Object Orientend Conception - Technical details and conception documents

Table of contents

Technical details

Overview of the technologies used

- Programming language: Java 8
- GUI: JavaFX
- Network communications: JSON over HTTP - Unirest and Pippo libraries
- Data persistence: BSON (JSON-like no-SQL DB model) - MongoDB and Jongo libraries

Use of the dependencies

We used maven to manage dependencies.

The client use JavaFX to handle the UI, and the libraries Pippo and Unirest for networking.

Both the client and the server use Pippo as an http server to handle incomming http requests, and the Unirest library on the client side to initiate http requests. All data is passed throught REST endpoints as JSON. The serialization/deserialization to and from POJOs (Plain Old Java Objects) is then handled directly by Pippo or with the help of the GSon library for Unirest.

To persist the data between session on the client side, the MongoDB java library and Jongo (a wrapper around Mongo with a mongo-shell like syntax to facilitate database requests). The class that handles the database is Database.

Networking

In p2p mode, the discovery is handled separatly from the rest of the networking by using udp broadcasts on the local network. All clients receiving the broadcast respond directly to the source of the broadcast with a traditional HTTP request to exchange the users info.

In the presence server mode, the server keeps a list of all connected and disconnected users. Each time a new user connect to the presence server, they receive the list of all users and each users is notified of the new user.

All users receive a UUID (Universal Unique IDentifier) the first time they launch the app. This ID is used throught the network to identify the user without any risk of collision (the UUID is generated randomly with a near impossible chance of having twice the same ID).

Each client start an HTTP server wich is used to communicate with other clients.

The presence server also starts an HTTP server.

Client REST API

The API is defined in the NetworkService class as follows:

- `/user`
  - GET: return the local user info
- `/user/connect`
  - POST, PUT: Connect to the user
- `/users`
  - DELETE: Delete yourself from the distant user list
- `/message`
  - POST: Send a message to the user
- `/notify`
  - POST: Notify the user of an username change

Server REST API

The API is defined in the ServerBackend class as follows:

- `/user`
  - GET: Connect to the presence server
  - PUT: Get the list of connected users
  - DELETE: Disconnect from the presence server
- `/user/name`

- POST: Notify the presence server of an username change

GUI

We used JavaFX with SceneBuilder to create the GUI. The controller class is GuiController. It handles event sent by JavaFX.

We paid particular attention to synchronize access to the underlying data : all modifications have to be done in the JavaFX thread.

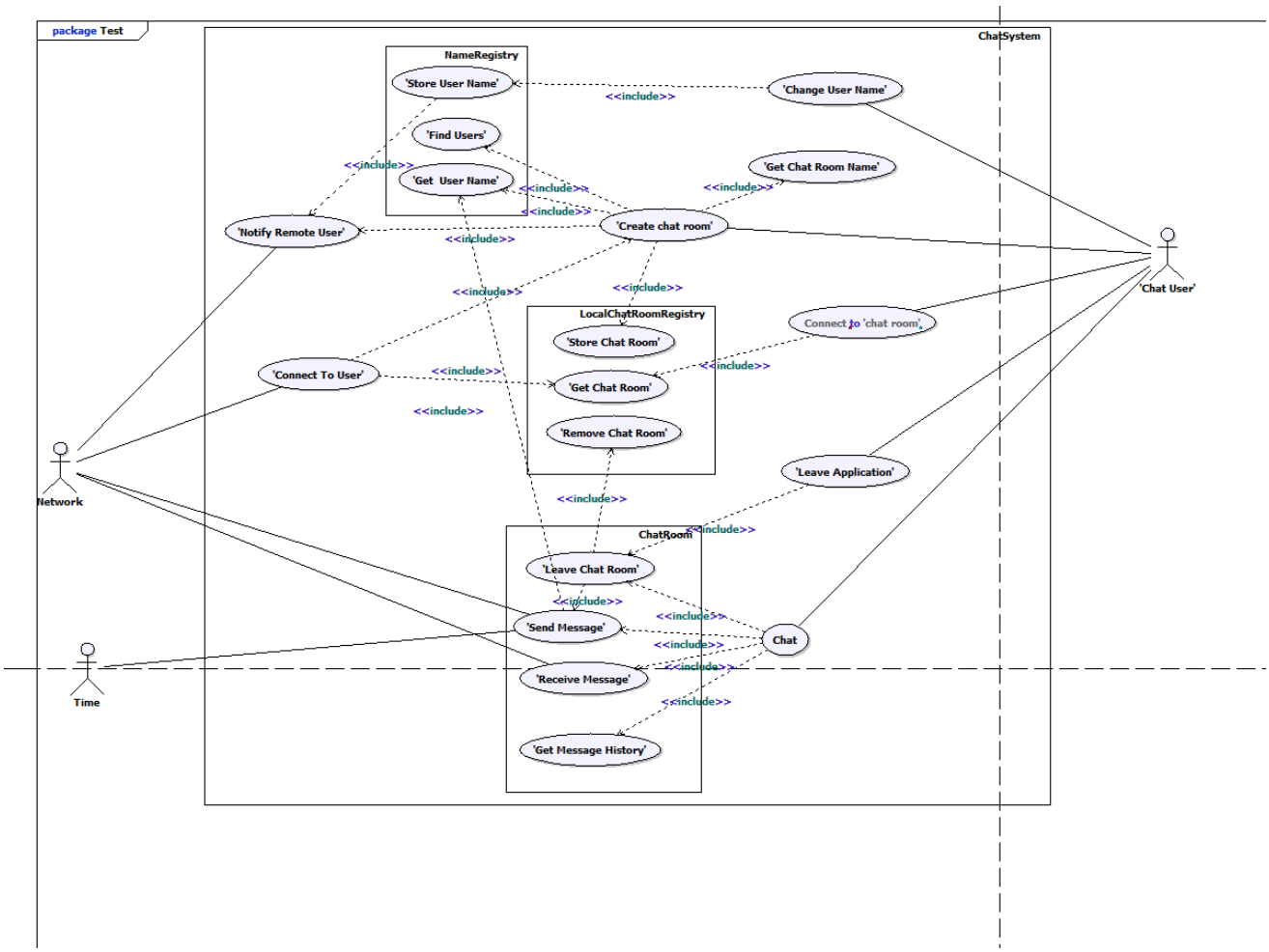
Conception documents

Since those conceptions documents were realised during the implementation they are only relevant to the 1st specification.

The class diagram still represent the global architecture of the application, even if class names were changed and a lot of things changed.

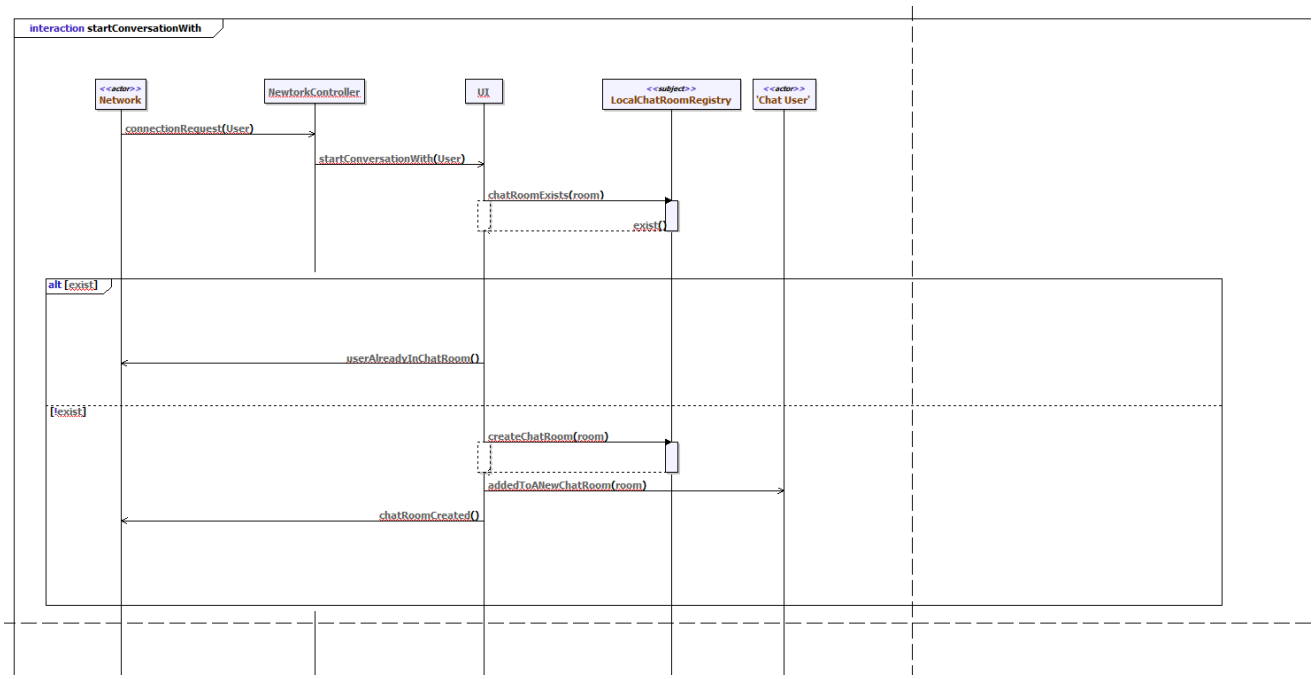
Most sequence diagrams are still relevant.

Use case diagrams

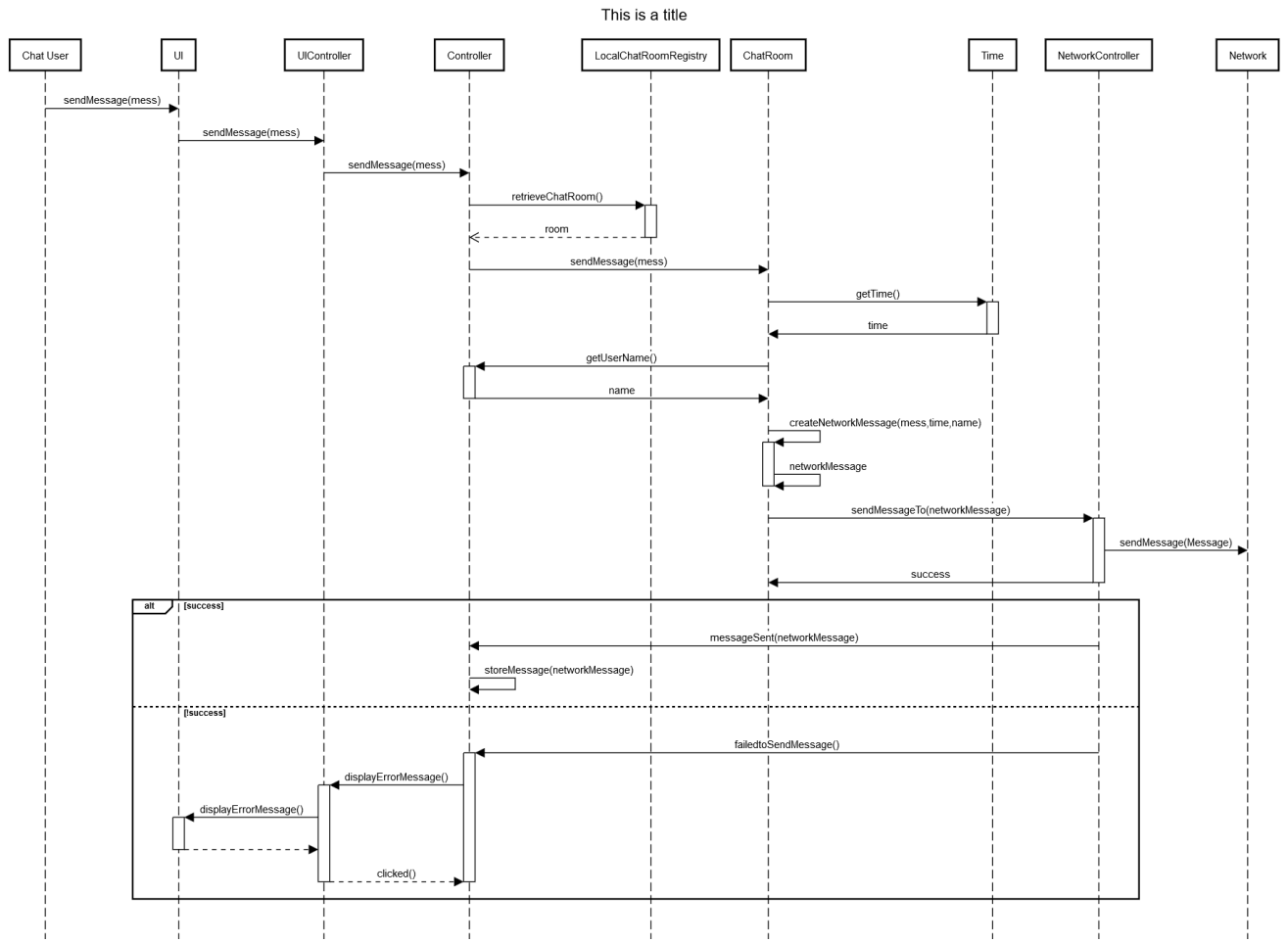


Chat user use-case

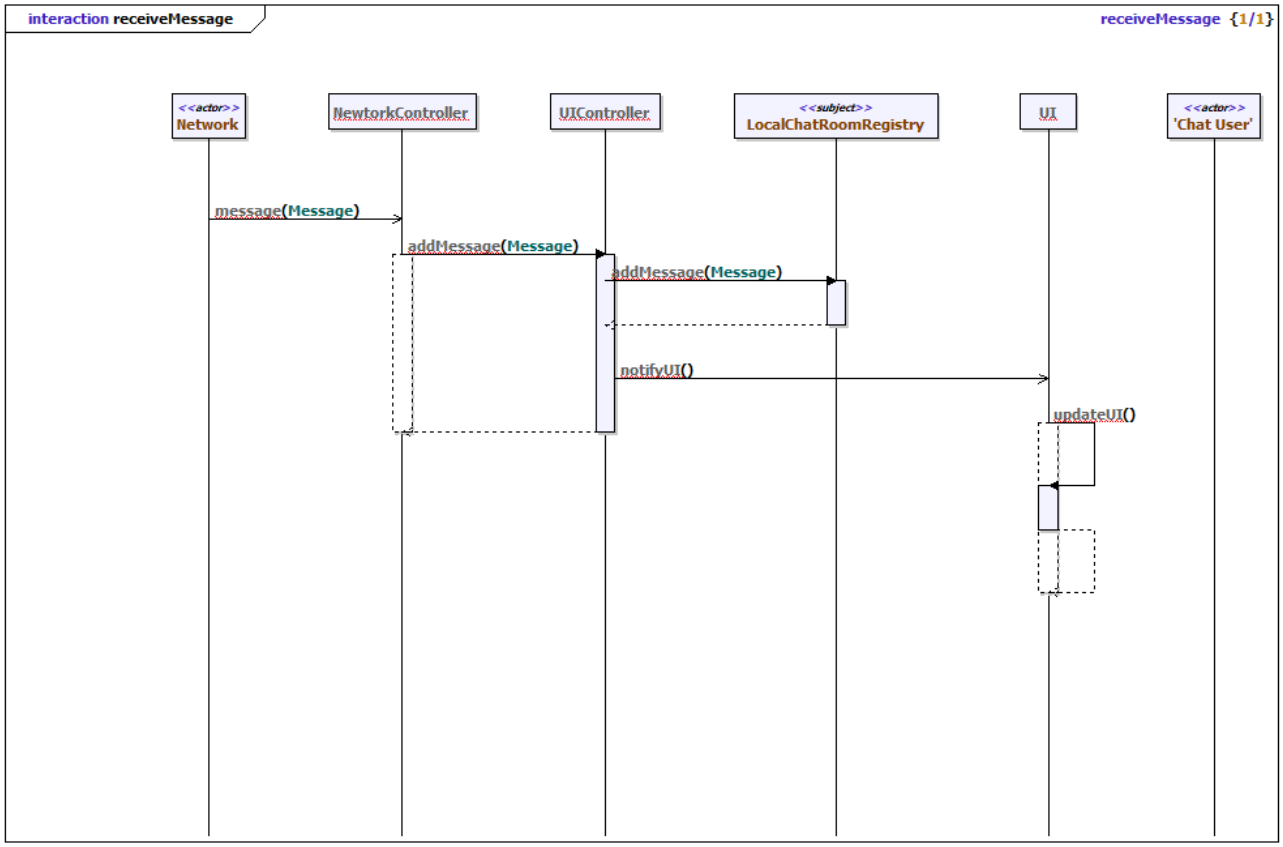
Sequence diagrams



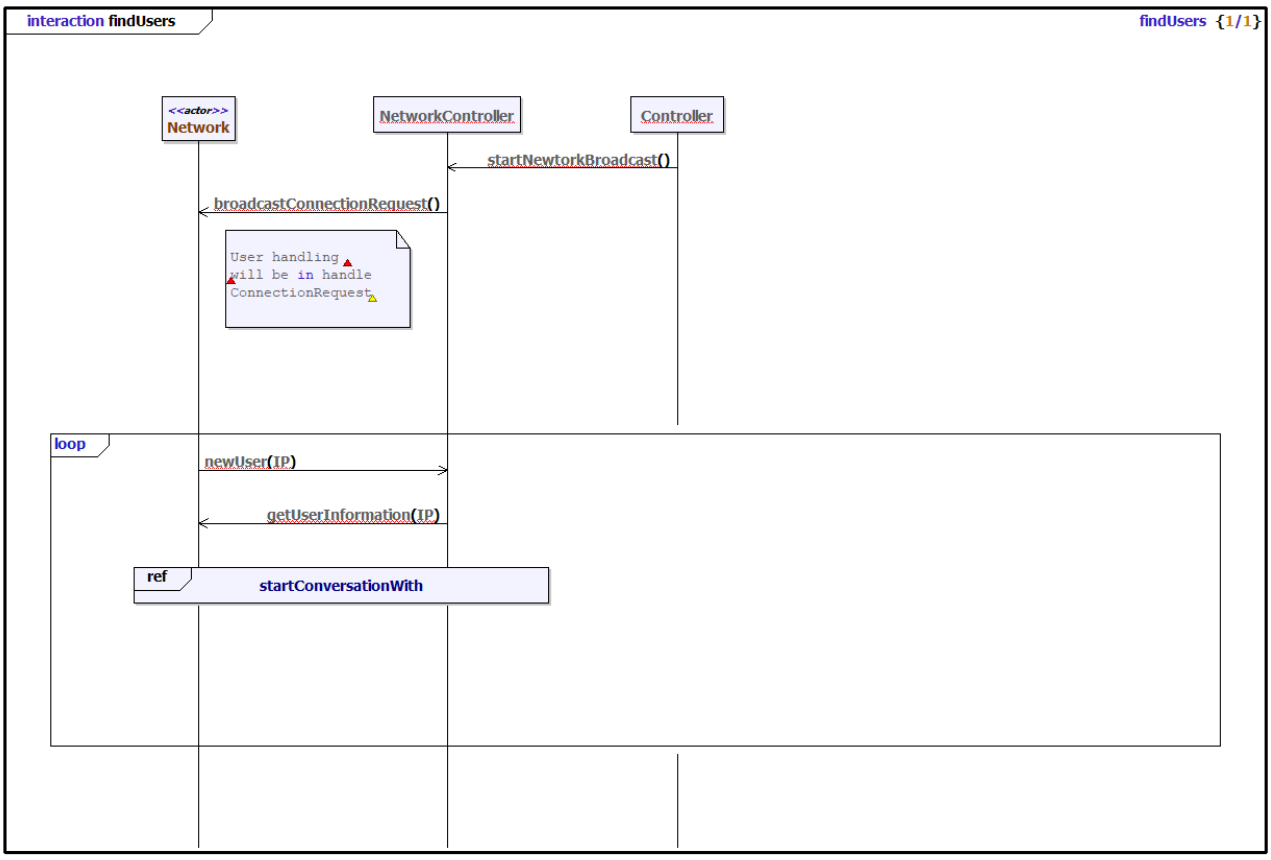
Sequence diagram "Start conversation with"



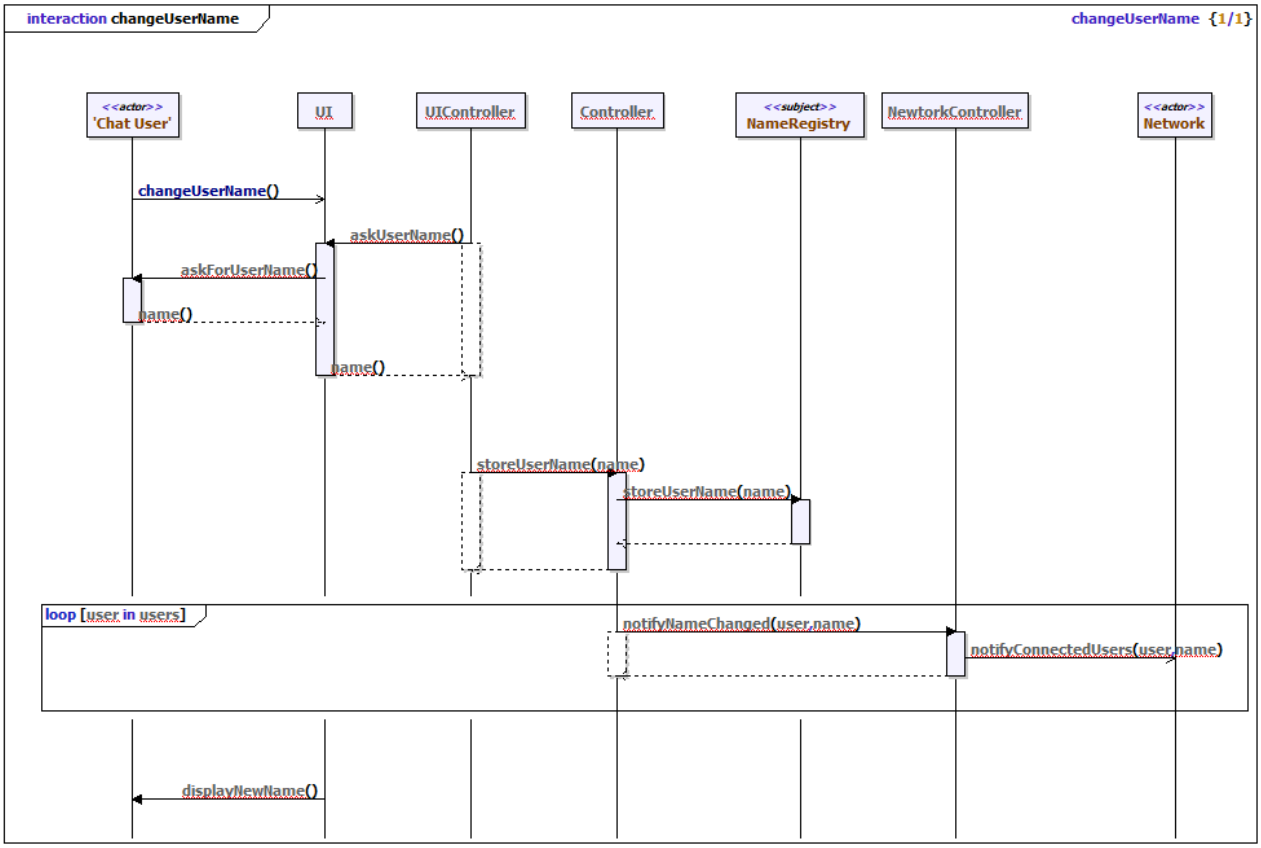
Sequence diagram "Send Message"



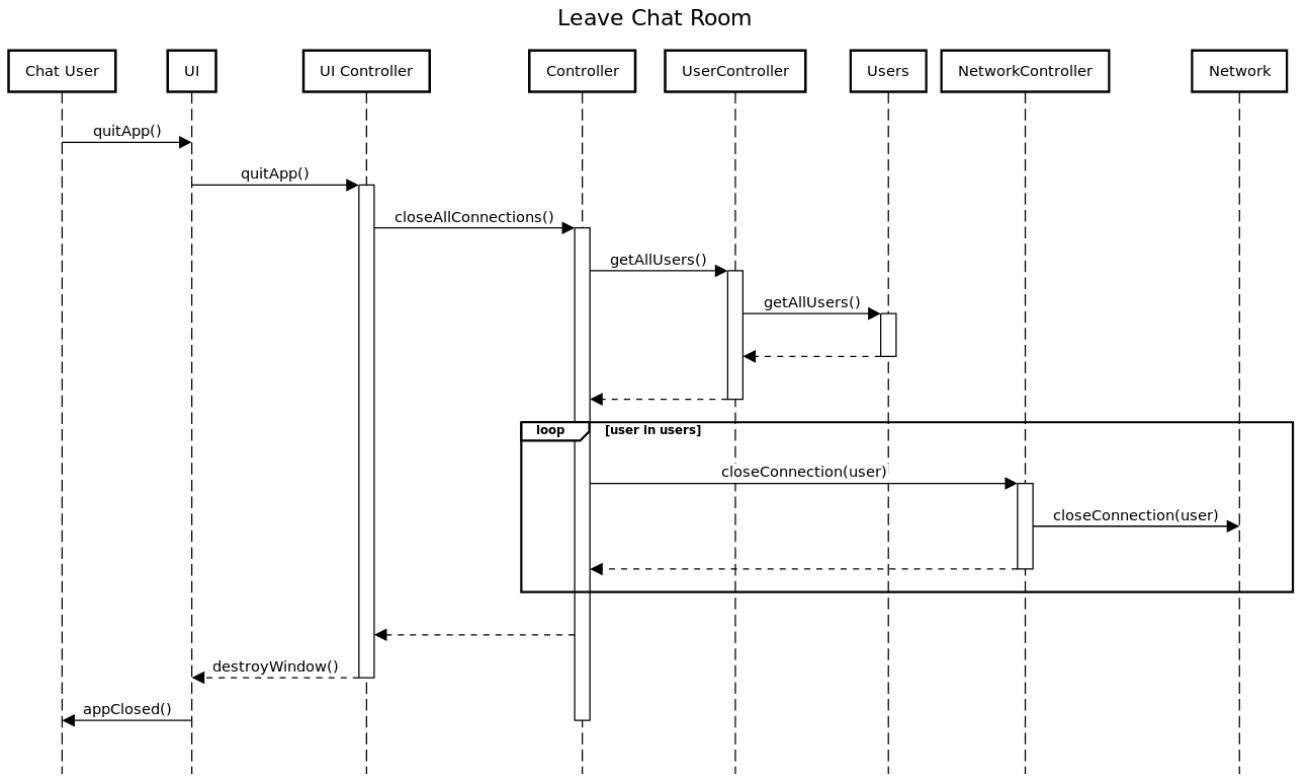
Sequence diagram "Receive Message"



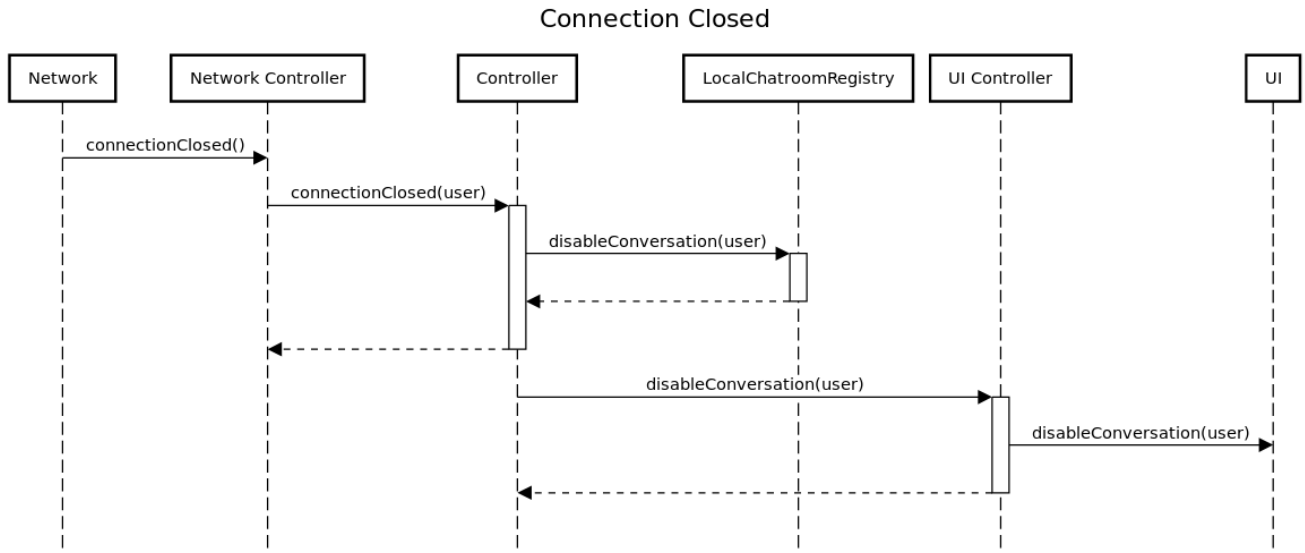
Sequence diagram "Find Users"



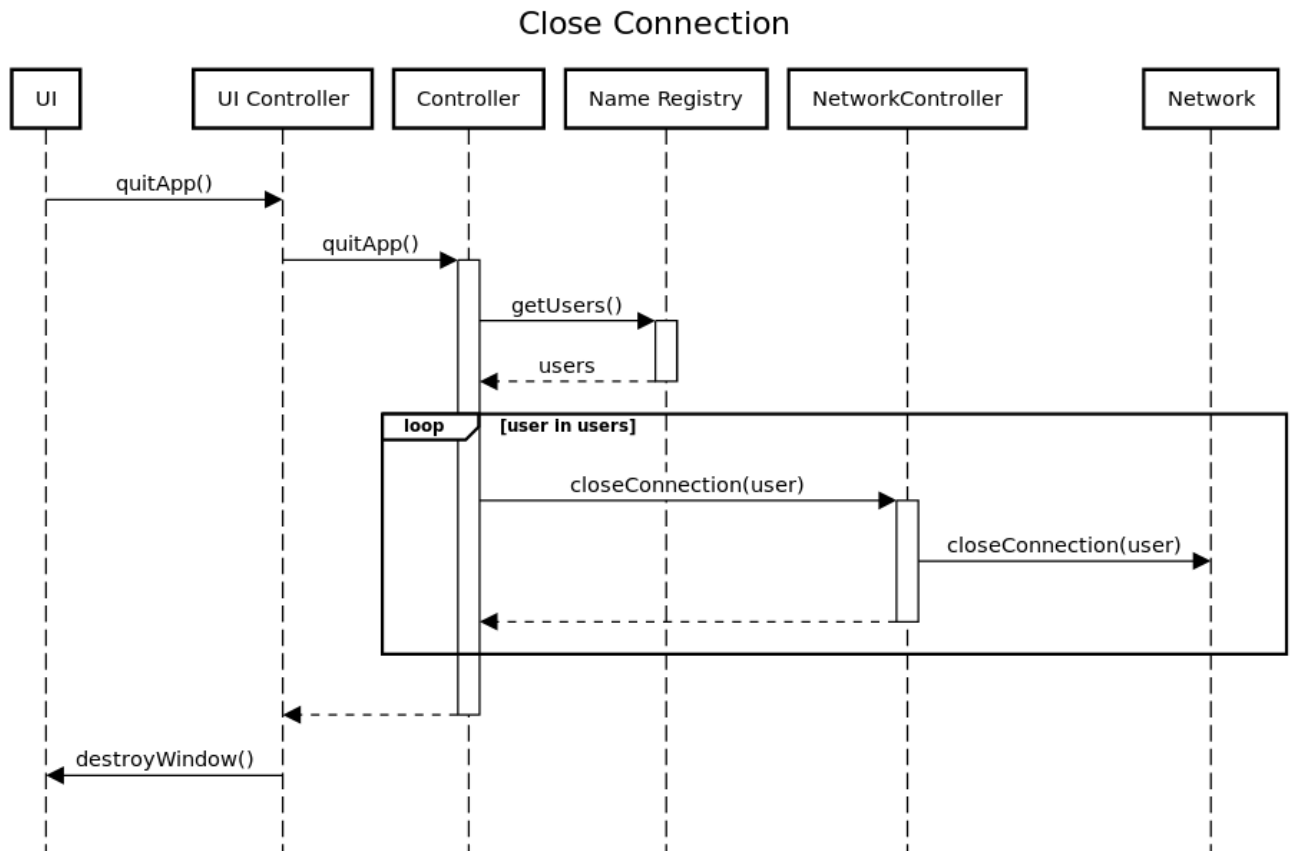
Sequence diagram "Change User Name"



Sequence diagram "Leave Chat Room"



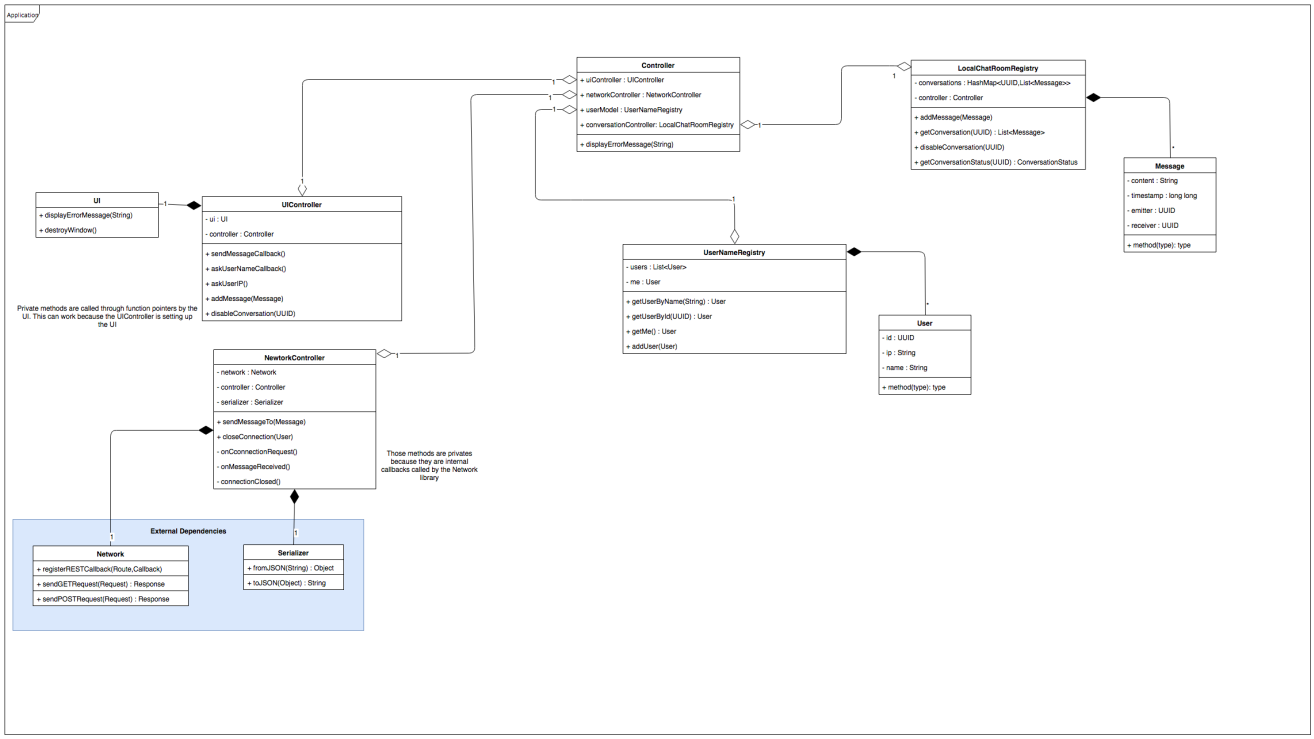
Sequence diagram "Connection Closed"



Sequence diagram "Close connection"

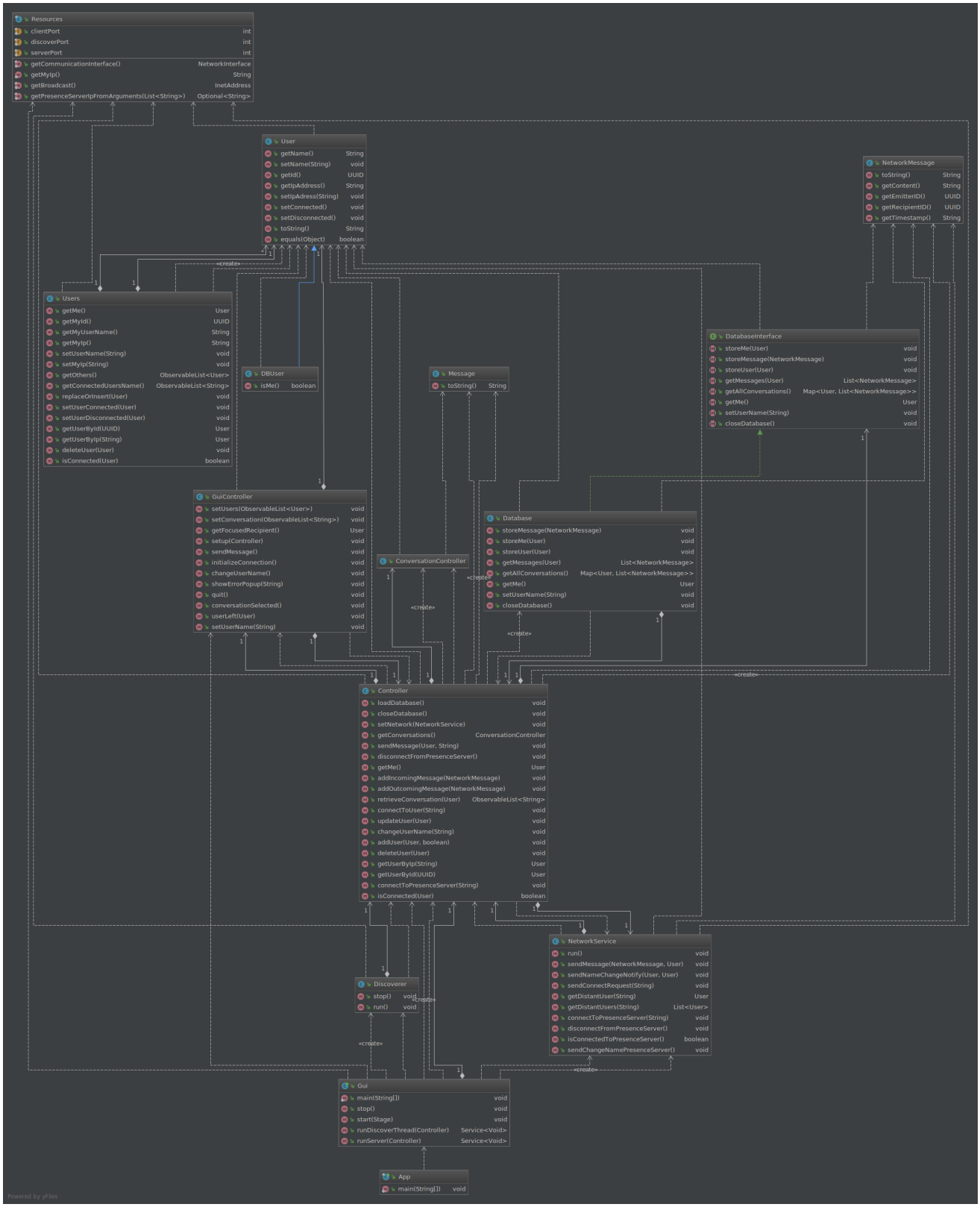
Class diagrams

First design



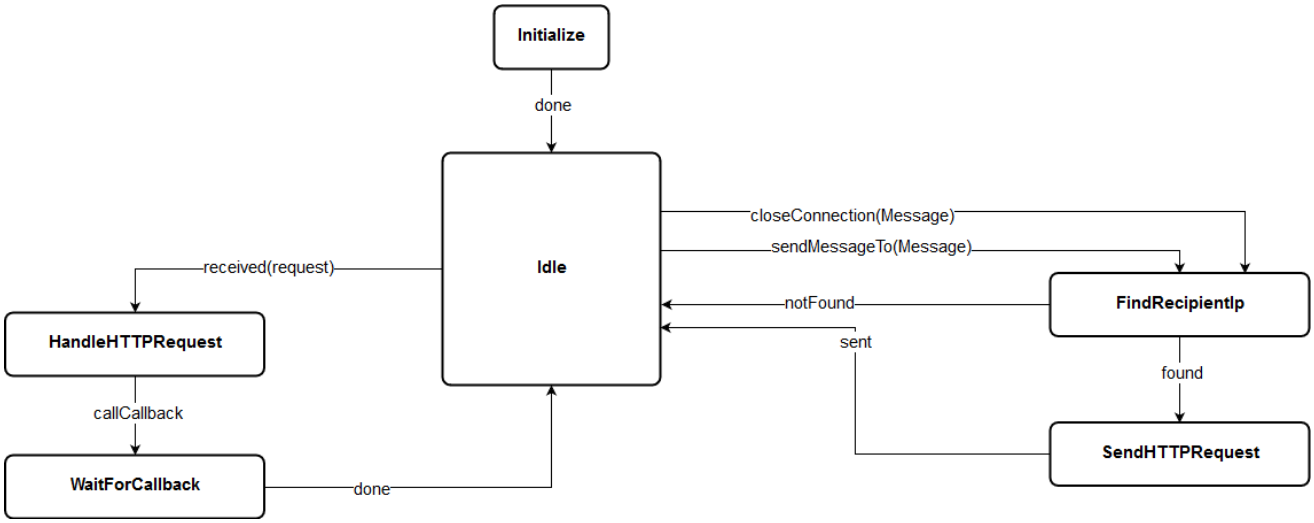
Initial class diagram

Current implementation



Current class diagram implementation

State machine diagrams



State machine diagram for the network controller on the client side