

SSH CONNECT

AN WEB CLIENT FOR SSH SERVERS

SPENCER HODGINS & TRI CAO

GROUP NUMBER: 58

Contents

1. Introduction	3
1.1 Context	3
1.2 Problem Statement	3
1.3 Results	4
1.4 Outline	5
2. Background Information	6
2.1 Secure Shell (SSH) And SSH File Transfer Protocol (SFTP):	6
2.2 RESTful Web Service and HTTP:	7
2.3 Deploy web service with Tomcat Server:	8
2.4 Front-end web application platform and Angular:	8
3. Methodology:	8
3.1 How we built the front-end:	8
3.2 How we built the back-end:	9
4. Results	10
4.1 The big picture:	10
4.2 The Front-end Angular application:	11
4.3 The back-end:	12
5. Evaluation and Quality Assurance	13
5.1 Student User Survey	13
5.2 Survey Result Summary:	13
5.3 Self Evaluation	15
6. Conclusion	16
6.1 Summary:	16
6.2 Relevance:	16
6.3 Future Work:	16
Contributions of Team Members	17
References	17
Final Word	18
Appendix 1: A Guide for installing and running SSH Connect	19
1. Install Tomcat server:	19
2. Open Tomcat manager and run RESTful web service:	19
3. Install nodeJS and Angular CLI:	20

Appendix 2: SSH Connect screenshots	21
Appendix 3: Table of Acronyms	25
Appendix 4: Survey questions	26
Appendix 5: Time testing charts	27

Title: SSH CONNECT

Date : April 13, 2018

Authors : Tri Cao, 100971065, tricao@cmail.carleton.ca

Spencer Hodgins, 101009135, spencerhodgins@cmail.carleton.ca

Group Number: 58

1. Introduction

1.1 Context

SSH Connect is a browser-based file manager accessing remote Secure Shell (SSH) servers. The inspiration comes from the need of a Graphical User Interface (GUI) for students to work with the Minix Virtual Machine (VM) on their exercises at the beginning of the course COMP3000, as the provided Minix machine supports only a command line environment. Noticing that Minix VM also provides an SSH server to be accessed in the host machine, we can exploit the server and use SSH File Transfer Protocol (SFTP) to create a file manager that is easy to use and ease the frustration when working with the VM. Furthermore, once we can connect with the Minix VM, we are able to connect to any SSH server with the right address, port, and user credentials.

The web browser is the most used application at the moment, and it is very great if our product could use its power to provide an easy to use working environment for users. That is the reason why we wanted to build the GUI as a web application.

1.2 Problem Statement

The biggest goal is to build a Web Application that allows users to connect to any remote SSH server. Therefore it is obvious to see the two biggest problems:

a. User Interface problem (front-end problem):

Almost every computer user uses file explorer program every day, so they are familiar with navigating around directories, open and edit files on the machine. Thus we need to make a user interface (UI) that supports these very basic features of a file manager and it is also necessary to have a file editor to read and modify remote files. The components

of the application need to provide information about files like name, type, size, permissions, etc.

The software needs not only to be easy to use, but it needs to be laid out nicely in order for users could feel comfortable. So our job is also to deploy the most modern styling practice to make our app not only functional but also a beautiful piece of software.

b. Connecting with SSH server problem (back-end problem):

All of the features on the front-end couldn't be done without connecting to the host machine via the SSH server. We need to find a way to transmit the files and directories from the server and to display them in the front-end, as well as make changes of the files currently on the VM when the user saves from the GUI. We also need to maintain a connection throughout the entirety of the application, as well as be able to transmit useful data if the connection is closed especially when the host machine is turned off, and it should be able to reconnect to the server when it is active again. The speed of the app depends on how good the connection design is implemented, and what tools we need to establish the connection.

1.3 Results

Although more features would still need to be implemented in order to use this application, we were able to successfully establish the connection with the SSH server and our front-end running on the browser to provide to user basic operations on files and directories grabbed from the server with a nice and easy to use user interface. A brief summary of the features.

- a. **Front-end:** Our front-end is an **Angular 5** (a JavaScript front-end framework) application, using **Bootstrap 4** and **Angular Material** for styling. It sends HTTP request to our

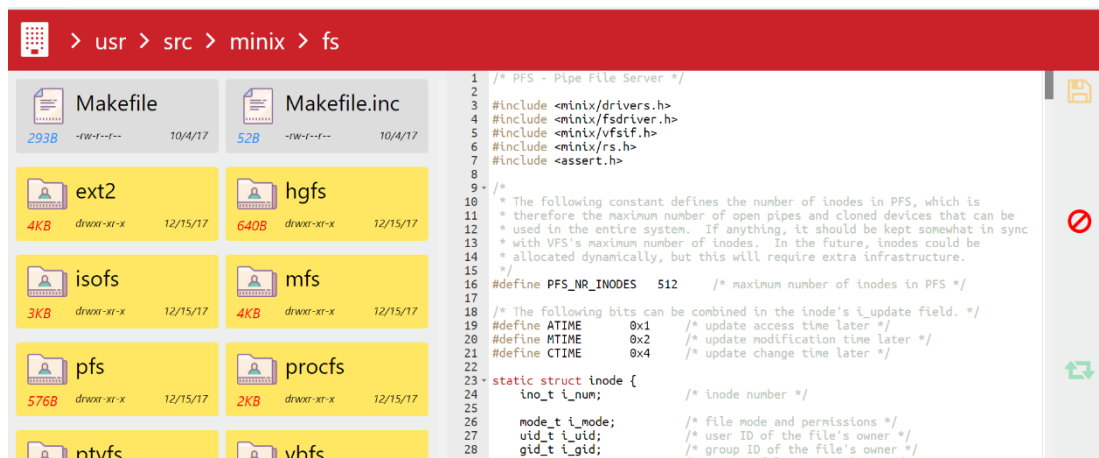


Figure 1 The main work console of the front end (screenshot)

application server (mentioned in the back-end section) to fetch and work with remote files. We are able to navigate through the file system of the remote machine, open and modify existing files. The front-end also has a login page to receive information to establish the connection.

- b. **Back-end:** The back-end is a Web service that conforms to the **Representational State Transfer (REST)**. It works as a bridge from our front-end application to the SSH server. We develop the back-end with java with the Jersey framework to create REST service and use **Java Secure Channel (JSCH)** library which is responsible to connect with the remote SSH server. First, the server receives **HTTP (Hypertext Transfer Protocol)** requests from the front-end app then interprets them and uses the SFTP channel provided by JSCH library to make commands to the host machine. With the SSH File Transfer Protocol (SFTP) channel being maintained, we can use it to download files from SSH server to display them in our front-end application, as well as send requested to modify files to the host machine.

1.4 Outline

The rest of this report includes these sections:

- Section 2 provides concepts and previous works related to our project. This includes details about the descriptions of how SSH and SFTP to get files from host machines, as well as details of all framework we used from front-end to back-end.
- Section 3 describes how we developed front-end, back-end, the code structure of our project
- Section 4 provides all our result that we achieve, what user can do with our project
- Section 5 describes how our result is evaluated. As it is a user application, we conduct a usability study involving fellow students.
- Section 6 summaries our achievements, and mentions our future goals for continuing developing this application.
- The rest of the report mentions contributions of team members in the project, references we used to write the report and all extra information you may want to read in the appendixes section

2. Background Information

2.1 Secure Shell (SSH) And SSH File Transfer Protocol (SFTP):

2.1.1 Definitions:

Secure Shell or SSH is a network protocol that supports a secure way to use network services over an unsecured network(1). One of its most popular use is for remote login to computer systems for their users. In the course COMP 3000, students can remotely access the virtual machines which run Minix or Linux Operating system using SSH port forwarding. Especially with Minix, which only supports only one single command line interface which is very limited in features, using SSH server is a solution to get multiple terminals working at the same time. It is very secure for the users to access remote system need to provide their credentials along with the SSH address (including IP address and port number) to be able to get in(1). There are GUI based clients for SSH server such as WinSCP which is a Windows app, but we didn't see any of it runs on the web, that's why we want to create an SSH client that runs on the browser.

We wanted to create a file manager with an editor to modify files, thus we needed a way to use the SSH channel to download the file from the host system. Luckily, we can use SSH File Transfer Protocol (also Secure File Transfer Protocol, or SFTP) to access, transfer and manage files and directories of the host system. It is based on SSH, so it is very secure. It also supports some extra capabilities that other protocols such as SCP (Secure copy protocol) and FTP (File Transfer Protocol) do not support such as interrupted transfers, directory listings, and remote file removal, which make SFTP looks like a remote file system protocol(2). With the power of SFTP, we hope that we could build a client that supports almost every feature of a file manager and fully control remote file system.

2.1.2 JSCH library:

As Java is the programming language of our choice in the back-end, with limited time, it is better for us to find some open source Java library that provides API (Application Programming Interface) to connect with the remote computer system using SSH. And JSCH is such library. Created by Jcraft team, JSCH is a pure Java implementation of SSH2 and especially it supports creating SFTP channel to transfer files and directories information from the SSH server(3). With JSCH, we have our most crucial key to enable our app.

2.2 RESTful Web Service and HTTP:

2.2.1 Definition:

So far we are able to connect to SSH server using JSCH, but we need to find some way to send the files we get to our browser-based GUI. Therefore, the easiest way to send such information is to use Hypertext Transfer Protocol (HTTP) for this kind of communication. HTTP is an application protocol for transferring data from client and server in any modern web application. For example in our app, when a login button is clicked, our front-end application prepare an HTTP request including user information to be ready to be sent back to the server. The server will then receives the data to work with SSH server and prepares back an HTTP response to send back to our front-end. If the information we provide is right, our front-end app will receive a successful response and navigate to the main console of our program.

But how we can have HTTP request send to our server, and how could the server know it is a request for list all files in a directory or it is a request for saving a modified file? For this purpose, we decide to follow Representational State Transfer (REST) style to create a RESTful web service. We can determine the purpose of any request based on two things: its HTTP method and an URL(4). For example: GET requests tend to be used to fetch data, while PUTs are used to update data on the server. The URL, possibly the request headers and body will describe what object we want to work with. For example in our application:

GET: <http://localhost:8080/minix-web-service/webapi/files/usr/src/minix/servers/pm>

The above request is to fetch all information from the directory with the path “/usr/src/minix/servers/pm” in the host machine. We followed the convention for a RESTful URL, a get request is used to fetch data and the information about what to fetch is represented inside the URL, making this very clear for users as they know what they should receive in their responses.

2.2.2 Java API for RESTful Web Services (JAX-RS) and Jersey framework:

JAX-RS is a Java programming language API spec that provides support creating RESTful web services(5). And Jersey is a framework that implements JAX-RS API convention to help developers making RESTful web services with Java(6). Learning the framework helps us draw a clear path on how to make our communication between front-end and back-end via HTTP requests available, and we are now able to connect parts of our project together at a unique software structure (see figure 2).

2.3 Deploy web service with Tomcat Server:

We also need a server to host our RESTful web service. Apache Tomcat (or Tomcat Server) is a good choice to deploy Java web application. Tomcat is an open-source Java Servlet container developed by the Apache Software Foundation(7). We run and debug our web service with a local Tomcat server, to access our service, the user will need to begin any URL with:

<http://localhost:8080/minix-web-service/...>

2.4 Front-end web application platform and Angular:

2.4.1 Definition:

Front-end web development is the creating of web pages to display for users in the browser. Traditionally, it is the combination of pure HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets) and JavaScript(8). The modern web page also uses those tools, but they are packaged with more support and modular structure, with the appearance of Front-end Development Framework. It becomes much easier to create a beautiful and modern web application with unlimited functionality and styles with the modern frameworks. We can even easily import pieces of previous work from other developers into our website, make it even easier to have a fancy browser-based application.

2.4.2 Angular and open source libraries:

Angular is a TypeScript-based open-source front-end web application platform developed by Google(9). Our front-end part of our application is a pure Angular application. We exploit the power of Angular to create a dynamic and fast web application and use pre-designed Angular components provided by open-source projects and libraries in our app. It is the key to make our app looks modern and attract users with a clean and nice graphical user interface.

3. Methodology:

3.1 How we built the front-end:

Angular JS is the front-end application platform of choice because it supports great tools for developers to have a modular, scalable codebase and of course a big community of developers who continuously make it even better and more beautiful. With angular, we developed our front-end components by using other smaller components. Each component has its own HTML template and style and all we need to do when we are done with all components is gather them into groups and display the view in the browser. Each component can also have its logic, for example in the login form component, each input field stores their value that user types in to send it later to our server. Here is the list of all components of our project:

- Login Page Component: everything included in the login form is declared within this component.
- Dialog Component: This component is used for displaying pops-up error and information dialog
- Work Console Component: this is all inside the main working console of the app, it contains smaller components:
 - Navigation Component: displayed at the top of the screen, helps users know where are they in the remote system and navigate to their parent directories up to the root directory
 - Entry List Component: The left part of the work-console, it lists files and directories of the current working folder, each entry has click listen to open file or go into another directory, as well as information about the file it is displaying.
 - File Content Component: The right part of the work console, it houses the editor to display and modify the content of files when opened and a tool-bar included 3 buttons to save, clear and reload the opened file. It also has a footer to show more information about the absolute path of the file, and whether if the file is editable or not.

Aside from components, we also make use of service to send HTTP requests and listen to responses. Most of the logic to connect with the java web service is inside the file “ssh-client.service.ts”. Every component then executes its logic actually sends information to the SSHClientService, the service will create HTTP requests based on the information and send to the server. The service is also responsible for creating event sources, thus when there are updates on the state of the app, it can inform the components to display the correct information.

3.2 How we built the back-end:

3.2.1 The SFTP service:

Our back-end receives the HTTP request from the front-end with the RESTful API, we then need to execute some logic and use JSCH library to make a connection with the remote SSH server. All of the logic will be inside SFTP service. We use a singleton design pattern here. Since once we connect to the SSH, we don’t want to reconnect again for every single request but just used the opened connection, this way we can improve the speed our application significantly. To maintain the connection, we need to use only one single instance of SFTPService class throughout the entirety of our applications connection. Thus the speed of our application will be faster compared to creating a new connection every request.

Once a user makes a login request, a new SSH session, and SFTP channel will be created, after that this single channel will be used to handle the calls to fetch files and directories, as well as modifying requests for files on the server. It is a two-way communication. On error, we catch SFTP exception thrown by the SFTP channel and produce error response back to the front-end.

3.2.2 Building the RESTful API:

Once we have a working SFTP service object that helps us to connect to the SSH remote system, we add the service into our RESTful resources, so its methods could be called when the web service receive HTTP requests. We made a list of what functionalities we wanted to be supported from the SSH server, and defined a route each functionality. For example, the route “/files” will fetch directories and files information of a given path. We follow JAX-RS rules to make these routes and use the Jersey framework to make the route work. Finally, we need a web server on which we can run our application, and Tomcat is that choice. We package the web service and deploy the application under a local Tomcat server.

4. Results

4.1 The big picture:

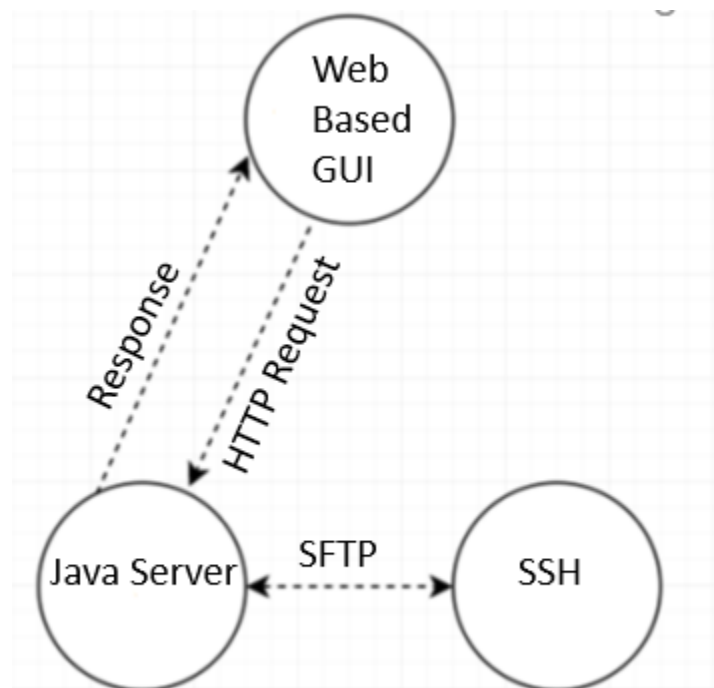


Figure 2 The big connection between our components in our application

To provide access to the remote SSH system and display them in the browser, we develop a Java RESTful Web service using Jersey. This back-end application will be our direct contact with the remote computer system using SFTP and communicates with our front-end Angular application with the REST API. As we can see from figure 2, there is no direct access from our web GUI to the remote system. It only requests information and updates on a remote server via HTTP request to our Java web service. Java web service then receives all information needed to use its SFTP channel to get and modify files on the host machine.

We will discuss these components in depth in the following sections.

4.2 The Front-end Angular application:

Allowing the user to connect any OS that has an SSH connection enabled to the GUI using the sign in page. The sign in page has the Host Name, Port Number, and the Username and Password that you use to sign in with that OS. An example of this for Minix would be Hostname: localhost Port#: 2222 Username: root Password: student. This would gain you access to the Minix file system in the GUI.

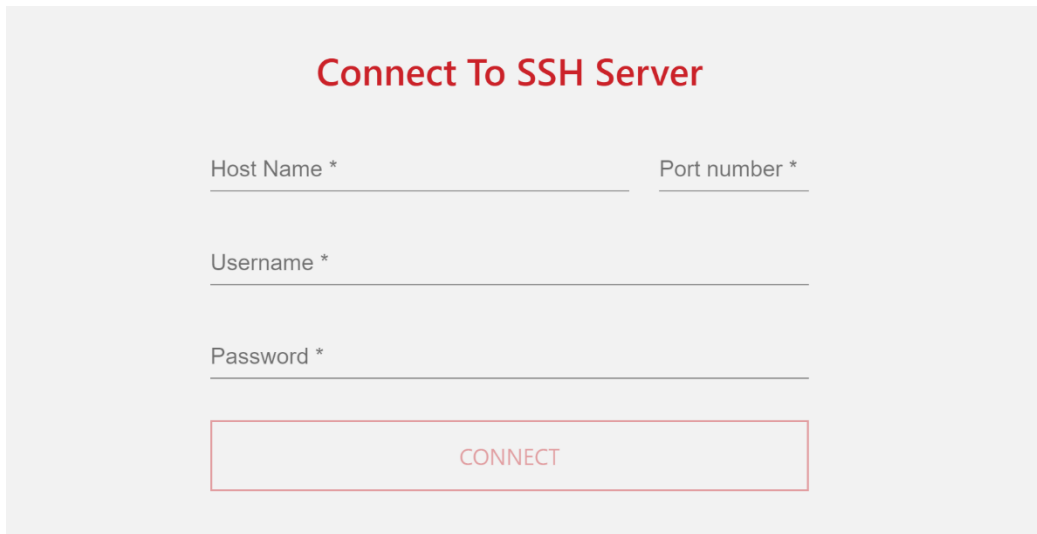
The image shows a web form titled "Connect To SSH Server" in red text. Below the title are four input fields: "Host Name *" and "Port number *" on the first line, "Username *" on the second line, and "Password *" on the third line. Each field has a horizontal line underneath it. At the bottom of the form is a rectangular button with a red border and the word "CONNECT" in red capital letters.

Figure 3 Screenshot of the login screen

Within the file system, there is a button that allows you to navigate directly to the root directory with one click, Icons to show folders contained in the current directory with what folder/file it is and the relevant information based on that file/folder. This useful information includes the size of the file/folder, whether it has read or write permissions, and the date it was last edited. There is also the directory address bar in the top left-hand corner that allows for quick navigation of the file system with the ability to click the folder you want to jump directly to. Double-clicking on the file will open it in the right-hand side of the window and will allow the user to read/write the file depending on the file permissions. Also on this side of the screen, there are save clear and reload buttons that help with file management. The save button allows to save the edits made in the GUI to be saved into the operating system, The reload button reloads the file from Minix allowing to see the updated files in the GUI, and the clear button allows you to clear anything contained in that file. The text is also displayed clearly and easy to read and edit. See appendix for more demonstrations.

4.3 The back-end:

4.3.1 The RESTful API:

All of the functionalities of our front-end side are not possible without a stable and fast web service. That's why our Java RESTful web service is actually the most important part of our project. Our web service is completely isolated from our front-end application. A new front-end app can just plug our web services to get the data, and it has the freedom to change its looks based on the opinions of its developer. It contains the logic to create SFTP channel and directly pipes with the remote computer system to get information of the file system and download files. URLs are exposed to be called from outside to perform actions. The core URLs including the base URL of the host Tomcat server and the route of the service, concatenating with the specific route for functionality:

[http://localhost:8080/minix-web-service/web-api/\[component-route\]](http://localhost:8080/minix-web-service/web-api/[component-route])

The component routes include:

- /login: a route for connecting to a new SSH server. It supports POST method, a JSON object with information for login (hostname, port, username, password), it returns a successful response on success and error response on failure.
-
-
- /files/{path:. *}: a dynamic route to fetch information of a given directory including all of its direct children. This is the route that enables display of the files and folder entries in the front-end, It supports GET method. For example, <http://localhost:8080/minix-web-service/web-api/files/root/test> will return this JSON object that contains information of

```
{
  "absolutePath": "/root/test",
  "directory": true,
  "lastModified": 1523048178000,
  "name": "test",
  "parentUrl": "http://localhost:8080/minix-web-service/webapi/files/root",
  "permissions": 16877,
  "permissionsString": "drwxr-xr-x",
  "size": 320,
  "uid": 0,
  "url": "http://localhost:8080/minix-web-service/webapi/files/root/test",
  "children": [
    {
      "lastModified": 1523048178000,
      "permissions": 16877,
      "permissionsString": "drwxr-xr-x",
      "size": 320,
      "type": "directory",
      "uid": 0,
      "url": "http://localhost:8080/minix-web-service/webapi/files/root/test/."
    },
    { ... }, // 7 items
    { ... } // 7 items
  ]
}
```

Figure 4: An example of JSON response

the directory `"/root/test"`.

- /content/{path:. *}: another dynamic route but for working with files. the GET method will download the file of the given path, and the PUT method for update the file of the

given path with new content. This route enables the file display and editor feature of the front-end app. The GET method will not send back JSON data, but the binary content of the file being requested.

5. Evaluation and Quality Assurance

5.1 Student User Survey

We demonstrated our project and recorded feedback of their experience through a google sheets document. We only asked students that were currently enrolled in this course or those who had taken it in the past. The feedback we gained from this was overall very positive and helpful for future work on this project. Overall, we had 16 responses to the Student User Survey and had some good criticism on the section about what they did not like about it. This gives us a good base on what we need to work on to improve this project in the future seen through an unobscured eye. (See Appendix 4 for the list of all questions)

5.2 Survey Result Summary:

5.2.1 Software behavior questions:

Almost all tester answer “Yes” to these questions, with only 1 exception of no in the question 2.



Figure 5 Behavior questions feedback chart

This proves that all existing features are working well with little or no bug. However, we may need to look at the only “No” feedback about the behavior of the application when the SSH server is shut down.

5.2.2 Positive feedbacks:

In section B of our survey, we asked users for their opinion about our software with 6 short-answer questions. The questions experiment what they like and don’t like about our product in different aspects. For most of the answers, we received very positive feedback:

- 100% asked students think about software works smoothly

- 62.5% asked students are satisfied with the speed of our software
- 100% asked students thinks it is easy to use and they know how to use it without any tutorial
- 68.75% asked students provide feedback about what they like most from the software (figure 6).
- 18.75% asked students think that they don't see anything in the software that they don't like
- 100% asked students would recommend our software to the upcoming students of the course COMP 3000

What do you like most about the software?

11 responses

Clean interface and adds file explorer functions to Minix which is needed
The UI and the reactivity and flexibility of the software
visual representation of minix would get rid of all the suffering endured by students this term with the basic minix. Would actually be fun to learn with this
It's a smoother way to interact with the file system.
nothing
Very fast and responsive
allows for an easy and pleasant way to view files
The GUI
GUI
That it gives user a usable interface to work with the VM
What i like most about the project that it makes the horrible experience of using the virtual machine, into a significantly better experience.

Figure 6 responses what they like most from about software

5.2.3 Negative feedbacks:

We also appreciate and take a much more deep look at the less positive feedback so that we can improve our software further in the future. Here are the problems we received from our testers:

- 50% asked students mentions that they can't create, move or delete new files and directories.
- 37.5% asked students think that login feature is very buggy, it takes a long time freezing before any error appears or for the work console rendered.
- 25% asked students mention they still feel it is slow to render big directory, or at least there is no indication that the page is loading, it just freezes.
- 18.75% asked students ask for hotkey support.
- 6.25% asked students don't like the color scheme.

5.3 Self Evaluation

We ran some time tests on how long it would take from button click until you went into the directory you wanted to go in as well as how long it took to sign in on multiple different operating systems in the VM. Through these results, we determined that our project has inconsistent load times in which the file that contains the most information in it will not always load the slowest. For this testing we added time variables into our front end code to determine how long it would take to create the request (for graphs see appendix #4 figures 1&2). Through the operating system login test we determined that overall the time to log in to Minix in the GUI was quick compared to other Operating Systems. However, these times do not translate well into the actual user interface which is sometimes slow and glitchy. Another inconsistency in this project was the login. Often the page would take several seconds to load and sometimes will not connect despite using the correct login information. This can be attributed to the computer it is run on since it does not scale very well with lower end systems.

There are a lot of limitations to the current project that can be implemented to make it more functional. These limitations include only being able to connect one user to a single OS where this project could have been used as a remote SSH client for multiple users to do work on the same OS. Another limitation we have encountered in this project is the ability to create and delete files from the GUI, this feature would have allowed users to exclusively use the GUI, but instead they will still be forced to navigate the native OS in order to create and delete files. Another limitation we discovered is that in order to configure the tomcat server localhost:8080 you would not be allowed to have any other applications hosted on that port, this would lead to frustration and a lot of troubleshooting for the user that might be seen as not worth the effort in the end since there are other GUI's available. This project has lots of limitations that make it so that this project is not useful for all users, and is only useful as a secondary solution since you are not able to create and delete your own files or use it as a group based GUI as well as speeds are not consistent among all computers since you are locally hosting the server.

6. Conclusion

6.1 Summary:

In conclusion, SSH Connect was created not only for the course's requirement, we also want to help students who will be taking COMP3000 course and possibly work with SSH remote server with a nice GUI. We created a back-end web service that could be used independently without the front-end GUI to get information of the remote file system. But we also create a good-looking browser-based Angular application to work as a file manager and a quick text editor. We used singleton design pattern to maintain the connection which speeds up out an application, created a RESTful API for the front-end to connect and finally a modular and scalable front-end code base. We believe with the base structure of the application, SSH Connect could scale fast to support users with more features in the future

6.2 Relevance:

SSH Connect provides users another way to work with SSH remote system. We applied knowledge from the course COMP 3000 about the file system and create a similar file manager and editor to give the same experience for the user from Windows, Linux or MacOS, with the only difference is that users are accessing remote files. We address the difficulties that students face in the course while they work with Minix OS with no GUI based environments. This project will help student ease their frustration and can focus on the content of the work they need to do in exercises, as it is easier now to know where to find the file they need to change and edit the file directly to the remote machine with the supported C-syntax editor.

6.3 Future Work:

With the limited amount of time, we could not finish every desired feature. Here is the list of all features that we will add in the future:

- Ability to edit the file system: creating, deleting, renaming, copying files and directories
- Support opening file with other formats: pdf, png...
- Support multiple sessions and SFTP channels to connect to multiple SSH servers at the same time
- Ability to download and upload file between the local machine and the remote server
- Ability to search for files and directories, memorize the recent worked files and directories
- Changing the styles of the front-end app to the favor of users: tabs, syntax highlighting for multiple languages
- Build a unit testings plan to help the project easier to detect severe errors and making sure the productivity of the application
- Ability to temporarily store working file if disconnection happens

Contributions of Team Members

- A. Tri Cao created the back-end RESTful web service, including creating API endpoints and SFTP service to create a connection between SSH remote system and our application. Tri is also responsible for the login feature of all front-end and back-end side.
- B. Spencer developed the front-end application with Angular. Spencer created all components in the front-end of the work console and the SSHClientService to send HTTP requests to our back-end. As well as final editing of the report.

References

- (1) Tools.ietf.org. (2006). *RFC 4251 - The Secure Shell (SSH) Protocol Architecture*. [online] Available at: <https://tools.ietf.org/html/rfc4251> [Accessed 12 Apr. 2018].
- (2) Tools.ietf.org. (2013). *draft-moonesamy-secsh-filexfer-00 - SSH File Transfer Protocol*. [online] Available at: <https://tools.ietf.org/html/draft-moonesamy-secsh-filexfer-00> [Accessed 12 Apr. 2018].
- (3) Jcraft.com. (n.d.). *JSch - Java Secure Channel*. [online] Available at: <http://www.jcraft.com/jsch/> [Accessed 12 Apr. 2018].
- (4) Ics.uci.edu. (n.d.). *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*. [online] Available at: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm [Accessed 12 Apr. 2018].
- (5) Jcp.org. (n.d.). *The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 311*. [online] Available at: <https://jcp.org/en/jsr/detail?id=311> [Accessed 12 Apr. 2018].
- (6) Jersey.github.io. (n.d.). *Jersey*. [online] Available at: <https://jersey.github.io/> [Accessed 12 Apr. 2018].
- (7) Tomcat.apache.org. (n.d.). *Apache Tomcat® - Welcome!*. [online] Available at: <http://tomcat.apache.org/> [Accessed 12 Apr. 2018].
- (8) Skillcrush. (n.d.). *Exactly What You Need to Know to Be a Front End Developer*. [online] Available at: <https://skillcrush.com/2016/02/11/skills-to-become-a-front-end-developer/> [Accessed 12 Apr. 2018].
- (9) GitHub. (n.d.). *Angular*. [online] Available at: <https://github.com/angular> [Accessed 12 Apr. 2018].

Final Word

We hope that SSH Connect will become actually useful for anyone who wants to work with remote SSH file system. We hope we could receive more support for continuing to develop our software. With more research and usability testing, we hope we can improve it in the favor of most of the users, bring better experience using SSH Connect

Appendix 1: A Guide for installing and running SSH Connect

We assume you have the zip file of our application and extract it into a folder. Here are the steps to set up and run our application.

1. Install Tomcat server:

Go to <http://tomcat.apache.org/index.html> to download Tomcat.

Install, remember to provide username and password when they ask in the setup. Make sure that the Tomcat server will run on port 8080 on your local machine.

2. Open Tomcat manager and run RESTful web service:



Tomcat Web Application Manager

Message:

Manager

[List Applications](#) [HTML Manager Help](#) [Manager Help](#) [Server Status](#)

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	<div>Start Stop Reload Undeploy</div> <div>Expire sessions with idle > 30 minutes</div>
/docs	None specified	Tomcat Documentation	true	0	<div>Start Stop Reload Undeploy</div> <div>Expire sessions with idle > 30 minutes</div>
/manager	None specified	Tomcat Manager Application	true	1	<div>Start Stop Reload Undeploy</div> <div>Expire sessions with idle > 30 minutes</div>
/mini-web-service	None specified	Archetype Created Web Application	true	0	<div>Start Stop Reload Undeploy</div> <div>Expire sessions with idle > 30 minutes</div>

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:

Deploy

WAR file to deploy

Select WAR file to upload No file selected.

Deploy

Go to <http://localhost:8080/manager> with your username and password. In the WAR file to deploy section, click Browse... button. Then choose the war file in the root folder of our project.

.git	4/11/2018 3:43 PM	File folder	
.metadata	4/11/2018 3:43 PM	File folder	
.recommenders	4/2/2018 4:10 PM	File folder	
back-end	4/3/2018 3:07 AM	File folder	
front-end	4/2/2018 12:07 PM	File folder	
RemoteSystemsTempFiles	4/2/2018 11:49 AM	File folder	
Servers	4/3/2018 12:23 AM	File folder	
.gitignore	4/2/2018 11:51 PM	Git Ignore Source ...	1 KB
Comp3000Tri.docx	4/9/2018 3:57 PM	Microsoft Word D...	23 KB
hotswap-agent.jar	3/2/2018 10:16 AM	Executable Jar File	1,412 KB
minix-web-service.war	4/9/2018 9:57 PM	WAR File	17,715 KB
README.md	4/2/2018 12:25 PM	MD File	1 KB

Click Deploy and you will have our RESTful web service works.

3. Install nodeJS and Angular CLI:

Our front-end application requires NodeJs, especially Node Package Manager (npm). Guide for installing NodeJs can be found here: <https://nodejs.org/en/download/>

After installing NodeJS, you can now install Angular CLI. Guide for installing Angular CLI can be found here: <https://cli.angular.io/>

4. Run our front-end Angular application:

Open terminal/command prompt, navigate into our front-end directory and run “ng s -o” to run our front-end application. The login page of our front-end app will be opened after that.

```
C:\Users\Tri Cao\Desktop\Project (master -> origin)
λ cd front-end\

C:\Users\Tri Cao\Desktop\Project\front-end (master -> origin)
λ ng s -o
|
```

5. Now make sure you have an active SSH remote server to connect as well as a credential to start using our application.

Appendix 2: SSH Connect screenshots

Here are some screenshots of our front-end application, to help you understand more about currently supported features of our application.

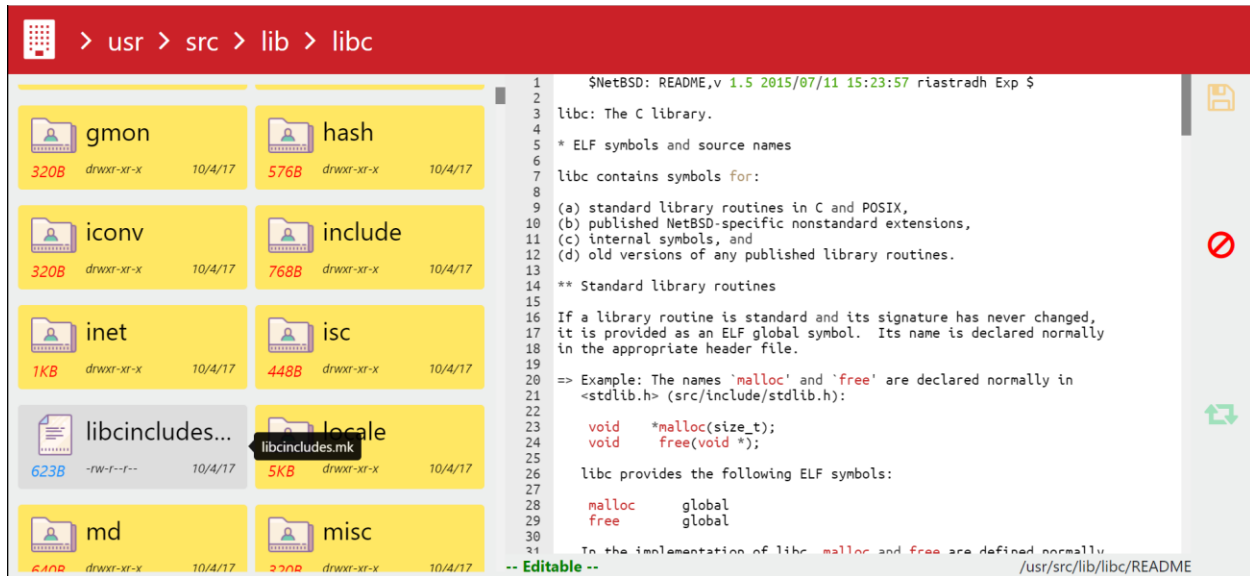


Figure 7 Opening an editable file

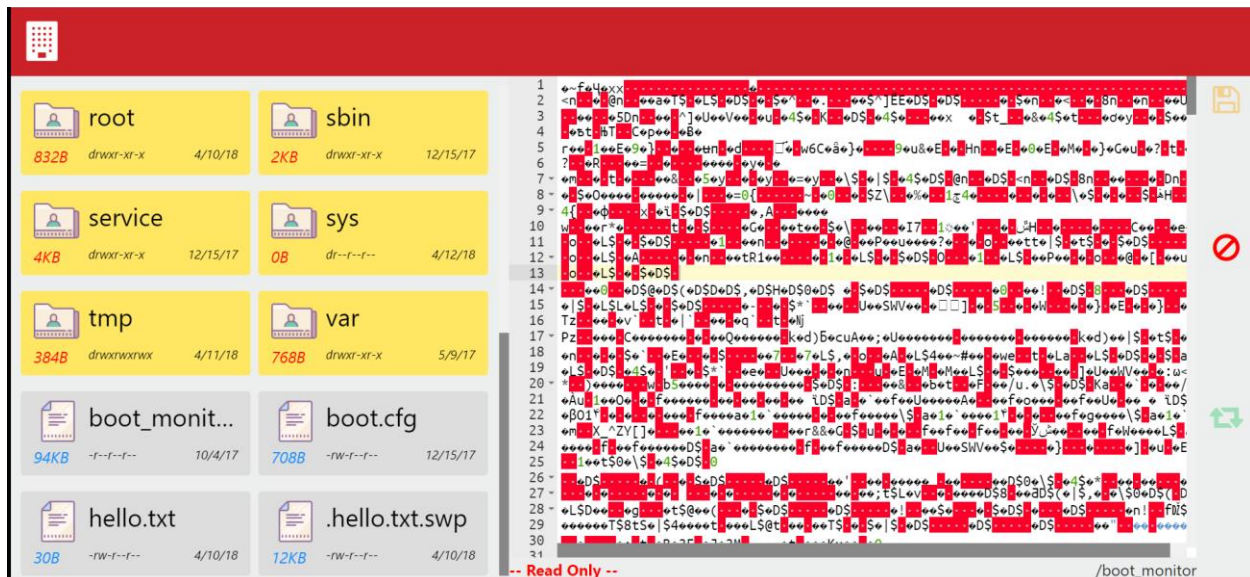


Figure 8 Opening a read-only file

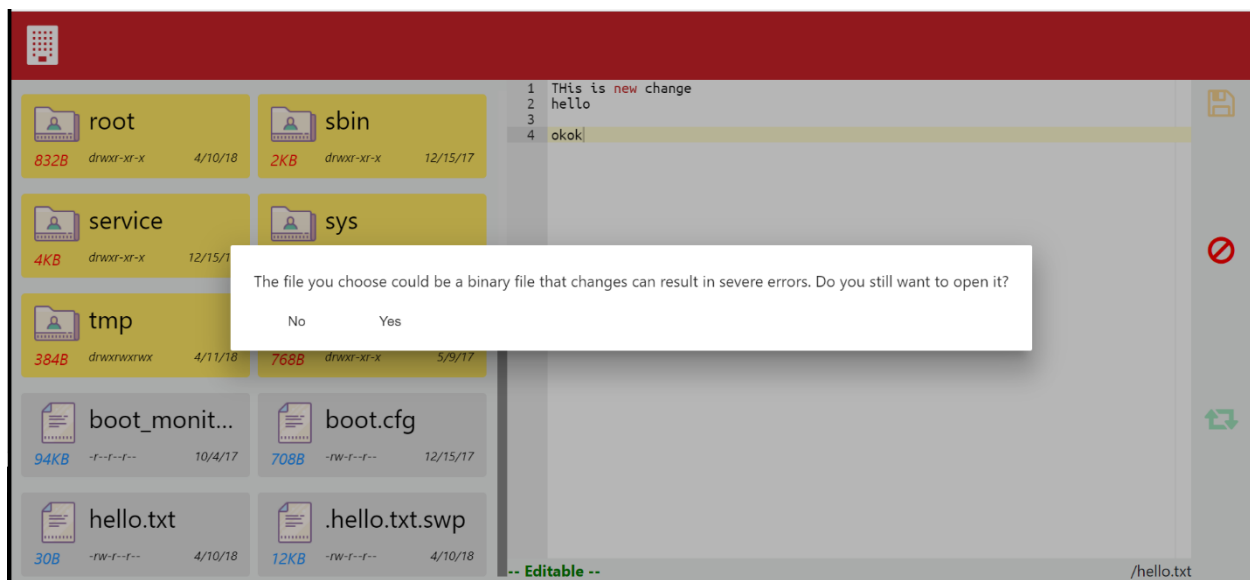


Figure 9 Warning when trying to open a binary file

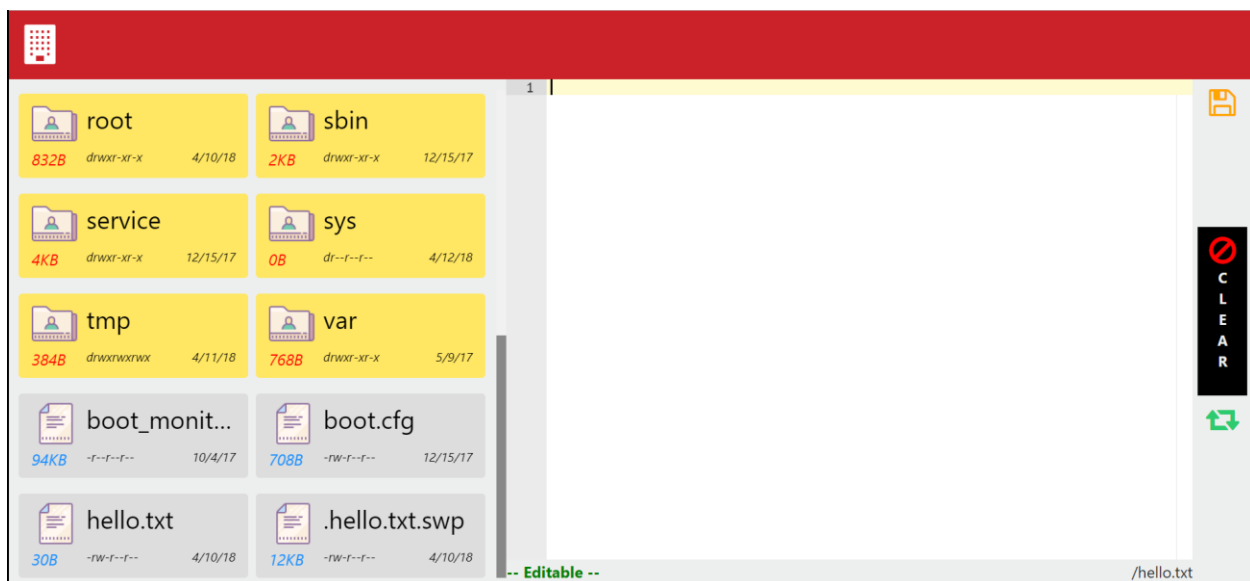


Figure 10 Clearing the editor

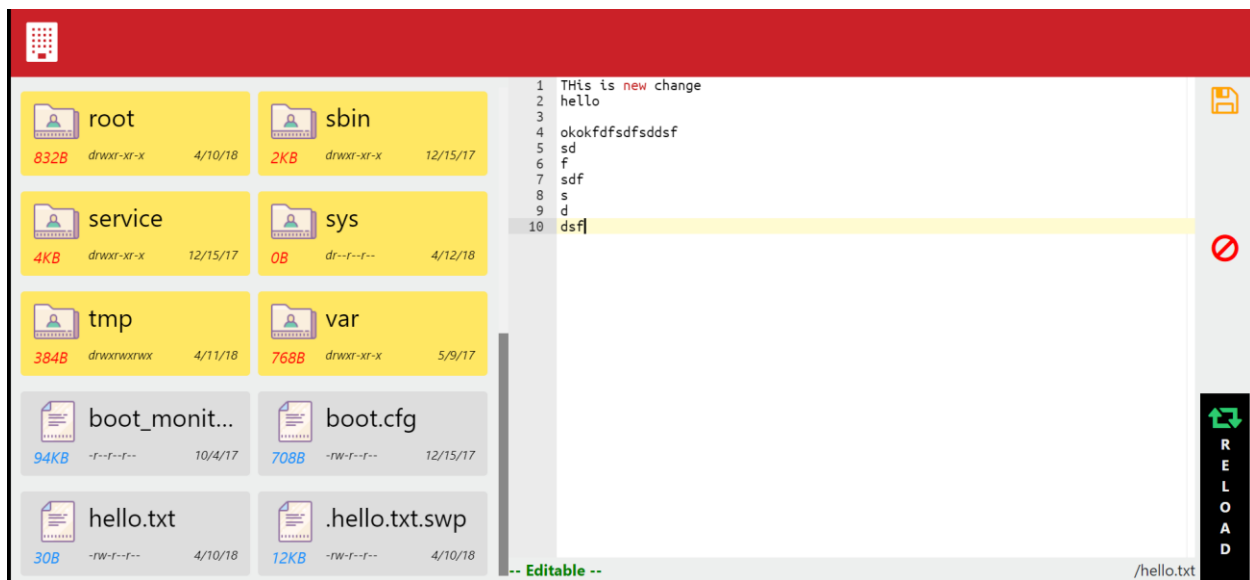


Figure 11 Reloading the actual content of the file on the remote system

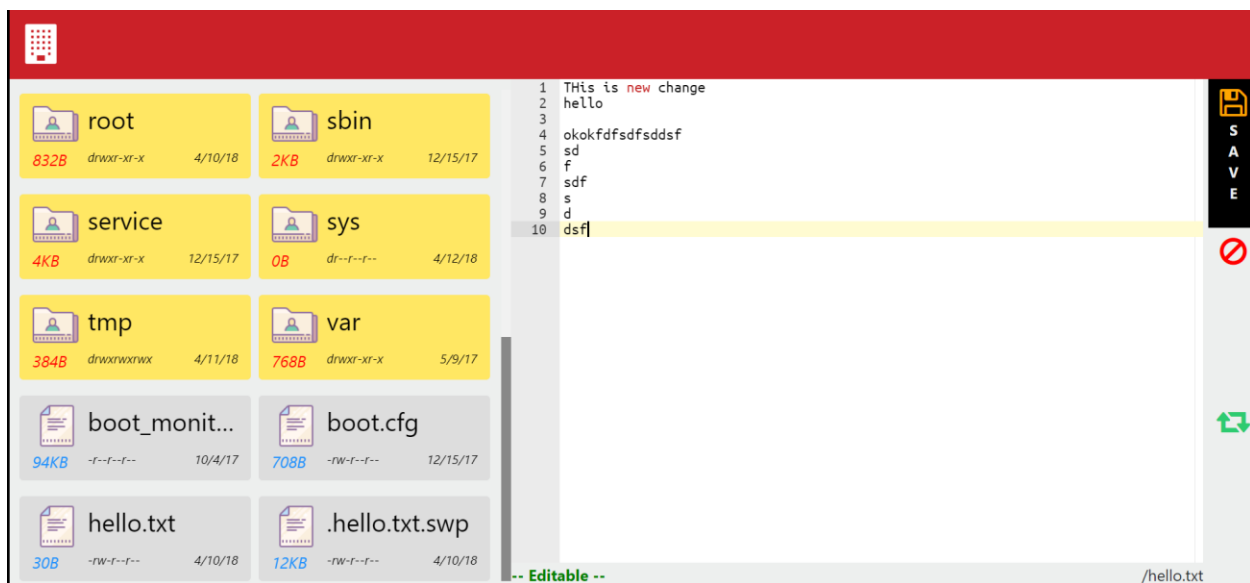


Figure 12 Saving file (update file on the remote server)

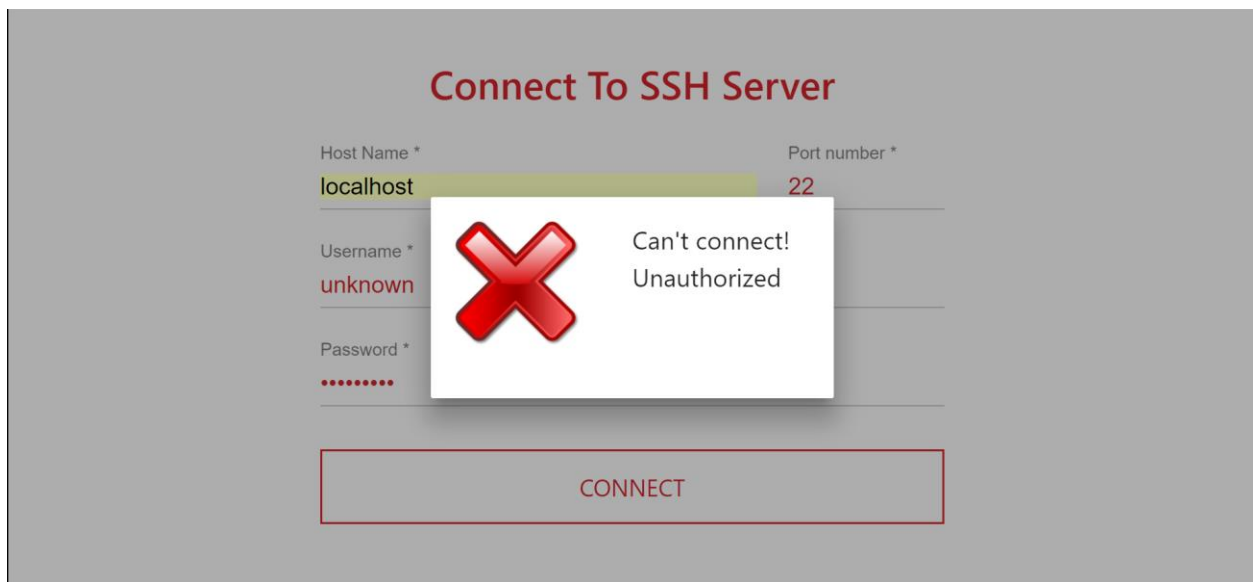


Figure 13 Error dialog when login with an unauthorized account

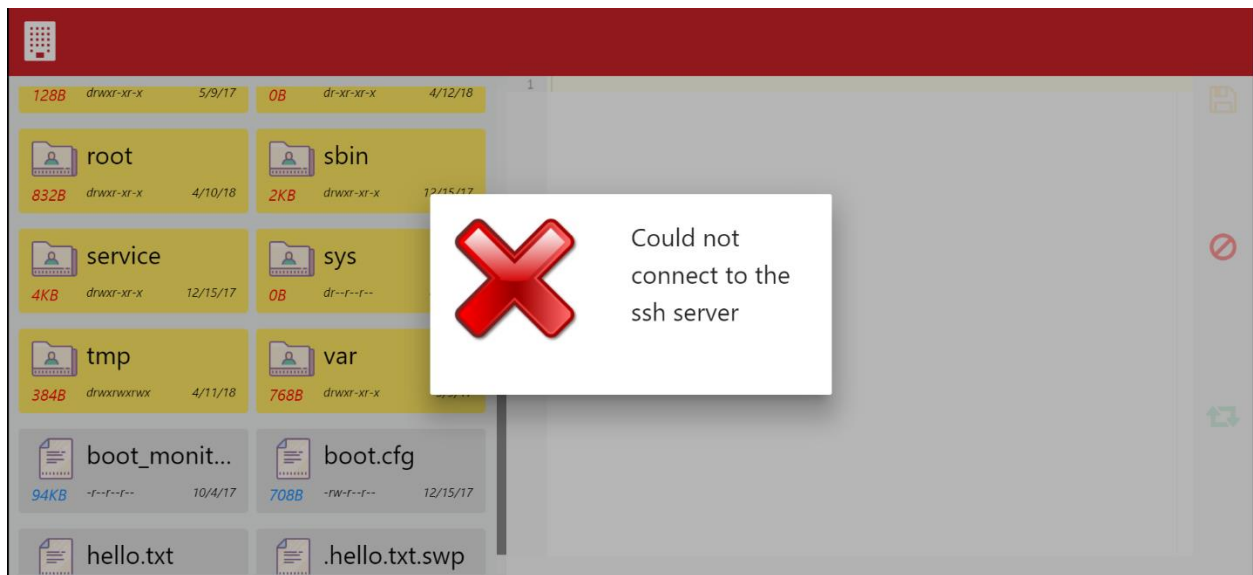


Figure 14 Error when the server is suddenly disconnected

Appendix 3: Table of Acronyms

Acronyms	Meanings
SSH	Secure Shell (A software package that allows a secure connection for file transfer and system features)
GUI	Graphical User Interface (An interface that allows users a more graphical experience to use software through mouse clicks and buttons)
VM	Virtual Machine(Allows for users to boot into a virtual image of an operating system while simultaneously using their default)
SFTP	SSH File Transfer Protocol (Protocol used for enabling Secure Shell Securities)
REST	Representational State Transfer ()
HTTP	Hypertext Transfer Protocol ()
JSCH	Java Secure Channel ()
JAX-RS	Java API for restful web services ()
SCP	Secure Copy Protocol()
FTP	File Transfer Protocol()

Appendix 4: Survey questions

This is the list of all questions in the survey that we received total 16 feedbacks from other students:

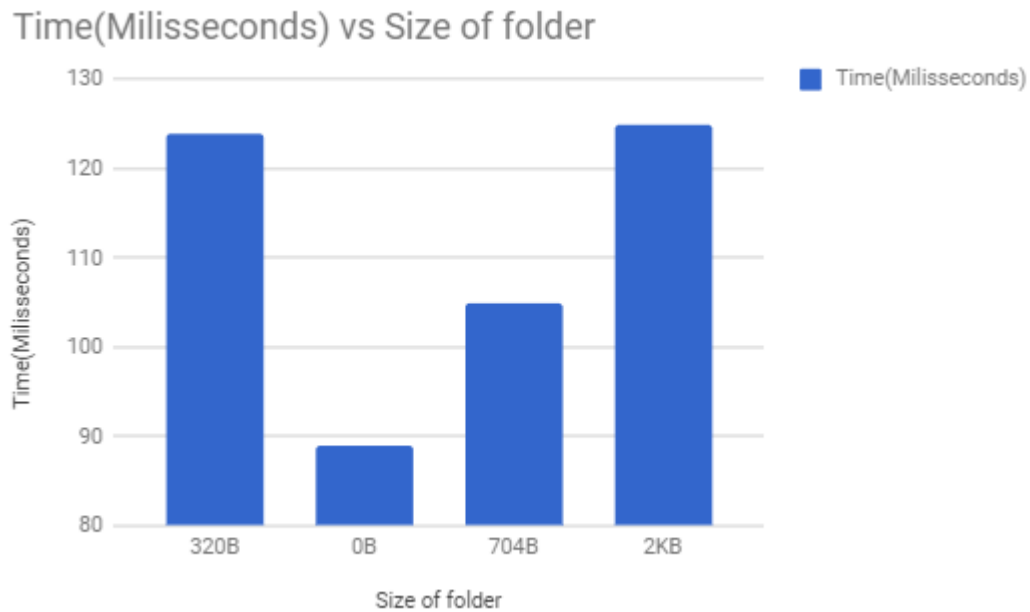
A. Software behavior questions:

1. Are you able to login to SSH server? (Yes/No)
2. Can the Application react to disconnection of server? (Yes/No)
3. Can you navigate through the file system? (Yes/No)
4. Are you able to open files and modify them if you have the write permission? (Yes/No)
5. Does the minix machine reflect the changes to the files that you modify and save with the application? (Yes/No)
6. Does the application warn you if you try to open a binary file? (Yes/No)

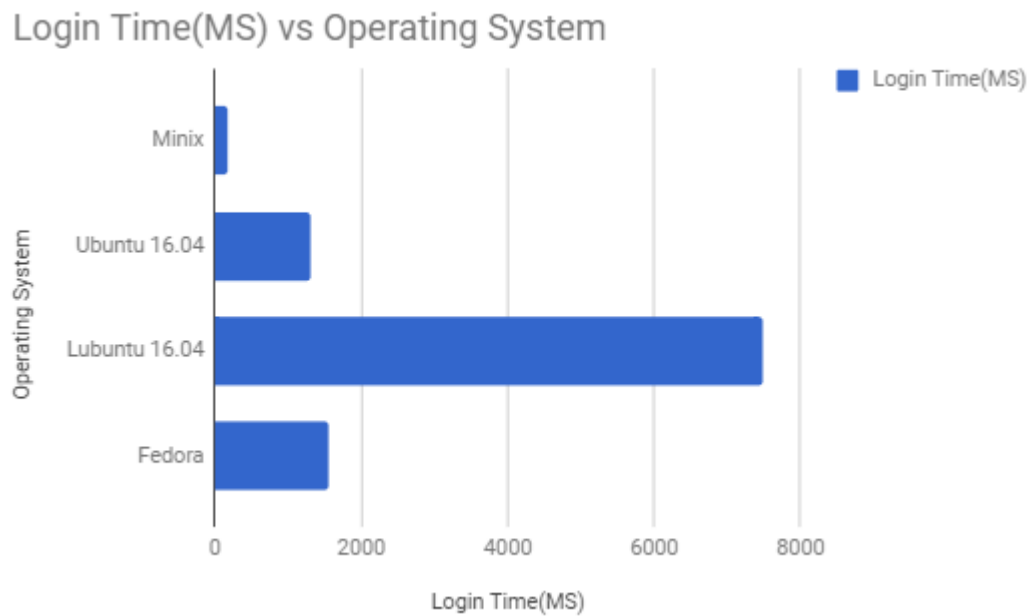
B. User opinion questions:

7. Do you think the software works smoothly?
8. How much are you satisfied with the speed of our software?
9. Do you think the application is easy to use?
10. What do you like most about the software?
11. What do you not like about it?
12. Would you recommend it for upcoming students of COMP 3000 using our software to work with their exercises?

Appendix 5: Time testing charts



Appendix #5 Figure 1 Time in Milliseconds to enter a folder



Appendix #5 Figure #2 Login Time vs OS