LockedMe.com
Virtual Key for Repositories

Prototype Specification Document

Simpilearn Full Stack Developer Course

Phase 1 Assessment

Developed by Sarah Phillips

April/May 2021

Table of Contents

GitHub Repository: https://github.com/saphilli/training-fullstack/tree/master/phase1

1. Objective

Develop a prototype of the application *LockedMe.com,* a virtual key for repositories. The prototype is a Java console application which provides the user with a platform for storing and performing operations on files.

2. Requirements

2.1. In Scope Functional Requirements

The minimum requirements set as part of project specification.

Welcome screen that displays: - Application name - Developer details - List of available options for user interaction
Functionality to parse user input to determine the operation to be performed.
Functionality to inform user of errors in their input.
Functionality to inform user of errors in an operation.
Exception handling to prevent application from exiting prematurely.
Option to retrieve and list all existing file names in ascending order.
Option to add a user-specified file to the application.
Option to delete a user-specified file from the application.
Option to search for a user-specified file.
Option to close the current execution context and return to the main context.
Option to close the application.

2.2. Out of Scope Functional Requirements

Requirements added to enhance usability.

Option to display list of available commands facilitated by the application.
Option to list all the existing directories in ascending order.
Option to add a user-specified directory to the application.
Option to delete a user-specified (non-empty or empty) directory from the application.
Option to change the current context to a user-specified directory.
Option to change the context to the parent directory of the current context.
Option to add a user-specified file and write to the created file.
Option to read and display the content of a user-specified file.
Option to search for a user-specified file and display the path of the found file.
Option to display list of available commands for user interaction.

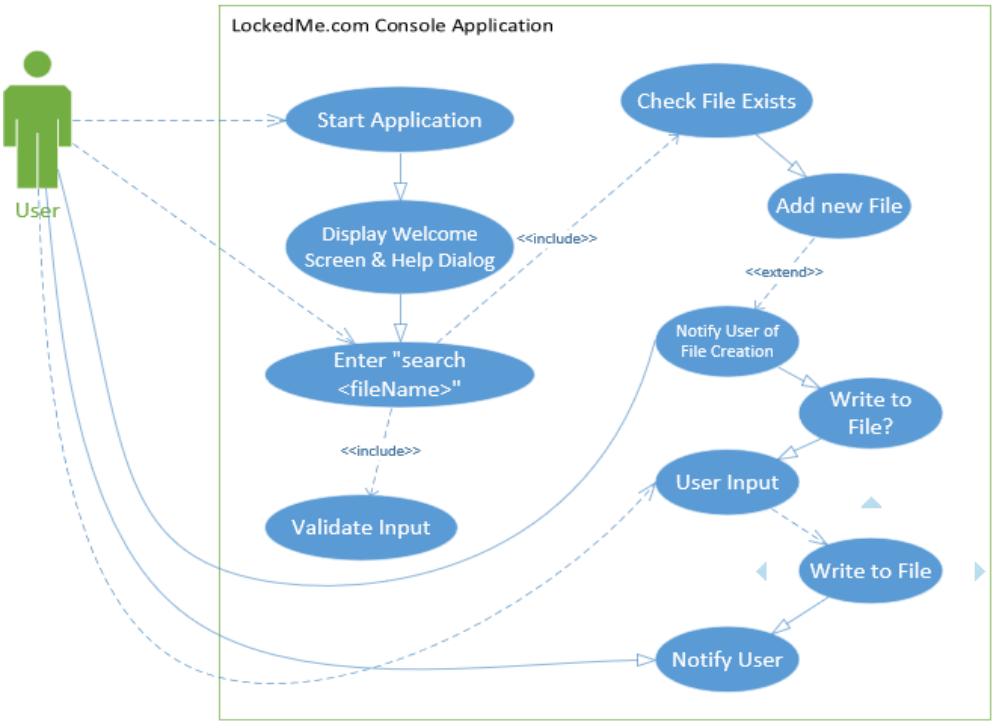
2.3. Development Requirements

Implement Unit Tests where applicable.
Use of the Scrum agile framework
Ensure all Unit Tests are passing.
Use of source control.
Implement logging.
Implement README document

3. Prototype Architecture

3.1. Key Use Cases

3.1.1. Adding a File



Actor: User

Entry Condition: User has installed the application

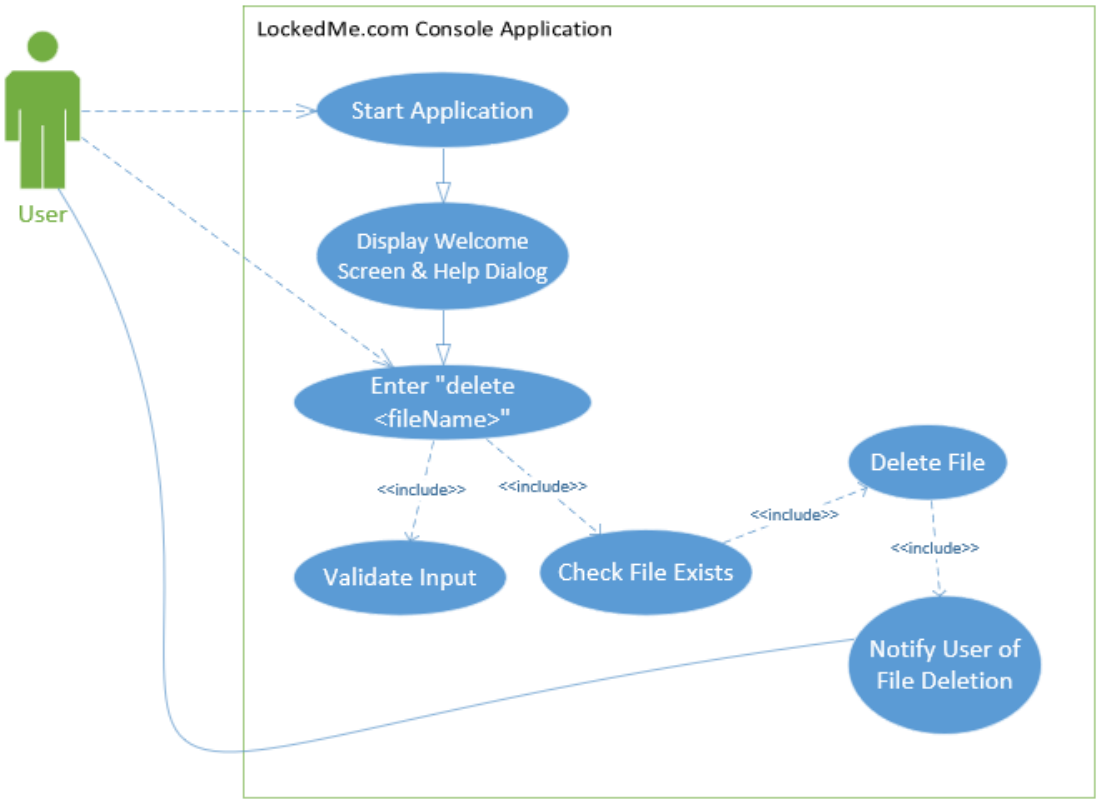
Normal Scenario:

1. User starts the application.
2. User enters the command “add” followed by the name of the file they want to add.
3. The application creates a new file and adds it to the file system.
4. The application prompts the User on whether they would like to add content to the file.
5. The User enters “Yes”.
6. The User enters input to be added to the file they created.
7. The application appends the User’s input to the file.
8. The application notifies the User that the file was successfully written to.

Error Scenario:

1. The file the User would like to create already exists in the application’s file system.
2. The already existing file cannot be overwritten, so a new file is not created.

3.1.2. Deleting a File



Actor: User

Entry Conditions:

- User has installed the application
- User has added at least one file to the application

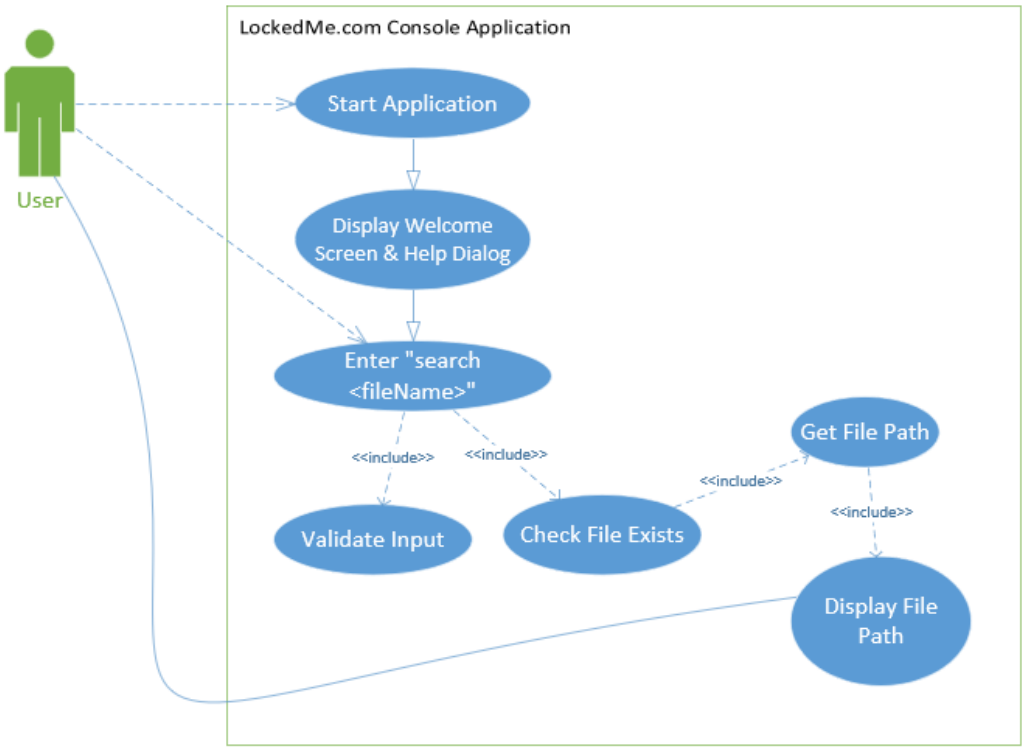
Normal Scenario:

1. User starts the application.
2. User enters the command “delete” followed by the name of the file they want to delete.
3. The application validates the User’s input.
4. The application checks if the file exists in the file system.
5. The application deletes the found file from the file system.
6. The application notifies the User that the deletion was successful.

Error Scenario:

1. The file the User would like to delete does not exist in the file system.
2. The application notifies the User that the file they tried to delete does not exist.

3.1.3 Search a File



Actor: User

Entry Conditions:

- User has installed the application
- User has added at least one file to the application

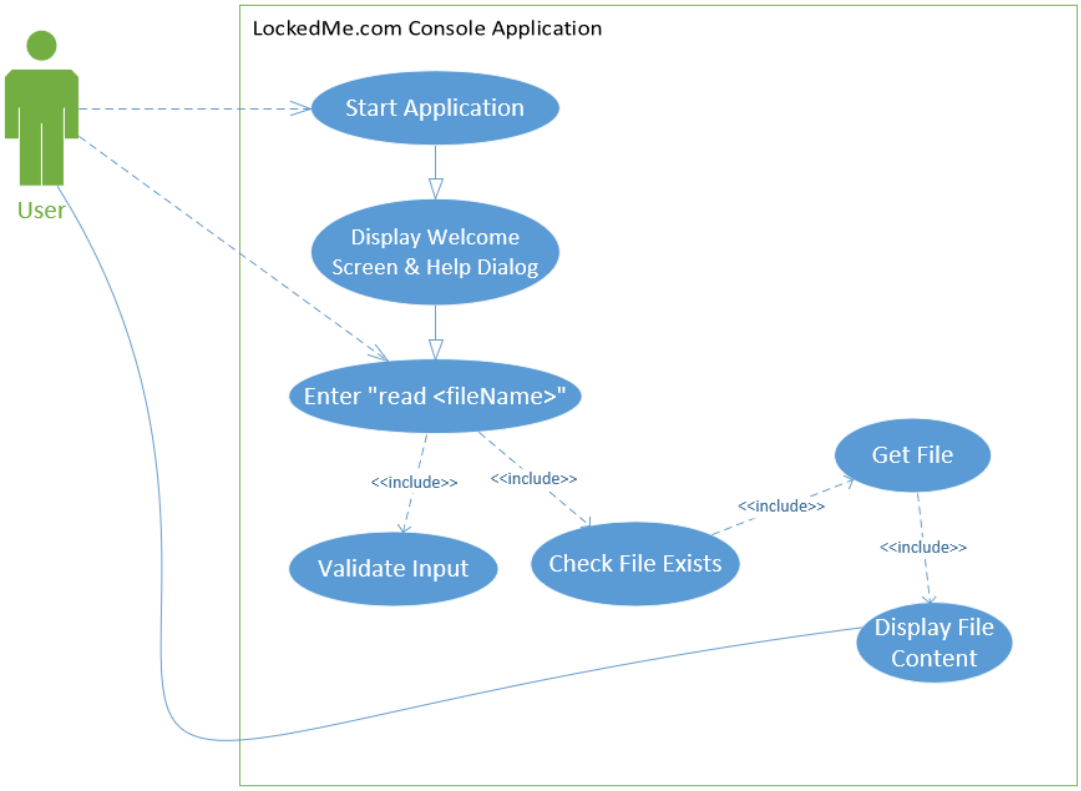
Normal Scenario:

1. User starts the application.
2. User enters the command “search” followed by the name of the file they want to delete.
3. The application validates the User’s input.
4. The application checks if the file exists in the file system.
5. The application gets and displays the path of the file to the User.

Error Scenario:

1. The file the User would like to search for does not exist in the file system.
2. The application notifies the User that the file they tried to search for does not exist.

3.1.4 Read Contents of a File



Actor: User

Entry Conditions:

- User has installed the application
- User has added at least one file to the application

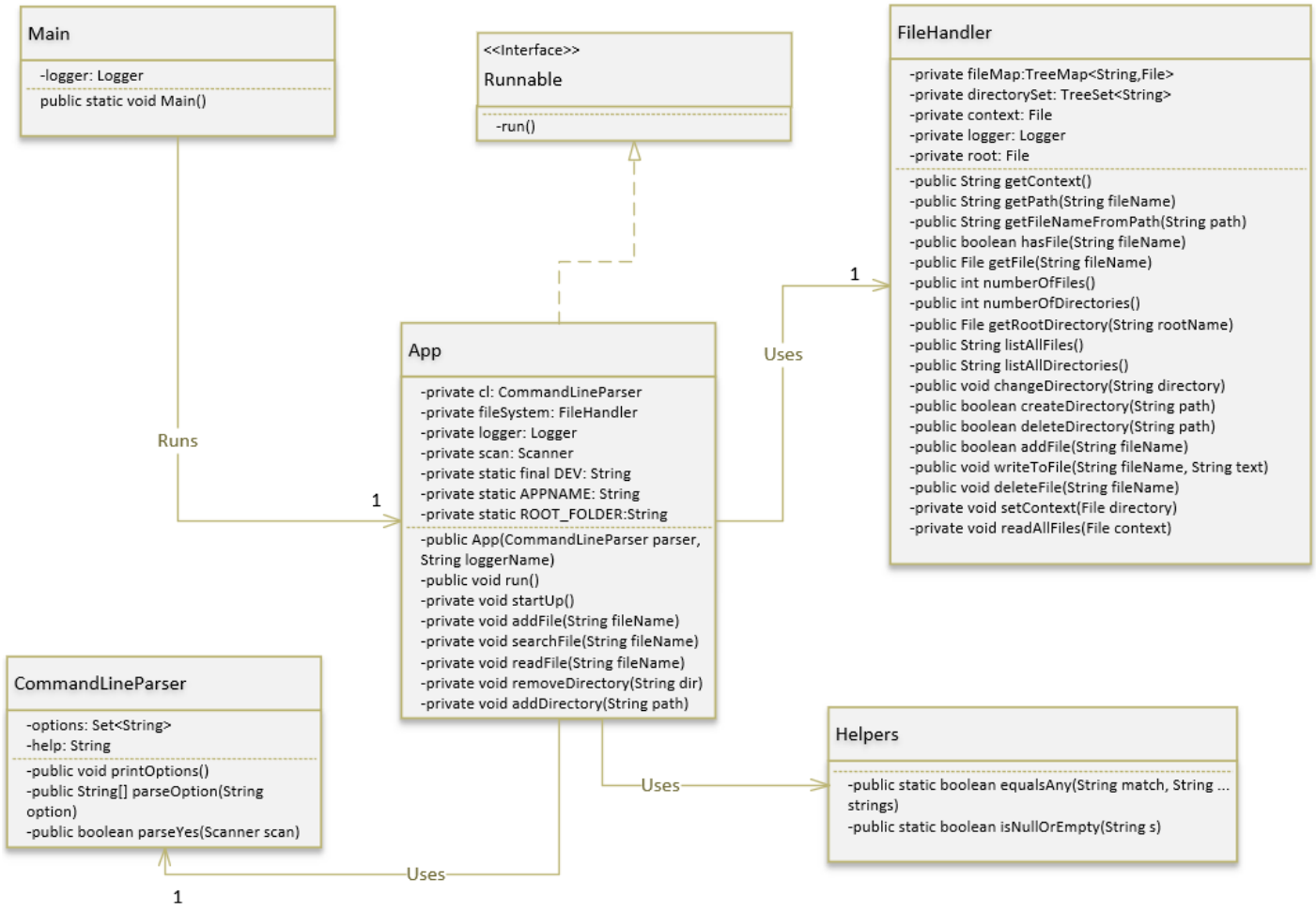
Normal Scenario:

1. User starts the application.
2. User enters the command “read” followed by the name of the file they want to read content from.
3. The application validates the User’s input.
4. The application checks if the file exists in the file system.
5. The application reads from the file.
6. The application displays the file contents to the User.

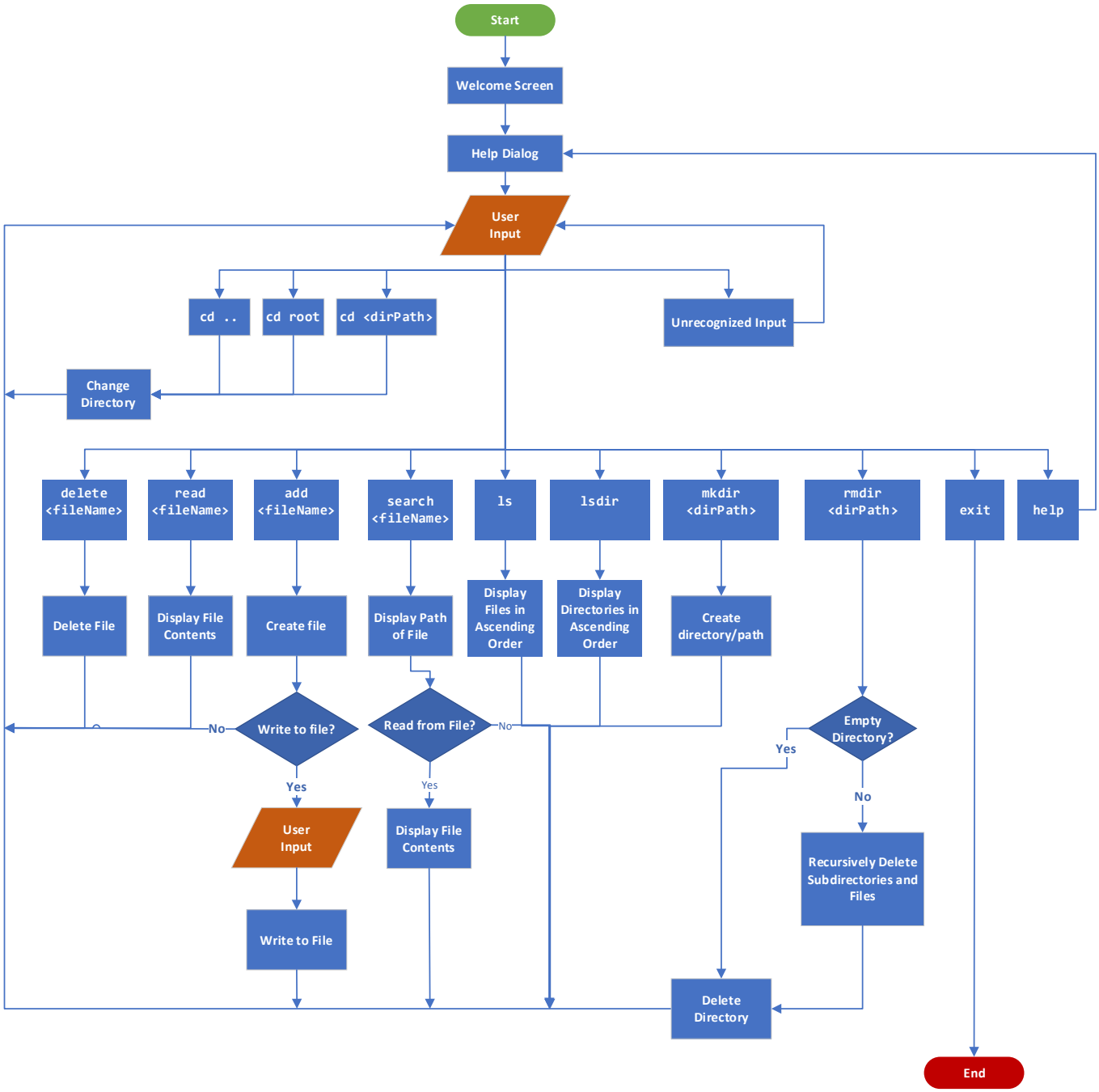
Error Scenario:

1. The file the User would like to search for does not exist in the file system.
2. The application notifies the User that the file they tried to search for does not exist.

3.2. Object Model



3.3. Application Flow



4. Sprint Planning

4.1. Breakdown of Stories and Corresponding Tasks

User Stories	Corresponding Developer Tasks
Welcome Screen **As a** User of LockedMe.com **I want** to be able to: 1. View a welcome screen when the application starts up 2. View a list of the available commands offered by the application **So that** I know: 1. The application has started up 2. What commands I can enter and the operations they perform	**1. Project Setup** **Acceptance Criteria:** The project is set up as a Maven project in Eclipse IDE. The project has a repository set up on Github and locally for source control. The project has a logger implemented. **2. Develop the Welcome Screen** **Acceptance Criteria:** The project has the App class that displays the Welcome Screen. A Main class calls the App class. The Welcome Screen displays the App name and Developer Details. It can display a list of commands available to the user.
User Command Input **As a** User of LockedMe.com, **I want** to be able to: 1. input commands to the application 2. And be aware of when I have entered an incorrect command 3. Enter an incorrect command without the app exiting **So that** I can perform various operations on my files with ease.	**3. Develop the Command Line Parser** **Acceptance Criteria:** The app accepts and validates the user input The app can accept and validate arguments to user inputted commands The app notifies the user if they entered an incorrect command. The application accepts a “help” command that displays all available commands to the user The application accepts an “exit” command which terminates the application.
Context Operations **As** a User of LockedMe.com, **I want** to be able to: 1. Be able to switch the context of the application to another directory **So that** I can easily navigate the file storage system and add/remove/search files.	**4. Develop the FileHandler** **Acceptance Criteria:** The FileHandler class gathers all existing files and directories in the app on startup. The class tracks context (the current directory) The class allows the user to switch the context to another directory. The app always displays the current context to the user.

Directory Operations **As a** User of LockedMe.com **I want** to be able to: 1. add a directory to the application 2. delete a directory from the application 3. see a list of all directories and subdirectories in ascending order **So that** I can add files to different directories	**5. Add Directory Operations to FileHandler** **Acceptance Criteria:** The FileHandler class allows the user to perform operations on directories. The user can create a new directory in the current directory. The user can delete a nonempty directory or subdirectory. The user can view a list of all directories in ascending order.
File Operations As a User of LockedMe.com I want to be able to: 1. add files to the app 2. delete files from the app 3. search for a file in the app 4. view the contents of a file in the app 5. see a list of all files in ascending order So that I can use the application as a platform for storing my files.	**6. Add File Operations to FileHandler** **Acceptance Criteria:** The FileHandler class allows the user to perform operations on files. The user can add a file to the current directory and write content to it. The user can delete a file. The app can read from an existing file and display the contents to the user. The app can search for a user specified file and display the path to the user if found. Else it displays a file not found message.
README As a User of LockedMe.com: I want to be able to view a README for the project So that I know how to use the application	**7. Complete README** **Acceptance Criteria** The README details: - How to run the application - The available commands and the operations they perform

For each Developer task, sufficient manual/unit tests and documentation are also completed, where appropriate.

4.2. Sprint Roadmap

Currently all tasks outlined below have a status of COMPLETE.

Task Board

Task ID	Task Name	Related User Story	Assigned To	Size	Sprint
1	Project Setup	Welcome Screen	Sarah Phillips	2	1
2	Develop the Welcome Screen	Welcome Screen	Sarah Phillips	3	1
3	Develop the Command Line Parser	Context Operations	Sarah Phillips	5	1
4	Develop the FileHandler	Context Operations	Sarah Phillips	5	1,2
5	Add Directory Operations to FileHandler	Directory Operations	Sarah Phillips	5	2
6	Add File Operations to FileHandler	File Operations	Sarah Phillips	5	2
7	Complete README	README	Sarah Phillips	1	2

Sprint 1 (Monday 3rd – Friday 7th May)

Monday 3rd	Tuesday 4th	Wednesday 5th	Thursday 6th	Friday 7th
Project Setup	Develop the Welcome Screen	Develop the Welcome Screen	Develop the CommandLineParser	Develop the FileHandler
		Develop the CommandLineParser		

Sprint 2 (Monday 10th – Friday 14th May)

Monday 11th	Tuesday 12th	Wednesday 13th	Thursday 15th	Friday 16th
Develop the FileHandler	Add Directory Operations	Add File Operations to FileHandler	Add File Operations to FileHandler	Complete README
Add Directory Operations				

5. Design and Implementation

5.1. Technologies

Technology		Reasons for Use
Language	Java	1. Object-oriented language so allows for modularity and reusable code. 2. Modularity means code is easier to test and scale. 3. Platform independency allows the app to be run on Windows, Mac or Linux and hence reach more users
Framework	Maven	1. Easy addition and management of external libraries and frameworks through pom file. 2. Can build project into jar or war 3. Fast project setup and test project setup.
Testing Framework	JUnit 4	Allows for Parameterized Tests through JUnitParams package, reducing boilerplate test code.
IDE	Eclipse	1. Free open-source software, hence reducing costs. 2. Simple plugin functionality, e.g. SonarLint 3. Intuitive Debugging interface 4. Code coverage reporting with Unit Tests.

5.2. Core Concepts

5.2.1. Data Structures

FileHandler Class

TreeMap<String,File> fileMap: Allows the `FileHandler` to keep track of the files the application contains at any point in time. Each entry in `fileMap` has a key of the `fileName` and value of the corresponding `File` object. The use of `TreeMap` allows for fast file searching of O(logN) complexity. The file search operation consists of a simple lookup using the file name entered by the user and returning the `path` field of the corresponding `File` object if found. Although this performance for searching isn't as fast as it would be with a `HashMap`, the main advantage of the `TreeMap` is clear when we need the files sorted in ascending order.

Getting the files in ascending order is trivial since `TreeMap` keeps entries in ascending order of keys regardless of insertion order. This feature in particular makes `TreeMap` an ideal data structure for holding the files, as this operation is simply done by getting the `KeySet` of the `TreeMap`. Overall, this is an O(N) operation since each `fileName` key needs to be iterated over to add to the final String of files displayed to the user.

TreeSet<String> dirSet: Allows the `FileHandler` to keep track of the directories the application contains at any point in time. Each entry in the set is the path of a directory. Since the data structure is of a Set type, it allows the app to ensure that duplicate directories are not created by the user. As with `TreeMap`, the `TreeSet` type also maintains ascending order of elements and hence listing the directories in ascending order is just as trivial.

CommandLineParser Class

HashSet<String> options: The `CommandLineParser` stores each command that is available to the user as string entries to the `options` HashSet. This allows for optimal performance when the CommandLineParser validates the command entered by the user. Validation is done through a HashSet lookup on `options` using the string entered by the user, an O(1) operation. If the lookup key doesn't have a corresponding value in the `HashSet`, the app quickly knows the user entered incorrect input.

Another advantage of `HashSet` is that it ensures all elements are unique, which is ideal for storing unique commands.

Basic data structures and objects such as `Array` and `StringBuilder` are also used throughout the application.

5.2.2. Algorithms

FileHandler Class

Recursive Directory Deletion: Since the File object does not allow directories to be deleted unless they are empty, an algorithm was added to delete files and subdirectories recursively from the directory to be deleted.

```
public boolean deleteDirectory(String path) throws IOException {
    var directory = new File(path);
    if(directory.isDirectory()) {
      File[] files = directory.listFiles();
      if(files != null) {
        for(File file : files) {
          var deleted = deleteDirectory(file.getPath());
          if(!deleted) return false;
        }
      }
    }
    else if(directory.isFile()) {
        var fileName = directory.getName();
        fileMap.remove(fileName);
    }
    Files.delete(Paths.get(path));
    directorySet.remove(path);
    return true;
}
```

5.2.3. OOP Concepts

Encapsulation

One of the principles of Object-Oriented Programming is encapsulation. Use of this principle can be seen in the source code of the application. An example of this is in the `FileHandler` class, which holds the private fields `fileMap` and `directorySet`. These are assigned private access modifiers to restrict the operations that can be done to these fields. Hence, they cannot be manipulated by outside objects directly and only through the provided public methods in the class. This preserves the state of the file system of the application. The App class can call the public methods of the `fileHandler` class to change the state of the file system, such as `addFile` and `deleteFile`.

This functionality is especially important with regards to the `context` field of the FileHandler class. This is also a private field and can be changed by external objects through the public `changeDirectory` method in the `FileHandler` class. This public method validates the change of context before updating the private field and therefore the state of the file system.

Abstraction

Similarly to encapsulation, the principle of Abstraction involves using objects to hide the implementation details of certain functionality from the classes that use the functionality. An example of this principle in the application is the `CommandLineParser` class. This class abstracts away the implementation of parsing and validating user input from the `App` class. By having this dedicated class for parsing user input, the `App` class code is less convoluted, and the functionality is easier to test.

The `FileHandler` class also abstracts away the implementation of the file system for the application and the operations it offers.

Inheritance

Some inheritance is seen in the application in the `App` class, which implements the `Runnable` interface through the `run` method. Although the application does not make use of multithreading at this time, this implementation allows for such functionality to be added later. The `App` class could be changed to run in multiple instances on different threads.

5.3. External Packages

Package	Object/Class
java.util	Scanner Set SortedSet HashSet TreeSet SortedMap TreeMap logging.Logger logging.Level
Java.lang	.Runnable
java.nio.file	.Files .Paths .Path .DirectoryNotEmptyException .DirectoryNotFoundException .NoSuchFileException .FileAlreadyExistsException
java.io	.File .FileInputStream .FileWriter .DataInputStream .FileNotFoundException .IOException
org.junit	.runner.RunWith .Test .Before .After
junitparams	.Parameters .JUnitParamsRunner

5.4. Source Code

GitHub Repository: https://github.com/saphilli/training-fullstack/tree/master/phase1

Project source code is located under ../phase1/LockedMe/src/main/java/com/

Unit tests are located under .../phase1/LockedMe/src/test/java/com

5.5. Final Application Screenshots

Welcome Screen

```
May 20, 2021 1:06:43 P.M. com.FileHandler <init>
INFO: Root directory is located at C:\Users\sarah_phillips\training\full-stack\phase1\LockedMe\root
May 20, 2021 1:06:43 P.M. com.Main main
INFO: Starting application...
May 20, 2021 1:06:43 P.M. com.App startUp
INFO: Application running

Welcome to LockedMe
Developed by: Sarah Phillips

Available commands:

ls                  Lists all files in the application in ascending order.
lsdir               List all directories and subdirectories.

cd <file_name>      Change context to the specified directory.
cd ..               Change context to parent directory.
cd root             Change context to root directory.

mkdir <path>        Creates new subdirectory(s) in the current directory.
rmdir <name>        Deletes a directory if it exists in the current directory and all of it's subdirectories and files.

search <file_name>  Searches for a file and displays the path if the file exists.
add <file_name>     Adds the specified file to the current directory.
read <file_name>    Shows the content of the specified file if it exists.
delete <file_name>  Deletes the specified file from the current directory.

exit                Exits the program.
help                Display available commands.


\root>
```

List Files and Directories

```
\root>  ls
3 files:
     2.txt
     d.txt
     e.txt
     fileWithContent.txt
     h.txt
     stuff.txt
     test1.txt
     x.txt

\root>
```

```
\root>  lsdir
5 directories:
root
root\folder1
root\folder4
root\folder4\folder7
root\folder5

\root>
```

Create Directory and Change Directory

```
\root\folder4>  mkdir myFolder
The path myFolder was created

\root\folder4>  lsdir
6 directories:
root
root\folder1
root\folder4
root\folder4\folder7
root\folder4\myFolder
root\folder5

\root\folder4>
```

```
\root>  cd folder4\folder7

\root\folder4\folder7>  cd ..

\root\folder4>  cd root

\root>  cd ..
\root has no parent directory.

\root>
```

Add New and Write to New File

```
\root\folder4>  add myFolder\myFile
File was successfully added. Path: root\folder4\myFolder\myFile
Write to the file? (Y/N)
Y
Type some words, then when you're happy press "Enter"
Hello World!
Successfully wrote to the file.

\root\folder4>  read myFile
Reading from the file myFile....
Hello World!

\root\folder4>  add myFile
May 20, 2021 2:25:10 P.M. com.FileHandler addFile
WARNING: myFile already exists in the application at root\folder4\myFolder\myFile and cannot be overwritten
Something went wrong when creating the file.

\root\folder4>
```

Search for File

```
\root\folder4>  search myFile
myFile was found at root\folder4\myFolder\myFile
Read the file? (Y/N)
Y
Reading from the file myFile....
Hello World!

\root\folder4>
```

```
\root>  search aNonExistingFile
The file aNonExistingFile does not exist.

\root>
```

Delete File and Nonempty Directory

```
\root\folder4>  search 2.txt
2.txt was found at root\2.txt
Read the file? (Y/N)
N

\root\folder4>  delete 2.txt
Successfully deleted the file 2.txt

\root\folder4>  ls
8 files:
     d.txt
     e.txt
     fileWithContent.txt
     h.txt
     myFile
     stuff.txt
     test1.txt
     x.txt

\root\folder4>
```

```
\root>  lsdir
6 directories:
root
root\folder1
root\folder4
root\folder4\folder7
root\folder4\myFolder
root\folder5

\root>  rmdir folder4
Successfully deleted directory folder4

\root>  lsdir
3 directories:
root
root\folder1
root\folder5
```

Help and Exit

```
\root>  randomCommand
Command not recognized: randomCommand
Enter 'help' to view list of possible commands

\root>  help

Available commands:

ls                  Lists all files in the application in ascending order.
lsdir               List all directories and subdirectories.

cd <file_name>      Change context to the specified directory.
cd ..               Change context to parent directory.
cd root             Change context to root directory.

mkdir <path>        Creates new subdirectory(s) in the current directory.
rmdir <name>        Deletes a directory if it exists in the current directory and all of it's subdirectories and files.

search <file_name>  Searches for a file and displays the path if the file exists.
add <file_name>     Adds the specified file to the current directory.
read <file_name>    Shows the content of the specified file if it exists.
delete <file_name>  Deletes the specified file from the current directory.

exit                Exits the program.
help                Display available commands.



\root>  exit
May 20, 2021 3:35:48 P.M. com.App run
INFO: Exiting the application...
```

User Input Handling

```
\root>  add
Not enough arguments entered for command: add
Enter 'help' to view list of possible commands

\root>  add file1 file2
Too many arguments entered for command: add
Enter 'help' to view list of possible commands

\root>  ls allfiles
Too many arguments entered for command: ls
Enter 'help' to view list of possible commands

\root>  mkdir
Not enough arguments entered for command: mkdir
Enter 'help' to view list of possible commands

\root>
```

6. Conclusion

6.1. Unique Selling Points

Context Switching: The application provides an intuitive interface for switching between directories that takes similar commands to a regular Command Line Interface. This functionality allows the user to specify where they add a file in a way that is familiar to most users. The user can change the context to a subdirectory, walk back to the parent directory by passing `cd ..`, or switch directly to the root directory with `cd root`. The current directory is always shown to the user in a form similar to a regular CLI (`>root\` if the context is the root directory for example). Intuitive navigation is a unique selling point is this application.

Writing Content to a New File: When adding a new file, the user is asked if they would like to write some content to the file. If yes is entered, the user can add to the file directly from the application's interface and save it. This saves the user having to open the file themselves to write to it.

Searching for and Reading from a File: The application allows the user to search for a particular file and get the path to it. In addition to this, the user can also search or read any file regardless of the current directory they are in. The application can display the contents of any file the user specifies if it exists.

Non-Empty Directory Deletion: The user can delete directories that contain contents. This functionality saves the user from having to traverse all the subdirectories and delete their contents.

Directory Creation: The application provides functionality for the user to just create a directory through `mkdir`, rather than limiting the user to only being able to create a directory when also creating a file (through its path). The user can create directories with subdirectories by simply passing a path. The application is also strict in that directories with the same path cannot be created.

Help option: If the user is unsure of what commands they can enter or the syntax for a command, they can enter the `help` command. This will list out each command, any arguments the command requires and a description of what the command does. The application will also remind the user of the `help` command if they enter an unrecognized command.

Logging: The application uses a logger to log information about any Exceptions thrown or other issues. This provides helpful information to the user about what went wrong during an operation at what point in the application.

6.2. Further Enhancements

File Editing: Allowing the user to append to or overwrite files directly from the application interface would add additional value to the user. In the current version, this can only be done when adding a file to the application.

File Opening: Allow the user to open a file directly from the application interface, providing them with more freedom to edit the file.

Get Files by Criteria: Allow the user to list the files in descending order, directory structure, created date, file type, etc.

Get File Metadata: Providing a command to get specific metadata about the file such as creation date, file type, user it was created by, etc.

User Authentication: Adding user functionality so different users running the application on the same machine can have different instances of the application file systems. This would also provide security for users.

File Encryption: Providing additional value to the user through secure file storage by encrypting the files created through the application. This could be combined with user authentication to ensure that only authorized users can read specific files.

Pretty Print Directories: Increasing the visualisation of the file system structure by displaying the files and directories in the tree-like format.