

sQuire: A Collaborative Software Development Tool

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

February 8, 2016

Contents

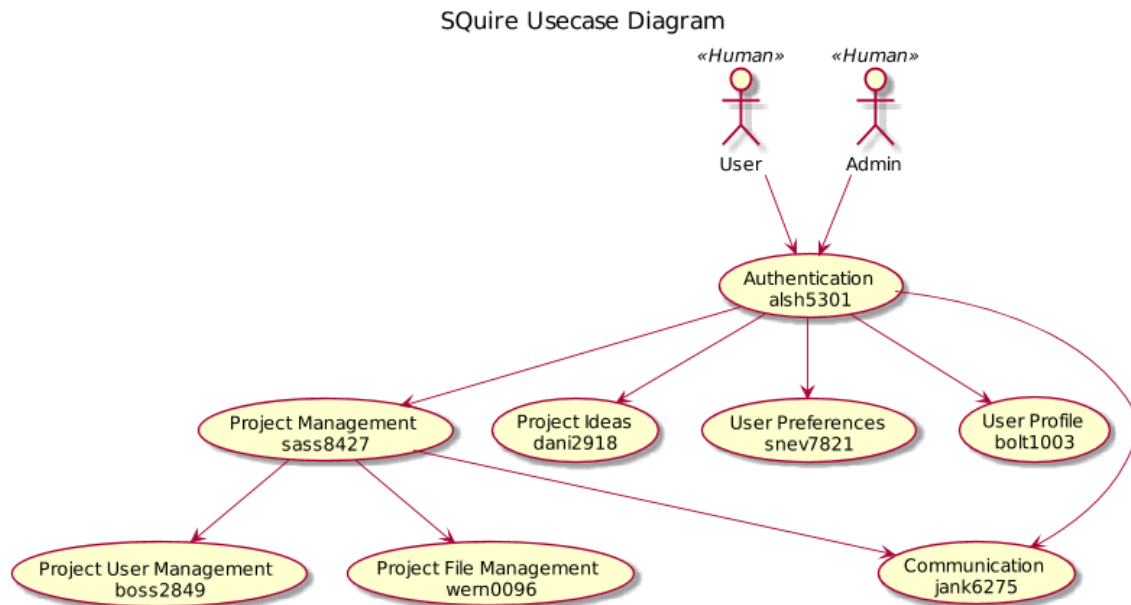
1	Introduction	2
1.1	Program Premise	2
1.2	Use Case Overview (jank6275)	2
2	Requirements Documentation	4
2.1	Functional Requirements	5
2.2	Non-Functional Requirements	5

Chapter 1

Introduction

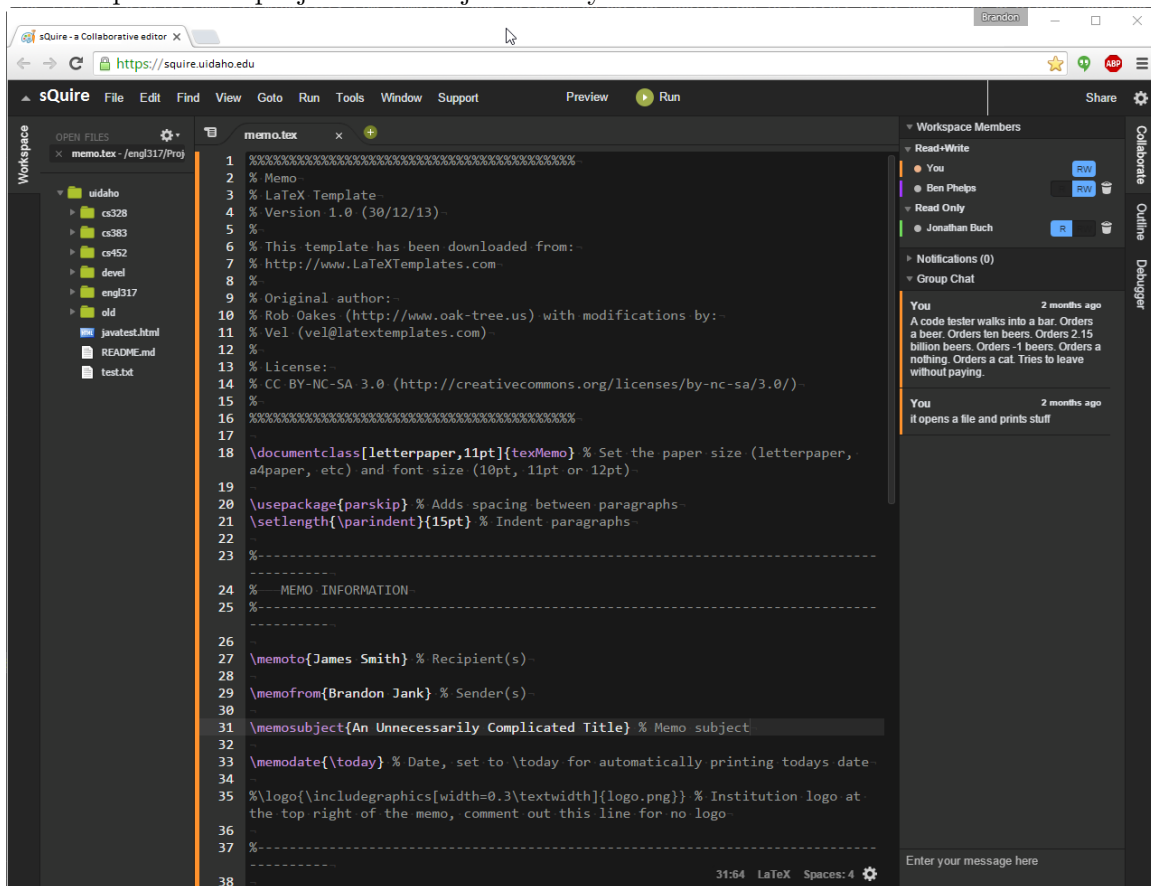
1.1 Program Premise

1.2 Use Case Overview (jank6275)



A usecase diagram that shows all of sQuire's 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 fledged 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 $i = 32$ 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:

- Business Rules
- Transaction corrections, adjustments and cancellations
- Administrative functions
- Authentication
- Authorization levels
- Audit Tracking
- External Interfaces
- Certification Requirements
- Reporting Requirements
- Historical Data
- Legal or Regulatory Requirements

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
- Capacity

- Availability
- Reliability
- Recoverability
- Maintainability
- Serviceability
- Security
- Regulatory
- Manageability
- Environmental
- Data Integrity
- Usability
- Interoperability