May15-31

SENIOR DESIGN I - CODERLAB

Design Document

Jacob Wallraff
Erich Kuerschner
Bryan Passini
Daniel Smith
Kyle Tietz
Jacob Wallraff

1 TABLE OF CONTENTS

2 PROJECT DEFINITION

By utilizing existing open source software and the power of the web, we propose a solution which will improve the experience of learning and teaching code; its name is CoderLab. The goal of CoderLab is to create a browser-based real-time collaborative code editing solution geared primarily for a classroom environment which creates an enhanced teacher-student experience when learning new coding concepts and languages. Instructors and laboratory guides can use CoderLab as a testbed for code demonstrations and allow students to collaborate with them on problems during class. Students in introductory courses at Iowa State can take advantage of this web application by revisiting code from class and by working jointly with partners on programming assignments.

3 GLOSSARY OF TERMS

Front-end: Refers to the code provided to and processed by the browser. This includes any JavaScript, HTML, CSS, and other assets, particularly those related to UI and visuals.

Back-end: General term to refer to any code running outside of the client's machine. This could refer to any code on the CoderLab web server, shareJS server, room manager server and any other supporting service.

Text operations: Common actions performed on blocks of text, such as adding and deleting characters.

Operational transforms: Class of algorithms behind the collaborative aspect of the project. The transforms allow for consistency of text operations across multiple browsers without errors and without overwriting data. Operational transforms are implemented by the ShareJS library.

ShareJS: Library that uses operational transforms to allow live, concurrent editing of text documents in the browser.

NodeJS: Software platform built on top of Chromium's V8 JavaScript engine and the libuv event library. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices or require high concurrency. Reference: nodejs.org

Client-facing web app: The CoderLab tool itself. Users who navigate to the website hosting the CoderLab application will be able to use all of the supported features: a collaborative code editing text box, a fully functional Linux shell, and access to user-specific classroom materials (class coding demos, notes, etc).

Single sign-on: Refers to the technology managed by Iowa State which allows students to log onto a number of different services using their netID and password, credentials which are provided the University.

Virtual machine (VM): Virtualized server running on a physical host machine. The host is responsible for allocating CPU time, memory, and hard drive space for the virtual machines. The virtual machine typically has no knowledge of the host service; it behaves as its own distinct operating system.

LXC (LinuX Container): A kernel feature that allows running isolated linux systems on one host operating system. Makes use of Linux cgroups to isolate processes, i/o, network, and filesystems, CPU, and memory.

Docker: Program capable of creating images and managing (LXC) containers. Images represent the initial state of containers, such as shared libraries, installed packages, and other dependencies. With respect to object-oriented programming, images are to containers as classes are to instances. Reference: www.docker.com

Container: A running image instantiated by the Docker service. Containers provide separation from the host file system by using LXC (Linux Containers) which are more light-weight than a VM. In general, VMs may require hundreds of megabytes of memory per instance, whereas containers require tens of megabytes since they share similar kernel code. Reference: www.docker.com

Docker Room (Docker container): Abstraction from a Docker container as a server. This represents a container holding the processes and data needed by a room.

Room manager: Server which takes requests from the CoderLab web app and initiates, destroys, and services Docker containers accordingly.

Room: Abstract term for the editing session shared by a group of users. The room includes a shared editor, a virtual filesystem, a shared shell, and users active within it. Rooms have owners who manage other users' access to the room and editing permissions to its files.

4 GOALS

The CoderLab project will be composed of a user web application for code development as well as a server backend responsible for compiling code, maintaining files, and managing user sessions.

A completed project will include the following:

- Web-based code editor, including
  - Multiple editor tabs
  - Basic code completion (suggesting variable/function names)
  - Syntax highlighting
  - Code compilation support for C, Java, and MATLAB
  - Support for using Makefiles to perform multi-file compilation
- Collaborative aspect to the editor, including
  - Multiple users simultaneously editing the same code
  - Support for multiple displayed cursors
  - System for inviting others to collaborate on a project
- An integrated shell, capable of
  - Accepting input from a user
  - Relaying the input to a running program

    - Compiling and executing the code
    - Displaying output from the program
- A file system, allowing
    - Persistent code project storage
    - Directory structure
- Security, in the form of
    - User authentication through ISU single sign-on
    - Permissions system to restrict editing and collaboration
    - Preventing malicious code from being executed in the virtual shell

5 System Level Design

5.1 Overview

5.1.1 Modules

5.1.1.1 Authentication and Authorization (Auth)

Provides an interface to the ISU Single Sign-on authentication and access control. Handles the server-side ShareJS message filtering needed to restrict editing permissions of files within a room.

5.1.1.2 Room Manager

Spawns new Docker Rooms and manages their lifetimes and associations with users. When the Web Application wants to initiate a new classroom, it requests the Room Manager to start up a new Docker Room.

5.1.1.3 Docker Room

Docker image setup with a web service that accepts code from a classroom and streams a shell to the clients.

5.1.1.4 Web Server

Serves web page requests and acts as a connector for the other modules. Interacts with and serves the client-side files. Based on the client-server interactions, it will also interact with the room manager, sharejs, and docker rooms.

5.1.1.5 ShareJS Server

Receives and sends text operational data with connected clients. Provides a hook to process message data and keeps the editing components synchronized with one another.

5.1.1.6 ShareJS Client

Resolves text operations from concurrent users using operational transforms.

5.2 System Requirements

5.2.1 System-Level

- VM running Room Manager

- VM hosting ShareJS server
- Authorization module for ISU Single Sign-on integration
- Server to host and serve uploaded files

5.2.2 User-Level

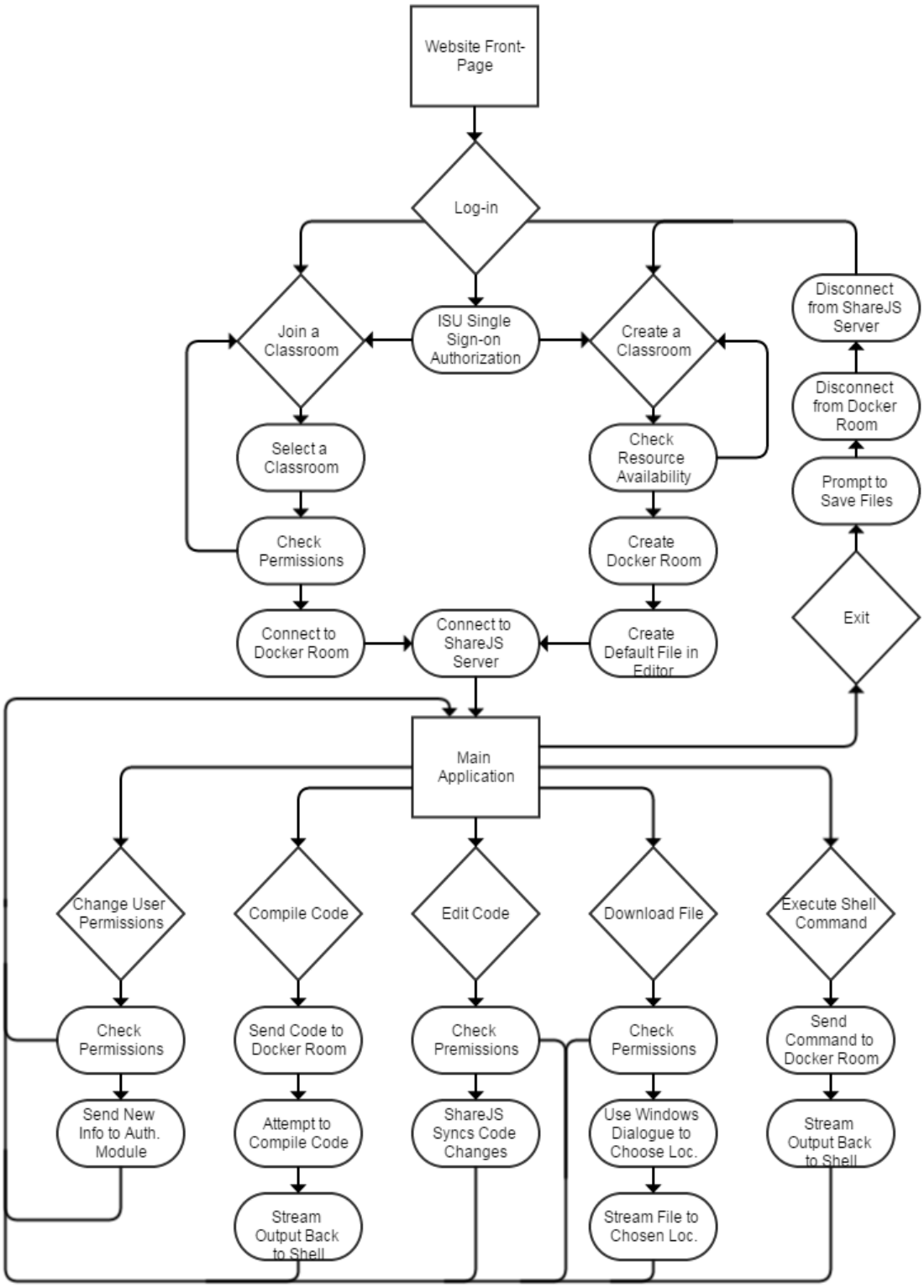
Required:

- An internet connection
- Any modern web browser (e.g., Chrome, IE, or Firefox)

Recommended:

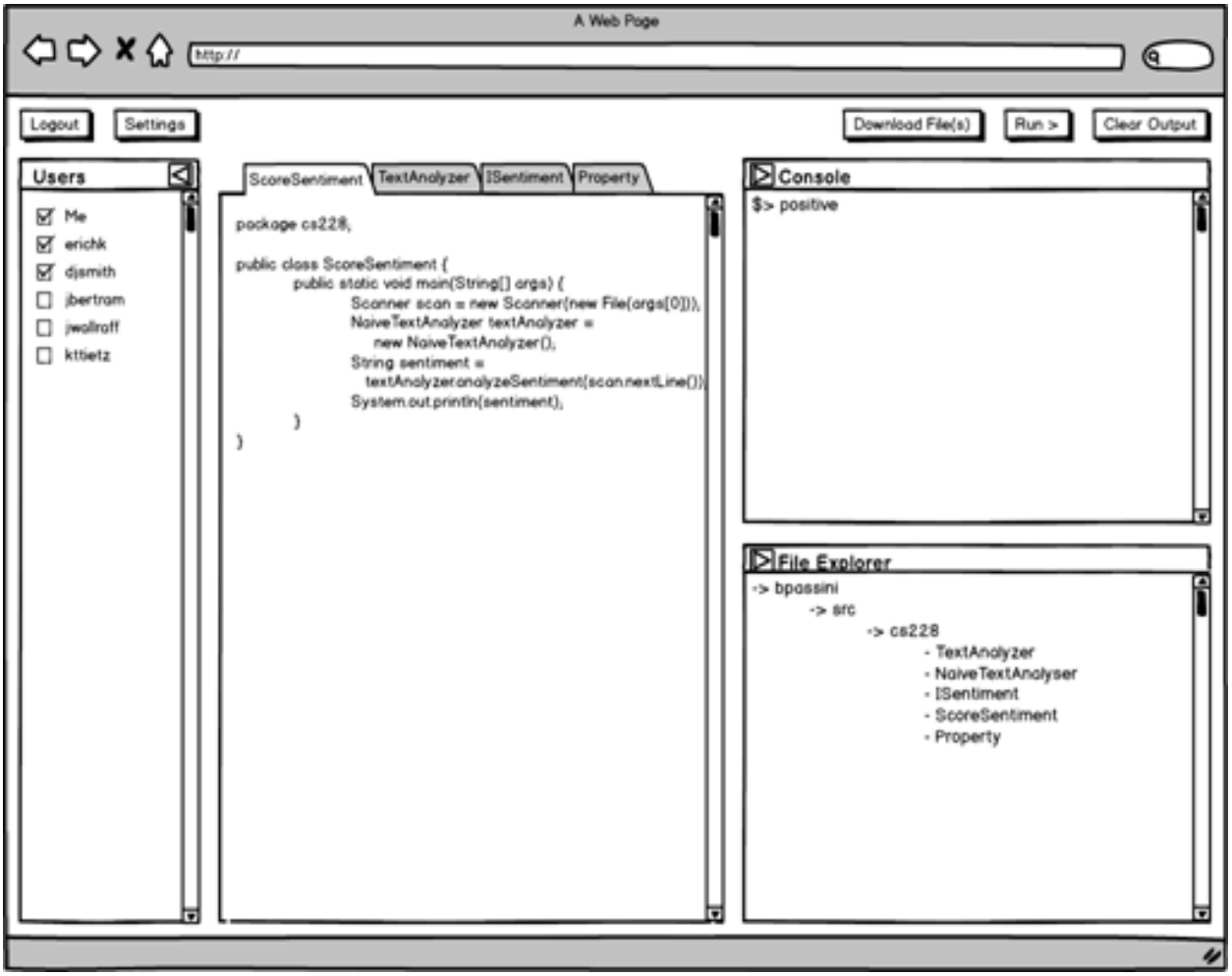
- For ISU Single Sign-on integration: an ISU NetID and password
- For editing permissions: an email address

5.3 FUNCTIONAL DECOMPOSITION



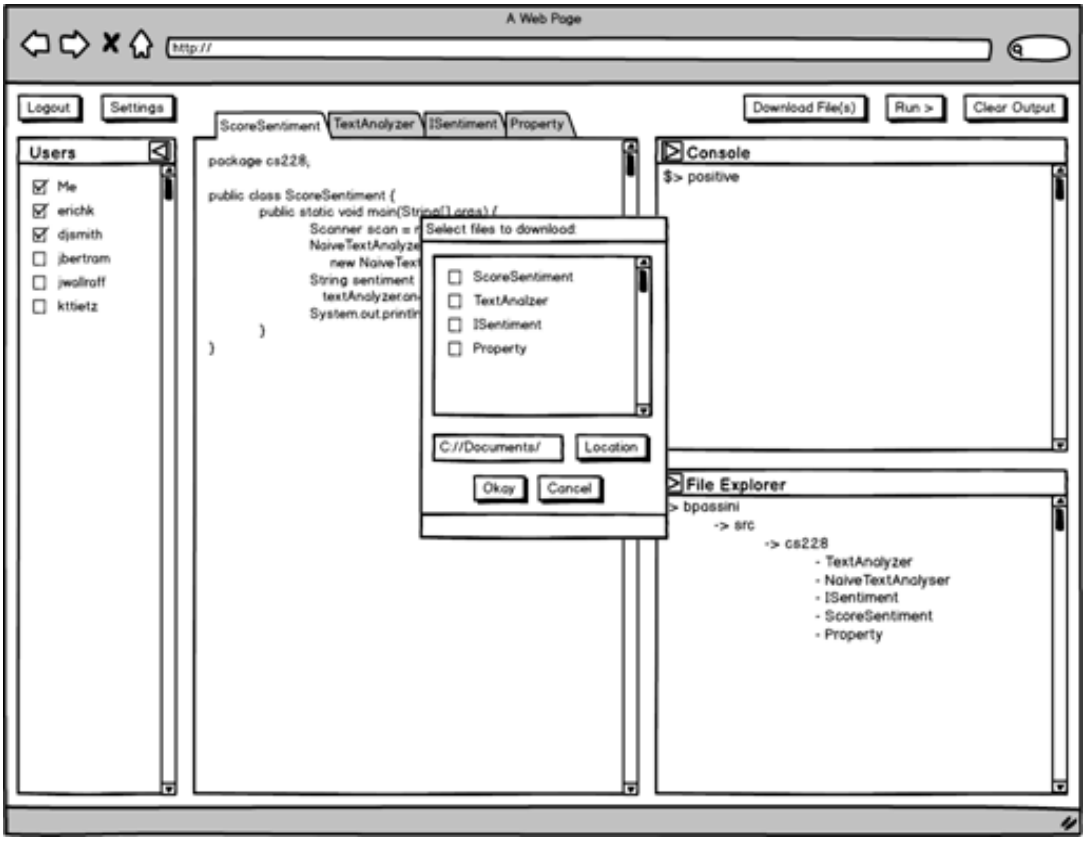
5.4 BLOCK DIAGRAMS

Instructor View



Instructor View with Minimized Windows

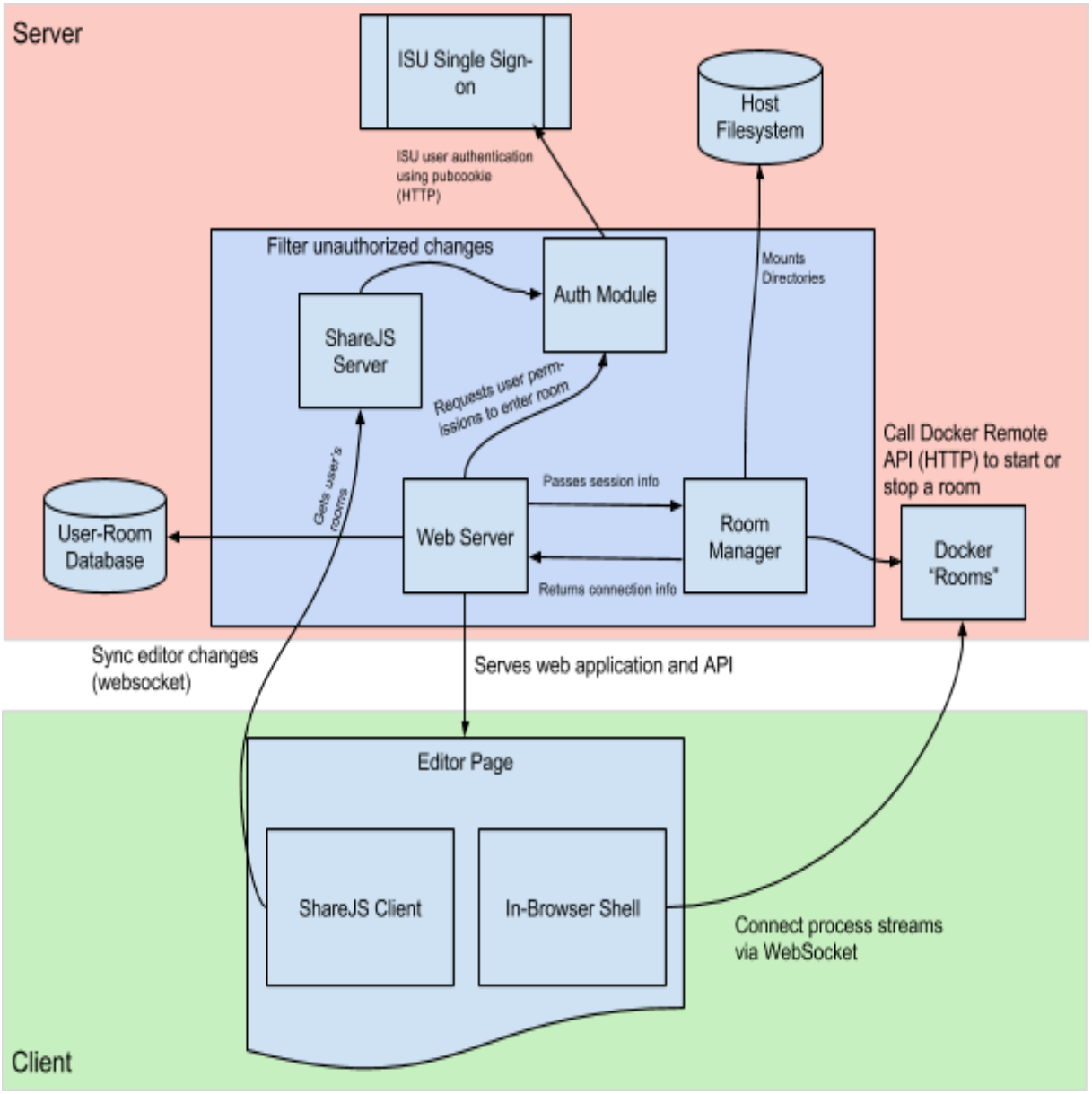
Instructor View with File Download Modal



6 DETAILED DESCRIPTIONS

6.1 I/O SPECIFICATIONS

6.1.1 Overview



6.1.2 Interface Specifications

Web Server API - Accepts requests and serves the static content of the web application. Also handles session information and user tracking.

- Input
  - Request to create a room
- Output

    - Room information
    - Request to store room information
    - Request for user permission to enter room
    - Request to room manager to create a room

Room Manager API - Oversees the creation and maintenance of docker rooms used by the web application.

- Input
    - Request from web server to start up or end a docker room
- Output
    - Room information back to web server
    - Request to mount host file system
    - stdout from compiled and executed src code

ShareJS Server API - Performs the operational transforms powering collaborative editing used in the web app.

- Input
    - Request to open a new document
    - Request to close an existing document
    - User-entered plain text from text editor
- Output
    - Connection to newly create document
    - Request status upon successful or unsuccessful closing of existing document
    - Updates made to existing document

User-Room Database API - Used to store and access from information

- Input
    - Request from web server to store room information
    - Request from web server to obtain room information
- Output
    - Return status message upon successful or unsuccessful store of room information.
    - Room information

Host File System API - Gives access to the host file system

- Input
    - Request to mount file system
    - Request to create, remove, or update a file
- Output
    - Return status message upon successful or unsuccessful mount of file system
    - Return status message upon successful or unsuccessful completion of file addition, creation, deletion or update

ISU Single Sign On API - Authenticates client using the ISU authentication system. Uses pub-cookie.

- Input
    - User login information
- Output
    - Status response message + redirect to the web server's content.

6.2 SOFTWARE/TECHNOLOGY SPECIFICATIONS

We use a number of different technologies that exist already so as not to reinvent the wheel with respect to those technologies. The combination of these technologies is what makes our project novel and unique.

Technology	Used In
Docker 1.2	Docker room manager
ShareJS 0.7	Collaborative code editor / server
TermJS	Shell rendering
PTYJS	Connects bash streams to Socket I/O streams
Socket I/O	Handles web socket streaming
NodeJS 10.x	Generic server platform

6.3 IMPLEMENTATION ISSUES

1) **Response time of operational transforms.** ShareJS has a robust system of operational transformations, but if a number of users with drastically different connection speeds are editing a document concurrently they might have problems editing the same code.

2) **Limited server space with which to store user projects.** Currently we have a VM with a static amount of storage space. Unless we were to implement a solution which has dynamically scaling server resources, continued use of the application would find us running out of server space eventually.

3) **Preventing users from running malicious code through the in-browser shell.** Providing shell access to a potentially malicious user can be dangerous. To prevent attacks or data mining on the host machine, we will use Docker containers to isolate users from the host filesystem and permission groups to prevent running unauthorized commands. Malicious code executed on a container may not affect the host system, and containers may be reset to their initial states from the host system if any abuse is detected.

4) **User trust.** When a user is given permissions to modify a document, they have full control, and if they were to break or erase a document, it would be unrecoverable unless we implemented a system to save many previous versions, which could be server-space prohibitive.

5) **Memory usage.** For a large number of classrooms or a large number of participants in a classroom, the VM could run out of memory. Fortunately, containers typically only require tens of megabytes (the current state of Docker Room container for example requires 30mb), as opposed to VMs which require hundreds. The simple solution to this problem is to acquire more RAM and to be proactive about terminating unused containers.

6.4 Testing Procedures

6.4.1 Verification

6.4.1.1 Static Testing

The entire procedure for static testing simply involves visual verification by team members. If possible, a member of the team who did not work on the feature or update will look at the code and give suggestions to improve it or reorganize it.

6.4.1.2 Integration Testing

As our project is very modular, it is relatively easy to develop a feature independently of the main file. Additionally, we host our project on a version-controlled repository so new features are developed in their own branches. The process for testing integration of new modules and features involves testing the feature in its own branch, fixing bugs, merging the branch to master, resolving any conflicts that arise and retesting the feature. At this point, further bugs will be identified and fixed in the feature's branch, and the process is repeated.

6.4.1.3 Stress Testing

Stress testing is more difficult to simulate without having an actual user base. It is possible to initiate a number of connections all from the same machine, but this won't simulate a real environment very well. That being said, as the project becomes more developed and more usable, we will stress test the system by initiating as many connections as we can from different devices and simulate a classroom environment with intense usage.

6.4.1.4 Security Testing

Security testing is important to maintain the integrity of the site. While we will make every effort to design a safe and secure system, it is our job to remind users that sensitive information is always vulnerable on the internet. The amount of extra development and testing time required to build and air-tight system is somewhat outside of the scope of our project.

6.4.2 Validation

6.4.2.1 Prototyping

We maintain a constant working copy of our project which is updated as more features are completed. This is important to validate that our ideas are working, whether they are ready for prime time or not. Having something to show others also gives us the opportunity to obtain valuable feedback.

6.4.2.2 Customer Testing

Enlisting testers who are not familiar with our project is vital to its usability. Our customer testing process will involve introducing users to our system and giving them a cursory overview of the system, and let them use it and provide feedback on the program flow and ease of use.

7 CONCLUSION

CoderLab, once complete, will combine technologies to provide an experience for students and teachers that has never before been realized in the way that we envision. Each service that we use is powerful on its own, but combining these seamlessly into a project that is useful and usable by students poses a challenge which, with the design specifications laid out in this document, we plan to meet.