sQuire: A Collaborative Software Development Tool

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

February 9, 2016

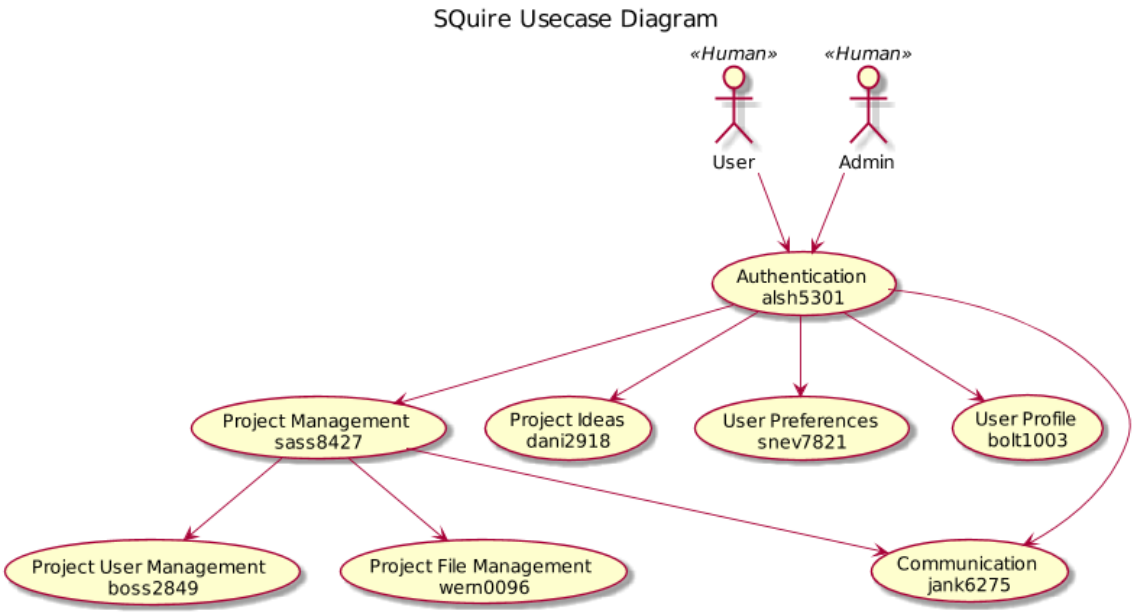
Contents

Chapter 1

Introduction

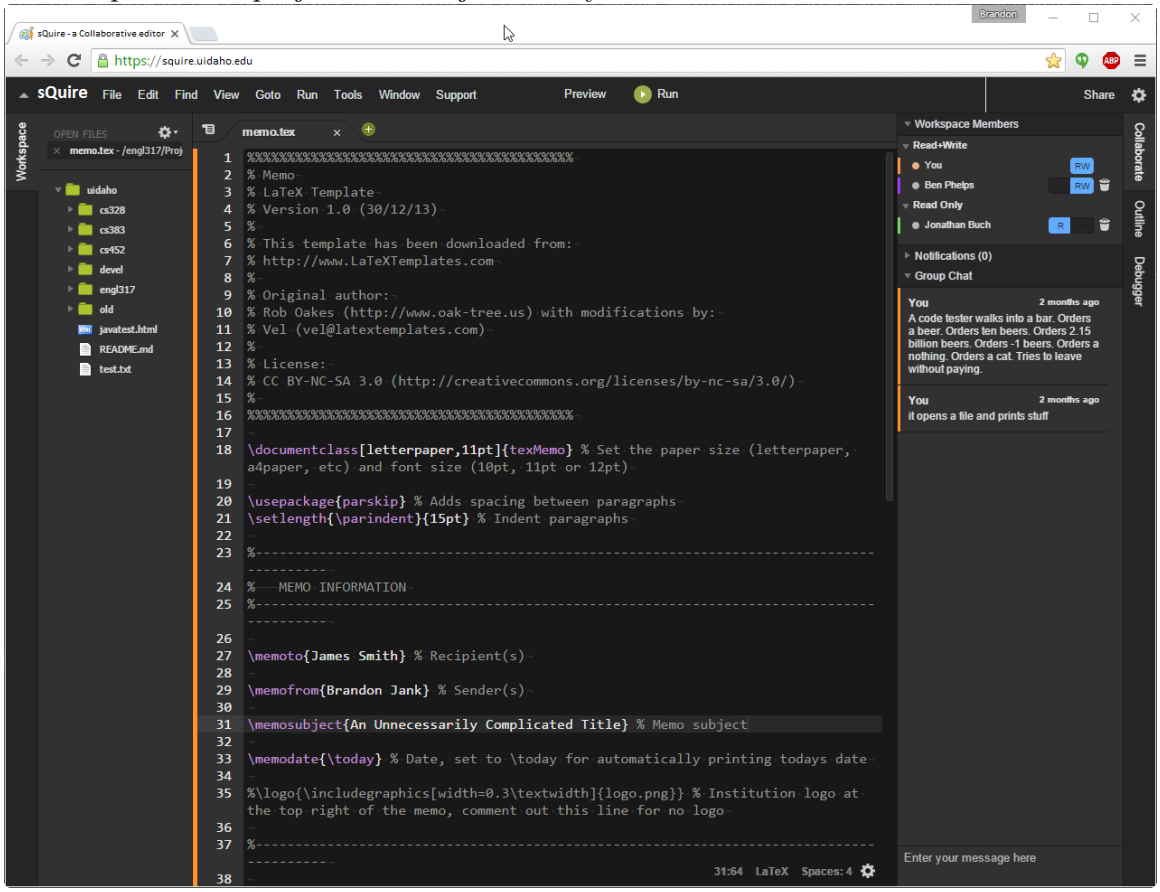
1.1 Program Premise

1.2 Use Case Overview (jank6275)



A usecase diagram that shows all of sQuireś features.

Figure 1.1: Squire will be 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. First, projects are 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 money.



Chapter 2

Requirements Documentation

Non-functional requirements describe how the system works, while functional requirements describe what the system should do. Dr. J's list of requirements:

- User profiles should include persistent data including project ownerships and memberships, friends, e-mail, profile image.
- Users should have easy access (background?) awareness information of other users, especially friends and members of shared projects.
- 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.
- Indication of who coded what could be background color, underline color, sidebar color, shade/fill pattern, or icon/avatar.
- Since multiple people might edit a given line over its history, support for past history or anyhow multiple persons in this indication is strongly recommended.
- Teams should decide whether write access to shared editing should be turn-based or simultaneous
- Teams should decide if voice or video is essential. Voice, if supported, might be restricted as to number of listeners and/or number of simultaneous transmitters. Video, if supported, might be restricted as to number of viewers and/or number of simultaneous
- Capable of supporting editing, compilation, and execution of Java programs. Programs to be composed as projects and support multiple source code files across multiple directories within a shared top-level directory.
- Ability to import/export AND/OR function on projects/source code stored in the ordinary local file system.
- Multiuser, up to 32 users can share an IDE session.

- Syntax coloring; visual indication to see who coded what.
- Shared sessions should allow people to move around their view (read-only, at least) independently, and to quickly jump to where other users are looking.
- Users can text chat to individuals, transient shared session members, persistent project member lists, and all logged in users.

2.1 Functional Requirements

Functional requirements will specify a behaviour or function, for example "Display the name, total size, available space and format of a flash drive connected to the USB port" Other examples are "add customer" and "print invoice". Some of the more typical functional requirements include:

- Authentication (mars2681)
- Project Management (mora5651)
- Project Ideas (snev7821)
- User Profile (brec9824)
  - User Preferences
  - User profiles should include persistent data including project ownerships and memberships, friends, e-mail, profile image.
- Project File Editor (jank6275)
  - Syntax coloring; visual indication to see who coded what.
  - Teams should decide whether write access to shared editing should be turn-based or sim
  - Indication of who coded what could be background color, underline color, sidebar color, shade/fill pattern, or icon/avatar.
  - Indication of who coded what could be background color, underline color, sidebar color, shade/fill pattern, or icon/avatar.
  - Write access to shared editing should be simultaneous
  - Since multiple people might edit a given line over its history, support for past history or anyhow multiple persons in this indication is strongly recommended.
  - Multiuser, up to 32 users can share an IDE session.
  - Shared sessions should allow people to move around their view (read-only, at least) independently, and to quickly jump to where other users are looking.
- Communication (bolt1003)
  - Built-in, global text chat per project.

  - Private communications between other users.
  - A friends list with friend status icons and avatar.
  - A global list of current memembers in the project.
  - A list of current users working on the project.
  - A dialog with the user's name and file history when hovering over their icon.
  - Text chat will be built using standard protocols such as XMPP or IRC.
  - Allows the use of third-party chat clients.

- 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.
  - Use password hashing on a trusted system to ensure password privacy.
  - Hide password entry on user interface.
  - Allow password change and reset in case of compromise.
  - Mitigate security threats by testing against common abuse cases and vulnerabilities.
  - Validate user integrity before any processing is performed.
  - Ensure proper character sets for all input given.
  - All validation failures must result in rejection.
  - Implement a force halt procedure for runaway programs.
  - Establish system inactivity timeout after arbitrary amount of time.
  - Enforce authorization controls on all system requests.
  - Restrict access to resources and files outside of the users given resources.
  - Deny access to security protocols and configurations.

- Compiler (boss2849)
  - Design as a 'plugin' to the system - initial will be a Java compiler, but allow new compiler plugins to be written for other languages.
  - Compile code for execution within IDE.
  - Compile code and package to a JAR.
  - Compile file, file and dependents, project sub-modules, or entire project.
  - Smart compilation - recompile only what has been changed.
  - Allow temporary code freeze before compilation.
  - Cache snapshots of code on compilation.
  - Capable of supporting editing, compilation, and execution of Java programs. Programs to be composed as projects and support multiple source code files across multiple directories within a shared top-level directory.

2.2 Non-Functional Requirements

Non-functional requirements cover all the remaining requirements which are not covered by the functional requirements. They specify criteria that judge the operation of a system, rather than specific behaviours, for example: "Modified data in a database should be updated for all users accessing it within 2 seconds." Some typical non-functional requirements are:

- Performance for example Response Time, Throughput, Utilization, Static Volumetric
- Scalability is important to keep in mind during development. The system should be designed in such a way that will easily and reliably scale to accommodate a growing user base. One method of dealing with scalability would be allowing the core system to reactively spawn new slaves to aid in computational needs, such as compilation (as that will be more resource intensive the user base grows). (boss2849)
- Capacity (boss2849)
  - Limited space for projects that haven't been initiated (enough for documentation, images, etc.)
  - Reactively increasing capacity for projects proportional to the absolute needs of the projects. (no fluff)
  - Encourage developers to store large files elsewhere, e.g. GitHub LFS, AWS, etc.
- Availability (brec9824)
- Reliability (bolt1003)
  - sQuire will be a web application and will leverage the strengths of web technologies to make it reliable. sQuire will use a webhost such as, Amazon Web services. The webhost's infrastructure provides, redunancy for hardware, power and internet service.
- Recoverability (mora5651)
- Maintainability (mora5651)
- Serviceability (bolt1003)
  - Running sQuire as a web application allows it to quickly and easily rollback to a previous version or rollforward to a new version. This allows for rapid bug fixing. Infrastructure is redunane so equipment can be taken offline for repair without interupting the users.
- Security (jank6275)
- Regulatory (mars2681)
  - User upon creating an account must confirm they at least 18 years of age. This is an requirement of COPPA. User must also agree

- Manageability (brec9824)
- Environmental (gall7417)
  - sQuire will run as a browser application in google chrome.
  - sQire will run on virtually all modern processors.
- Data Integrity (gall7417)
  - sQuire will use tcp to for reliable data communications.
  - sQuire will use error checking upon large changes to ensure no drastic data corruption occurs.
  - sQuire will use a history/autosave feature in case of data loss.
- Usability (snev7821)
- Interoperability (snev7821)

2.3 Unused Requirements

- Since multiple people might edit a given line over its history, support for past history or anyhow multiple persons in this indication is strongly recommended.
- Teams should decide if voice or video is essential. Voice, if supported, might be restricted as to number of listeners and/or number of simultaneous transmitters. Video, if supported, might be restricted as to number of viewers and/or number of simultaneous. - NO, we have skype for a reason.