Squire: A Collaborative Software Development Tool

Brandon Jank (jank6275@vandals.uidaho.edu)

March 5, 2016

Contents

Chapter 1

Overview and Scope

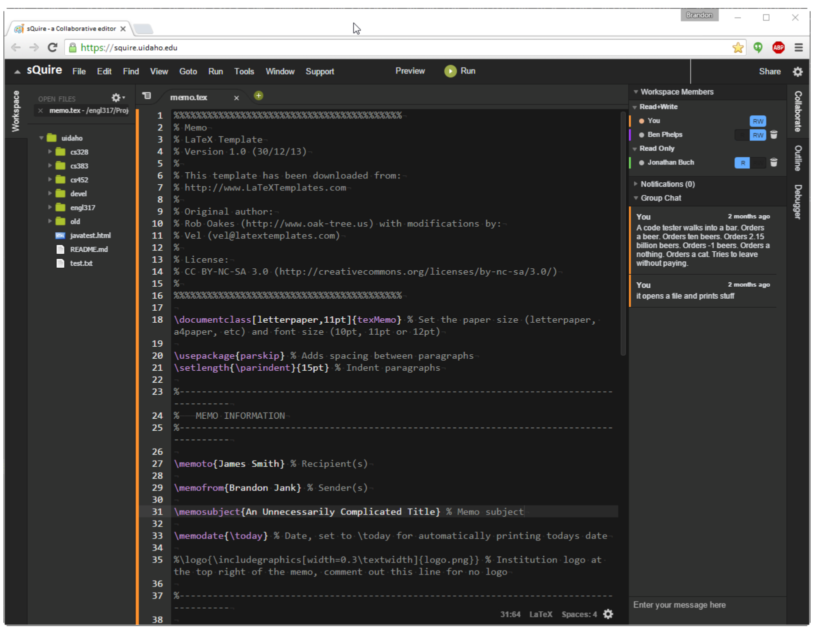


Figure 1.1: "Squire is a web-based collaborative software development environment with a project development center. Squire will allow multiple users to edit files and communicate in real time. Projects can be "stubbed" out by a user and then other users can join and/or vote to support for their favorite projects. After a certain amount of support, planning, and documentation is reached for a project, the project becomes a fully fleged project and then community development can start. Think "kickstarter for code" where people pledge their help with the project and not just financial support."

1.1 Core Features

- Web based
- Some Common IDE features

- Collaborative tools
- Encapsulated Workspaces
- Compile and Download
- Self Destructing Rooms

The central idea behind sQuire is a somewhat akin to a code hostel. A collaborative editor which provides private, but shareable on demand spaces, with a low overhead for creation and ease of use. These spaces are meant for short term use, over an indeterminate period. Once the collaboration is done, the space is cleared and made available for others to use.

With this vision in mind, sQuire best accomodates its users by being a browser based application. There should be no need for a user to download a client program, which would require periodic updates and ultimately require deletion on the users' end. By doing all of the collaboration and storage on a central server, accessed by the browser, we can make it more accessible to a wider audience. It may also encourage the development of collaborative "toy" projects by making it easier to start project spaces and find assistance when a project stalls.

sQuire will be focused on coding in the Java language, though there are many worthwhile languages to choose from. Focusing on a single language will allow us to add more IDE-like features to assist in collaboration and make the language more accessible to those learning it. IDE-like Features:

- Key word color coding
- Parenthesis mismatch detection
- Missing end of line detection/prompting

sQuire is a collaborative tool, rather than a fully featured IDE with step through run-time debugging. Its features should be geared towards making it easier to debug code collaboratively, in the browser, without over-reliance on other tools. At a minimum, it needs:

- Native chat functionality
- Author attribution for code (colored underlining, footnote, etc.)
- Ability to jump to another user's cursor
- Ability to import/export code as plain text
- Account based access to projects

Collaborative features that would be nice:

- Ability to save and restore to snapshots of the project
- Achievement and statistic tracking

User workspaces will be encapsulated, for both security and privacy. This could be accomplished using a container solution such as Docker or lmctfy (a free Google version). Containers have a lower performance hit to the host server than virtual machines, which are also a commonly implemented solution.

While security is a concern, we hope to address this using user space encapsulation, and by not running user code on the server. When users compile code, the compiled jar is downloaded and run locally from their own machines. While this means users are responsible for vetting the function of the code before running it on their machines, it means that our server resources are not being used as part of a bot net or similar exploit.

Finally rooms must be self-destructing. For the convenience of the user, deletion of the space should be automatic after an appropriate period of inactivity. While a notification email should be sent to the space owner prior to deletion, again no user intervention should be required.

1.2 Security Nightmare

The issue of security comes to mind every time compiling and executing an outside application in a secure area. Rooting or destroying data and infrastructure is trivial when you are freely allowed to execute arbitrary Java. In order to allow for this functionality we must implement some type of security strategy. We could limit or block functionality in Java which is a cat and mouse game that breaks usability. We could compile and execute code on the client side. Using a web browser, this would require a compiler written in JavaScript and cause compatibility issues on certain clients with reduced permissions or an exotic build environment. Or we could just Docker.

Docker allows us to create containers that contain not only the user's build environment, but the server infrastructure to host their project. Containers include the application and all of its dependencies, but share the kernel with other containers. Simply put, users can destroy their own containers but nobody elses. Obviously we will want to harden against known and obvious vectors, but at least any damage a user can do is limited to their own project. Containerization also allows us to take snapshots so that even if something goes wrong, we can always revert back to a working state.

Chapter 2

Requirements Documentation

2.1 Functional Requirements

Functional requirements will specify a behaviour or function. Squire's functional requirements are:

2.1.1 Communication (jank6275)

- Global chat when anywhere in program. (Usecase 3.1.2 and Class 4.2)
- Project chat when a project is open. (Usecase 3.1.1 and Class 4.2)
- Closeable project chat. (Usecase 3.1.3 and Class 4.2)
- Closeable global chat. (Usecase 3.1.4 and Class 4.2)

2.2 Non-Functional Requirements

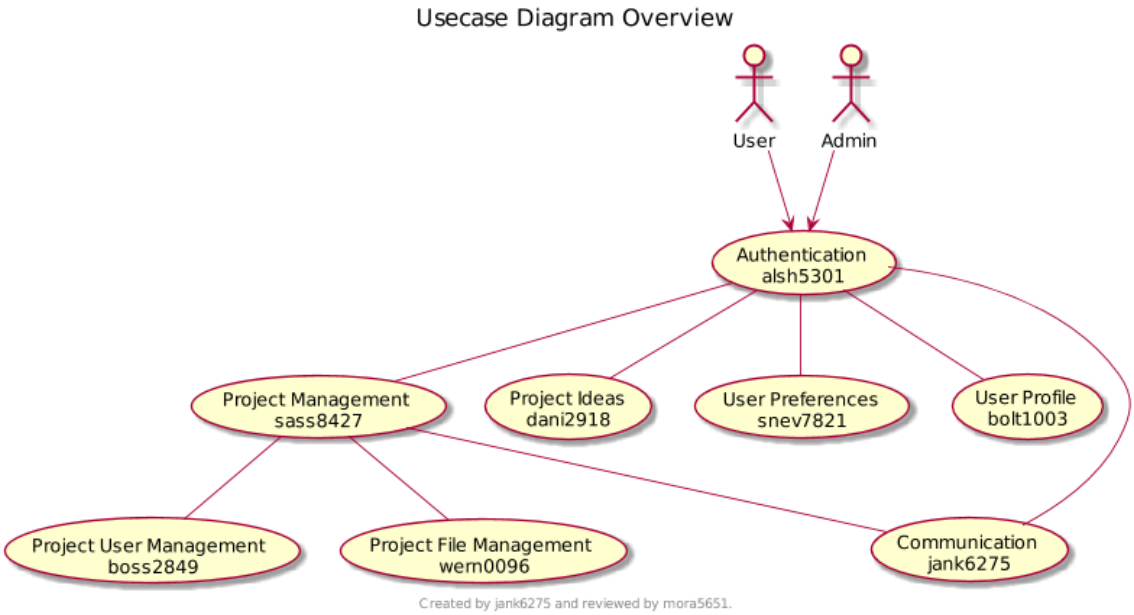
Non-functional requirements cover all the remaining requirements which are not covered by the functional requirements. They specify criteria that judge the operation of a system, rather than specific behaviours. Squire's non-functional requirements are:

- The current location of any user's cursor can be quickly jumped to by any user with the project open. (Usecase 4.5.8 and Class 5.9)
- Text written by the user in the editor should be visible instantaneously. (Usecase 4.5 and Class 5.9)
- Text written by other users in the editor should be visible within 2 seconds. (Usecase 4.5 and Class 5.9)
- Text will be underlined in the user's color, if they edited/created that text. (Usecase 4.5.9 and Class 5.9)
- The line number will be highlighted in the users color, if they created the line. (Usecase 4.5.9 and Class 5.9)

- Each line edited by a user should be saved to create a edit history for each user in a document. (Usecase 4.5.4 and Class 5.9)

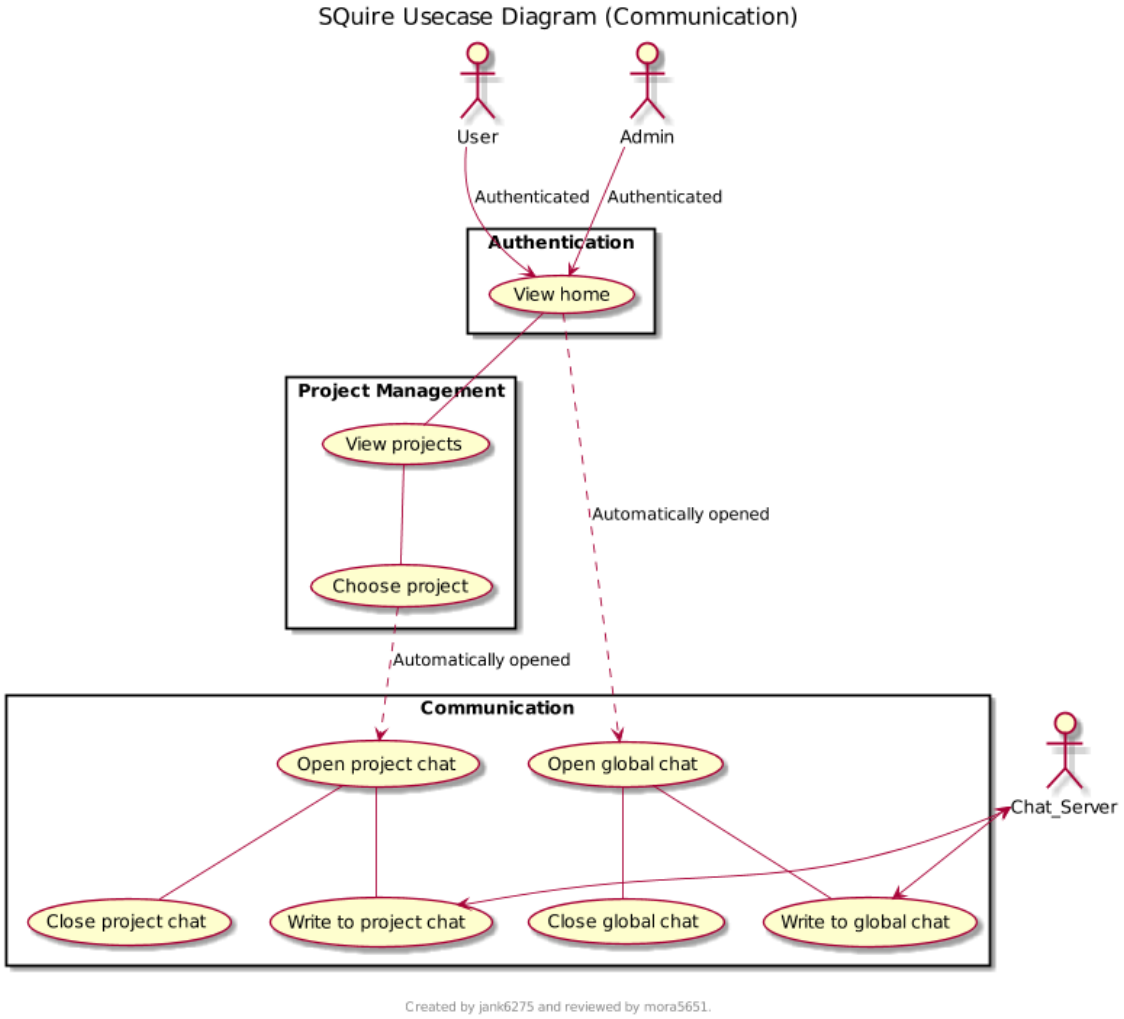
2.3 Use Case Diagrams

2.3.1 Overview



A usecase diagram that relates major sections of Squire.

2.3.2 Communication



A usecase diagram for sQuireś communication features. Used in Class Diagrams 4.2.1 and 4.2.2.

Chapter 3

Use Case Descriptions

3.1 Communication (jank6275)

3.1.1 Open project chat (Class Diagram 4.2.1)

Actors:	User
Goals:	To open the project chat window.
Pre-conditions:	User must be registered, signed in, and have a open project.
Summary:	User opens a project and the project chat automatically opens. The chat window displays chat history and updates when new chat messages are received.
Related use cases:	Join global chat.
Steps:	1. User opens a project. 2. Chat is notified that user has joined. 3. System displays project chat window to the user.
Alternatives:	None.
Post-conditions:	None.

3.1.2 Open global chat (Class Diagram 4.2.1)

Actors:	User
Goals:	To open the global chat window.
Pre-conditions:	User must be registered, signed in, and anywhere on website.
Summary:	User authenticates with the server and the global chat automatically opens. The chat window displays chat history and updates when new chat messages are received.
Related use cases:	Join project chat.
Steps:	1. User clicks open global chat. 2. Chat is notified that user has joined. 3. System displays global chat window.
Alternatives:	None.
Post-conditions:	None.

3.1.3 Close project chat (Class Diagram 4.2.1)

Actors:	User
Goals:	To close the project chat window.
Pre-conditions:	User must be registered, signed in, and in editor Mode.
Summary:	User clicks on close project chat and the chat window closes.
Related use cases:	Close global chat.
Steps:	1. User clicks close project chat. 2. Chat is notified that user has left. 3. Client closes project chat window.
Alternatives:	None.
Post-conditions:	None.

3.1.4 Close global chat (Class Diagram 4.2.1)

Actors:	User
Goals:	To close the global chat window.
Pre-conditions:	User must be registered, signed in, and anywhere on website.
Summary:	User clicks on open global chat and the chat opens, displaying chat history and updating when needed.
Related use cases:	Close project chat.
Steps:	1. User clicks close global chat. 2. Chat is notified that user has left. 3. Client closes global chat window.
Alternatives:	None.
Post-conditions:	None.

3.1.5 Write to project chat (Class Diagram 4.2.1)

Actors:	User
Goals:	To send text to project chat.
Pre-conditions:	User must be registered, signed in, a project opened, with the project chat window open, and the text box selected.
Summary:	User clicks in the project chat text box and then types a message then either presses enter or clicks the submit button. The text is displayed to all users in the chat, including the user.
Related use cases:	Write to global chat.
Steps:	1. User clicks in the project chat box. 2. User types a message and then presses enter or clicks submit button. 3. Message is relayed to all clients with project chat open. 4. Message is displayed.
Alternatives:	None.
Post-conditions:	None.

3.1.6 Write to global chat (Class Diagram 4.2.1)

Actors:	User
Goals:	To send text to global chat.
Pre-conditions:	User must be registered, signed in, anywhere on website, with the global chat window open, and the text box selected.
Summary:	User clicks in the global chat text box and then types a message then either presses enter or clicks the submit button. The text is displayed to all users in the chat, including the user.
Related use cases:	Write to project chat.

Steps:

1. User clicks in the global chat box.
2. User types a message and then presses enter or clicks submit button.
3. Message is relayed to all clients with global chat open.
4. Message is displayed.

Alternatives:	None.
Post-conditions:	None.

3.1.7 Modify chat font (Settings Class Diagram)

Actors:	User
Goals:	To change a users font style inside the global and project chat.
Pre-conditions:	User must be registered, signed in, the user settings window opened, and the chat settings tab open.
Summary:	The user clicks the settings menu and changes their font style for both the project and global chat through a drop down box of available fonts.
Related use cases:	Modify chat color.
Steps:	1. User clicks the settings menu. 2. User clicks chat settings tab. 3. User clicks chat font drop down box. 4. User clicks desired font. 5. User clicks save. 6. The user's selection is saved in the database. 7. All further chat messages will use the selected font.
Alternatives:	None.
Post-conditions:	None.

3.1.8 Modify chat color (Settings Class Diagram)

Actors:	User
Goals:	To change a users font color inside the global and project chat.
Pre-conditions:	User must be registered, signed in, the user settings window opened, and the chat settings tab open.
Summary:	The user clicks the settings menu and changes their font color for both the project and global chat through a drop down box of available colors.
Related use cases:	Modify chat font.
Steps:	1. User clicks the settings menu. 2. User clicks chat settings tab. 3. User clicks chat color drop down box. 4. User clicks desired color. 5. User clicks save. 6. The user's selection is saved in the database. 7. All further chat messages from the user will use the selected color.
Alternatives:	None.
Post-conditions:	None.

Chapter 4

Class Diagrams

4.1 Overview (brec9824, jank6275)

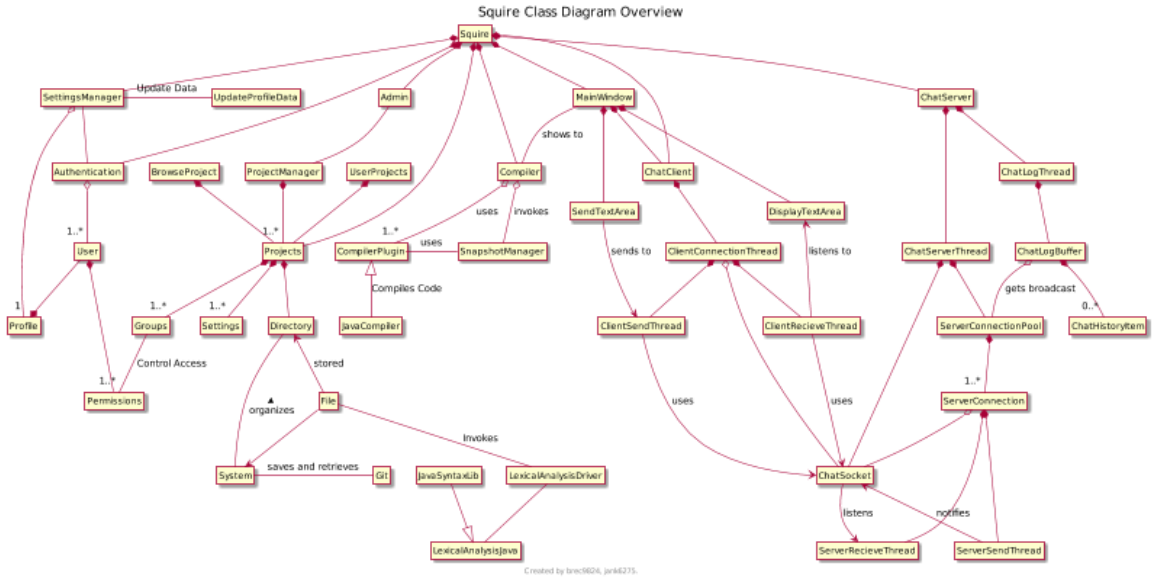


Figure 4.1: Class overview of Squire displaying the connections between each subsystem and their classes.

4.2 Communication (jank6275)

4.2.1 Client

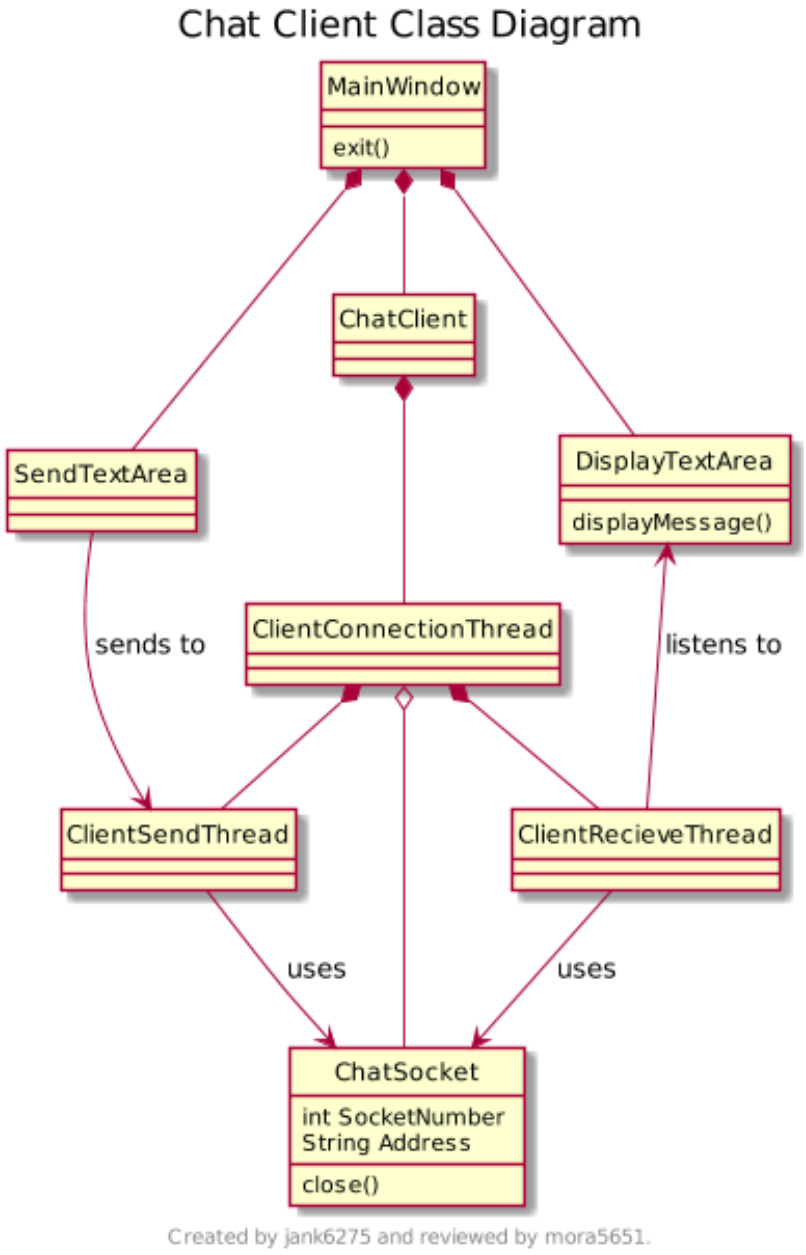


Figure 4.2: The ChatClient class will handle text communication in conjuction with the ChatServer class. The Main Window will be home to the ChatClient. The ChatClinet will consist of client send/recieve threads to handle user input/output in the Main Window via the ChatSocket.

4.2.2 Server

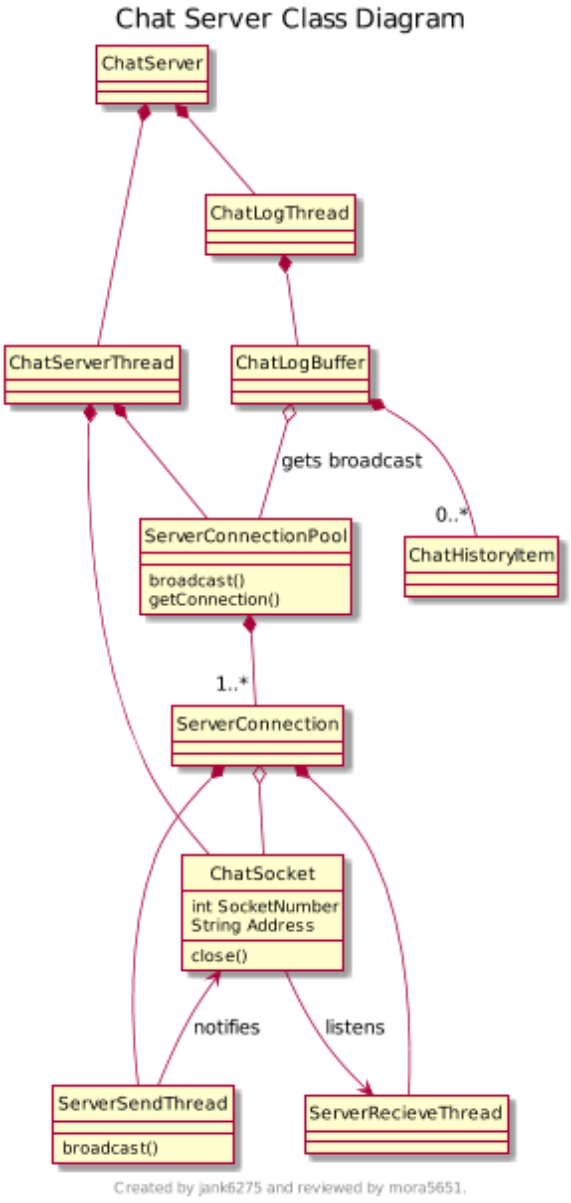


Figure 4.3: The ChatServer class will handle text communication between ChatClient(s). The ChatLogThead records any messages broadcast by chat clients in the ServerConnection-Pool as a ChatHistoryItem. Each ServerConnection consists of a send and recieve thread that utilize the ChatSocket to broadcast messages.

Chapter 5

Sequence Diagrams

5.1 Authentication (jank6275)

5.1.1 Login

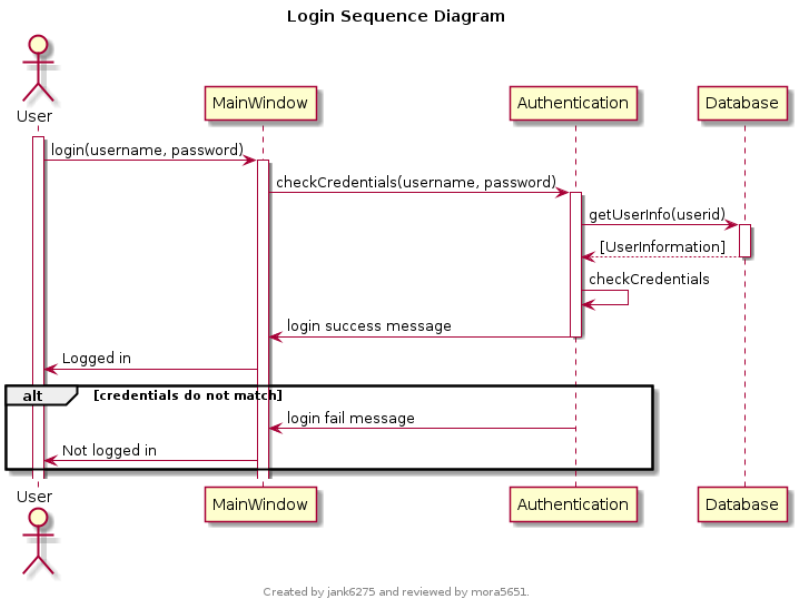


Figure 5.1: A sequence diagram for logging in to Squire.

5.1.2 Logout

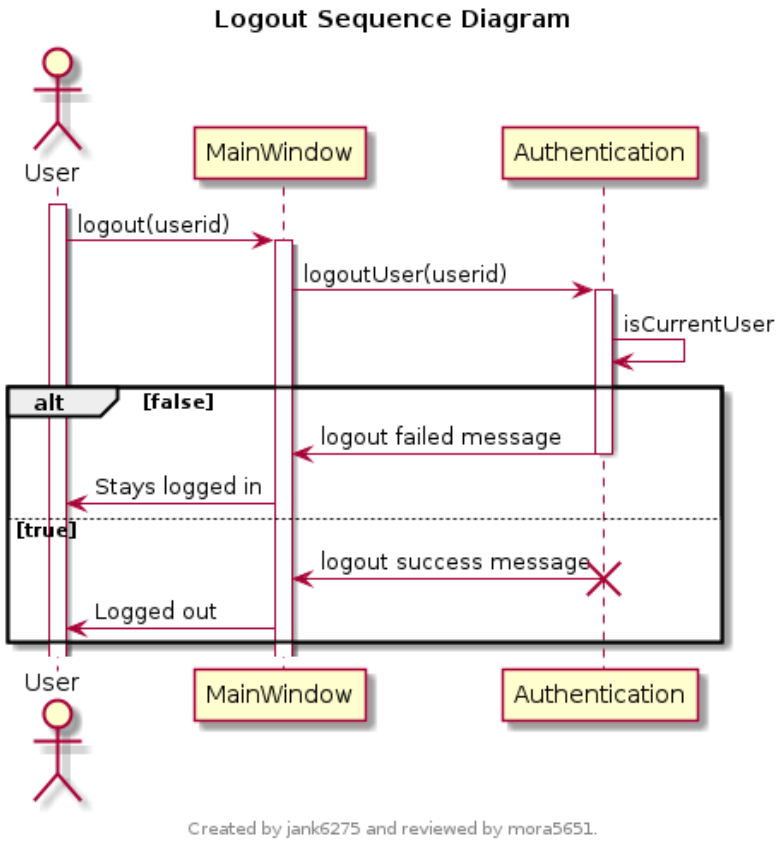


Figure 5.2: A sequence diagram for logging out of Squire.

5.1.3 Register

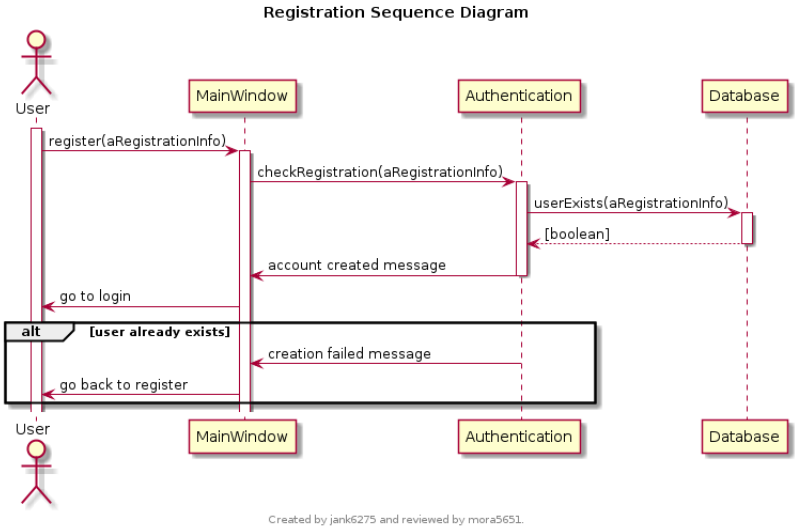


Figure 5.3: A sequence diagram for registering a nmew account with Squire.