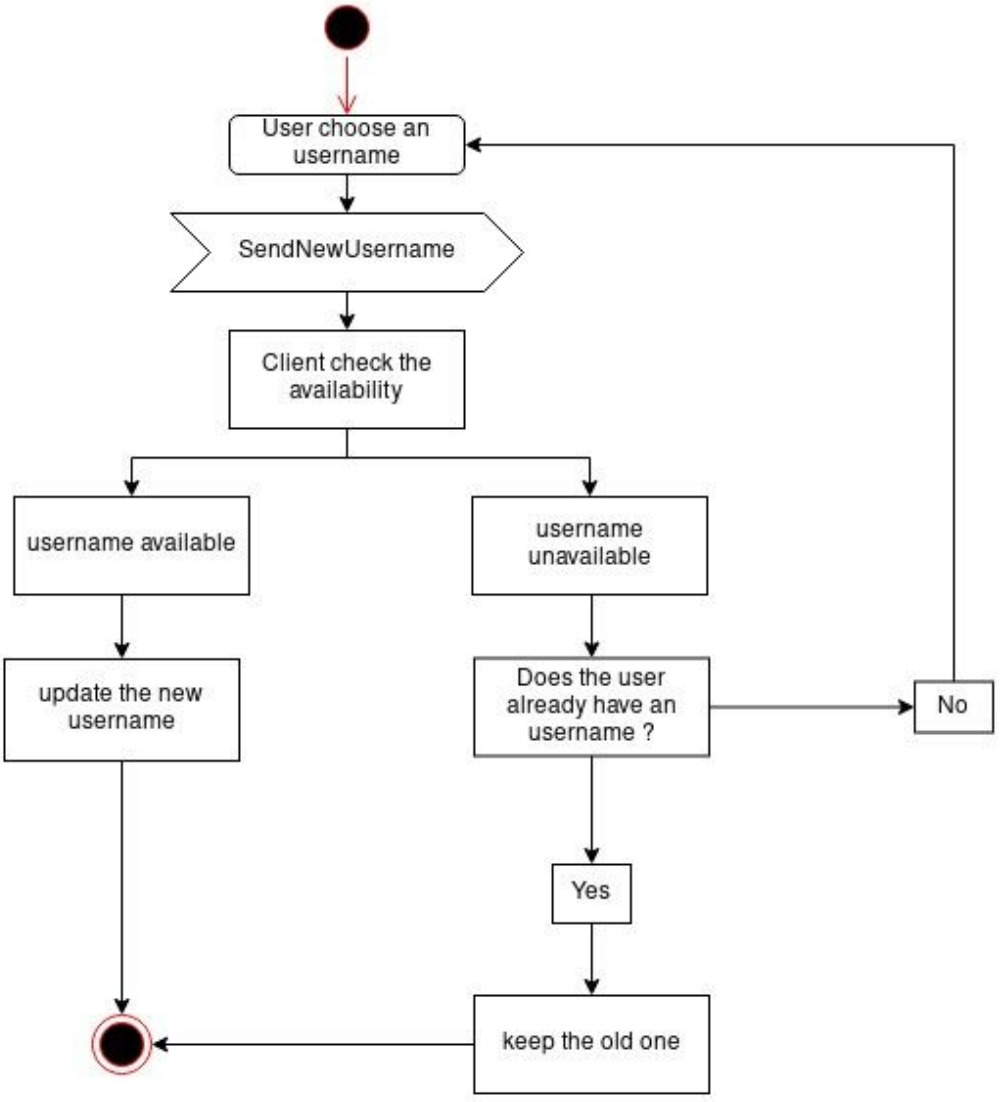

<<interface>>
EncryptionHandler
None
+ encrypt(Key, String): String
+ decrypt(Key, String): String
MessageHistoryHandler
- MESSAGE_HISTORY_PATH: String
- messageHistoryList: ArrayList <MessageHistory>
- createMessageHistory(User): void
+ readMessageHistoryFile(): void
+ writeMessageHistoryFile(): void
+ getMessageHistory(User): MessageHistory
+ addMessage(User, Message): void
MessageHistory (Iterable)
- recipientUser: User
- MessageList: ArrayList <Message>
+ addMessage(Message): void
+ getRecipientUser(): User
+ getMessageList(): ArrayList <Message>
+ next()
+ hasNext()
User
ername: String
gerprint: String
ser(String): User
etUserName(): String
etFingerprint(): String
{abstract} Client
- WITNESS_FILE_PATH: String
- WITNESS_REFERENCE_VALUE: String
- MAIN_USER: MainUser
- SERVER_ADDRESS: InetAddress
- userKey: String
- MessageHistory: MessageHistoryHandler
- ActiveUsersList: ArrayList <User>
- ActiveChatSessionsList: ArrayList <ChatSession>
- networkListener: NetworkListener
+ refreshActiveUsersList(): void
+ updateUserName(): void
+ login(): void
+ getActiveUsersList(): ArrayList <User>
+ addActiveUser(User): void
+ openChatSession(User): void
+ sendMessage(User, Message): void
+ {abstract} connect(User): void
+ {abstract} notifyOnlineActivity(): void
+ {abstract} notifyUpdatedUsername(): void
ChatSession
- RecipientUser: User
- messageHistory: MessageHistory
+ ChatSession(MessageHistory): ChatSession
+ getRecipientUser(): User
+ getMessages(): Message
{abstract} Message
- timestamp: TimeStamp
- recipientUser: User
+ getTimeStamp(): TimeStamp
+ getRecipientUser(): User
+ {abstract} getContent(): Stream
+ {abstract} setContent(String): voi
MainUser
None
+ setUserName() : void
Thread
NetworkListener
- port: Integer
+ run()
MessageFile
- content: File
+ setContent(String): voi
+ getContent(): Stream
+ toString(): String
MessageString
- content: String
+ setContent(String):
+ getContent(): Strea
+ toString(): String
RemoteClient
None
+ connect(User): void
+ notifyOnlineActivity(): void
+ notifyUpdatedUsername(): v
LocalClient
None
+ connect(User): void
+ notifyOnlineActivity(): void
+ notifyUpdatedUsername(): v
1
*
1
1
1
1
1
1

Composite Structure Diagrams





States Diagram


User choose an username
SendNewUsername
Client check the availability
username available
username unavailable
update the new username
Does the user already have an username ?
No
Yes
keep the old one