System Design Document for CEYMChat

Carl stling, Erik Gunnarsson, Mohamad Almasri, Yazan Ghafir

09/10-2018

1 Introduction

This is an in-depth documentation for the System Design of CEYMChat, a desktop chatapplication.

1.1 Definitions, acronyms and abbreviations

”Group 21” refers to Erik Gunnarsson, Carl stling, Yazan Ghafir and Mohamad Almasri.
”CEYMChat” refers to the application built by group 21 during this project.
”GUI” refers to a Graphical User Interface.
”.net” refers to the java.net library used for network connections in the project.
”Multithreading” refers to the usage of multiple threads in a program in order to execute several tasks at the same time.
”CLI” refers to ”Command Line Interface”

2 System architecture

2.1 Description of subsystems involved

2.1.1 The Server

All communication will pass through the server. The server will handle all stored data including logged in users, on-going sessions, stored messages. It runs on any machine capable of running a .jar-file and with the necessary ports open to the WAN. The Server is simply stopped by exiting the CLI.

2.1.2 The Client

The Client is the users application that runs on a desktop. It allows the user to chat with another user on a separate machine, anywhere in the world.
The Client is simply stopped by exiting the window.

2.1.3 The "Flow"

Before any interaction between other users can happen, a connection is established to the server. The server will uniquely identify you by the username that you entered upon connecting.
In order to chat to another client, a receiver is selected in the GUI. Thereafter, the message that the sender wishes to send passes through the socket-connection with the server. The server then processes the message and passes it on to the receiver selected. The receiver receives the message, processes it and then prints it to the client GUI.

2.2 Subsystem decomposition

2.2.1 The Server

The server component is composed of a model in which all logic is handled, such as performing commands, finding requested users as well as storing data. Alongside the model there is a sockethandler which is responsible for creating new connections and allowing clients to connect to the server. Lastly the server is also made up of a a list of users, each having their own read and write thread to allow the server to constantly read what users are sending as well as sending back messages to specific users.+ The writer thread is also responsible for sending a list of currently online users to each active client.

2.2.2 The Client

The client is composed of a model, a view, and services. The model is responsible for all logic, the view for representing the program as a GUI and the services are responsible for all I/O connections on the client-side. Alongside this there is a controller connecting the model to the view and services. The controller mainly works as a gateway for GUI events such as keystrokes and mouse-clicks.

2.2.3 Model-Lib

A library consisting of a message class, a command class, a userDisplayInfo class as well as a messageFactory was also implemented. The message class is a generic class consisting of some sort of data, a string containing the senders username as well as a string containing the receiver. The command class was implemented to let the server know that a message was sent to the server itself rather than a user so that a command

such as ”SETUSERNAME” or something similar could be sent to the server without the risk of sending that message on to another user as it may contain personal data. UserDisplayInfo is the item created to send a list of users with GUI items such as profile pictures to clients. Lastly there is a MessageFactory responsible for creating messages in a controlled manner.

2.3 Client

The client component is responsible for displaying and sending messages at run-time as well as keeping track of which user it is representing.

The component can be divided into four packages, Model, View, Controller and Services.

2.3.1 The Model

The Model package holds classes that are responsible for keeping track of the current state of the client application and classes handling the logic needed to complete the clients responsibilities.

2.3.2 The View

The View package holds classes and FXML-files responsible for the client GUI.

2.3.3 The Controller

The Controller package contains classes responsible for controlling the model, view and services by taking inputs from the view and services packages and updating the view, send messages via the services or run logic in the model.

2.3.4 The Services

The Services package is composed of an interface IService and all classes that implements said interface. These services are responsible for all I/O connection client-side such as sending and receiving messages to and from the server as well as reading all currently logged in users.

2.3.5 Diagrams

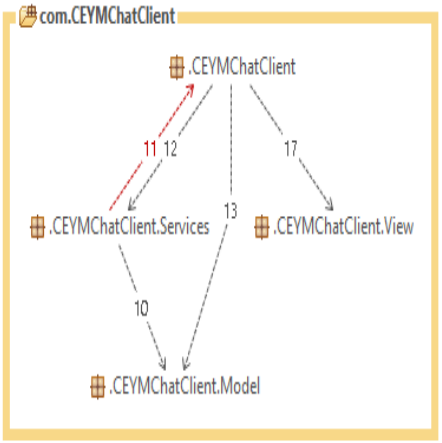

Figure 1: STAN4J generated dependency diagram of Client module

SECQUENCE DIAGRAM
For a class diagram, click here.

2.4 Model-lib

This module has no child-packages.

2.4.1 Diagrams

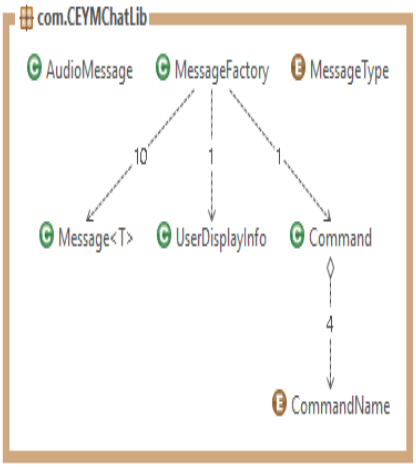

Figure 2: STAN4J generated dependency diagram of model-lib module

SECQUENCE DIAGRAM
For a class diagram, click here.

2.5 Server

2.5.1 Model

This package holds the classes representing the state of the server-side and the logic used to perform different operations changing the model.

2.5.2 Services

Contains network I/O-classes responsible for sending/receiving and processing messages. Also contains the service for connecting new clients to the model.

2.5.3 Diagrams

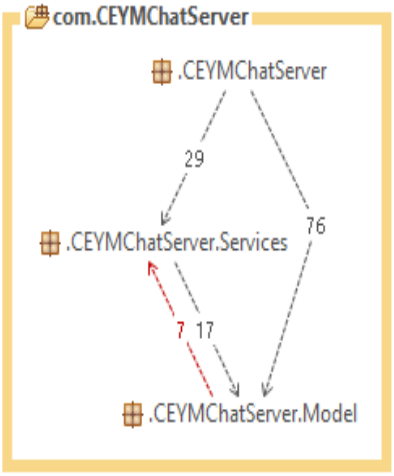

Figure 3: STAN4J generated dependency diagram of Server module

SECQUENCE DIAGRAM
For a class diagram, click here.

Quality

- Tests for each module can be find in their respective /src/test/java/com
- Quality tool reports: (INSERT PMD REPORT HERE)

2.6 Server

3 Persistent data management

All the visible resources are stored in a resource folder that are then used by the client. When a user is sent a file, that file is then stored in a folder. When a user closes the client, their chatlog is saved locally on their computer and will reload upon launch.

4 Access control and security

There are no ”admin”-roles in the client, instead an admin has control of the server meaning that only the operator of the system can look at critical data and overview the system at run-time. The server class should never be shared with anyone but those developing it meaning that usage of the server will be limited. All users of the client class have the same privileges and will not gain access to information they shouldn’t have access to.

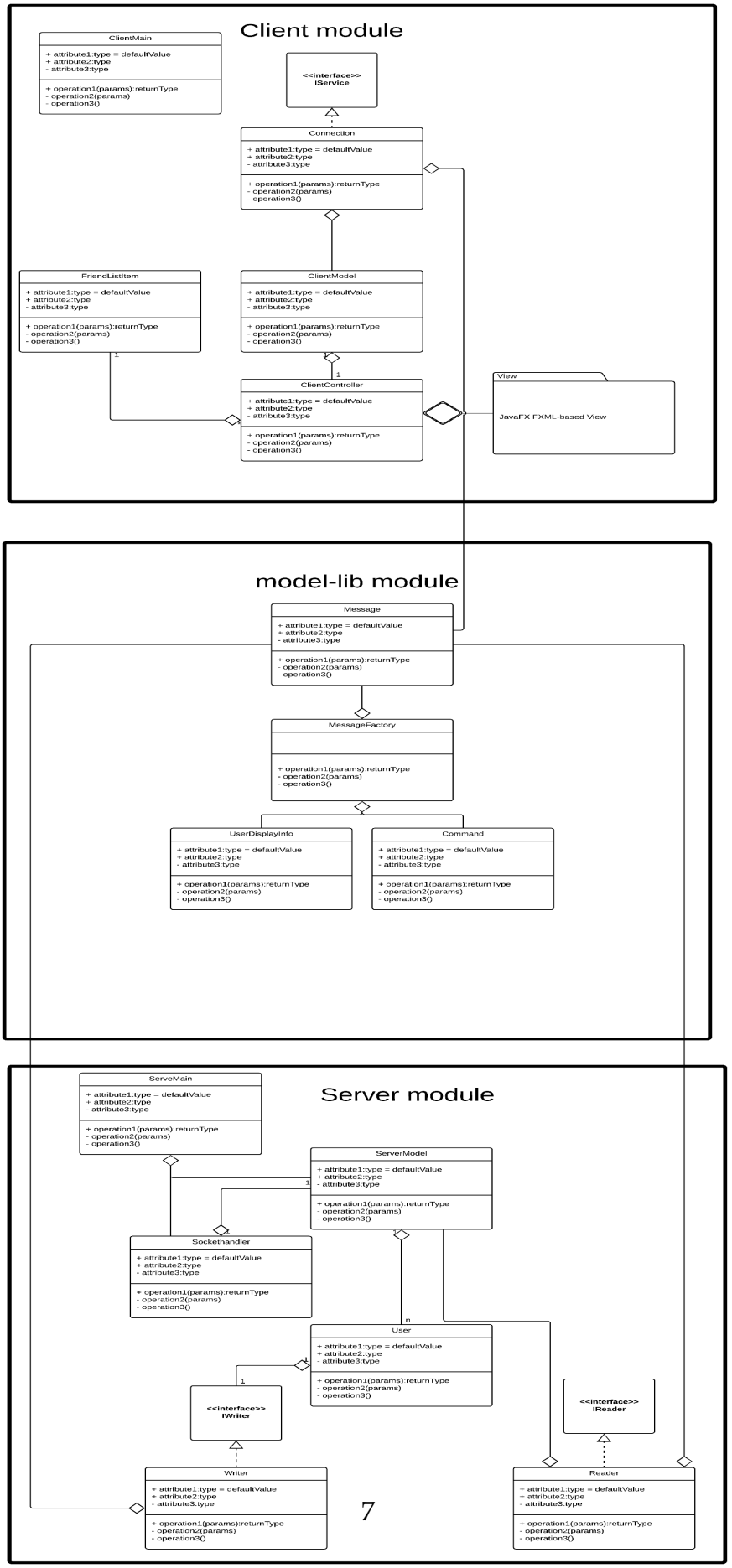

Figure 4: Class Diagram of all 3 modules