Squire: A Collaborative Software Development Tool

jank6275, mora5651, boss2849, bolt1003, gall7417, brec9824, snev7821, mars2681

March 4, 2016

Contents

Chapter 1

Overview and Scope

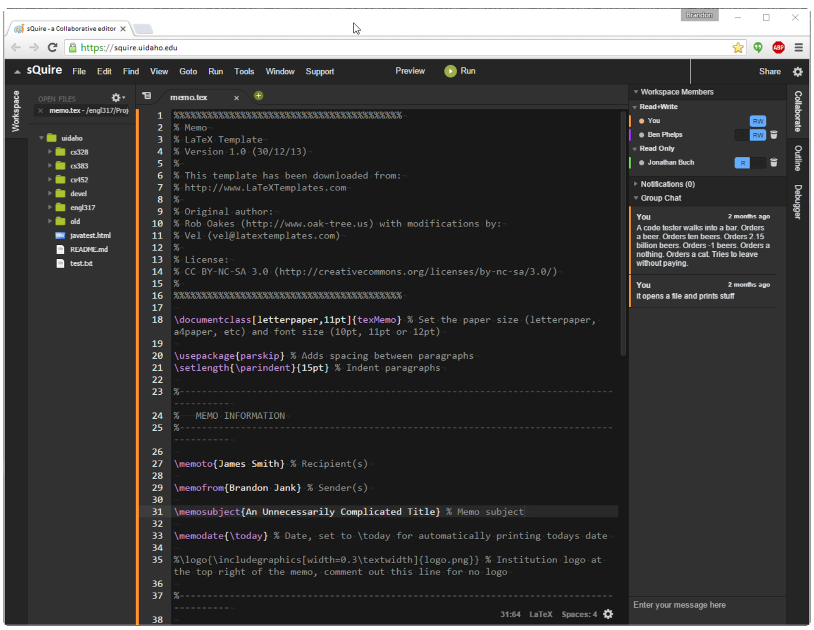


Figure 1.1: "Squire is a web-based collaborative software development environment with a project development center. Squire will allow multiple users to edit files and communicate in real time. Projects can be "stubbed" out by a user and then other users can join and/or vote to support for their favorite 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/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 Storage and Organization

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.

1.3 Execution

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.

1.4 Collaboration

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.

1.5 Achievement System

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).

1.6 Chat System

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.

1.7 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.

1.8 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.

1.9 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.

Chapter 2

Requirements Documentation

2.1 Functional Requirements

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

2.1.1 Authentication (mora5651)

- Sign In
- Sign Up
- Forgot Username
- Forgot Password
- Sign Out

2.1.2 Project Management (bolt1003)

- Admin control features.
- Public and private sharing of project.
- Controlled reading and writing permissions to users.
- Password controls.
- File management.

2.1.3 Project Ideas (mars2681)

- Forum to browse potential projects
- Up- and down-votes for project selection
- Different ways to sort projects (date, projected team size, votes)

- Ability to post a project
- Ability to delete a project (but only by project author)
- Convert potential project stub into active working project

2.1.4 Settings - Preferences and Profile (brec9824)

- Viewable profile by other users and by oneself
- Includes editable email, profile image, password, and personal bio.
- Setting for public, friends only, or private viewing of online status, email address, personal bio, project ownerships, project memberships, and friends list.
- Option for settings users prefered color and shape to be displayed in projects if available.
- Option to have account deleted.
- Direct access to projects that are listed under project ownerships and project memberships.

2.1.5 Compiler (boss2849)

- Compile project sub-modules or entire project.
- Compile and run code within IDE.
- Compile code and package to a JAR.
- Impose temporary code freeze during compilation.

2.1.6 Syntax (gall7417)

- Parse lines of users code as it is written
- Color code various objects in the code such as variables and conditionals
- Report error feedback for any syntactical errors found

2.1.7 Communication (jank6275)

- Global chat when anywhere in program. (Usecase 3.? and Class 4.?)
- Project chat when a project is open. (Usecase 3.? and Class 4.?)
- Closeable project chat. (Usecase 3.? and Class 4.?)
- Closeable global chat. (Usecase 3.? and Class 4.?)

2.1.8 Project File Editor (snev7821)

- Create a project.
- Add new file to project.
- Import existing file in to project.
- Delete a file.
- Delete a project.
- Export a project.
- Export a file.
- Open file in new tab.
- View line numbers.
- View users and user history.
- Find and replace.
- Display the currently typing user.
- Syntax highlighting

2.1.9 Security (gall7417)

- Resource defense strategy that includes not subjecting any sQuire server to becoming unresponsive due to a runaway program, and not allowing any sQuire server to give up shell access via an executing program in sQuire.
- Require authentication to access all user files and user information.
- Ensure confidentiality of all user information.
- Mitigate security threats by testing against common abuse cases and vulnerabilities.
- Ensure proper character sets for all input given.
- Enforce authorization controls on all system requests.
- Restrict access to resources and files outside of the users given resources.

2.2 Non-Functional Requirements

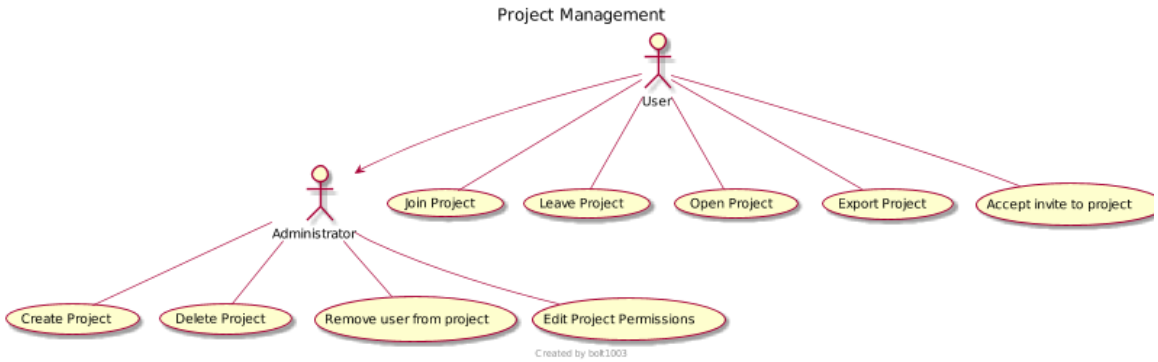
Non-functional requirements cover all the remaining requirements which are not covered by the functional requirements. They specify criteria that judge the operation of a system, rather than specific behaviours. Squire's non-functional requirements are:

- The editor will be multi-user, up to 32 users can share an IDE session. (jank6275)
- The current location of any user's cursor can be quickly jumped to by any user with the project open. (jank6275)
- Text written by the user in the editor should be visible instantaneously. (jank6275)
- Text written by other users in the editor should be visible within 2 seconds. (jank6275)
- Text will be underlined in the user's color, if they edited/created that text. (jank6275)
- The line number will be highlighted in the users color, if they created the line. (jank6275)
- Each line edited by a user should be saved to create a edit history for each user in a document. (jank6275)
- Users will be able to edit the same document concurrently. (jank6275)
- Any user can save the document at any time. (jank6275)
- Scalability is important to keep in mind during development. The system should be designed in such a way that will easily and reliably scale to accommodate a growing user base. One method of dealing with scalability would be allowing the core system to reactively spawn new slaves to aid in computational needs, such as compilation (as that will be more resource intensive the user base grows). (boss2849)
- Limited space for projects that haven't been initiated (enough for documentation, images, etc.) (boss2849)
- Reactively increasing capacity for projects proportional to the absolute needs of the projects. (no fluff) (boss2849)
- Encourage developers to store large files elsewhere, e.g. GitHub LFS, AWS, etc. (boss2849)
- Since sQuire is web based, downtime for the servers most be kept to a minimum and be no more then once or twice a week for a few hours. This downtime will allow for database uprgrades and backups. (brec9824)
- sQuire will leverage technology developed for the web to ensure reliablity. Technologies such as redunant hardware, redunant power providers, redunant internet services, load balancing and virtualization will enable sQuire to be reliable.(bolt1003)

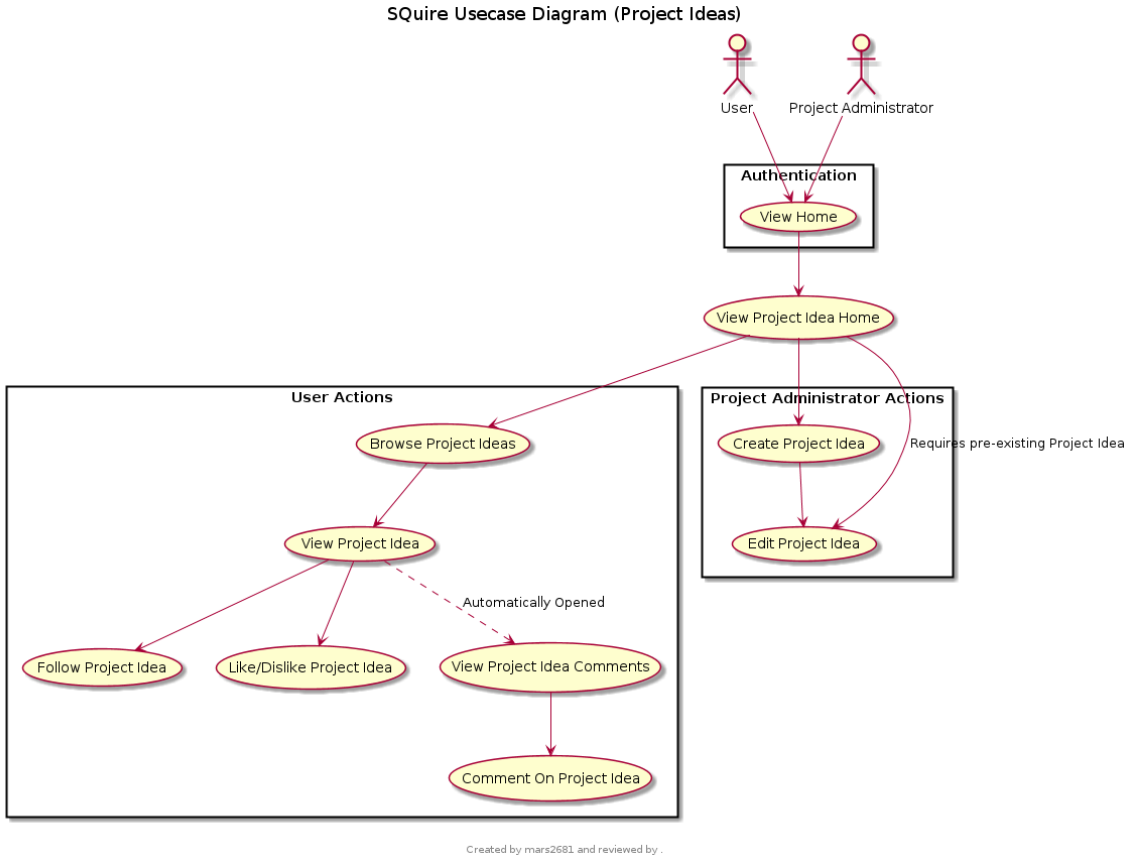
- SQuire will have the ability for the user to save on sQuire's database for recoverability isurance. It will also incorporate autosave feature. (mora5651)
- Since sQuire is a web based application running on a webhost. There are many maintenance tools that can be used to track performance. Maintainability of sQuire will also include regular backup schedules, speed test, and security monotoring. (mora5651)
- sQuire will run in a virtual machine on top of redunant hardware. Using a virtual machine allows for mutliple instance to be running and tested. The backend will run on redunant hardware which will prevent hardware failure from affecting sQuire usage. In turn, allowing infrastructure to be serviced without affecting sQuire. (bolt1003)
- Each project should be containerized so that users can't harm each others projects. (jank6275)
- Upon creation of account, user must agree to terms and conditions. (mars2681)
- User must comply with DMCA (Copy right law) (mars2681)
- User must confirm they are 18 or older (COPPA law) (mars2681)
- Will include monitoring of subsystems to collect and report performance data. (brec9824)
- Subsystem monititory must be highly efficient and take up less than 10% of resources to keep noticeability to a minimum. (brec9824)
- Subsystems must be modular to allow for efficient monitoring, implementation of subsystems, and the addition of new subsystems. (brec9824)
- sQuire will run as a browser application in google chrome. (gall7417)
- sQire will run on virtually all modern processors. (gall7417)
- sQuire will use tcp to for reliable data communications. (gall7417)
- sQuire will use error checking upon large changes to ensure no drastic data corruption occurs. (gall7417)
- sQuire will use a history/autosave feature in case of data loss. (gall7417)
- Provides code compilation and file system for users with limited resources. (snev7821)
- Easy to learn system, helpful syntax highlighting. (snev7821)
- More satisfying than competitors products, because of social/kickstarter aspect. (snev7821)
- Provides github integration. (snev7821)
- Runs on any of the main browsers (chrome, firefox, safari). (snev7821)
- Due to web based design, works on lower end machines. (snev7821)

2.3 Use Case Diagrams

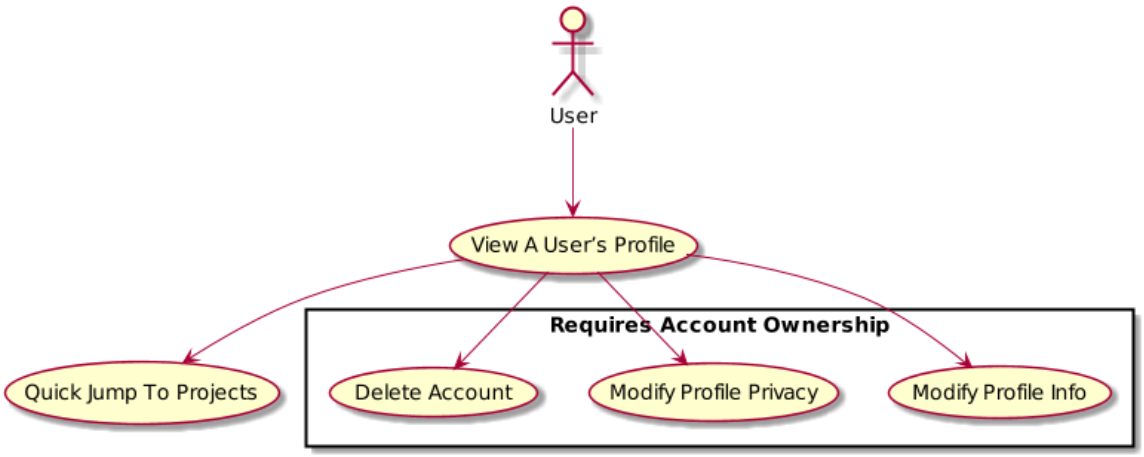
2.3.1 Project Management (bolt1003)



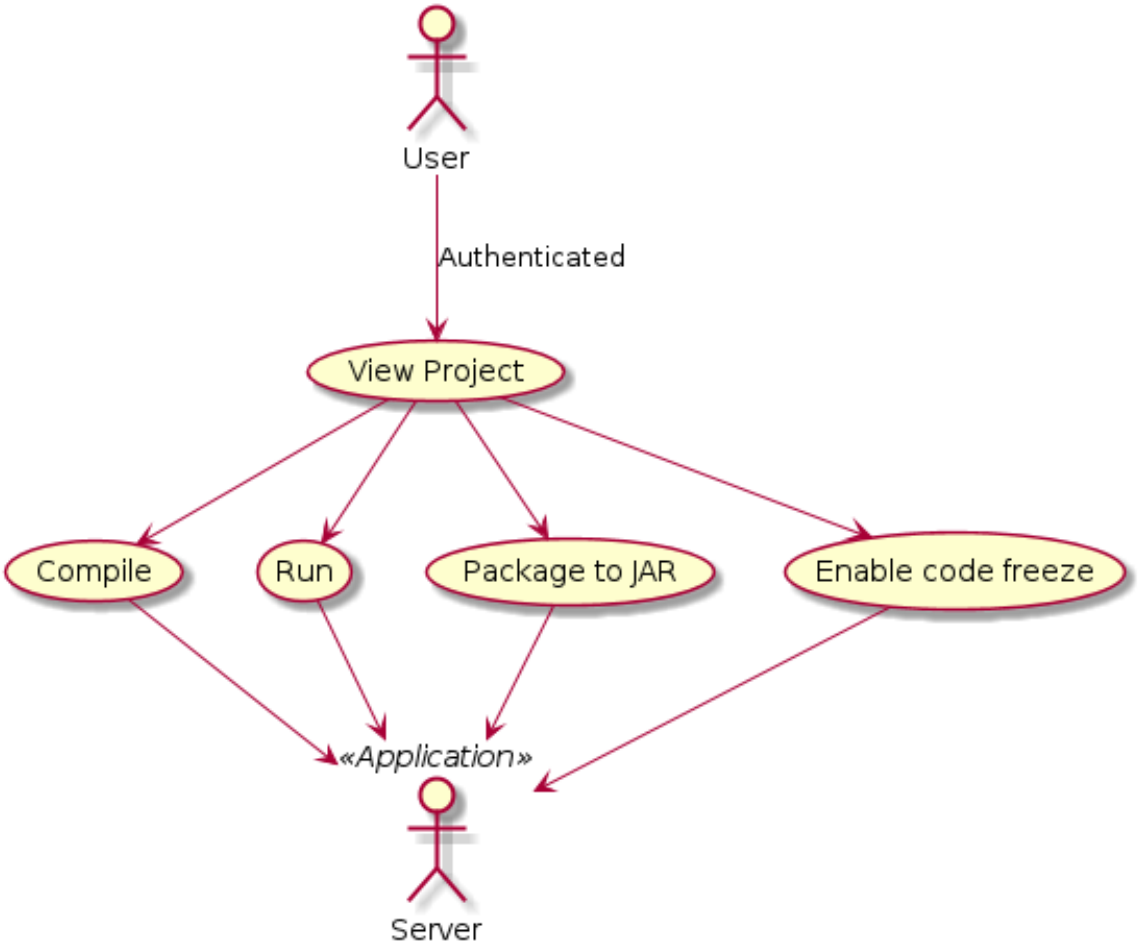
2.3.2 Project Ideas (mars2681)



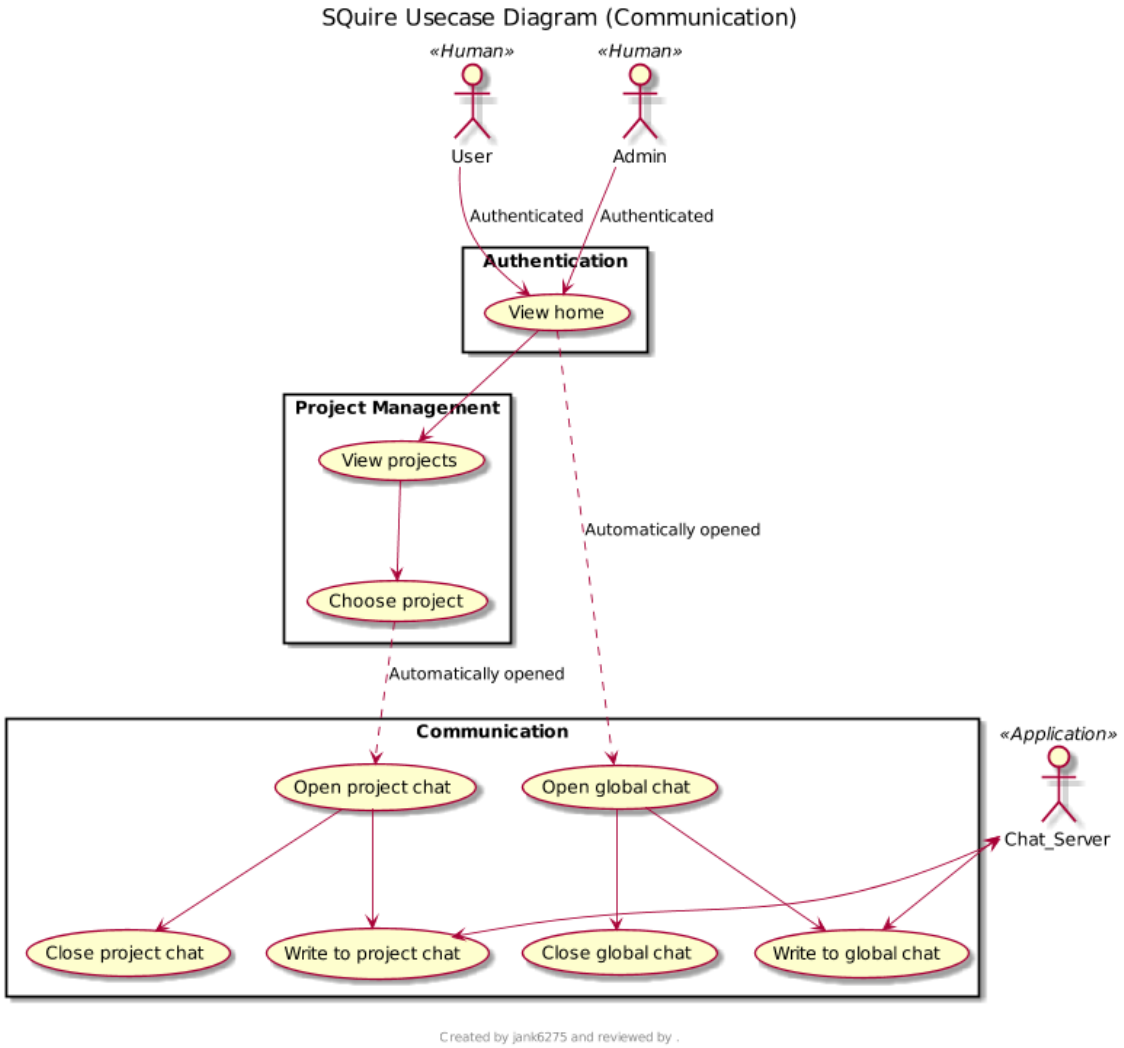
2.3.3 Settings, Preferences, and Profile (brec9824)



2.3.4 Compiler (boss2849)

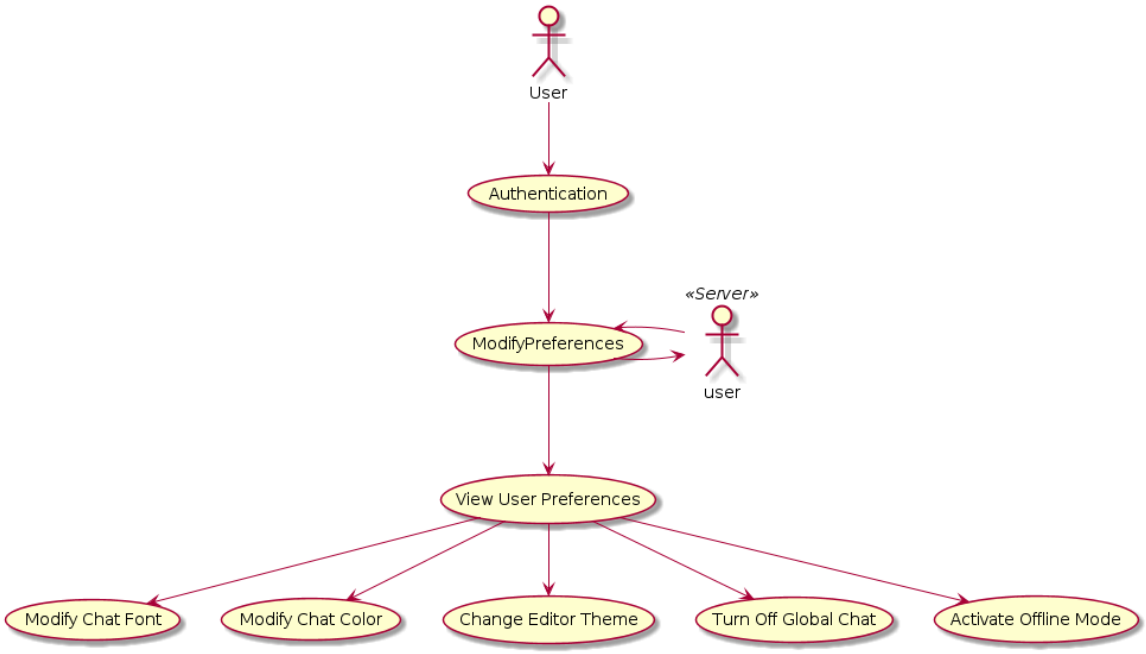


2.3.5 Communication (jank6275)



A usecase diagram for sQuireś communication features.

2.3.6 User Editor Preferences (snev7821)



Chapter 3

Use Case Descriptions

3.1 Authentication (mora5651)

3.1.1 Sign up (mora5651)

Actors:	User.
Goals:	To register and create an account in sQuire.
Pre-conditions:	None.
Summary:	The user signs up and creates an account using their email address and, creates a username and password.
Related use cases:	None.
Steps:	1. User is prompted to enter email, username and password. 2. System sends confirmation email. 3. User verifies email. 4. System saves information, and redirectes user to sign in page.
Alternative 1:	User already has an account.
Alternative 2:	User doesn't confirm email. Delete request after timeout period.

3.1.2 Sign in (mora5651)

Actors: Users.

Goals: Pre-existing user signs into profile.

Precondition: User must already have an account

Summary: User wishes to access their account, projects and info.

Steps:

1. User is prompted to enter username/e-mail, and password.
2. System verifies information
3. Correct information prompts user to their home page.

Alternatives: Information is incorrect, user tries again. Or makes a new account

3.1.3 Logout (mora5651)

Actors:	Users.
Goal:	Existing user logs out
Precondition:	User must be logged in
Summary:	The user can log-out of the program at any time.
Steps:	1. User clicks the "log-out" button. 2. System prompts user to ensure all unsaved work has been saved. 3. User verifies. 4. System logs user out.
Alternative 1:	The program will send notification to ask if the user is sure to sign out.
Alternative 2:	User cancels on step two. Return to home page.

3.1.4 Forgotten Username (mora5651)

Actors:	Users.
Goal:	Recover forgotten username.
Precondition:	User must already have an account.
Summary:	User has forgotten their username, and wishes to recover it.
Steps:	1. User clicks the "Forgotten username" button. 2. User inputs their email address. 3. System validates their email address with an account, and sends an email with the username.
Alternative 1:	Information is incorrect, user tries again. Or makes a new account.

3.1.5 Forgotten Password (mora5651)

Actors:	Users.
Goal:	Recover forgotten password.
Precondition:	User must already have an account.
Summary:	User has forgotten their password, and wishes to recover it.
Steps:	1. User clicks the "Forgotten password" button. 2. User inputs their email address. 3. System validates their email address with an account. 4. System sends an email for the password reset.
Alternative 1:	Information is incorrect, user tries again. Or makes a new account.

3.2 Project Ideas (dani2918)

3.2.1 Browse Project Ideas (dani2918)

Actors: User

Goals: Examine a list of available project ideas

Pre-conditions: User is signed in

Summary: User looks through posted project ideas to find projects to work on/discuss

Related use cases: Comment on Project Idea, Vote on Project Idea, Share Project Idea

Steps:

1. Actor selects Browse Project Ideas button
2. Actor refines search by selecting from list of project categories as desired
3. Actor enters terms into search field as desired and views a list of top projects
4. Actor selects desired project
5. System displays detailed project information

Alternatives: None.

Post-conditions: None.

3.2.2 Create Project Idea Thread (dani2918)

Actors:	Project Administrator
Goals:	Generate public interest in project idea
Pre-conditions:	Prospective project administrator is signed in
Summary:	A user with interest in heading up own project can post ideas to get feedback and/or recruit collaborators
Related use cases:	Manage Project Idea Thread
Steps:	1. Actor selects Create Project Idea button 2. Actor enters prospective project title and thoughts and ideas as a description 3. Actor selects Submit button
Alternatives:	None.
Post-conditions:	None.

3.2.3 Manage Project Idea Thread (dani2918)

Actors:	Project Administrator
Goals:	Respond to feedback and manage project idea
Pre-conditions:	Prospective project administrator is signed in and has navigated to one of his or her own project threads
Summary:	A prospective project administrator responds to others' questions/comments
Related use cases:	Create Project Idea Thread
Steps:	1. Actor selects Reply button on a comment 2. Actor types feedback to other user 3. Actor selects Submit button 4. System shows confirmation that feedback was received
Alternatives:	Actor selects Delete Comment instead of Reply to remove harmful feedback.
Post-conditions:	None.

3.2.4 Comment on Project Idea (dani2918)

Actors:	User
Goals:	Provide detailed feedback on project ideas
Pre-conditions:	Actor is signed in, has navigated to a project idea
Summary:	User provides feedback to or asks questions about a prospective project.
Related use cases:	Browse Project Ideas, Vote on Project Idea, Manage Project Idea Thread
Steps:	1. Actor selects Comment button 2. Actor types feedback into field 3. Actor clicks Submit button 4. System shows confirmation that feedback was received
Alternatives:	None
Post-conditions:	None.

3.2.5 Vote on Project Idea (dani2918)

Actors:	User
Goals:	Support promising project ideas or offer criticism to unfavorable ones
Pre-conditions:	Actor is signed in, has navigated to a project idea
Summary:	User offers support/discourages a project idea so that prospective project administrators get feedback and promising project ideas get publicity
Related use cases:	Comment on Project Idea
Steps:	1. Actor selects and Up Vote or Down Vote button 2. Actor selects Submit button 3. System highlights which button the user has selected
Alternatives:	None
Post-conditions:	None.

3.2.6 Share Project Idea (dani2918)

Actors:	User, Project Administrator
Goals:	Generate excitement about a project
Pre-conditions:	Actor is signed in, has navigated to a project idea
Summary:	Prospective project administrators or users show other users which projects they believe are worthwhile
Related use cases:	Comment on Project Idea
Steps:	1. Actor selects Share button 2. Actor selects an audience and method with which to share selected project 3. Actor selects Submit button 4. System notifies or shows audience the shared project idea
Alternatives:	None
Post-conditions:	None.

3.3 Communication (jank6275)

3.3.1 Open project chat (jank6275)

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.

3.3.2 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 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.

3.3.3 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.

3.3.4 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.

3.3.5 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.

3.3.6 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.

3.3.7 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 userś selection is saved in the database. 7. All further chat messages will use the selected font.
Alternatives:	None.
Post-conditions:	None.

3.3.8 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 userś selection is saved in the database. 7. All further chat messages from the user will use the selected color.
Alternatives:	None.
Post-conditions:	None.

3.4 Project Management

3.4.1 Create Project (Created by bolt1003)

Actors:	Users of sQuire.
Goals:	Create a Project.
Pre-conditions:	The user is logged in and at the dashboard.
Summary:	The user creates a project.
Related use cases:	None.
Steps:	1. User selects the "+" icon and a wizard appears. 2. A name is choosen for the project. 3. Language is selected from a drop down menu. 4. User clicks finish.
Alternatives:	Create project from the editor.
Post-conditions:	The user assigns permissions to access the project.

3.4.2 Open a project (Created by bolt1003)

Actors:	Users of sQuire.
Goals:	Choose the desired project and open it.
Pre-conditions:	One or more projects are available, the user is logged in and at the dashboard.
Summary:	User looks through a list of projects and selects the desired project.
Related use cases:	None.
Steps:	1. User clicks on projects in the menu bar. 2. A list of projects appears and the user clicks on the desired project.
Alternatives:	Open a project from recent projects.
Post-conditions:	User closes sQuire.

3.4.3 Join Project (Created by sass8427, Revised by bolt1003)

Actors:	Users of sQuire.
Goals:	Join an existing project.
Pre-conditions:	Must be registered, logged in and have permission to join a project.
Summary:	The user logs in, chooses a project, and joins the project.
Related use cases:	Invite user to project, Accept user invite.
Steps:	1. The user selects a project. 2. The user chooses the "Join". 3. The project is added to the users projects bar. 4. The user selects the project and selects "open".
Alternatives:	User may decline an inventation to join a project.
Post-conditions:	None

3.4.4 Leave project (Created by sass8427, Revised by bolt1003)

Actors:	User
Goals:	Remove member status from project.
Pre-conditions:	Logged in, member of the respective project, not project owner.
Summary:	A member of a project can unjoin that project at any time as long as they are not the project owner. To prevent mistakenly unjoining a project, the user is asked to confirm their decision.
Related use cases:	
Steps:	1. User selects a project. 2. User clicks "Unjoin". 3. User is promted to confirm their decision 4. User clicks "Confirm". 5. User is removed from project member list.
Alternatives:	User clicks "Cancel" at step 4, in which case the task is ends at that point.
Post-conditions:	None.

3.4.5 Delete Project(Created by mora5651, Revised by bolt1003)

Actors:	Users of sQuire.
Goals:	Delete an existing project.
Pre-conditions:	The user has the appropriate permissions to delete project.
Summary:	A user deletes a project from the project workspace.
Related use cases:	Create a project.
Steps:	1. The user selects a project. 2. The user clicks on the "Delete project" button. 3. A dialog is displayed. 4. User select "delete" to delete the project.
Alternatives:	User may choose not to delete the project in the confirmation display.
Post-conditions:	None.

3.4.6 Export Project(Created by knic1468, Revised by bolt1003)

Actors:	User of sQuire.
Goals:	Export a workspace to a local file.
Pre-conditions:	The user needs permission to export the project.
Summary:	User saves a file containing the project settings and files to a local machine.
Related use cases:	Importing a project, Creating a new project.

Steps:

1. The user clicks on the "Export File" button.
2. System prompts the user to select a location and filename.
3. User selects a file location.
4. The user enters a file name.
5. The user selects "export".

Alternatives:	The user cancels the export, The system prompts that a file already exists with the same name.
Post-conditions:	None.

3.4.7 Accept Invite to Project (Created by carl7595, Revised by bolt1003)

Actors:	User who received the invite.
Goals:	Gain access to a Project.
Pre-conditions:	User has a valid email address.
Summary:	Access is granted to a project using an invitation email.
Related use cases:	Create an account.
Steps:	1. Invitee clicks on the link received by email. 2. The link opens in a browser. 3. Dialog appear welcoming them to the project. 4. The project is added to their Projects list.
Alternatives:	The user ignores the invite.
Post-conditions:	Email link is deactivated.

3.4.8 Remove User from Project (Created by carl7595, Revised by bolt1003)

Actors: User of sQuire

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

Pre-conditions: The user has permission to edit the Project access list.

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

Related use cases: Add users to a project.

Steps:

1. The user selects the access list for the project.
2. The user selects an account.
3. The user selects "Remove from Project".
4. The user is prompted for confirmation
5. The user selects 'Yes'.

Alternatives: The user selects 'No' and the access list is not modified.

Post-conditions:

- The user that was removed is notified of the change.
- The user is prevented from accessing files.

3.4.9 Edit Project Permissions (Created by benz5834, Revised by bolt1003)

Actors:	User of sQuire
Goals:	Edit the permissions for a project
Pre-conditions:	The user is logged in.
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. The user selects a project. 2. The user selects settings. 3. The user selects permissions. 4. The user selects user from list of users. 5. The user adds read or write permissions to user. 6. The user selects save to save changes. 7. The user exits settings.
Alternatives:	User can remove read or write permission instead in step 6. User can discard changes instead in step 7.
Post-conditions:	None.

3.4.10 Modify read/write access (boss2849)

Actors:	User
Goals:	Modify a userś permissions.
Pre-conditions:	User is signed in and holds Admin rights for the currently selected Project
Summary:	User modifies another Userś read/write permissions to portions of the project.
Related use cases:	None.
Steps:	1. User clicks Permissions Management 2. System displays permissions management window 3. User selects a file, multiple files, directory or entirety of project and grants/revokes read/write access 4. System modifes the target Userś permissions and notifies them.
Alternatives:	3. User selects cancel, System discards changes.
Post-conditions:	None.

3.4.11 Remove User (boss2849)

Actors:	User
Goals:	Remove a user from project.
Pre-conditions:	User is signed in, in project with Admin rights, and is on User Management page
Summary:	User removes a selected user from the Project
Related use cases:	Invite User, Modify Read/Write Access
Steps:	1. User clicks Remove User button. 2. System displays list of active users for project. 3. User selects one or more other users from the list and presses Remove. 4. System prompts User for verification. 5. User presses Confirm. 6. System removes the selected users from the project. 7. System revokes read and write access from the selected users. 8. System notifies selected users that they have been removed from the project.
Alternatives:	User presses Cancel in steps 3 or 5, System returns user to User Management page
Post-conditions:	None.

3.4.12 Invite User to Project (boss2849)

Actors:	User
Goals:	Invite user(s) to project
Pre-conditions:	User is signed in, in project with Admin rights, and is on User Management page
Summary:	User invites user(s) to the current project.
Related use cases:	Remove User, Join Project

Steps:

1. User clicks Invite Users button
2. System prompts user to enter username(s)/email(s)
3. User enters username(s)/email(s) of the user(s) to invite and presses Ok.
4. System looks up the specified user(s) and notifies them of invitation to the Project

Alternatives:

1. User presses cancel in step 3, System returns User to User Management page
2. In step 4, username(s)/email(s) donť match any users, System notifies User of failed invitiations.

Post-conditions:	None.

3.4.13 Promote User to Admin (boss2849)

Actors:	User
Goals:	Promote a specified User to Admin
Pre-conditions:	User is signed in, in project with Admin rights, and is on User Management page
Summary:	User selects another User to be given Admin rights for the project.
Related use cases:	Demote Admin
Steps:	1. User selects Promote to Admin. 2. System displays a list of non-Admin active users. 3. User selects user(s) and presses Submit. 4. System prompts user for confirmation. 5. User selects Confirm. 6. System grants Admin permissions to the selected user(s).
Alternatives:	User presses cancel in steps 3 or 5, no action taken.
Post-conditions:	None.

3.4.14 Demote Admin (boss2849)

Actors: User

Goals: Demote Admin to user

Pre-conditions: User is signed in, in project with Admin rights, and is on User Management page

Summary: User demotes selected Admins to normal Users for the project.

Related use cases: Promote User to Admin

Steps:

1. User selects Demote Admin
2. System displays list of Admins
3. User selects Admin(s) to demote and presses Submit.
4. System prompts User for confirmation.
5. User presses Confirm.
6. System revokes Admin rights from selected User(s)

Alternatives:

1. User presses cancel in steps 3 or 5, no action taken
2. User attempts to demote Admin that is the Owner of the project, System rejects request and notifies User.

Post-conditions: None.

3.4.15 Block User (boss2849)

Actors:	User
Goals:	Block a user from the project
Pre-conditions:	User is signed in, in project with Admin rights, and is on User Management page
Summary:	User blocks a user from the project, making them unable to view the project.
Related use cases:	Demote Admin
Steps:	1. User clicks Block User. 2. System displays a list of active users. 3. User selects other user(s) to block and presses Submit. 4. System prompts User for confirmation. 5. User presses Confirm. 6. System blocks selected user(s) from the project, revoking read/write access, and revoking Admin status as necessary.
Alternatives:	User presses cancel in steps 3 or 5.
Post-conditions:	None.

3.5 Settings - Preferences and Profile (brec9824)

3.5.1 View A User's Profile (brec9824)

Actors: User of sQuire.

Goals: User views a profile page.

Pre-conditions:

1. The user is logged in.

Summary: User clicks on their username or another user's name and a goes to a new page with the selected user's profile page.

Related use cases: Modify Profile Info.

Steps:

1. The user clicks on their username or another user's name.
2. The user's system sends a request to the main sQuire system for the selected users profile information.
3. sQuire system approves the request and sends the selected user's full profile info.
4. The user's system loads a new page displaying the selected user's full profile info.

Alternatives: In step 3 sQuire approves the request but because of the selected user's privacy settings only partial profile info is sent to the user.

Post-conditions: None.

3.5.2 Modify Profile Info (brec9824)

Actors: User of sQuire.

Goals: User updates their profile info including project preferences.

Pre-conditions:

1. The user is logged in.
2. The user is at their profile page.

Summary: User clicks on the edit button, modifies their info, clicks save and their info gets saved.

Related use cases: Modify Profile Privacy.

Steps:

1. The user clicks the edit button.
2. The user's system sends a request to the sQuire system.
3. sQuire system recieves the request and verifies the user's credentials.
4. The user's system loads a new page displaying the user's profile but with editable text boxes.
5. The user edits their desired info.
6. The user clicks the save button and the user's system sends the updated info to the sQuire system.
7. sQuire system recieves the new data, verifies the data meets predefined requirements and aprroves the update.
8. User is returned to their profile page as before with their updated info.

Alternatives:

1. In step 3 the sQuire system denies the request because the user was idle to long and is not logged in anymore.
2. In step 7 the sQuire system denies the user's request to update their profile: 1. the user's email was invalid 2. the user's password didn't meet the security requirements 3. the user used ineligible words or phrases. User is notified of the denial and is returned to step 5.

3.5.3 Modify Profile Privacy (brec9824)

Actors: User of sQuire.

Goals: User updates their profile info.

Pre-conditions:

1. The user is logged in.
2. The user is at their profile page.

Summary: User clicks on the privacy level checkbox, clicks save and their new privacy level is saved.

Related use cases: Modify Profile Info.

Steps:

1. The user clicks the appropriate privacy checkbox next to the data they wont to change the privacy of.
2. The user clicks the save button.
3. sQuire system recieves the request and verifies the user's credentials.
4. sQuire system recieves the updated privacy settings and aprroves the update.
5. User is returned to their profile page as before with their updated info.

Alternatives:

1. In step 3 the sQuire system denies the request because the user was idle to long and is not logged in anymore.

Post-conditions: None.

3.5.4 Delete Account (brec9824)

Actors: User of sQuire.

Goals: Delete the user's account.

Pre-conditions:

1. The user is logged in.
2. The user is at their profile page.

Summary: User clicks on the delete account button, then confirms their choice and their account is deleted from sQuire servers after a set amount of time.

Related use cases: Modify Profile Info.

Steps:

1. The user clicks the delete account button.
2. System covers the room window with a new window that is dark and nearly transparent.(Gives the appearance that the page is dimmed)
3. System opens a pop-up window that contains a confirm button, a cancel button, and text that asks the user if they are sure and notifies them that this action is permanent.
4. The user clicks the submit button.
5. System closes the pop-up windows and the dim window in the background.
6. System kicks the user from their account and adds the account to a list for future deletions.
7. User is returned to sQuire's home page.

Alternatives:

1. If the user in step 4 clicks cancel or clicks out of the pop-up window and onto the dim window in the background. The dim window created in step 2 and the pop-up window in step 3 closes and action is taken.

Post-conditions: None.

3.5.5 Quick Jump To Projects (brec9824)

Actors: User of sQuire.

Goals: Go to a projects page that is listed in a user's profile.

Pre-conditions:

1. The user is logged in.
2. The user is at a user's profile page.

Summary: User clicks on the appropriate project name and then is redirected to the projects home page.

Related use cases: None.

Steps:

1. The user searches through the projects listed in the user's profile page and clicks the project name they would like to go to.
2. System recieves the request and searches for the specified project in the project data base.
3. System finds the project and sends the redirect info.
4. The user is then redirected to the given projects home page screen.

Alternatives:

1. None.

Post-conditions: None.

3.6 Compile (boss2849)

Actors:	User
Goals:	Compile source
Pre-conditions:	User is logged in and viewing project.
Summary:	User requests that the code be compiled, the server compiles the code.
Related use cases:	Run, Compile To Jar
Steps:	1. User selects "Compile" from "Build" dropdown menu for the current module. 2. The Server receives the request to compile. 3. The Server caches the current state of the project using the SnapshotManager and compiles it using the active CompilerPlugin. 4. The Server returns the results of compilation to the User.
Alternatives:	In step 1, user chooses to compile entire project, including all sub modules.
Post-conditions:	None.

3.7 Run (boss2849)

Actors:	User.
Goals:	Run the program.
Pre-conditions:	User is logged in and viewing a project.
Summary:	User chooses to run the program and the server compiles it or executes the last compilation result if no changes.
Related use cases:	Compile
Steps:	1. User selects "Run" from the "Build" menu drop down. 2. The Server receives the request to execute. 3. The Server retrieves the most recent compilation from the SnapshotManager. 4. The Server spawns a new window for the client that is the interface to the program.
Alternatives:	In step 3 the SnapshotManager has either an out of date compilation or no last compilation, the Server invokes the compiler to compile the project.
Post-conditions:	None.

3.8 Package to Jar (boss2849)

Actors:	User
Goals:	Compile and package source to a jar
Pre-conditions:	User is signed in and viewing a project.
Summary:	User selects to build the project to a jar, the server outputs a jar on the project path.
Related use cases:	Compile
Steps:	1. The user seclects "Compile To Jar" from "Build" dropdown menu. 2. The Server receives the request to build a jar. 3. The Server fetches the most recent compilation from the SnapshotManager. 4. The Server packages the result of the last compilation to a jar and outputs it on the project path. 5. The Server notifies the user of success.
Alternatives:	In step 3 the SnapshotManager has either an out of date compilation or no last compilation, the Server invokes the compiler to compile the project. In step 4 or 5, the compilation process fails and the Server notifies the user with the reason of failure.
Post-conditions:	None.

3.9 Enable code freeze (boss2849)

Actors:	User
Goals:	Impose a code freeze on the project.
Pre-conditions:	User is signed in, viewing a project, and has admin rights.
Summary:	User places a code freeze on the project, preventing editing until undone.
Related use cases:	None.
Steps:	1. The User selects "Code Freeze" from the dropdown menu. 2. The Server receives the request for code freeze. 3. The Server restricts all editing of project files.
Alternatives:	None.
Post-conditions:	None.

Chapter 4

Class Diagrams

4.1 Overview (brec9824, jank6275)

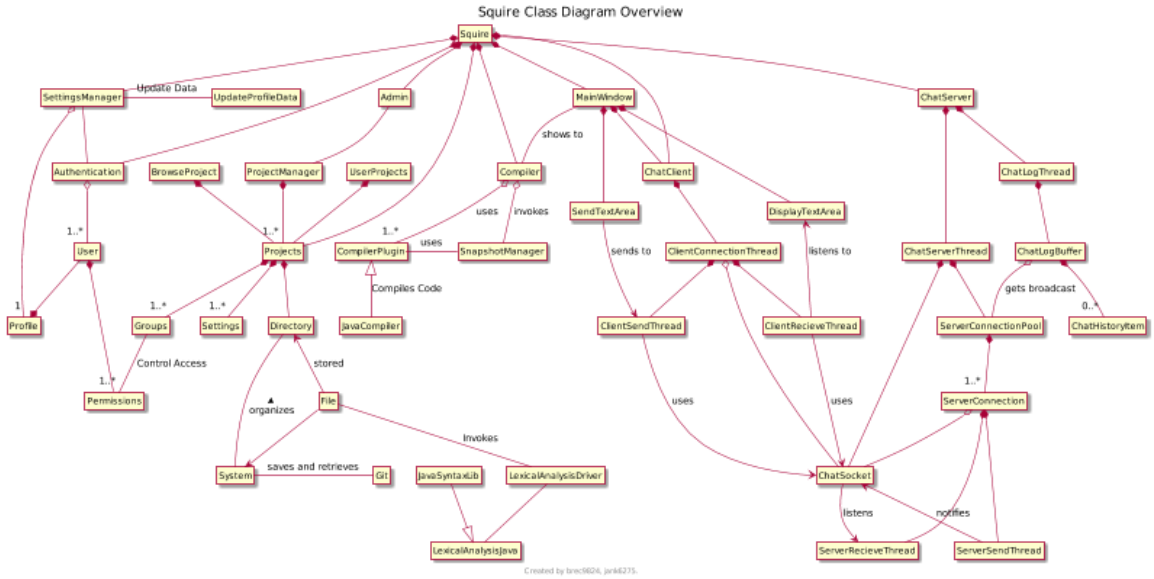


Figure 4.1: Class overview of Squire displaying the connections between each subsystem and their classes.

4.2 Authentication (mora5651)

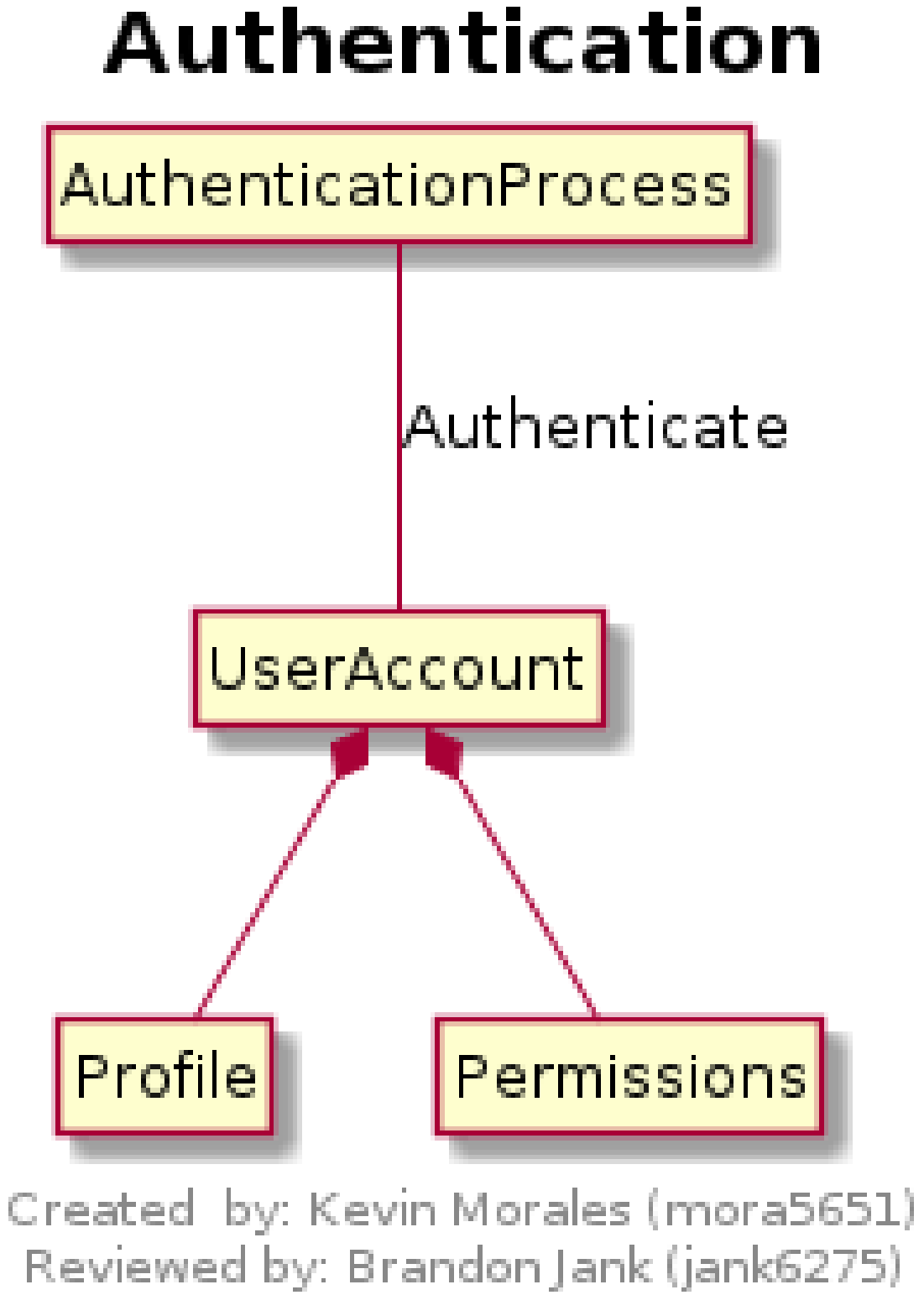


Figure 4.2: Authentication is a security precaution in sQuire that allows the system to verify a user. Users will provide a username and password, and will be validated on the server side.

4.3 Project Management (bolt1003)

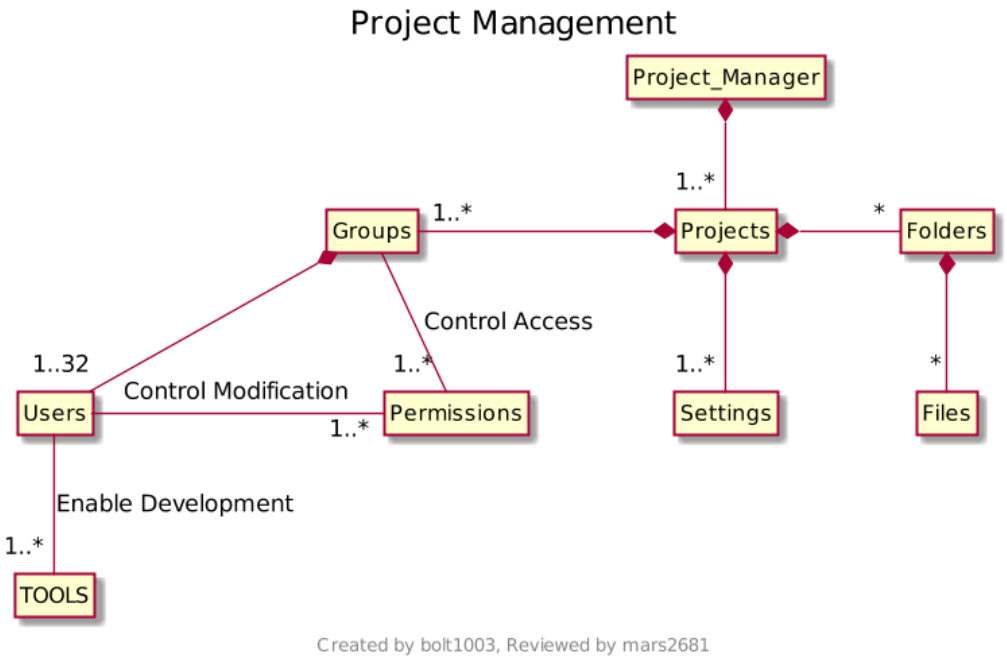


Figure 4.3: Project management allows for the group to delicate permissions and create projects

4.4 Project Ideas (mars2681)

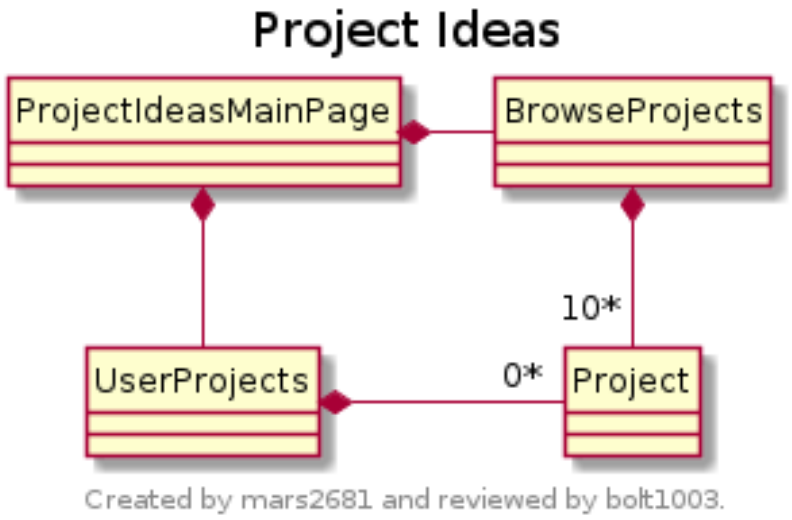


Figure 4.4: The Project Ideas page allows the user to browse other user's projects ideas, as well as create and customize their own project ideas. It starts at the main page. From here, the user can select to view their own projects, and from there create, delete, or edit their ideas. Also at the main page, users can view other projects. Here will be categories of projects, the most popular projects, and new projects. Through browsing these pages, the user can see the description of projects, the amount of support, likes, dislikes, and they can follow and pledge their assistance to the project.

4.5 Settings, Preferences, and Profile (brec9824)

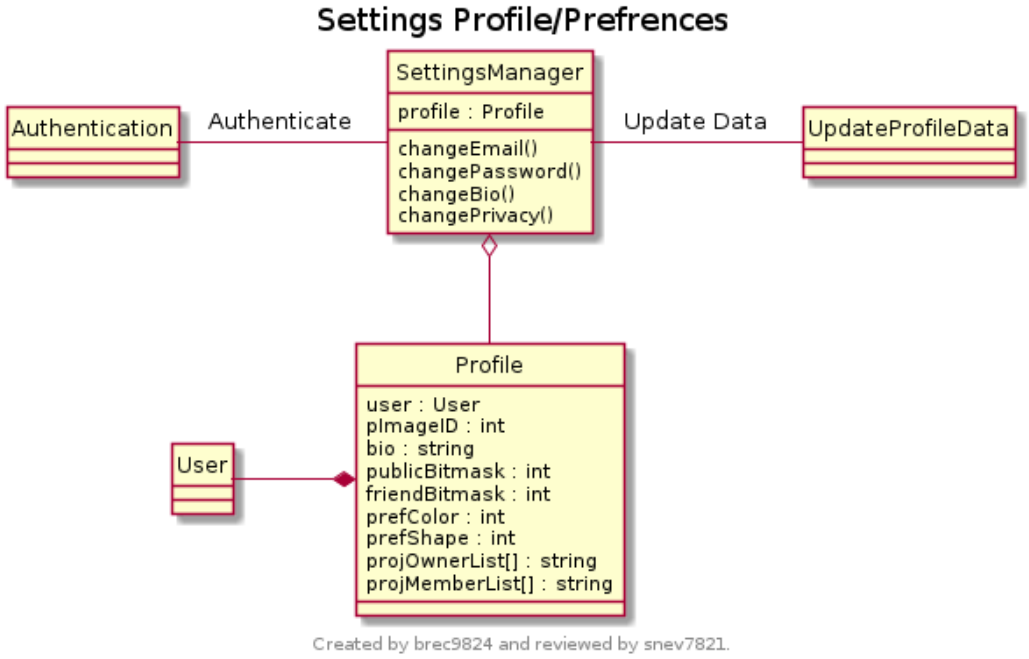


Figure 4.5: Settings Profile/Prefrences allows for profile viewing and management while maintaining speed with the use of push updates. SettingsManager uses Authentication to verify valid input and to authentic the users data. While SettingsManager uses UpdateUser-Profile to push the users data that needs to updated to the server.

4.6 Compiler (boss2849)

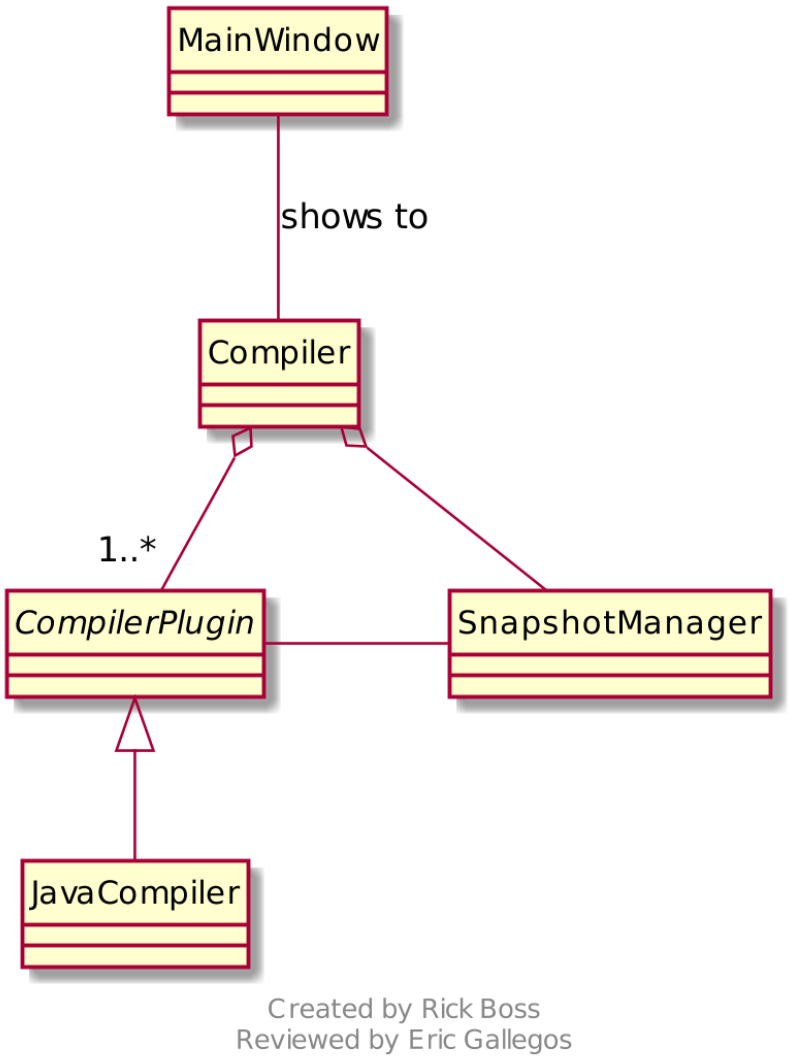


Figure 4.6: The Compiler is invoked from the main window. From here, the Compiler will select the appropriate CompilerPlugin determined by the configured compiler for the project. The compiler also invokes the SnapshotManager, which stores the current state of the code to be used during compilation. In this simple example, only a JavaCompiler plugin is present, but there can be more than one plugins available in the future. The JavaCompiler plugin retrieves the code snapshot from the SnapshotManager and compiles the code.

4.7 Syntax (gall7417)

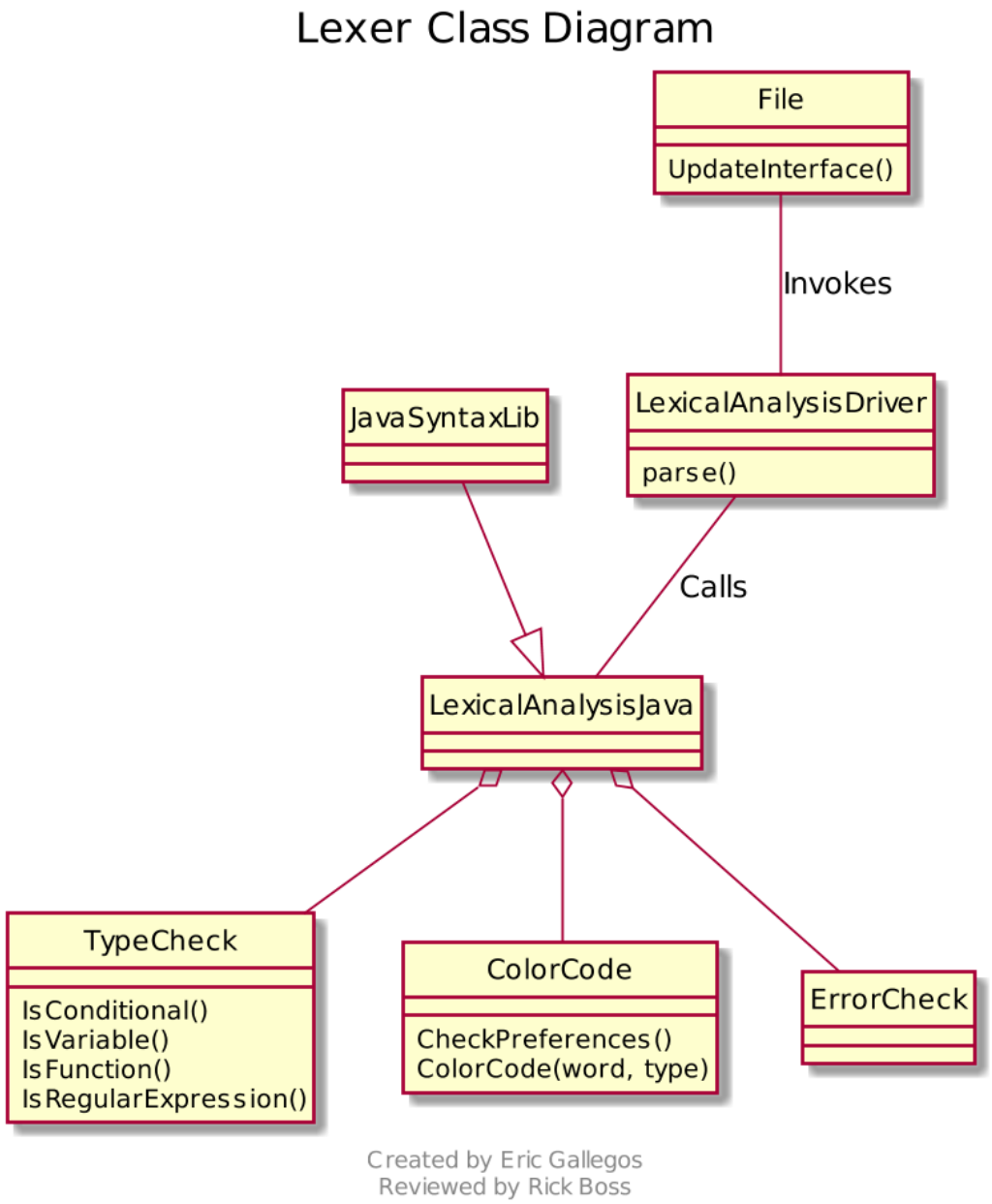


Figure 4.7: The File class regularly invokes the Lexical Analysis class to give feedback to users code input. The Lexical Analysis driver calls the corresponding Lexical Analysis class for the language used (Java). This language specific class checks for errors using the ErrorCheck class, searches for word types using the TypeCheck class, and will assign the various words colors to reflect the word types using the ColorCode class.

4.8 Communication (jank6275)

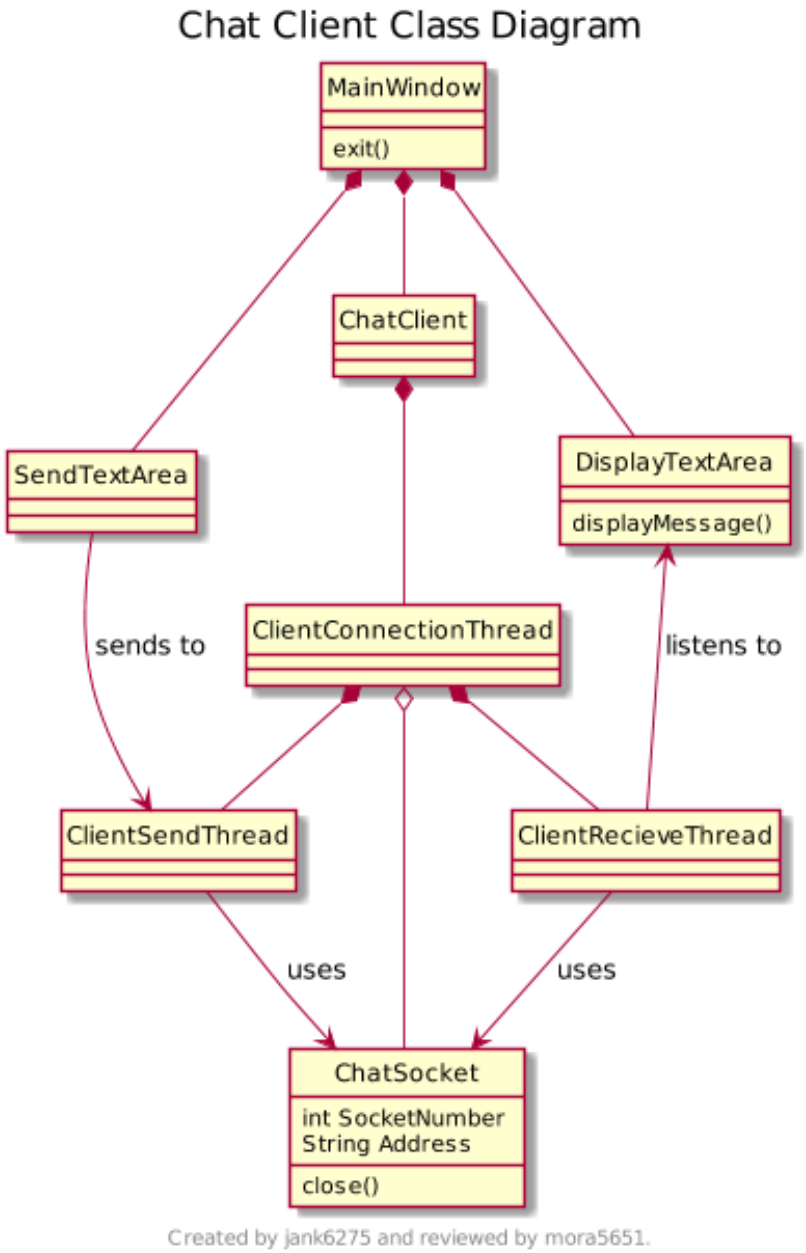


Figure 4.8: The ChatClient class will handle text communication in conjuction with the ChatServer class. The Main Window will be home to the ChatClient. The ChatClinet will consist of client send/recieve threads to handle user input/output in the Main Window via the ChatSocket.

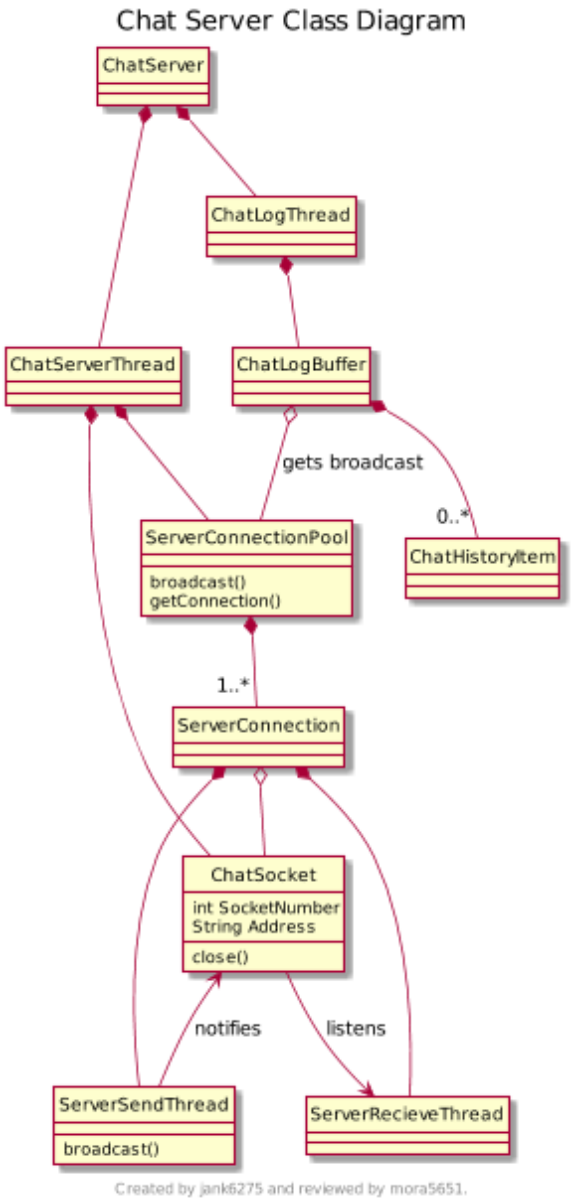


Figure 4.9: The ChatServer class will handle text communication between ChatClient(s). The ChatLogThead records any messages broadcast by chat clients in the ServerConnection-Pool as a ChatHistoryItem. Each ServerConnection consists of a send and recieve thread that utilize the ChatSocket to broadcast messages.

4.9 Project File Editor (snev7821)

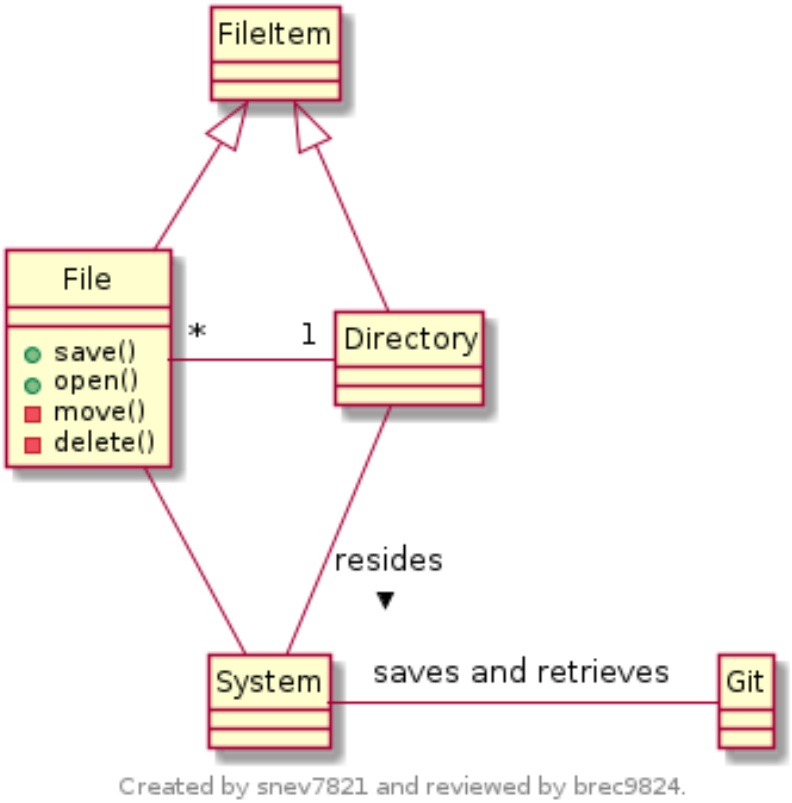


Figure 4.10: The project File Editor is simply the file system for Squire. It provides basic read/write permission for any user related to a project. Only admins of a project may move project files, delete old files, and create new files.

Chapter 5

Sequence Diagrams

5.1 Authentication (jank6275)

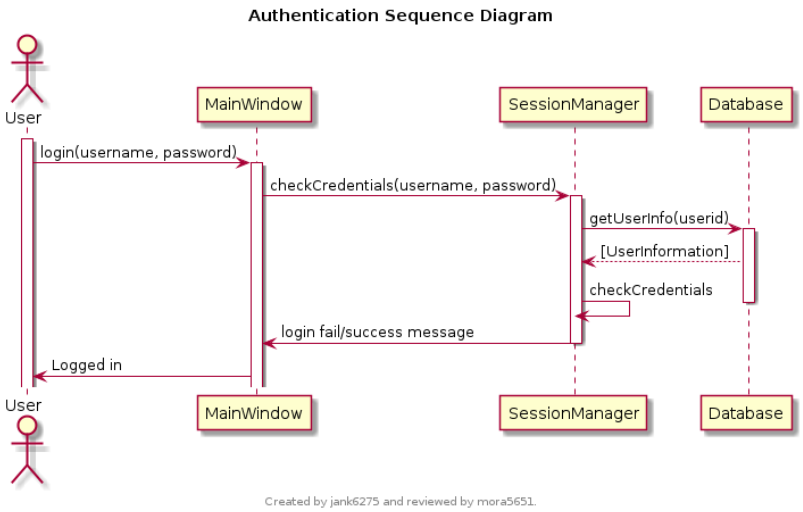


Figure 5.1: A sequence diagram for logging in to squire.

5.2 Project Management (mars2681)

Figure 5.2: Authentication authenticates stuff.

5.3 Project Ideas (snev7821)

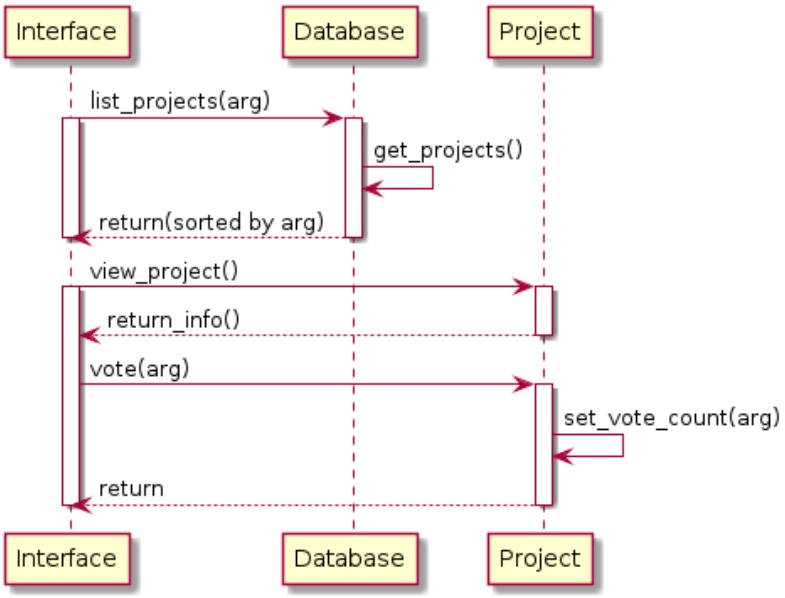


Figure 5.3: Authentication authenticates stuff.

5.4 Settings, Preferences, and Profile (gall7417)

Figure 5.4: Authentication authenticates stuff.

5.5 Compiler (bolt1003)

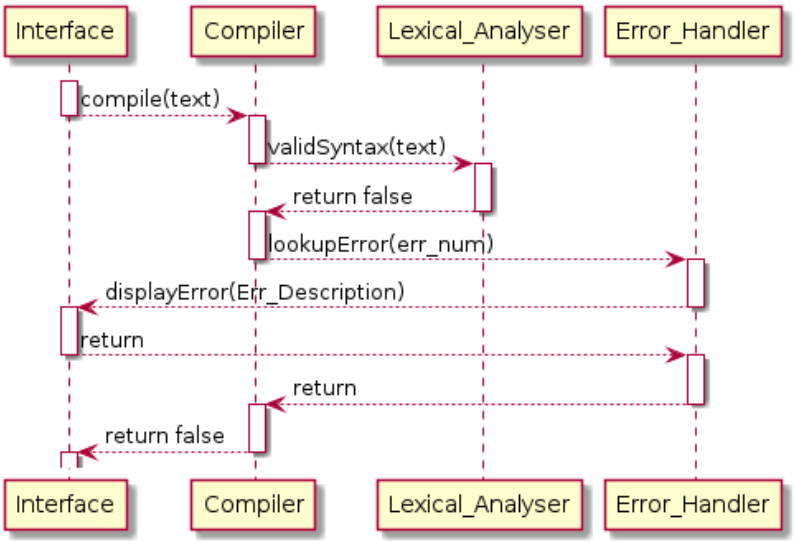


Figure 5.5: The sequence of a user attempting to compile a program with a syntax error.

5.6 Syntax (mora5651)

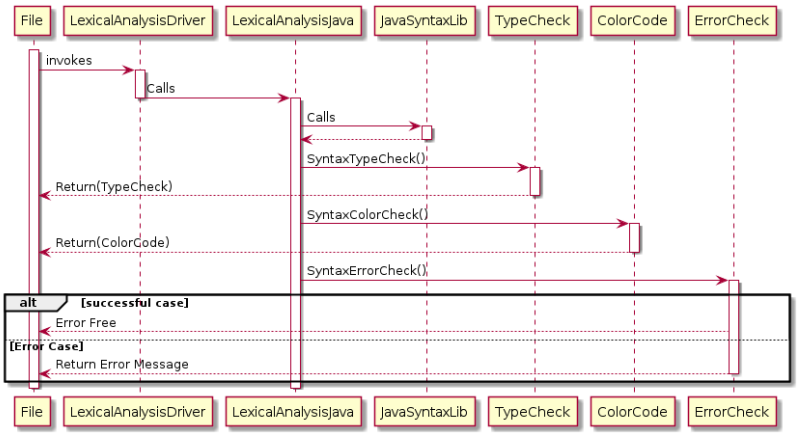


Figure 5.6: Authentication authenticates stuff.

5.7 Communication (boss2849)

Figure 5.7: Authentication authenticates stuff.

5.8 Project File Editor (brec9824)

Figure 5.8: Authentication authenticates stuff.