May15-31

SENIOR DESIGN - CODERLAB

Final Report

Jake Bertram
Erich Kuerschner
Bryan Passini
Daniel Smith
Kyle Tietz
Jacob Wallraff
Advisor – Joe Zambreno

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 code development solution geared primarily for a classroom environment. A collaborative code editor in the browser will allow multiple users to develop code simultaneously and seamlessly. Users will be able to join sessions through Iowa State Single Sign-On and control edit access.

This level of interaction will allow CoderLab to create 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. This collaboration can help instructors identify common mistakes made by their students and can help improve their teaching methods. 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.

This project will be concerned with the development of the backend server environment and webpage frontend in order to create this solution. CoderLab will be designed and developed by the team with guidance from the advisor client.

3 GROUP MEMBERS & ROLES

Jake Bertram – Communications
Focused on the backend web application, specifically using Docker and the classroom environment for compiling and running code. Used prior knowledge of node.js and web development to help the team. As communications role, was responsible for reaching out to departments for resources as well as staying in contact with the adviser.

Erich Kuerschner – Webmaster
Front end developer. Focused on the file system interface which allows users to create, edit and share their files with other users in the current CoderLab session.

Bryan Passini – Communications
Focused on front end development related to the collaborative coding panel, including the Ace editor and the sharejs technology.

Daniel Smith – Key Concept Holder
Responsible for the in-browser terminal and its communication with the shell. Configured Jira for use as a project/ticket manager.

Kyle Tietz – Team Lead
Maintained long-term outlook on the project and kept team focused. Implemented client session management, including user permissions and server-client messaging.

Jacob Wallraff – Webmaster
Responsible for most of the application UI and program flow. Served as webmaster, updating the website with project documents and other details.

4 GLOSSARY OF TERMS

Apache Reverse Proxy: Sits in front of the web application, routing traffic to the web application. Also ensures users are authenticated with Single Sign-On and forwards information about a user to the application. The reverse proxy used in this project was the Apache httpd mod_proxy module.

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.

Classroom: 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.

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 functional Linux shell, and access to user-specific classroom materials (class coding demos, notes, etc).

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: 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

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.

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, visuals, and user input handling.

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 file systems, CPU, and memory.

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

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.

PTY.JS: Library used to fork shell processes for the in-browser terminal to connect to.

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

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

Single sign-on (SSO): 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.

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

TermJS: Javascript clone of the Xterm terminal emulator. Interprets shell data and generates the proper UI for the terminal.

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.

5 GOALS

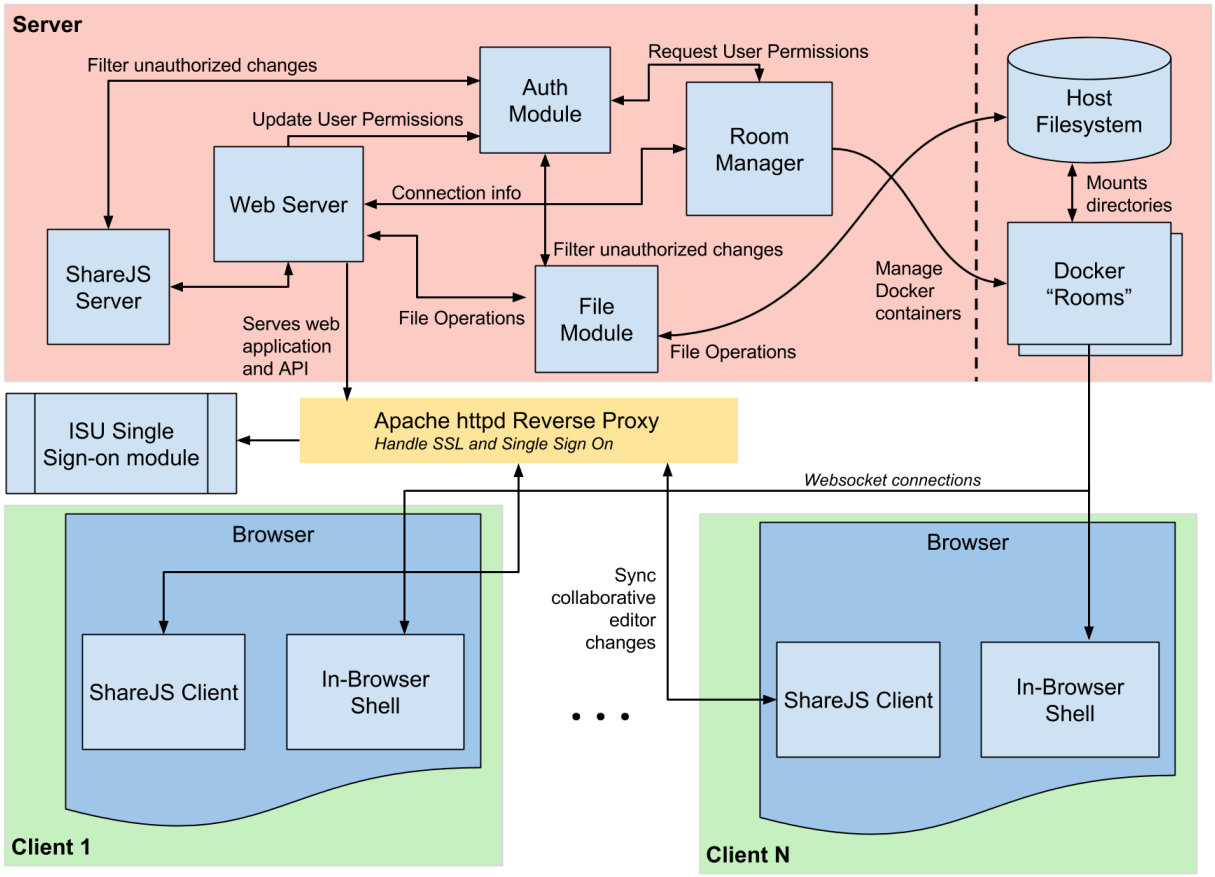
The CoderLab project will be composed of a user web application for code development as well as a server backend responsible for compiling code, maintaining files, and managing user sessions.

A completed project will include the following:

- A web-based code editor, including
    - Multiple editor tabs
    - Syntax highlighting
    - Code compilation support for C and Java
    - Support for using makefiles to perform multi-file compilation
- A collaborative aspect to the editor, including
    - Multiple users simultaneously editing the same code
    - System for discovering other users' classrooms
- 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 project code storage
    - Directory structure modifiable in the shell
- 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

6 System-Level Design

6.1 System Overview



6.2 Module Descriptions

Web Server - NodeJS server that handles main communication with client browser. The web server acts as a middle-man between the client and other modules. It passes messages across a socket in order to keep the client browsers synchronized with the server modules.

ShareJS Server - Manages live collaborative documents and passing file updates to all client code editors in real time. These documents are kept in sync through operational transforms. It maintains in-memory documents of the open files being edited by clients. These live documents can be saved to the filesystem.

Auth module - Maintains list of client connections and retrieves user permissions from the Room Manager. It filters unauthorized changes made to files and user permissions from the browser by checking against permissions received from the room manager. It ensures that all clients receive updates when permissions are changed.

Docker Classrooms - A Docker image configured to act as an isolated Linux environment for connected users. The image contains the web service that provides a collaborative terminal session and the ability

to compile and run code. Running containers stream a terminal session to the client-side, and also takes requests to compile and run the current file opened in the user interface's editor. As a linux environment with basic build tools installed, it can also let users manually run build tools on their own, such as make, gcc, or javac.

Room manager - Manages creation, deletion, startup, and shutdown of Docker classrooms upon requests from the web server. Contains classroom information, including the Docker container status, who created the classroom, and permissions of users currently in the classroom.

File module - Communicates with the host file system to create, load, and delete files in the classroom. It handles incoming client-side edits and syncs them with the file system. Communicates with the auth module to ensure the user has proper permissions to write to a file.

Host file system - Contains project files specific to each classroom. Classrooms mount these directories, allowing the files to sync between host and classroom.

ISU Single Sign-on - Apache httpd module that interacts with ISU's Single Sign-on (SSO) to ensure that users are authenticated.

Apache httpd Proxy - The Web Server sits behind a reverse proxy, which handles HTTPS and the Single Sign-On integration.

ShareJS client - Keeps files in the code editor in sync across edits from multiple clients. This module automatically communicates with the ShareJS module on the server end. Communicates with the auth module to ensure the user has proper permissions to write to a Share document.

In-browser shell - Communicates with a shell process on the classroom container to allow command execution and code compilation.

6.3 FUNCTIONAL REQUIREMENTS

6.3.1 Authentication Requirements

1. The product shall require users to log on using ISU's single sign-on system.
2. The product shall allow the user to log out.

6.3.2 Classroom Management Requirements

1. The product shall allow users to create classrooms.
2. When the user selects to create a classroom, the product shall create a new classroom on the server, assign the user as the classrooms owner and add the newly created classrooms to the list of existing classrooms.
3. While the user is logged in, the product shall display a list of all available class rooms to join.
4. When the user selects to join a classroom, the product shall add the user to the classroom and shall redirect the user to the classroom webpage.
5. While the user is an owner of a classroom, the product shall allow the user to stop that classroom.
6. When a user selects to stop a classroom, the product shall notify all users still in the classroom that the classroom has been stopped and disable future users from being able to join the classroom.

7. The product shall allow the user to leave the classroom.
8. When the user selects to leave the classroom, the product shall remove the user from the classroom list and redirect the user to the classroom page.
9. While the user is in a classroom, the product shall display a list of users, a collaborative coding environment, an interactive UNIX shell and a list of files belonging to the classroom.

6.3.3 File System Requirements

1. The product shall allow users to create new files.
2. When the user selects to create a new file, the product shall create a new file local to the user and shall not let other users view the file until the user selects to save the file.
3. When a user selects to save a newly created file, the product shall save the file to the file system and update the list of file for all users.
4. When a user makes a change to a saved file in the collaborative coding environment, the product shall make the change visible to all users viewing the file.
5. The product shall allow users to download files that belong to the user's classroom.
6. The product shall not allow a user to view files that belongs to classrooms other than the classroom the user is currently in.

6.3.4 Shell Requirements

1. When a user types in the interactive UNIX shell, the product shall make the change visible to all users viewing the UNIX shell.
2. While the user has shell permissions, the product shall allow the user to interact with the terminal.
3. While the user does not have shell permissions, the product shall allow the user to see shell interactions, but shall not allow the user to interact with the terminal.

6.3.5 Collaborative Editing Requirements

1. While the user has edit permissions, the product shall allow the user to edit documents in the collaborative coding environment.
2. While the user does not have edit permissions, the product shall allow the user to see updates made to documents in the text environment, but shall not allow the user to edit the documents.

6.3.6 Permissions Requirements

1. While the user has admin permissions, the product shall allow the user to change the permissions of other users.

6.4 NON-FUNCTIONAL REQUIREMENTS

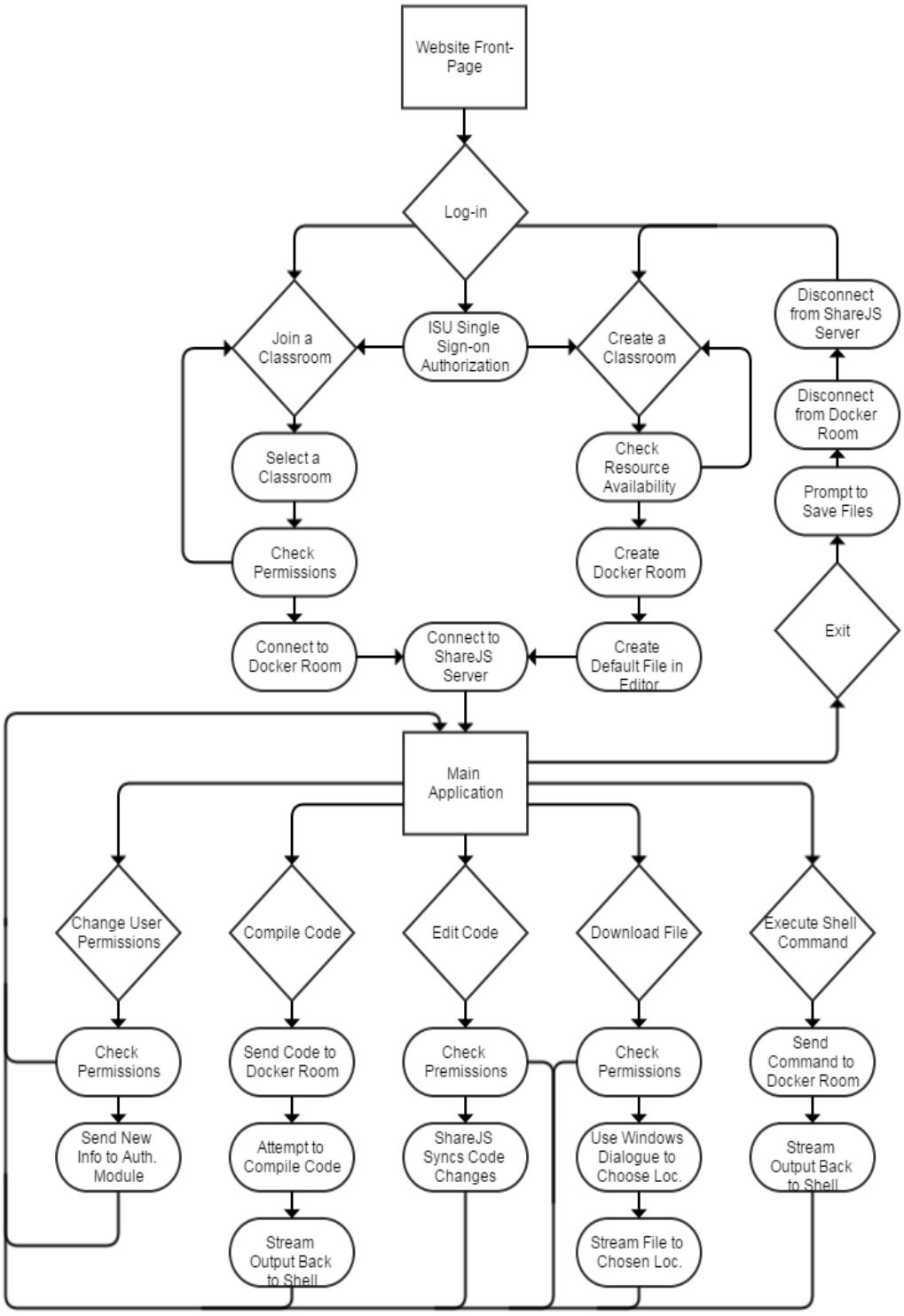
1. The system shall be reliable and fault tolerant.
    - Fit Criterion: The product shall have a downtime no greater than 95%.
    - Fit Criterion: In 95% of all classroom crashes, the system should still provide 95% of its functionality.
2. When a user creates a classroom, the system shall quickly respond with the newly created classroom.
    - Fit Criterion: 85% of all classrooms created shall be created in 5 seconds or less.
3. When a user selects to compile and execute code, the system shall be fast and responsive.

- Fit Criterion: For 85% of all code compilations, the time from when the user selects to compile to when the output appears in the terminal should be no greater than 5 seconds.

4. The product shall be easy to use.
    - Fit Criterion: 75% of all surveyed users shall rate the ease of use of the product with at least a 75% approval rating.
5. The website shall have a professional look and feel.
    - Fit Criterion: 75% of all surveyed users shall rate the look and feel of the product with at least a 75% approval rating.
6. The system shall be scalable.
    - Fit Criterion: The system shall meet at least 95% of all requirements for at least 30 simultaneously running classroom sessions with up to 20 users in each classroom.

6.5 FUNCTIONAL DECOMPOSITION

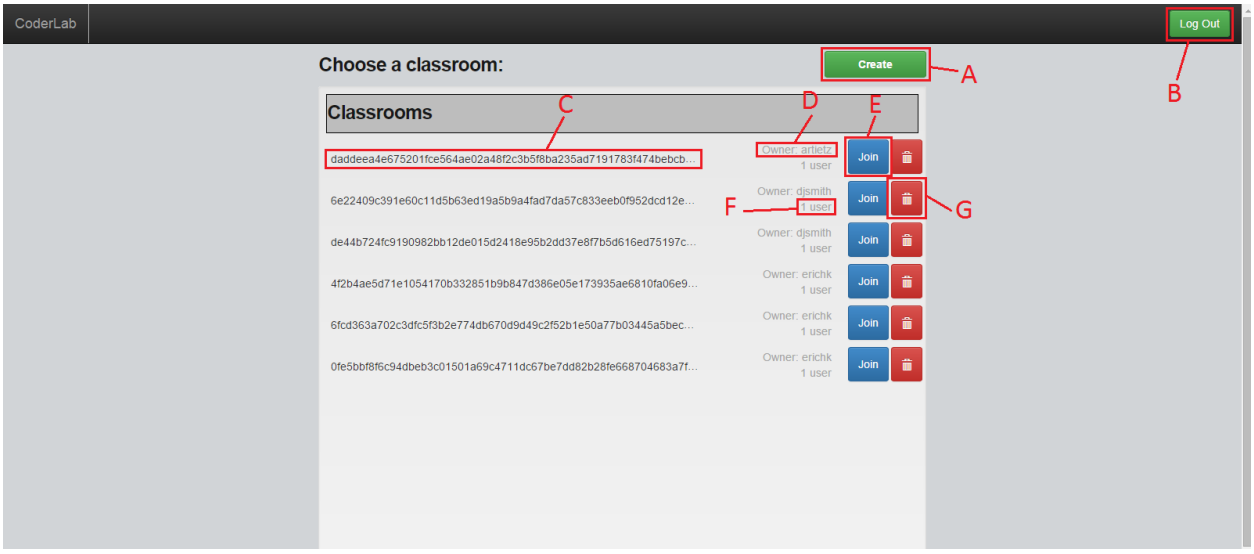
This image shows general program flow and operation sequences. It demonstrates how actions are connected to consequences and movement through the webpages.



6.6 UI IMAGES

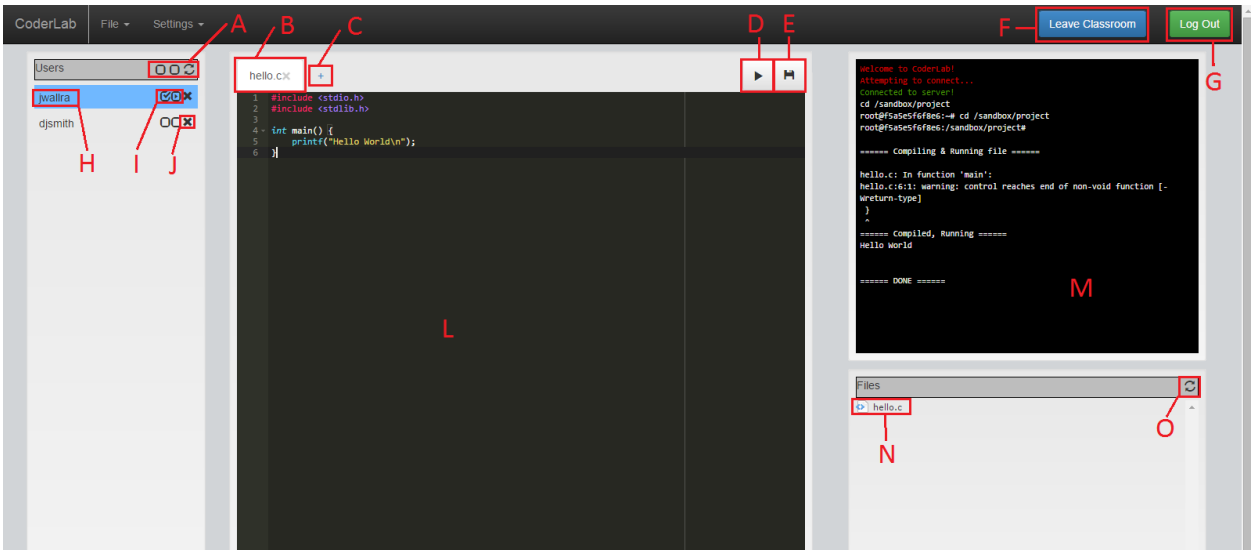
These are the pages in the current web application look like.

6.6.1 Classroom Selection Page



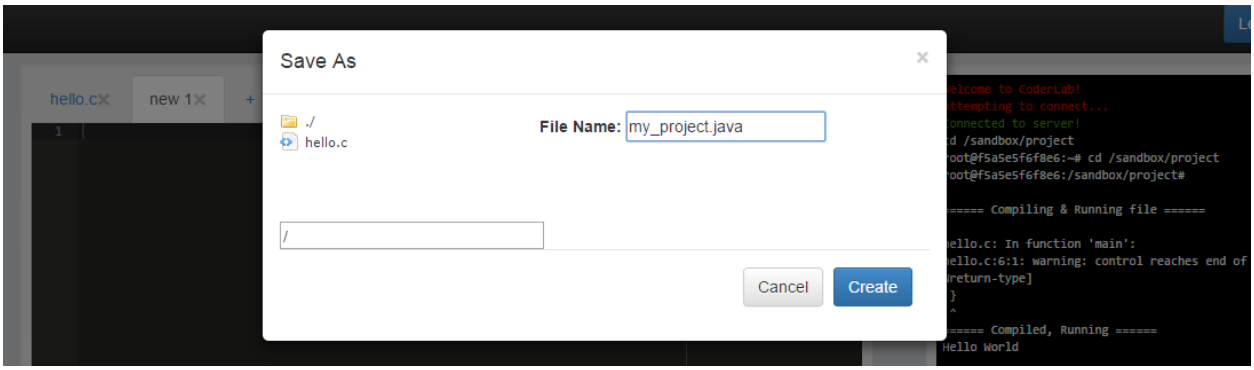
A – Classroom creation interface
B – Sign out of ISU Single Sign-On
C – Classroom name (hash code)
D – Username of classroom creator
E – Join a classroom
F – Number of users in a given classroom
G – Delete the classroom (only work for that classroom's creator)

6.6.2 Main Applcation Interface



A – Set permissions for every user (the box icons), or refresh the user list
B – A file tab showing the file name with a file close option
C – Add a new file
D – Compile and run the current file (C or Java)
E – Save the current file (also available through the file menu)
F – Leave the current classroom and return to the classroom selection page
G – Log out of ISU Single Sign-On
H – Username (your username is highlighted in blue)
I – Individual editor and shell user permissions buttons
J – Remove a user from the classroom
L – Collaborative editor
M – Collaborative shell
N – Files present in the current classroom
O – Refresh the file list

6.6.3 Save File Modal Dialog



The save file modal dialog has textboxes which show the name of the file the user is saving (top) and the file path of the same file (bottom). On the left-hand side the files in the current directory are shown and the user may navigate the file system.

7 IMPLEMENTATION AND TESTING DETAILS

7.1 SOFTWARE AND TECHNOLOGY

CoderLab is mainly an integration project; the team brought together many existing technologies to provide a web-based coding experience. Much of the work involved leveraging these existing programs and libraries to provide features to the user. Listed below are the main technologies used in the project:

Technology	Role
Docker 1.2	Manages container environments for classrooms, providing isolation and compilation tools
Ace Editor	Open-source embeddable Javascript code editor that is the center of the main development page
ShareJS 0.7	Enables real-time collaboration for the code editor
TermJS	Renders the shell in the browser
PTYJS	Connects shell to Socket I/O streams that talk with the Docker classrooms
Socket I/O	Handles websocket streaming for shell
BrowserChannel	Allows websocket message passing to synchronize the client browsers with the server
NodeJS 10.x	Generic server platform
Bootstrap 3.3	Web app styling and menu dialogs
JQuery FileTree	Displays list of files in the browser

7.2 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 potentially malicious users can be dangerous. To prevent attacks or data mining on the host machine, we will use Docker containers to isolate users from the host file system 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.

7.3 Testing Procedures

7.3.1 Verification

7.3.1.1 Static Testing

The 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.

7.3.1.2 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 the 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 can be identified and resolved in the feature's branch, and the process is repeated.

7.3.1.3 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 does not simulate the typical environment very well. To help simulate a larger user base our team gathered several times throughout the development to run the project with everyone logged in and running commands. The goal of these meetings was to push the software to its limits and identify any stress related issues that may be present in the system. While our team was a considerably smaller user base than the intended user base (6 vs 20-30 students in a classroom), these 'mini stress tests' still provided us some useful feedback with regards to how our features scaled to multiple users. After the development was complete, we brought

CoderLab to a ComS 228 recitation to try using it in the classroom setting it is intended for. With this live demo we reached a max of about 15 concurrent users. This experiment was deemed an overall success after observing the CoderLab server keep up with the test user base.

7.3.2 Validation

7.3.2.1 Prototyping

We maintained a constant working copy of our project which was updated as more features were completed. This was important to validate that our ideas were working, whether they were ready for prime time or not. Having something to show our client/advisor and potential users gave us ample opportunities to obtain valuable feedback.

7.3.2.2 Consumer Testing

We enlisted several 'non-technical' friends and classmates to try their hands at CoderLab. This ended up being very useful for uncovering user interface related issues and bugs. Each user was observed as they played around with the software. Notes were taken down as the user encountered problems or became confused and needed guidance through a particular process. This helped us shape CoderLab's user interface into a more comfortable and intuitive experience.

As mentioned above, we brought CoderLab to a ComS 228 recitation to be used by real students. The students' demo left them satisfied and genuinely excited about the potential for a tool like CoderLab at their disposal. We obtained some valuable feedback regarding the features the students would like improved and would like to see in later versions of the product.

8 CONCLUSION

CoderLab combines a multitude of powerful web technologies to deliver an experience that reinvents the way students and instructors interact in programming courses. CoderLab provides a complete end-to-end browser development experience and could be used in classrooms today. The application features a collaborative editor and shell, isolating sets of these into classrooms. These utilities provide a set of developer tools that enables code development, compilation, and execution all in one place. With the exception of a few discarded features we ultimately ended up with the software we set out to build seven months ago. This project represents the culmination of our educational experience at Iowa State University.

Appendix I Operational Manual

Here you will find the basics of installing and using the application from a systems administrator perspective, as well as a brief description of how an end user would interact with the system.

1 Minimum System Requirements

1.1 Required Server Specs

These are loose requirements that should provide adequate performance. The specific hardware and OS versions mentioned reflect what was used for development, and will be the most likely setup that works.

- **Redhat Enterprise Linux (RHEL).** Note that EL6 was the version used, although similar systems, such as RHEL7, Fedora, or CentOS 7 may also work. The VM must also run a modern kernel that supports Docker. For best results, 3.1 or above is fine, but RHEL also provides 2.6.32-504.8.1.el6, which will support docker. For specific constraints, see the docker documentation for supported kernels and operating systems.
- **Sufficient RAM.** At least 1.5GB of RAM must be available, along with the same amount of swap. This will be sufficient for a large number of classrooms.
- **CPU** should be at least a dual-core, clocked at least to 2.4GHz. The system was tested on a virtual machine with an Intel Xeon CPU e5-2697 clocked at 2.70GHz.
- **Storage Space** recommended is 35GB for all required components.
- **SSL Certificates** will be needed, because shibboleth requires websites to use HTTPS.

1.2 Required Software Specs

Additionally, several packages must be installed and setup on the system. Specific setup instructions are beyond the scope of this document, but any special considerations will be mentioned.

- **node.js and npm** are required for the web application to run. They may be installed via yum install nodejs or through the node version manager, nvm.
    - The application is built with npm and grunt. To install grunt, npm install -g grunt-cli
- **git** is required to pull the source and resolve certain dependencies.
- **Docker** must be installed, since this is what provides the application with isolated container environments. Docker may be installed from the docker-io package via yum install docker-io.
- **httpd (v2.2)**, aka the Apache httpd web server, is used as a reverse proxy as well as a way to integrate with Iowa State's Single Sign-On service (known as shibboleth)
    - mod_proxy and mod_shibb are two required httpd modules
    - Iowa State University provides documentation for how shibboleth may be set up.
- **shibboleth** is Iowa State's current Single Sign-On implementation. The server must be set up with the shibboleth daemon, shibd. Iowa State's ITS department provides support for the setup of shibboleth with httpd 2.2. Consult student IT handbook for more information.

2 INSTALLATION

With the above completed, the application source must be cloned and built.

1. **Get the source** from GitLab.

```
git clone https://git.ece.iastate.edu/coderlab/sharejs-project.git
# OR (requires ssh key setup with gitlab)
git clone git@git.ece.iastate.edu:coderlab/sharejs-project.git
```

   It is recommended to use ssh keys, and the project may require it when pulling in the dependencies from gitlab. See git.ece.iastate.edu/help/ssh/README.md for more information

2. **Use npm** to fetch the project dependencies. With this newly downloaded project, cd into the folder and run the following

```
npm install
```

   This will pull in third-party dependencies as well as a couple dependencies from gitlab.

3. Use **grunt** to build the client-side files

```
grunt
```

Grunt will then assemble the client-side files so that they may be used by a browser.

3 DOCKER IMAGE INSTALLATION

In order to start creating and using rooms, the Docker image for the "classroom" server needs to be built.

1. **Get the source** from GitLab.

```
git clone https://git.ece.iastate.edu/coderlab/coderlab-
classroom.git
```

```
# OR (requires ssh key setup with gitlab)
git clone git@git.ece.iastate.edu:coderlab/coderlab-classroom.git
```

2. **Build the base image** using its build script. From the git project's root directory:

```
cd docker-base
./build.sh
```

This builds the base Ubuntu image with basic dependencies, directories, and environment variables set up.

3. **Build the classroom image** using its build script. From the git project's root directory:

```
cd docker-classroom
./build.sh
```

This packages up the bin/www server, injects the code into the new image, installs node dependencies, and tells Docker to create a port mapping for the web server.

4. Check that the images are properly built. Run docker images to see the images built. You should see something like this:

```
$ docker images
 REPOSITORY          TAG         IMAGE ID            CREATED              VIRTUAL SIZE
 coderlab/server     0.4.2       <image id>          About a minute ago   757.2 MB
 coderlab/base       0.1.0       <image id>          About a minute ago   737.1 MB
```

Once these images are built, the CoderLab application can use them to create containers for rooms.

The project should now be ready to launch.

4 Starting the Application From an Administrator Perspective

With the above set up, starting the application requires just a few configuration options to be set as environment variables. Here is a full example of starting the server (must be root):

```
DOCKER_HOST=http://127.0.0.1 PORT_START=13000 DEBUG="app*,cl*" PORT=2000 node lib/index.jS
```

The environment variables control the following settings:

- DOCKER_HOST is where the Docker Remote API has been set up to listen. Depending on the docker setup, change this accordingly. The Docker Remote Api needs to be running on the same server, in order for the classroom to sync files with the application.
- PORT_START defines what port the new classroom servers should attempt to listen on. When the port specified cannot be listened on, the application attempts to listen on the next available port, so choose a high number that won't conflict with other services.
- DEBUG helps setup additional debugging information. Recommended value to see output from our application is "app*,classroom*". Note that this is only for debugging purposes.
- PORT is the port that the web application should be listening on. It should be set to the port that the reverse proxy is sending traffic to.
- and lastly, node lib/index.js is just the startup command.

If the reverse proxy routes traffic from :443 to :8080, the minimal command to start up the server (as root) would be:

```
PORT=8080 node lib/index.js
```

You should now be able to access the application from wherever the reverse proxy is listening.

5 USAGE

This section describes how a user should use the system. It does not describe how a user will interact with the system.

5.1 ACCESS THE HOMEPAGE

The main entrypoint to the application is the front page, currently accessible from coderlab.ece.iastate.edu. From here, a user can see the current list of rooms, and attempt to join them.

5.2 STARTING A CLASSROOM SESSION

From the homepage, a user can start up a classroom session via the Start Classroom button, which will produce a dialog. Upon submitting the dialog, a classroom will be started. This typically will take a few seconds, due to the time involved with starting a docker container.

5.3 JOINING A CLASSROOM SESSION

To join a classroom, a classroom owner may redistribute the unique URL of a classroom session to others. Alternatively, users may search existing classrooms from the homepage and pick out the one they would like to enter.

5.4 EDITING A DOCUMENT

By default, users do not have permissions to edit documents. Instead, the owner of a room must explicitly grant permissions to the users. Once this is done, the document is collaborative, meaning multiple users may be editing a file at the same time, and changes will be propagated to each other in real time.

5.5 USING THE TERMINAL

Like with editing a document, users must be granted permissions to use the terminal. Once this is done, it too is collaborative, so use with some caution. The terminal runs in an isolated environment, and is a largely functional bash shell.

5.6 TROUBLESHOOTING PROBLEMS

In general, a first step in diagnosing problems is to refresh the page. The few issues that rarely arise can be fixed by doing this.

Appendix II Alternative Design Considerations

1 Exploring Similar Existing Products

We found several web-based code development tools that already exist: CoderPad, FirePad, EtherPad, Cloud9, CodeBunk and IDEOne. In exploring them, we identified characteristics that we did and did not want to include in development of CoderLab. CoderPad provides a real time code execution window and a user list window, both features that were consistent with our design plans. Cloud9 provides a full blown IDE in the browser, which was too much for our design; we did not want to overwhelm new student users with a complex IDE. While some of these tools are very similar to what we envisioned CoderLab to be, each of them lacked one important item: integration with ISU. We wanted our product to be tailored specifically for ISU students and faculty in order to promote classroom learning.

2 Code Completion

We had initially planned to include code completion for all of our supported languages. This would have improved coding efficiency and given the editor the feel of a real IDE. However, as these features weren't supported natively by the ace editor, it was difficult to implement. Additionally, creating a smart code completion engine from scratch could be a project in and of itself. As such, we decided for practicality reasons and time constraints that code completion was not an essential feature of the application. One upside to this is that it forces students to remember the libraries that they have to use, and more accurately represents the situation they will be in when they would take a test.

3 MATLAB

One of the original programming languages slated for development and compilation support in CoderLab was MATLAB. This language is used in several engineering disciplines at Iowa State and it thus was an attractive option for support. However, two major challenges prevented us from including it in our development. First was the issue of licensing. MATLAB is a proprietary product developed by MathWorks and would have proved a challenge in exploring how it could be used in the context of CoderLab. The other issue was how tightly the usefulness of MATLAB was tied to its IDE. Much of the power of MATLAB involves the IDE interpreter, the variable list, and graphing utilities, which would not be available in CoderLab.

4 Other Editors

When we started designing CoderLab, we had the choice between using Ace or Coder Mirror as our code editor. Both editors provide a wealth of features, including syntax highlighting, bracket

matching, various themes, etc. In the end however, the decision to use Ace over Code Mirror, came from the ability to easily integrate Ace with Sharejs. Sharejs provided built in support for Ace and did not provide this support for Code Mirror. Since a significant portion of CoderLab depends on collaboration, it was clear that we needed to choose the editor that would best support collaboration.