



JW01: Data Forensics Tool for Investigating Subjects’ Suspicious Cloud Activities

By:

Ryan O’Connor – 17209382

Supervised By:

Jacqueline Walker

Submitted to the University of Limerick in partial fulfilment of the
requirements for:

Bachelor of Science in Mobile Communications & Security

(Cyber Security and IT Forensics)

Department of Electronics & Computer Engineering

University of Limerick

Mar19 2021

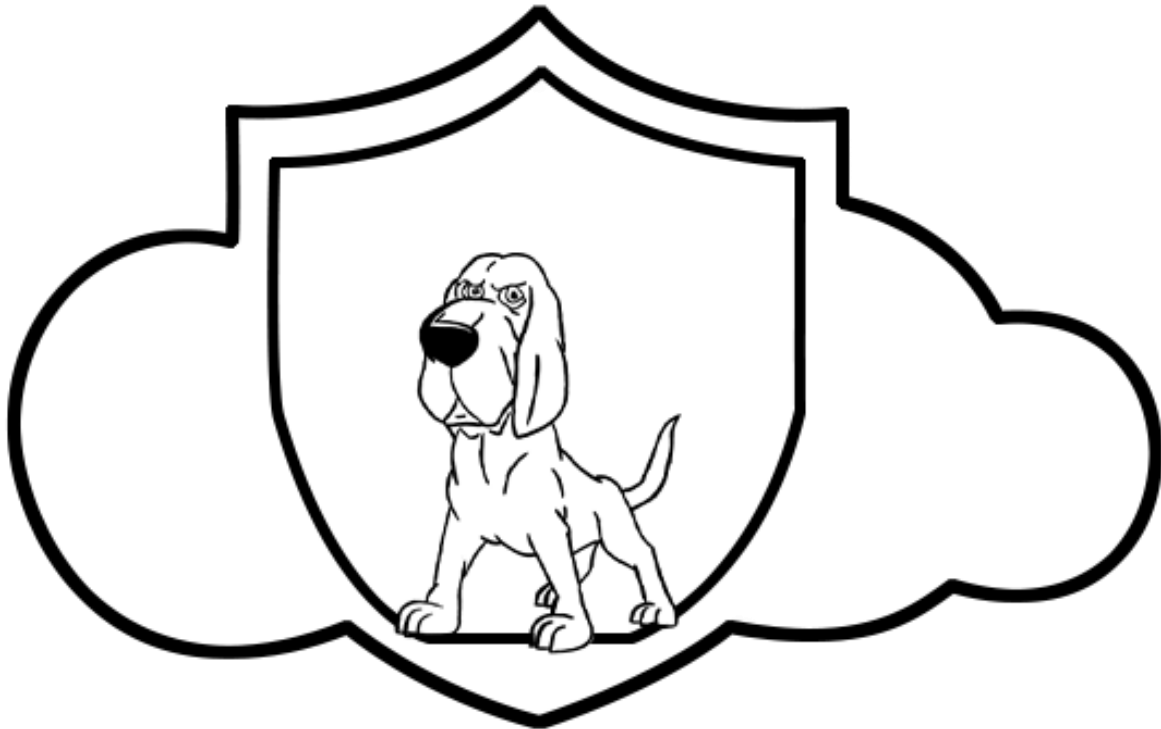


Figure 1: Cloud-topsy Logo

Cloud-topsy

Abstract

The issue of **forensics** in the field of **cloud computing** is paramount in any organisation. It is well established that cyber security **specialists** spend **countless hours** determining the use of cloud computing when investigating cases. *Cloud-topsy* aims to bring an **easy-to-use software** which can quickly **identify** and **provide evidence** that cloud storage platforms were used. Specifically, it investigates **remnants** left behind on the suspect's physical machine. This forensic analysis is fully **automated** and **report** data can be **searched**. This data can then be aggregated into different case groups. *Cloud-topsy* is designed to assist Investigators and their teams in the process of **analysing** and **recording** fragments from a large volume of disk images. *Cloud-topsy* could become an **essential time-saving tool** in forensic laboratories worldwide.

This documentation will explain what *Cloud-topsy* is, which techniques it uses for forensic analysis, how the setup is configured and how it is used.



Acknowledgement

Foremost, I would like to thank my supervisor Dr. Jacqueline Walker for providing her invaluable guidance, comments, and suggestions throughout the course of the project. I could not have imagined having a better advisor and mentor for this project.

I would also like to acknowledge the support of my friends, both in terms of technical aspects and moral support. Whenever I needed someone to give me a little guidance, they were there to support me.

Finally, I would like to thank my family, who were there to support me with countless cups of coffee and putting up with the aggressiveness of my keyboard clicking in the early hours of the morning.



Declaration of Authorship

I, Ryan O'Connor, declare that this report, titled "JW01: Data Forensics Tool for Investigating Subjects' Suspicious Cloud Activities (Cloud-toppsy)" and the project described within are my own.

I confirm that:

- This work was done as part fulfilment of the requirements of the Bachelor of Science in Mobile Communications and Security (Cyber Security and IT Forensics).
- Where I have used or consulted the published work of others, it is clearly stated. Apart from the consultation of those sources, the work is entirely my own.
- I have acknowledged all sources used for this work.
- This report has not previously been submitted or published before submission.

Signed: Ryan O'Connor

Date: 19/03/21



Table of Contents

Abstract.....	2
Acknowledgement.....	3
Declaration of Authorship	4
Table of Contents	5
Table of Figures	8
Table of Tables.....	9
Table of Code Extracts	10
Chapter 1. Introduction & Project Outline	11
1.1 Project Description	11
1.2 Project Rationale.....	11
1.3 Theory	12
1.4 Literature Review	14
Chapter 2. Similar Systems.....	17
2.1 EnCase Forensic.	17
2.1.1 Overview	17
2.1.2 Functionality	17
2.1.3 Opinion	18
2.2 Autopsy	19
2.2.1 Overview	19
2.2.2 Functionality	19
2.2.3 Opinion	20
Chapter 3. Project Resources.....	21
3.1 NetBeans	21
3.2 Visual Studio Code	21
3.3 Java & Swing.....	21
3.4 Apache Ant	21
3.5 The Sleuth Kit.....	21
3.6 XAMPP	22
3.7 MySQL.....	22
3.8 SQLite	22
3.9 CSV	22
Chapter 4. Specifications and Design.....	23
4.1 Software Lifecycle Model.....	23
4.1.1 Waterfall Model.....	23
4.1.2 Agile Approach.....	24
4.2 Use Case Diagram.....	25



4.3 Use Case Descriptions	25
4.3.1 Login	25
4.3.2 Create User	26
4.3.3 Remove User	27
4.3.4 Obtain Case Reports.....	27
4.3.5 View Active Cases	28
4.3.6 View All Reports	29
4.3.7 Create Case	29
4.3.8 Open Case.....	30
4.3.9 Establish Cloud Use	30
4.3.10 Directory Search	31
4.3.11 File Search	32
4.3.12 Close Cases	32
4.3.13 Obtain Case Report	33
4.4 Describing Hardware	34
4.5 Describing Software.....	35
4.6 System Architecture.....	35
4.6.1 Candidate Class Identification	36
4.6.2 System Analysis Sketches	37
4.6.3 Communication Diagrams	38
4.7 GUI Sketches.....	39
Chapter 5. System Implementation	40
5.1 Configuration TSK	40
5.2 Design Patterns	41
5.2.1 Model View Controller (MVC) Architectural Pattern	41
5.2.2 Data Factory Pattern.....	42
5.2.3 Data Access Object	43
5.2.4 Singleton Pattern	44
5.2.5 Broker Architectural Pattern.....	44
5.3 Application pages	45
5.3.1 Login	45
5.3.2 Menu Frames	46
5.3.3 Create Case	47
5.3.4 Open Case.....	48
5.3.5 Establish Cloud Use	48
5.3.6 Directory & File Search	49
5.3.7 View Active Cases	50



5.3.8 Close Case	51
5.3.9 Obtain Case Reports.....	52
5.3.10 Logout	52
5.4 GUI Design.....	53
5.5 Database Structure & Security	54
5.5.1 MYSQL Database	54
5.5.2 SQLite Database.....	55
5.5.3 Security.....	55
5.6 Issues Faced during implementation.....	56
5.6.1 Library Configurations	56
Chapter 6. Testing and Results	57
6.1 Junit Testing	57
6.2 Run-through Testing	58
6.3 Result Outputs	58
6.3.1 Onscreen Output	58
6.3.2 CSV Output	59
6.4 Recovered Architecture System Class Diagram.....	60
Chapter 7. Discussion & Conclusion	61
7.1 Discussion	61
7.2 Conclusion.....	61
References.....	63
Copy of Poster.....	66



Table of Figures

Figure 1: Cloud-topsy Logo.....	2
Figure 2: EnCase GUI Sample [17]	18
Figure 3: Autopsy GUI Sample [19]	20
Figure 4: The Waterfall Model [14]	23
Figure 5: The Agile Approach [15]	24
Figure 6: Use Case Diagram	25
Figure 7: Software Outline.....	35
Figure 8: Package Diagram.....	35
Figure 9: System Analysis Sketch (Class Diagram)	37
Figure 10: Communication Diagram for manager adding a user to the system.	38
Figure 11: Communication Diagram for Investigator to login and create a case.	38
Figure 12: Basic GUI Drawings.....	39
Figure 13: Final GUI Design	53
Figure 14: MYSQL Table Structure.....	54
Figure 15: SQLite Table	55
Figure 16: JUnit Testing Success Investigator Class	57
Figure 17: JUnit Testing Success Admin Class	58
Figure 18: Onscreen Output Sample	58
Figure 19: CSV Output Sample	59
Figure 20: Recovered Class Diagram.....	60



Table of Tables

Table 1: Login CD.....25

Table 2: Create User CD.....26

Table 3: Remove User CD.....27

Table 4: Obtain Case Reports CD27

Table 5: View Active Cases CD28

Table 6: View All Reports CD.....29

Table 7: Create Case CD29

Table 8: Open Case CD30

Table 9: Establish Cloud Use CD30

Table 10: Directory Search CD.....31

Table 11: File Search CD.....32

Table 12: Close Cases CD32

Table 13: Obtain Case Report CD.....33

Table 14: Candidate Class Table.....36



Table of Code Extracts

Code 1: GUI Calls Application Logic	41
Code 2: Application Package Calls Model Package	41
Code 3: Model Package Calls Database Access	42
Code 4: 1/2 Factory Methods Used for Constructing Broker	42
Code 5: 2/2 Factory Methods Used for Constructing Broker	43
Code 6: Factor Class to Build Range of Custom Objects.....	43
Code 7: Call to MYSQL Database	43
Code 8: Singleton Example	44
Code 9:DBWriteBroker Interface	44
Code 10: DBReadBroker Interface	45
Code 11: Call to Application Logic Login Function.....	45
Code 12: Login Function	45
Code 13: User Instance Check call.....	46
Code 14: Current User Variable Calls	46
Code 15: User Instance Check Function	46
Code 16: Call to Investigator Logic and Checks	47
Code 17: The Sleuth Kit Case DB and Passing Data in	47
Code 18: Browse Button Singleton Declaration	48
Code 19: Drop Down Singleton Declaration	48
Code 20: ECU Frame Calls Investigator Logic	49
Code 21: Establish Cloud Use Function	49
Code 22: Key Word Searching Example.....	50
Code 23: GUI Update After Case Selection	50
Code 24: Setting Date to Close Case.....	51
Code 25: CSV Generating Function.....	51
Code 26: Browsing Local Computer for Save Location	52
Code 27: Call to Singleton Function	52
Code 28: Current User Set to Null	52
Code 29: Sample MYSQL Code.....	54
Code 30: Hash & Salt Function	55
Code 31: Prepared Statement Example	56
Code 32:JUnit Test Code Admin and Investigator Class	57



Chapter 1. Introduction & Project Outline

1.1 Project Description

In the field of data forensics, investigating whether subjects have tried to **conceal suspicious data** in the **cloud** can be very difficult for reason of access and lack of information about how the data is stored in the cloud. A necessary first step, is **establishing that access to the cloud has been made** and this is usually done by **examining personal devices** used to transfer data to and from the cloud. Even this is not as simple as it sounds as the continuous evolution of personal device operating systems, combined with knowledge of how to delete such data by culpable users may impede the Investigator. Keeping information about this topic up to date is a constant battle for forensic Investigators.

In this project, the student will **investigate** a limited number of examples of **cloud storage systems** e.g., Amazon S3, Dropbox and also a limited number of **personal device operating systems** and fully **establish the tell-tale signs of usage** that may be **left behind**. Development of an **automated investigative tool** may be part of this project. In the project we will naturally have to **leave aside legal issues**, although it would be useful if the student also carried out a **brief investigation** into the necessary requirements in this jurisdiction.

1.2 Project Rationale

As mentioned in the project description, a necessary first step in data forensics is establishing that access to the cloud has been made. This project aims to create a platform for cyber security individuals to work alongside their team to determine whether or not suspects have accessed cloud storage systems. It aims to investigate supplied disk images looking for indications that specific cloud storage platforms have been used. The Investigator can create reports of their findings, logging relevant information like; evidence found, file identification number, date & location of evidence, general data from the image and lastly information about the Investigator who took the case. These reports are made available for export, to be used for further investigation and potentially used in a court case situation.

The application aims to seamlessly connect individuals with their teams and superiors. The superiors have access to all regarding cases and their reports. They can see cases that are



in progress and access the reporting list to determine how far the Investigator is into the investigation. They can also access all reports that have been created by searching completed cases. The superior's main function is to create new Investigators as required and monitor the progress of the team, gathering information as required.

The application is written in JAVA and integrated into The Sleuth Kit using the Java Native Interface (JNI) storing user data and extracted information inside a MYSQL database. The java aspect to the project will develop user functionality related to accounts and the reporting of the data collected. The disk image that is uploaded will be investigated using The Sleuth Kit and extracted information stored. The application will automate the process of determining whether a suspect has accessed a cloud storage system saving valuable time for an Investigator in the investigation of a case. It will investigate remnants left behind on Windows based images. A report on the procedure of investigation of non-windows images such as; Mac, iOS, and Android smartphones will be carried out.

1.3 Theory

What is cloud computing? Amazon defines cloud computing as the on-demand delivery of IT resources over the internet with pay-as-you-go pricing. They state that instead of buying, owning, and maintaining physical data centres and servers, one can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider like Amazon Web Service (AWS) [1].

In Cloud Computing there are three main subcategories which include: Public, Private and Hybrid. Public is used to deliver services across the internet, Private is aimed at internal use of an organization and hybrid is a combination of both Public and Private. Inside this there are three types of services offered to users. The first is IaaS (Infrastructure as a Service) it contains the basic building blocks for cloud IT. Typically it provides access to computers, networking features and data storage space. The second is PaaS (Platform as a Service) which removes the need to manage underlying infrastructure and allows focus on the deployment and management of applications. Finally, SaaS (Software as a Service) which provides a complete product that is managed by the service provider. It allows the user to not have to worry about how the service is maintained or how the infrastructure is managed [1]. SaaS is the type of



cloud computing in which cloud storage systems such as Google Drive, Dropbox and Evernote fall under.

Gartner, one of the world's leading research and advisory companies, predicts that at the end of 2020, public cloud revenue will grow by up to 6.3% which means it will grow from \$242.7 billion (2019) to \$257.9 billion. They state that SaaS remains the largest market segment and is forecast to grow to \$104.7 billion in 2020. COVID-19 [2] had a major impact on not only the growth of SaaS, but the growth of the entire world of cloud computing. A report from the Irish Times [3] has found that at least 40 percent of paid hours worked by employees in Ireland were performed from home at the height of the pandemic. COVID-19 could therefore be regarded as a catalyst for the inevitable change to a cloud-based business model for companies around the world. For this reason, it is important to understand how to conduct forensic based examinations into cloud computing.

With the increased usage of cloud computing, comes a surge in cyber-crime. Cloud computing crime can stem from several approaches; Stealing personal data stored and outsourced in the cloud and even attacks that disrupt a company's day to day workflow. Cloud storage services can be used to store and hide incriminating and illegal material or material that goes against copyright laws. Service providers are attempting to prevent their services from being exploited by these criminals. Companies such as Dropbox have implemented a software that detects data related to child abuse. It searches through the files stored on the service to identify breaches of policies that the perpetrator has agreed to follow. Similarly, Microsoft developed a software called PhotoDNA that is designed to identify similar data [4].

In order to conduct a successful forensic investigation into cyber-crimes involving digital evidence, the Investigator must be able to collect the evidence of the incident or crime that involved both cloud servers and the client device that was used to access the cloud service. It is important for the Investigator to be able to locate and report on data remnants that can be located on the suspects' personal machine. This project will focus on the retrieval of these fragments.



1.4 Literature Review

Cloud computing and storage solutions provide companies and private users with the capability to store their data in third part data centres [5]. This data can be targeted by criminals. These attacks can range from stealing personal information from the cloud to disrupting a company's business operations [4].

On the 11th of March 2020, the World Health Organization characterized COVID-19 as a pandemic. This upended the worlds business ecosystem as we know it. To maintain business productivity, the enterprise working model turned into one of a working from home model using BYOD (Bring your own device). As a result of the change in working environment, cyberattacks, which previously targeted individuals were now a risk for businesses [6].

Cloud service providers have attempted to prevent exploitation of their services; examples include Dropbox child abuse detection software and Microsoft's PhotoDNA. Other security solutions have been proposed, ranging from privacy-preserving to intrusion detection. Despite the existence of these solutions, cybercrimes are still committed. To prosecute cyber criminals, it is necessary to gather digital evidence to prove that the crime has been committed. This process is known as digital forensics [4]. Even though the log of a cloud server can tell the history of a user's action, the hosting companies are not always willing to release information to the Investigators, to protect clients' privacy. However, traces of user actions are left in the user's device. Webb-Hobson stated that even though user-made files such as documents and photos are not stored on local machines, traces, or fragments of them which are related to their actions can be found on the machine.[7]

Buyu and Abade [8] analysed artifacts left behind by Dropbox on windows 10 machines. By identifying these remnants, it is possible to get a better understanding of the remaining artifacts, to help with forensic investigation. The information sources include client software installation files and browser related artifacts.

Marturana in [9] deduced that on windows 7, browser artefacts, sync logs, timeline of recently opened, modified, deleted files by Dropbox could be obtained.



McClain [10] discovered that for Dropbox, remnants could be found on windows XP that included installation directory, log files, database files and uninstallation data.

The use of a cloud storage service will leave traces in all local devices such as PCs and Smartphones. Therefore, all devices that can access an individual's cloud storage usage must be examined when undertaking digital forensics on that storage. The Centre for Information Security Technologies (CIST) in Korea proposed a successful process model for investigating artifacts of all accessible devices including Windows, Mac, iPhone, and Android. The process involves identification, collection, analysis and reporting of artifacts related to these machines. Through the use of their model, they successfully identified the location of artifacts related to cloud storage services left behind on these machines [11].

The first element for the CIST team was to examine logfiles of web browsers and artifacts of client application that are installed on the machine. The team took two popular web browsers: Internet Explorer and Firefox. Web browser log files are stored in profile directories and the files consist of cache, history, cookies and downloaded files. The history files contain URLs that a user has visited, titles of Web pages, the times of visits, and their number of visits. The cookie files store information about hosts, paths, cookie modification times, cookie expiration times, names, and values; Valuable information in terms of cyber forensics. The team also collected artifacts of client applications installed on the device such as database files and application log files in the form of txt, log and htm files. The database files contained information about credentials, creation time, modification time and files that have been accessed. The log files store general information about authentication information, history of users' behaviours and general user data. [11]

The second element for the CIST team was to analyse the collected data and check if traces of cloud storage services existed in the collected data. If user credentials and any other information were found, a search and seizure warrant should be issued. The user's location in the cloud storage service is a private space. This mean that if an Investigator were to login and obtain data without a warrant the evidence would not be admissible in the court of law. The Investigator is only allowed to investigate artifacts that remain inside the client's device. [11]

Forensics examiners often use open-source software for their investigations according to Altheide and Carvey in [12]. Using freely available software to learn digital forensics has



advantages. The software tools allow the Investigator to execute, examine options and output, and examine the code that produced the output. This gives the Investigator a better understanding of the tool's operations. Aspects of an investigation may be too small to justify spending a significant amount of money on a software to complete it. For these scenarios, an Investigator can install and run this software on any hardware that is available to them free of charge [12].

Altheide and Carvey state that the biggest benefit to open-source software is that the code is provided. The ability to review and modify the source code that gets compiled into a working program is invaluable. The Sleuth Kit is one such example, describing different ways to review bug fixes in the software. If there is a change to the software, an Investigator can look at the freely accessible bug trackers maintained at The Sleuth Kit project site [13]. Proprietary forensic products which develop in a "black box" make the identification software modifications less obvious. Lack of access to the source code acts as an additional layer of abstraction between the Investigator and the truth. In this case the layer of abstraction acts as a source of error [12].

Buyu and Abade in their investigation used several software's, some open source, and some closed including Access Data FTK Imager (FTK), and Autopsy. FTK was used for imaging and analysing the virtual machines that they created. They then used Autopsy for the forensic analysis of VM images. Using live forensic installs, artifacts were identified. This was an arduous process to determine what artifacts were created at the installation phase. Had they known what files to look for to begin with, they could have skipped the live forensic investigation and invested more time into the decryption of the artifacts that were eventually collected [8].

The determination and detection of artifacts using live forensic investigation, is a lengthy process that can impair the Investigator. If the latter had access to software that could locate the artifacts and allow access to the relevant data, it would accelerate the advancement of the criminal case. Had the Investigator known that there were no artifacts to be restored on the image, they could have saved valuable time. They would not have had to carry out a live forensic investigation process. Using research in [8] and [11], a software could have been developed that accelerates the location of artifacts for the Investigator.



Chapter 2. Similar Systems

The number of Forensic investigative tools has increased dramatically in the last number of years. Several have been designed with the intention of the Investigator purchasing the licence to use them. There are a number of software's that are open source. Among them are EnCase; a licenced and Autopsy an open-source software. For the purpose of this report I will be comparing Autopsy and EnCase against *Cloud-toppsy*.

2.1 EnCase Forensic.

2.1.1 Overview

The technology surrounding EnCase is shared within a suite of digital investigation products by Guidance Software. It comes in several products designed for forensic, cyber security, security analytics, and e-discovery use. It is traditionally used to recover evidence from seized hard drives.

2.1.2 Functionality

EnCase allows the Investigator to conduct in-depth analysis of user files to collect evidence such as document, pictures, internet history and Windows Registry information. A number of key words that describe the functionality of EnCase are; Triage, Collect, Decrypt, Process, Investigate and Report. One of the major advantages EnCase has over other software in the market is its track record of court-acceptance.

EnCase is a licenced software, which means that Investigators must pay to use it. This made investigating it quite difficult. Further investigation was deemed necessary due to the fact that it is the market leader. A request for a demonstration version of their software was submitted, through a form on their website and by email, but they failed to reply to either case. For the purpose of developing knowledge, a video was watched to understand the working of the software [16].

The video brought the viewer through a full-scale efficient examination within the EnCase Forensic software. It covers a number of core search types such as; raw searching, tag searching and index searching.



1. Raw searching is conducted on non-indexed case data, typically used at the triage stage of the investigation prior to the imaging of the hardware. It determines whether information exists to warrant seizure or imaging in the case.
2. Tag searching is searching for items flagged with user defined tags. It takes place after the Investigator has already started an investigation.
3. Index searching involves searching for terms within a generated index. It is preformed through evidence processor tags within the EnCase software.

The graphical user interface of EnCase has developed over the last two decades. In 1998 Andrew Rosen released EnCase after working on it for three years out of his home. It was first developed at the time where there was no GUI forensic tools available. Figure 2 shows EnCase Forensics main screen. It allows new cases to be created, opened, searched, edited, deleted and many more. The screen also shows the name of the device that is currently being investigated. Lastly, it shows data related to the files that have been found.

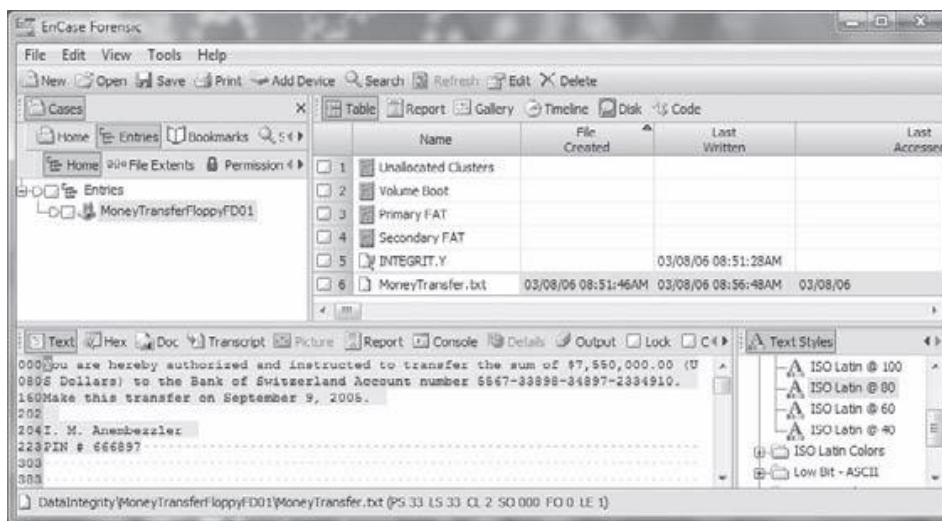


Figure 2: EnCase GUI Sample [17]

2.1.3 Opinion

There are several aspects of this software that are similar to **Cloud-toppsy**. The ability to search by index and tag is a major similarity. Allowing general case information to be recorded and stored is also remarkably similar, but it seems that the majority of the software on the market contain these similarities. However where **Cloud-toppsy** is a software designed to connect both the Investigator and their superiors, it does not seem like EnCase has this functionality. **Cloud-toppsy** allows the admins/managers to monitor the progress of each of the cases. **Cloud-**



topsy goes a step further by allowing the exportation of information about the case whenever the admin deems it necessary, this is achieved through the use of the MYSQL database.

2.2 Autopsy

2.2.1 Overview

Autopsy is a digital forensics platform and graphical interface to The Sleuth Kit and other digital forensics tools. It is used by law enforcement, military, and corporate examiners to investigate what happened on a computer. It also allows recovery of photos from a camera's memory card. Brian Carrier has developed most of the code in both The Sleuth Kit and Autopsy. It was designed with the aim to provide as much information as possible to the Investigator.[18]

2.2.2 Functionality

Autopsy was designed to be an end-to-end platform with modules that come with it upon installation. It is design to keep an open format, meaning that users can verify, learn, and develop with it. It was also designed with the aim to educate people with a hands-on approach. Some of the modules provided include; keyword search, web artefacts and multimedia. As budgets are decreasing, cost effective digital forensics solutions are essential in the world of computer forensics today.

Autopsy is an open source, HTML-based software that can connect to the Autopsy server from any platform using a HTML browser. It provides a 'file manager'-like interface and shows details about deleted data and file system structures.

Through installing Autopsy an investigation was undertaken. The case management system was examined. Elements such as the reports and logging were investigated. Several search techniques were tested such as; File Listing, File Content and Key Word Search.[18]

1. File Listing includes analysing the files and directories, this includes the names of the deleted files and files with Unicode-based names.
2. File Content allows the contents of files to be viewed in raw, hex, or the ASCII value. When data is interpreted, Autopsy sanitizes it to prevent damage to the local analysis system. Autopsy does not use any client-side scripting languages.



Chapter 3. Project Resources

There are several resources that aid in the creation of Java applications. The technologies used in the development of this project are discussed further in the following subsections.

3.1 NetBeans

NetBeans is an open source, integrated development environment (IDE) for Java. It has a multi-language editor, debugger, profiler, and versioning control. Applications can be developed from a set of modular software components called modules. It supports Windows, macOS, Linux and Solaris. In terms of compilers the IDE supports a variety such as Oracle Solaris Studio, GNU, Clang/ LLVM, Cygwin and MinGW. [20] NetBeans was the main IDE used in the development of *Clous-topsy*.

3.2 Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on the desktop and is available for Windows, macOS and Linux. It has support tools for debugging, version control and task running. It provides built in support for MYSQL development allowing tables to be created, altered, or dropped directly inside server explorer.

3.3 Java & Swing

Java is a programming language, designed to be concurrent, class-based, and object-oriented. A large amount of applications and websites will not work unless java is installed. As part of its framework, swing is an excellent cross-platform application development language. It is considered to be one of the best cross-platform development frameworks, providing different components that can be used in the environment. [21]

3.4 Apache Ant

Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extensions points dependent upon each other. It can also be used to build java applications.

3.5 The Sleuth Kit

The Sleuth Kit is a collection of command line tools and a C library that allows Investigators to analyse disk images and recover files from them. As mentioned previously it is used behind the scenes in Autopsy, *Cloud-topsy* and a number of other open source and commercial forensic tools.



3.6 XAMPP

XAMPP is an open-source cross-platform web server solution stack package. It consists of mainly Apache HTTP Servers, MariaDB (MySQL) databases and interpreters for scripts written in PHP and PERL programming languages.

3.7 MySQL

MySQL is an open-source relational database management system (RDBMS) with a client-server model. RDBMS is a software or service used to create and manage databases on a relational model.

3.8 SQLite

SQLite is a C-Language library that implements a small, fast, self-contained SQL database engine. It is one of the most used database engines in the world. The file format is backwards compatible, and the developers pledge to keep it that way.

3.9 CSV

CSV is a simple file format used to store tabular data, such as spreadsheet or databases. Files in the CSV format can be imported and exported from programs that store data in tables, such as *Cloud-toppsy* and Excel.



Chapter 4. Specifications and Design

4.1 Software Lifecycle Model

Software development lifecycle (SDLC) is a framework that defines the steps involved in the development of software at each phase. It is the process used by the software industry to design, develop and test high quality software. The aim is to produce a software that is high in quality, meets the needs of the customer and is completed within the specified timeframe and cost estimates. It was one of the first aspects that came to mind when developing a concept for the project brief. Several approaches were investigated, two of which are discussed below.

4.1.1 Waterfall Model

The first of the software lifecycle models which was researched for this project was the Waterfall model. It breaks down the project activities into linear sequential phases. Each phase is dependent on the completion of the previous phase and corresponds to a specialization of tasks. It uses a clear structure when compared with other methodologies. Each step must be completed in full before moving onto the

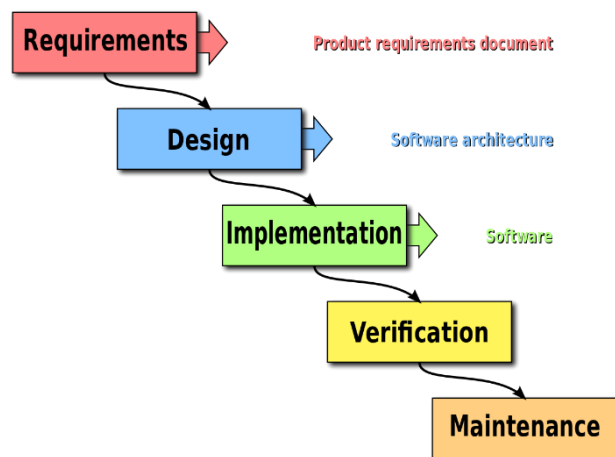


Figure 4: The Waterfall Model [14]

next. If any problems are encountered, they are addressed right away, stopping the project progression in its tracks, leaving the developer with a complete and polished product as a result. This model permitted me to start right away without any steep learning curve to slow down my process. The model determines the end goals early and transfers information well as it is a highly methodical approach. When investigating further into the waterfall model, it was found that any project changes that were required, would be difficult to implement. It left no room for unexpected changes or revisions. It was also found that its delayed testing until after completion of the project. This would not be a good solution due to the time constraints for this project. For these reasons it was decided to look further into other approaches that could possibly use.



4.1.2 Agile Approach

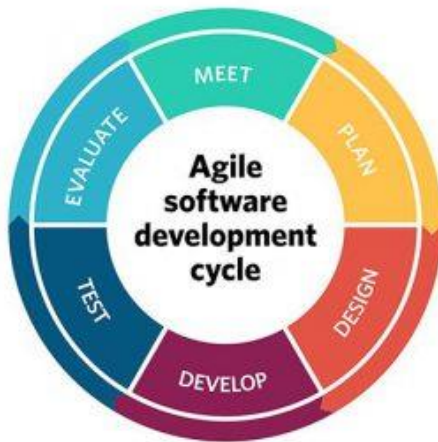


Figure 5: The Agile Approach [15]

the short timeframe for project development made this an ideal approach. This model would allow the testing of each of the software elements as they are being delivered. It would permit late project changes such as extra implementations at any developmental stage. This would be advantageous since many of the concepts were new.

The next software lifecycle that was decided on was the Agile Approach. It was found that the Agile method promotes a disciplined project management approach that encourages frequent inspection and adaptation. The approach corresponds with the Agile Manifesto which has several principles including Delivering working software frequently, getting tasks done in assigned amounts of time and allowing changes in the project at any stage in the development. It was found that



4.2 Use Case Diagram

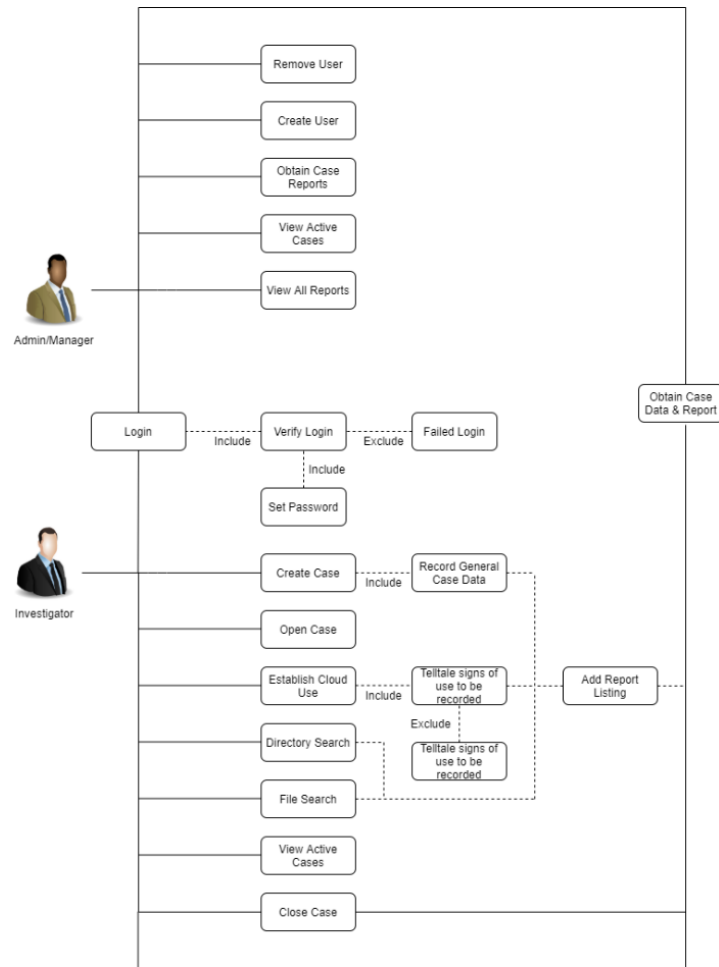


Figure 6: Use Case Diagram

4.3 Use Case Descriptions

4.3.1 Login

Table 1: Login CD

Use Case 1	<i>Login</i>
Goal in Context	Actors can login to the system to use actions that are related to their status.
Precondition	Account must be made and knowledge of credentials.
Success End Conditions	Actor successfully able to login and access features.
Failed End Condition	Actor login is unsuccessful.
Trigger	The application is launched.
Actors	All Actors.



Description	Step	Action
	1	Actor launches the application.
	2	Window asking for login credentials is presented.
	3	Login information is passed in.
	4	System checks first if the account exists and then if the password is correct.
Extensions	Step	Branching Action
	4	If the credentials are incorrect, the system will display a message indicating it.

4.3.2 Create User

Table 2: Create User CD

Use Case 2	Create User	
Goal in Context	Actor creates users who can perform investigations on images.	
Precondition	Knowing basic information about the person who is going to be set up.	
Success End Conditions	Actor successfully creates an account.	
Failed End Condition	Actor is unable to create an account.	
Trigger	Actor presses button named 'Create User'.	
Actors	Admin/ Manager.	
Description	Step	Action
	1	Actor clicks create button.
	2	Actor collects and enters required personal information about the new user.
	3	System saves the data and creates an account.
Extensions	Step	Branching Action
	4	The account is saved. Upon initial login the user will be asked to set a password.



4.3.3 Remove User

Table 3: Remove User CD

Use Case 3	<i>Remove User</i>	
Goal in Context	Actor removes a user from the database.	
Precondition	Knowing basic information about the person who is going to be removed.	
Success End Conditions	Actor successfully removes an account.	
Failed End Condition	Actor is unable to remove an account.	
Trigger	Actor presses button named 'Remove User'.	
Actors	Admin/ Manager.	
Description	Step	Action
	1	Actor clicks remove button.
	2	Actor selects the user which is going to be removed.
	3	System checks that the user exists in the database.
	4	System removes all information related to the user from the database.
Extensions	Step	Branching Action
	4	The system will not remove any case information related to the user. It will also store the name of the user to use for case referencing.

4.3.4 Obtain Case Reports

Table 4: Obtain Case Reports CD

Use Case 4	<i>Obtain Case Reports</i>	
Goal in Context	Actors can collect information found for a case.	
Precondition	Knowing basic information about case.	
Success End Conditions	Actor successfully gets displayed report on-screen or in the form of a CSV file.	
Failed End Condition	Actor does not get on-screen or CSV report.	
Trigger	Actor selects button 'Display Reports'.	
Actors	All actors.	



Description	Step	Action
	1	Actor selects obtain case report button.
	2	Actor has selected the case to be reported on.
	3	System gets information related to the case.
	4	System returns the results in required format.
Extensions	Step	Branching Action
	4	Options are chosen previously in relation to on-screen or CSV file.

4.3.5 View Active Cases

Table 5: View Active Cases CD

Use Case 5	View Active Cases	
Goal in Context	Actor can see on-screen result of active cases	
Precondition	Actor is logged in.	
Success End Conditions	Actor can see the on-screen result of the active case in the database.	
Failed End Condition	Actor is unable to see the result of active cases.	
Trigger	Actor selects button 'View Active Cases'.	
Actors	All Actors.	
Description	Step	Action
	1	Actor presses button to view active cases.
	2	Actor selects case to view more details for investigation. The items are the list of active cases inside of the database.
	3	System returns the information related to the case in on-screen table format.
	4	Actor can select elements to add to findings.
Extensions	Step	Branching Action
	4	Findings are then added to MYSQL database and can be reported upon later.



4.3.6 View All Reports

Table 6: View All Reports CD

Use Case 6	<i>View All Reports</i>	
Goal in Context	Actor can see on-screen result of all cases.	
Precondition	Actor is logged in.	
Success End Conditions	Actor views information related to case selected from the list of all cases.	
Failed End Condition	Actor is unable to view information related to case.	
Trigger	Actor selects 'View All Reports' button.	
Actors	Admin/Manager	
Description	Step	Action
	1	Actor selects button to view all reports.
	2	Actor selects case in the dropdown of all cases.
	3	System returns all information related to the case selected in on-screen format.

4.3.7 Create Case

Table 7: Create Case CD

Use Case 7	<i>Create Case</i>	
Goal in Context	Create a new case.	
Precondition	Actor must have image file of suspect's machine.	
Success End Conditions	A case is successfully created, and case information is entered into the database.	
Failed End Condition	Case is not created, and data is not entered into the database.	
Trigger	Actor selects 'Create Case' button.	
Actors	Investigator	
Description	Step	Action
	1	Actor selects button to create case.
	2	Actor enters name of the case.
	3	Actor enters description of the case.
	4	Actor selected image to be uploaded to database.
	5	System takes image and stores it inside SQLite database.



Extensions	Step	Branching Action
	5	System uses The Sleuth Kit action sequence for creating a case.

4.3.8 Open Case

Table 8: Open Case CD

Use Case 8	<i>Open Case</i>	
Goal in Context	Actor opens case to enable case investigation.	
Precondition	Create case function must be completed.	
Success End Conditions	Actor can access information regarding the case in further functions.	
Failed End Condition	Actor unable to access case data.	
Trigger	Actor selects 'Open Case' button.	
Actors	Investigator	
Description	Step	Action
	1	Actor selects open case button.
	2	Actor can select database location or select from list found inside database.
	3	System selects case to be opened.
	4	System sets user singleton to the chosen case.
Extensions	Step	Branching Action
	2	The dropdown list is a list of all the open cases.
	4	The director for the case is set to current director in current user singleton.

4.3.9 Establish Cloud Use

Table 9: Establish Cloud Use CD

Use Case 9	<i>Establish Cloud Use</i>
Goal in Context	Find remnants related to Dropbox, Google-Drive, Evernote, and One Drive.
Precondition	Open case function must be completed.



Success	End	System returns list of remnants found inside the case related to the cloud storage systems.	
Failed End Condition		System does not return a list of the remnants.	
Trigger		Actor selects 'Establish Cloud Use' button.	
Actors		Investigator	
Description		Step	Action
		1	Actor selects the establish clous use button.
		2	Actor clicks the start search button.
		3	System returns a tick or a cross depending on whether elements related to the respective cloud storage are found.
		4	Actor selects show findings.
		5	System returns a full list of findings in table format to Actor.
		6	Actor can select remnants to add to case findings.
Extensions		Step	Branching Action
		5	Actor is able to select multiple remnants at once by using the ctrl and shift buttons while selecting.

4.3.10 Directory Search

Table 10: Directory Search CD

Use Case 10	Directory Search	
Goal in Context	Return list of directories containing search word.	
Precondition	Actor knows the word they wish to search for.	
Success	End	A list of files and directors are returned containing the word that the Actor has selected.
Failed End Condition		List is not returned for the Actor to investigate.
Trigger		Actor selects the 'Director Search' button and enters word.
Actors		Investigator.
Description		Step Action
		1 Actor selects the directory search button.
		2 Actor enters the name of the directory they wish to search for.



	3	System returns a list of directories and files related to the search value in table format.
	4	Actor can select an element to enter into the case findings.
Extensions	Step	Branching Action
	4	Actor selects the entry and clicks submit select button.

4.3.11 File Search

Table 11: File Search CD

Use Case 11	File Search	
Goal in Context	Return list of files containing search word.	
Precondition	Actor knows word they wish to search for.	
Success End Conditions	A list of files is returned containing the word that the Actor has selected.	
Failed End Condition	List is not returned for the Actor to investigate.	
Trigger	Actor selects the 'File Search' button and enters word.	
Actors	Investigator.	
Description	Step	Action
	1	Actor selects the file search button.
	2	Actor enters the name of the file they wish to search for.
	3	System returns a list of files related to the search value in table format.
	4	Actor can select an element to enter into the case findings.
Extensions	Step	Branching Action
	4	Actor selects the entry and clicks submit select button.

4.3.12 Close Cases

Table 12: Close Cases CD

Use Case 12	Close Cases	
Goal in Context	Close case that actor has selected.	
Precondition	Actor knows basic information about case to close.	
Success End Conditions	The case is closed and information inside the database is updated.	



Failed End Condition	Database is not updated	
Trigger	Actor selects 'Close Case' and selects case to close.	
Actors	Investigator.	
Description	Step	Action
	1	Actor selects close case button.
	2	Actor selects case to close.
	3	System updates database with relevant information to close case.
Extensions	Step	Branching Action
	3	The case will no longer be displayed inside active cases and the investigator will no longer have access to this case. The case is not deleted, it is stored to be investigated by Admin/Managers.

4.3.13 Obtain Case Report

Table 13: Obtain Case Report CD

Use Case 13	<i>Obtain Case Report</i>	
Goal in Context	Actor gets access to report finding of case.	
Precondition	Actor knows basic information about case.	
Success End Conditions	Actor gets CSV file containing file information stored in specified location.	
Failed End Condition	CSV file is not created.	
Trigger	Actor selects 'Obtain Case Report' button.	
Actors	Admin/ Manager	
Description	Step	Action
	1	Actor selects obtain case report button.
	2	Actor selects a case from the list of all cases.
	3	Actor selects a location to save the CSV file.
	4	Actor clicks the Create CSV button.
	5	System gathers information related to the case selected and creates CSV with relevant information.
Extensions	Step	Branching Action



	3	By default, the location for save will be the same location as the database is stored.
	5	The CSV file contains all information related to the case.

4.4 Describing Hardware

Due to the recent COVID-19 pandemic, access to any of the University of Limerick's (UL) facilities are restricted. This means that the project was developed and ran on personal hardware. Access to the laboratories in UL was not possible due to COVID restrictions; so, the project was undertaken from home.

The system which develops the software is running an Intel(R) Core (TM) i5-4460 central processing unit (CPU) with 3.20GHz. It has 8GB of random-access memory (RAM) installed and a Nvidia 960 graphical processing unit (GPU). For the purposes of testing and deployment, VMWare Workstation 15 Pro was used.

This software was chosen because it allows the simultaneous running of virtual machines which were used to create different scenarios which will be described later in this report. Several USB thumb drives were also configured to contain elements of cloud storage solutions. Lastly, a control USB was configured with data not related to cloud storage solutions. This was used as a comparison to the configured drives.



4.5 Describing Software

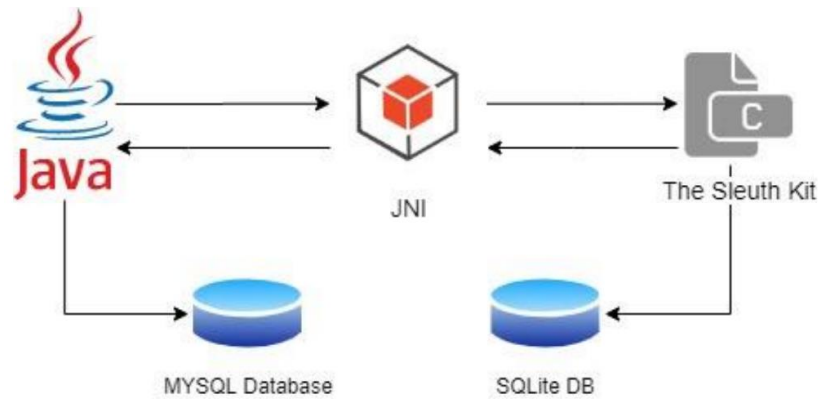


Figure 7: Software Outline

The software consists of a Java interface which integrates The Sleuth Kit (TSK) using a Java Native Interface (JNI). TSK uses an SQLite database to store basic information regarding cases and sources of data. The Java interface uses a MySQL database which contains user specific data and information related to case reports and findings. The SQLite database is self-contained meaning that it is an embedded database that runs as part of the app. On the other hand, the MySQL database is a relational database management system (RDBMS) used to store, retrieve, modify, and administrate a database.

4.6 System Architecture

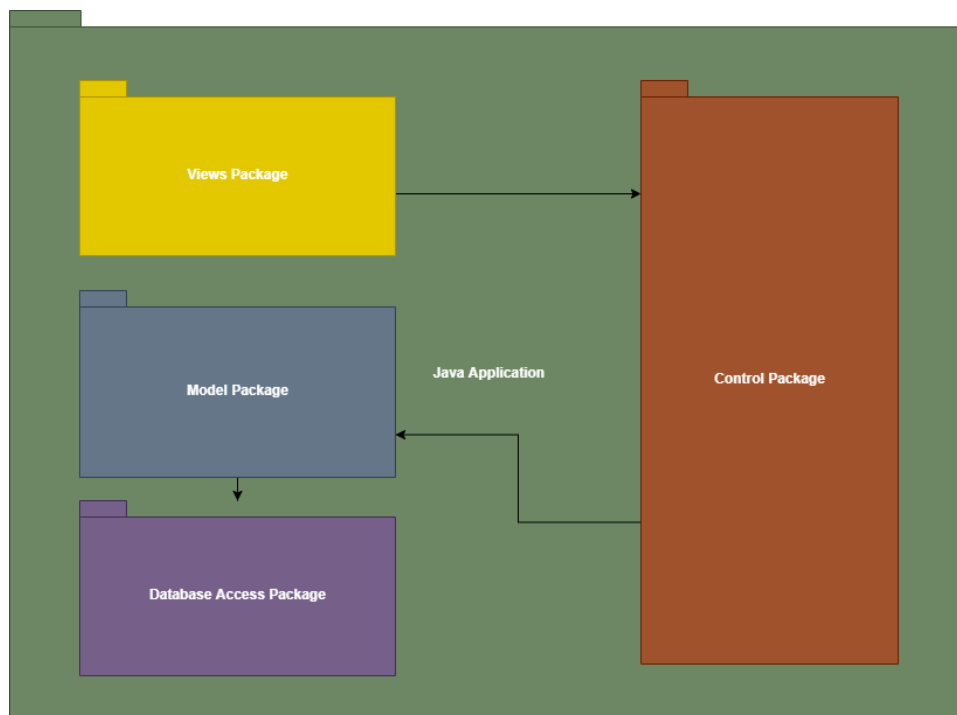


Figure 8: Package Diagram



At this stage in the project, it was decided that a database was needed to implement the Model-View Control (MVC). It was decided that it would make most sense to use a Java application for the use of Investigators and Admins/ Managers. The reasoning behind the Java implementation is that the program would only exist on the computers inside of the forensic laboratory.

4.6.1 Candidate Class Identification

It was decided to gather the initial classes by using the noun method of class identification, this was done using the first draft of the project rational for the interim report. A table was formatted, and the cells highlighted as the implementation took place.

Table 14: Candidate Class Table

Data Forensics	Access	Cloud	Team
Cloud Storage	Disk Images	Investigator	Reports
Evidence found	File ID number	Location	Date
Image Data	Investigator Info	Court Case Situation	Admin/ Manager
Cases	Gathering Info	Java	Accounts
Time	Remnants	Windows Images	macOS

	Classes that were implemented.
	Classes that are yet to be implemented
	Classes that were not used.

This method was an excellent starting point for the development of the project, however there was a need for many additional classes for the program to function currently. These additional classes were derived through assessing requirements and common design pattern definitions. The classes that were not used may be due to the fact that they physically could not become classes, or they simply were just not implemented due to time constraints and other factors.



4.6.2 System Analysis Sketches

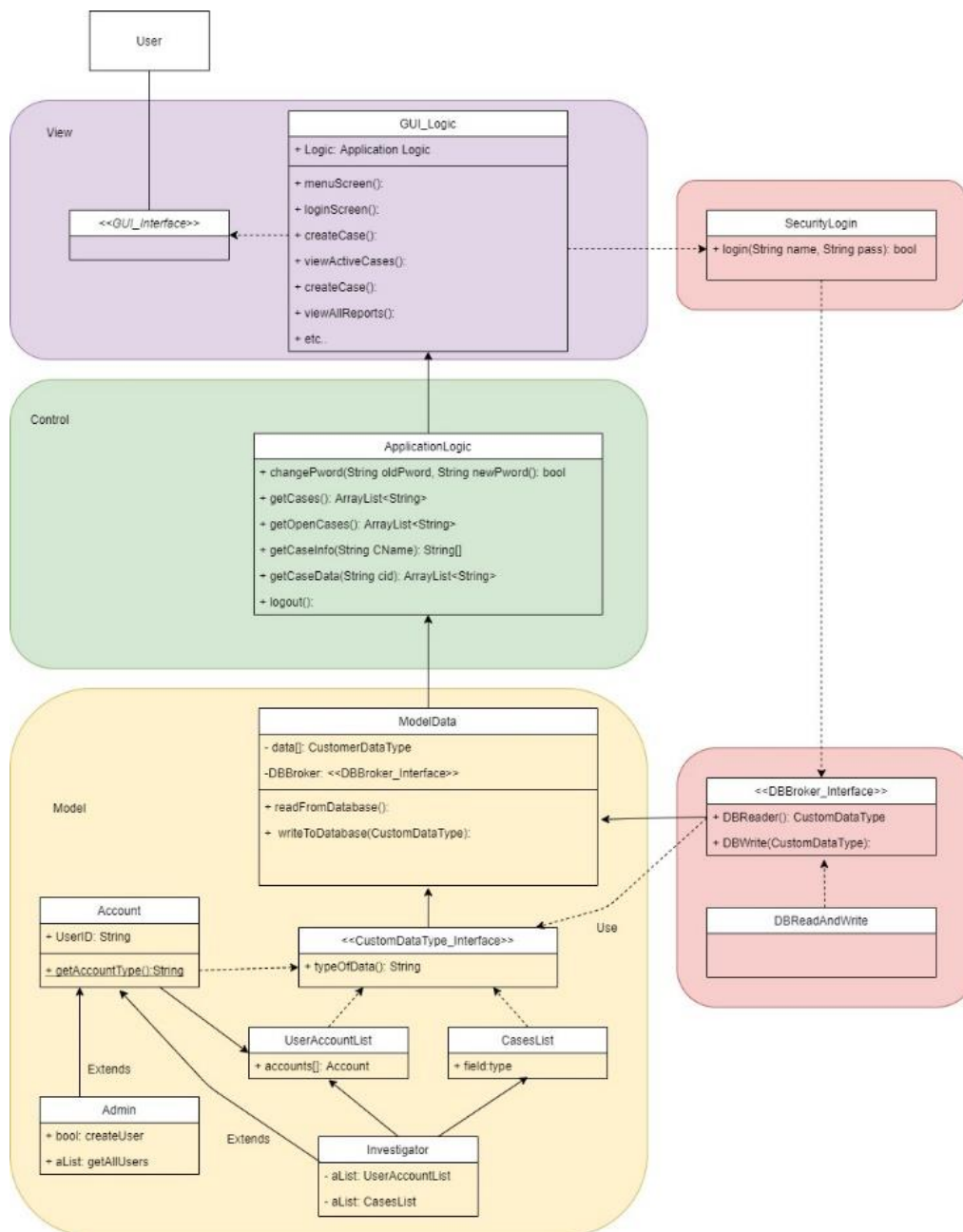


Figure 9: System Analysis Sketch (Class Diagram)

The class diagram can usually be described as the main building block of object-oriented modelling. It is used for the general conceptual modelling of the structure of an application. It is a structural UML diagram that shows that static structure of the classifiers in a system. It helps the development as it transitions into the structure of the programming code.



4.6.3 Communication Diagrams

a. Admin / Manager

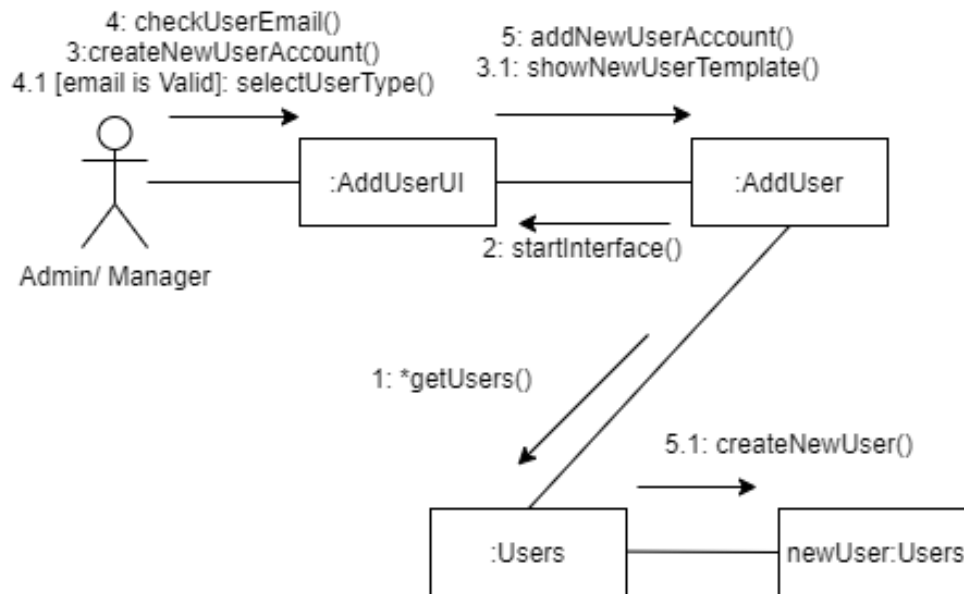


Figure 10: Communication Diagram for manager adding a user to the system.

b. Investigator

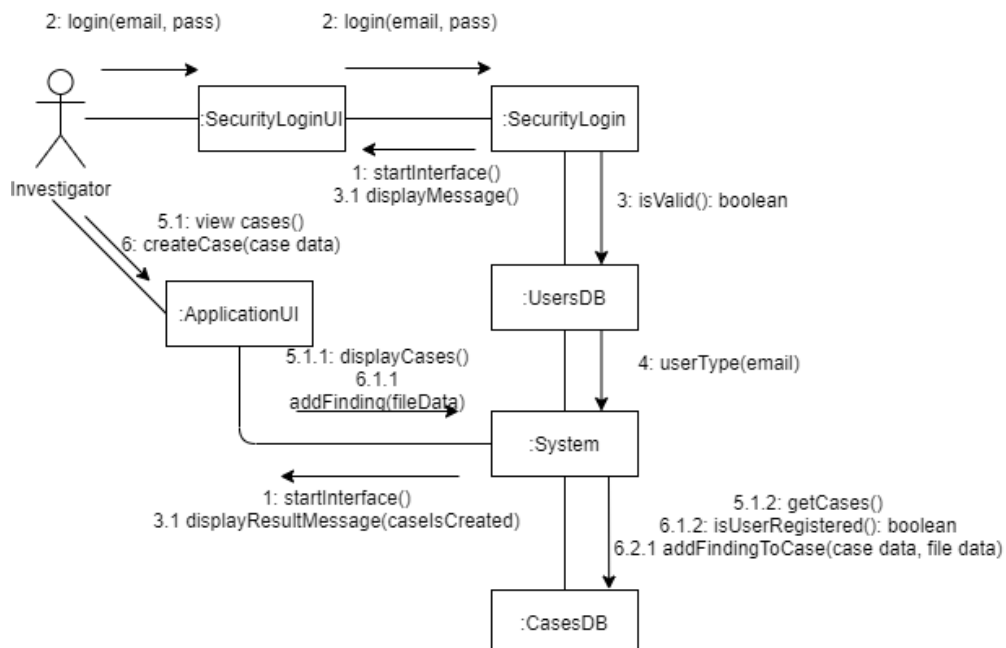



Figure 11: Communication Diagram for Investigator to login and create a case.



4.7 GUI Sketches

Cloud-Topsy Landing Page

Title of Project	
	
Login	Learn more

Log-in

username :		
username entry		
password :		
password entry		
Back	clear	Submit

Investigator menu

Welcome username	
Current Case DB: CaseDB	
Create Case	Open Case
establish Cloud use	Directory Search
file Search	view Active Case
Close Case	change Password
Logout	

Admin Menu

Welcome username	
view All Reports	view Active Cases
Obtain Case Report	change Password
Create user	Remove user
Logout	

The Graphical User Interface (GUIs) were designed with the user in mind. All of the frames are simple to use with no need for any learning of how to use the program.

These basic GUIs were drawn with Java JFrame in mind. The GUIs designed had to be simple enough for the development inside the Java programming language.

create case





Create New Case :		
Case Name :	input name	
Case Description :	input Desc	
upload Image :	<input type="button" value="Browse"/>	
case database will be stored as : Case database store location		
Back	Clear	Submit

open case

Open an Existing Case :		
select Database :	<input type="button" value="Browse"/>	
OR		
select a case	<input type="button" value="Dropdown"/>	
case database that will be opened : Case database store location		
Back	Clear	Submit

Provided are a selection of GUIs which are deemed to be the most fundamental aspects of the *Cloud-topsy* software.

Establish Cloud use

Establish uses of Cloud Platform		
		
		
Back	start Search	Show findings

view Active cases

Select Case to investigate:		
<input type="button" value="Dropdown"/>		
Case db location :	database loc	
Case open date :	open date	
Case close date :	close date	
Case Description :	case description	
id	File name	File Directory
file id	file name	file directory
file id	file name	file directory
file id	file name	file directory
file id	file name	file directory

Square black boxes with text inside indicate buttons, blue elements are representing elements that must be inserted by the user and pink is related to data returned by the database.

Figure 12: Basic GUI Drawings



Chapter 5. System Implementation

In the following section, the design implementation will be discussed. It will outline the configuration of TSK and the building of the libraries that it depends on. Following that, the design patterns will be discussed in detail, providing sample code fragments to demonstrate where the pattern takes place. The application pages section will detail elements of each main function within the software, also providing code fragments to demonstrate the functions. Next, there will be a short discussion on the structure of the database. Lastly, some issues faced during the implementation will be specified.

5.1 Configuration TSK

Configuration of The Sleuth Kit (TSK) was a tedious task. It had a number of requirements which included; Java JDK, Apache Ant, Visual Studio, and the Jar files listed created by the ivy.xml file supplied within TSK. The first stage was to build the Jar files, and then to place them in the appropriate directory. Several libraries were obtained and stored in the same directory.

To build the JAR files (used for the JNI) the win32 Visual studio solution was used. The JDK_HOME environment variable had to be set to the root director of the JDK. The build was executed, and the files were acquired. The next stage was to put the libraries into the correct location for TSK package to be able to recognise them. There was an error in the code that will be discussed in section 5.6 of this document.

Libewf and zlib had to be manually downloaded and installed into the NetBeans package as The Sleuth Kit relied on them to have a successful configuration. Several other libraries had to be configured, this was done through the use of Ant. These libraries needed to be stored in the same location as the dynamic libraries, being inside the 'dist' folder of The Sleuth Kit package.



5.2 Design Patterns

5.2.1 Model View Controller (MVC) Architectural Pattern

MVC is a software design pattern commonly used for developing user interfaces that divides the related programming logic into three interconnected elements [22].

MVC pattern was implemented by separating the application into three packages, one for the data model, one for the GUI and one for the application logic. Inside of the data model package, a data access package was added that holds broker implementations. These implementations dealt with reading and writing data to the database.

When a user wishes to view any stored data, they use the presented GUI functions; these functions call the application logic package layer, which in turn calls the data model package layer. If the data is not already present in the data model, it calls on the broker functions to retrieve the required data.

In short, if a user wishes to modify data in the application, they may only do so through using the functions provided by the application logic. The advantages of implementing MVC are, that it saves time, creates an effective use of resources and most importantly, any modifications do not affect the entire model. Code 1, 2 and 3 below demonstrate the MVC process of calling functions:

```
fnameVar = fname.getText();
lnameVar = lname.getText();
unameVar = username.getText();
emailVar = email.getText();
pwordVar = password.getText();

if(adButton.isSelected()){
    permID = 1;
    if(adLogic.createUser(unameVar,fnameVar,lnameVar,emailVar,pwordVar,permID)){
        JOptionPane.showMessageDialog(null, "The user has been created!");
        parent.setVisible(true);
        dispose();
    }else{
        JOptionPane.showMessageDialog(null, "There has been an error: The Admin has not been created!");
    }
}
```

Code 1: GUI Calls Application Logic

```
public boolean createUser(String uname, String fname, String lname, String email, String pword, int permID){
    //Call to db to create the user
    String hashedPassword = generateHash(pword);
    boolean checkCreateUser = DBWriter.createUser(uname, fname, lname, email, hashedPassword, permID);

    return checkCreateUser;
}
```

Code 2: Application Package Calls Model Package



```
public static boolean createUser(String uname, String fname, String lname, String email, String pword, int permID){
    boolean result = false;
    Connection connection;
    PreparedStatement ps;

    try
    {
        Class.forName("com.mysql.jdbc.Driver");
        connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/cloudtopsy?zeroDateTimeBehavior=convertToNull","root","");
        ps = connection.prepareStatement("INSERT INTO users (uname,hashpword,fname,sname,email,permissionID)VALUES(?,?,?,?,?,?)");
        ps.setString(1,uname);
        ps.setString(2,pword);
        ps.setString(3,fname);
        ps.setString(4,lname);
        ps.setString(5,email);
        ps.setInt(6,permID);
        ps.execute();
        result = true;
        connection.close();
    }catch (Exception ex){
        System.out.println(ex);
        Logger.getLogger(ApplicationLogic.class.getName()).log(Level.SEVERE, null, ex);
        result = false;
        System.out.println(result + "inside DBWriter");
    }

    return result;
}
```

Code 3: Model Package Calls Database Access

5.2.2 Data Factory Pattern

The software implements two types of Factory Patterns; a factory class, and some factory methods. The methods are for choosing which type of broker is to be instantiated, while the factory class is used to construct the other custom object types, using information given from a general object that has been called 'data'.

```
public interface CustomDataFactory {
    public CustomDataType buildData(Data data);
}
```

Code 4: 1/2 Factory Methods Used for Constructing Broker



```
private void dbReadBrokerFactory(){
    dbReader = new DBReader();
}

private void dbWriteBrokerFactory(){
    dbWriter = new DBWriter();
}
```

Code 5: 2/2 Factory Methods Used for Constructing Broker

```
public class DataFactory implements CustomDataFactory{

    @Override
    public CustomDataType buildData(Data data){
        CustomDataType builtData = data;

        switch(data.getDataName()){
            case "Admin": builtData = buildUser(data);
                        break;
            case "Investigator": builtData = buildUser(data);
                        break;
            default:
                        break;
        }
        return builtData;
    }

    private CustomDataType buildUser(Data data){
```

Code 6: Factor Class to Build Range of Custom Objects

5.2.3 Data Access Object

The data access object is a pattern which allows the software to interact with a database by means of an abstract interface. In the case of this software, it was deemed necessary to implement this in order to gain access to case and user information.

```
public static ArrayList<String> getCases() throws ClassNotFoundException {
    ArrayList<String> caselist = new ArrayList<String>();
    Connection connection;
    PreparedStatement ps;

    try{
        Class.forName("com.mysql.jdbc.Driver");
        connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/cloudtoppsy?zeroDateTimeBehavior=convertToNull","root","");
        ps = connection.prepareStatement("SELECT cname FROM cases");
        ResultSet result = ps.executeQuery();

        while(result.next()){
            String casename = result.getString(1);
            caselist.add(casename);
        }
        connection.close();
    }catch (SQLException ex) {
        Logger.getLogger(ApplicationLogic.class.getName()).log(Level.SEVERE, null, ex);
    }

    return caselist;
}
```

Code 7: Call to MYSQL Database



5.2.4 Singleton Pattern

It was decided to use a singleton pattern to represent the current user; it seemed a logical option as only one person can log into the software instance at a time.

The lazy initialization style was used for the singleton, as it did not need to be instantiated until after the program was running and the user had confirmed their credentials.

```
public class CurrentUserSingleton {
    private static Users currentUser = null;

    private CurrentUserSingleton(){
        //Private constructor
        //There can only be one
    }

    public static Users getInstance(){
        if(currentUser == null){
            currentUser = new Users();
        }
        return currentUser;
    }

    public static Users getInstance(String uType,int uID, String uName, String uFName, String uLName){
        //Lazy initialization #GangOfFour
        if(currentUser == null){
            switch(uType){
                case "Admin": //admin permission ID is 1
                    currentUser = new Admin(uID, uName, uFName, uLName);
                    break;
                case "Investigator": //invest permission IF is 2
                    currentUser = new Investigator(uID, uName,uFName, uLName, "");
                    break;
                default:
                    currentUser = new Users();
                    break;
            }
        }
        return currentUser;
    }

    public static void logout(){
        currentUser = null;
    }
}
```

Code 8: Singleton Example

5.2.5 Broker Architectural Pattern

The broker architectural pattern was set up with a view to providing the application with a means to easily swap between using the current database and writing to the local machine. The broker also enables an easy way to change databases or database types if it is chosen to do so in the future.

The following code fragments demonstrate what would have been called, in the event that a reading or writing from the database occurred.

```
public interface DBWriteBroker {
    public void writeToDB(String instruction, Data data) throws IOException;
}
```

Code 9:DBWriteBroker Interface



```
public interface DBReadBroker {  
    public Data readFromDB(String instruction, String keywords);  
}
```

Code 10: DBReadBroker Interface

5.3 Application pages

Throughout the project, there are several common components across the classes. These similarities are in the way the frames are initiated and configured. This section will detail the sections of the project that are considered to be the most technical, including code extracts from these sections. For a general overview of the project and to understand its workings, refer to chapter 4 subsections 2 & 3.

5.3.1 Login

The first element of the application that was implemented was the login-in function. Upon launch, the application presents the user with a login function. It requires the username and password from the user. Once inserted the sign-in activity is started. The login frame calls the application logic login function.

```
try {  
    value = appLogic.login(usern, passw);  
} catch (IOException ex) {
```

Code 11: Call to Application Logic Login Function

Inside the application logic, there is a function call to the database reader, to check if the username and corresponding password exist. Using the generate hash function, the software calculates the hash of the entered password with its salt and completes the check request. If true, the current user singleton function is instantiated.

```
public boolean login(String uname, String pword) throws IOException, ClassNotFoundException{  
    String hashedPassword = generateHash(pword);  
    boolean login = DBReader.login(uname, hashedPassword);  
    System.out.println();  
    if(login == true){  
        //user exists: call database to get user details  
        Data curUser = model.dbRead("GetUser",uname);  
        if(curUser.getData().size()>0){  
            currentUserSingleton.getInstance(curUser.getDataName(), Integer.parseInt(curUser.getData().get(0).get(0)),uname, "uFName","uLName");  
        }else{  
            //User exists but was not read in correctly; assign default.  
            currentUserSingleton.getInstance();  
        }  
        return true;  
    }else{  
        return false;  
    }  
}
```

Code 12: Login Function



Back to the login frame, the software now has a logged in user with the current user singleton set. The system checks to see whether the user instance is an instance of an admin or an Investigator and returns the corresponding menu frame.

```
if( CurrentUserSingleton.getInstance() instanceof Admin){ //Admin
    CTMenuFrameAdmin menu = new CTMenuFrameAdmin(parent, new AdminLogic()); //adLogic here
}else if (CurrentUserSingleton.getInstance() instanceof Investigator){ //FInvestigator
    CTMenuFrameInvst menu = new CTMenuFrameInvst(parent, new InvstLogic());
}else{
```

Code 13: User Instance Check call

5.3.2 Menu Frames

The menu frame of the Investigator and admin are configured similarly. They contain a welcome message to the user, several button options, and a log out option. The welcome message pulls the username from the current user singleton as seen in the code below. In the case of the Investigator it also pulls the location of the open database.

```
curUser = CurrentUserSingleton.getInstance();
uname = curUser.getUserName();
curDir = curUser.getCurDir();
```

Code 14: Current User Variable Calls

The call to the current user singleton gets the instance of the user and then calls depending on that instance, allowing the frame to display information related to the singleton. It implements the lazy initialization otherwise known as the gang of four as seen below:

```
public static Users getInstance(String uType,int uID, String uName, String uFName, String uLName){
    //Lazy initialization #GangOfFour
    if(currentUser == null){
        switch(uType){
            case "Admin": //admin permission ID is 1
                currentUser = new Admin(uID, uName, uFName, uLName);
                break;
            case "Investigator": //invest permission IF is 2
                currentUser = new Investigator(uID, uName,uFName, uLName, "");
                break;
            default:
                currentUser = new Users();
                break;
        }
    }
    return currentUser;
}
```

Code 15: User Instance Check Function



5.3.3 Create Case

The create case frame requires the Investigator to enter details related to the case such as; name, description, and location of the disk image. It then shows the Investigator where the location of the created database will be stored. As will be shown in the next section, the location of the database is important for opening a case, but there is also a shortcut created. After the submit button has been pressed, a call is made to the functions that store the case data and create the case database.

```
OptionPane.showMessageDialog(null, "Creating database now: This may take a long time! You will be notified upon completion!");
result = inLogic.createCase(cimagepath, cname, cdesc, cdbpath);
if(result == true){
    if(inLogic.storeCaseData(cname, cdesc, cimagepath, cdbpath)){
        JOptionPane.showMessageDialog(null, "The case has been created!");
        curUser.setCurDir(curDir);
        parent.setVisible(true);
        dispose();
    }else{
        JOptionPane.showMessageDialog(null, "The case database has been created, but we could not store case details!");
    }
}else{
    JOptionPane.showMessageDialog(null, "There has been an error: There has been a problem creating the case!");
}
}else{
    JOptionPane.showMessageDialog(null, "There has been an error: Please ensure all fields are filled!");
}
```

Code 16: Call to Investigator Logic and Checks

The create case function is the first time that The Sleuth-Kit-Case object is used. It is declared and an SQLite database is created. The ‘make-Add-Image-Process’ function is then called and run. This function takes the data inside the image file that was passed in and stores it inside the database it just created.

```
try{
    //Creates the database and configures it
    SleuthkitCase sk = SleuthkitCase.newCase(imagepath + ".db");

    //Timezone input
    String timezone = "";
    curUser = CurrentUserSingleton.getInstance();
    String curUserName = curUser.getUserName();

    //Add image process, number of steps process but this returns an object that allows it to happen
    //Timezone, addUnallocSpace, noFatFsOrphans
    SleuthkitJNI.CaseDbHandle.AddImageProcess process = sk.makeAddImageProcess(timezone,false,false,"");

    //Creates a new arraylist of strings
    ArrayList<String> paths = new ArrayList<String>();
    //adds the image path to the array list
    paths.add(imagepath);

    //Trying to do something
    try{
        //System.out.println("Logger output 1");
        process.run(UUID.randomUUID().toString(), paths.toArray(new String[paths.size()]), 0);
        DBWriter.addCase(cname, cdesc, cdbpath, curUserName);
        result = true;
        //System.out.println("Logger end 1");
    } catch (TskDataException ex){
        //System.out.println("Logger output 2");
        Logger.getLogger(CTCreateCase.class.getName()).log(Level.SEVERE, null, ex);
        //System.out.println("Logger end 2");
    }
}
```

Code 17: The Sleuth Kit Case DB and Passing Data in



5.3.4 Open Case

The Investigator has two options when opening a case; they can select a database file using the browse function or they can select a case from the existing case drop down. In both instances, the selected case is presented to the Investigator before they decide to open it. After selecting the case to open, the system sets the currently opened case inside the user singleton to the case that the Investigator has chosen.

```
class OpenL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        fileChooser = new JFileChooser();
        int rval = fileChooser.showOpenDialog(CTOpenCase.this);
        if (rval == JFileChooser.APPROVE_OPTION) {
            CImageFlab.setText("Database Chosen");
            CImageF.setText(fileChooser.getCurrentDirectory().toString() + "\\ " + fileChooser.getSelectedFile().getName());
            CImageF.setForeground(Color.BLACK);
            dbDir.setText(fileChooser.getCurrentDirectory().toString() + "\\ " + fileChooser.getSelectedFile().getName());
            curUser.setCurDir(fileChooser.getCurrentDirectory().toString() + "\\ " + fileChooser.getSelectedFile().getName());
            curDir = fileChooser.getCurrentDirectory().toString() + "\\ " + fileChooser.getSelectedFile().getName();
            caseLab.setText("Database Chosen");
        }
        if (rval == JFileChooser.CANCEL_OPTION) {
            CImageF.setText("No Image Selected");
            CImageF.setForeground(Color.red);
        }
    }
}
```

Code 18: Browse Button Singleton Declaration

```
//action listener for the combobox
fileext.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){

        JComboBox fileext = (JComboBox) event.getSource();
        Object selected = fileext.getSelectedItem();

        try {
            curDir = inLogic.getDBLoc(selected.toString());
            curUser.setCurDir(curDir);
            CImageFlab.setText("Case Chosen");
            caseLab.setText("Case Chosen");
            dbDir.setText(curDir);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(CTOpenCase.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
});
```

Code 19: Drop Down Singleton Declaration

5.3.5 Establish Cloud Use

To establish the use of cloud storage systems, the Investigator clicks the start search button. The frame calls the Investigator logic function to check for cloud use and passes the directory location with it.



```
try {
    files = inLogic.checkCloudUse(curDir);
} catch (TskCoreException | SQLException | ClassNotFoundException ex) {
    Logger.getLogger(CTEstablishCU.class.getName()).log(Level.SEVERE, null, ex);
}
```

Code 20: ECU Frame Calls Investigator Logic

The Sleuth Kit object is set to open the case, passing in the path to the image that is associated with the open case. It then reads through every element of the case database looking for instances where data related to the cloud storage system are used. The system then returns a list of remnants found to the Investigator, to allow the addition of the finding to the case report.

```
public static ArrayList<String[]> checkCloudUse(String imagepath) throws TskCoreException, SQLException, ClassNotFoundException{
    SleuthkitCase existingCase = SleuthkitCase.openCase(imagepath);
    ArrayList<String[]> fileDataList = new ArrayList<String[]>();
    String[] fileData = {};

    // print out all the images found, and their children
    List<Image> images = existingCase.getImages();
    for (Image image : images) {
        System.out.println("Found image: " + image.getName());
        System.out.println("There are " + image.getChildren().size() + " children.");
        for (Content content : image.getChildren()) {
            System.out.println("'" + content.getName() + "' is a child of " + image.getName());
        }
    }

    // print out all .txt files found
    List<AbstractFile> files = existingCase.findAllFilesWhere("LOWER(parent_path) LIKE LOWER('%')");

    Object [][] recordArr = new Object[files.size()][3];
    int i = 0;

    for(AbstractFile file: files){
        fileData = new String[4];
        fileData[0] = (String.valueOf(file.getId()));
        fileData[1] = file.getName();
        fileData[3] = file.getParentPath();

        //Dropbox
        if(fileData[1].contains("dropbox")){
            fileData[2] = "Dropbox";
            //System.out.println("It has dropbox file");
            i++;
        }else if(fileData[3].contains("dropbox")){
            fileData[2] = "Dropbox";
            //System.out.println("It has dropbox dirr");
            i++;
        }
    }
}
```

Code 21: Establish Cloud Use Function

5.3.6 Directory & File Search

Directory and file search are both a form of key word searching. The table is updated with data which contains the string that is passed in from the user. After calling the Investigator logic function, The Sleuth Kit Case object is opened, using the open case function call. The system searches the database to check if the directory or the file, contain the word that the user has passed in. This is done by using the ‘find-all-files-where’ function call as can be seen below.



```

public ArrayList<String[]> getExtFiles(Object selected, String imagepath) throws TskCoreException, SQLException{
    SleuthkitCase existingCase = SleuthkitCase.openCase(imagepath);

    ArrayList<String[]> fileDataList = new ArrayList<String[]>();

    // print out all the images found, and their children
    List<Image> images = existingCase.getImages();
    for (Image image : images) {
        System.out.println("Found image: " + image.getName());
        System.out.println("There are " + image.getChildren().size() + " children.");
        for (Content content : image.getChildren()) {
            System.out.println("'" + content.getName() + "' + " is a child of " + image.getName());
        }
    }

    // print out all .txt files found
    List<AbstractFile> files = existingCase.findAllFilesWhere("LOWER(name) LIKE LOWER('%" + selected + "')");

    for (AbstractFile file : files) {
        String[] fileData = new String[3];
        fileData[0] = (String.valueOf(file.getId()));
        fileData[1] = file.getName();
        fileData[2] = file.getParentPath();
        fileDataList.add((String[])fileData);
    }
    existingCase.close();
    return fileDataList;
}

```

Code 22: Key Word Searching Example

5.3.7 View Active Cases

The view active cases GUI allows the user to select from a list of active cases. After they make the selection, the case information and table are updated. To get the names of all the cases for the dropdown list, a call to the application logic is made to the function ‘get-open-cases’. The GUI is then updated after selection with the following code. There are two other function calls inside this code; they get the data related to the case information and get the findings of the case that have been selected previously. This is done using the Data Access Object design pattern demonstrated in section 5.2.3.

```

//action listener for the combobox
fileext.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){

        tabmodel.setRowCount(0);
        JComboBox fileext = (JComboBox) event.getSource();
        Object selected = fileext.getSelectedItem();
        String column[]={"ID","File","Directory"};
        String [] caseinfo = null;
        try {
            caseinfo = appLog.getCaseInfo(selected.toString());
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(CTViewReport.class.getName()).log(Level.SEVERE, null, ex);
        }

        cdbLoc.setText(caseinfo[0]);
        copen.setText(caseinfo[1]);
        cclose.setText(caseinfo[2]);

        cdesc.setText(caseinfo[4]);

        try {
            row = appLog.getCaseData(caseinfo[3]);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(CTViewReport.class.getName()).log(Level.SEVERE, null, ex);
        }

        Iterator i = row.iterator();
        while(i.hasNext()){
            tabmodel.addRow((String[]) i.next());
        }
    }
});

```

Code 23: GUI Update After Case Selection



5.3.8 Close Case

The close case function allows the Investigator to close the case and generate a CSV of the findings from the case. The information related to the case is displayed, in the same format as ‘View Active Cases’ above. Following that, the Investigator has two options, to generate the CSV or to close the case. ‘Close-case’ updates the database with the current date to the close-date field.

```
public static boolean closeCase(String cdbloc) throws SQLException{
    Boolean result = false;
    Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/cloudtopsy?zeroDateTimeBehavior=convertToNull","root","");
    PreparedStatement ps;
    try
    {
        Class.forName("com.mysql.jdbc.Driver");
        connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/cloudtopsy?zeroDateTimeBehavior=convertToNull","root","");
        ps = connection.prepareStatement("UPDATE cases SET closedate = CURDATE() WHERE cdbdir = ?;");
        ps.setString(1,cdbloc);
        ps.execute();
        result = true;
        connection.close();
    }
}
```

Code 24: Setting Date to Close Case

The following function writes the data from the case to the CSV file and stores it in the location where the database is stored. The file will have the name formatted as follows: ‘nameOfDB_Cloudtopsy.csv’.

```
public boolean writeCSV(String cname) throws IOException, ClassNotFoundException{
    boolean result = false;
    String [] results = DBReader.getCaseInfo(cname);
    System.out.println("Save location : " + results[0]);
    FileWriter csvWriter = new FileWriter(results[0]+"_Cloudtopsy.csv");
    csvWriter.append("CaseID");
    csvWriter.append(",");
    csvWriter.append("CaseName");
    csvWriter.append(",");
    csvWriter.append("CaseDesc");
    csvWriter.append(",");
    csvWriter.append("Investigator");
    csvWriter.append(",");
    csvWriter.append("OpenDate");
    csvWriter.append(",");
    csvWriter.append("CloseDate");
    csvWriter.append("\n");
    String investID = DBReader.getInvestID(cname);
    String investName = DBReader.getInvestName(investID);
    csvWriter.append(results[3]); //CID
    csvWriter.append(",");
    csvWriter.append(cname); //CName
    csvWriter.append(",");
    csvWriter.append(results[4]); //CDesc
    csvWriter.append(",");
    csvWriter.append(investName); //Investigator name
    csvWriter.append(",");
    csvWriter.append(results[1].toString()); //Investigator name
    csvWriter.append(",");
    csvWriter.append(results[2]); //Investigator name
    csvWriter.append("\n");
    csvWriter.append("\n");
    csvWriter.append("FileID");
    csvWriter.append(",");
    csvWriter.append("FileName");
    csvWriter.append(",");
    csvWriter.append("FileDir");
    csvWriter.append("\n");
    ArrayList<String[]> casedata = getCaseData(results[3]);
    for(String[] rowData : casedata){
        csvWriter.append(String.join(",", rowData));
        csvWriter.append("\n");
    }
    csvWriter.flush();
    csvWriter.close();
}
```

Code 25: CSV Generating Function



5.3.9 Obtain Case Reports

In order for an Admin/ Manager to obtain a report related to a case, they must select a case from the dropdown menu similar to that of the ‘View-Active-Case’ in 5.3.7. Following that, they must select a save location for the file using the browse function.

```
class OpenD implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        fileChoser = new JFileChooser();
        fileChoser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int rVal = fileChoser.showOpenDialog(CTCaseReports.this);
        if (rVal == JFileChooser.APPROVE_OPTION) {
            caseName = fileext.getSelectedItem().toString();
            directory = fileChoser.getSelectedFile() + "\\";
            String caseLoc = caseName + " saved to " + directory;
            Details.setText(caseLoc);
        }
        if (rVal == JFileChooser.CANCEL_OPTION) {
        }
    }
}
```

Code 26: Browsing Local Computer for Save Location

Once the save location is selected, the system uses the same CSV generating function seen in 5.3.8 to generate the CSV file related to the case selected.

5.3.10 Logout

The logout function returns the user to the **Cloud-toppsy** launch page and resets all elements of the current user singleton. This is to ensure that when a new user logs in, none of the previous user’s data is presented.

```
public void logout(){
    CurrentUserSingleton.logOut();
}
```

Code 27: Call to Singleton Function

```
public static void logOut(){
    currentUser = null;
}
```

Code 28: Current User Set to Null



5.4 GUI Design

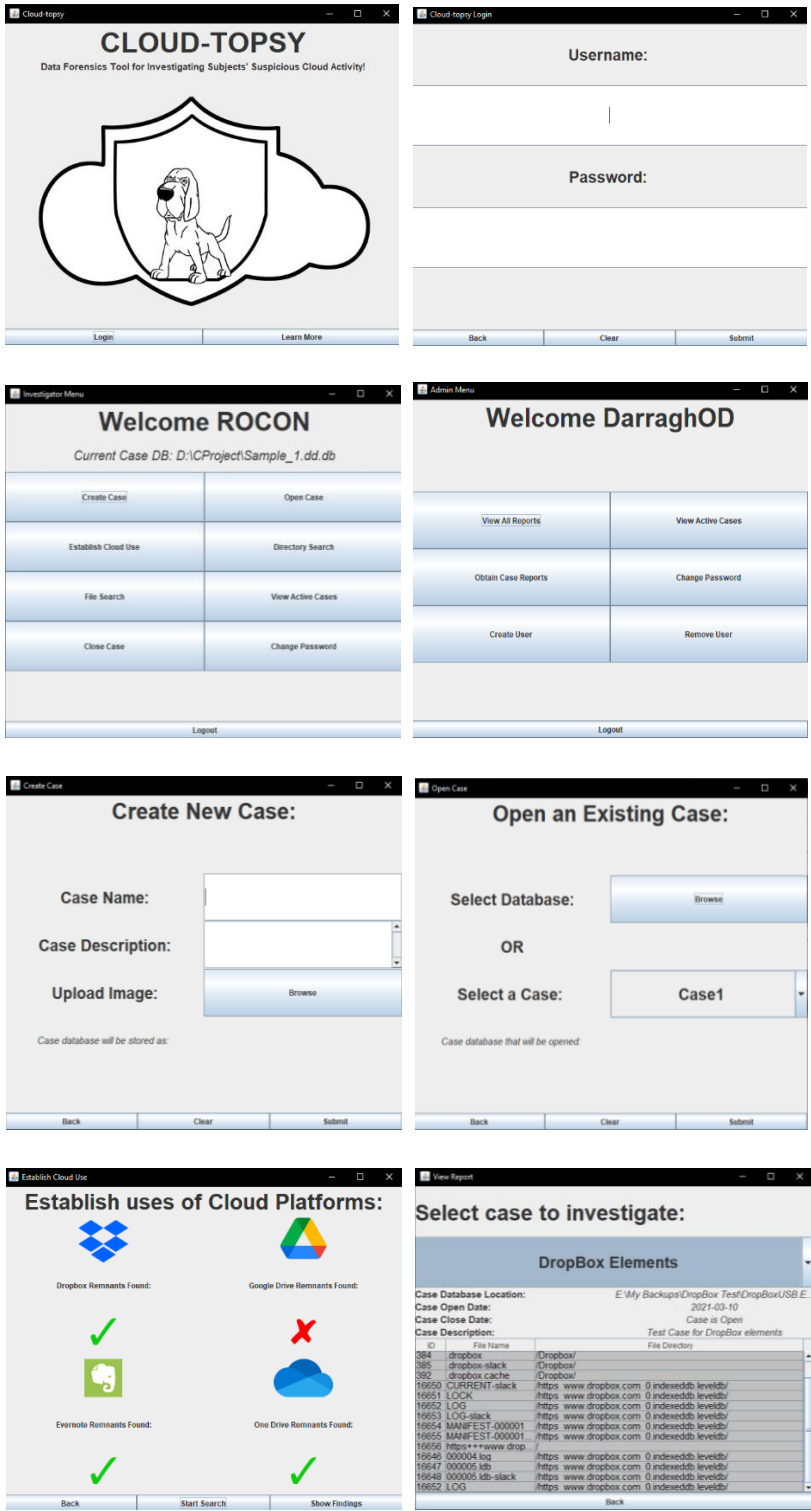


Figure 13: Final GUI Design

Displayed are a selection of the GUIs that have been created. They maintain the simple design that was originally planned, allowing the user to be able to navigate through the software with no prior knowledge of it.

The GUIs were created through the use of Java Swing JFrame, primarily using the Grid-Layout and Border-Layout styles. They are a form of dynamic user interfaces, as they change depending on which user is currently logged in, and whether or not a case has been opened.

In order to open a case, two options are presented. The user can either search the system for the database file, or they can select from the

cases that already exist inside of the database. J-Tables were used to return case remnant information from the database in the form of a table. In some cases, the user can select items from the table created and submit them into the case findings database.



As discussed in the design patterns section of this report (Section 5.2), the Model View Controller architectural pattern has been implemented. This means that if a modification or update to the GUI were to be implemented, the entire model would not be affected. In short, this means that the entire GUI could be swapped out for a more modern style, without the backend code that has been developed being affected.

5.5 Database Structure & Security

5.5.1 MYSQL Database

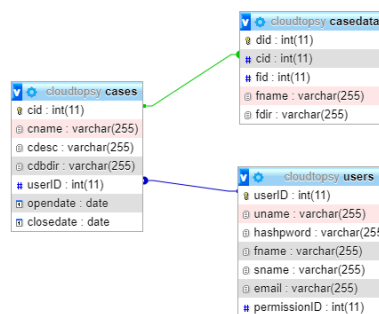


Figure 14: MYSQL Table Structure

The MYSQL database shown above was developed within phpMyAdmin using the MySQL coding language to construct the tables as well as to define the relationship between each table. The code extract below shows a sample of how the cases table was created:

```
CREATE TABLE cases (
  cid INT NOT NULL AUTO_INCREMENT ,
  cname VARCHAR(255) NOT NULL,
  cdesc VARCHAR(255) NOT NULL,
  cdbdir VARCHAR(255) NOT NULL,
  userID INT NOT NULL,
  opendate DATE,
  closedate DATE,
  PRIMARY KEY (cid));

ALTER TABLE cases
  ADD FOREIGN KEY(userID)
  REFERENCES users(userID)
  ON UPDATE CASCADE ON DELETE CASCADE;
```

Code 29: Sample MYSQL Code



5.5.2 SQLite Database

The SQLite database is created and filled by The Sleuth Kit package when the 'Create case' function is called, as described in 5.3.3 'Create Case'. Below are the tables inside the database that have been created:

account_relationships	tsk_db_info
account_types	tsk_db_info_extended
accounts	tsk_event_descriptions
blackboard_artifact_tags	tsk_event_types
blackboard_artifact_types	tsk_events
blackboard_artifacts	tsk_examiners
blackboard_attribute_types	tsk_file_layout
blackboard_attributes	tsk_files
content_tags	tsk_files_derived
data_source_info	tsk_files_derived_method
file_encoding_types	tsk_files_path
ingest_job_modules	tsk_fs_info
ingest_job_status_types	tsk_image_info
ingest_jobs	tsk_image_names
ingest_module_types	tsk_objects
ingest_modules	tsk_pool_info
reports	tsk_tag_sets
review_statuses	tsk_vs_info
tag_names	tsk_vs_parts

Figure 15: SQLite Table

5.5.3 Security

1. Salt & Hash Passwords

In order to secure passwords stored inside of the database, a salt and hash were used. A salt is a unique value that was added to the start of the password to create a different hash value. It added a layer of security to the hashing process, primarily against brute force attacks. This way, it would be difficult for a hacker to access the system without knowing the username and password of an Investigator or an admin. The hashing aspect means that the password has been turned into a scrambled representation of itself. It is a type of algorithm commonly used in the security of sensitive data. The hash and salting function can be seen in the code extract below:

```
public String generateHash(String input){
    input = SALT + input;
    StringBuilder hash = new StringBuilder();
    try {
        MessageDigest sha = MessageDigest.getInstance("SHA-1");
        byte[] hashedBytes = sha.digest(input.getBytes());
        char[] digits = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                           'a', 'b', 'c', 'd', 'e', 'f' };
        for (int idx = 0; idx < hashedBytes.length; ++idx) {
            byte b = hashedBytes[idx];
            hash.append(digits[(b & 0xf0) >> 4]);
            hash.append(digits[b & 0x0f]);
        }
    } catch (NoSuchAlgorithmException e) {
        // handle error here.
    }

    return hash.toString();
}
```

Code 30: Hash & Salt Function



2. SQL-Injection Protection

Due to issues with possible SQL-injection attacks, software measures were taken to prevent the possibility of such attacks. Pre-written SQL statements allow for supplying parameters in order for the statements to be executed, and allows the database to recognise the code, distinguishing it from the input data. Furthermore, the user input is automatically detected, and the supplied input is not able to change the intent; hence preventing an SQL-injection attack. This can be seen in every database call that is made inside the software:

```
PreparedStatement ps;  
try  
{  
    Class.forName("com.mysql.jdbc.Driver");  
    connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/cloudtopsy?zeroDateTimeBehavior=convertToNull","root","");  
    ps = connection.prepareStatement("UPDATE cases SET closedate = CURDATE() WHERE cdbdir = ?;");  
    ps.setString(1,cdbloc);  
    ps.execute();  
    result = true;  
    connection.close();  
}
```

Code 31: Prepared Statement Example

5.6 Issues Faced during implementation

5.6.1 Library Configurations

When configuring the libraries for The Sleuth Kit, the dynamic libraries and other dependencies had to be sourced. Regardless of the location they were stored in, The Sleuth Kit did not recognise that they existed. To overcome this problem, The Sleuth Kit JAR was broken down and the code was read through, to see if it was pointing to the correct location. Through trial and error, printing out return values of specific functions, it was found that the package was looking for the libraries in a folder that did not exist. This value was then hard coded to the location where the created libraries were stored, and the JAR files were successfully repackaged.



Chapter 6. Testing and Results

6.1 Junit Testing

The Junit 4.12 framework was selected for implementing test cases, it proved to be particularly useful when debugging specific sections of code. The two cases provided are for the Investigator class and the Admin class. The test runs through the get-permissions, get-last-name, get-current-directory and get-first-name for the Investigator. In terms of the Admin class, it tests the get-permissions, get-first-name and get-last-name. The code extracts and results are provided below:

```
/**
 * Test of getFirstName method, of class Investigator.
 */
@Test
public void testGetFirstName() {
    System.out.println("getFirstName");
    Investigator instance = new Investigator();
    String expResult = "Unknown";
    String result = instance.getFirstName();
    assertEquals(expResult, result);
}
/**
 * Test of getCurDir method, of class Investigator.
 */
@Test
public void testGetCurDir() {
    System.out.println("getCurDir");
    Investigator instance = new Investigator();
    String expResult = "Unknown";
    String result = instance.getCurDir();
    assertEquals(expResult, result);
}
/**
 * Test of getLastName method, of class Investigator.
 */
@Test
public void testGetLastName() {
    System.out.println("getLastName");
    Investigator instance = new Investigator();
    String expResult = "Unknown";
    String result = instance.getLastName();
    assertEquals(expResult, result);
}
/**
 * Test of getPermissions method, of class Investigator.
 */
@Test
public void testGetPermissions() {
    System.out.println("getPermissions");
    Investigator instance = new Investigator();
    int expResult = 0;
    int result = instance.getPermissions();
    assertEquals(expResult, result);
}

/**
 * Test of getFirstName method, of class Admin.
 */
@Test
public void testGetFirstName() {
    System.out.println("getFirstName");
    Admin instance = new Admin();
    String expResult = "Unknown";
    String result = instance.getFirstName();
    assertEquals(expResult, result);
}
/**
 * Test of getLastName method, of class Admin.
 */
@Test
public void testGetLastName() {
    System.out.println("getLastName");
    Admin instance = new Admin();
    String expResult = "Unknown";
    String result = instance.getLastName();
    assertEquals(expResult, result);
}
/**
 * Test of getPermissions method, of class Admin.
 */
@Test
public void testGetPermissions() {
    System.out.println("getPermissions");
    Admin instance = new Admin();
    int expResult = 0;
    int result = instance.getPermissions();
    assertEquals(expResult, result);
}
```

Code 32:JUnit Test Code Admin and Investigator Class

```
Testsuite: ModelLayer.InvestigatorTest
getPermissions
getLastName
getCurDir
getFirstName
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.071 sec

----- Standard Output -----
getPermissions
getLastName
getCurDir
getFirstName
-----
test:
Deleting: C:\Users\oconn\AppData\Local\Temp\TEST-ModelLayer.InvestigatorTest.xml
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 16: JUnit Testing Success Investigator Class



```
Testsuite: ModelLayer.AdminTest
getPermissions
getLastName
getFirstName
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.074 sec

----- Standard Output -----
getPermissions
getLastName
getFirstName
-----
test:
Deleting: C:\Users\oconn\AppData\Local\Temp\TEST-ModelLayer.AdminTest.xml
```

Figure 17: JUnit Testing Success Admin Class

6.2 Run-through Testing

A run through test can be seen in the video that has been submitted alongside this report. The video covers all fundamental aspects of the software. It does this by demonstrating a sample case. The case is based on a USB thumb drive that contains elements of the cloud storage solution known as Dropbox.

6.3 Result Outputs

6.3.1 Onscreen Output

The on-screen output displays data that was found to contain any **remnants** of the **cloud storage system**, in this case 'Dropbox'. It is in table format, displaying each element's file number, file name and the path to the file's directory. Location of the database, case open & close date and the case description are displayed above the remnants table.

Directory Search			View Report	
ID	File Name	Storage	File Directory	
403	3fa3a04209fb19d0...	Dropbox	/Dropbox/	dropbox.cache/new files/
404	3fa3a04209fb19d0...	Dropbox	/Dropbox/	dropbox.cache/new files/
405	c5c3ab360ab5fa10...	Dropbox	/Dropbox/	dropbox.cache/new files/
406	c5c3ab360ab5fa10...	Dropbox	/Dropbox/	dropbox.cache/new files/
407	d0b7435929c4274...	Dropbox	/Dropbox/	dropbox.cache/new files/
408	d0b7435929c4274...	Dropbox	/Dropbox/	dropbox.cache/new files/
409	3ac700cc02cd8e2...	Dropbox	/Dropbox/	dropbox.cache/new files/
410	3ac700cc02cd8e2...	Dropbox	/Dropbox/	dropbox.cache/new files/
411	3fa3a04209fb19d0...	Dropbox	/Dropbox/	dropbox.cache/new files/
412	3fa3a04209fb19d0...	Dropbox	/Dropbox/	dropbox.cache/new files/
413	c5c3ab360ab5fa10...	Dropbox	/Dropbox/	dropbox.cache/new files/
414	c5c3ab360ab5fa10...	Dropbox	/Dropbox/	dropbox.cache/new files/
415	d0b7435929c4274...	Dropbox	/Dropbox/	dropbox.cache/new files/
416	d0b7435929c4274...	Dropbox	/Dropbox/	dropbox.cache/new files/
417	old files	Dropbox	/Dropbox/	dropbox.cache/
418	.	Dropbox	/Dropbox/	dropbox.cache/old files/
419	.	Dropbox	/Dropbox/	dropbox.cache/old files/
420	47ddc7c93a29b74...	Dropbox	/Dropbox/	dropbox.cache/old files/
421	.	Dropbox	/Dropbox/	dropbox.cache/old files/
422	.	Dropbox	/Dropbox/	dropbox.cache/old files/
423	tmp dirs	Dropbox	/Dropbox/	dropbox.cache/
424	.	Dropbox	/Dropbox/	dropbox.cache/tmp dirs/
425	.	Dropbox	/Dropbox/	dropbox.cache/tmp dirs/
8681	dropbox.png	Dropbox	/Dropbox/My PC (DESKTOP-0K7FC9U)/Documents...	
8692	dropbox.png-slack	Dropbox	/Dropbox/My PC (DESKTOP-0K7FC9U)/Documents...	
8693	evernote.png	evernote	/Dropbox/My PC (DESKTOP-0K7FC9U)/Documents...	
8694	evernote.png-slack	evernote	/Dropbox/My PC (DESKTOP-0K7FC9U)/Documents...	
8699	onedrive.png	OneDrive	/Dropbox/My PC (DESKTOP-0K7FC9U)/Documents...	
8700	onedrive.png-slack	OneDrive	/Dropbox/My PC (DESKTOP-0K7FC9U)/Documents...	
Back			Submit Selection	

Select case to investigate:		
DropBox Elements		
Case Database Location: E:\My Backups\DropBox Test\DropBoxUSB.E...		
Case Open Date: 2021-03-10		
Case Close Date: Case is Open		
Case Description: Test Case for DropBox elements		
ID	File Name	File Directory
379	dropbox device	/
380	dropbox device-sla...	/
384	dropbox	/Dropbox/
385	dropbox-slack	/Dropbox/
392	dropbox.cache	/Dropbox/
16650	CURRENT-slack	/https www.dropbox.com 0 indexeddb leveldb/
16651	LOCK	/https www.dropbox.com 0 indexeddb leveldb/
16652	LOG	/https www.dropbox.com 0 indexeddb leveldb/
16653	LOG-slack	/https www.dropbox.com 0 indexeddb leveldb/
16654	MANIFEST-000001	/https www.dropbox.com 0 indexeddb leveldb/
16655	MANIFEST-000001	/https www.dropbox.com 0 indexeddb leveldb/
16656	https+++www.drop...	/https www.dropbox.com 0 indexeddb leveldb/
16646	000004.log	/https www.dropbox.com 0 indexeddb leveldb/
16647	000005.ldb	/https www.dropbox.com 0 indexeddb leveldb/
Back		

Figure 18: Onscreen Output Sample



6.3.2 CSV Output

The outputted CSV file contains the elements chosen from the onscreen results. These are files chosen by the **Investigator** that are deemed to be **suspicious** on the **suspect's machine**. This file contains information related to the case such as; case identification, case name, case description, Investigator name and dates related to the case. It also contains file information similar to that found in the **on-screen view**.

CaseID	CaseName	CaseDesc	Investigator	OpenDate	CloseDate
5	DropBox Elements	Test Case for DropBox elements	1	10/03/2021	Case is Open
FileID	FileName	FileDir			
379	.dropbox.device	/			
380	.dropbox.device-slack	/			
384	.dropbox	/Dropbox/			
385	.dropbox-slack	/Dropbox/			
392	.dropbox.cache	/Dropbox/			
16650	CURRENT-slack	/https_www.dropbox.com_0.indexeddb.leveldb/			
16651	LOCK	/https_www.dropbox.com_0.indexeddb.leveldb/			
16652	LOG	/https_www.dropbox.com_0.indexeddb.leveldb/			
16653	LOG-slack	/https_www.dropbox.com_0.indexeddb.leveldb/			
16654	MANIFEST-000001	/https_www.dropbox.com_0.indexeddb.leveldb/			
16655	MANIFEST-000001-slack	/https_www.dropbox.com_0.indexeddb.leveldb/			
16656	https+++www.dropbox.com	/			
16646	000004.log	/https_www.dropbox.com_0.indexeddb.leveldb/			
16647	000005.ldb	/https_www.dropbox.com_0.indexeddb.leveldb/			
16648	000005.ldb-slack	/https_www.dropbox.com_0.indexeddb.leveldb/			

Figure 19: CSV Output Sample



6.4 Recovered Architecture System Class Diagram

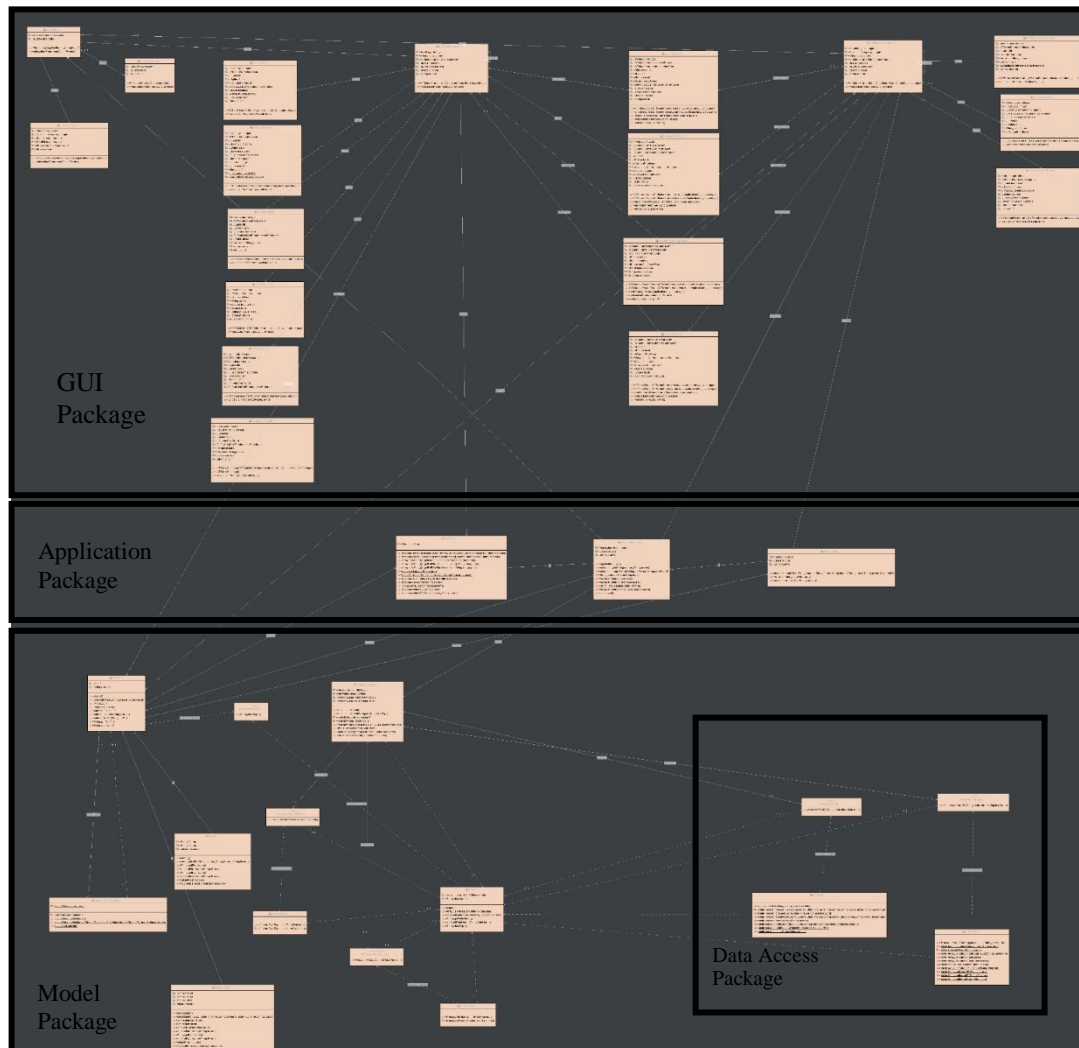


Figure 20: Recovered Class Diagram



Chapter 7. Discussion & Conclusion

7.1 Discussion

The implementation of *Cloud-topsy* has successfully allowed the Investigator to prove that cloud storage platforms were present on the suspects device. The objective of this project has therefore been achieved.

It is interesting to note that the current software GUI is widget sized. This is advantageous because it can be used simultaneously with other applications; determining whether specific image files are relevant for further investigation.

Through further development, this project could be improved in several ways. The first improvement would be to provide access to each of the files obtained in the investigation of the case. The Investigator would then be able to not only identify that cloud storage solutions were used, but also do a more in-depth analysis into what exactly took place within the remnants.

An updated GUI would enhance the user experience and make it more aesthetically pleasing. This would make *Cloud-topsy* more marketable product.

The addition of other toolkits would enhance the usability of *Cloud-topsy* and allow for more thorough investigations to take place within the software.

7.2 Conclusion

The development of *Cloud-topsy* has successfully resulted in an easy-to-use software that can quickly identify and provide evidence that cloud storage platforms were used by locating remnants on suspects' machines. It can be concluded that NetBeans was the correct solution for the implementation. It provided a simple code editor and made the build and run process seamlessly simplistic. It also played a major part in the testing of the software. It would not have been possible to create this software without the features of the SDK that is NetBeans.

It can also be concluded that Java and The Sleuth Kit were the correct choices for this software development. The features of Java allowed the application processes to take place and also to implement a graphical user interface around it. The Sleuth Kit allowed a full-scale



analysis to be undertaken on the supplied disk image, making it the backbone of the entire software. Both Java and The Sleuth Kit are well supported and documented as they are widely used in software development.

XAMPP was useful in the creation of this software as it allowed the software to access the locally hosted database from any machine inside of the network. This made a very necessary step straight forward.

Cloud-toppsy could become the next major computer forensic platform for smaller forensic investigation laboratories. Its ability to seamlessly connect Investigators with their superiors while undertaking case investigations is both timesaving and collaborative.



References

- [1] Amazon, "What is Cloud Computing", Amazon Web Services, Inc., 2020. [Online]. Available: <https://aws.amazon.com/what-is-cloud-computing/>. [Accessed: 05- Oct- 2020].
- [2] HSE Ireland, "Coronavirus", HSE, 2020. [Online]. Available: <https://www2.hse.ie/coronavirus/>. [Accessed: 05- Oct- 2020].
- [3] E. Burke-Kennedy, "Ireland had one of highest rates of home-working during Covid-19 crisis", The Irish Times, 2020. [Online]. Available: <https://www.irishtimes.com/business/work/ireland-had-one-of-highest-rates-of-homeworking-during-covid-19-crisis-1.4369346>. [Accessed: 08- Oct- 2020].
- [4] K. R. Choo, C. Esposito and A. Castiglione, "Evidence and Forensics in the Cloud: Challenges and Future Research Directions", IEEE Cloud Computing, 2017. [Online]. Available: <https://ieeexplore.ieee.org/document/7962121>. [Accessed: 10-Oct-2020].
- [5] M. Sang Chang, "Forensic investigation of Amazon Cloud Drive on Windows 10", Ijiset.com, 2016. [Online]. Available: http://ijiset.com/vol3/v3s6/IJISSET_V3_I6_69.pdf. [Accessed: 15- Oct- 2020].
- [6] M. I. Ali et al., "Security Challenges and Cyber Forensic Ecosystem in IoT Driven BYOD Environment", IEEE Access, 2020. Available: <https://ieeexplore.ieee.org/document/9199866>. [Accessed: 15- Oct- 2020].
- [7] E. Webb-Hobson, "Digital Investigations in the Cloud", Docuri.com, 2017. [Online]. Available: https://docuri.com/download/digital-investigations-in-thecloud_59c1e21af581710b286a47a1_pdf. [Accessed: 15- Oct- 2020].
- [8] W. Buyu and E. Odira Abade, "Forensic Analysis of Dropbox Data Remnants on Windows 10", ResearchGate, 2020. [Online]. Available: https://www.researchgate.net/publication/342991973_Forensic_Analysis_of_Dropbox_Data_Remnants_on_Windows_10. [Accessed: 21- Oct- 2020].



- [9] F. Marturana, G. Me and S. Tacconi, "A case study on digital forensics in the cloud", ResearchGate, 2012. [Online]. Available: https://www.researchgate.net/publication/234063077_A_Case_Study_on_Digital_Forensics_in_the_Cloud. [Accessed: 21- Oct- 2020]. 23
- [10] F. McClain, "Dropbox Forensics - Forensic Focus", Forensic Focus, 2020. [Online]. Available: <https://www.forensicfocus.com/articles/dropbox-forensics/>. [Accessed: 21-Oct2020].
- [11] H. Chung, J. Park, S. Lee, and C. Kang, "Digital Forensic Investigation of Cloud Storage Services", ResearchGate, 2012. [Online]. Available: https://www.researchgate.net/publication/257687927_Digital_Forensic_Investigation_of_Cloud_Storage_Services. [Accessed: 21- Oct- 2020].
- [12] C. Altheide and H. Carvey, Digital forensics with open source tools. Burlington, MA: Syngress, 2011.
- [13] B. Carrier, "Trackers - SleuthKitWiki", Wiki.sleuthkit.org, 2020. [Online]. Available: <http://wiki.sleuthkit.org/index.php?title=Trackers>. [Accessed: 04- Nov- 2020].
- [14] P. Smith, "Waterfall model", En.wikipedia.org, 2010. [Online]. Available: https://en.wikipedia.org/wiki/Waterfall_model. [Accessed: 11- Nov- 2020].
- [15] J. Denman, "Get started with these Agile basics", SearchSoftwareQuality, 2020. [Online]. Available: <https://searchsoftwarequality.techtarget.com/feature/Agile-basics-FAQGetting-started-with-Agile>. [Accessed: 11- Nov- 2020].
- [16] "How to Conduct Efficient Examinations with EnCase Forensic 8 06", Youtube.com, 2021. [Online]. Available: <https://www.youtube.com/watch?v=n5y1qtFf4XA>. [Accessed: 06-Feb- 2021]
- [17] "Fig. 1. EnCase Forensic main screen", ResearchGate, 2021. [Online]. Available: https://www.researchgate.net/figure/EnCase-Forensic-main-screen_fig1_238042997. [Accessed: 06- Feb- 2021]



[18] "Autopsy: Description", *Sleuthkit.org*, 2021. [Online]. Available: <https://www.sleuthkit.org/autopsy/desc.php>. [Accessed: 06- Feb- 2021]

[19] "Autopsy - Digital Forensic Program and Sleuth Kit GUI - SecTechno", *SecTechno*, 2021. [Online]. Available: <https://sectechno.com/autopsy-digital-forensic-program-and-sleuth-kit-gui/>. [Accessed: 09- Feb- 2021]

[20] "A brief comparison of Java IDE's: NetBeans Vs Eclipse – Linux Hint", *Linuxhint.com*, 2021. [Online]. Available: https://linuxhint.com/netbeans_vs_eclipse/. [Accessed: 09- Feb- 2021]

[21] "Best Frameworks for Desktop Application Development - DZone Open Source", *dzone.com*, 2021. [Online]. Available: <https://dzone.com/articles/best-frameworks-for-desktop-application-developmen>. [Accessed: 12- Feb- 2021]

[22] M. Phan, "MVC architecture pattern", *Ducmanhphan.github.io*, 2021. [Online]. Available: [https://ducmanhphan.github.io/2019-07-27-MVC-architecture-pattern/#:~:text=Model%2DView%2DController%20\(usually,and%20accepted%20from%20the%20user](https://ducmanhphan.github.io/2019-07-27-MVC-architecture-pattern/#:~:text=Model%2DView%2DController%20(usually,and%20accepted%20from%20the%20user). [Accessed: 12- Feb- 2021]

