sQuire: A Web Based Collaborative Editor

benz5834, bolt1003, carl7595, guan2264, jank6275, knic1468, mora5651, ratc8795

January 27, 2016

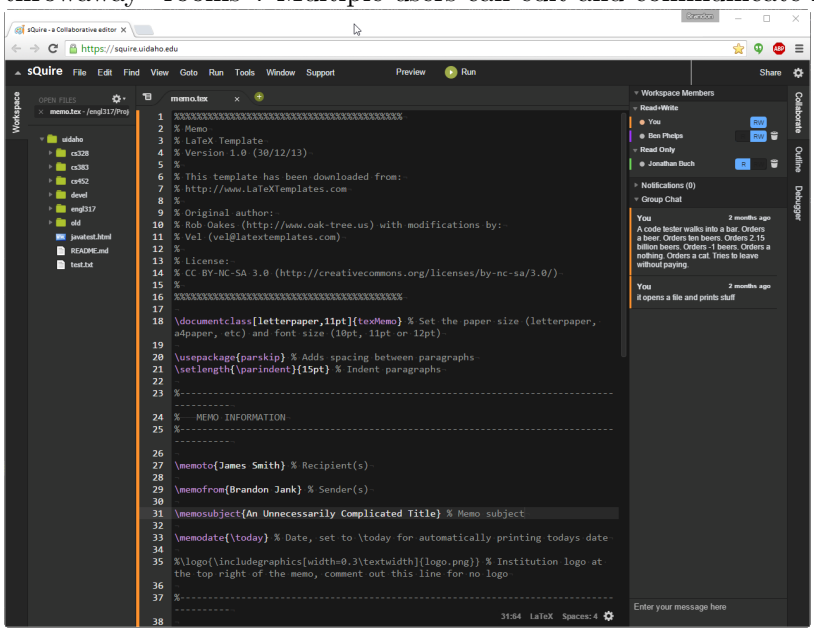
Contents

Chapter 1

Application Domain Specification

1.1 Program Premise

Figure 1.1: Squire will be a web-based collaborative text editor that compiles java code in throwaway "rooms". Multiple users can edit and communicate in real time.



1.1.1 Core Features (carl7595)

- 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/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.

1.1.2 Storage and Organization (benz5834)

The squire program will have a top down directory structure. Under the top domain, each individual registered user will have the ability to create projects (rooms) where they can build their program. Inside the room the user will be able to create any number of files or pages to keep track of their project. They will also have the ability to import and export files to and from their own machine and the project. After creation the user will be able to switch between projects and create and delete pages from within the project itself. They will have the ability to share their projects and invite other users to read or edit their work. All of the user data, as well as each users relevant project and page information, as well as who has access to each project will be stored and maintained by a MySQL database.

1.1.3 Security Nightmare (jank6275)

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. 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.

1.1.4 Execution (guan2264)

The Squire program will have a browser based client. Squire users can use this client to login their account, manipulate their project, and logout. The Squire editor will run clientside through Ace. Ace is an embeddable code editor written in JavaScript. It can be easily embedded in any web page, JavaScript application, and browser based client. On the other

hand, the user data, user project, and edit information will saved in a MySQL database on the owner's hard drive.

1.1.5 Collaboration (ratc8795)

The code editor will be a collaborative editor. Multiple users can edit the same file at the same time. This means that any changes one user makes need to be synced to the server, and then to the editors of any other users editing the same file. Some research shows that all popular collaborate editors (Google Docs, Cloud 9, etc) use an algorithm called Operational Transformation. On a basic level, this algorithm works by treating every change to a file as either an insert command, or a delete command. These commands are synced across clients to keep the file state the same. Squire will need something similar. It looks like there are several open source implementations of this algorithm, notably OT.js. These may or may not be adequate for use in Squire.

1.1.6 Achievement System (ratc8795)

In order to make Squire interesting, but still retain its usability, Squire will contain an achievement system. Achievements will be publicly visible to all users in the user's profile. When a user gains an achievement, it will be announced in the project chat. Achievements should be fun, and should be obtainable just by writing code (i.e, users shouldnt have to actively try to get achievements). Some possible achievements: Lucky Break (a file compiles successfully on the first try), Social Butterfly (a user joins a project that has 10+ users), Job Security (Write a 1000+ line file without any comments).

1.1.7 Permissions System (bolt1003)

Squire will have a permissions system that encompasses files, folders, users and rooms. The administrator will have the ability to delegate permission to other users. The administrator will have the ability to set global permissions to users to control read, write, execute, room creation and sharing. Each room with have its own set of permissions. These permissions are controlled by the moderator that is assigned to that room and the administrator. By default the creator of the room is the moderator of the room but this duty can be delegated to another if desired. The system has three default user groups, Administrator, Moderators and Users. Users have the ability to read, write and execute code by default. Moderators have the ability to manage rooms by adding or removing users to their rooms, creating sub-rooms and managing moderators for those rooms. Administrators have all encompassing power over all permissions in the system.

1.1.8 Interface Structure (mora5651)

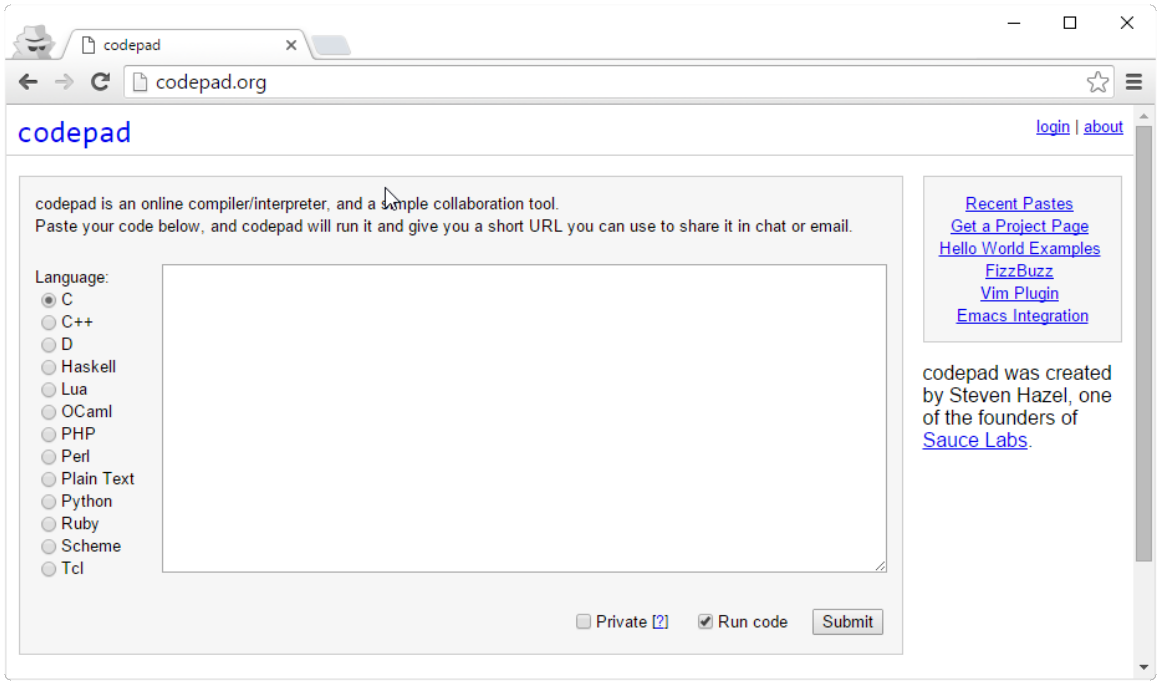
Since sQuire is an online collaborative editor, and interface structure plays a key roll in organizing all the features to fit properly. We decided in using Cloud9's interface as a base for sQuire's interface layout/structure. Cloud9 incorporates many of the same features as sQuire including file management, editor, and chat options. Other collaborative IDEs to consider Floobits, Kobra, and Codebox.

1.1.9 Chat System (knic1468)

We will most likely use a websocket-based chat system for sQuire, as it allows the easiest modification; Users and their coded colors will be listed, and their names next to their chat lines. The goal is to make a simple and lightweight, secure chat that allows users to collaborate with each other in an area outside the workspace.

1.2 Application Domain Study

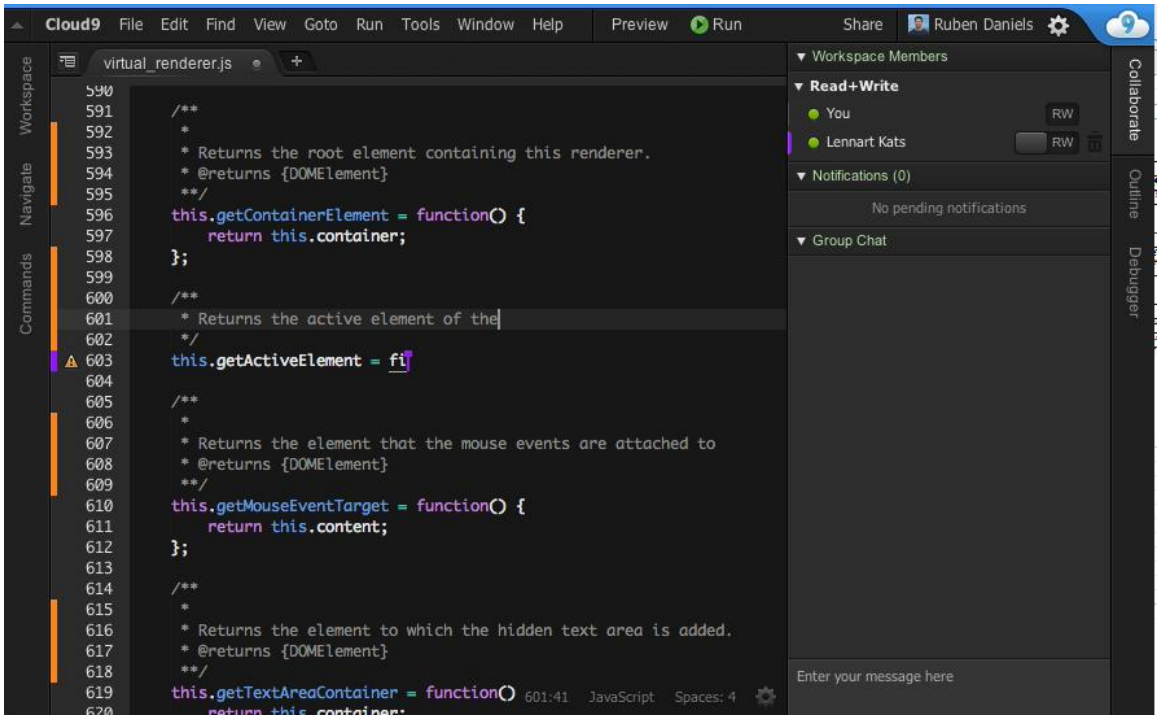
1.2.1 codepad.org (jank6275)



Codepad is an online compiler and simple collaboration tool that compiles interactive code from the web. Safely compiling and executing arbitrary code from the web will always be a security problem. Steven Hazel, the creator of codepad.org, used Docker to containerize each development environment and then execute the users code. This limits the damage an unwitting or malicious user can do to just their container. We should utilize this technology with sQuire. The webserver, database, front-end, and user application can be copied from a template and a virtual machine spun up just for that users project.

Even though codepad.org uses cool technology for solving the security issue, it is not without its flaws. There is no live chat functionality and communication is critical for online collaboration. You cannot edit code with someone at the same time, you can only save-and-share. This slows down the development process and can cause code "collisions". These areas are a niche for sQuire to shine in among a large swath of online collaborative IDEs.

1.2.2 Cloud9 IDE - c9.io (guan2264)



Cloud9 IDE is a freeware online integrated development environment. It supports hundreds of programming languages, including PHP, Ruby, Perl, Python, and JavaScript. It enables users to get started with coding immediately with pre-setup work spaces, collaborate with their peers with collaborative coding features, and web development features like live preview and browser compatibility testing. It is written almost entirely in JavaScript, and uses Node.js on the back-end. The editor component uses Ace.

1.2.3 IDEone (ratc8795)

Ideone (ideone.com) is a online compiling and debugging system. It allows online compiling and debugging of over 60 languages. It is implemented using Sphere-engine's api. Sphere-engine provides a system to allow execution of untrusted code. This is reflected in Ideone. For example, when you try and run a program that does the equivalent of "sleep 60", "ping google.com", or "rm -rf /", an error is thrown and the code is not run. Ideone is extremely simple. It provides a basic code editor, an option to select the language, a box to type in any standard input, and an output box showing any errors or output. It also provides functionality to save and share scripts with other people. Notable limitations are that there is only one file, and no collaborative editing. As far as a simple way to execute code from many different languages, Ideone does a very good job.

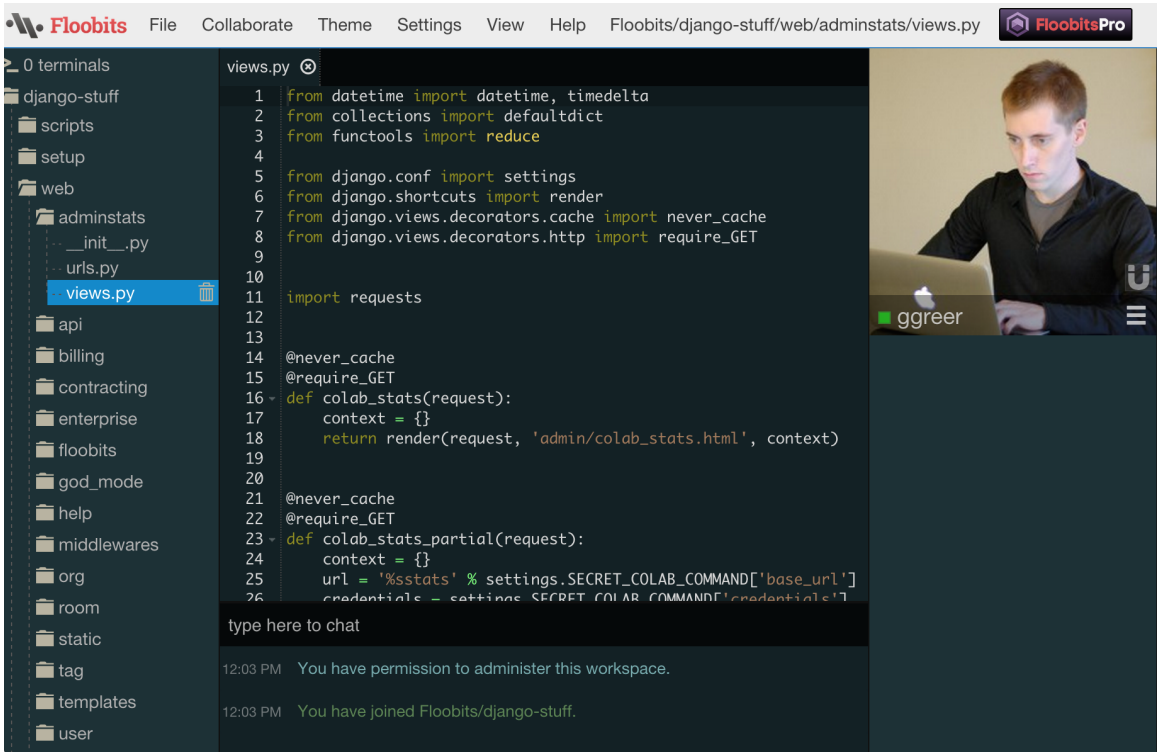
1.2.4 Codenvy (bolt1003)

Codenvy is an online IDE that supports a variety of languages. This includes C, C++, Go, Python, Java, JavaScript, PHP and Ruby. Codenvy has a dedicated virtual machine per

project (implemented using docker) to compile and execute code within. Codenvy allows for multiuser editing of code in real-time. Codenvy allows a user to create a temporary clone of a project to modify and execute in its own dedicated VM. Codenvy integrates with popular subversion control systems and hosting such as GitHub. SSH is provided so that users can login to the VM directly to make modification and execute code.

Codenvy provides basic features free of charge, most of which would be fine for moderate use by a single user or small group. Their business model is targeted at larger business so premium features come at a premium. The integrated interpreter is difficult to setup initially and often provides cryptic error messages. Logging in via ssh and executing code is more consistent experience. Codenvy lacks an integrated chat client so a third party client is needed.

1.2.5 Floobits (mora5651)



Floobits is a web based application that allows users to collaborate on the same code while using there favorite editor. Examples of editors that are supported in Floobits include Emacs, Sublime, Atom, and more. Although Floobits focuses towards "paired programming" it also supports multi user groups integrating voice chat, as well as a video chat options using Google Hangouts. Floobits is fairly simple to customize giving the users options to install different plugins for different functionality Floobits has to offer. However, to access some of these features you will need an upgraded paided account. Floobits also suffers from security issues.

1.2.6 Google Docs (benz5834)

Google Docs is a collaborative document editor built around the Google ecosystem. Anyone with an account can create documents, spreadsheets, slideshows, and forms. Within each one of these documents there is a menu to share your document and even give editing privileges. In this sense it is very similar to what we want to accomplish with Squire. Account based document creation with the ability to collaborate and work together on the same document at the same time. There are even a few features that google docs has that our squire program could benefit from. A running comment thread within each document allows for editors and viewers to be able to leave notes and comments about what they have done outside of the document itself. There are a large number of document templates included within each document type as well. The project management page is also very clean and easy to organize with both a list and grid view of saved documents. And because accounts are tied in with email addresses, it is very easy to send messages to collaborators about your documents. Google Docs is well used for every type of user so I think it is a very important tool to study while creating Squire.

1.2.7 Sharelatex (carl7595)

Sharelatex is a collaborative text editor targeted towards the Latex language. Sharelatex.com uses a similar email based registration scheme, and does allow for compiling in the browser. It also includes a history feature of changes made to the document, which is one way that snapshots could be added at working compile. I particularly like that in the interface there is a share button that immediately prompted for the email address of the person you want to share with. The chat window is a pane that opens on the right side, squeezing the rest of the windows to accomodate this. This is not a problem in full screen mode, but I think I would prefer the chat to be always open and near the bottom of the screen. It has an option for creating a sample project, which would be a great feature for giving someone a template container to work with to make things more accessible to new coders.

1.2.8 Mibbit and Ratchet (knic1468)

Because sQuire will require a chat function to truly be a collaborative IDE, I took a look at two lightweight chat clients, mibbit and the demo on Ratchet. Both were lightweight, but one used irc protocols and connected to servers, while Ratchet is websocket based, and the chat rooms can be more custom and hosted on the local server. Since the Ratchet demo is a simple example of the Ratchet API, it allows one to really see what it can let you do. As a PHP-based API, it is developed for being browser-based as well. While not as lightweight as a built-in irc client, the websocket-based client would be more secure and modifiable.

1.3 Use Case Descriptions

1.3.1 Open project chat (jank6275)

Actors:	User
Goals:	To open the project chat window.
Pre-conditions:	User must be registered, signed in, and in editor Mode.
Summary:	User clicks on open project chat and the chat opens, displaying chat history and updating when needed.
Related use cases:	Join global chat.
Steps:	1. User clicks open project chat. 2. Chat is notified that user has joined. 3. System displays project chat window.
Alternatives:	None.
Post-conditions:	None.

1.3.2 Open global chat (jank6275)

Actors:	User
Goals:	To open 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:	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.

1.3.3 Close project chat (jank6275)

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.

1.3.4 Close global chat (jank6275)

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.

1.3.5 Write to project chat (jank6275)

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.

1.3.6 Write to global chat (jank6275)

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.

1.3.7 Modify chat font (jank6275)

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.

1.3.8 Modify chat color (jank6275)

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.

1.3.9 Create Project (bolt1003)

Actors:	Users of sQuire.
Goals:	Create a Project.
Pre-conditions:	The user is logged in.
Summary:	User opens the project manager and creates a project.
Related use cases:	None.
Steps:	1. User selects the project manager from the main menu. 2. User selects open. 3. Create new project is selected from the task bar. 4. A name is choosen for the project. 5. A location is selected for the project. 6. Language is selected from a drop down menu. 7. User clicks finish.
Alternatives:	None.
Post-conditions:	None.

1.3.10 Choose Project (bolt1003)

Actors:	Users of sQuire.
Goals:	Open a Project.
Pre-conditions:	A project has been created and the user is logged in.
Summary:	User looks through the list of project and selects the desired project.
Related use cases:	None.
Steps:	1. User selects the project manager from the main menu. 2. User selects open. 3. User selects the desired project. 4. User clicks open in the list of options.
Alternatives:	None.
Post-conditions:	None.

1.3.11 View User Profile(bolt1003)

Actors:	Users of sQuire.
Goals:	Users looks at their profile page.
Pre-conditions:	The user has an account and is logged in.
Summary:	User opens their profile from the main menu to view.
Related use cases:	User modifies their profile.
Steps:	1. The user selects their profile picture from the main menu. 2. The user selects view Profile from the drop down menu.
Alternatives:	None.
Post-conditions:	None.

1.3.12 Modify User Profile(bolt1003)

Actors: Users of sQuire.

Goals: Users updates their profile.

Pre-conditions: The user is on the view profile page.

Summary: User modifies their profile information.

Related use cases: None.

Steps:

1. The user selects modify.
2. The user changes their username.
3. The user clicks update.
4. The user updates their contact email.
5. The user clicks update.
6. The user changes their "About me" information
7. The user clicks update.
8. The user clicks finish to disable modify.

Alternatives: None.

Post-conditions: None.

1.3.13 Login (guan2264)

Actors:	Squire User
Goals:	Allows user to login to the system and use functions of the system.
Pre-conditions:	The user must have user account.
Summary:	User types in the correct user name and password, click the login button, and login to the system. When login successfully, the system will display the access page for user.
Related use cases:	Choose project, create project, join project, delete project, edit project, invite user to project, accept invite to project, remove user from project, project chat, global chat, roject permissions, user preferences, import file, export file, gain achivement, view user profile, and logout.
Steps:	1. The system displays the Login page that asks the user to enter username and password. 2. The user types in their username and password. 3. The user click the login button. 4. The system checks the username and password from the data base. 5. The system verifies username and password. 6. The system displays the access page for the user.
Alternatives:	If the user do not enter the correct username and password, the system will ask the user to enter the correct username and password
Post-conditions:	User logs in and use functions of the system.

1.3.14 Logout (guan2264)

Actors:	Squire User
Goals:	Allows user to logout to the system.
Pre-conditions:	The user must have user account and already login to the system.
Summary:	User click the logout button and their session is terminated.
Related use cases:	Close project chat, close global chat, close project file, and login.
Steps:	1. The user click the logout button. 2. The system invalidates the session. 3. The system logs the user out. 4. The system redirects to the default login page.
Alternatives:	None
Post-conditions:	User's previous session is terminated and the system displays the login page.

1.3.15 Invite user to project (ratc8795)

Actors:	User sending invites, user receiving invites
Goals:	Give another user access to a project.
Pre-conditions:	User has the permissions necessary to invite a user to a project.
Summary:	User opens a dialog to invite users, inputs one or more users, and the level of access to the project that they should be given.
Related use cases:	Accept invite to project, project permissions

Steps:

1. User clicks on a button to invite users.
2. A dialog is displayed to the user.
3. User types in one or more emails.
4. User specifies the level of access for each user.
5. User clicks submit.
6. The dialog disappears.
7. Emails are sent to the email addresses specified containing invites.

Alternatives:	User does not have permission to invite others to the project, so the button to invite users is not displayed.
Post-conditions:	None.

1.3.16 Edit project file (ratc8795)

Actors:	One or more users editing a file
Goals:	Make changes to a file and sync those changes with other users editing the same file.
Pre-conditions:	User has the permissions necessary to edit the project, and the file is open.
Summary:	A user makes a change to a file in a text editor, and those changes are reflected to other users editing the same file.
Related use cases:	None.

Steps:

1. User makes a change in the editor.
2. The editor of other users editing the same file reflects the change
3. The editor updates any syntax highlighting and formatting
4. A note in the margin updates to show the user was the last one to edit that line
5. The editor of other users editing the same file shows the same note in the margin

Alternatives:	User does not have permission to edit the file, so the editor ignores any input from that user.
Post-conditions:	None.

1.3.17 Open project file (ratc8795)

actors:	User
goals:	A user selects a file from the project for editing or viewing.
pre-conditions:	The file already exists
summary:	A user selects a file from the project which opens, replacing any file that is already open.
related use cases:	Close a project file
steps:	1. A user selects a file from the list of files in the project. 2. If a file is currently open, that file is closed. 3. The file is opened in the editor. 4. The editor shows the user as being active in the file.
alternatives:	None.
post-conditions:	None.

1.3.18 Close a project file (ratc8795)

actors:	User
goals:	Closing a file.
pre-conditions:	A user clicks a button to close an open file and the editor is closed.
summary:	The file is closed.
related use cases:	Open a file.
steps:	1. A user clicks on a button to close the file, or has selected another file to open. 2. The file is closed, and removed from the editor. 3. The editor now longer shows the user as being active in the file.
alternatives:	none.
post-conditions:	none.

1.3.19 Create a project file (ratc8795)

actors:	User
goals:	Creating a new file.
pre-conditions:	The user has permission to create a new file.
summary:	A user new project file is cr
related use cases:	Move a file, Delete a file
steps:	1. A user clicks on a button to create a file. 2. A dialog is displayed asking for a file path. 3. The user types in the file path. 4. The file is created.
alternatives:	The specified file name contains invalid characters, so an error is displayed.
post-conditions:	None.

1.3.20 Move a file (ratc8795)

actors:	User
goals:	A user moved from one location to another
pre-conditions:	The user has permission to move a file.
summary:	A file is moved from it's location to a new location specified by the user
related use cases:	Create a file, Delete a file
steps:	1. A user clicks on a button next to a file to rename it 2. A dialog is displayed asking for a file name. 3. The user types in the file name. 4. The file is moved to the new location
alternatives:	The user types in a location that is invalid; an error is displayed.
post-conditions:	None.

1.3.21 Delete a file (ratc8795)

---

actors: User

goals: A file is deleted

pre-conditions: The user has permission to delete a file.

summary: A user clicks a button to delete a file, and it is removed from the project

related use cases: Create a file, Delete a file

steps:

1. A user clicks on a button next to a file to delete it
2. A dialog is displayed asking for confirmation.
3. The user clicks yes.
4. The file is deleted.

alternatives: The user clicks no in the confirmation so the file is not deleted.

post-conditions: None.

1.3.22 Gain an achievement (ratc8795)

actors:	User
goals:	Notify the user that they gained an achievement
pre-conditions:	The user has met the requirements for the achievement.
summary:	A user is informed that the earned an achievement, and users in the project chat are informed that the user earned an achievement, and the user's profile is updated to include the achievement
related use cases:	None.
steps:	1. A user earns an achievement. 2. A pop up is displayed to the user informing them of the achievement. 3. After a couple seconds, the pop up goes away. 4. A message is sent to the group chat informing other users of the achievement.
alternatives:	None.
post-conditions:	None.

1.3.23 Join Project(mora5651)

Actors:	User.
Goals:	To join an existing project.
Pre-conditions:	Must be registered, logged in, then must choose a project to join.
Summary:	User has logged in, user then chooses a project to join, if user is allowed then the user joins the project.
Related use cases:	Invite user to project, Accept user invite.
Steps:	1. The user chooses the "join a project" button. 2. If the project is held in a public room then the user joins. 3. If the project is held in a private room then the user will need permission. 4. If the admin of the project the user wants to join is accepted. Then the user joins the project.
Alternatives:	User may decline an inventation to join a project.
Post-conditions:	You entered a Project.

1.3.24 Delete Project(mora5651)

Actors:	User with permission.
Goals:	To delete an existing project.
Pre-conditions:	Must have permission to delete project.
Summary:	Users are done with the project, and if the user has permission then they are allowed to delete the project. project is then deleted.
Related use cases:	Project Permissions.
Steps:	1. The user clicks on the "Delete project" button. 2. A dialog is displayed. 3. User clicks either yes or no to delete project. 4. Project is then deleted.
Alternatives:	User may choose not to delete the project in the confirmation display.
Post-conditions:	Deleted Project.

1.3.25 Import File (Knic1468)

Actors:	User with permission.
Goals:	To upload a file to the workspace.
Pre-conditions:	Must have permission to read/write.
Summary:	User uploads a file into the workspace for collaborative editing.
Related use cases:	Project Permissions.
Steps:	1. The user clicks on the "Import File" button. 2. System asks for a file to upload. 3. User picks a file to upload. 4. System reads file and uploads it into workspace.
Alternatives:	System may reject a file if it is not the correct format.
Post-conditions:	None.

1.3.26 Export Project(knic1468)

Actors:	User with permission.
Goals:	To export a workspace to a local file.
Pre-conditions:	Must have read permission.
Summary:	User saves a file to the local storage.
Related use cases:	Project Permissions.
Steps:	1. The user clicks on the "Export File" button. 2. System promts the user to select a location and name. 3. User selects a file location. 4. System exports the file to the location.
Alternatives:	System will display an error message of the location is write-protected.
Post-conditions:	None.

1.3.27 Accept Invite to Project (carl7595)

Actors: User who received the invite (Invitee), sQuire user who sent the invite (Referrer), and Project Owner (Owner. May be the same as the Referrer).

Goals: Gain access to a shared Project

Pre-conditions: Authorized user with access to the project sends an email invite to another user.

Summary: Access is granted to a project space using an invitation email.

Related use cases: Create an account (if the invitee does not already have one), Login (if user is not logged in), and Create project (if the project has been deleted).

Steps:

1. Invitee clicks on the link received by email.
2. New browser window opens with sQuire site.
3. Message appears that invitee's account has been added to the access list for the Project.
4. The project is added to their Projects list.
5. Owner and Referrer (who may be the same) receive an email informing them a new collaborator has been added
6. Project is opened in active pane of sQuire tab after a short delay.

Alternatives: 2B. If invitee is not logged in to Squire, they are prompted for their credentials.
2C. If invitee is does not have an Squire account, they are prompted to first create an account.
3B. Project has been deleted. Invitee is informed the project does not exist and asks if they would like to create a new project with that name.
3C. Join link has been deactivated because of previous use/time limit. Invitee is informed that the invite has expired and is no longer valid, instructing them to contact the Project Owner for access.

Post-conditions: Invitee has been added to Project access list. Invite link is deactivated to prevent mulitple joins off of old links. Owner has been notified of new collaborator.

1.3.28 Remove User to Project (carl7595)

Actors: sQuire user with access to the Project (Member), Authorized user who requests the removal (Agent), and Owner of the Project (Owner. May be the same as the Agent)

Goals: Revoke access to the Project for a single or multuple users.

Pre-conditions: Agent has permission to edit the Project access list, and agent is logged into Squire and has opened this Project.

Summary: One or more user accounts are removed from the access list for a Project.

Related use cases: None.

Steps:

1. Agent selects the access list for this Project.
2. Agent selects an account (or multiple accounts) and selects "Remove from Project".
3. Agent is prompted for confirmation, and selects 'Yes'.
4. The access list is modified to remove the selected account(s).
5. Owner and Member(s) receive an email informing of the change of access.
6. Member's Project lists are updated to no longer display the Project.
7. Member's Project panes related to this Project forced to close on next refresh.

Alternatives:

- 2B. Agent does not have correct permissions. Error messsage informs them that they cannot edit Project access list.
- 3B. Agent clicks 'No'. Access list is not modified.

Post-conditions:

- Members have been removed from the Project, and their client windows closed to prevent continued access.
- Owner and Member have been notified of the change.

1.3.29 Edit Project Permissions (benz5834)

Actors:	Squire user
Goals:	Edit the permissions for a project
Pre-conditions:	User must have the project that he wants to edit the permissions of open.
Summary:	User opens up the settings menu and navigates to permissions, adds (or removes) users individual access rights to the project.
Related use cases:	Add user to project, Remove user from project.
Steps:	1. Click the settings button in project. 2. System opens settings menu. 3. User clicks permissions link. 4. System opens the permissions menu. 5. User selects user from list of users. 6. User adds read or write permissions to user. 7. User saves changes and exits permissions.
Alternatives:	User can remove read or write permission instead in step 6. User can discard changes instead in step 7.
Post-conditions:	None.

1.3.30 Open User Preferences (benz5834)

Actors: Squire user

Goals: Open preferences for current user

Pre-conditions: User must be logged into squire.

Summary: User opens up the settings menu, navigates to preferences.

Related use cases: Change user color, Change password.

Steps:

1. Click the settings button.
2. System opens settings menu.
3. User navigates to preferences.
4. System opens preferences menu.

Alternatives: None

Post-conditions: None.

1.3.31 Edit User Color (benz5834)

Actors:	Squire user
Goals:	Change user color from within preferences
Pre-conditions:	User must have preferences open.
Summary:	User selects their color and saves their changes.
Related use cases:	Open user preferences.
Steps:	1. User clicks select color. 2. System opens color selection window. 3. User selects their color. 4. System updates and previews color. 5. User saves changes.
Alternatives:	User can discard changes by canceling on color selection window.
Post-conditions:	None.

1.3.32 Change User Password (benz5834)

Actors:	Squire user
Goals:	Change user password from within preferences
Pre-conditions:	User must have preferences open.
Summary:	User types a new password and saves their changes.
Related use cases:	Open user preferences.

Steps:

1. User clicks change password.
2. System opens password changing window.
3. User types their new password.
4. System requests confirmation.
5. User saves changes.

Alternatives:	User can discard changes by canceling on confirmation window.
Post-conditions:	None.