Squire: A Collaborative Software Development Tool

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

May 13, 2016

Contents

Chapter 1

Overview and Scope

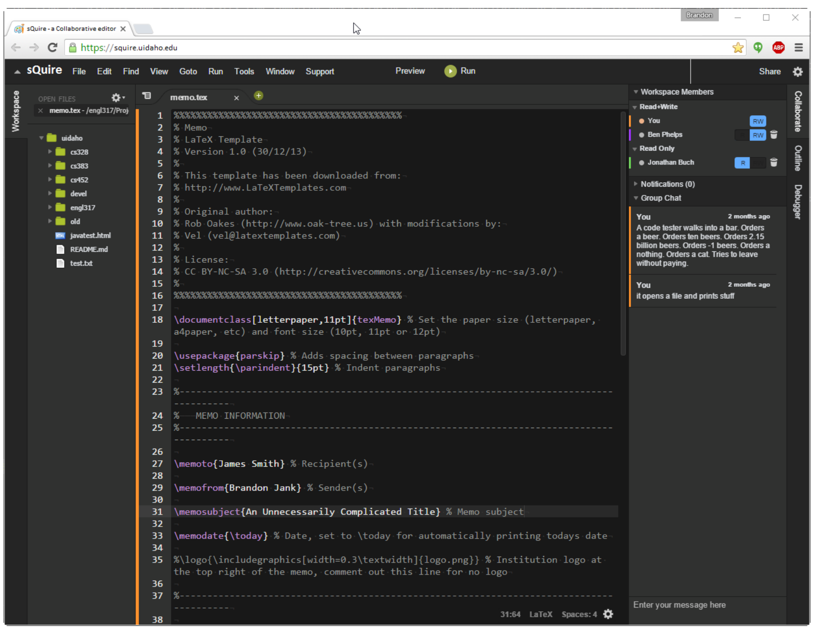


Figure 1.1: "Squire is a web-based collaborative software development environment with a project development center. Squire will allow multiple users to edit files and communicate in real time. Projects can be “stubbed” out by a user and then other users can join and/or vote to support for their favorite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

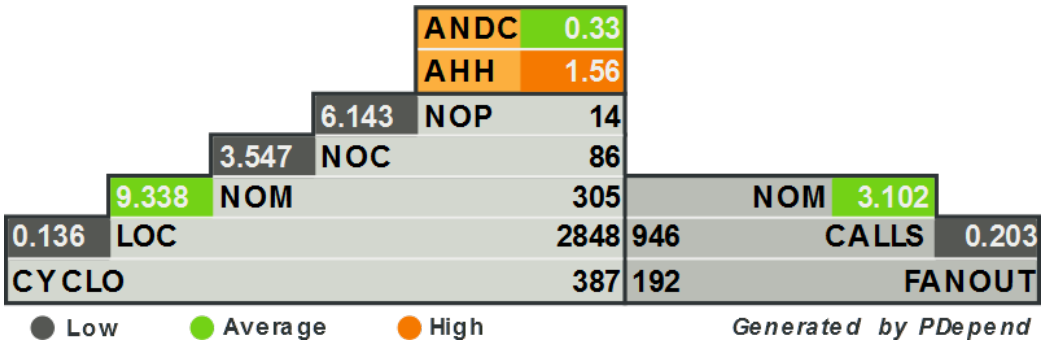
1.9 Security Nightmare (jank6275)

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

Metrics (jank6275)



More information about the pyramid can be found at: `https://pdepend.org/documentation/`

- ANDC: Average Number of Derived Classes – The average of direct subclasses of a class.
- AHH: Average Hierarchy Height – The average of the maximum lenght from a root class to ist deepest subclass subclass.
- NOP: Number of Packages
- NOC: Number Of Classes
- NOM: Number Of Methods
- LOC: Lines Of Code
- CYCLO: Cyclomatic Complexity Number – Count of the available decision paths in a software fragment to determine its complexity.
- CALLS: Number of Method or Function Calls
- FANOUT: Number of Fanouts – Referenced Classes

Number of directories:	68 directories
Number of files:	333 files
Number of classes:	86 classes
Number of methods:	305 methods
Largest class:	Project class (451 loc)
Source lines of code:	2,848 lines
Static disk requirements for binary distribution:	5.04 MB (134 MB installed with prereqs)
Runtime memory requirements:	4.24 MB per concurrent request
Github issues closed:	108 issues
Github pull requests:	44 merged pulls
Github wiki pages:	10 documents
Github squashed commits:	451 times
PHPdocs:	https://squireproject.com/docs/index.html

Chapter 3

Distribution (jank6275)

How can some use the project?

You can deploy the project to your own server or you can use our website. The sQuire webpage can be found at `https://squireproject.com`.

How can someone download and run the project?

The sQuire Project server files can be downloaded from our github repository at `https://github.com/uidaho/squireproject`. A direct link to a zip of the repo can be found at `https://github.com/uidaho/squireproject/archive/master.zip`.

This PHP, HTML, and java based project will run on any standard webserver with the following packages:

- PHP 5.6+
- MySQL, SQLite, MariaDB, or Postgres Database
- Composer 1.0+

A guide for setting up an Ubuntu 14.04+ server can be found in the Squire Requirements section of the projects INSTALL.md.

Chapter 4

Usability (mars2681)

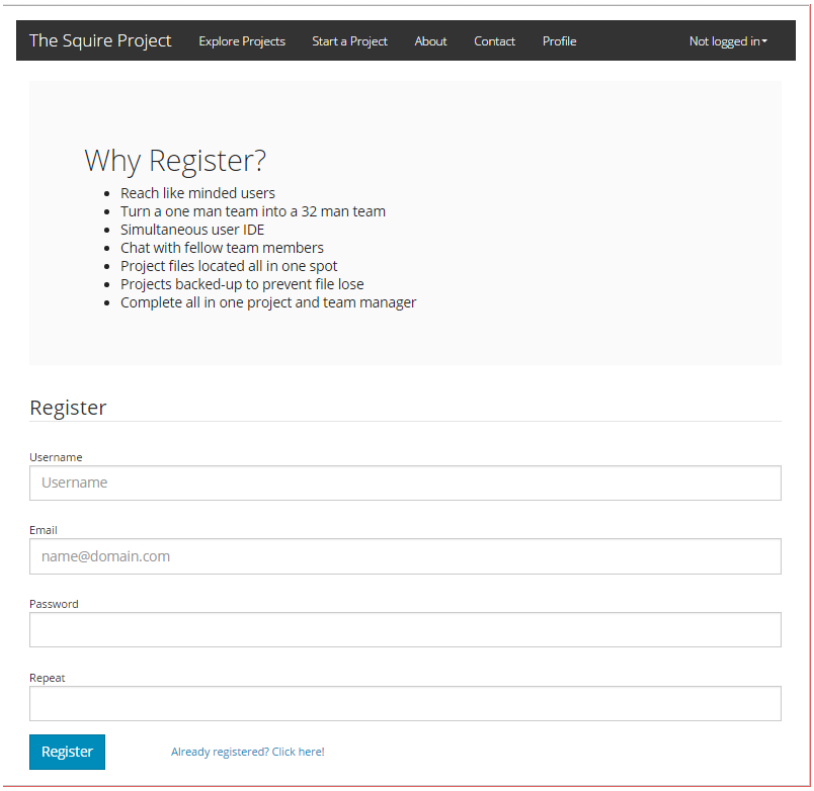
4.1 Getting Started

4.1.1 Introduction

Hello user! You are here because you want to use the exciting new tool Squire! Squire is a developmental tool, allowing you to create projects online, and share them with other users and gain team members to help you complete the project. To get started, you first need to navigate to the Squire website at: https://squireproject.com/

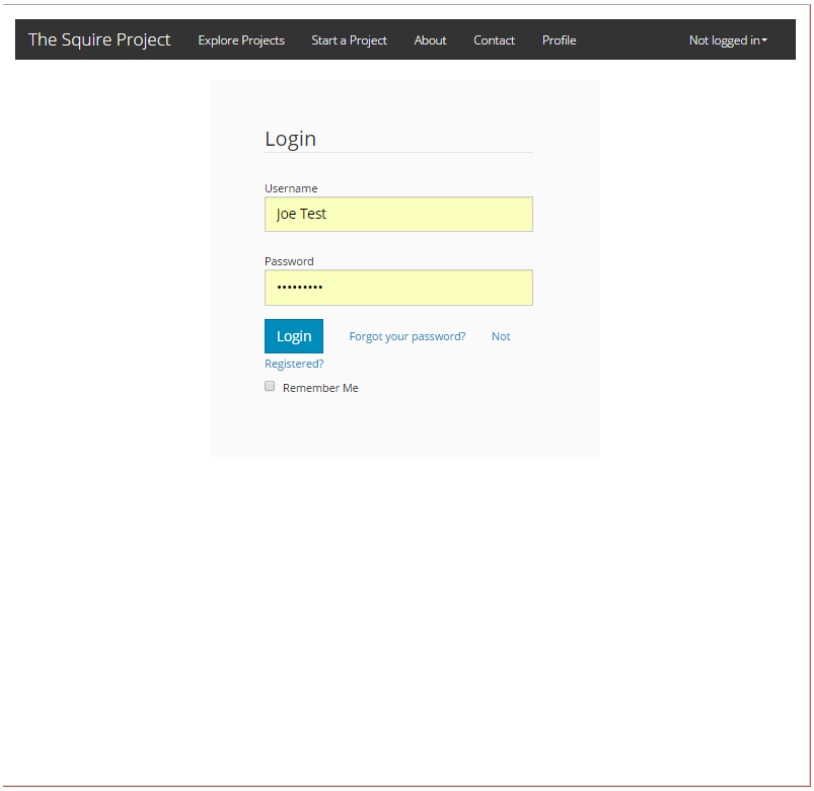
4.1.2 Create an account

Click on the Register/Login button on the front page to reach the registration page.



The register page allows you to create an account. To create an account, enter in a nickname that is at least 6 characters long. This is the name that other users will know you by. Then enter your valid email address. Then enter and reenter a safe password that is at least 6 characters long. Click Register and a confirmation email will be sent to your email account. Once confirmed, you are the brand new owner of a Squire account!

4.1.3 Log In



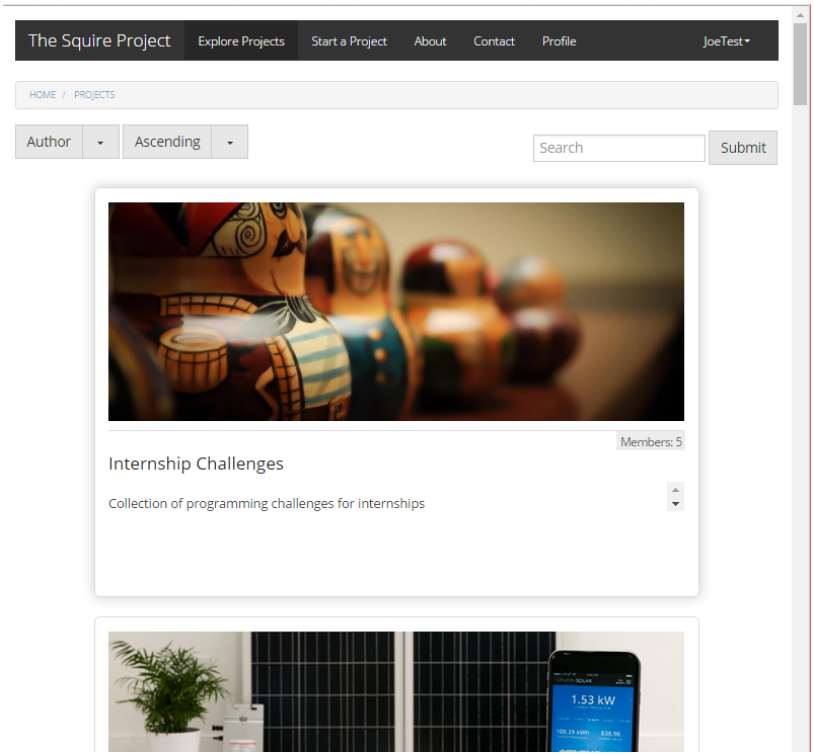
On the log in page, enter in your username and password, then click Login to access your account. Your browser may automatically remember your credentials, as seen in the picture above. If you forgot your password, click on the link Forgot your password?, enter your email address on the following page, and our system will send you a new temporary password.

4.2 Browsing projects

4.2.1 Front Page

Once you have registered or have logged in, you will be brought to the main page! This is where you can browse projects, create your own project, or edit your settings

4.2.2 Browse projects



All projects can be seen on the main page. These can be seen by scrolling down on the main page. To see more projects, you can select another page at the bottom of the screen. To learn more about a project, click on it to be brought to the project page.

4.2.3 Browse Projects By Type

To search for a certain kind of project, you can sort all projects by author, title, creation date and last modified date. To sort, click on the sort button in the top left hand side of the page, and select a sorting type from the top down menu.

4.2.4 Search For Project By Name

You can search for a project by it's specific name using the Search bar in the top right hand side of the page.

4.2.5 Project Page

Through project pages, you can learn more about a project, as well as follow, contribute to, and comment on a project. Each project has a picture, a short and a long description to give the user a better understanding of the project.

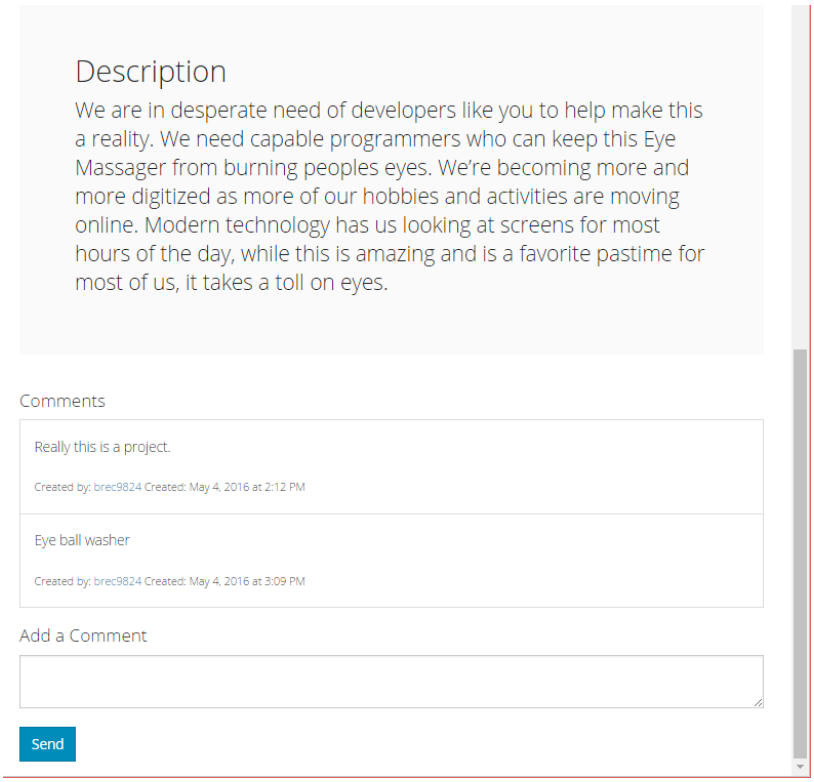
4.2.6 Follow A Project

To follow a project, click on the Follow button below the page picture. This will give you updates on the progress of the project, such as new changes or updates to the description.

4.2.7 Join A Project

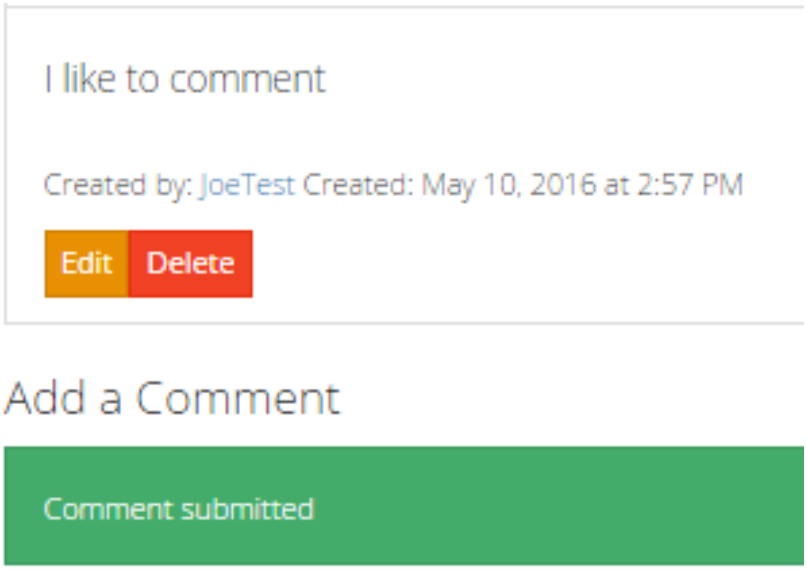
If you would like to contribute code to a project, click on the Join Project button. This will send a join request to the project administrator, who will then decide if you can join or not. Once you are accepted, an access button will be added to this page for you to access the project member's page.

4.2.8 Create A Comment



At the bottom of the Project page are comments! The Squire community comments are what makes this website such a unique and helpful development tool. To create a comment, type your comment in the Add a comment input box and click Send. Your comment, along with its timestamp and your username, will then be seen by anyone on this project page!

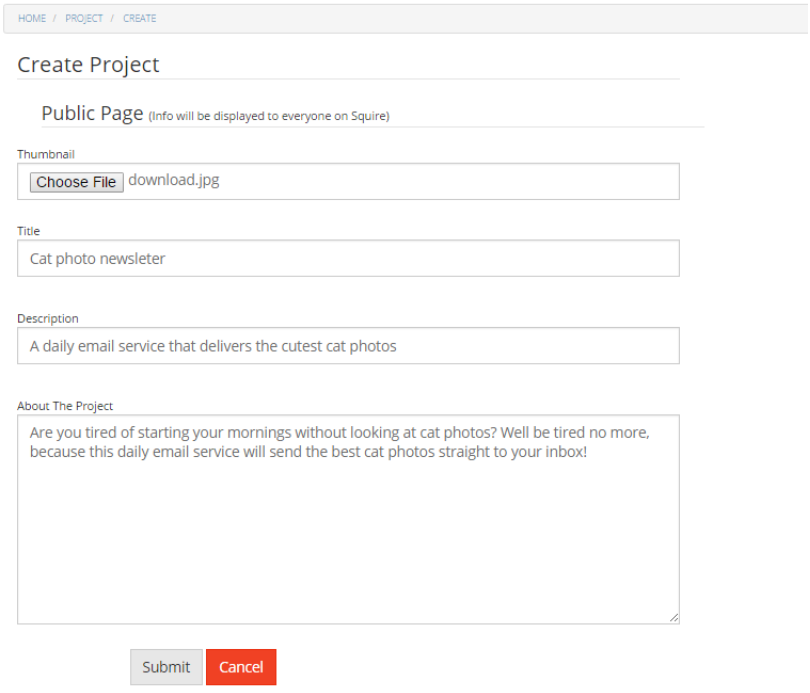
4.2.9 Edit Comments



If you are unhappy with your comment, you can click Delete to remove the comment. Or you can click Edit to change your comment.

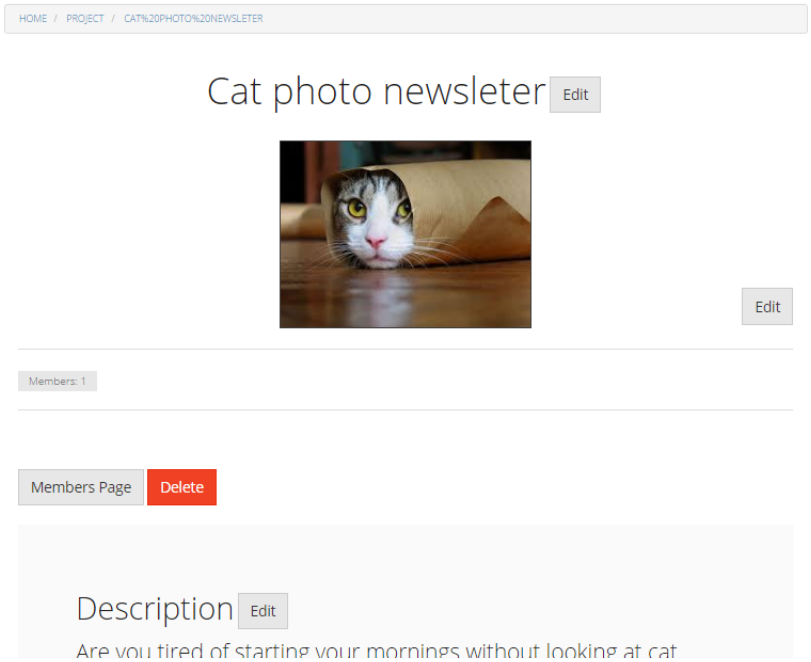
4.3 Creating A Project

4.3.1 Create Project



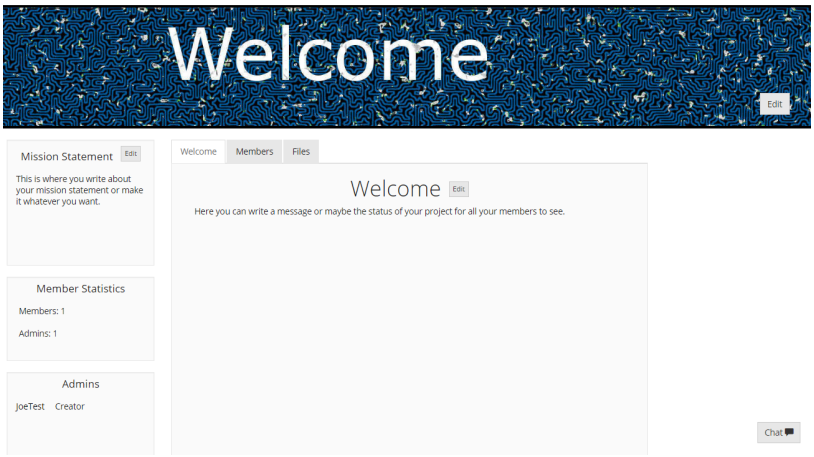
To create a project, click on Start a project on the top menu bar to get to the create project page. Every project requires an image, a title, a description of at least 10 characters, and a more detailed description of the project with at least 100 characters. After inputting all these into the Create Project page, and clicking Submit, you will be redirected to your Project Page.

4.3.2 Editing Project Profile



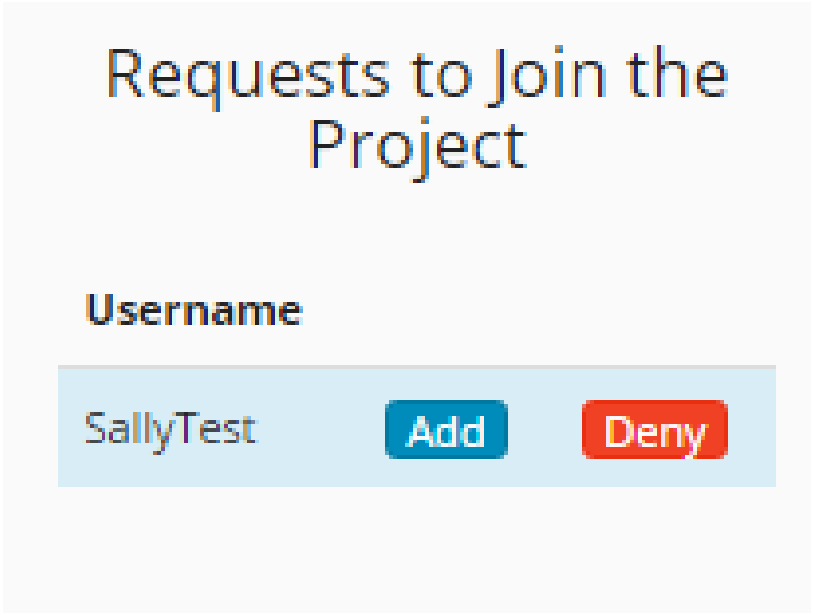
This page is similar to the Project Page, except you can edit the data on this page. To do this, click on the Edit button next to each page element. This page can be re accessed from your profile page by clicking on your Owned Projects. On this page is a button to the Project's Member Page. Click on it to be redirected to it.

4.3.3 Project's Member Page



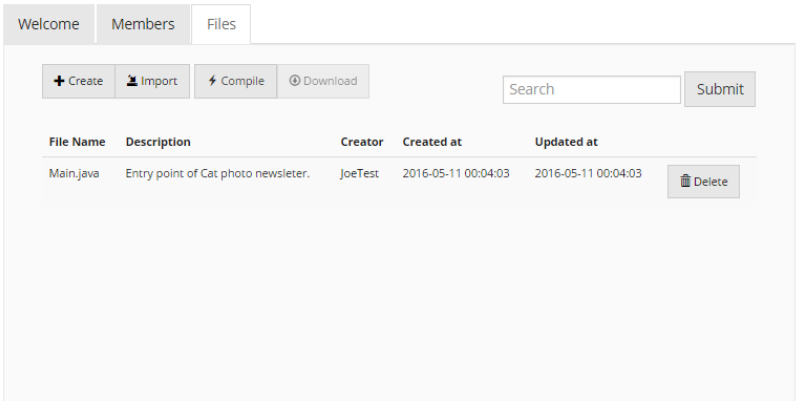
Only you, and members of your project can see this page. From here, you can accept (or deny) join requests and create the files for your project.

4.3.4 Join Requests



Once you get a Join Request, you will see it under Request to Join the Project. You can click Add to add that User to be a part of your project, or you can click Deny to reject their invite.

4.3.5 File Management



To start creating some files, click on the Files tab. Here, it will show you a list of all your files. Each project has a Main.java file by default. By clicking on this file, you will open it up in Squires editor page. To add new files, click either Create to make a new blank file, or Import to import a file from your computer.

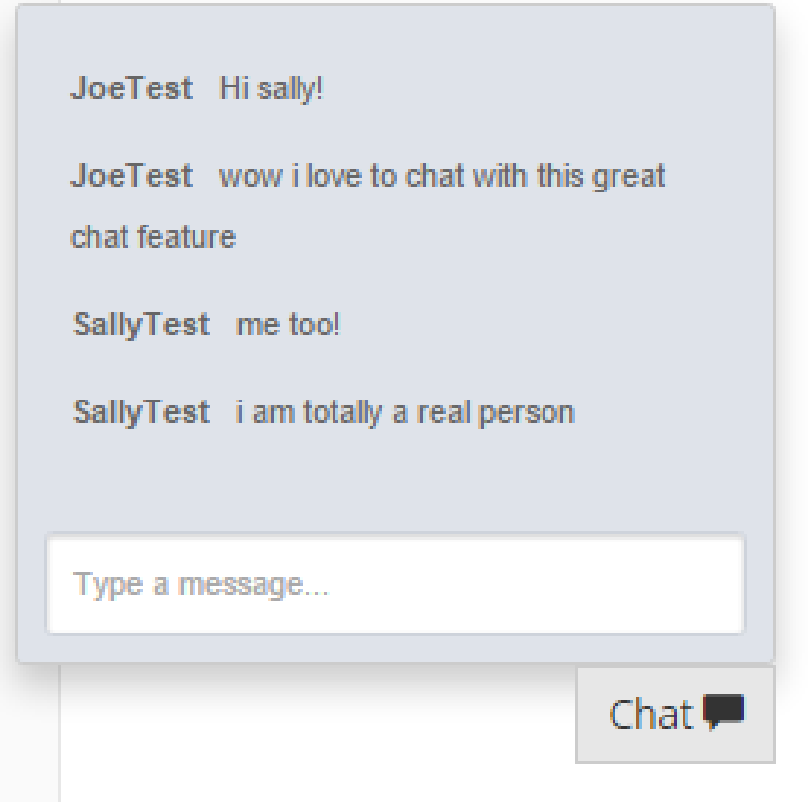
4.3.6 Editor

The editor page is like most other code editor programs, except it allows two users to work on it at the same time. You can see all the current user's editing on the left, with the color being the color of the font they type. Once you are done typing your code, it will automatically be saved.

4.3.7 Compiler

You can compile your code by pressing the Compile button. This will export a .jar file that will execute your program.

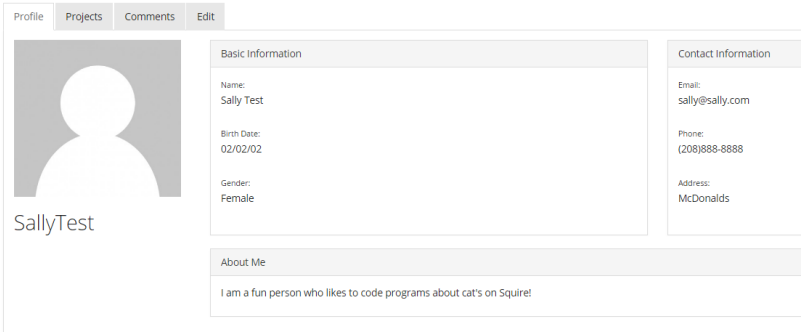
4.3.8 Chat



Each project has their own chat system built in so members can talk to each other. This feature can be found in the lower right hard corner of the projects member page, or in the editor. Once it opens, you can see all the chat messages from this project. You can type in your message in the Type a message input field.

4.4 User Profile

4.4.1 Profile Tab



Each user has their own profile page. This profile page contains personal information, like an avatar, birthdate, gender, and your bio. By default, this information will be blank, so

you will want to edit them in the Edit tab.

4.4.2 Edit Tab

Edit your information here, and hit submit to change your Profile information. Note that other user's can see this information, so use caution with what information you share.

4.4.3 Projects Tab

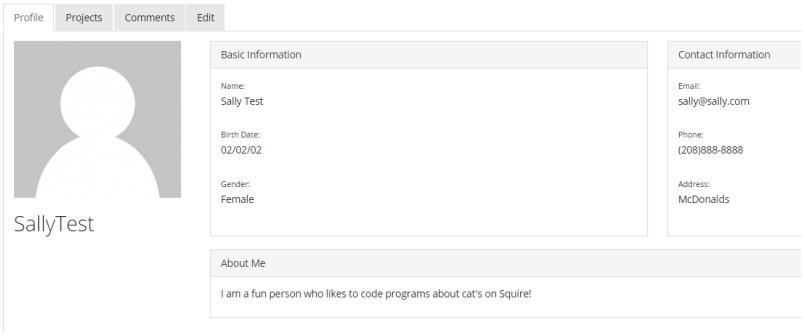
This tab contains all the projects you own. By clicking on one, you will be redirected to it's Edit Project Profile page. From there, you can reach the Project Member's Page.

4.4.4 Comments Tab

This tab contains all the comments you submitted to project pages.

4.5 User Settings

4.5.1 Settings



Each user has their own settings. You can reach your settings page by clicking on your username in the top right corner, and clicking Settings in the dropdown menu. In this page, you have several options that change the way you edit your project. You can disable the chat feature, change the color of your font, and the type of font. Once you are done editing your settings, click submit and your settings will be saved!

Chapter 5

Requirements Documentation

5.1 Functional Requirements

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

5.1.1 Authentication (mora5651)

- Login to squire. (Usecase 4.1.2 and Class 5.2)
- Register for squire. (Usecase 4.1.1 and Class 5.2)
- Forgot username. (Usecase 4.1.4 and Class 5.2)
- Forgot password. (Usecase 4.1.5 and Class 5.2)
- Logout of squire. (Usecase 4.1.3 and Class 5.2)

5.1.2 Project Management (bolt1003)

- Create Project (Usecase 4.5.1 and Class 5.3)
- Open a project (Usecase 4.5.2 and Class 5.3)
- Join Project (Usecase 4.5.3 and Class 5.3)
- Leave project (Usecase 4.5.4 and Class 5.3)
- Delete Project (Usecase 4.5.5 and Class 5.3)
- Export Project (Usecase 4.5.6 and Class 5.3)
- Accept Invite to Project (Usecase 4.5.7 and Class 5.3)
- Remove User from Project (Usecase 4.5.8 and Class 5.3)
- Edit Project Permissions (Usecase 4.5.9 and Class 5.3)

- Invite User to Project (Usecase 4.5.10 and Class 5.3)
- Promote User to Admin (Usecase 4.5.11 and Class 5.3)
- Demote Admin (Usecase 4.5.12 and Class 5.3)
- Block User (Usecase 4.5.13 and Class 5.3)

5.1.3 Project Ideas (mars2681)

- Page to browse potential projects (Usecase 4.2 and Class 5.4)
- Up- and down-votes for project selection (Usecase 4.2 and Class 5.4)
- Different ways to sort projects (date, projected team size, votes) (Usecase 4.2 and Class 5.4)
- Ability to post a project (Usecase 4.2 and Class 5.4)
- Ability to edit a project (Usecase 4.2 and Class 5.4)
- Ability to delete a project (but only by project author) (Usecase 4.2 and Class 5.4)
- Ability to follow a project (Usecase 4.2 and Class 5.4)

5.1.4 Settings - Preferences and Profile (brec9824)

- Viewable profile by other users and by oneself (Usecase 4.6.1 and Class 5.5)
- Includes editable email, profile image, password, and personal bio. (Usecase 4.6.2 and Class 5.5)
- Setting for public, friends only, or private viewing of online status, email address, personal bio, project ownerships, project memberships, and friends list. (Usecase 4.6.3 and Class 5.5)
- Option for settings users prefered color and shape to be displayed in projects if available. (Usecase 4.6.2 and Class 5.5)
- Option to have account deleted. (Usecase 4.6.4 and Class 5.5)
- Direct access to projects that are listed under project ownerships and project memberships. (Usecase 4.6.5 and Class 5.5)

5.1.5 Compiler (boss2849)

- Compile project sub-modules or entire project. (Usecase 4.7.1 and Class 5.6)
- Compile and run code within IDE. (Usecase 4.7.2 and Class 5.6)
- Compile code and package to a JAR. (Usecase 4.7.3 and Class 5.6)
- Impose temporary code freeze during compilation. (Usecase 4.7.4 and Class 5.6)

5.1.6 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

5.1.7 Communication (jank6275)

- Global chat when anywhere in program. (Usecase 4.3.2 and Class 5.8)
- Project chat when a project is open. (Usecase 4.3.1 and Class 5.8)
- Closeable project chat. (Usecase 4.3.3 and Class 5.8)
- Closeable global chat. (Usecase 4.3.4 and Class 5.8)

5.1.8 Project File Editor (snev7821)

- Add new file to project. (Usecase 4.4.1 and Class 5.9)
- Import existing file in to project. (Usecase 4.4.2 and Class 5.9)
- Delete a file. (Usecase 4.4.3 and Class 5.9)
- Export a file. (Usecase 4.4.4 and Class 5.9)
- Open file in new tab. (Usecase 4.4.5 and Class 5.9)
- View line numbers. (Usecase 4.4.6 and Class 5.9)
- View References. (Usecase 4.4.7 and Class 5.9)
- View Dates. (Usecase 4.4.8 and Class 5.9)
- View users and user history. (Usecase 4.4.9 and Class 5.9)
- Format Document. (Usecase 4.4.10 and Class 5.9)
- Find and replace. (Usecase 4.4.11 and Class 5.9)
- Display the currently typing user. (Usecase 4.4.12 and Class 5.9)

5.1.9 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.

5.2 Non-Functional Requirements

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

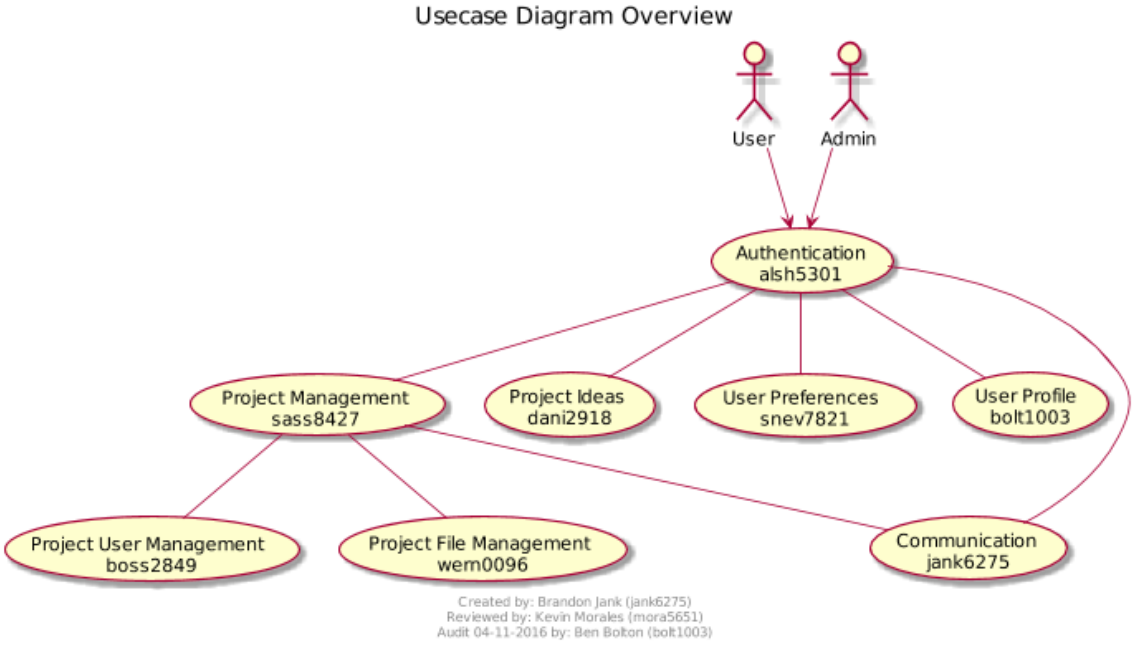
- The current location of any user's cursor can be quickly jumped to by any user with the project open. (Usecase 4.5.8 and Class 5.9)
- Text written by the user in the editor should be visible instantaneously. (Usecase 4.5 and Class 5.9)
- Text written by other users in the editor should be visible within 2 seconds. (Usecase 4.5 and Class 5.9)
- Text will be underlined in the user's color, if they edited/created that text. (Usecase 4.5.9 and Class 5.9)
- The line number will be highlighted in the users color, if they created the line. (Usecase 4.5.9 and Class 5.9)
- Each line edited by a user should be saved to create a edit history for each user in a document. (Usecase 4.5.4 and Class 5.9)
- 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. (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)

Chapter 6

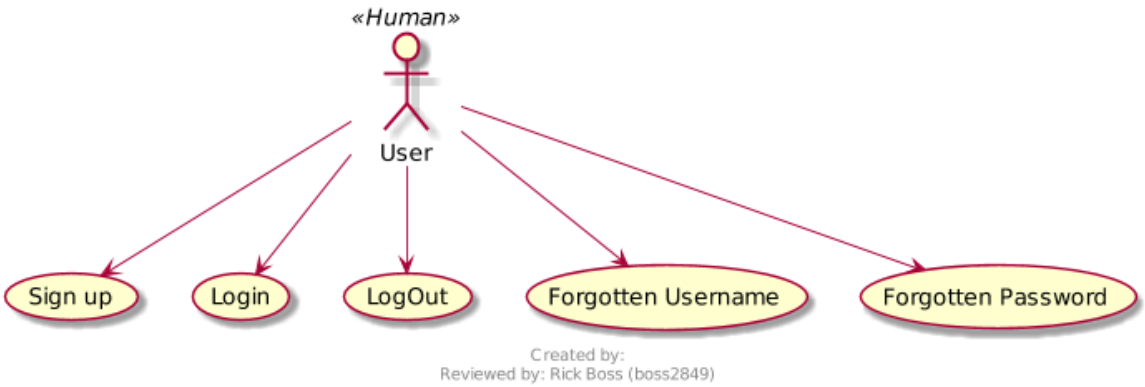
Use Case Diagrams

6.1 Overview (jank6275)

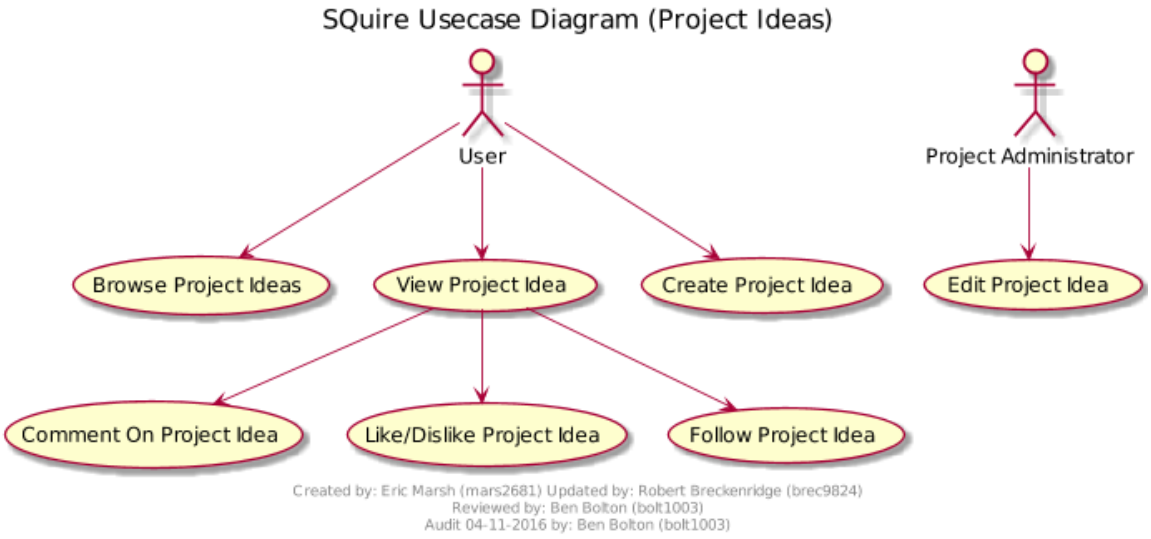


A usecase diagram that relates major sections of Squire.

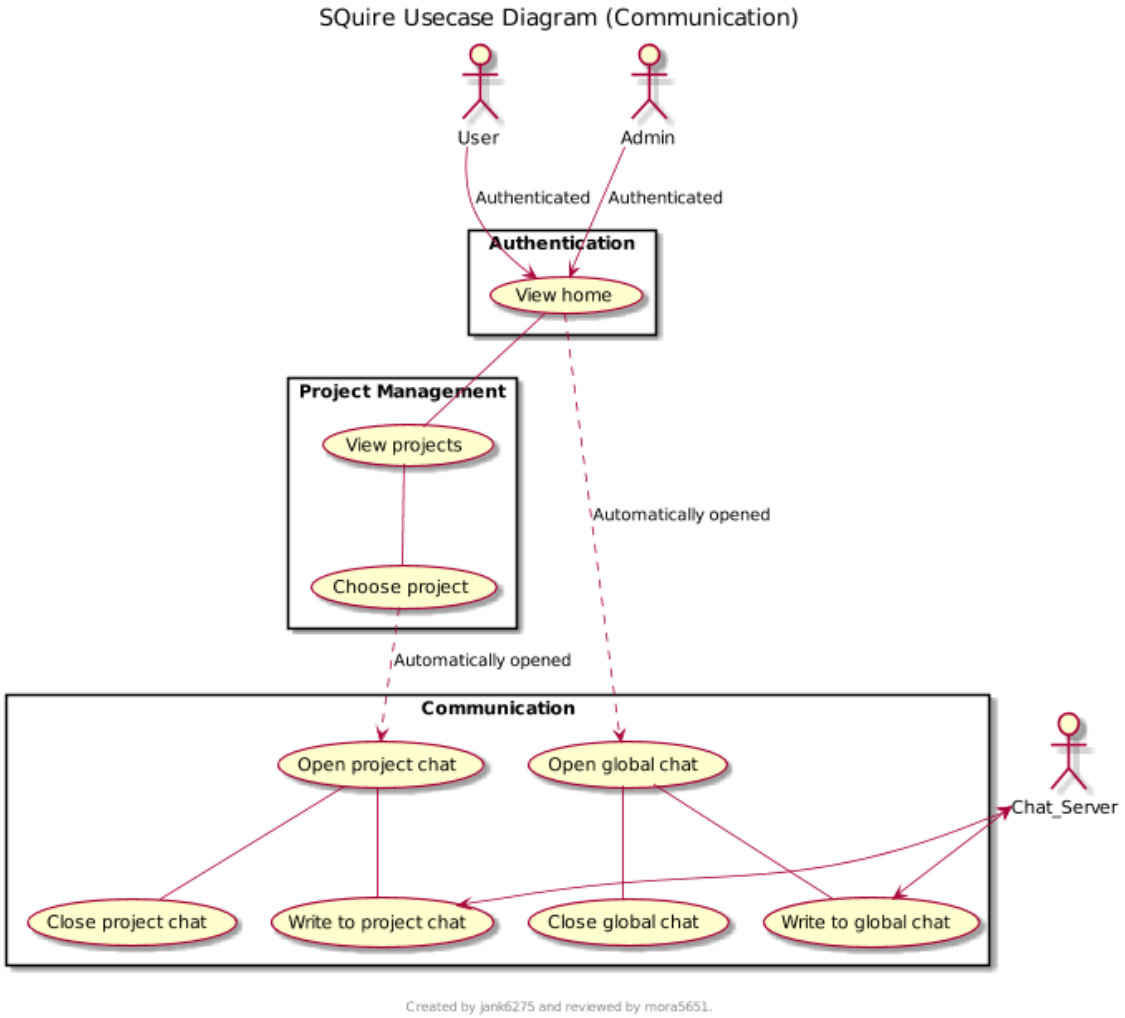
6.2 Authentication (mora5651)



6.3 Project Ideas (mars2681)

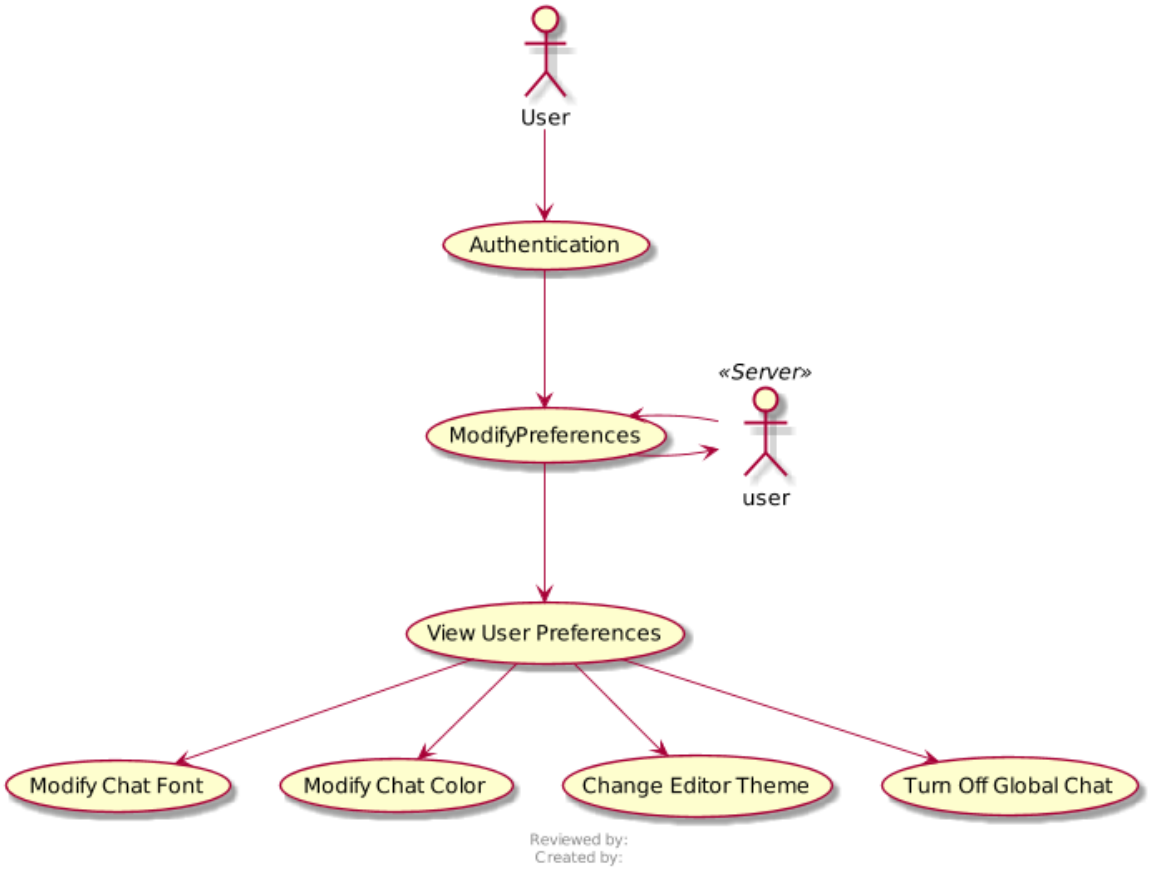


6.4 Communication (jank6275)



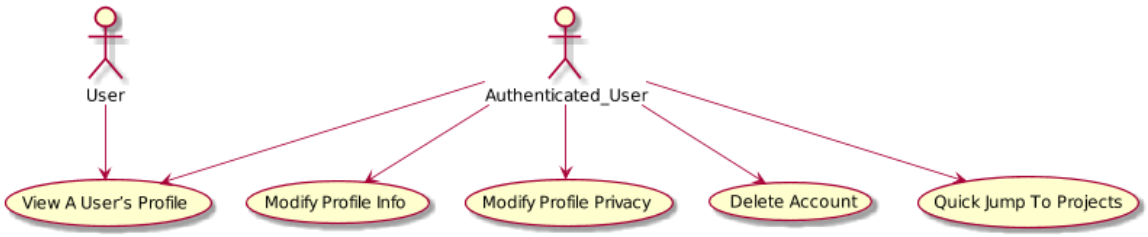
A usecase diagram for sQuireś communication features. Used in Class Diagrams 4.2.1 and 4.2.2.

6.5 User Preferences (snev7821)

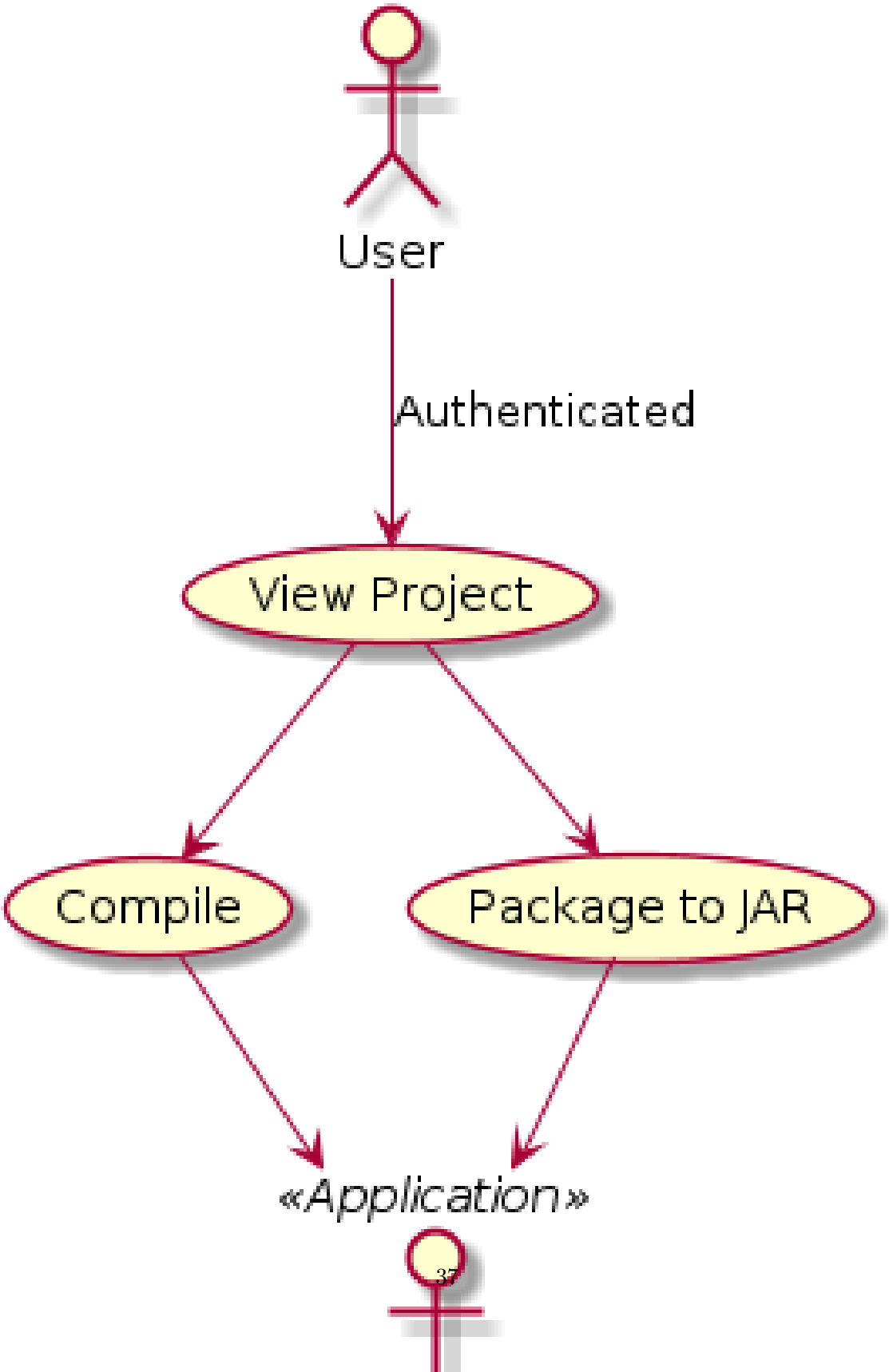


6.6 Project Management (bolt1003)

6.7 Settings, Preferences, and Profile (brec9824)



6.8 Compiler (boss2849)



Chapter 7

Use Case Descriptions

7.1 Authentication (mora5651)

7.1.1 Sign up (Use Case Diagram 3.2)

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:	Sign Up.
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.

7.1.2 Sign in (Use Case Diagram 3.2)

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.
Related use cases:	Comment on Project Idea
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

7.1.3 Logout (Use Case Diagram 3.2)

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.

7.1.4 Forgotten Username (Use Case Diagram 3.2)

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.

7.1.5 Forgotten Password (Use Case Diagram 3.2)

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.

7.2 Project Ideas (mars2681)

7.2.1 Browse Project Ideas

Actors:	User
Goals:	Browse new projects, popular projects, and user's projects
Pre-conditions:	User is signed in
Summary:	User looks through posted project ideas to find projects to work on/discuss
Related use cases:	View Project Idea
Steps:	1. Actor selects Browse Project Ideas button, they are then brough to Project Ideas page 2. System displays the project pages that the user is following, also the most popular projects, and new projects
Alternatives:	None.
Post-conditions:	None.

7.2.2 View Project Idea

Actors:	User
Goals:	View project idea page
Pre-conditions:	User is signed in and in the Browse Project Ideas page
Summary:	User views a project ideas page to look at its information and follow/comment/like or dislike.
Related use cases:	Browse Project Ideas
Steps:	1. Actor selects a project from the Browse Project Ideas Page 2. System brings actor to the project's info page 3. Actor can then look at the projects information, as well as continue on to follow or comment or vote on the project idea
Alternatives:	None.
Post-conditions:	None.

7.2.3 Create Project Idea

Actors:	Project Administrator
Goals:	Generate public interest in project idea
Pre-conditions:	Prospective project administrator is signed in
Summary:	Admin can make a page to show off their project idea to other users
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 4. System adds the project page to the database for other users to see
Alternatives:	None.
Post-conditions:	None.

7.2.4 Edit Project Idea

Actors:	Project Administrator
Goals:	User edits the information on their project idea page
Pre-conditions:	Prospective project administrator is signed in and is now in the project idea page
Summary:	User edits their pre-existing project idea page
Related use cases:	Create Project Idea Thread
Steps:	1. Actor clicks Edit Project Idea Page 2. Actor can then edit the title, description, and other information on the project idea page 3. Actor clicks Submit 4. System saves changes, updates page
Alternatives:	Actor selects Delete Changes and returns to Browse Project Ideas page.
Post-conditions:	None.

7.2.5 Comment on Project Idea

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 5. System adds comment to Project Idea page's comments
Alternatives:	None
Post-conditions:	None.

7.2.6 Like/Dislike Project Idea

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.

7.2.7 Follow Project Idea

Actors:	User, Project Administrator
Goals:	Follow project updates
Pre-conditions:	Actor is signed in, has navigated to a project idea
Summary:	User follows project to receive updates and information about it
Related use cases:	View Project Idea
Steps:	1. Actor selects Follow button 2. System notifies user that they are now following the project 3. System automatically updates followed project information on Browse Project Ideas page
Alternatives:	None
Post-conditions:	None.

7.3 Communication (jank6275)

7.3.1 Open project chat (Class Diagram 5.8)

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.

7.3.2 Open global chat (Class Diagram 5.8)

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.

7.3.3 Close project chat (Class Diagram 5.8)

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.

7.3.4 Close global chat (Class Diagram 5.8)

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.

7.3.5 Write to project chat (Class Diagram 5.8)

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.

7.3.6 Write to global chat (Class Diagram 5.8)

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.

7.3.7 Modify chat font (Settings Class Diagram)

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

7.3.8 Modify chat color (Settings Class Diagram)

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

7.4 Project File Editor (snev7821)

7.4.1 Add New File to Project

Actors: User of sQuire

Summary: The user performs this task to add a new file to the project.

Pre-conditions:

1. User must be registered.
2. User must be logged in.
3. User must have a project open.

Steps:

1. User clicks File in the top menu bar.
2. System opens a drop-down menu.
3. User navigates to Add New File.
4. System opens an Add New File dialog window.
5. User selects the file type and names the file.
6. User clicks Add.
7. System adds the file to the project.

Alternatives:

1. Step 1: The user right clicks in the project panel and the system continues on to step 2 above.
2. Step 5: The user clicks Cancel and a new file is not added to the project.

Post-conditions:

1. A new file is added to the project.
2. The database is updated to reflect the changes.

Related: Add Existing File to Project

7.4.2 Add Existing File to Project

Actors: User of sQuire

Summary: The user performs this task to add an existing file to the project.

Pre-conditions:

1. User must be registered.
2. User must be logged in.
3. User must have a project open.

Steps:

1. User clicks File in the top menu bar.
2. System opens a drop-down menu.
3. User navigates to Add Existing File.
4. System opens an Add Existing File dialog window.
5. User selects PC or SQuire or Github.
6. System updates the dialog to reflect the selected source.
7. User navigates to the file's location and selects it.
8. User clicks Add.
9. System adds the file to the project.

Alternatives:

1. Step 1: The user right clicks in the project panel and the system continues on to step 2 above.
2. Step 5-7: The user clicks Cancel and a new file is not added to the project.

Post-conditions:

1. An existing file is added to the project.
2. The database is updated to reflect the changes.

Related: Add New File to Project

7.4.3 Delete File

Actors: User of sQuire

Summary: The user performs this task to delete a file from the project.

Pre-conditions:

1. User must be registered.
2. User must be logged in.
3. User must have a project open.
4. User must be administrator of project.
5. Current project must have at least one file.

Steps:

1. User right clicks a file in the project pane.
2. System opens a drop-down menu.
3. User navigates to *Delete*.
4. System opens an *Delete File* dialog window, asking if the user is sure.
5. User selects *Yes*.
6. System deletes the file from the project.

Alternatives:

1. Step 5: The user clicks *Cancel* instead and the file is not deleted from the project.
2. The user selects multiple files before step 1.

Post-conditions:

1. The file is deleted from the project.
2. The database is updated to reflect the changes.

Related: Delete Project

7.4.4 Export Files

Actors: User of sQuire

Summary: The user performs this task to download a number of files from a project.

Pre-conditions:

1. User must be registered.
2. User must be logged in.
3. User must have a project open.
4. Must have at least one file in the project.
5. User must have download permissions.

Steps:

1. User clicks File in the top menu bar.
2. System opens a drop-down menu.
3. User navigates to Export Files.
4. System opens an Export dialog window showing the project files on the left panel and the export location in the right panel.
5. User selects a number of files on the left pane.
6. User navigates to the export location in the right pane.
7. User clicks Export.
8. System downloads the selected files to the specified location.

Alternatives:

1. Step 1: The user right clicks in the project panel and the system continues on to step 2 above.
2. Step 5: User selects a folder and all files under that folder are selected.
3. Step 5-6: The user clicks Cancel and the project is not exported.

Related: Export Project

7.4.5 Open File in New Tab

Actors: User of sQuire

Summary: Allows users to open a file.

Goals: Opening files is essential in being able to work on a project.

Pre-conditions:

1. User is registered.
2. User is logged in.
3. User has a project open.
4. Current project contains at least one file.
5. User has read permission.

Steps:

1. User double clicks a file.
2. The editor opens its contents in a new tab and focuses on it.

Alternatives: Step 1: Instead of double clicking a file, the user right clicks it and navigates to Open.

7.4.6 View Line Numbers

Actors: User of sQuire

Summary: Allows the user to hide line numbers to the left of the document.

Goals: In case user wants to hide line numbers so they have more space for text.

Pre-conditions:

1. Must be registered.
2. Must be logged in.
3. User has view permission.
4. A file is open.
5. Line numbers are on

Steps:

1. User selects the View menu option.
2. System displays a drop-down with various options.
3. User selects the Hide Line Numbers option.
4. System hides line numbers to the left of the document.

Related:

1. View References
2. View Dates
3. View Authors

7.4.7 View References

Actors: User of sQuire

Summary: Allows the user to view the number of references to a given function.

Goals: It is useful to know the number of references to a given function for optimization and debugging purposes.

Pre-conditions:

1. Must be registered.
2. Must be logged in.
3. User has view permission.
4. A *code* file is open.

Steps:

1. User selects the View menu option.
2. System displays a drop-down with various options.
3. User selects the View References option.
4. System displays the number of references above each method declaration.

Related:

1. Hide Line Numbers
2. View Dates
3. View Authors

7.4.8 View Dates

Actors: User of sQuire

Summary: Allows the user to view the last date that each block of a document was edited. Blocks are defined as any number of lines that was written by a single user. Minimum block size is one line.

Goals: This provides a useful metric for how up-to-date parts of the document are.

Pre-conditions:

1. Must be registered.
2. Must be logged in.
3. User has view permission.
4. A file is open.

Steps:

1. User selects the View menu option.
2. System displays a drop-down with various options.
3. User selects the View Dates option.
4. System displays the last date that each block of a document was edited.

Related:

1. Hide Line Numbers
2. View References
3. View Authors

7.4.9 View Authors

Actors: User of sQuire

Summary: Allows the user to view the last author that edited each block of a document. Blocks are defined as any number of lines that was written by a single user. Minimum block size is one line.

Goals: This is an accountability tool allowing other users to identify who is responsible for a change to a document.

Pre-conditions:

1. Must be registered.
2. Must be logged in.
3. User has read permission.
4. A file is open.

Steps:

1. User selects the View menu option.
2. System displays a drop-down with various options.
3. User selects the View Authors option.
4. System displays the name of the last editor of each line of the document.

Related:

1. Hide Line Numbers
2. View References
3. View Dates

7.4.10 Format Document

Actors: User of sQuire

Summary: Allows the user to format the document to a specified format

Goals: An easy tool for making sweeping changes to a large part of a document.

Pre-conditions:

1. Must be registered.
2. Must be logged in.
3. User has read/write permission.
4. A file is open.
5. The document has formatting options set.

Steps:

1. User selects the Edit menu option.
2. System displays a drop-down with various options.
3. User selects the Format Document option.
4. System formats the current document to the formatting settings currently set.

Alternatives:

1. If no formatting settings are currently set, display a dialog box after step 3 and give the option for the user to do so now.

Related:

1. Find/Replace

7.4.11 Find/Replace

Actors: User of sQuire

Summary: Allows the user to find and/or replace phrases.

Goals: This is a powerful tool that allows a user to make safer, quicker, and more efficient changes to a document.

Pre-conditions:

1. Must be registered.
2. Must be logged in.
3. User has read/write permission.
4. A file is open.

Steps:

1. User selects the Edit menu option.
2. System displays a drop-down with various options.
3. User selects the Find/Replace option.
4. System displays a small form in an unobtrusive location.
5. User enter the phrase to find and selects find.
6. System highlights and focuses on the first occurrence of the phrase and all highlights all other occurrences.

Alternatives:

1. User selects option to replace in step 5 and enters a phrase with which to replace the found occurrences of the searched phrase. The system replaces each occurrence.

Related:

1. Format Document
2. Find/Replace

7.4.12 Display Typing User

Actors: User of sQuire

Summary: As the user types, the system displays their name, their typing, and their caret, in a different color, to other users.

Goals: Differentiate who is typing what.

Pre-conditions:

1. Must be registered.
2. Must be logged in.
3. User has read/write permission.
4. A file is open.
5. Other users have the same document open.

Steps:

1. User begins typing.
2. System displays the user's typing, the user's name, and the user's caret, in a different color, to Other Users.
3. Other Users see User typing, his username, and his caret, in a different color.

7.5 Project Management (bolt1003) (Use Case Diagram: 3.6)

7.5.1 Create Project (Use Case Diagram: 3.6)

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.

7.5.2 Open a project (Use Case Diagram: 3.6)

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.

7.5.3 Join Project (Use Case Diagram: 3.6)

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

7.5.4 Leave project (Use Case Diagram: 3.6)

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.

7.5.5 Delete Project (Use Case Diagram: 3.6)

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.

7.5.6 Export Project (Use Case Diagram: 3.6)

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.

7.5.7 Accept Invite to Project (Use Case Diagram: 3.6)

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.

7.5.8 Remove User from Project (Use Case Diagram: 3.6)

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.

7.5.9 Edit Project Permissions (Use Case Diagram: 3.6)

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.

7.5.10 Invite User to Project (Use Case Diagram: 3.6)

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.

7.5.11 Promote User to Admin (Use Case Diagram: 3.6)

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.

7.5.12 Demote Admin (Use Case Diagram: 3.6)

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.

7.5.13 Block User (Use Case Diagram: 3.6)

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.

7.6 Settings - Preferences and Profile (brec9824)

7.6.1 View A User's Profile

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.

7.6.2 Modify Profile Info

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.

7.6.3 Modify Profile Privacy

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.

7.6.4 Delete Account

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.

7.6.5 Quick Jump To Projects

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.

7.7 Compiler (boss2849) (Use Case Diagram 3.8)

7.7.1 Compile (Use Case Diagram 3.8)

Actors: User

Goals: Compile source

Pre-conditions: User is logged in and viewing project or viewing a file.

Summary: User requests that the code be compiled, the server compiles the code and returns success/failure.

Related use cases: Download Compiled Jar

Steps:

1. User selects "Compile" from button menu.
2. The Server receives the request to compile.
3. The Server downloads the file(s) from Firebase to a temporary directory and compiles the file(s).
4. The Server returns the results of compilation to the User, and enables downloading of result.

Alternatives: In Step 4, the compilation fails, don't allow downloading of result, redirect and display exact errors.

Post-conditions: None.

7.7.2 Download Compiled Jar (Use Case Diagram 3.8)

Actors:	User
Goals:	Compile and package source to a jar
Pre-conditions:	User is signed in and viewing a project or a file in the project, and has successfully compiled the project.
Summary:	User selects "Download" from the button menu and downloads the executable packaged jar.
Related use cases:	Compile
Steps:	1. The Server packages the result of the last compilation of the project to an executable jar. 2. The Server builds a download link with the path of the jar, and returns the download to the user.
Alternatives:	None.
Post-conditions:	None.

Chapter 8

Class Diagrams

8.1 Overview (brec9824, jank6275)

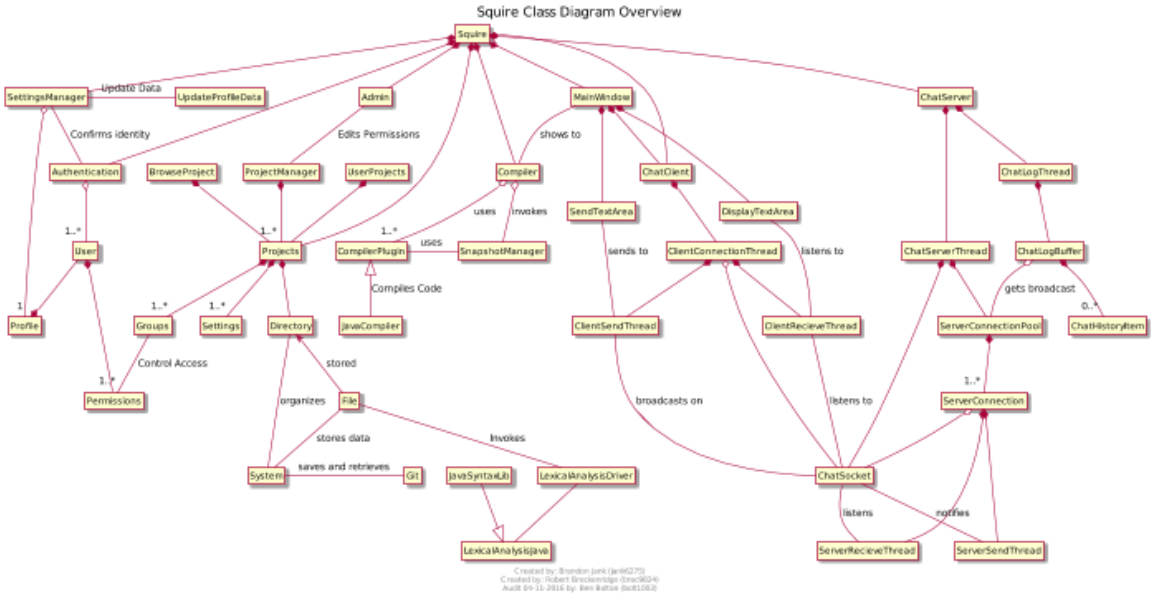


Figure 8.1: Diagram Dictonary: Class overview of Squire displaying the connections between each subsystem and their classes. More specific diagrams follow.

8.2 Authentication (snev7821)

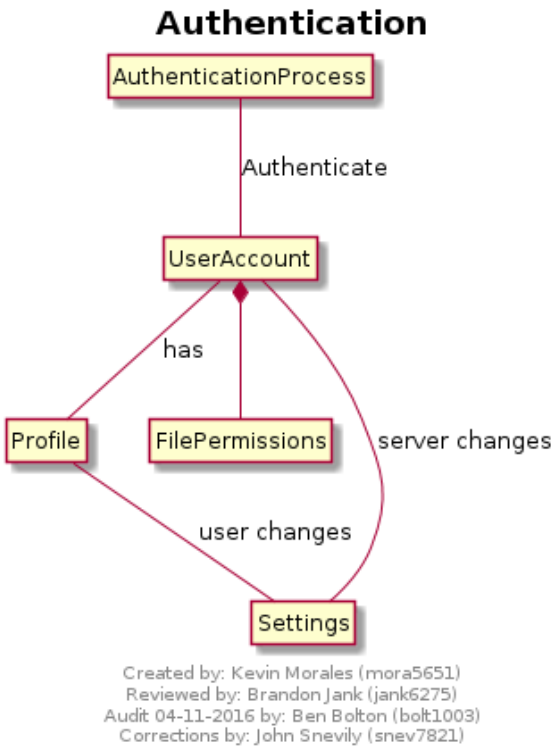


Figure 8.2: Diagram Dictonary: Authentication is a security precaution in sQuire that allows the system to verify a user. Users will provide a username and password, and will be validated on the server-side via email. Different settings are available to the user and the admin. Sequence diagram for Authentication can be found at 6.1.

8.3 Project Management (bolt1003) (Sequence Diagram 6.2)

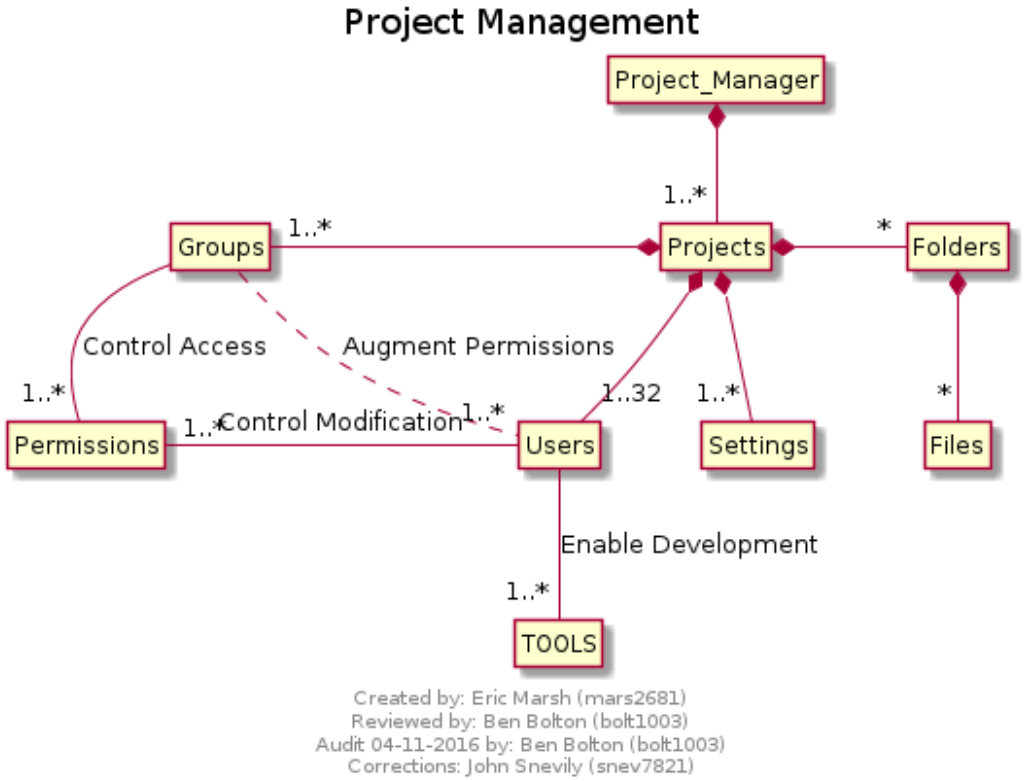


Figure 8.3: Diagram Dictonary: Project management allows for the group to deligate permissions and create projects. Project manager is an overview page, containing user projects. A project is a colleciton of files and folders. Grouping allows permission and user control over a project.

8.4 Project Ideas (snev7821)

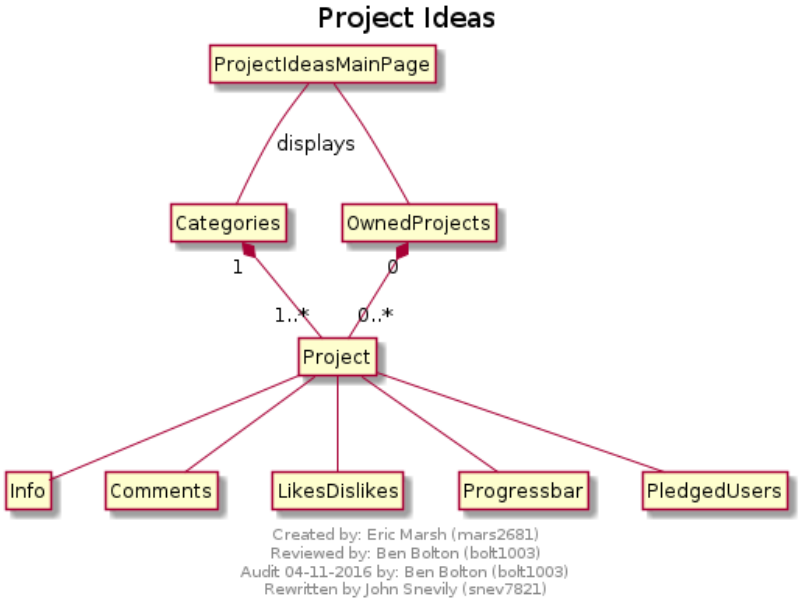


Figure 8.4: Diagram Dictonary: Provides a page to view projects in. User can either view other's projects by category, or view their own projects. Projects pages are a collection of data about that project, such as number of votes, coders pledged, etc.

8.5 Settings, Preferences, and Profile (brec9824)

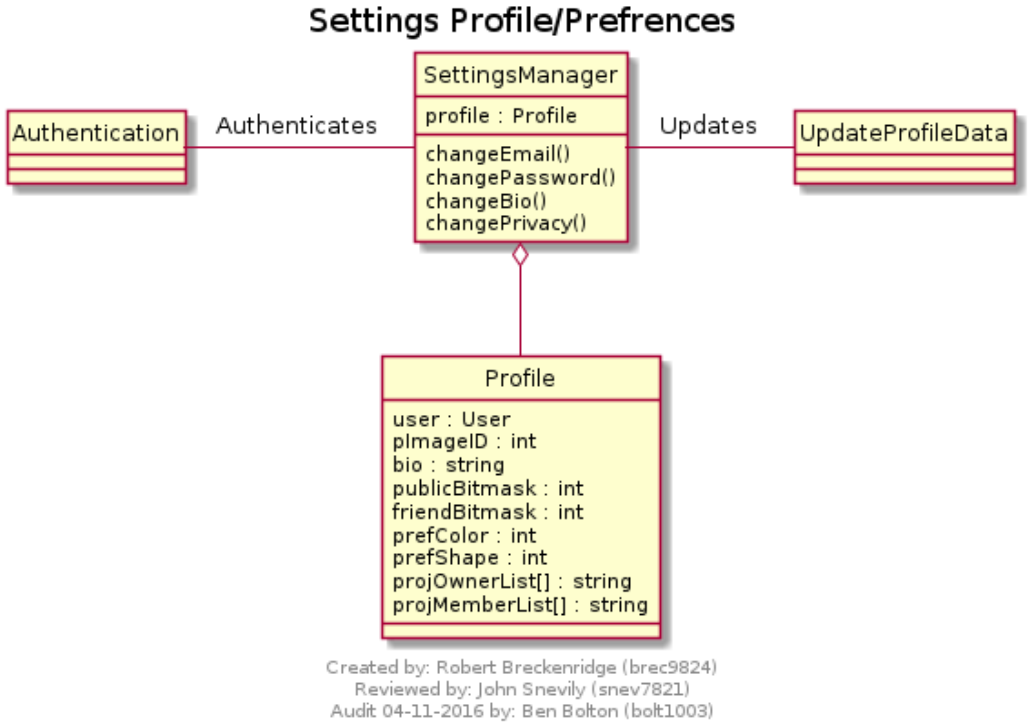


Figure 8.5: Diagram Dictonary: Settings Profile/Prefrences allows for profile viewing and management while maintaining speed with the use of push updates. SettingsManager uses Authentication to verify valid input and to authentic the users data. While SettingsManager uses UpdateUserProfile to push the users data that needs to updated to the server. Sequence Diagram for a settings change can be found at 6.4

8.6 Compiler (boss2849) (Sequence Diagram 6.5)

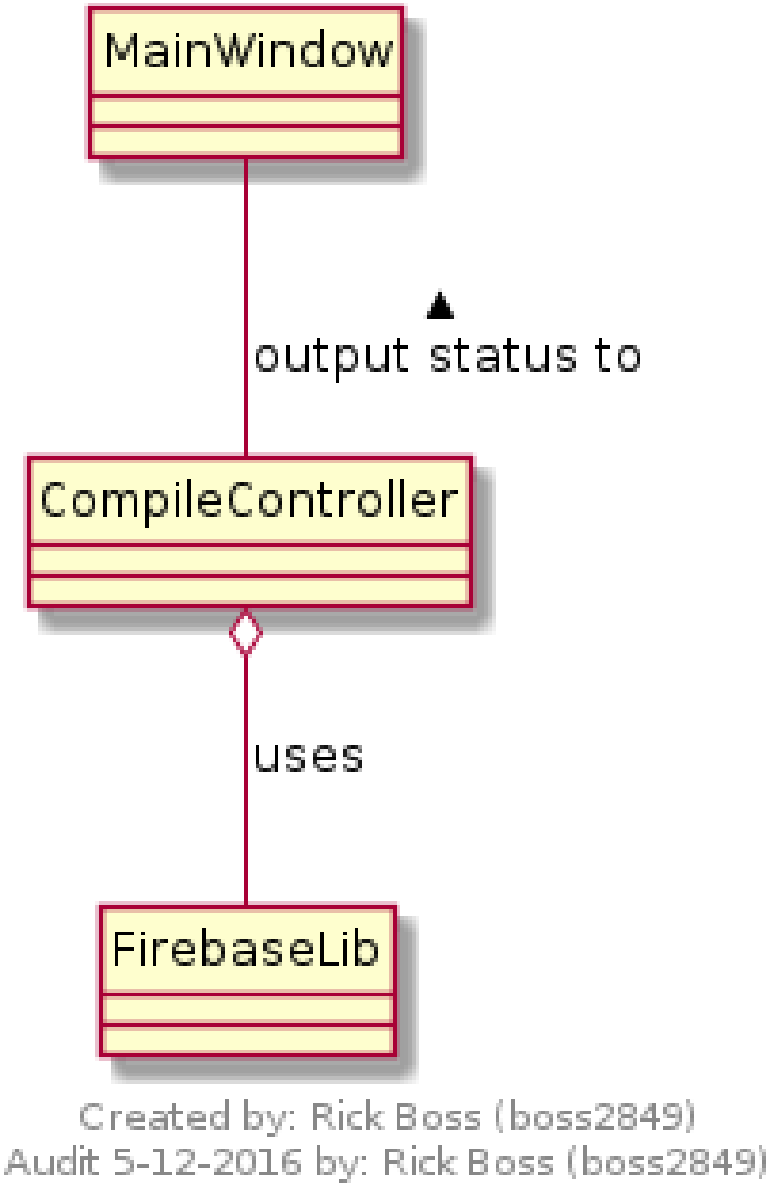


Figure 8.6: Diagram Dictonary: The CompilerController is invoked from the main window. From here the controller download the files using FirebaseLib. Once the files have been copied to the local filesystem, the controller invokes the standard Java Compiler through javac for compilation, capturing and returning the result.

8.7 Syntax (gall7417)

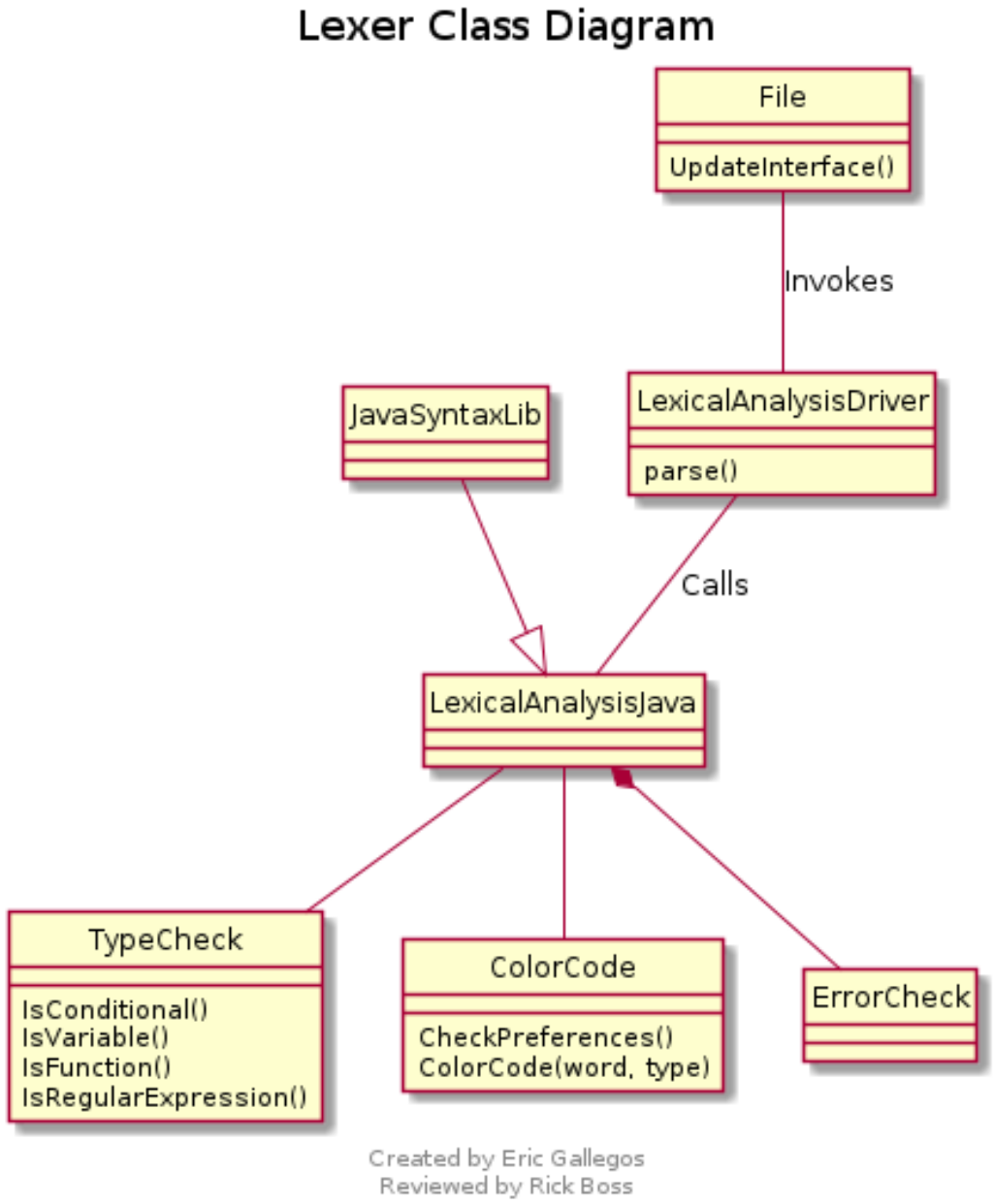


Figure 8.7: Diagram Dictonary: The File class regularly invokes the Lexical Analysis class to give feedback to users code input. The Lexical Analysis driver calls the corresponding Lexical Analysis class for the language used (Java). This language specific class checks for errors using any of the ErrorCheck classes, searches for word types using the TypeCheck class, and will assign the various words colors to reflect the word types using the ColorCode class.

8.8 Communication (jank6275)

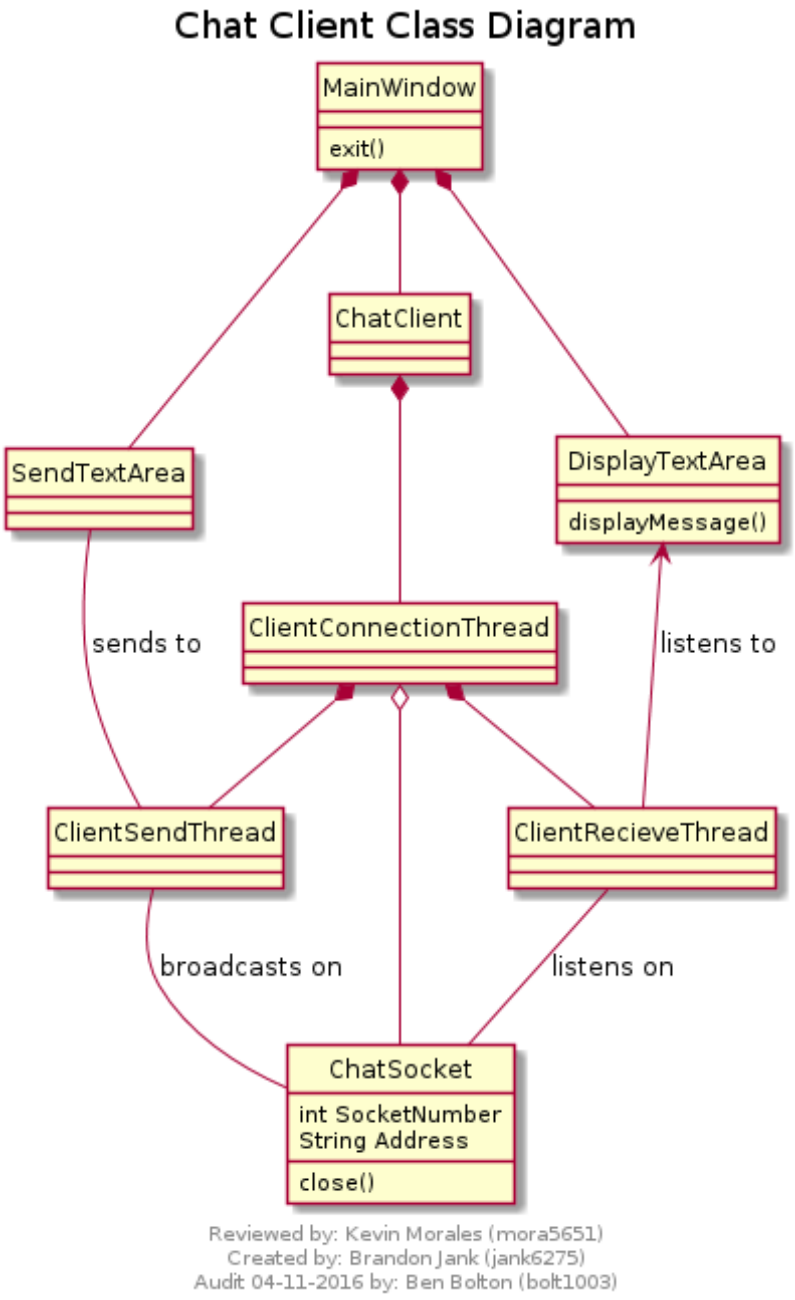


Figure 8.8: Diagram Dictonary: The ChatClient class will handle text communication in conjuction with the ChatServer class. The Main Window will be home to the ChatClient. The ChatClinet will consist of client send/recieve threads to handle user input/output in the Main Window via the ChatSocket.

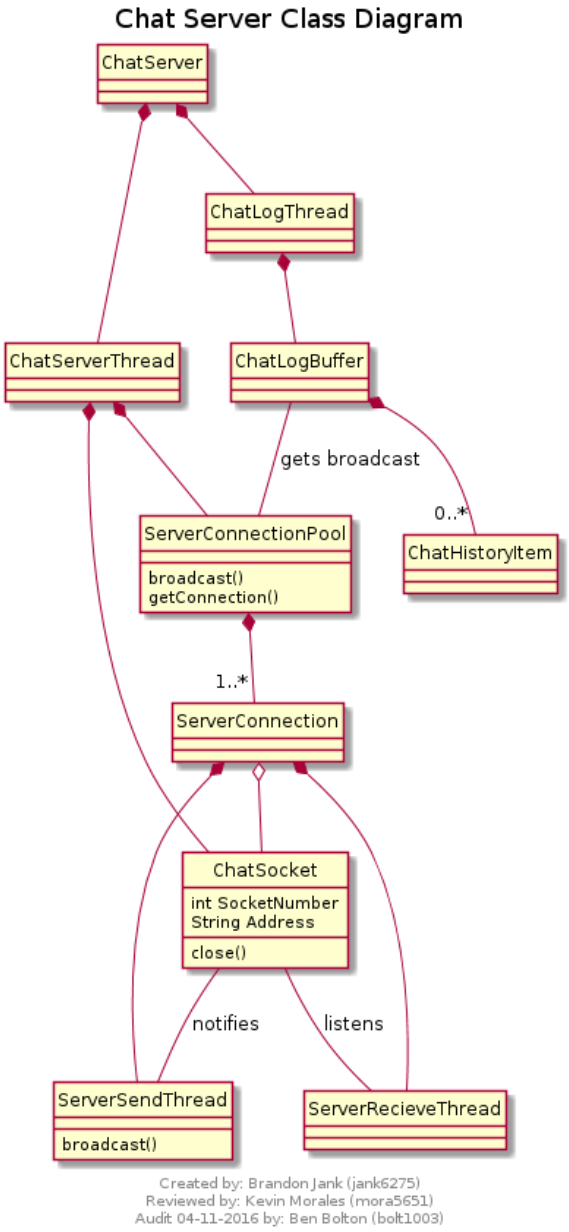


Figure 8.9: Diagram Dictonary: The ChatServer class will handle text communication between ChatClient(s). The ChatLogThead records any messages broadcast by chat clients in the ServerConnectionPool as a ChatHistoryItem. Each ServerConnection consists of a send and recieve thread that utilize the ChatSocket to broadcast messages.

8.9 Project File Editor (snev7821)

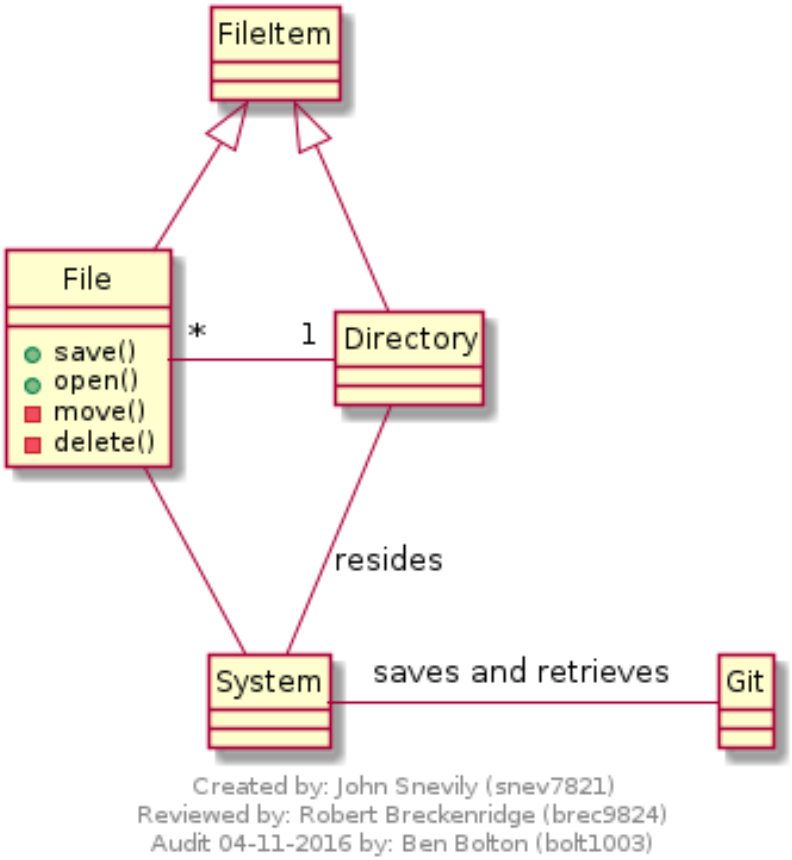


Figure 8.10: Diagram Dictonary: The project File Editor is simply the file system for Squire. It provides basic read/write permission for any user related to a project. Only admins of a project may move project files, delete old files, and create new files. Github integration is provided for storage and as a way to keep files up to date.

Chapter 9

Sequence Diagrams

9.1 Authentication (jank6275)

9.1.1 Login

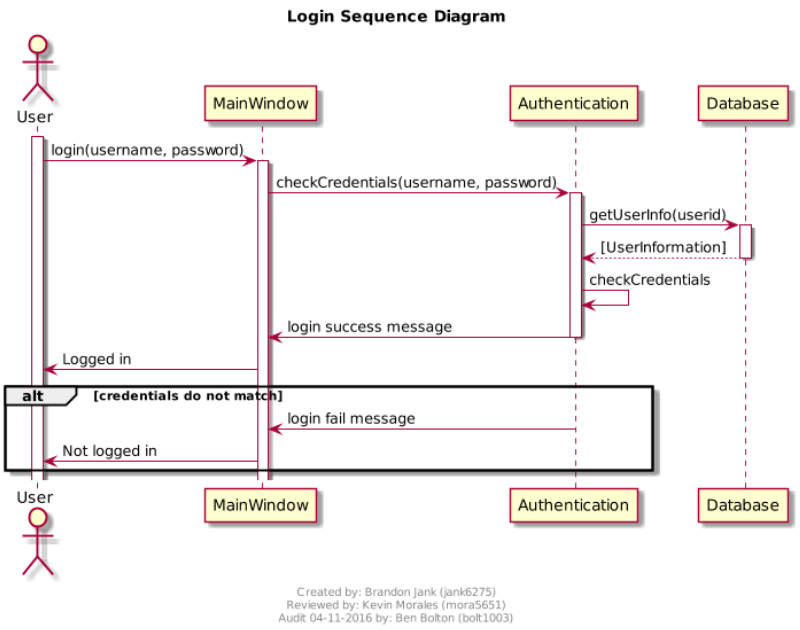


Figure 9.1: A sequence diagram for logging in to Squire.

9.1.2 Logout

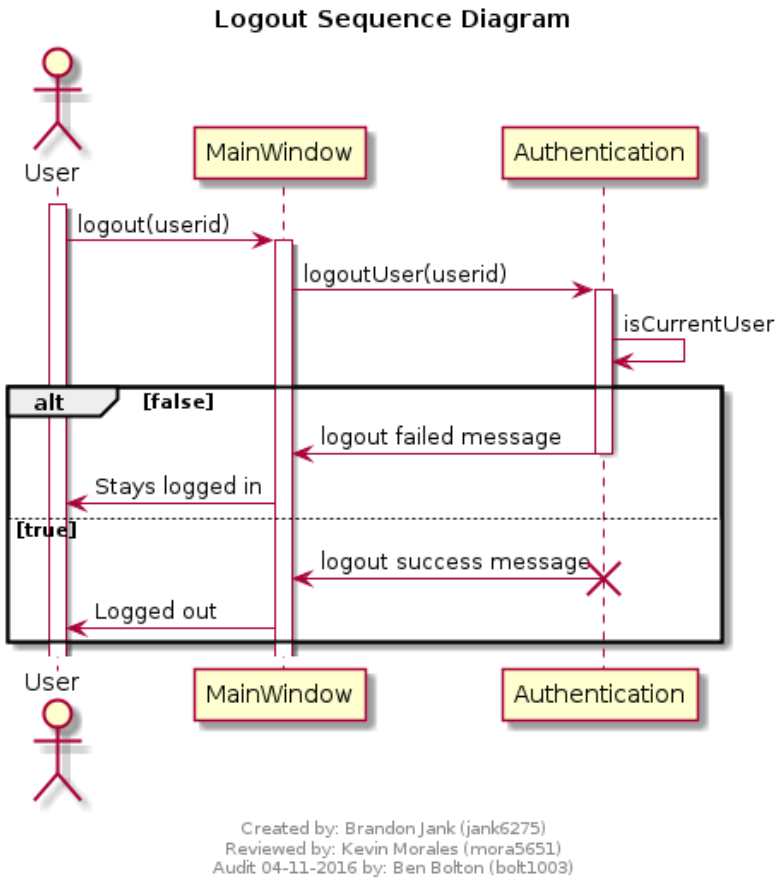


Figure 9.2: A sequence diagram for logging out of Squire.

9.1.3 Register

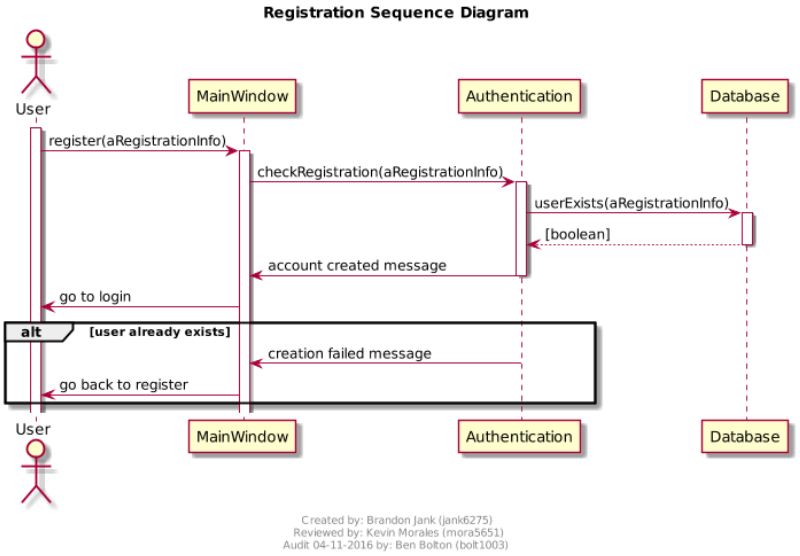


Figure 9.3: A sequence diagram for registering a nmew account with Squire.

9.2 Project Management (mars2681)

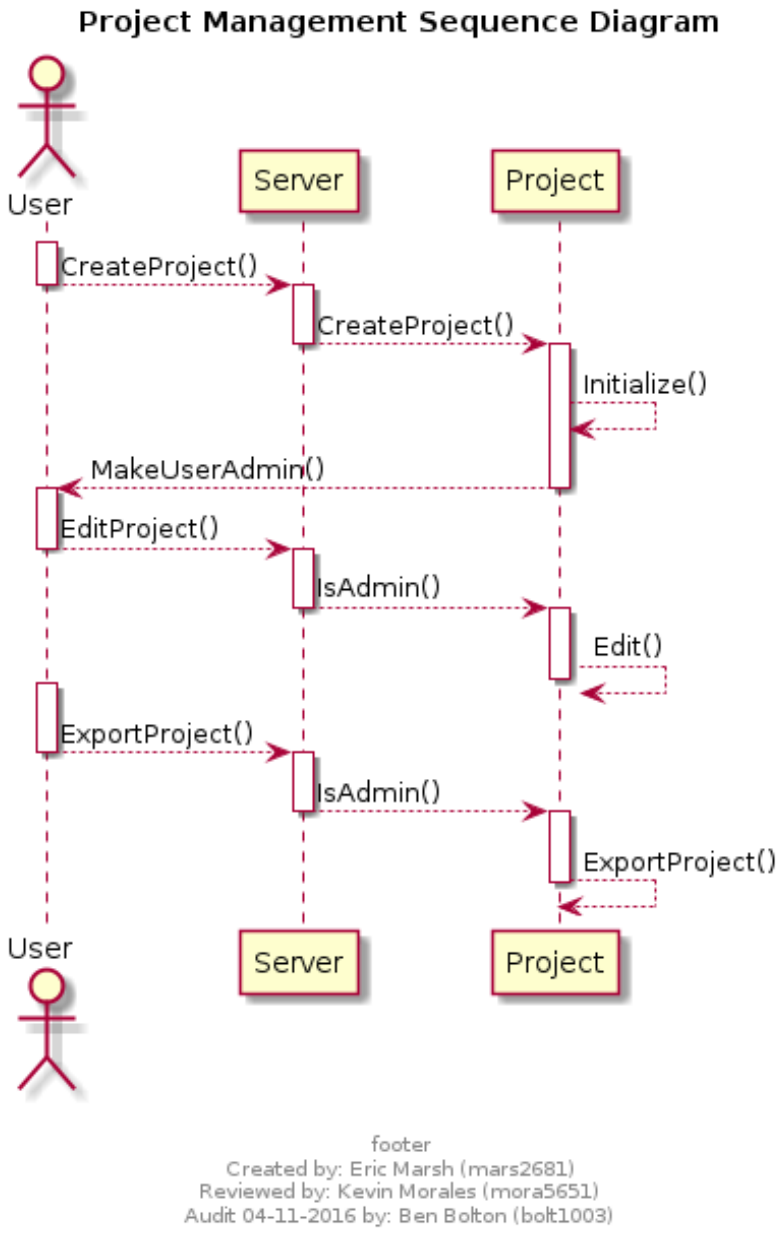
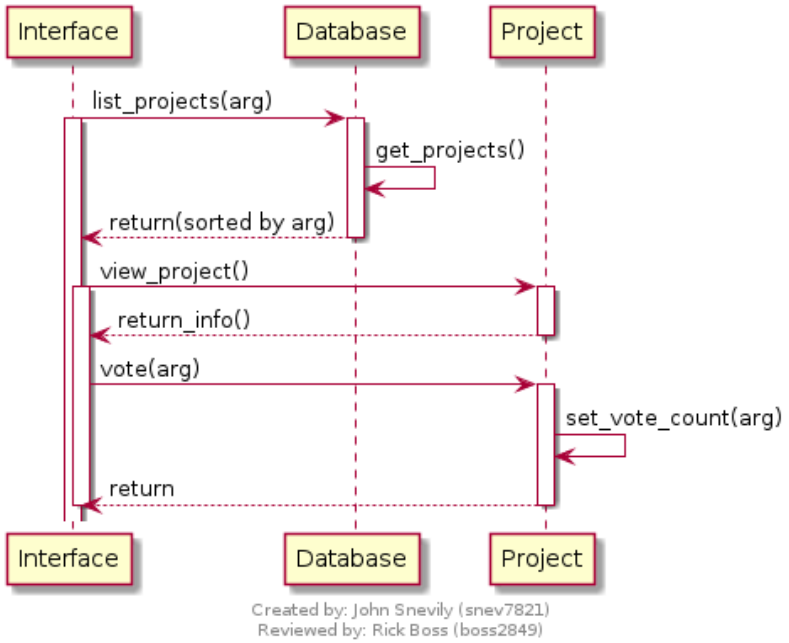


Figure 9.4: A sequence diagram showing how user can manage their projects

9.3 Project Ideas (snev7821)



9.4 Settings, Preferences, and Profile (gall7417)

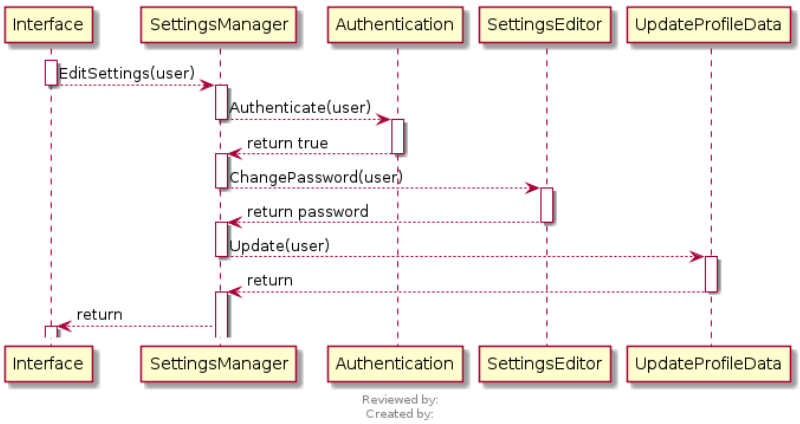


Figure 9.5: Example of a sequence during a users profile settings change. In this case the user is changing his password, however a similar sequence would apply for other profile settings.

9.5 Compiler (bolt1003)

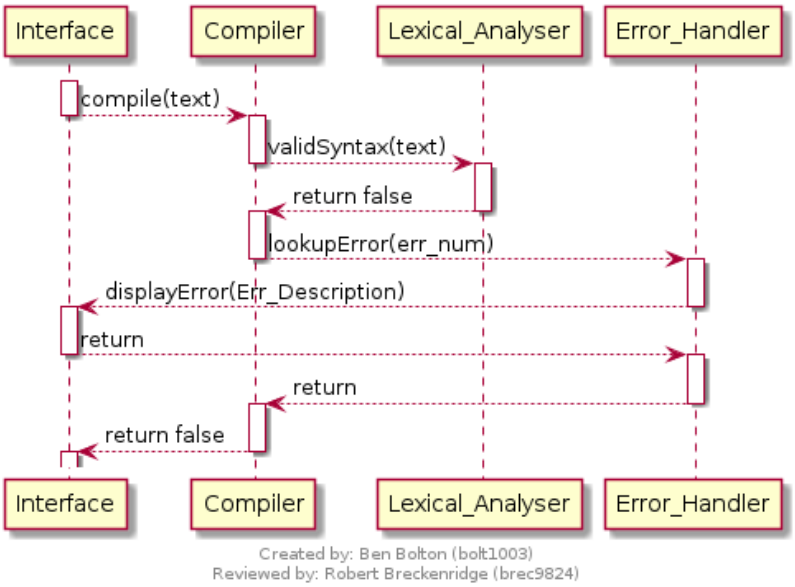


Figure 9.6: The sequence of a user attempting to compile a program with a syntax error.

9.6 Syntax (mora5651)

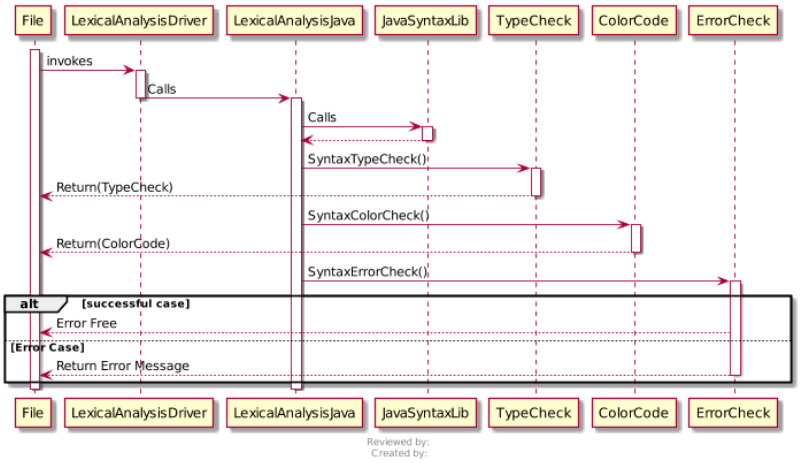


Figure 9.7: Sequence diagram for java syntax in the file editor being updated in real-time.

9.7 Communication (boss2849)

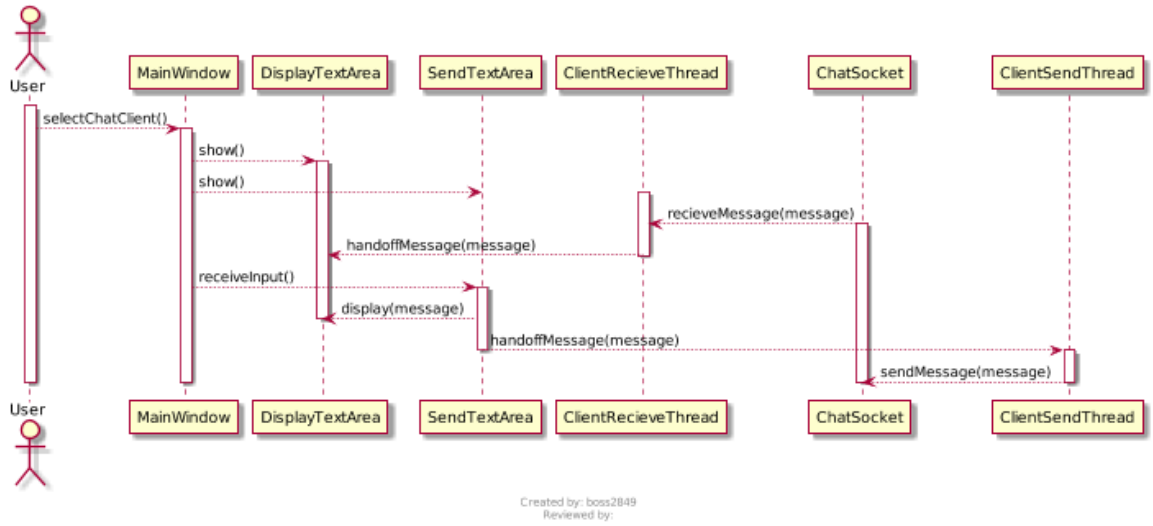


Figure 9.8: Sequence diagram for chat client interaction.

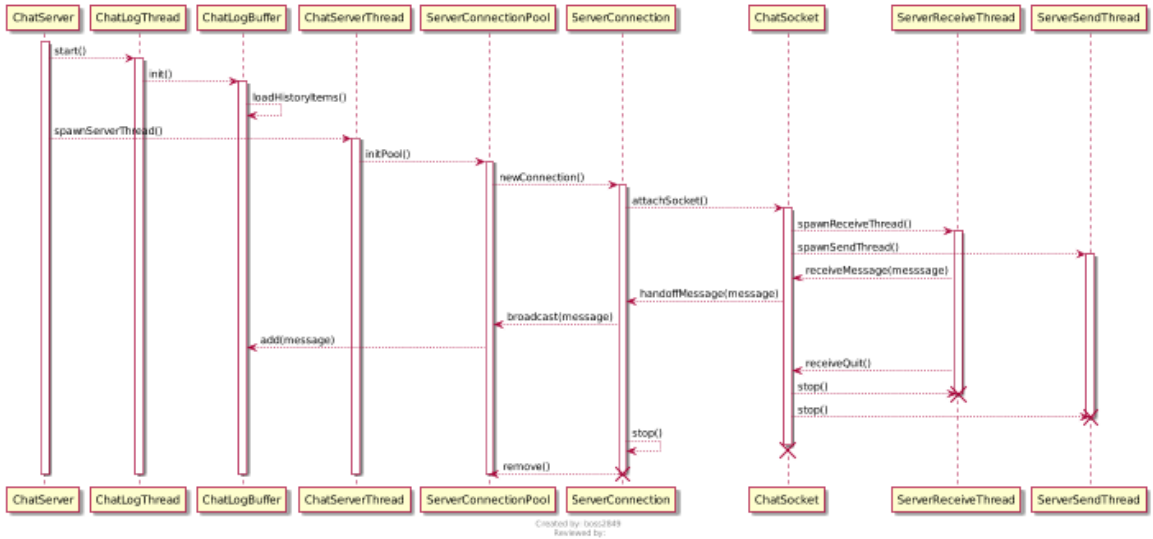


Figure 9.9: Sequence diagram for chat server interaction.

9.8 Project File Editor (brec9824)

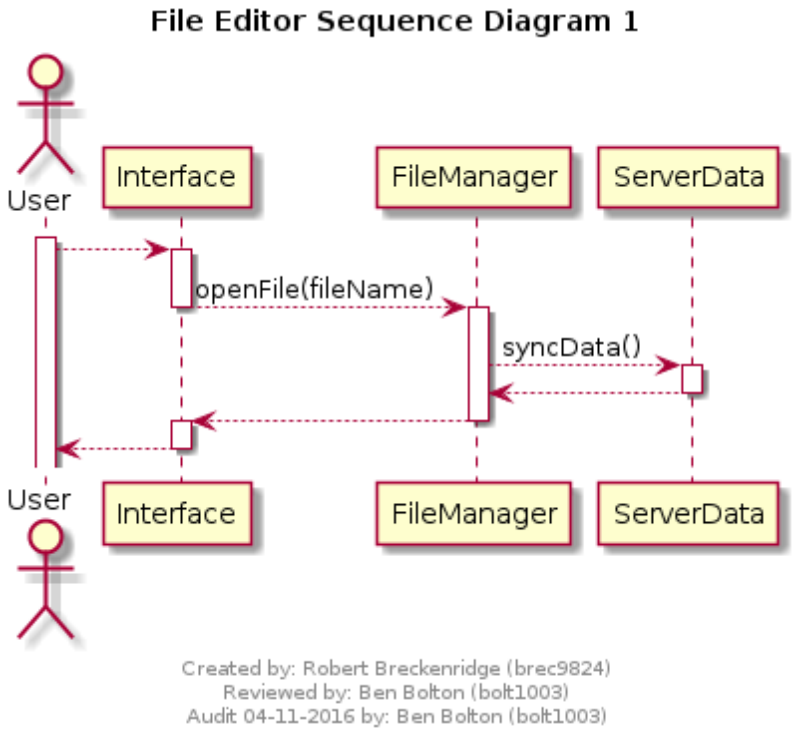
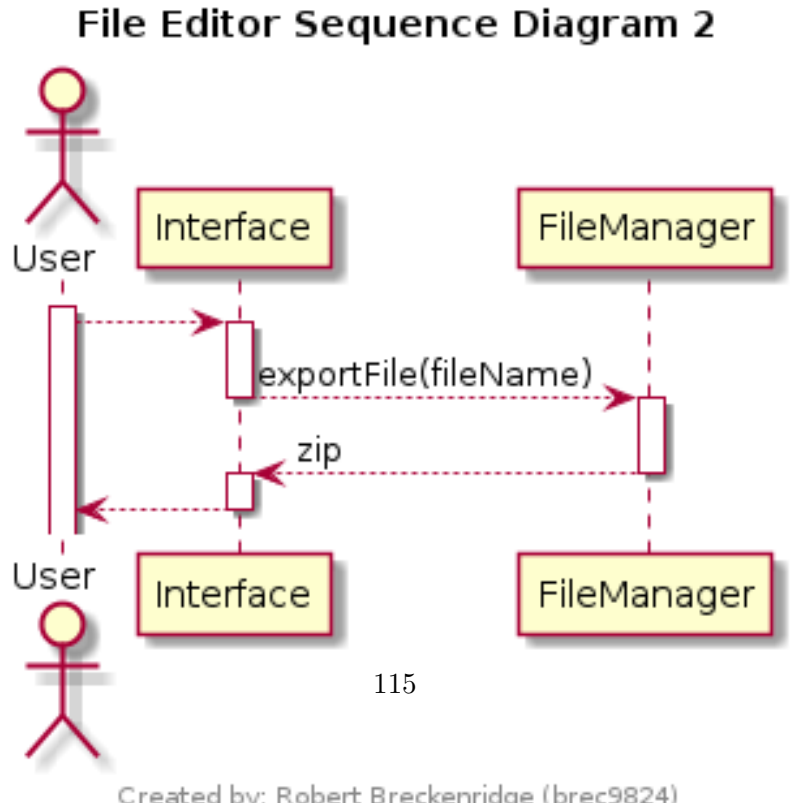


Figure 9.10: Sequence diagram showing the control structure of opening a file in the project file editor. This sequence also relates to delete file, import file and add file.



Chapter 10

Statecharts

10.1 Project Forum (snev7821)

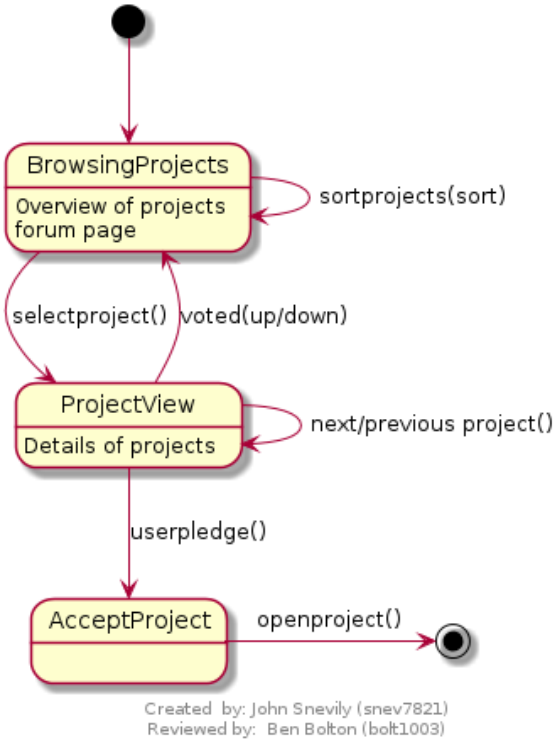


Figure 10.1: Statechart showing the flow of a user browsing the project forum page

10.2 Settings/Preferences (gall7417)

Figure 10.2: Statechart showing the flow of a user changing a user setting

10.3 Compile (boss2849)

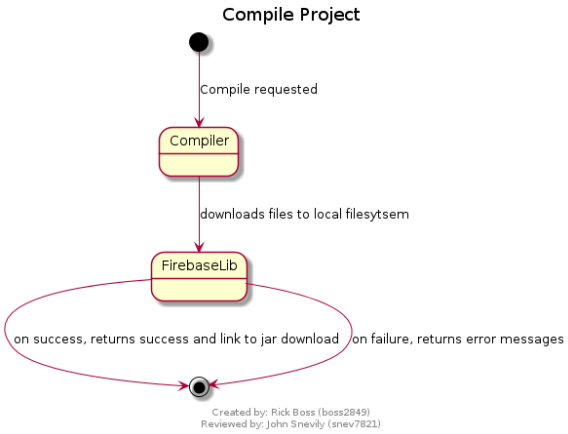


Figure 10.3: Statechart showing the flow of a user compiling a file, or entire project.

10.4 Project Management (bolt1003)

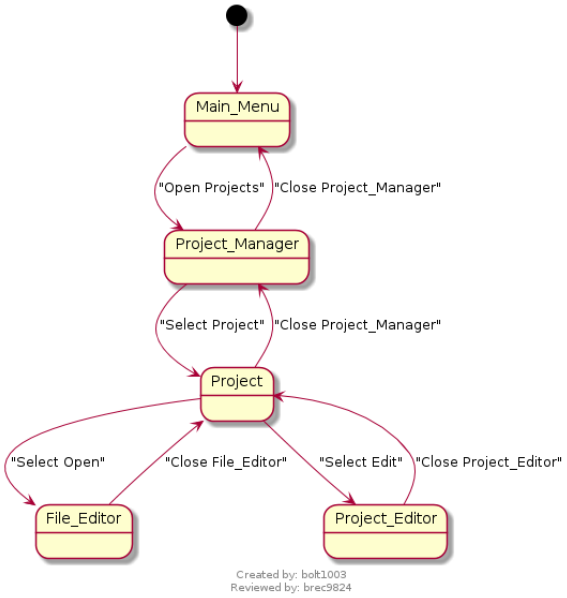


Figure 10.4: .

10.5 Communication (jank6275)

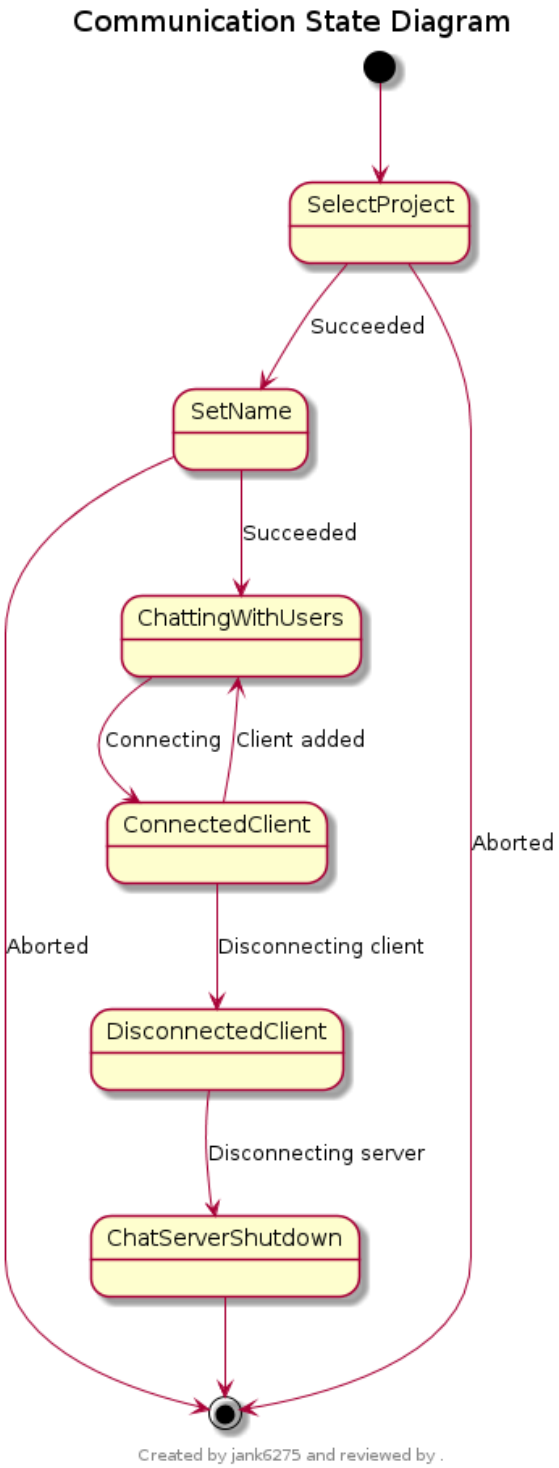


Figure 10.5: A statechart that shows the various states of the communication system.

10.6 Authentication (brec9824)

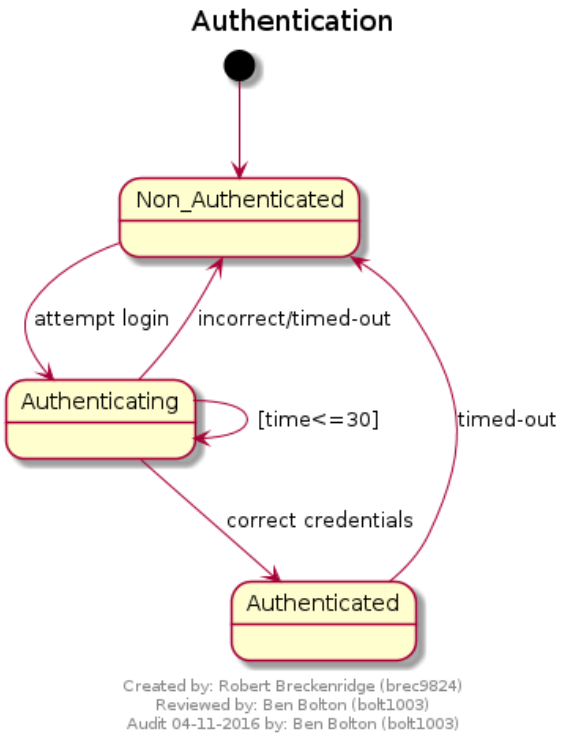


Figure 10.6: Statechart showing the flow between states for authentication.

Chapter 11

Mockups

11.1 Project Management (bolt1003)

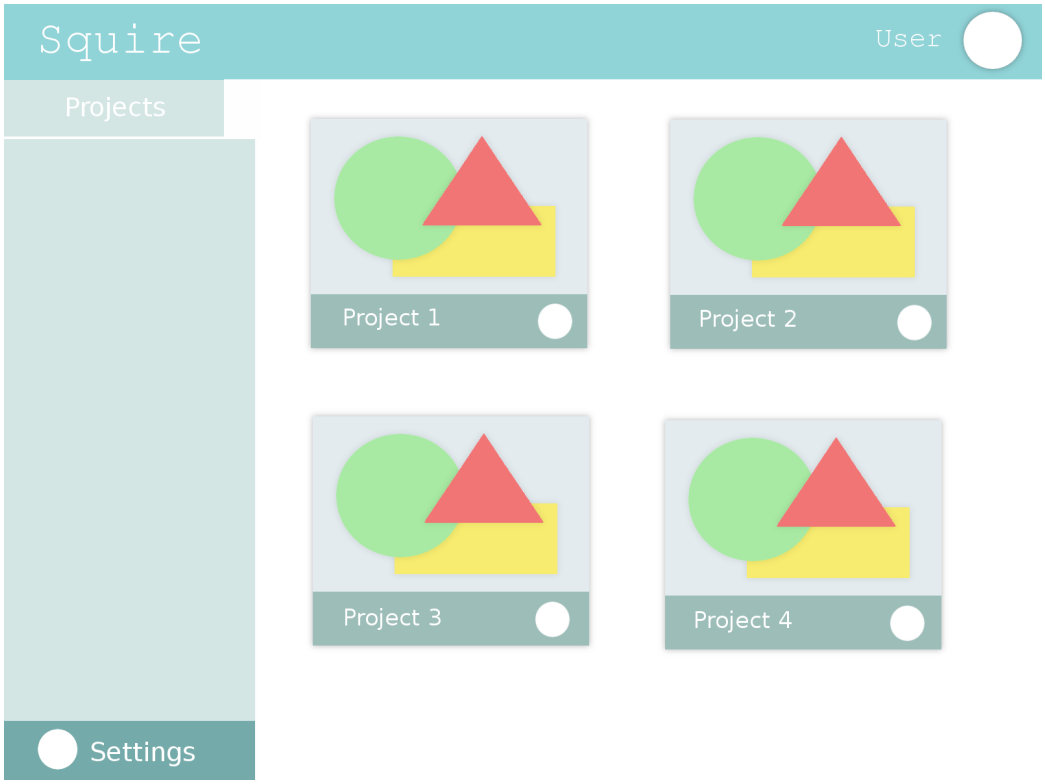


Figure 11.1: A possible look for the project management page

11.2 Settings and Preferences (gall7417)

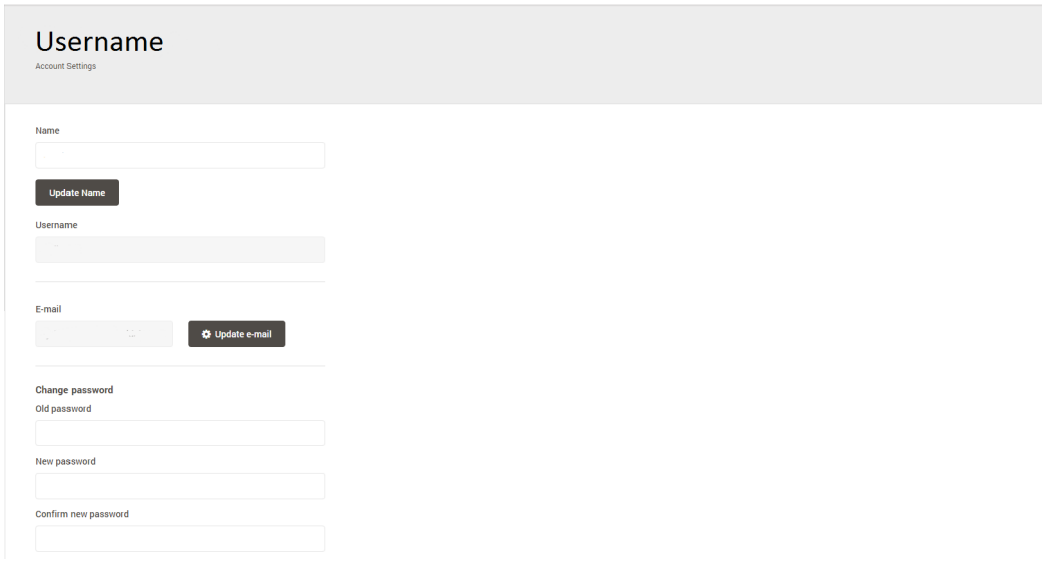


Figure 11.2: Po

11.3 Squire Editor (jank6275)

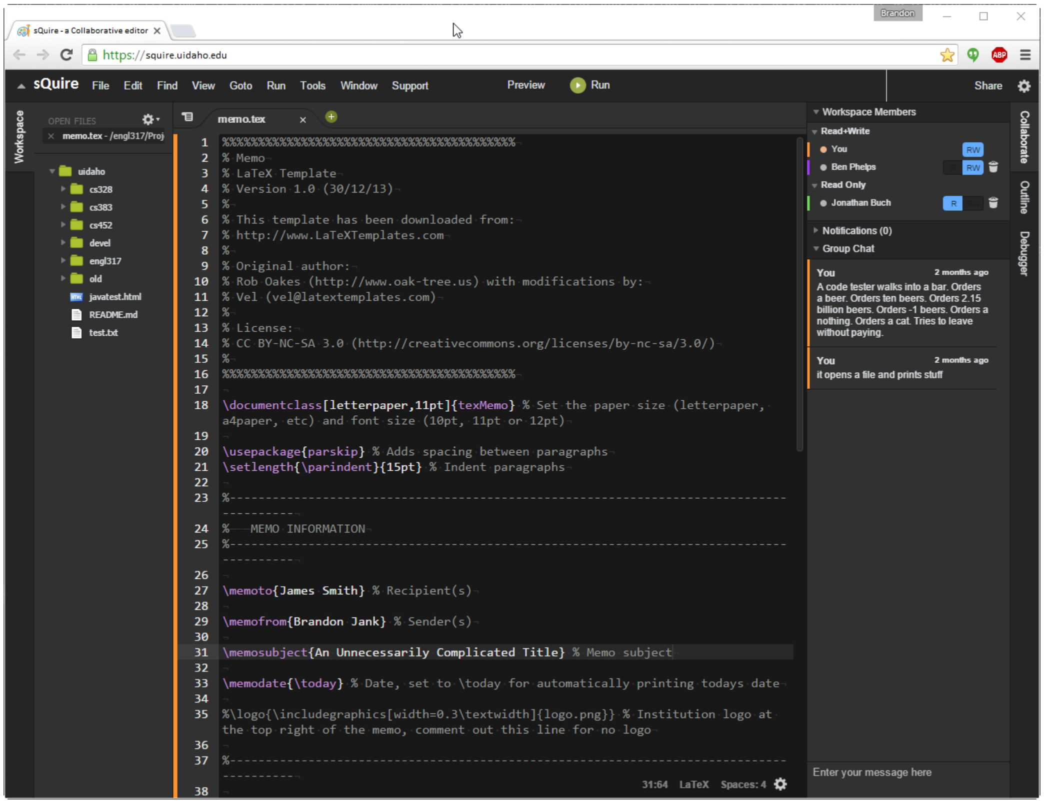


Figure 11.3: What we envision Squire IDE to look like.

11.4 Squire Chat (jank6275)

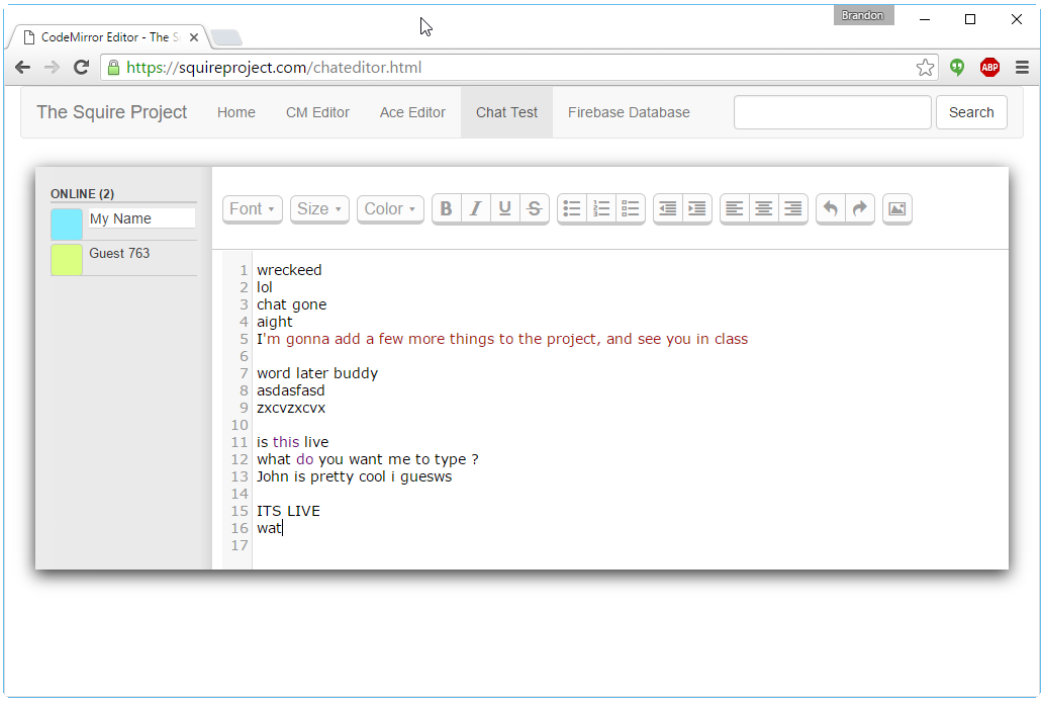


Figure 11.4: How communication could work in squire.

11.5 Project Browser (snev7821)

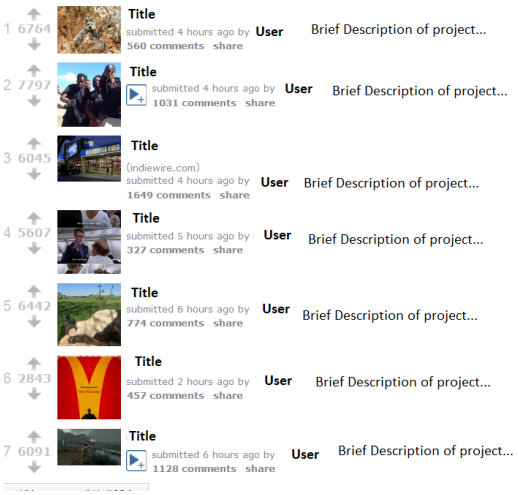


Figure 11.5: Created from Paint and Reddit, the project browser may end up looking somewhat similar

11.6 Compile Mockup (boss2849)

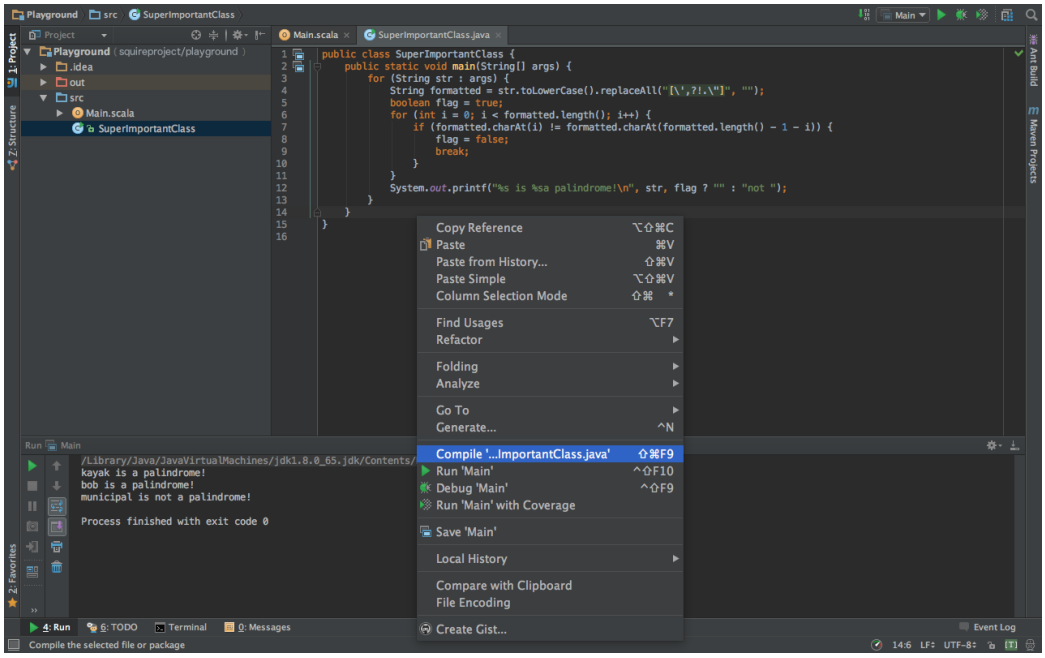


Figure 11.6: A mockup, based off IntelliJ of compiling a file.

11.7 Syntax Mockup (mars2681)

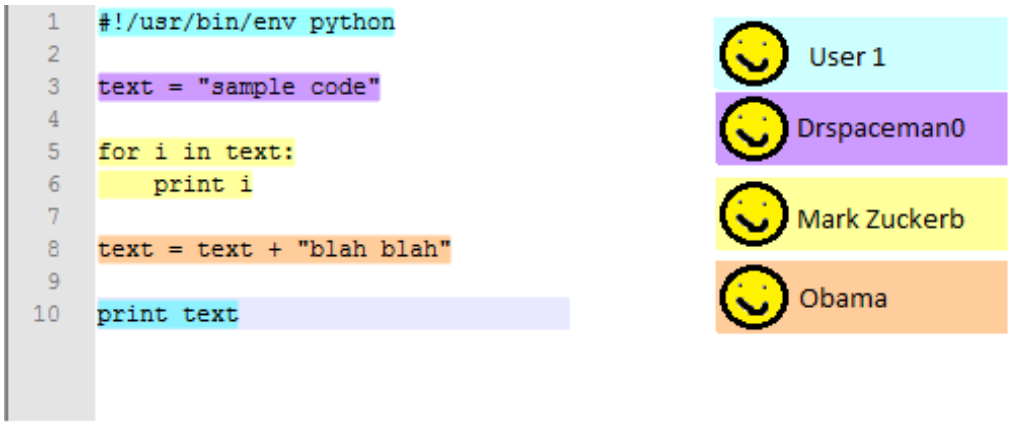


Figure 11.7: A mockup, created from Notepad++ and paint. Each user's input has its own shade. The light blue shade on Line 10 shows current user's cursor position.

11.8 Home (brec9824)

Figure 11.8: A mockup for the look of squire's home page. Created using html and css.

11.9 Register (brec9824)

Figure 11.9: A mockup for the look of squire's register page. Created using html and css.

11.10 Login (brec9824)

Figure 11.10: A mockup for the look of squire's login page. Created using html and css.

11.11 Project Finder (brec9824)

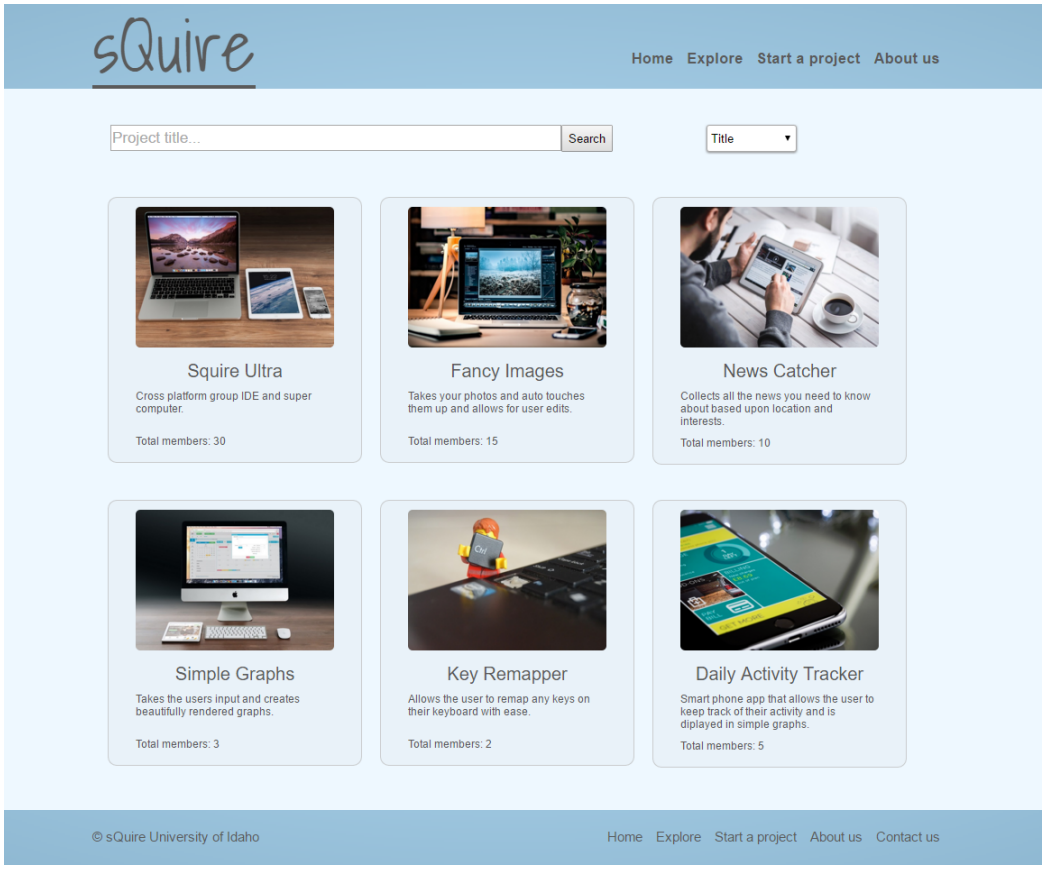


Figure 11.11: A mockup for the look of squire's project finder page. Created using html and css.

11.12 Project Idea (mora5651)

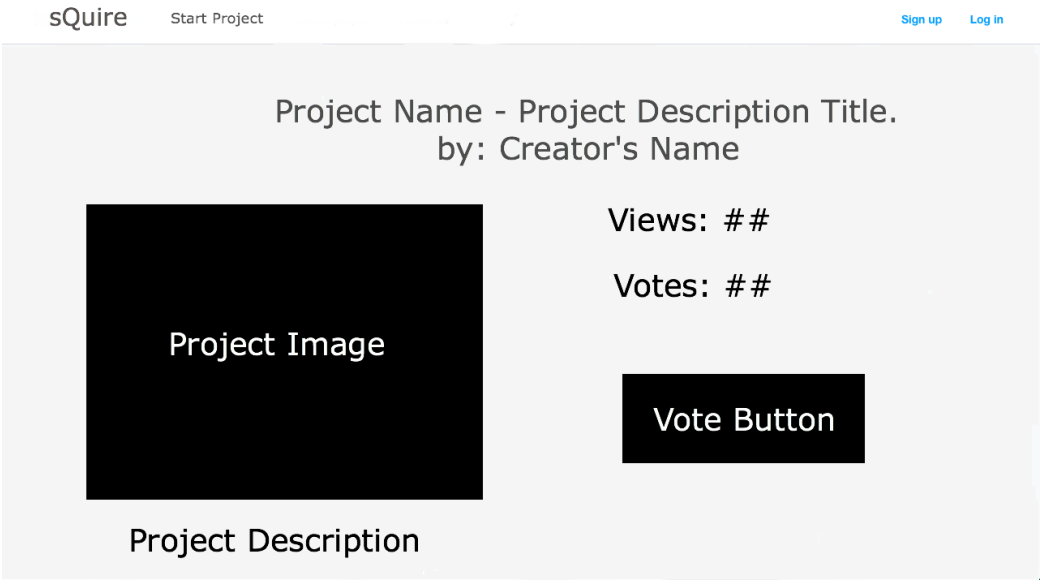


Figure 11.12: A mockup of how the project idea page should look like.

Chapter 12

Implementation

12.1 Overview

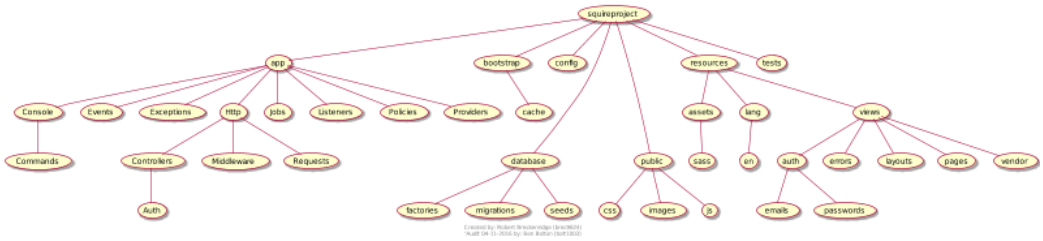


Figure 12.1: Squire's file structure is organized to allow for a simple and concise project flow. This structure includes seven main subfolders app, bootstrap, config, database, public, resources, and tests. Each containing crucial files allowing squireproject.com to exist.

- Main Subfolders:
  - app: Contains the files that allow for the main functionality of Squire
  - bootstrap: Contains files for configuring auto loading and for bootstrapping the framework.
  - config: Contains project squires configuration files.
  - database: Contains files pertaining to database migrations and seeds.
  - public: Contains the front controller and web page assets.
  - resources: Contains raw assets, front end views, and localization files.
  - tests: Contains files for automated testing.

12.1.1 app folder

- Console: Artisan commands for Laravels built in commands.
  - Commands: Contains console cammands for artisan.
    - PopulateProjectsTable.php: Auto populates the mysql database `projects_table` with some premade data.

- Events: Event classes for alerting different sub-systems to given actions.
- Exceptions: Exception handlers for dealing with exceptions.
  - Handler.php: Exceptions that should and should not be reported.
- Http: Controllers for routing data to and from squireproject.com. Middleware usable by different aspects like authentication. Requests
  - Controllers: Contains all files dealing with routing data.
    - Auth: Contains files dealing with authentication routing.
      - AuthController.php: Routes data dealing with authentication to and from the user.
      - PasswordController.php: Routes data pertaining to passwords from the user to the database.
    - Controller.php: Base class that all controllers extend for routing data.
    - PagesController.php: Routes data dealing with html/php pages like index.blade.php, projectfinder.blade.php and etc from and to the user.
  - Middleware: Included software for simplifing certain tasks.
    - Authenticate.php: Handles incoming authentication requests and checks if the user is autherized to see the page and if not direct them to a 401 page.
    - EncryptCookies.php: Encrypts cookies and sets which cookies to ignore.
    - RedirectIfAuthenticated.php: If user already authentcated then redirectes them to '/' page.
    - VerifyCsrfToken.php: Verifies csrf tokens.
  - Kernel.php: The applications HTTP middleware configuration.
  - routes.php: All routes are sent here then routed to the appropriate controller class.
- Jobs: Queueable jobs for Squires request cycle.
  - Job.php: This is the base class to extend when creating new jobs.
- Listeners: Handler classes for events to allow for performing actions based on triggered events.
- Policies: Authorization policy classes for dealing with user action.
- Providers: Addable 3rd-party features for extending built in features.
  - AuthServiceProvider.php: Services used for authentication.
  - RouteServiceProvider.php: Used for mapping web routes through middleware.
- User.php: Sets variables as hidden and/or mass assignable to allow for users to have hidden passwords and grouped assignable variables.
- Project.php: Defines the project model that is used in the MySQL database table named projects. Allows interacting with entries from the table through PHP code.

12.1.2 bootstrap folder

- cache:
  - services.php: Arrays of paths to different Laravel services.
- app.php: Sets up some app singletons.
- autoload.php: Sets up Composer to take care of needed classes.

12.1.3 config folder

- app.php: Configuration for the applications environment, debugging, localization, encryption, service providers, and etc.
- auth.php: Configuration for authentication options like what database users data is stored in and timeout time.
- broadcasting.php: Configuration for how events are broadcasted over websockets.
- cache.php: Configuration for cache storage settings and locations.
- compile.php: Configuration for custom classes artisan commands can create.
- database.php: Configuration for how mysql should be setup for example port number, username, password, and etc.
- filesystems.php: Configuration for local and cloud disk paths and settings.
- mail.php: Configuration for email services that should be used for example smtp, port number, username, and etc.
- queue.php: Configuration for Laravel's backend queue system.
- services.php: Configuration for storing third party service's credentials
- session.php: Configuration for lifetime of user sessions while idle.
- view.php: Configuration of paths the system should look for views (php/html pages).

12.1.4 database folder

- factories: Used primarily in unit testing, allows for entries to be made with a default set of attributes.
  - ModelFactory.php:
- migrations: Contains migrations for use with the Artisan migrate command. Migrations are used to create the tables in the database as well as modify their schema in the future. Also allows for rolling back of migrations to "reset" the state of the database.

  - `create_users_table.php`: Database table used for storing user's info.
  - `create_password_resets_table.php`: Database table used for storing password reset info.
  - `create_projects_table.php`: Database table used for storing project data.
- seeds: Contains file used with the Artisan seeder command, allows for databases to be seeded with initial/test entries.
  - DatabaseSeeder.php: Base class used for seeder commands.

12.1.5 public folder

- css: CSS files for html styling.
  - main.css: Contains all css styling for all of squireproject.com
- images: Folder for general images to be used on the site, not those for specific projects.
- js: Contains Javascript files.

12.1.6 resources folder

- assets: raw assets like SASS.
  - sass: For storing raw SASS files.
- lang: setting messages and their languages used for user messages.
  - en: English based strings for messages.
    - auth.php: Authentication messages.
    - pagination.php: Pagination messages.
    - passwords.php: Password messages.
    - validation.php: Validation messages.
- views: front end html pages users see when viewing squireproject.com.
  - auth: Contains views for authentication
    - emails: Contains view for resetting password via email.
      - password.blade.php: Contains message that will be sent to the user with a link to reset their password.
    - passwords: Contains views for resetting password
      - email.blade.php: View users will see when they want to get an email to reset their password.
      - reset.blade.php: View for actually reseting a user's password.
    - login.blade.php: View where user can login to the website.
    - register.blade.php: View where user can register to become a part of sQuire
  - errors: Contains views for general errors.

    - 503.blade.php: The view to show a user when they encounter a 503 error
  - layouts: Contains layouts which can be extended by other views, provides inheritance for html files.
    - `main_layout.blade.php`: The main layout for most pages. Includes the Header, Footer, and Nav bar. Leaves the body to be filled by its descendants.
  - pages
    - about.blade.php: The view for the About Us page, contains information about sQuire and what it is that its trying to accomplish.
    - create.blade.php: The view for the Create Project page. This page is where users can create a new project with an image, title, description, and full explanation of the projects intent.
    - index.blade.php: The view for the landing page for sQuire. Redirects logged in users to the Project Finder Page, otherwise pitches sQuire to the viewer and leads them to Register/Login.
    - project.blade.php: The view where users can see the information about a project that has already been created. Shows a picture, title, description and full explanation of the project.
    - projectfinder.blade.php: The view containing all the projects on squire for users to browse through. Also has a search bar to filter projec
  - vendor

12.1.7 tests folder

- ExampleTest.php: An example test to extend from in the future.
- TestCase.php: The base testing case for the project.

12.2 Prototype Implementations

12.2.1 Server Stack (jank6275)

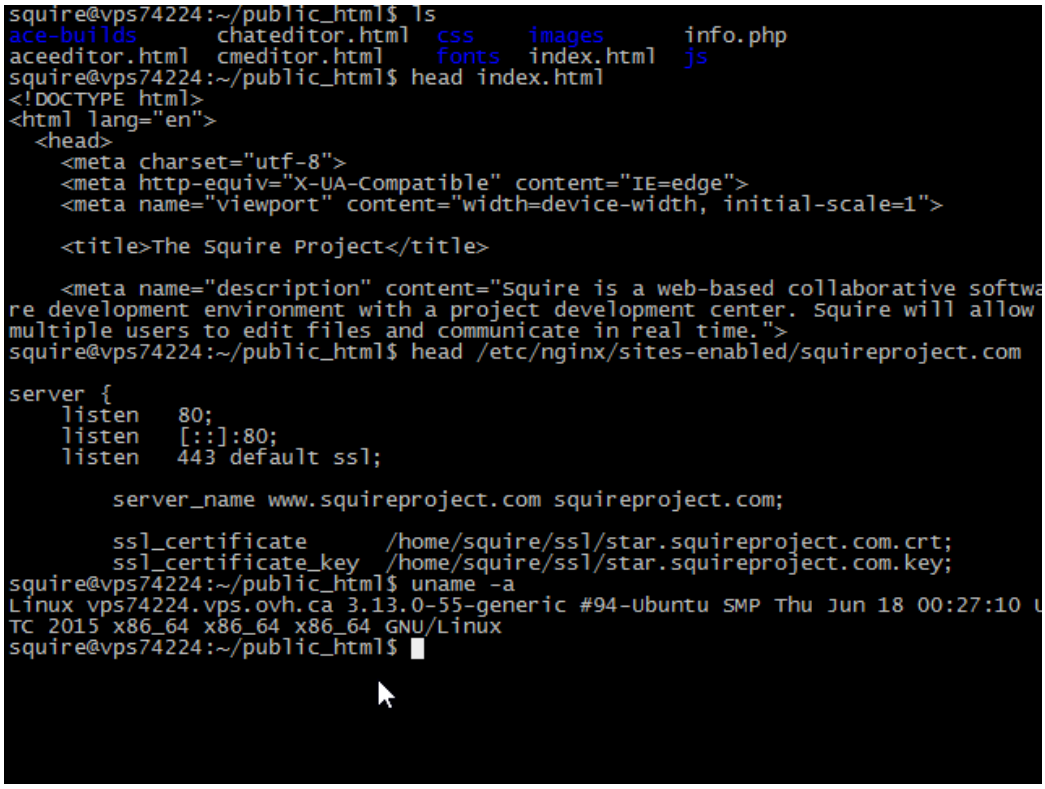


Figure 12.2: The internet domain squireproject.com was registered and an SSL certificate was generated for secure communications. A virtual private server was provisioned and Ubuntu 14.04 installed and then hardened to common network attack vectors. Nginx and MySQL were installed and configured. The web content was written in HTML and bootstrap CSS.

12.2.2 Editor (jank6275)

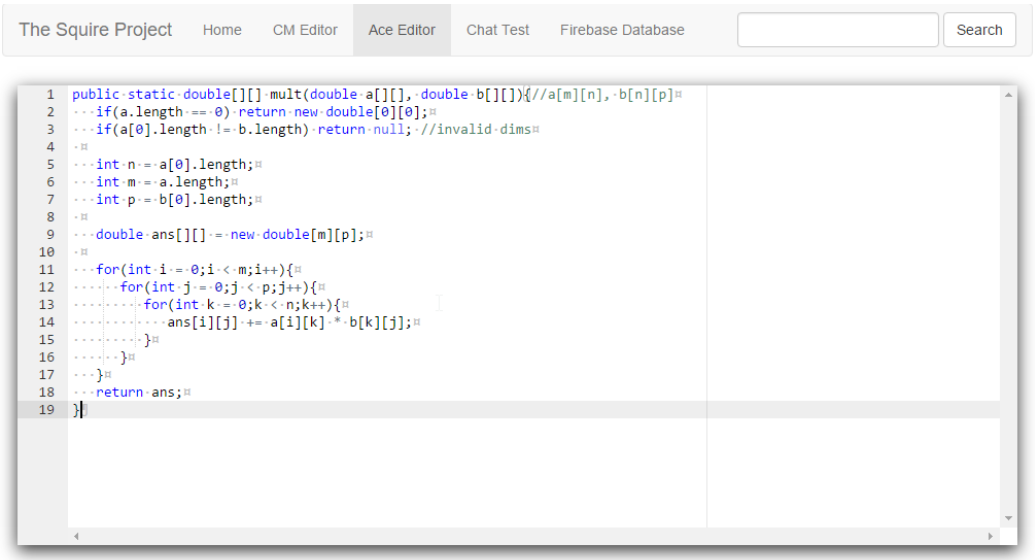


Figure 12.3: Java syntax highlighting, ability to save, and persistent storage prototypes implemented.

12.2.3 Create Project (boss2849)

Figure 12.4: The create project page was made for users to submit new project ideas. It allows them to upload a thumbnail, add a short description (used in brief views of the project) and a full project description. Submitting will create a new project and take the user to that page.

12.2.4 View Project (boss2849)

Figure 12.5: Each project will have a page dedicated just to the description of the project. Here other users can see the main project image, a short description, and the full explanation of the project.

12.2.5 Text Search (snev7821)

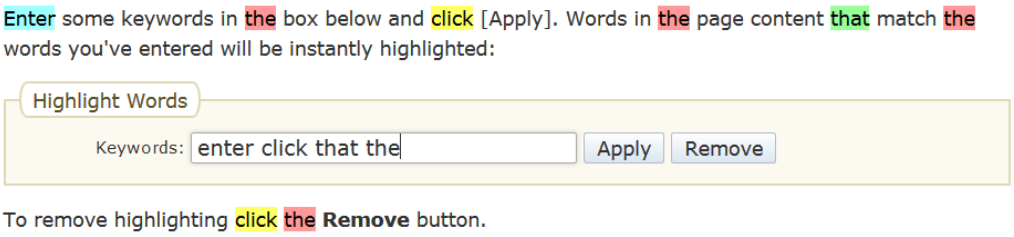


Figure 12.6: Text should be searchable per page, with each term being highlighted a different color.