Systems software Java shell project
1
Group Fii4

Matthew Boote N0944562
Derice McDonald N0945044

SOFT202122 Systems software
Pedro Machado

COMP20081 Systems Software

Nottingham Trent University
Department of Computer Science

April 2022

Video demonstration: https://www.youtube.com/watch?v=5l9w01JQ2Sg

Table of Contents

Abstract

All software systems require a user interface that allows the user to interact with it, a type of interface is the text-based command line which accepts textual input from the user and displays output in the same way.

Remote access to user interfaces over networks using programs such as telnet or netcat services have been used to access systems, similarly, graphical access over client program has been possible via protocols such as Remote Desktop protocol. However, these program do not offer a way to access a server without relying on specialized client websites to provide the user interface.

Java shell is designed to be a light-weight server that generates and serves HTML pages over a HTTP/HTTPS connection on the that are directly sent to the server to the client browser via a web address and port allowing the user easy access to a semi-text based shell interface.

The shell aims to be secure, rather than rely on the security of the underlying operating system, it implements a per-user white list of permitted external programs that limits access to commands that could be dangerous and it limits access to the user's home directory to prevent access to other users

files. It was implemented with Java NetBeans in four different classes: Main, a main class that set up the connections, Session that represented a single user session, User that represented a single user and Command that represented a command execution context.
The following libraries were used java.io, java.net.ServerSocket, java.net.Socket to provide network access java.security.KeyStore, java.util.concurrent.Semaphore, javax.net.ssl.KeyManagerFactory, javax.net.ssl.SSLContext, javax.net.ssl.SSLServerSocket, javax.net.ssl.SSLServerSocketFactory,
javax.net.ssl.SSLSocket and javax.net.ssl.TrustManagerFactory.
The results of the project shows that a lightweight, efficient and portable graphical shell interface can be implemented with minimal overhead.

Introduction

Remote access via a network server program is common; services such as HTTP/HTTPS and SSH provide user access to services. However, the security of these services often only restricts access via authentication on connection and access data stored on the system; any user can navigate to and access the files outside their allocated user directory. Current remote systems use authentication and file permissions to restrict user access to information, Users will often require access to programs and data that are installed, and the system must be configured to allow read access., However this is insufficient to prevent users from knowing about the presence of files outside of their home directory. An example is when programs on a UNIX system have their permissions set to execute to permit execution; the program file will still be visible to the user; this could cause a security hazard by exposing files and directories to users that should
The project has developed and implemented a system that restricts both read and write access outside of the user's home directory to both the user and running programs.

Background research

The following tools were used to create the program:

NetBeans was used to implement the program in Java

The diagrams were created using Lucid Chart

Design

Feasibility of the project

The project is feasible, because similar concepts have been used to restrict access to information; the use of virtual machines such as VMWare and compatibility layers like Wine to filter and interpret data between programs and the system. This demonstrates that the concept is feasible and that it can be implemented.

Functional requirements

A list of functional requirements was created to set out what was expected from the program in the form of the detailed requirement, the priority of the requirement, any implications of implementing the task in the

Must have requirements

#	Requirements	Priority	Implications	Task
1 * (needs prompt)	It must display a prompt and accept user input	MUST	Requires networking to be implemented and for sockets to be also implemented	T1.1 Implement shell prompt
2 *	It must execute external programs and display the output to the user	MUST	Requires external process execution and input and output to be redirected to sockets.	T2.1 Implement process execution with Process Builder.
3*	It must be able to add new users	MUST	Must be able to read user list to check if user exists and update user list by writing to user list file. Must use mutexes to prevent concurrent access.	T3.1 Implement method to add user.
4*	It must be able to remove users	MUST	Must check if user exists before removing user. Must use mutexes to prevent concurrent access.	T4.1 Implement method to remove user.
5	It must be able to execute administrator commands	SHOULD	Must check the username before executing the commands and that the command is valid	T5.1 Implement super command.
6	It must be able to change user passwords	SHOULD	Must check if the superuser if changing other users' password. Must check user exists. Must use mutex to prevent concurrent access to the user's file.	T6.1 Implement method to change user password

7*	It must be able to move files	MUST	Must be able to execute mv external command	T7.1 Make sure that mv is available
8*	It must be able to copy files	MUST	Must be able to execute cp external command	T8.1 Make sure that cp is available
9	It must be able to change the user account type	MUST	Must check if superuser Must check if user exists. Must check if user account type if valid	T9.1 Implement method to change user account type
10*	It must be able to show the current username	MUST	Username must be stored in a variable.	T10.1 Implement whoami method.
11*	It must be accessible over a network	MUST	Each network connection should be a separate object.	T11.1 Must implement networking methods. It must implement an initialization, accept, read, and write method.
12	It must be able to run commands as superuser	MUST	Need to check username and deny access if not root	T13.1 Implement getusername function.

Figure 1: Must have requirements

Should have requirements

#	Requirements	Priority	Implications	Task
1 *	It should allow the user to log in	SHOULD	The system should prompt for username and password. The system should keep a list of users and implement a method to authenticate users. The passwords should be hashed to protect them.	T14.1 Implement method to authenticate users.
2*	It should allow the user to log out	SHOULD	The system should terminate the user’s thread	T15.1 Implement logout method.

3*	It should allow multiple users to connect	SHOULD	It should use threading to allow multiple users to connect.	T16.1 Implement threading.
4*	It should prevent users from moving outside their home directory	SHOULD	It should check the real path of the current directory and return an error if it is not within the user’s home directory	T17.1 Implement path checking in chdir
5*	It should prevent external programs from accessing files outside the user's home directory	SHOULD	It should check the command-line arguments and any input and output from external programs	T18.1 Implement filtering of input and output to external programs
6*	It should implement a chdir method to set the current directory	SHOULD	It should store the real path and create a virtual path from it.	T19.1 Implement chdir method.
7*	It should implement a getcwd method to get the current directory	SHOULD	It should return the virtual path, not the real path	T20.1 Implement getcwd method.
8*	It should only allow permitted external commands	SHOULD	It should store a per-user list of permitted commands and implement a command validation method to check it before executing an external command.	T21.1 Implement command validation method.
9*	It should filter command outputs to remove references to paths outside the user's directory	SHOULD	It should remove any references to the real path in the command-line arguments, input, and output.	T22.1 Implement filtering of input and output.
10	It should encrypt user files and directories	SHOULD	Should implement a method that compresses files	T23.1 Implement extract and decrypt files on sign in and

			into a zip file with the user's password. Change zip password when user changes their password	encrypt and compress on sign out.

Figure 2: Should have requirements

Could have requirements

#	Requirements	Priority	Implications	Task
1	It could have SSL security	1	Implement HTTPS	T24.1 Implement SSL layer.
2	It could allow the superuser to change user accounts	3	Need to save the original username before changing the username. It should store the user ID and the effective user ID.	T25.1 Implement user ID checking and superuser commands
3	It could have help manual	1	Need to write help manual	Write help manual using Doxygen

Figure 3: Could have requirements

Will not have requirements

#	Requirements	Priority	Implications	Task
1*	It will not have file transfer capabilities	1	No need for file transfer capabilities	No task.

Figure 4: Will not have requirements

Non-functional requirements

#	Requirements	Priority	Implications	Task

Figure 5: Non-functional requirements

Risk assessment

Risk	Probability	Impact	Mitigation plan

Damage to files (data/programs)	3	5	Make use of olympuss file history by backing up code regularly
Damage to hardware	2	5	Use alternative hardware.
Extreme weather (flood/earthquake/hurricane etc)	2	5	Postpone the project or use alternative location.
Communication problems between team members	3	3	Regular Teams meetings weekly to discuss what has been done/needs to be done
Resources not available to complete project	3	5	Use alternative resources.
The project is too complex to complete	2	5	Simplify project.

Figure 6: Risk assessment

Gantt chart

Tasks, milestones, and deliverables	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
T1.1 Implement shell prompt	X																								
T2.1 Implement process execution with Process Builder.		X																							
T3.1 Implement method to add user.			X																						
T4.1 Implement method to remove user.				X																					
T5.1 Implement super command.					X																				
T6.1 Implement method to change user password						X																			
T7.1 Make sure that mv is available							X																		

T8.1 Make sure that cp is available								X																
T9.1 Implement method to change user account type									X															
T10.1 Implement whoami method.										X														
T11.1 Must implement networking methods. It must implement an initialization, accept, read, and write method.											X													
T12.1 Implement getusername function.												X												
T13.1 Implement method to authenticate users.													X											
T14.1 Implement logout method.														X										
T15.1 Implement threading.															X									
T16.1 Implement path checking in chdir																X								
T17.1 Implement filtering of input and output to external programs																	X							
T18.1 Implement chdir method.																		X						
T19.1 Implement getcwd method.																			X					
T20.1 Implement command validation method.																				X				
T21.1 Implement filtering of input and output.																					X			
T22.1 Implement extract and decrypt files on sign in and encrypt and																						X		

compress on sign out.																								
T23.1 Implement SSL layer.																							X	

Figure 7: Gantt chart

Implementation

Use case diagram

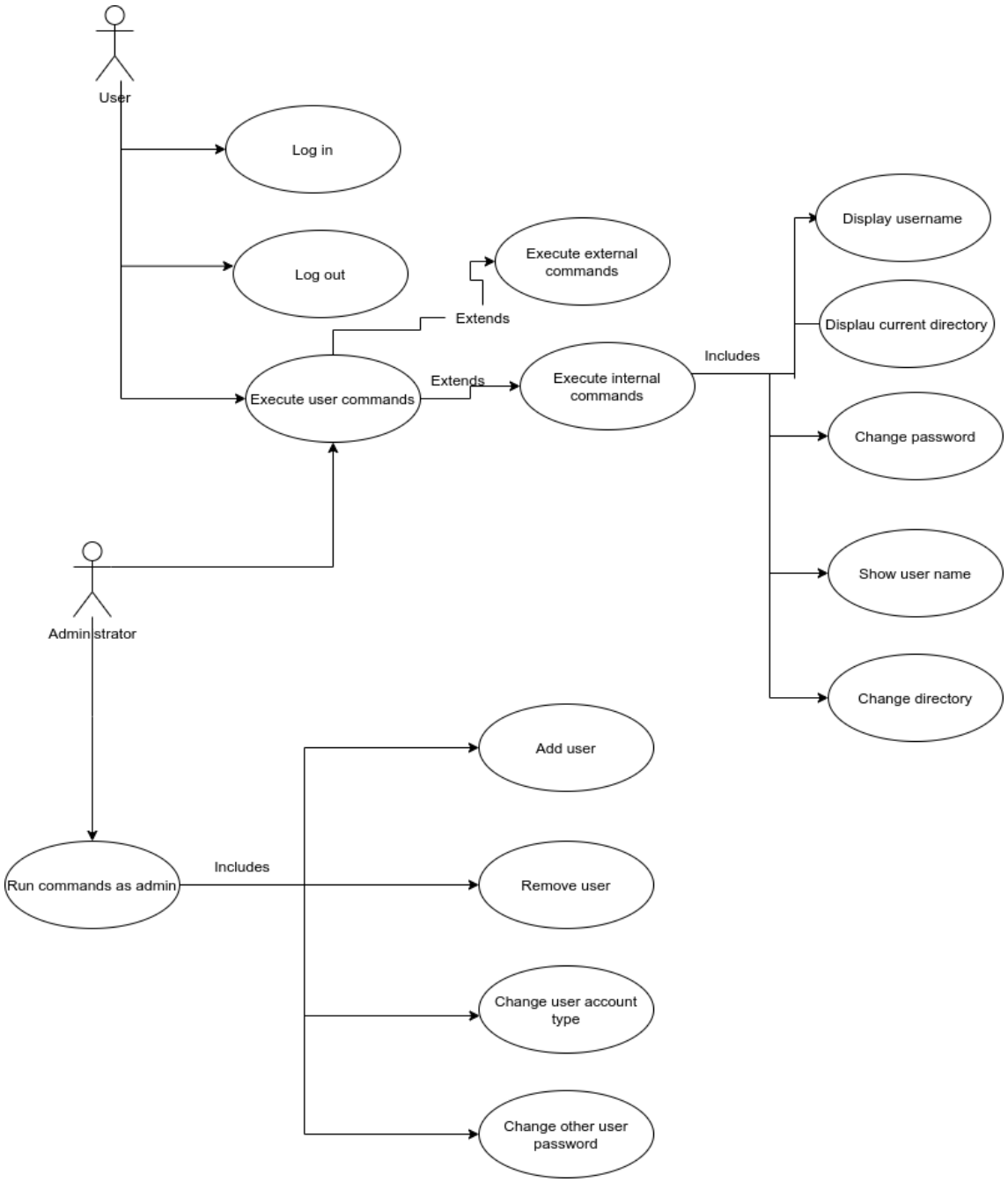
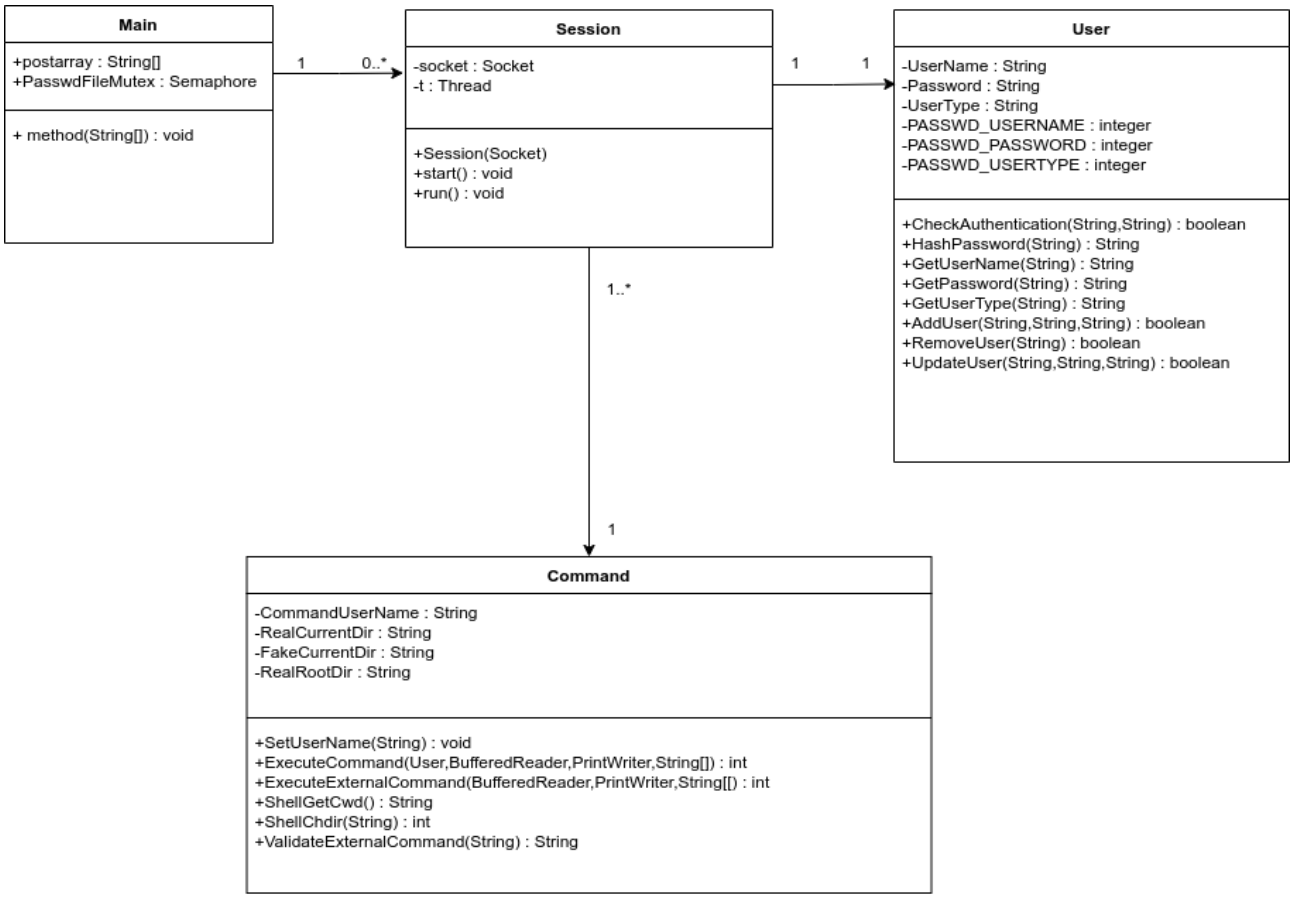


Figure 8 Use case diagrams
The use case diagram shows a high-level view of the project. It maps the *use cases* or the different uses of the system by the users and the relationships between the possible actions of the users with the *actors* or the users.

Class diagram

Figure 9: class diagram

The class diagram was created from the use case diagram and shows a low-level view of the project



where each part of the diagram shows a class or a part of the program, which are used to construct objects that are when used to create the program. This contrasts with the use-case diagram, which shows only a high-level view of the program.

Sequence diagram

A series of sequence diagrams were created to show the steps carried out to perform each of the use cases; it shows the classes that are involved in the sequence and the interactions between them.

Sign in

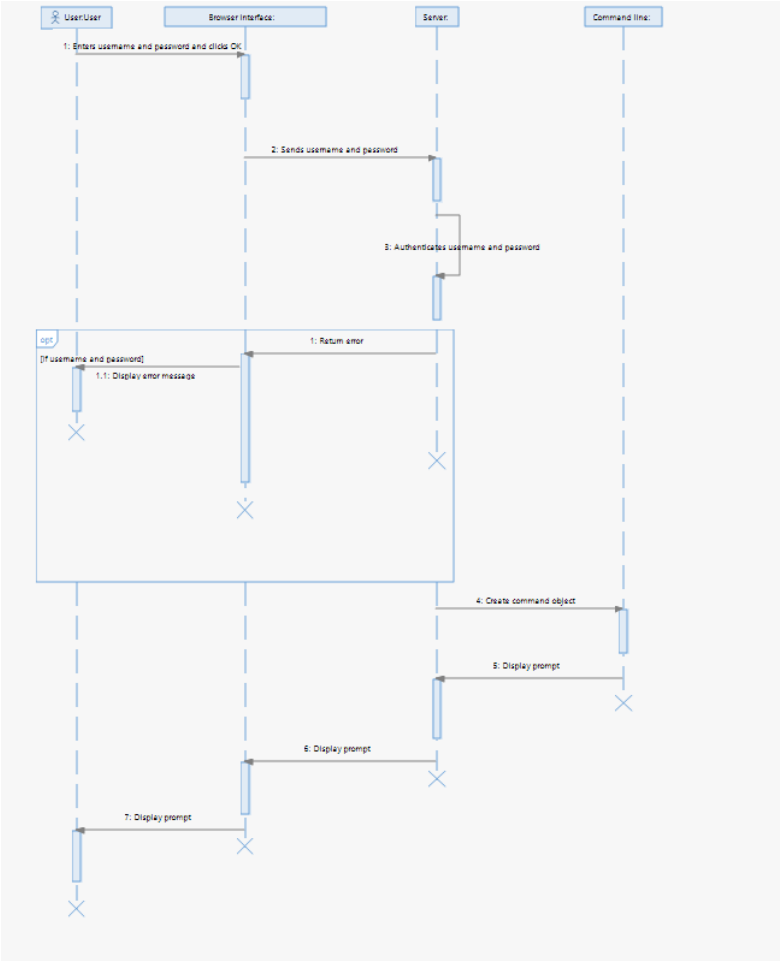


Figure 10: Sequence diagrams
A sequence diagram was created to show the steps taken by methods.

Process command

The project was to create a command-line user interface, which is necessary to allow for user interaction with a system, most computer systems are directly used by users and all systems that permit interactive input from users must use some method of obtaining user input.

The Java Shell uses a command-line interface where the user controls the computer via a series of text commands that are submitted by the user and the system responds with textual output. The command-line interface has the advantage that it does not need to implement any specific features, interaction between the user and the shell is unformatted text. The command-line interface was designed to be within a graphical interface created in HTML and CSS, rather than use a purely textual user interface the program displays a simple GUI interface consisting of a form containing a textbox to display output, an input box to read input from the user and a button to submit the information.

The different functionalities will be, as standard practice in Object-Oriented Programming, be divided into the classes; the abstraction of each of the classes provides for flexibility in the use of the classes, first because it separates and hides the various parts of the system from each other; it its

considered good practice for programs to be divided into cohesive, weakly coupled system where each class is responsible for a single aspect of the program or *separation of concerns*. This permits the re-use of parts of the program, with increased flexibility, avoids duplication of code and increases the speed of development; when the desired features from the functional requirements are implemented, they can be reused without extra effort accordint to Ian Sommervillin in The Fundemental Practice of Software Engineering “In general a good composition principle to follow is the principle of separation of concerns. That is, you should try to design your system so that each component has a clearly defined role” (Sommerville, 2016)

A similar idea is *separation of ideas and policy*, where a feature and the implementation of the feature are separated, an example is the Session class, which represents a single connection session and is created every time that a user connects; it abstracts the processing of the connected session from the Main class, which handles the initial connection.

The project is divided into four classes: Main, Session, User and Command. Each of the classes encapsulates, or hides the class attributes and methods within it, however, it is often necessary to access the class attributes within a class.

A type of design pattern used in the Command and User classes, the *getter* and *setter* patterns are used to expose class attributes outside the class cleanly without external references, using *import, for* example in other classes to reference it. This keeps any code that accesses the class attribute within a method that is easily modified. An example of a getter method within the program is GetUserName() which returns the username.

The Main class is the main point of entry for the program, its function is to initialize the program so that it is ready to accept user connections.

It creates a ServerSocket object to function as a master socket that listens for connections on localhost, using port 8080 and uses the object’s accept method to create a private Socket object for that client. This a standard method of creating network connections that splits creation of the socket and handling of user connections based on the Berkeley sockets model of network connections. [cite]. Following the user connection, it creates a session object, the session object represents a single user session from when the user connects until they disconnect, each user has their own session object to ensure that the principle of each user and their data being completely separated from each other, for security reasons, each user should not be able to interfere with the other’s connections.

The system accepts connections from users, however there may be multiple users sharing the server’s resources; it must find a way to share time and resources fairly between the users. A method the Java language has is to use threading to simultaneously manage multiple users; the program’s time is split between the threads, with each user appearing to have exclusive control of the system. However, each user thread runs for a period or quantum and the next thread is run, using multi threading is an efficient way to manage multiple connections [cite], because of the threads appearing to run simultaneously. [cite]. Implementation of the threads is with a Java interface which is derived from the Runnable class used to create classes used to spawn threads, according to Oracle, “The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread” [cite].

Java interface classes, however, unlike ordinary classes which provide the methods in the class, only define the methods that should be implemented by the class and not the methods themselves; the child class using the interface is expected to implement the functions that override them [cite]. Using Interface classes to create a class to run the threads defines an empty template that the Session class adds to implement threading. The class is expected to define two functions, that accepts no parameters, start() and run() that implement the functionality for starting and executing the thread. [cite]. The method start () is executed when the thread is started and is expected to initialize threading. The run() method is the main method for the thread. In Java Shell, the start() method creates a new Thread object class member that represents the thread, the thread uses this keyword to execute within the Session object that is in use, it will spawn a thread that continues in the start method and then invokes the thread's start method, the thread then executes the run method that continues the threading.

The session object handles most of the user connection within its run() method, moving the connection I/O away from the initial connection. each user has a InputStream to handle input, it is implemented on top of InputStream (more?). The output uses a similar method, with a PrintWriter object to handle buffered output, it is created on top of the socket's getOutputStream() Method. Using buffered I/O, rather than sending and receiving raw data is a more abstract way to handle data and the InputStream and PrinrWriter methods can read and write lines of text, which is the method used by a HTTP server to communicate.

A User object is created to represent a user associated with the session and a Command object to represent command execution; these classes will be examined later.

The HTTP/HTTPS protocol is used to fetch information from the server, it is a text-based line-oriented protocol where information is sent and received as ASCII text. The protocol uses a call-and-response communication sequence to handle requests where a request is sent to the server and a response is sent back in return, for example, when the user connects to the server via a browser, a GET request is sent, followed by optional lines to set the server and language:

```
GET / HTTP/1.1
Host: localhost

Accept-Language:en
```

(https://developer.mozilla.org/en-US/docs/Web/HTTP/Session)

The server responds with a response message; the first line is a status string with the HTTP version, a status number and status string, in this case the server is using HTTP version 1.0 and the request succeeded using

```
HTTP/1.0 200 OK
```

Following the status is a line that sets the type of data and character used in any documents, permitting multiple languages and alphabets

```
Content-Type: text/html; charset=utf-8
Server: MINISERVER
```

The document data is sent following these headers, the server makes use of two different headers stored as HTML within a String.

After the sign in page is submitted it sends data to the server USING the POST method, which sends data from the form entered by the user; processing the information with this method sends information using a series of fields of the format id=value delimited by the & character. An example is when the user enters their username and password if the username was test and the password was qwerty, then the post data would be user=test&password.

On success, the terminal page is sent to the browser; the page is split into two halves a top half that displays the text output, followed by and output and a bottom half that displays the input text box and a submit button. The page is divided into two halves to allow the page to use dynamically generated output created by commands, an example is when an internal command completes and displays a message, it will appear to be displayed on a page within a HTML area, with a text input box and submit button (Image 1)

On failure, it resends the sign in page with an error message, to allow the user to try again (Image 2)

HTML is a *stateless* protocol, where the previous requests have no effect on future requests. This affects how the server is designed in respect to keeping track of which stage it is at in the processing, because it must keep track of which step a connection currently is at using the HTTP referer metadata provided by the browser, which refers to the previous page that the user has visited is used to infer which step of processing it is currently at. For example, if there is no referrer data, then it is at the sign in page, otherwise the form on the page has been submitted and the username and password are being checked. This method of keeping track of the sign-in state only distinguishes between whether the user is currently entering information into the sign in form or have submitted the form, because the server only needs to know which of these states the user currently is in, so that once it has switched to having submitted the form it can authenticate the user and display the terminal interface if successful. However, to know whether it should authenticate the user or has already authenticated the user and display the terminal, it checks the post data submitted by the page. This method simplifies the state checking logic by separating the checking whether the user has attempted to sign in from processing the sign in request; because both pages are forms that submit post data to the server, it acts a common feature of the pages that varies depending on which state the server is in.

The user class represents a single user of the system, it stores information, such as username, password and user type and manages the users through its methods. It manages the authentication of users and management of user accounts. The authentication of users is an important part of the User class, it checks whether the user is who they say they are. It extracts the username and password form the post data submitted from the sign in form hashes the password and compares them with the lines stored in the passwd file, if they match if returns true, false otherwise. The password is encoded with a one-way hashing function that encodes the data in a way that makes it impossible to decrypt. The password can be checked, but not be revealed, protecting them from being stolen.

The management of adding, removing, and updating users by updating the user information stored as a series of colon-separated fields in a flat text file, this method was chosen because of the

simplicity of use and the low overhead. However, accessing the file is done in a sequential manner, inserting, and removing and updating information in a sequential-access file requires that iterating over all the other records in the file before updating the required record, which is slow and inefficient. In the case of removing and updating users, it must copy any non-relevant entries and the updated entry to a temporary file. This temporary file is then moved to the original passwd file.

Password file I/O uses the RandomFile class to access the file in any location in the file, it permits files to be both read and written, In the case of adding users, it must iterate over existing entries to check if the user already exists, if does not, a new file entry is appended to file.

Because the shell is a multi-user system, there is the possibility of multiple users adding, removing, or updating the passwd file. This may cause *race conditions* where simultaneous access to data may return incorrect results. There are two types of race conditions, *write after read* where one thread writes to data that another thread is due to read from, for example:

Thread A	Thread B
Write 1	Write 2
Read	

Thread A expects to read the value 1, but instead reads the value 2, which is incorrect.

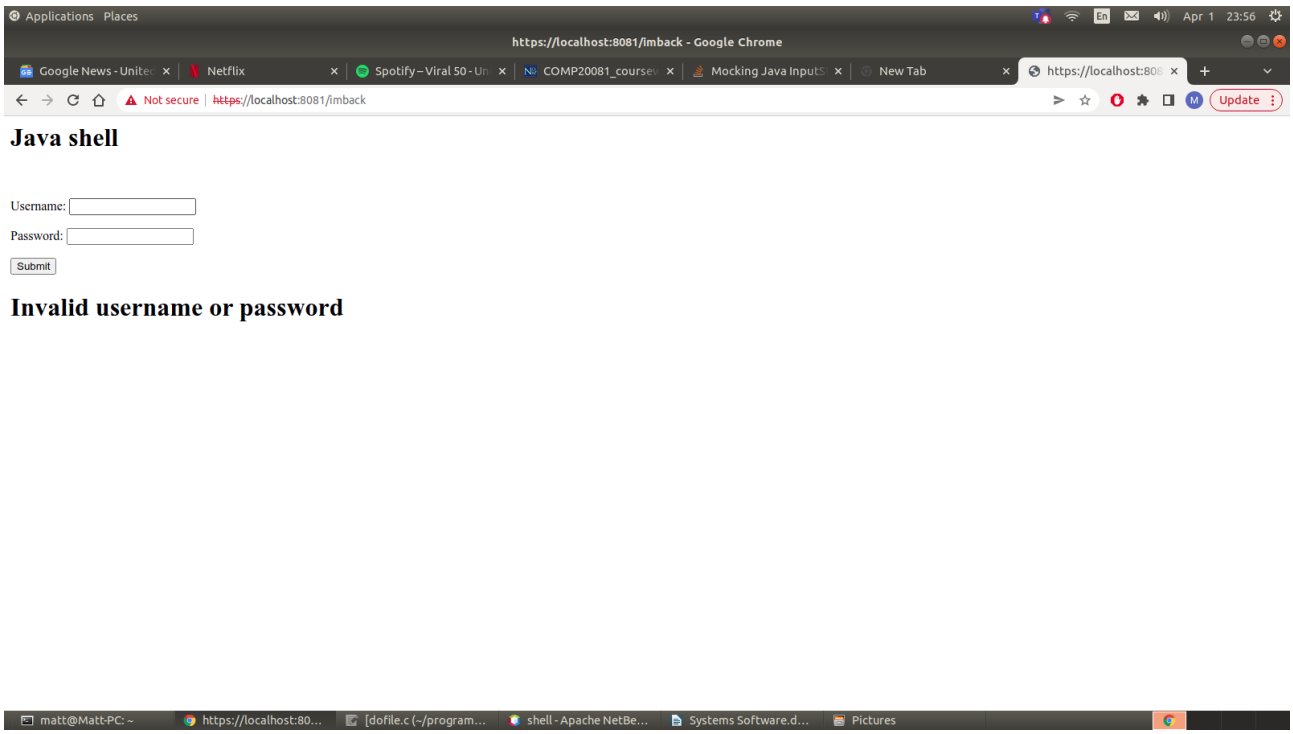
The second type of error is *written after read*, where the information read by one thread but is updated by another thread; the data read by the first thread will be incorrect:

Thread A	Thread B
Read X (1)	Write X (2)

In this example, thread A reads the value 1 from X and thread B writes the value 2 to X. Thread A will expect the value of X to be 1, but it has been modified……

To prevent these data hazards, access to the data must be serialized using a mutex to ensure that only one thread can access it at one time. A single mutex is created in the main thread, ensures that a single method of regulating access to the passwd file is used. The Mutex attempts to acquire access using mutex.acquire(), it either gains access or waits for access to be granted and afterwards releases the mutex to ensure that other threads can use the mutex.

The Command class handles execution of user commands, it contains two methods for executing user commands ExecuteCommand, which executes a user command, either internal or external and ExecuteExternalCommand, which creates processes to execute external commands. The execution of external commands was placed in a different method from ExecuteCommand to divide the responsibility of running external commands from internal commands.



Testing

Junit Testing

In the program we implemented a popular unit-testing framework called Junit 5. This allows us to write and run repeatable black-box tests which tested the functionality of each of the methods with several types of input and checked that outputs were as expected.

This ensures that the program can be tested to meet certain requirements and ensures that it works in many different scenarios. This ensures that during development of the program, whenever any new code is written, we can test that it is still doing the jobs needed, and that nothing has been broken. Every time new code is added, the tests should pass to ensure that the newly written code has been implemented correctly without any issues.

Test plan

ID	1	Description	Read input from user	
Test Type		Quantity/ **quality**	Success criteria	The text is read by program
Number of attempts	1			
List of equipment/requirements	Manual testing			
Setup instructions	Enter input into program			

Failure correction procedure	On failure, test with dummy input and check that the test works, try again with live input.	
Engineer/ technician	Derice McDonald	
Individual result	Data is successfully read from the input	
Test date	31/03/22	Result The text is read by the program

ID	2	Description	Read input from user	
Test Type		Quantity/ **quality**	Success criteria	Text is output by the program
Number of attempts	1			
List of equipment/requirements	Junit test to output text			
Setup instructions	Run Junit test for text output			
Failure correction procedure	Check socket and stream output declaration			
Engineer/ technician	Derice McDonald			
Individual result	Data is successfully read from the input			
Test date	31/03/22	Result The text is output by the program		

ID	3	Description	Read input from user	
Test Type		Quantity/ **quality**	Success criteria	Gets the current directory
Number of attempts	1			
List of equipment/requirements	Junit test			
Setup instructions	Run Junit test testShellGetCwd()			

Failure correction procedure	Check if signed in, has connection	
Engineer/ technician	Derice McDonald	
Individual result	Returns current directory	
Test date	31/03/22	Result Returns current directory

ID	3	Description	Test whoAmI command from terminal and GetUserName method	
Test Type		Quantity/ **quality**	Success criteria	Displays username
Number of attempts	1			
List of equipment/requirements	Junit test			
Setup instructions	Run Junit test TestWhoAmI			
Failure correction procedure	Check if signed in and user object has been initialized by SetUserName			
Engineer/ technician	Derice McDonald			
Individual result	Displays username			
Test date	31/03/22		Result Displays username	

ID	4	Description	Test cd command from terminal	
Test Type		Quantity/ **quality**	Success criteria	Changes directory
Number of attempts	1			
List of equipment/requirements	Manual input			
Setup instructions	Manual testing by input			
Failure correction procedure	Check if signed in and user object has been initialized by SetUserName			

Engineer/ technician	Derice McDonald	
Individual result	Changes directory	
Test date	31/03/22	Result: Changes directory

ID	5	Description	Test cd command with invalid directory	
Test Type		Quantity/ **quality**	Success criteria	Displays error message
Number of attempts	1			
List of equipment/requirements	Manual testing by input			
Setup instructions	Run Junit test TestWhoAmI			
Failure correction procedure	If no error message, check if test directory does not exist or user another directory name			
Engineer/ technician	Derice McDonald			
Individual result	Displays username			
Test date	31/03/22		Result Displays username	

ID	6	Description	Test directory virtualization	
Test Type		Quantity/ **quality**	Success criteria	Will not able to navigate outside home directory
Number of attempts	1			
List of equipment/requirements	Junit test			
Setup instructions	Run Junit test testShellChdir()			
Failure correction procedure	Check if not in subdirectory.			

Engineer/ technician	Derice McDonald	
Individual result	Displays error message	
Test date	31/03/22	Result Displays error message

ID	7	Description	Test: Valid login	
Test Type		Quantity/ **quality**	Success criteria	User can enter username and password and will display terminal.
Number of attempts	1			
List of equipment/requirements	Manual testing			
Setup instructions	Manually test with username and password			
Failure correction procedure	Check if username and password are valid			
Engineer/ technician	Derice McDonald			
Individual result	Displays terminal			
Test date	31/03/22	Result Displays terminal		

ID	8	Description	Test: Invalid login	
Test Type		Quantity/ **quality**	Success criteria	User cannot login
Number of attempts	1			
List of equipment/requirements	Manual testing			
Setup instructions	Manually test with invalid username and password			
Failure correction	Check if username and password are invalid			

procedure	
Engineer/ technician	Derice McDonald
Individual result	Displays “Invalid Username or password” message

ID	9	Description	Test: showDir command	
Test Type		Quantity/ **quality**	Success criteria	User can enter username and password and will display terminal.
Number of attempts	1			
List of equipment/requirements	Manual testing			
Setup instructions	Manually test by entering showDir command			
Failure correction procedure				
Engineer/ technician	Derice McDonald			
Individual result	Displays current directory			

ID	10	Description	Test: Checks whether the user can move files	
Test Type		Quantity/ **quality**	Success criteria	User can copy a file
Number of attempts	1			
Setup instructions				
Failure correction procedure	Check if file exists			
Engineer/ technician	Derice McDonald			
Individual	File is moved			

result	

ID	11	Description	Test: Checks whether the user can copy files	
Test Type		Quantity/ **quality**	Success criteria	File is copied
Number of attempts	1			
List of equipment/requirements	Manual testing by executing copy command			
Setup instructions	Sign in to shell.			
Failure correction procedure	Check if file exists			
Engineer/ technician	Derice McDonald			
Individual result	File is copied			

ID	12	Description	Test: Checks whether a superuser command can be run by admin users	
Test Type		Quantity/ **quality**	Success criteria	Admin user can run command
Number of attempts	1			
List of equipment/requirements				
Setup instructions				
Failure correction procedure	Check if admin user			
Engineer/ technician	Derice McDonald			
Individual result	Displays result of super command			

ID	13	Description	Test: Checks whether a standard command

			cannot be run by admin users	
Test Type		Quantity/ **quality**	Success criteria	Standard user cannot run command
Number of attempts	1			
List of equipment/requirements				
Setup instructions				
Failure correction procedure	Check if standard user			
Engineer/ technician	Derice McDonald			
Individual result	Displays error message			

ID	14	Description	Test: Checks whether a user can log out	
Test Type		Quantity/ **quality**	Success criteria	User logs out
Number of attempts	1			
List of equipment/requirements	User is signed in.			
Setup instructions				
Failure correction procedure	Check if signed in			
Engineer/ technician	Derice McDonald			
Individual result	Displays goodbye message and disconnects			

ID	15	Description	Test: Checks whether users can be added by admin users	
Test Type		Quantity/ **quality**	Success criteria	Admin user can add users

Number of attempts	1
List of equipment/requirements	
Setup instructions	
Failure correction procedure	Check if admin user
Engineer/ technician	Derice McDonald
Individual result	Displays New user *username* added.

ID	16	Description	Test: Checks if users cannot be added by standard	
Test Type		Quantity/ **quality**	Success criteria	Admin user can add users
Number of attempts	1			
List of equipment/requirements				
Setup instructions				
Failure correction procedure	Check if admin user			
Engineer/ technician	Derice McDonald			
Individual result	Displays New user *username* added.			

#	Name	Description	Tester	Acceptance criteria	Results	Output
16	Cannot remove users as ordinary user	Checks whether ordinary users can remove users	Derice McDonald	Displays error message and user is not removed	Pass – Doesn't non-admin users to remove users	Will get permission error when trying to use super command

ID	18	Description	Test: Checks wether the user can connect	
Test Type		Quantity/ **quality**	Success criteria	User can connect
Number of attempts	1			
Setup instructions	Open browser.			
Failure correction procedure	Check if file exists			
Engineer/ technician	Derice McDonald			
Individual result	Connected successful.			

Conclusion

A secure, remote shell was implemented that allowed secure, remote access to a computer systems and execute commands.

The program, with respect to the analysis and design matched the goals set in the requirements list. When evaluating whether the program fulfilled the goals in the analysis, development, and implementation, the initial goals for the project should be examined and compared with the requirements.Using the requirements list and the design and comparing them, it was found that the programs conforms with the functional requirments and the analysis and design put forth in the use case diagram, where the use cases in the diagram match the implementation in the program code, with the code being mostly efficient in the design.

The implementation of the program was a basic shell, it lacks many of the features, for example to or command history, conditional operations and output redirection to files that many other shells. The simplicity of a project is not a problem, because many simple command-line interpreters have been written. However, the implementation left a lot to be desired with respect to stability, for example, it often did not sign users in after adding a new users, because of bugs in the program affected updating the users file where blank lines were often errornously insered in the the user file.

Some of the features that were implemented, while functioning correctly were not implemented in the most efficient manner. The handling of user information was done, in a simple, but inefficient way, with the information stored in a flat text file. Removing and updating information was difficult and time-consuming with each entry being read, modified and copied to a temporary file which was copied back to the original file. This method is very inefficient where system with large number of users is only modifiying the information for a single user. A more efficient alternative would be to

use a flat database library such as MongoDB to store the user information.The second advantage of using a database library is that is abstracts the data storage from the rest of the program, with known code that has fewer bugs to cause problems.

The splitting of each user session into a individual threads contributed to the stability and efficiency of the program,

The testing of the program was done on a unit-test basis, with black box testing done on the inputs and outputs, but edge cases and corner cases were not covered in the testing, causing the previously mentioned bugs to not be detected. The testing should have been much more comprehensive to check every possible input and output from the functions to check for both the most probable and less probable bugs that could arise.

In conclusion, the program works as expected but requires a number of changes to make it a fully usable program.

Bibliography

Appendix

Images

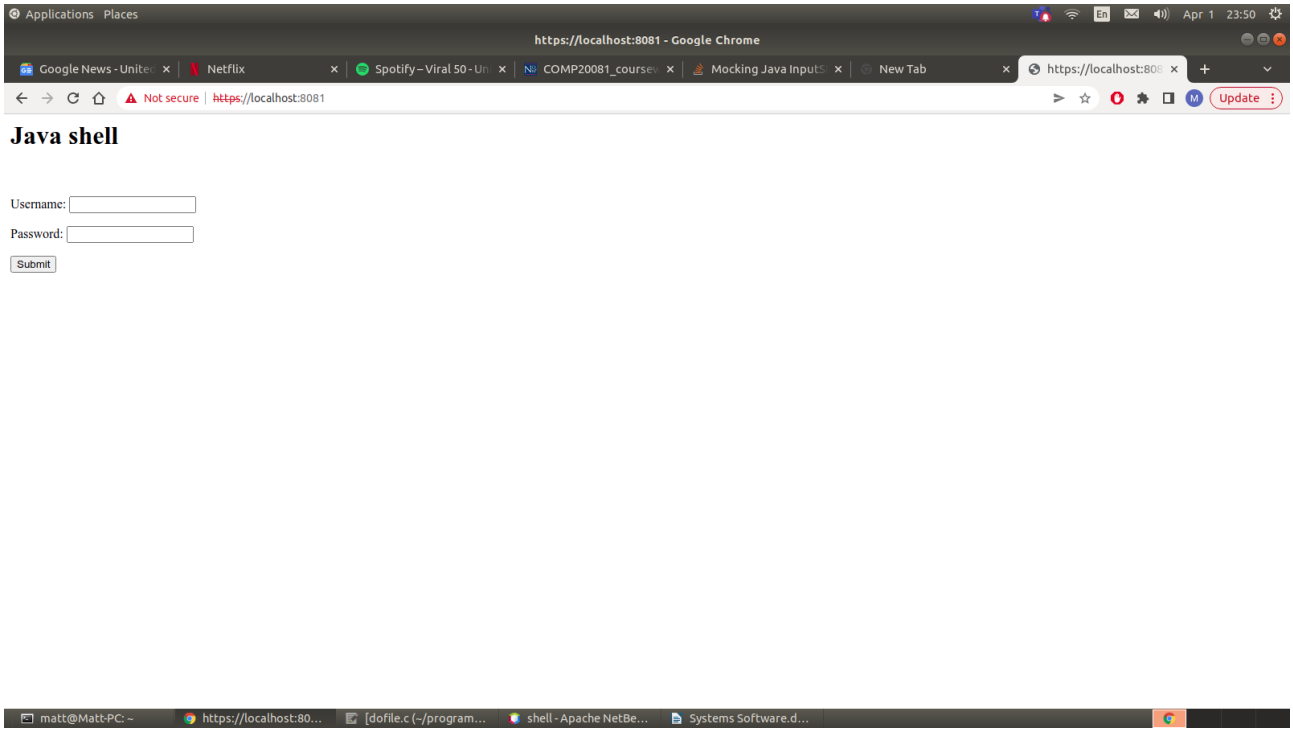


Image 1. Initial sign in page

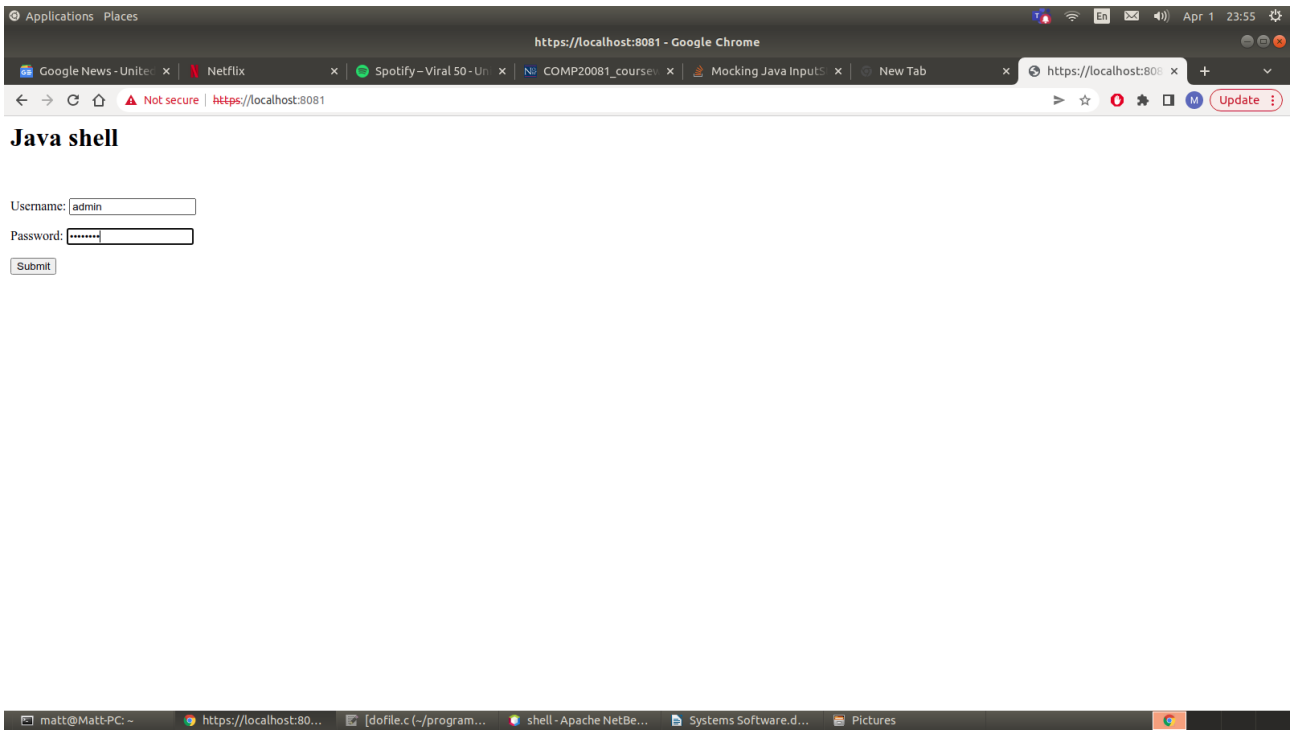


Image 2: Entering username and passwords

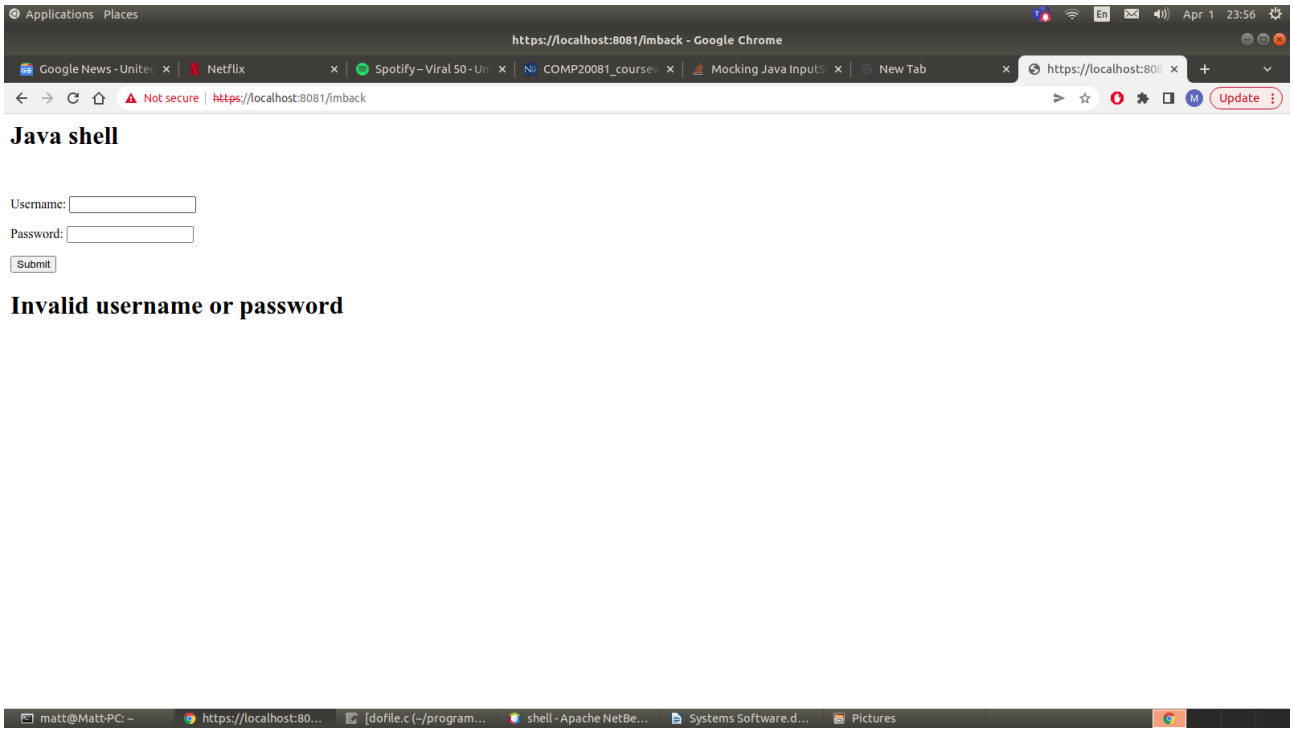


Image 3: Invalid sign in attempt

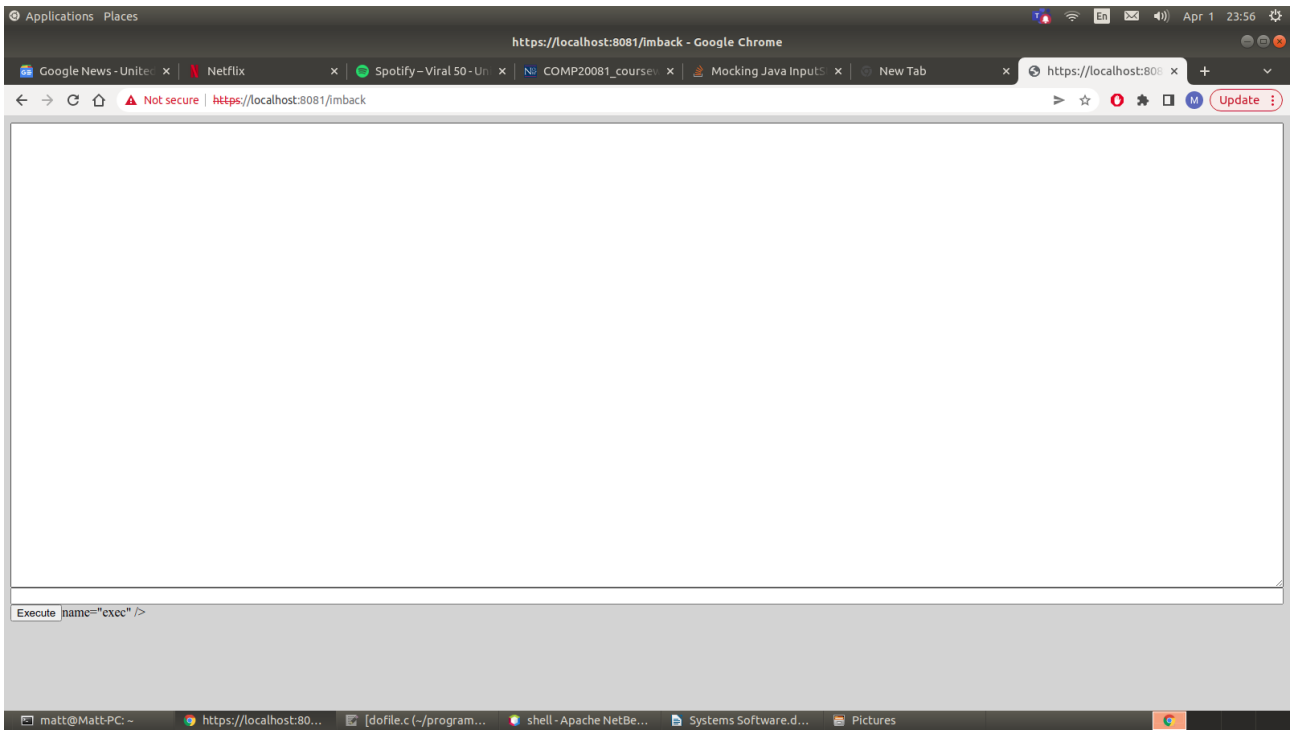


Image 4: Initial terminal interface

Images

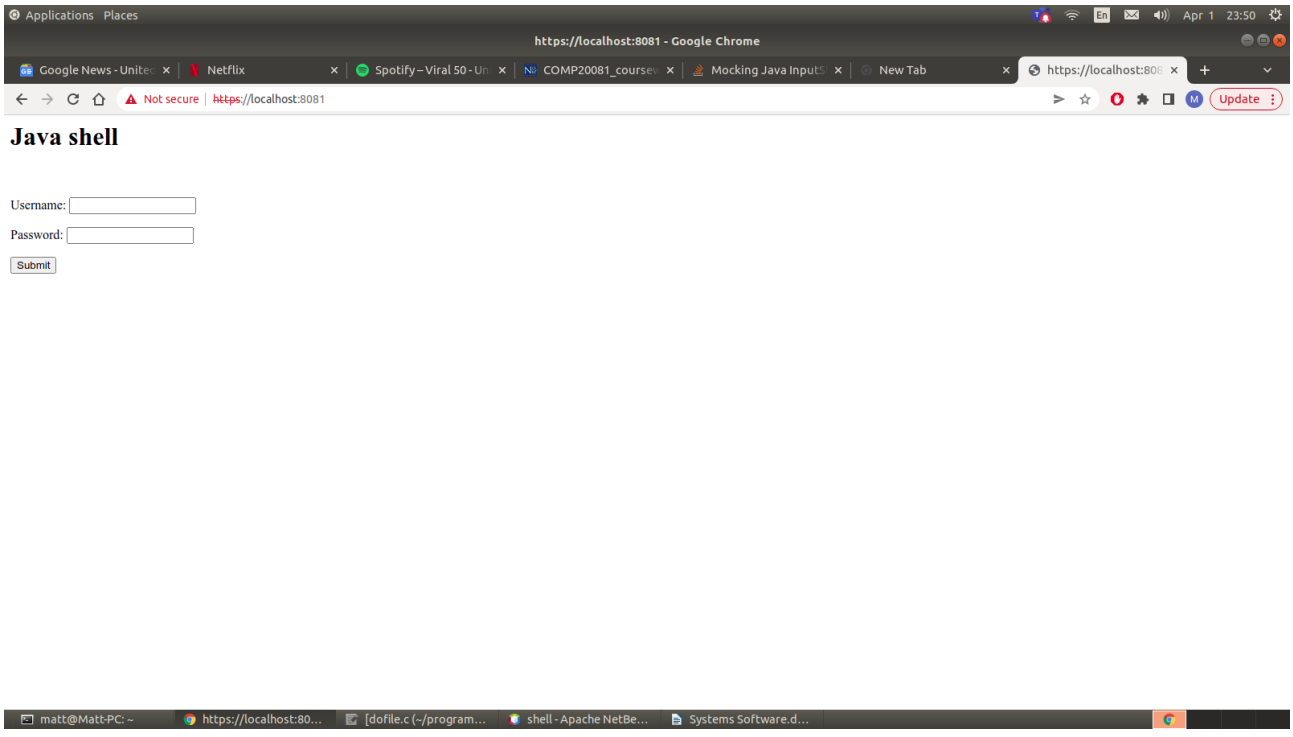


Image 1. Initial sign in page

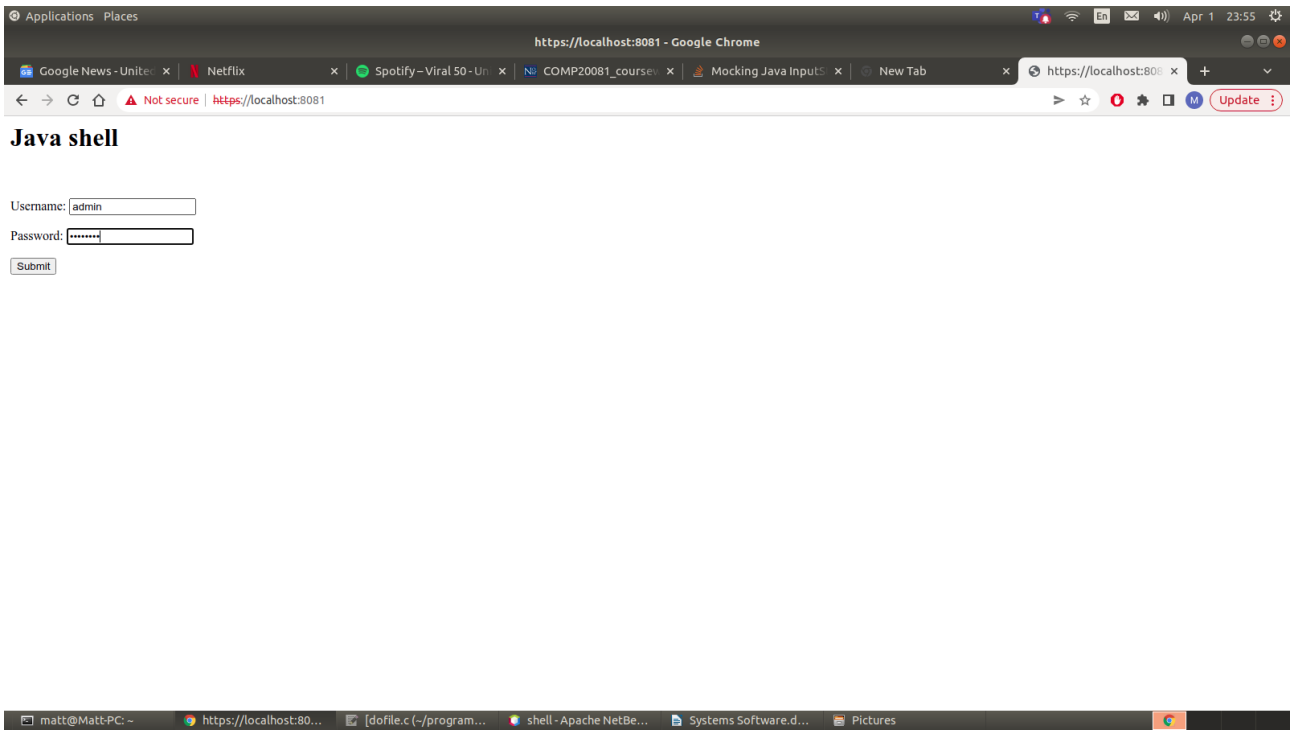


Image 2: Entering username and passwords

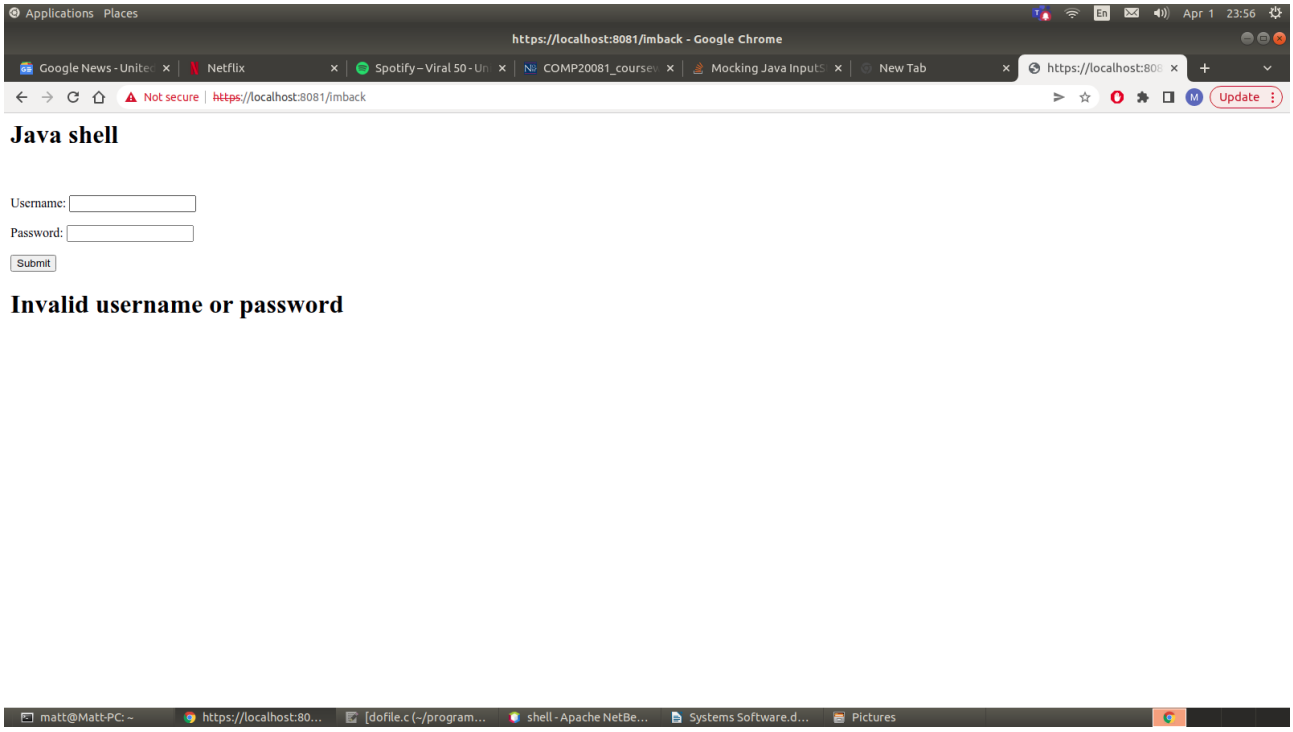


Image 3: Invalid sign in attempt

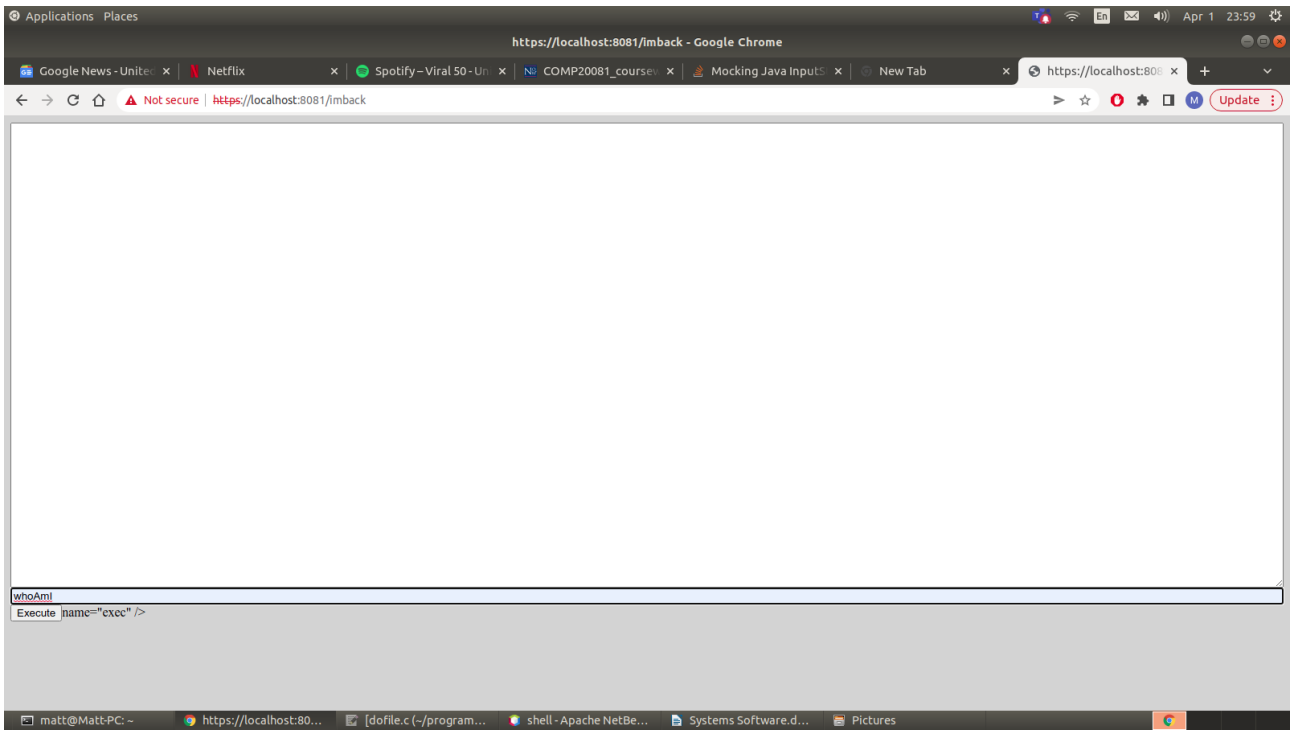


Image 4: Executing an internal command, whoAmI

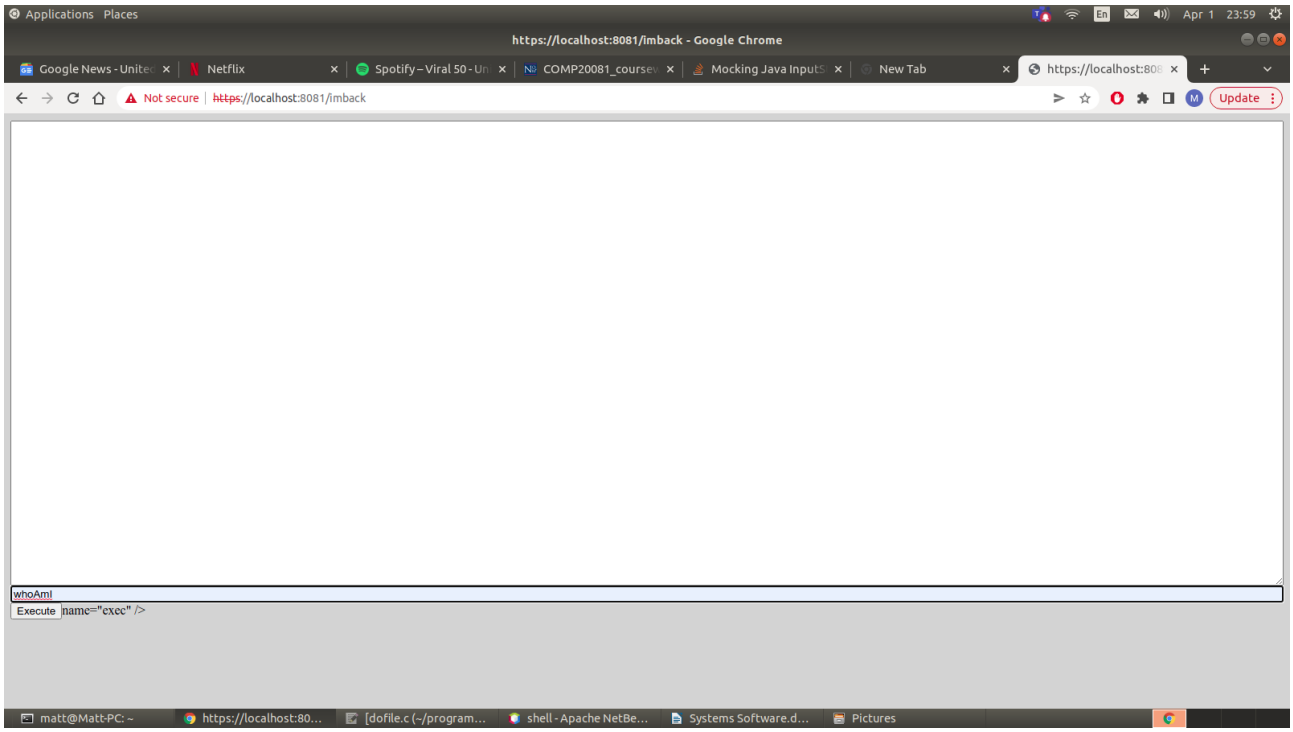


Image 5: Executing external commands ls -l