System Design Document for CEYMChat

authors

date
version

1 Introduction

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

1.1 Definitions, acronyms, and abbreviations

Some definitions etc. probably same as in RAD

2 System architecture

The Server All communcation 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 neccessary ports open to the WAN.

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 "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 and prints it to its client.

TODO: Describe how to start and stop the system.

2.1 Subsystem decomposition

Describe in this section each identified system component (that you have implemented).

2.2 'first component'

What is this component responsible for and what does it do.

Divide the component into subsystems (packages) and describe their responsibilities. Draw an UML package diagram for the top level. Describe the interface and dependencies between the packages. Try to identify abstraction layers. Think about concurrency issues.

If your application is a standalone then:

- Describe how MVC is implemented
- Describe your design model (which should be in one package and build on the domain model)
- Give a class diagram for the design model.

otherwise:

- MVC and domain model described at System Architecture

Diagrams

- Dependencies (STAN or similar)
- UML sequence diagrams for flow.
- Describe which design patterns you have applied.

Quality

- List of tests (or description where to find the test)
- Quality tool reports, like PMD (known issues listed here)

NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by ..., uses ..., etc.

2.3 'next component'

As above, and continue for all components.

3 Persistent data management

How does the application store data (handle resources, icons, images, audio, ...). When? How? URLs, pathes, ... data formats... naming..

4 Access control and security

Different roles using the application (admin, user, ...)? How is this handled?

5 References