Computer Engineering Department

Mini Search Engine

Prepared by

Serdar MUSTAFAPAŞA

Spring 2020-2021

ABSTRACT

People have searched and researched a lot from ancient times; searching and researching is in human nature, so different search methods have been developed in different areas, and in computer science, searching is the process of finding or identifying a given item with a value point in a list of values, it is the algorithmic process of finding a specific item, it decides if there is a match of search key that is presented in the data or not.

A search engine is a software implemented by programming and guided by computer and user; many search engines have been built and can be found in operating systems and big companies, also the internet. A mini search engine is designed that can be used by using the Eclipse IDE and an operating system. This search engine is provided with security basics and extensible unique command patterns. The project is developed with the Java programming language since it includes natural processing language and safe; it also comes with better high-level concurrency tools. For backEnd, object-oriented Java and data structures were used; and for frontEnd, Java Swing is used since it is easier for users to deal with GUI interfaces.

The information and search results are searched by a search algorithm (B-Tree). This search algorithm matches results; these results are sent to and fetched from an In-Memory Database connected to the application since it has a faster response time than traditional databases, so the search results are faster. Finally, there is user authentication; the username will be connected to the running Operating System and will automatically know that Operating System. Also, the password will be set by the user and encrypted automatically for security improvement. If the user enters a wrong password, an error message will occur.

Following is the test analysis section, which discusses whether the proposed system met its objectives. Performance is also evaluated near the end of the paper, along with possible extensions of the system.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Meaning
OS	Operating System
FS	File System
OOP	Object-Oriented Programming
DS	Data Structure
BST	Binary Search Tree
DFS	Depth-First Search
BFS	Breadth-First Search
Regex	Regular Expression
Parsing	Syntax Analysis
Lexing	Lexical Analysis / Tokenization
UI	User Interface
UML	Unified Modeling Language
RAD	Rapid Application Development
IDE	Integrated Development Environment
AWT	Abstract Window Toolkit
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
JDT	Java Development Tools
JNI	Java Native Interface
Javac	Java Programing Language Compiler
JAR	Java Archive
POJO	Plain Old Java Object
WORA	Write Once Run Anywhere

GLOSSARY

Several points need to explain by definition. Hence, here are definitions of terms that need to know.

❖ **General:**

- **File system:** A file system is an operating procedure that manages the data on a storage area; it is a logical component that manages the storage area's internal operations such as storage management, file naming, directories and folders, data, access rules, and privileges.
- **File path traversal:** It is also known as (directory traversal). It is a security attack that allows an attacker to access the file system; after that, the attacker will be able to access the files stored outside the root folder; this might include code and data.
- **DS: Data structure.** A data structure is a collection of data information, functions, and operations that can be applied to the data and their relationships.
- **OOP: Object-Oriented Programming.** It is a programming paradigm based on creating objects that contain both data and functions.
- **Cryptography:** The application and study of techniques for secure communication in the presence of enemies. Typically it is about creating and analyzing protocols that overcome the influence of enemies and deal with various aspects of information security.
- **Encryption:** It is the process of encoding a message that contains information that is converted into a secret code that hides the information's true meaning; this process converts the original representation of the information.
- **Command pattern:** The command pattern is an object design pattern that allows us to wrap all information needed for decoupling between the sender and the receiver. This information includes the method name, encapsulating a request as an object that owns the method and values for the method parameters.
- **In-memory database:** This database is designed to attain minimal response time by eliminating the need to access disks because it relies primarily on memory for data storage.
- **Tree data structure:** Tree stores the information naturally in the form of hierarchy. A tree contains nodes (data) and connections (edges).
- **Root:** It is the top node in the tree, the prime ancestor.
- **Child Node:** A node contains a value or condition. The child node is connected to another node when moving away from the root, an immediate descendant.

- **Leaf:** A node with no children.
- **B-Tee:** B-Tree is used for locating special keys, The algorithm uses the key from the key-value pair to find a location, and then it stores the key–value pair at that specific location; it has an efficient search time. It allows to keep a sorted list of numbers; the value of nodes on the left is less than the value of the root. Each node has a maximum of two children.
- **Tree traversal:** It is also known as (Tree search); it refers to the process of checking and updating each node in a tree data structure; it has two different categories (DFS) (BFS) and can be classified by the order in which nodes are visited.
- **DFS: Depth-First Search.** It is an algorithm for traversing the tree; the algorithm starts at the root node and searches as far as possible along each branch.
- **BFS: Breadth-First Search.** It is an algorithm for traversing the tree; the algorithm starts at the root node and searches through all neighboring nodes before moving to the next depth in the tree.
- **UNIX:** It is an operating system.
- **Regex: Regular Expression.** It is used in search engines, and it is a sequence of characters that allows the creation of patterns that helps match, locate, and manage text.
- **Parsing:** It is also called (Syntax Analysis), and it is the process of analyzing a string of symbols and conforming to the rules of formal grammar.
- **Lexing:** It is also called (Lexical Analysis or Tokenization), and it is the process of converting a sequence of characters into a sequence of tokens.
- **Grep:** It is a UNIX command that can be used in other operating systems also. It is used to search plain-text data sets for lines that match a regular expression; it can also open and look through files directly.
- **Epoch Time:** It is a Unix time, and it counts second by second since 1980.
- **Functional Requirement:** Functional requirements are features "visible" to the user. usually initiated by the stakeholders of the system such as report generation, login, and sign-up
- **Non-Functional Requirements:** They are requirements that define how the system will work what to do, for example, security, reliability, and sustainability.

- **Diagrams:**

- **Block Diagram:** A block diagram is a custom, high-level Flow chart used in engineering. It is used to design new systems or identify and improve existing ones. Its structure provides a high-level overview of fundamental system components, key process participants, and critical business relationships.
- **Flow Chart:** A flowchart is a visual representation of the steps and decisions required to carry out a transaction. Each next step is outlined in a diagram. Connecting lines and direction arrows connect steps. This allows logically follow the process from start to finish.
- **UML: Unified Modeling Language.** UML diagram is a diagram intended to visually represent a system with its main actors, roles, actions, or classes to understand the document information about the system.
- **Class Diagram:** Describes the system's structure by showing the system's classes, attributes, and methods with the relationship.
- **Object Diagram:** It provides a minimal architecture of the instances in the system and the relationships between them; it uses a notation similar to that used in the class diagram.
- **Activity Diagram:** It is another essential diagram in UML, and it is an advanced version of a flow chart that models flow from one activity to another.
- **Sequence Diagram:** It shows how operations are carried out and what messages are sent, and when.
- **Colloboration Diagram:** It is used to show how objects are defined and clarify the roles of the objects that perform particular flow events of a particular use case.
- **Use-Case Diagram:** It specifies the expected behavior and not the exact method of making it happen (what) and (how); it helps design a system from the user’s perspective.
- **Component Diagram:** It is used to model the physical side of object-oriented systems that are used to document and visualize component-based systems.
- **Deployment Diagram:** It shows the configuration and the components of runtime processing nodes; it shows the physical aspects of an object-oriented system.
- **Waterfall Model**: A waterfall model transfers the project activities into sequential phases, where each phase depends on the deliverables of the previous one.
- **RAD: Rapid Application Development.** This model allows changes in any stage which was very costly, it has multiple cycles, and each cycle brings new features.
- **Gantt Chart:** It is helpful for planning and scheduling projects. It helps to evaluate the time of the project, identify the resources needed, and plan the order in which to complete tasks. They are also helpful for managing dependencies between tasks.

- ❖ **Development Environment:**

- **AWT: Abstract Window Toolkit.** It is considered a subpackage of the java package. It is a built-in package that includes elementary GUI entities and containers used to hold components in a specific layout.
- **Swing:** It is a much more comprehensive graphics library that enhances the AWT. It also stands under the Java package as a subpackage. In comparison with AWT component classes, it begins with the prefix "J."
- **JDK: Java Development Kit.** It is considered a file since it contains a collection of tools and libraries for developing java applications. For windows, the Java Development Kit (JDK) extension directory is located at <Java_Home>\jre\lib\ext.
- **JRE: Java Runtime Environment.** It is considered a file inside Java Development Kit (JDK); the Java runtime environment (JRE) is the on-disk system that takes the code, combines it with necessary libraries, and executes it. The Java runtime environment (JRE) contains libraries and software that java programs need to run.
- **JVM: Java Virtual Machine.** It is a program whose purpose is to execute other programs.
- **JDT: Java Development Tools.** This package contains utilities for annotation processing in command-line builds and ant scripts.
- **Javac: Java programming language compiler.** It is a compiler for the programming language java.

❖ **Code Related:**

- **Sha 256:** It is a patented cryptographic hash function that outputs a value that is 256 bits long.
- **Lambda Function:** It is an anonymous function. It is an argument that being passed to higher-order functions or used to construct a higher-order function that needs to return the function. It can take any number of arguments but can only have one expression.
- **File Manifest:** In computing, it is a file containing metadata for a group of accompanying files that are part of a set of the coherent unit.
- **POJO: Plain Old Java Object.** It is a Java object that not requires any classpath. It is used for increasing the readability and re-usability of a program.
- **Flag:** It is a way to set options and pass in arguments to the commands that are run.
- **Map:** It is an object that maps keys to values, each key can map to at most one value, and it models a mathematical function abstraction.
- **Builder Pattern:** It is a design pattern that allows for the step-by-step creation of complex objects using the correct sequence of actions.
- **Proxy Pattern:** It provides an object that acts as a substitute for a real service object used by a client.
- **JFileChooser:** File choosers provide a GUI for navigating the file system, and then either choosing a file or directory from a list, or entering the name of a file or directory **[1]**.

CHAPTER 1: INTRODUCTION

1 CHAPTER 1: INTRODUCTION

In recent years, search engines on the web are spreading rapidly since it has a structure that grows and becomes more complex with the increase of users. However, the search engines that search for thousands of files in the file system are less developing because they have a fewer user base.

Search engines essentially act as filters for the wealth of information available on a location; they allow users to find the correct information easily and quickly. Web search engines are web-based tools that allow users to locate specific information on the World Wide Web. They get their information by crawling through the pages using their web crawlers. These web crawlers are commonly referred to as "spiders," the spider checks for the standard filename addressed to it. The file contains directives for search spiders, telling it which pages to crawl **[2]**, and then brings the result matched. At the same time, file systems typically have directories that allow the user to group files into different collections. This may be implemented by linking the file name with an index in a table of contents or an inode in a Unix-like file system. Instead of crawling through pages, search engines in file systems search through the directories and files, with the help of the search algorithms, which are procedures used to locate specific data among a collection of data by matching the search by key from the key-pair value that is stored before in the file system.

Nowadays, securing and transferring information has been a critical factor since increasing vulnerabilities to security threats and branches. Security testing using cryptography is a method of securing information through codes, the earliest known example of cryptography being used to protect sensitive information. In computer science, cryptography refers to the techniques that are derived from algorithms that are used for cryptographic key generation, digital signing, verification to protect data privacy, and for securing the communication and data in the presence of adversaries; it is about constructing and analyzing protocols that prevent third parties from reaching the private message, it is not only protecting data from being theft but can also be used for user authentication.

1.1 BACKGROUND

The file system is the process that manages operations on the data on a storage disk, such as storing, accessing, and managing. Sometimes the file system and operating system are so intercomplex that it is difficult to separate the file system functions; an interface should be provided by the operating system between user and file system, such as a textual command-line interface. Unix-like operating systems create a virtual file system that all the existing files are under one root directory in a single hierarchy. There are several aspects of a file system, one of these aspects is the directory which allows the user to group files into separate collections; this may be implemented by linking the filename with an index or an inode in a Unix-like file system and the other aspect is a filename which is a name used to identify uniquely a file stored in a file system, it may include hardware device, directory tree, file and type components, these components are required to identify a file across operating systems, Unix-like file systems allow a file to have more than one name, the names are hard links to the file's inode. Because file names have to be exchanged between the file system and software environments, it is important not to lose filename information between applications, filenames allow any character as long as the file system safe, within a single directory, filenames must be unique since filename syntax also applies for directories, it is impossible to create a file and directory with the same name in the directory. Every file system has a sort of database. Depending on the operating system type, the database can be partially or even completely loaded into random access memory (RAM) to help speed up the process of locating the position on the storage.

Cryptography referred to encryption, which is the process of converting ordinary information (called plaintext) into an unintelligible form (called ciphertext) **[3]**. Encryption is the process of encoding a message which contains information converted into secret code that hides the information's true meaning; this process converts the original representation of the information. The information contained in an encrypted message is called plaintext. In its encrypted, unreadable form, it is called a ciphertext. It is the way of scrambling data so that only authorized parties can understand the information. It has several benefits. It can be used to detect accidental or intentional alteration in data; also, it can be used to verify whether or not the author of the document is really. Encryption has limits. It can't prevent an attacker from deleting the data. An attacker might modify the program to use a key different from the one provided or might record all of the encryption keys in a special file for later retrieval. Also, an attacker could access the file before it is encrypted.

The Mini Search Engine inside the file system is making use of B-Tree and command patterns. The application will take a keyword input with one of the stated search commands from the user and parse it with the specific filtering mechanism that is chosen; after that, the lexing process will convert the sequence of characters into a sequence of tokens to check it in the file system with the help of the B-Tree, the tree algorithm will search and locate the entered input to be displayed as a result from GUI panel to the user with the help of in-memory database that is inside the system. There will be a Login screen connected to Operating System API with user authentication using encryption that fetches the data from in-memory Database for security standards.

1.2 PROJECT MOTIVATION

This project represents the structural contents and hardware features of Mini Search Engine; the purposed system is for collecting information of data's inside the file system and search through the file system using data structures, tree algorithms, and in-memory database; filtering those searches by command pattern and bring the absolute results to the user is the target. Also, the application must be secure by providing a login screen with cryptography and encryption algorithms. All these are needs that move the wheel of motivation to do this project that considered as a motivation of this project.

1.3 PROBLEM STATEMENT

In computer science, traditional ways of searching data and information involve many unnecessary issues and risks, it includes manual handling of searching and the file that is searched may be inside among hundreds of files and folders in the file system, this will occur error in searching and wrong results will be encountered, we are in an era where time is very important, and people want to use it efficiently, this mini search engine inside the file system project will help users needs in less time in comparison with various users from different places and devices. In this project, the target users and businesses are the ones who get the benefit from this application since it is helpful for safe, fast, and accurate search. However, this application encourages users to improve individual needs in terms of productivity by bringing quick results. So, I improved this application in a way that benefits users rather than stealing their time trying to search among thousands of data manually; thus, this application is ethical since it helps people's needs.

1.4 PROJECT QUESTION

The focus of this project is on developing a search engine and a desktop app. The project questions are:

- How to develop a search engine?
- How to make the application and its interface user-friendly?
- What different idea should be added to the search engine project?
- How to integrate a command pattern into the search engine project?
- How to improve the app's security?
- What are the user requirements towards the use of search engine functionality?
- What are the features used in the search engine that could help it to accomplish its tasks?

1.5 PROJECT OBJECTIVES

The main objective of this project is to provide fast, accurate, and correct results according to search filters inside the file system; other benefits are:

- Demonstrating and explaining the file systems and tree algorithms clearly.
- Explaining the importance and usage of cryptography and encryption algorithms.
- Explaining the unlimited usage of command patterns with regular expressions.
- Helping the people who are interested in such projects.
- Provide information and results about doing the project and encourage people to make projects similar to this one.
- Applying and merging knowledge and ideas to create something new.

1.6 PROJECT SIGNIFICANCE

The suggested search engine application allows some interested people and institutions to use the features of the search engine in useful ways and to let the students learn how to merge new ideas with algorithms and methods. Consequently, the significance of this study arises from several points as follows:

- Applying some of the latest software languages and technologies (Java, Eclipse IDE).
- Applying the most important methods and algorithms in programming by using object-oriented programming merged with data structures and encryption algorithm using an in-memory database.
- Breaking the ice of creating software that uses low-level programming language in some parts of it and merging algorithms.
- Integrating a unique command line to the application.
- Creating user-friendly UI.
- Using computers and Windows, Linux, or Unix operating systems to interact with software.
- Raising the realizing of programming concepts to the people who are interested in coding.

1.7 PROJECT SCOPE

The project scope can be explained for a large segment of people, but, for specific, the people who are wanting to search inside the directory and the file system in the fastest and most efficient way are the segments who are expected to benefit the most from this project. There will be several types of searching to be identified. The user should type one of the stated commands to select any of the search filters; he can search by name, first letter, first two letters, and such a rise; but when the user wants to select a search filter that is provided, he can select searching by regular expression commands with unlimited possibilities and also the user can search by date using epoch time. In other words, three different search filters with lots of options are made.

CHAPTER 2: LITERATURE REVIEW

2 CHAPTER 2: LITERATURE REVIEW

This chapter includes a literature review of search engines and Java programming language.

2.1 DEFINITION OF SEARCH ENGINE

A search engine is a software program designed to mine data available in databases and open directories; it helps users to find the information they are looking for using keywords or phrases. The search results are presented in lines of listed results. The information may be a mix of documents, files, images, videos, and other types of files.

2.2 LITERATURE REVIEW

BUTTCHER (2007), ‘’ Multi-User File System Search. ‘’ [4]. While desktop search is only concerned with the data owned by a single user, UNIX file system search needs to explicitly take into account the existence of multiple users. While desktop search usually focuses on certain parts of the data stored in the file system, such as a user’s documents, and the contents of the My Documents folder in Windows, file system search needs to cover a broader spectrum, providing efficient access also to man pages, header files, kernel sources, etc.. In essence, a file system search engine ought to be able to index all data in the file system that allow a meaningful textual representation. File system search should be an operating system service, similar to opening a file or sending a message across a network.

Requirements of File System Search The general search scenario outlined above leads to a set of five fundamental requirements that should be met by any file system search engine.

Requirement 1: Low Latency Query Processing Whenever a user submits a query to the search engine, the search results should be returned within at most a few seconds, preferably less, similar to what users of Web search engines are accustomed to.

Requirement 2: Query Flexibility The set of query operations provided by the search engine should be flexible enough for the search engine to be accessed by the user indirectly. In order for this to be possible, the query language provided by the search engine needs to be as flexible and expressive as possible without sacrificing query performance beyond reason.

Requirement 3: Index Freshness Whenever possible, the current version of the index, and thus the results to a search query, should reflect the contents of the file system. Changes in the file system – file deletions, creations, and modifications – need to be propagated to the search engine’s index structures in real time.

Requirement 4: Non-Interference The search engine should not interfere with the normal operation of the computer system. In addition, the search engine should only exhibit significant CPU or disk activity if this activity does not derogate the performance of other processes running in the system.

Requirement 5: Data Privacy If the computer system is accessed by multiple users, then the search engine must respect the security restrictions defined by the file system.

Obviously, some of these requirements work against each other. A more general query language makes it possible to specify search operations that are more expensive to carry out. Overly tight constraints on the freshness of the search engine's index will force the index to interfere with other processes running on the same machine.

COLE (2005), ‘’ Search engines tackle the desktop. ‘’ [5]. As PC hard drives get bigger and new information sources become available, users will have much more data of different types, including multimedia, on their computers. This makes it increasingly difficult to find documents, spreadsheets, and other files. Current desktop-based search capabilities, such as those in Windows, are inadequate to meet this challenge.

Major Web search providers and other companies are offering engines for searching PC hard drives. This requires new search approaches because desktop-based documents are generally structured differently than those on the Web.

Desktop search features built into current operating systems, and other applications have far fewer capabilities than Web search engines. They generally offer only simple keyword searches of a set of files, usually of a single file type.

On the Web, search engines can exploit information organized into a common HTML format with standardized ways of identifying various document elements. The engines can use this information, along with links to other documents, to make statistical guesses that increase the likelihood of returning relevant results. The desktop is more complicated to search because Microsoft Word and other applications format different types of documents in various ways. In addition, desktop files can be either structured or unstructured.

The function and meaning of structured files-such as information in a relational database or a text document with embedded tags-are clearly reflected in their structure. The easily identified structure makes searching such files easier. This is not the case with unstructured information, which includes natural-language documents, unformatted text files, speech, audio, images, and video.

Therefore, desktop search engines must add capabilities in different ways than Web search applications.

Desktop search engines employ one or more file crawler programs-similar to those used by Web search engines-that, upon installation, move through disk drives. The crawlers use an indexer to create an index of files; their location on a hard drive's hierarchical tree file structure; file names, types, and extensions (such as .doc or .jpg); and keywords. Once existing files are indexed, the crawler indexes new documents in real time. During searches, the engine matches queries to indexed items to find relevant files faster.

GORTER (2004), ‘’ Database file system – an alternative to hierarchy based file systems. ‘’ [6]. File and directory management is an essential and inevitable part of everyday computer usage. We are faced with the problem of locating files efficiently. Conventional file systems impose a hierarchical structure of storage on the user – a combination of its location and filename.

NIELSEN (1996), ‘’ The death of file systems. ‘’ [7]. Features like ‘symbolic links’ allow a file to be accessed through more than one path. However, a strict enforcing of the path which does not necessarily depict the meaning of the file itself still exists for all files. Users rarely name files properly, however the human mind is more tuned to remember the contents of the file rather than its name itself.

FOO; HENDRY, ‘’ Desktop Search Engine Visualization and Evaluation. ‘’ [8]. In contrast to the developments of online tools to search the Web, desktop search tools for searching increasing large volumes of documents held on local computers hard drives, have been slower to develop. While search tools are now being incorporated into the latest desktop operating systems, their search result visualizations have taken the traditional display of textual output. With result lists becoming increasingly longer, the challenge is to improve efficiency and effectiveness of search and result selection.

HUANG; WU, ‘’ The Research of Fast File Search Engine Based on NTFS and Its Application in Fast Electronic Document Destruction. ‘’ [9]. File search is a very common operation in computer application. Although Windows operating system provides the attached file search function and has a great improvement in the efficiency at new version, it still needs waiting for a long time, especially searching all disks. The fastest filename search tool developed can complete building hundreds of thousands of files’ index within a minute and locate files or folders by name instantly, the search speed is very egregious.

2.3 JAVA PROGRAMMING LANGUAGE

2.3.1 WHAT İS JAVA?

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers write once, run anywhere (WORA) **[10]**, meaning that compiled Java code can run on all platforms that support Java without the need for recompilation **[11]**. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++ but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages.

Over the years, Java has been the brain of thousands of projects, from daily to complex scientific software. The worldwide community of makers, students, programmers, and professionals has gathered around this open-source platform. Their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

2.3.2 WHY JAVA?

Thanks to its simple and accessible user experience, Java has been used in millions of different projects and applications. The Eclipse IDE software is easy to iterate beginners yet flexible enough for advanced users. It runs on Windows, Linux, and Unix.

Programmers, teachers, and students use it to build scientific software by using programming and algorithms. Designers and architects build interactive prototypes. Java is a key tool to learn new things. Programmers can start coding just following the step-by-step instructions or sharing ideas online with other members of the Java and Oracle community.

There are some advantages of Java programming language and Eclipse IDE for programmers, teachers, students, and interested amateurs over other systems; these advantages are:

- It is simple to use, write, compile, debug, and learn than alternative programming languages.
- It is object-oriented, so it permits the user to form standard programs and reusable code.
- It is platform-independent and has a cross-platform; Java code runs on any machine that is JVM presented on it. The Eclipse IDE runs on Windows, Linux, and Unix operating systems.
- It has distributed computing since it helps developing applications on networks that contribute to both data and application functionality.
- It has a security manager that defines the access of classes.

- It has the potential to perform many tasks at the same time.

2.4 RELATE WORK

There are several similar projects locally and internationally, and some of them use some similar components to this particular project. The table below shows the differences by summarizing the main features of this mini search engine from other projects.

Main Features
Improved security system using encryption and hashing algorithms
Structure that complies with existing programming standards
Using programming algorithms together
Integrated extensible and unique command-system
Easy and understandable usage of application
User-friendly UI that keeps the application looking smooth
Being intimate with regular expressions
Using and understanding epoch timer
Providing faster search results with B-tree and in-memory database

CHAPTER 3: METHODOLOGY

3 CHAPTER 3: METHODOLOGY

This chapter describes the adopted methodology throughout the project and will describe the methodology phases used to accomplish the project and the phases of these methodologies consist. The reasons for choosing these methodologies will be listed; after that, each phase of each methodology will be described. Finally, it represents the requirement engineering methodologies followed in the project.

3.1 ADOPTED METHODOLOGY

The adopted methodologies during the development of this project will be shown below. The listed methodologies are followed to accomplish this project since they were the most suitable methodologies for small and medium systems or subsystems like mini search engine because the requirements are well-understood.

3.1.1 WATERFALL METHODOLOGY

There are several sources or frameworks for classifying non-functional requirements. The first waterfall methodology is developed to help with the software development process. Waterfall now is the most widely used methodology, and it gets its name from the analogy of water falling downward. The most commonly used version available includes a corrective feedback mechanism.

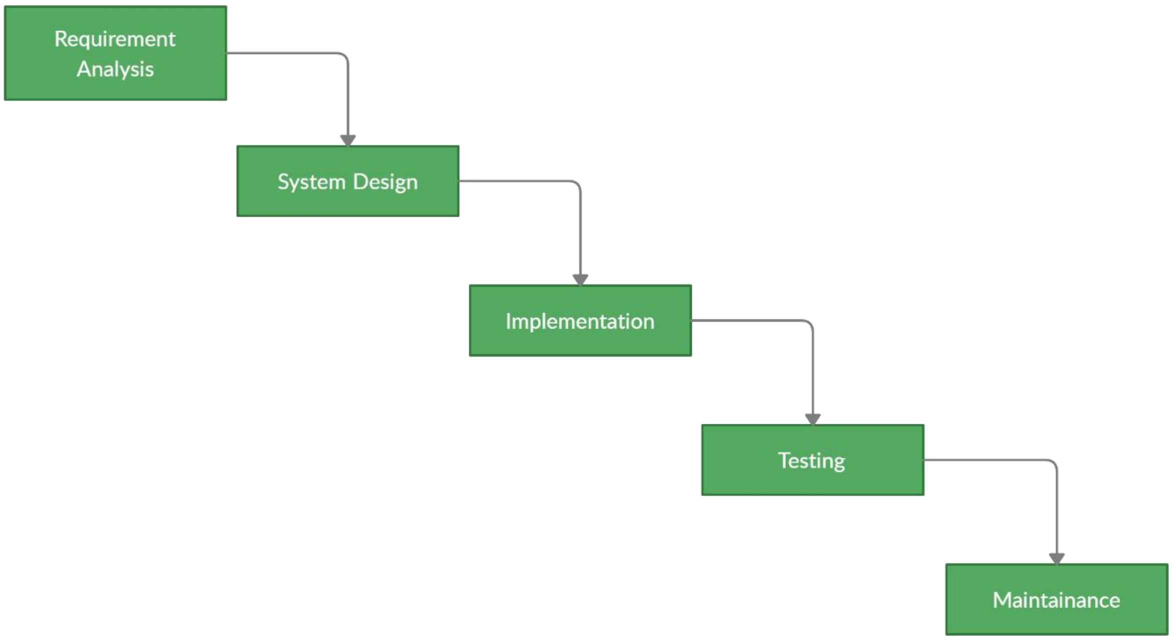


Figure 1. Waterfall Methodology

A **waterfall** model transfers the project activities into sequential phases, where each phase depends on the deliverables of the previous one.

Here, in **Figure 1**, we can see that this project has begun with Requirement Analysis; after analyzing part, the System Design part is done, and the implementation, testing, and maintenance parts will be done till the end of this project.

3.1.2 WHY ADOPTING WATERFALL IS SUITABLE FOR MINI SEARCH ENGINE?

This methodology has been chosen because:

- The requirements are clear, simple, and well understood for the project.
- Can be developed step by step.
- Understand the requirements of the system significantly.
- It is suitable for small and medium systems.

3.1.3 IMPLEMENTATION OF WATERFALL

The five steps of the waterfall will be implemented to accomplish this project; each step has its functions.

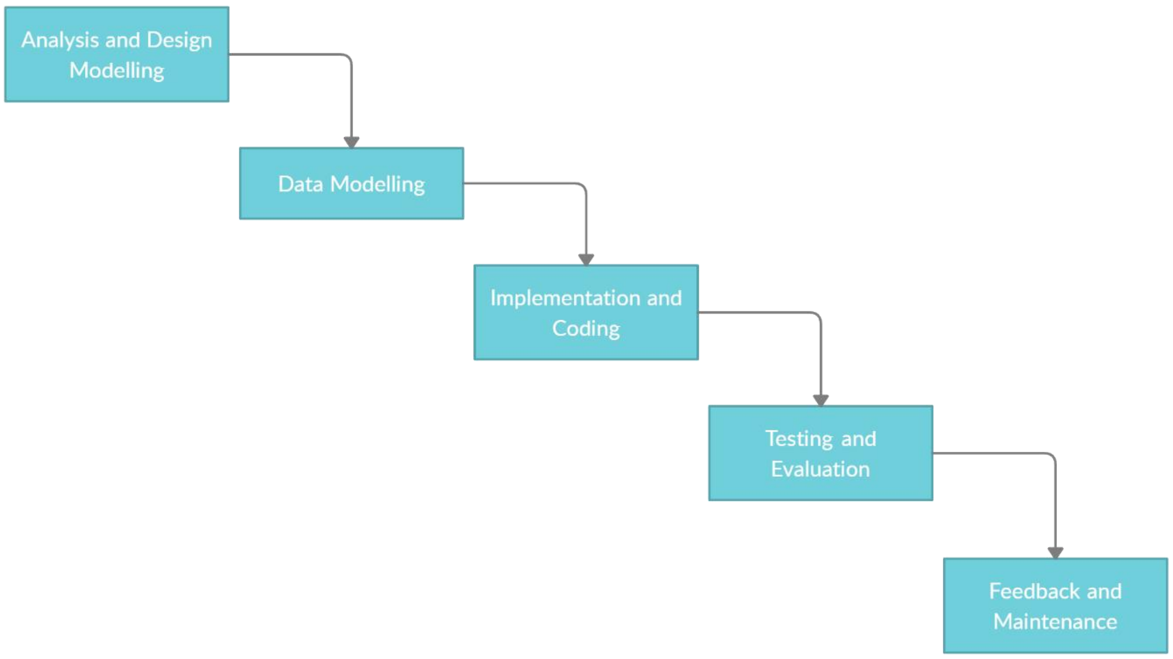


Figure 2. Waterfall General Steps

In **Figure 2**, This model allows changes in any stage which was very costly, it has multiple cycles, and each cycle brings new features. This project started with Analysis and Design Modelling; after that, the Data Modelling, Implementation and coding, testing and evaluation, and Feedback and Maintenance are handled till the end of the project.

The detailed steps of figure 2 will be shown below; each step handles different topics.

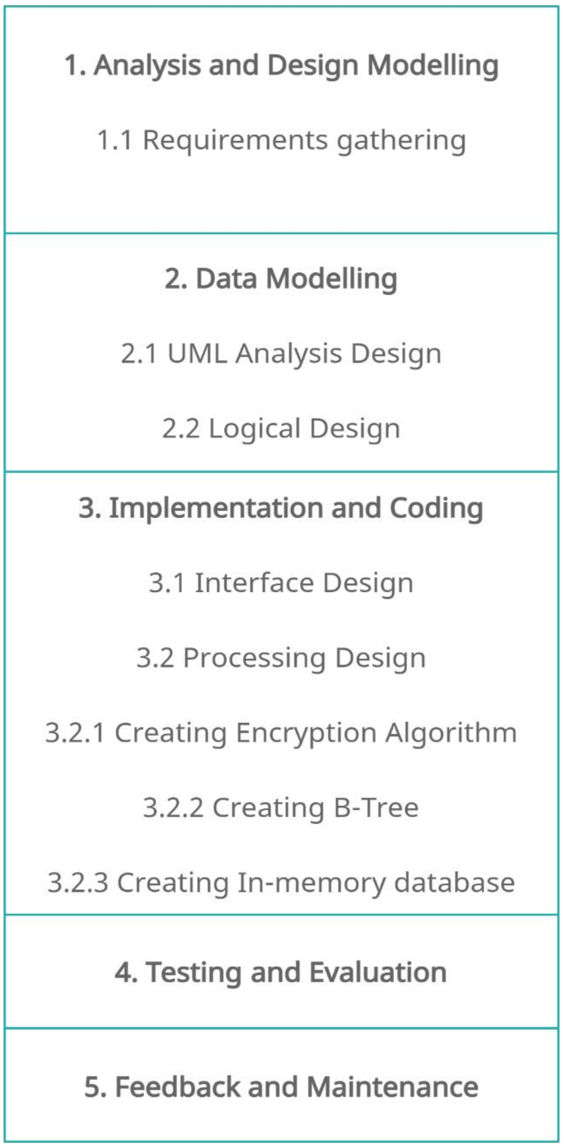

1. Analysis and Design Modelling
1.1 Requirements gathering
2. Data Modelling
2.1 UML Analysis Design
2.2 Logical Design
3. Implementation and Coding
3.1 Interface Design
3.2 Processing Design
3.2.1 Creating Encryption Algorithm
3.2.2 Creating B-Tree
3.2.3 Creating In-memory database
4. Testing and Evaluation
5. Feedback and Maintenance

Figure 3. Waterfall Detailed Steps

In **Figure 3**, the Analysis and Design Modelling contains requirements gathering. The Data Modelling contains UML design analysis and logical design. Implementation and coding contain Interface design, processing design which handles creating encryption algorithm, creating B-tree, and creating an in-memory database. Testing and evaluation, and Feedback, and Maintenance are handled till the end of the project.

CHAPTER 4: ANALYSIS, DESIGN AND STANDARDS

4 CHAPTER 4: ANALYSIS, DESIGN AND STANDARDS

This chapter will provide a complete description of the system and its users. Then it depicts the functional and non-functional requirements that have been collected using several methods from brainstorming and interview. After determining the essential requirements, requirement analysis was adopted using several tools and UML diagrams.

4.1 USER DESCRIPTION

There is one main user iterate proposed system. Users can perform several different functions during the usage of the system. These functions were determined according to the design of the proposed system to iterate the system more effectively and efficiently.

User can use the search engine and search any file inside the file system; he can filter the searches by using stated commands which are unique and extensible.

4.2 REQUIREMENTS DEVELOPMENT

Brainstorming sessions were held to develop the proposed system’s functional and non-functional requirements. Through these sessions, the system was analyzed, and some of the requirements were generated according to the previous models.

4.3 SYSTEM REQUIREMENTS

Before creating software, it is necessary to visualize the layout, design and features intended to be included. Requirements are required attributes in the system, which define a capability, characteristic or quality factor to providing value and benefit to users. Once the requirements are determined, developers can initiate other technical work, including system design, development, testing, implementation, and operation.

There are functional and non-functional requirements for any system to be considered while determining the system's requirements.

4.3.1 FUNCTIONAL REQUIREMENTS

Functional requirements have been developed by reviewing the literature review and revising similar systems, brainstorming sessions.

The user of the system has particular functional requirements that enable a user to use the system. These requirements explain what the mini search engine should do.

The primary user requirements to use the application are:

- having a Computer.
- The computer must include an operating system (Windows, Linux, Unix).
- The User must set up the software of an integrated development environment (Eclipse IDE) to start using the application.

4.3.2 NON-FUNCTIONAL REQUIREMENTS

Several things can be considered while developing a software application. The system must be practical and straightforward to achieve quality metrics and be understandable and easy to learn and use.

A questionnaire was designed and developed to assess the importance of sessions and interviews to obtain the most basic requirements of the system. Then, that the survey results are considered while creating the mini search engine application.

4.4 SYSTEM CONTEXT

In this part, the system interaction with its environment will be shown with a system block diagram that shows parts and functions are represented by blocks connected by lines showing relationships of the blocks.

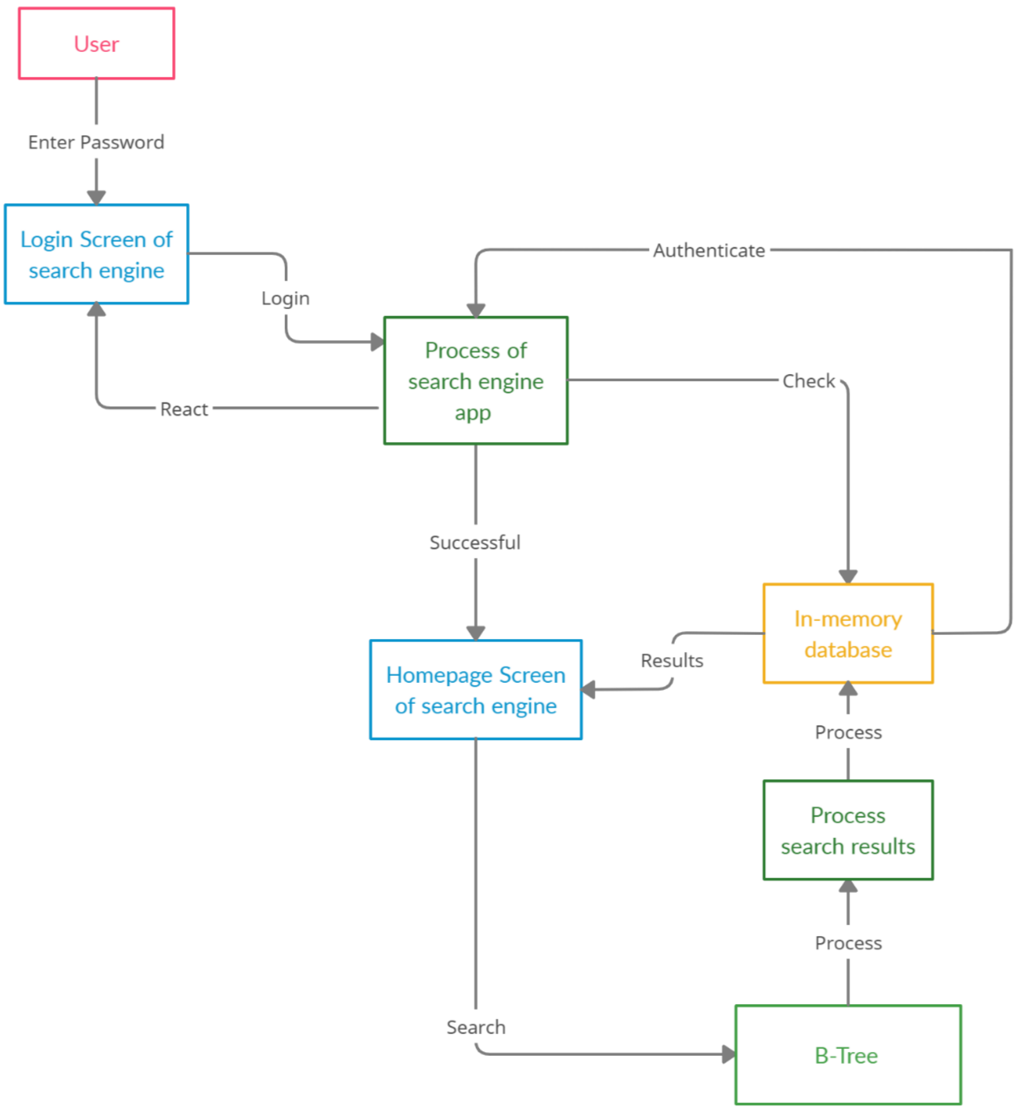


Figure 4. Block Diagram

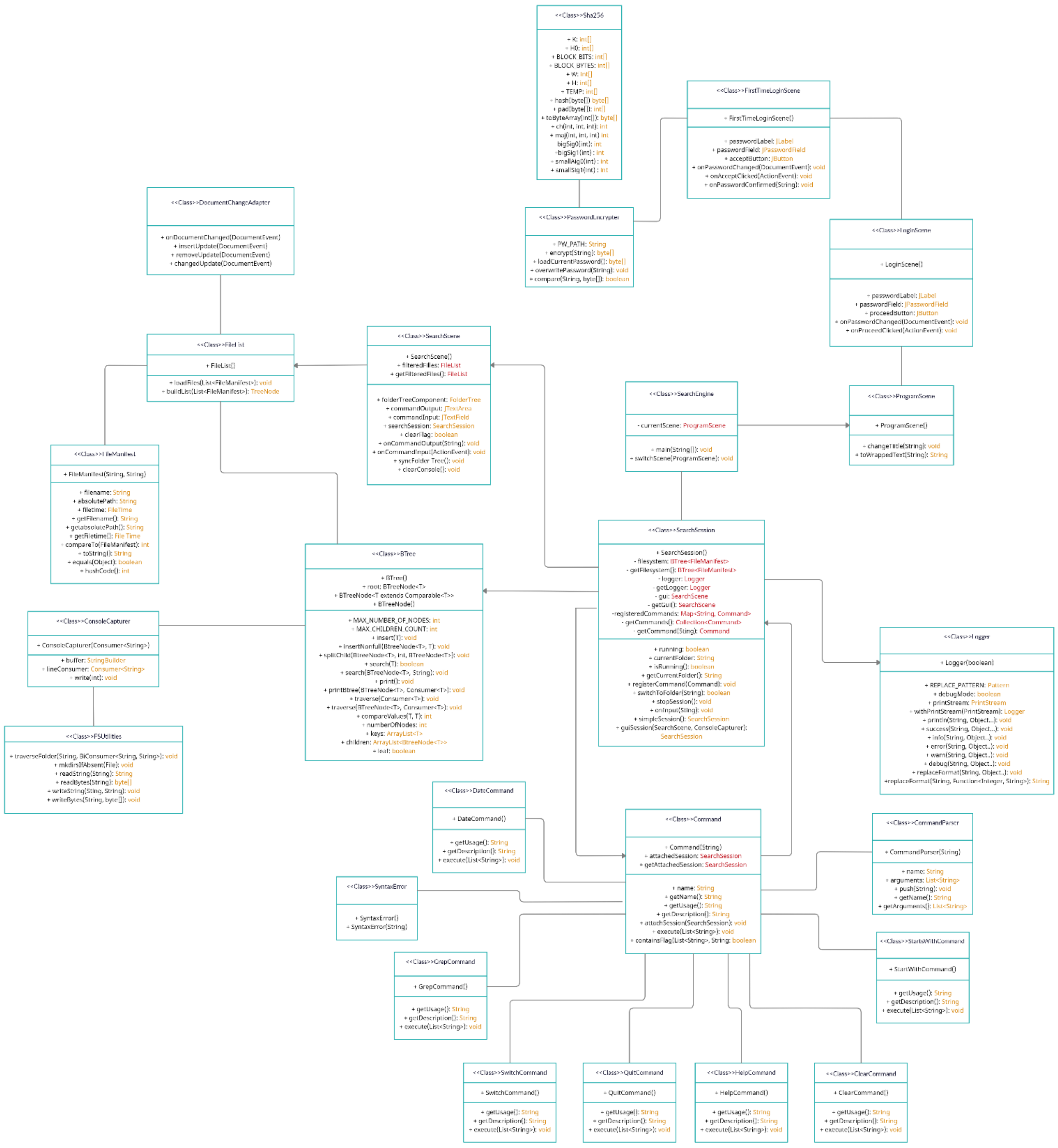
Here in **Figure 4**, the user enters the data on the login page. The system checks whether it is valid input data or not; after that, the system reacts. After login successfully to the search engine's homepage, when the user searches an input keyword, it will be searched through the file system by the B-tree algorithm, and the process of search results will be transferred to the in-memory database. Search results will be processed to bring accurate results to the user.

4.5 SYSTEM DESIGN

4.5.1 DESIGN METHODS AND STANDARDS

In this part, the **UML** (Unified Modeling Language) will be handled, and the detailed information of every diagram will be explained.

4.5.1.1 CLASS DIAGRAM



<<Class>>Sha256
+ K: int[]
+ H0: int[]
+ BLOCK_BITS: int[]
+ BLOCK_BYTES: int[]
+ W: int[]
+ H: int[]
+ TEMP: int[]
+ hash(byte[]) byte[]
+ pad(byte[]): int[]
+ toByteArray(int[]): byte[]
+ ch(int, int, int): int
+ maj(int, int, int) int
bigSig0(int): int
+bigSig1(int) : int
+ smallAlg0(int) : int
+ smallSig1(int) : int
<<Class>>FirstTimeLoginScene
+ FirstTimeLoginScene()
+ passwordLabel: JLabel
+ passwordField: JPasswordField
+ acceptButton: JButton
+ onPasswordChanged(DocumentEvent): void
+ onAcceptClicked(ActionEvent): void
+ onPasswordConfirmed(String): void
<<Class>>PasswordEncrypter
+ PW_PATH: String
+ encrypt(String): byte[]
+ loadCurrentPassword(): byte[]
+ overwritePassword(String): void
+ compare(String, byte[]): boolean
<<Class>>LoginScene
+ LoginScene()
+ passwordLabel: JLabel
+ passwordField: JPasswordField
+ proceedButton: JButton
+ onPasswordChanged(DocumentEvent): void
+ onProceedClicked(ActionEvent): void
<<Class>>DocumentChangeAdapter
+ onDocumentChanged(DocumentEvent)
+ insertUpdate(DocumentEvent)
+ removeUpdate(DocumentEvent)
+ changedUpdate(DocumentEvent)
<<Class>>FileList
+ FileList()
+ loadFiles(List<FileManifest>): void
+ buildList(List<FileManifest>): TreeNode
<<Class>>SearchScene
+ SearchScene()
+ filteredFiles: FileList
+ getFilteredFiles(): FileList
+ folderTreeComponent: FolderTree
+ commandOutput: JTextArea
+ commandInput: JTextField
+ searchSession: SearchSession
+ clearFlag: boolean
+ onCommandOutput(String): void
+ onCommandInput(ActionEvent): void
+ syncFolder Tree(): void
+ clearConsole(): void
<<Class>>SearchEngine
- currentScene: ProgramScene
+ main(String[]): void
+ switchScene(ProgramScene): void
<<Class>>ProgramScene
+ ProgramScene()
+ changeTitle(String): void
+ toWrappedText(String): String
<<Class>>FileManifest
+ FileManifest(String, String)
+ filename: String
+ absolutePath: String
+ filetime: FileTime
+ getFilename(): String
+ getabsolutePath(): String
+ getFiletime(): File Time
+ compareTo(FileManifest): int
+ toString(): String
+ equals(Object): boolean
+ hashCode(): int
<<Class>>SearchSession
+ SearchSession()
- filesystem: BTree<FileManifest>
- getFilesystem(): BTree<FileManifest>
- logger: Logger
- getLogger: Logger
- gui: SearchScene
- getGui(): SearchScene
-registeredCommands: Map<String, Command>
- getCommands(): Collection<Command>
- getCommand(Sting): Command
+ running: boolean
+ currentFolder: String
+ isRunning(): boolean
+ getCurrentFolder(): String
+ registerCommand(Command): void
+ switchToFolder(String): boolean
+ stopSession(): void
+ onInput(Sting): void
+ simpleSession(): SearchSession
+ guiSession(SearchScene, ConsoleCapturer): SearchSession
<<Class>>BTree
+ BTree()
+ root: BTreeNode<T>
+ BTreeNode<T extends Comparable<T>>
+ BTreeNode()
+ MAX_NUMBER_OF_NODES: int
+ MAX_CHILDREN_COUNT: int
+ insert(T): void
+ insertNonfull(BtreeNode<T>, T): void
+ splitChild(BtreeNode<T>, int, BTreeNode<T>): void
+ search(T): boolean
+ search(BTreeNode<T>, String): void
+ print(): void
+ printBtree(BTreeNode<T>, Consumer<T>): void
+ traverse(Consumer<T>): void
+ traverse(BTreeNode<T>, Consumer<T>): void
+ compareValues(T, T): int
+ numberOfNodes: int
+ keys: ArrayList<T>
+ children: ArrayList<BtreeNode<T>>
+ leaf: boolean
<<Class>>ConsoleCapturer
+ ConsoleCapturer(Consumer<String>)
+ buffer: StringBuilder
+ lineConsumer: Consumer<String>
+ write(int): void
<<Class>>Logger
+ Logger(boolean)
+ REPLACE_PATTERN: Pattern
+ debugMode: boolean
+ printStream: PrintStream
+ withPrintStream(PrintStream): Logger
+ println(String, Object...): void
+ success(String, Object...): void
+ info(String, Object..): void
+ error(String, Object..): void
+ warn(String, Object..): void
+ debug(String, Object..): void
+ replaceFormat(String, Object..): void
+replaceFormat(String, Function<Integer, String>): String
<<Class>>FSUtilities
+ traverseFolder(String, BiConsumer<String, String>): void
+ mkdirsIfAbsent(File): void
+ readString(String): String
+ readBytes(String): byte[]
+ writeString(Sting, String): void
+ writeBytes(String, byte[]): void
<<Class>>DateCommand
+ DateCommand()
+ getUsage(): String
+ getDescription(): String
+ execute(List<String>): void
<<Class>>Command
+ Command(String)
+ attachedSession: SearchSession
+ getAttachedSession: SearchSession
+ name: String
+ getName(): String
+ getUsage(): String
+ getDescription(): String
+ attachSession(SearchSession): void
+ execute(List<String>): void
+ containsFlag(List<String>, String: boolean
<<Class>>CommandParser
+ CommandParser(String)
+ name: String
+ arguments: List<String>
+ push(String): void
+ getName(): String
+ getArguments(): List<String>
<<Class>>SyntaxError
+ SyntaxError()
+ SyntaxError(String)
<<Class>>StartsWithCommand
+ StartWithCommand()
+ getUsage(): String
+ getDescription(): String
+ execute(List<String>): void
<<Class>>GrepCommand
+ GrepCommand()
+ getUsage(): String
+ getDescription(): String
+ execute(List<String>): void
<<Class>>SwitchCommand
+ SwitchCommand()
+ getUsage(): String
+ getDescription(): String
+ execute(List<String>): void
<<Class>>QuitCommand
+ QuitCommand()
+ getUsage(): String
+ getDescription(): String
+ execute(List<String>): void
<<Class>>HelpCommand
+ HelpCommand()
+ getUsage(): String
+ getDescription(): String
+ execute(List<String>): void
<<Class>>ClearCommand
+ ClearCommand()
+ getUsage(): String
+ getDescription(): String
+ execute(List<String>): void

Figure 5. Class Diagram

From **Figure 5**, the classes with their methods and attributes will be listed.
Note: The classes will be listed according to the stage of construction.

- **Sha256:** It includes; “K, H0, BLOCK_BITS, BLOCK_BYTES, W, H TEMP, hash(byte[]), pad(byte[]), toByteArray(int[], ch(int, int, int), maj(int,int,int), bigSig0(int), bigSig1(int), smallAig0(int), smallSig1(int)”. This class is associated with **PasswordEncrypter** class.

- **PasswordEncrypter:** It includes; “PW_PATH, encrypt(String), loadCurrentPassword(), overwritePassword(String), compare(String, byte[])”. This class is associated with **FirstTimeLoginScene** class.

- **FSUtilities:** It includes; “traverseFolder(String, BiConsumer<String, String>), mkdirsIfAbsent(File), readString(String), readBytes(String), writeString(String, String), writeBytes(String, byte[])”. This class is associated with **ConsoleCapturer** class.

- **ProgramScene:** It includes; “ProgramScene(), changeTitle(String), toWrappedText(String)”. This class is associated with **LoginScene** class. Also, **SearchEngine** class has a directed Association with this class.

- **FirstTimeLoginScene:** It includes; “FirstTimeLoginScene(), passwordLabel, passwordField, acceptButton, onPasswordChanged(DocumentEvent), onAcceptClicked(ActionEvent), onPasswordConfirmed”. This class is associated with **LoginScene** class.

- **SearchEngine:** It includes; “currentScene from the **ProgramScene** class, main(String[]), switchScene(ProgramScene). This class is directly associated with **ProgramScene** class.

- **LoginScene:** It includes; “LoginScene(), passwordLabel, passwordField, proceedButton, onPasswordChanged(DocumentEvent), onProceedClicked(ActionEvent)”. This class is associated with both **FirstTimeLoginScene** and **ProgramScene** classes.

- **BTree:** It includes; “BTree(), root, BTreeNode<T extends Comparable<T>>, BTreeNode(), MAX_NUMBER_OF_NODES, MAX_CHILDREN_COUNT, insert(T), insertNonfull(BTreeNode<T>,T), splitChild(BtreeNode<T>, int, BTreeNode<T>), search(T), search(BTreeNode<T>, String), print(), printBtree(BTreeNode<T>, Consumer<T>), traverse(Consumer<T>), traverse(BTreeNode<T>, Consumer<T>), compareValues(T, T), numverOfNodes, keys, children, leaf”. This class is associated with both **FileList** and **ConsoleCapturer** classes. Also, **SearchSession** class has a directed Association with this class.

- **FileManifest:** It includes; “ FileManifest(String, String), filename, absolutePath, filetime, getFilename(), getabsolutePath(), getFiletime(), compateTo(FileManifest), toString, equals(Object), hashCode()”. This class is associated with **FileList** class.

- **Command:** It includes, “Command(String), attachedSession and getAttachedSession from the **SearchSession** class, name, getName(), getUsage(), getDescription(), attachSession(SearchSession), execute(List<String>), containsFlag(List<String>,

String)”. This class is directly associated with **SearchSession** class; the **SearchSearch** class has a directed association with this class. Also, this class is associated with **SwitchCommand**, **QuitCommand**, **CommandParser**, **GrepCommand**, **DateCommand**, **HelpCommand**, **ClearCommand**, and **SyntaxError** classes.

- **SwitchCommand:** It includes; “SwitchCommand(), getUsage(), getDescription, execute(List<String>)”. This class is associated with **Command** class.

- **QuitCommand:** It includes; “QuitCommand(), getUsage(), getDescription, execute(List<String>)”. This class is associated with **Command** class.

- **CommandParser:** It includes; “CommandParser(String), name, arguments, push(String), getName(), getArguments()”. This class is associated with **Command** class.

- **SearchSession:** It includes; “SearchSession(), fileSystem and getFilesystem() from the **BTree<FileManifest>** class, logger and getLogger from **Logger** class, gui and getGui() from **SearchScene** class, registeredCommands, getCommands(), and getCommand(String) from **Command** class, running, currentFolder, isRunning(), gerCurrentFolder(), registerCommand(Command), switchToFolder(String), stopSession(), onInput(String), simpleSession(), guiSession(SearchScene, ConsoleCapturer)”. This class is directly associated with **ProgramScene**, **BTree**, **Command**, **SearchScene** and **Logger** classes; the Command class has a directed association with this class. Also, this class is associated with **SearchEngine** class.

- **ConsoleCapturer:** It includes; “ConsuleCapturer(Consumer<String>), buffer, lineConsumer, write(int)”. This class is associated with both **Btree** and **FSUtilities** Classes.

- **SearchScene:** It includes; “SearchScene(), filteredFiles and getFilteredFiles() from **FileList** class, folderTreeComponent, commandOutput, commandInput, searchSession, clearFlag, onCommandOutput(String), onCommandInput(ActionEvent), syncFolder Tree(), clearConsole()”. This class is directly associated with **FileList** class. Also, the **SearchSession** class has a directed association with this class.

- **Logger:** It includes; “Logger(boolean), REPLACE_PATTERN, debugMode, printStream, withPrintStream(PrintStream), println(String, Object…), success(String, Object…), info(String, Object…), error(String, Object…), warn(String, Object…), debug(String, Object…), replaceFormat(String, Object…), replaceFormat(String, Function<Integer, String>)”. The **SearchSession** class is directly associated with this class.

- **FileList:** It includes; “FileList(), loadFiles(List<FileManifest>), buildList(List<FileManifest>)”. The **SearchScene** class is directly associated with this class. This class is associated with **BTree**, **FileManifest** and **DocumentChangeAdapter** classes.

- **GrepCommand:** It includes; “GrepCommand(), getUsage(), getDescription, execute(List<String>)”. This class is associated with **Command** class.

- **StartsWithCommand:** It includes; “StartsWithCommand(), getUsage(), getDescription, execute(List<String>)”. This class is associated with **Command** class.
- **DateCommand:** It includes; “DateCommand(), getUsage(), getDescription, execute(List<String>)”. This class is associated with **Command** class.
- **HelpCommand:** It includes; “HelpCommand(), getUsage(), getDescription, execute(List<String>)”. This class is associated with **Command** class.
- **ClearCommand:** It includes; “ClearCommand(), getUsage(), getDescription, execute(List<String>)”. This class is associated with **Command** class.
- **SyntaxError:** It includes; “SyntaxError(), SyntaxError(String)”. This class is associated with **Command** class.
- **DocumentChangeAdapter:** It includes; “onDocumentChanged(DocumentEvent), InsertUpdate(DocumentEvent), removeUpdate(DocumentEvent), changedUpdate(DocumentEvent)”. This class is associated with **FileList** class.

4.5.1.2 OBJECT DIAGRAM

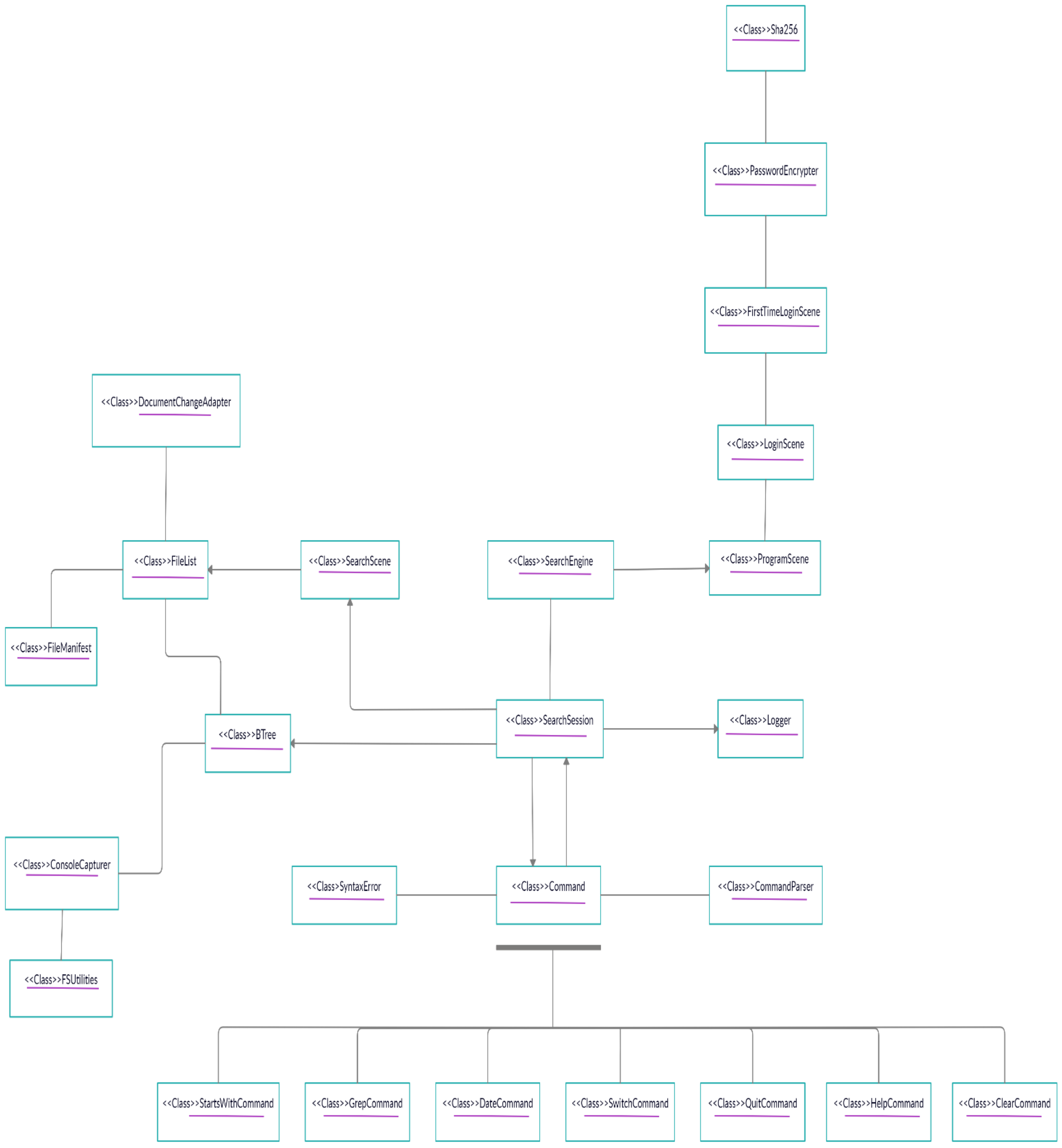


Figure 6. Object Diagram

As we can see from **Figure 6** above, we can notice that the object diagram uses a notation that is similar to that used in the class diagram; it provides a minimal architecture of the instances in the system with the relationships.

4.5.1.3 SYSTEM AND ACTIVITY DIAGRAM

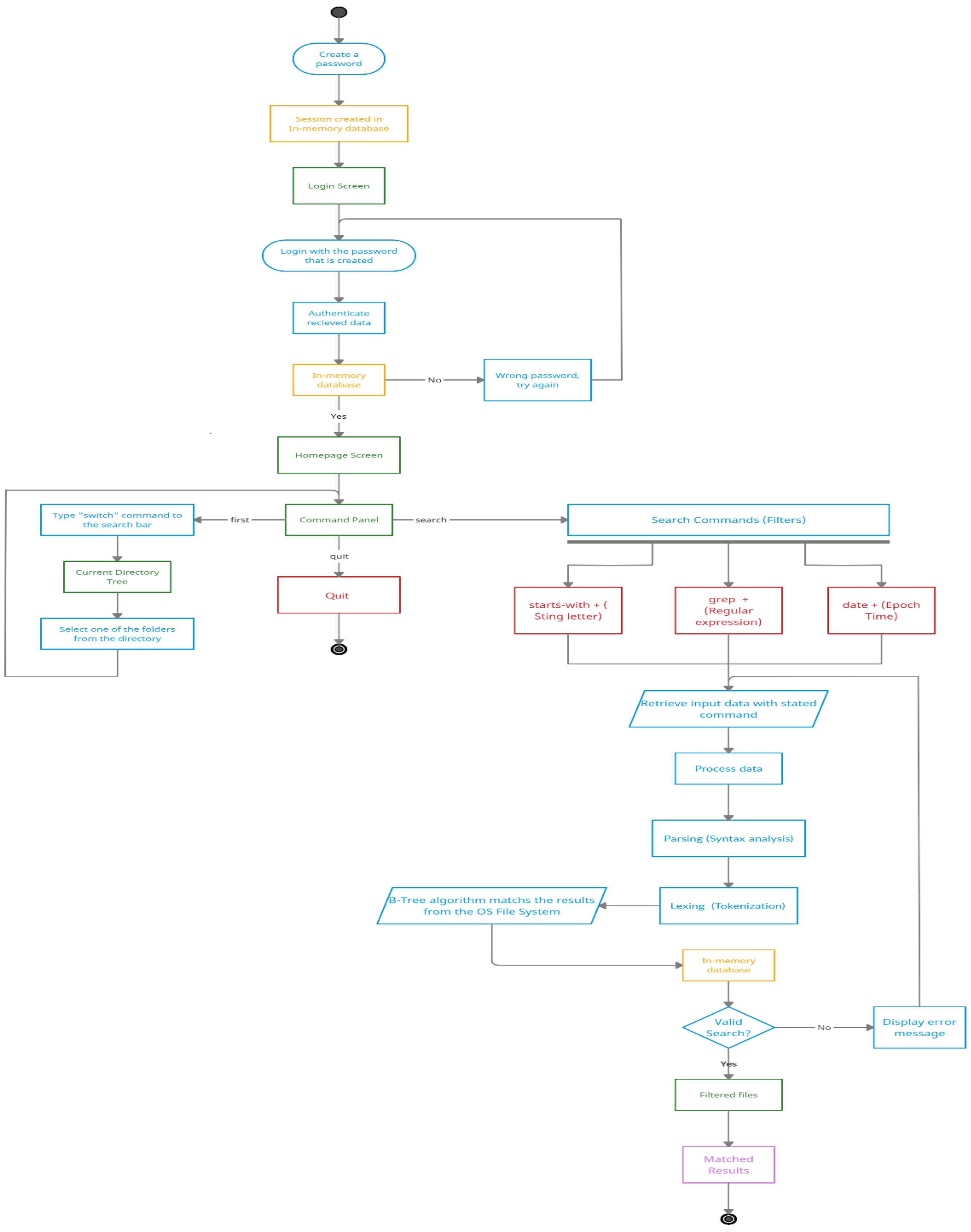

Create a password
Session created in In-memory database
Login Screen
Login with the password that is created
Authenticate recieved data
In-memory database
No
Wrong password, try again
Yes
Homepage Screen
Command Panel
first
Type "switch" command to the search bar
Current Directory Tree
Select one of the folders from the directory
quit
Quit
search
Search Commands (Filters)
starts-with + (Sting letter)
grep + (Regular expression)
date + (Epoch Time)
Retrieve input data with stated command
Process data
Parsing (Syntax analysis)
Lexing (Tokenization)
B-Tree algorithm matchs the results from the OS File System
In-memory database
Valid Search?
No
Display error message
Yes
Filtered files
Matched Results

Figure 7. System And Activity Diagram

In **Figure 7** above. At first, the user must create a password to access the Login Screen; when the user enters the password, a session will be created in the in-memory database; after that, the user will be able to access the Login Screen. When the user enters the password that he created before, this password will be authenticated in the in-memory database; if the password is false, the process will start again with an error message. After entering the correct password, the user will be able in the Homepage Screen of the application. In the Homepage Screen, there are three different panels; at first, the user must type the "switch" command to access the folders and list the chosen folder in the Current Directory Tree panel; after that, the user can select the stated search commands from the Command Panel, every search command has nearly the same general process. At first, when the user enters the keyword input, the program will receive the data and process it; the parsing process, also called (Syntax analysis) will analyze a string of symbols that received. Then the lexing process, also called (Tokenization) will convert a sequence of characters into a sequence of tokens; after that, the search tree algorithm will be applied to match the key from the key-value pair to find the result, the data, and information will be fetched from the in-memory database. If there is a valid search result, the result will be displayed to the user; otherways it will display an error message.

4.5.1.4 COLLOBORATION DIAGRAM

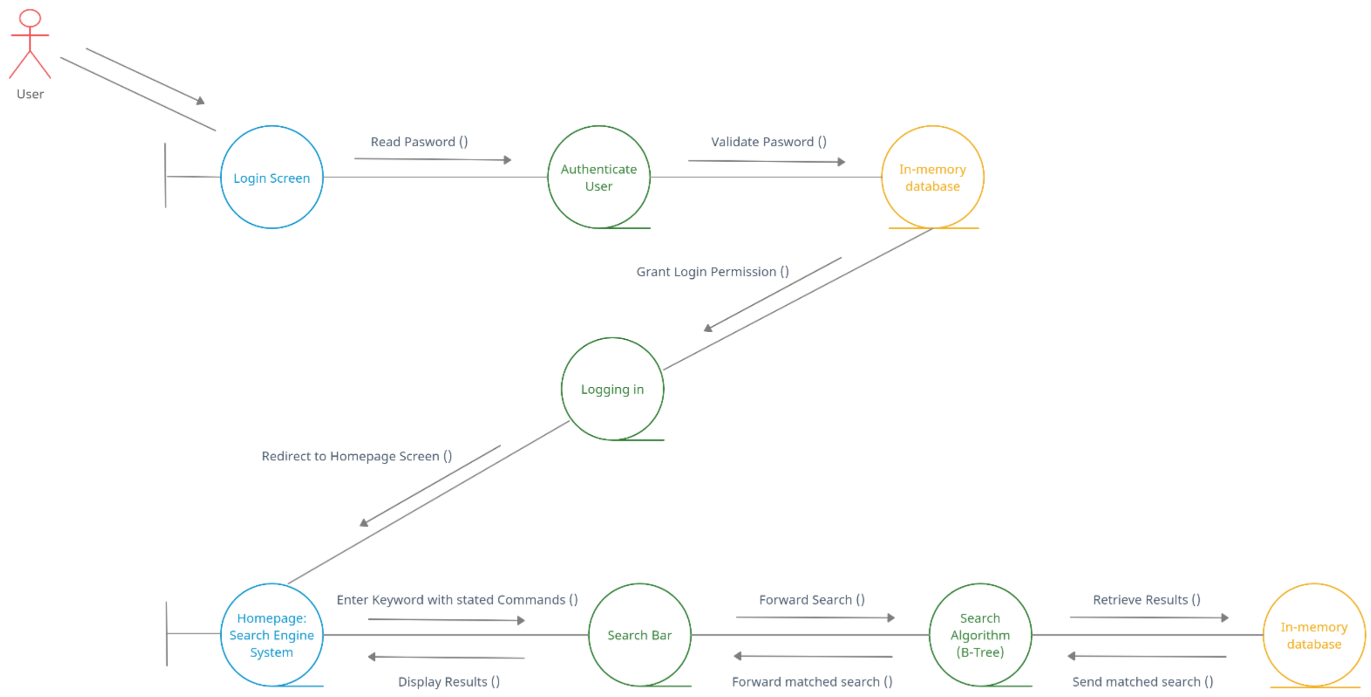


Figure 8. Colloboration Diagram

In **Figure 8** above, the user will see the Login Screen at first, and the information that he entered will be authenticated and will be validated from the in-memory database; the in-memory database will grant login permission, and the user will be redirected to the homepage of the search engine, when the user enters a keyword to the search bar, the search algorithm will search and pass the matched results through the in-memory database. The results will be displayed to the user.

4.5.1.5 USE-CASE DIAGRAM

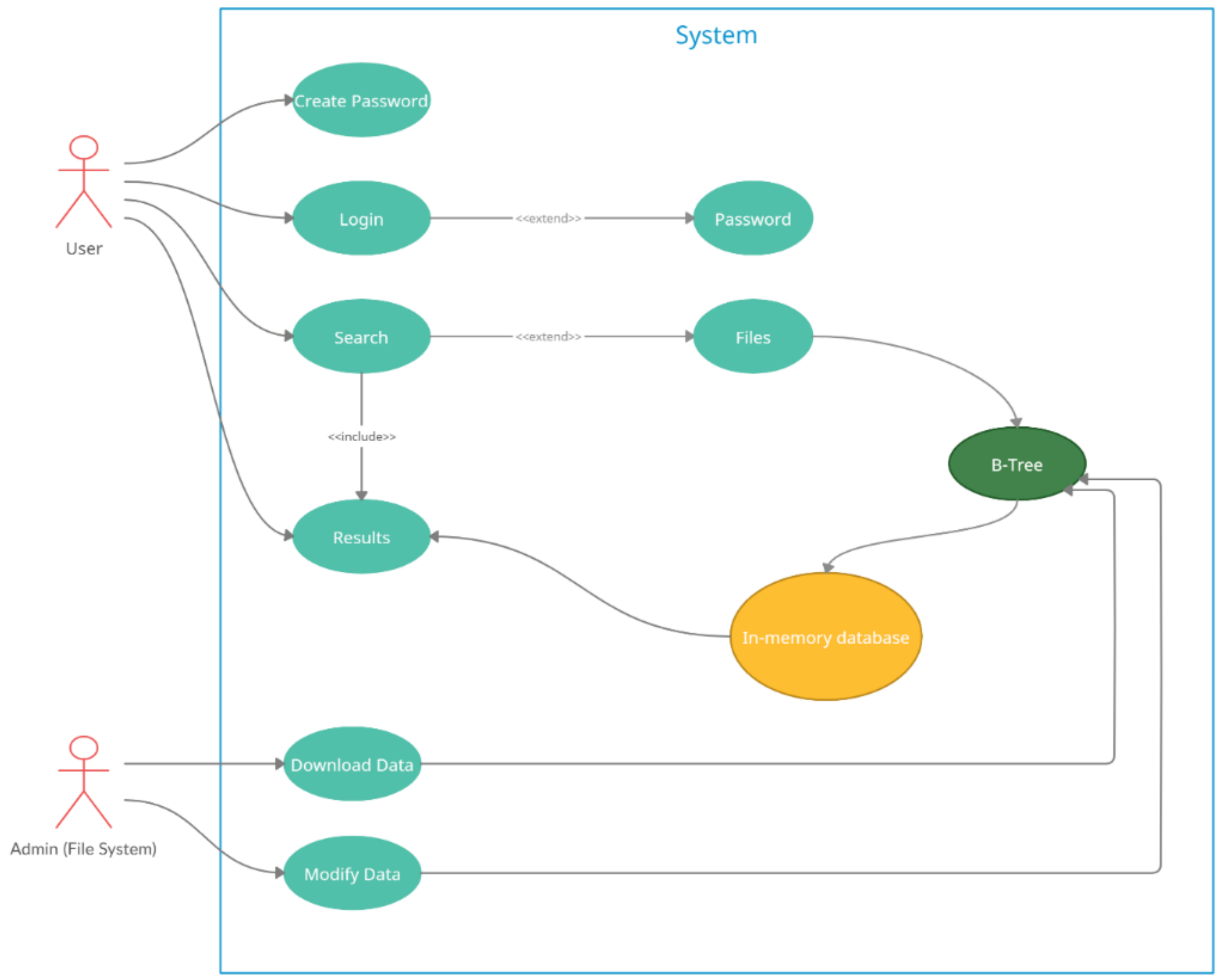


Figure 9. Use-Case Diagram

In **Figure 9**, The user must create a password and can log in by password; searching inside the file system will be through the search algorithm (B-Tree), and this algorithm will pass the matched results to the in-memory database, and the user will see results of searching which are coming from the in-memory database, while the admin can only download and modify data in the system.

4.5.1.6 COMPONENT DIAGRAM

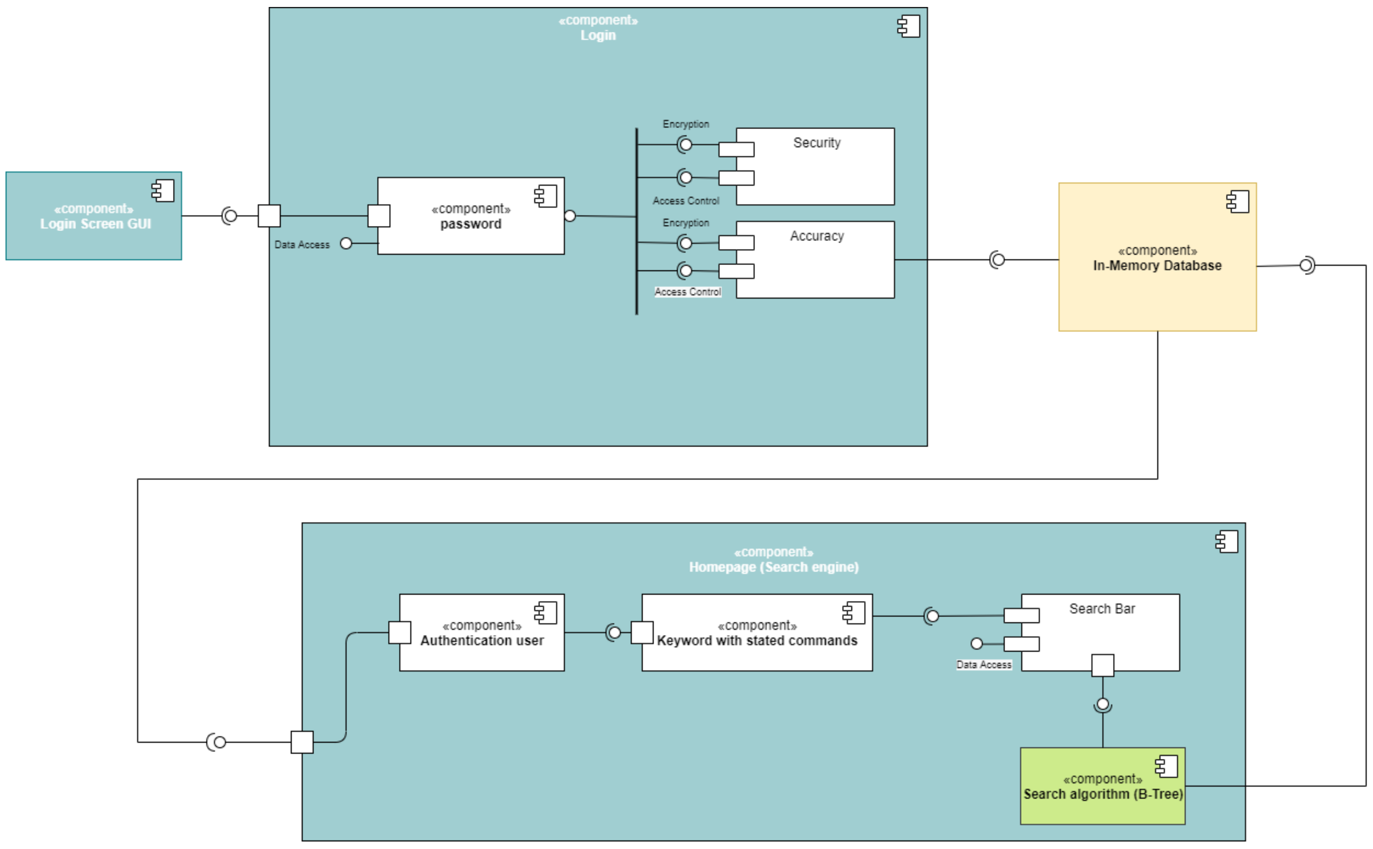

«component» Login
«component» Login Screen GUI
Data Access
«component» password
Encryption
Security
Access Control
Encryption
Accuracy
Access Control
«component» In-Memory Database
«component» Homepage (Search engine)
«component» Authentication user
«component» Keyword with stated commands
Search Bar
Data Access
«component» Search algorithm (B-Tree)

Figure 10. Component Diagram

In **Figure 10**, at first, we have a required Login Screen component that is connected to the entire provided Login component by a port, the Login component includes password component which has data access provided, this component is required for provided security and accuracy which they have an Encryption and Access control, the username is connected internally to Operating System API. It will be created automatically as the same username of the running Operating System. After that, the Login component is required for In-Memory Database that is required for Homepage (Search engine) component and connected with port, Homepage (Search engine) component includes an Authentication user component that is required for Data name (Keyword) component that is required for search bar, Search algorithm component is required for the search bar and search bar provided for it, finally the Search algorithm component is required for In-Memory Database that is provided for it.

4.5.1.7 DEPLOYMENT DIAGRAM

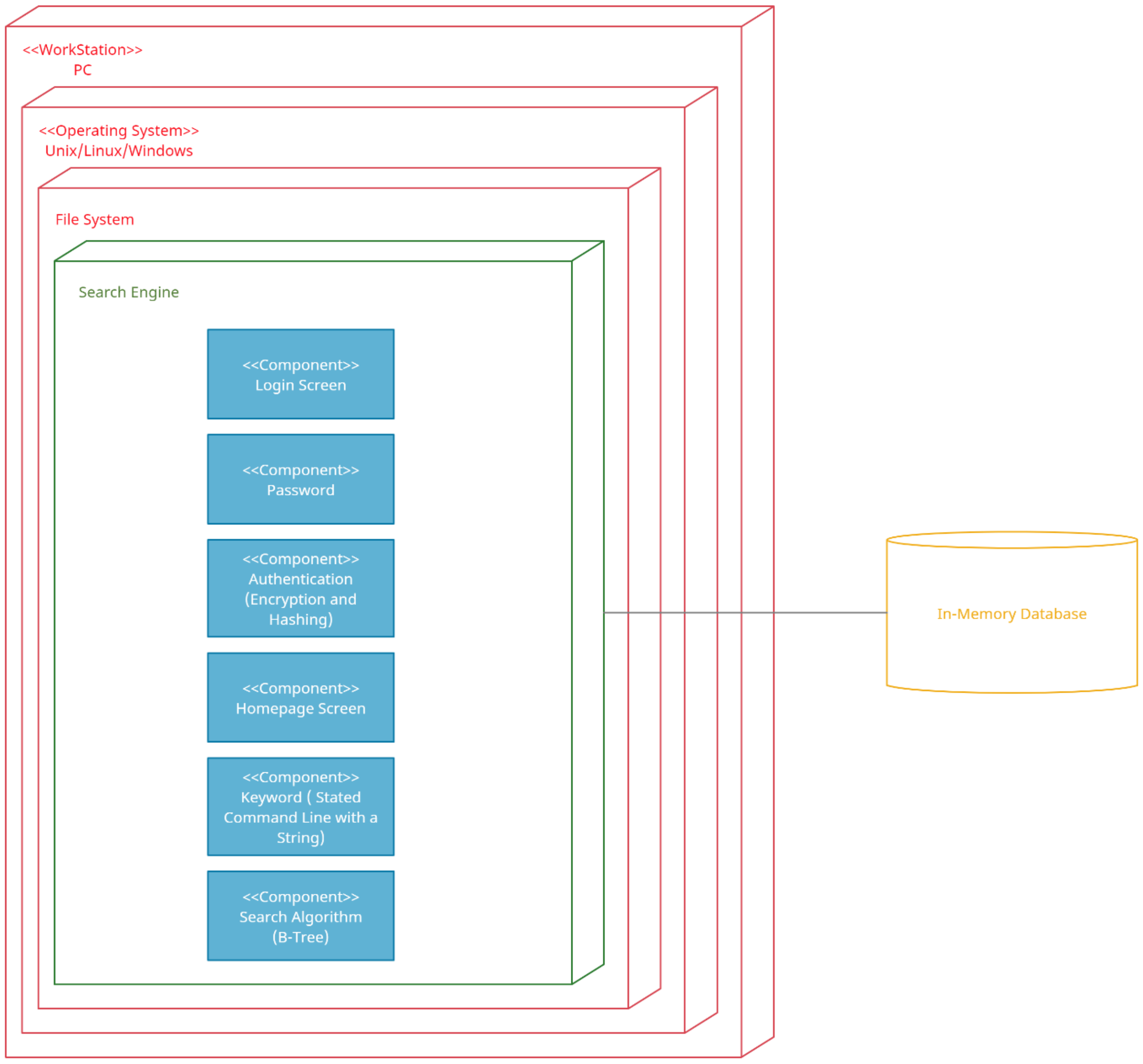


Figure 11. Deployment Diagram

As we can see in **Figure 11**, we have a workstation, and the workstation for this project is the local PC; inside of it, we have an operating system that includes a file system; inside the file system, we have the search engine. It includes Login Screen, Password, Homepage Screen, Authentication user, Keyword, and search algorithm components connected to In-Memory Database.

4.5.2 DOCUMENTATION STANDARDS

Documentation is an essential part of software engineering; it guides to understand of the project context and expectations. Also, it helps to prepare a project plan to achieve outputs. It sets a direction to the project life cycle, and there are three critical parts to take care of in documentation, organization, clarity, and consistency. There are several types of documentation, these parts are:

- **Requirements:** It is the foundation for what is implemented with statements that identify Attributes, capabilities, and characteristics.
- **Design:** The environment and construction principles used in software components.
- **Technical:** Documentation of code, algorithms, interfaces.
- **End-user:** Manuales for the user.

Documentation standards in a software project are essential because documents are the only way of representing the software and the software process. Standardized documents have a consistent view, structure, and quality. There are several types of documentation standards:

- **Documentation process standards:** Defines the process that should be followed for document production.
- **Document standards:** These standards manage the structure and presentation of documents, and they should apply to all documents produced during a software development project.
- **Document interchange standards:** Ensures that all electronic copies of documents are compatible. The use of these standards allows documents to be transferred electronically and re-created in their original form.

In this report, there are three different standard parts that paid attention to them.

1. **Process:** Which includes defining the mission, vision, goals, and tactics, also monitoring and reporting process performance, coming up with process improvement plans.

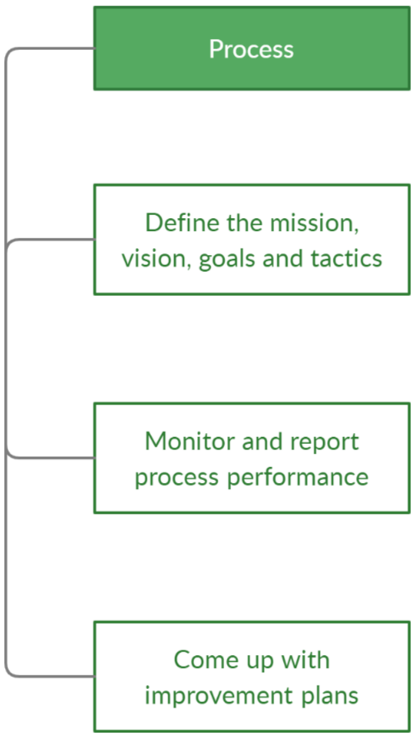

Process
Define the mission, vision, goals and tactics
Monitor and report process performance
Come up with improvement plans

Figure 12. Process

2. **Documentation:** Which includes ensuring that all records are up-to-date, also making sure documents are available for inspection at any time, maintaining an accurate and up to date filing system, and archiving, storing, and destroying obsolete documents.

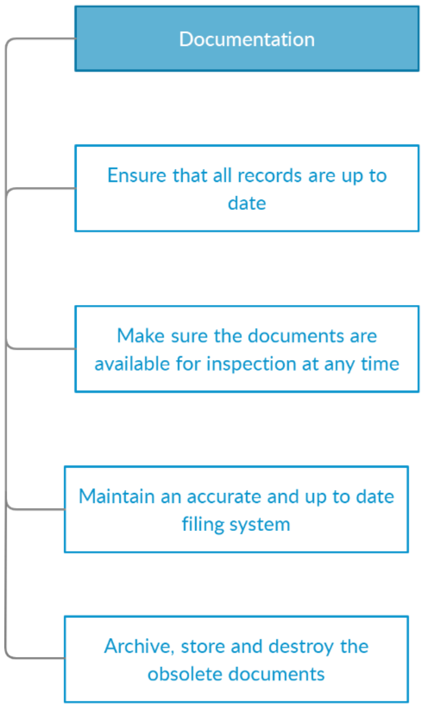

Documentation
Ensure that all records are up to date
Make sure the documents are available for inspection at any time
Maintain an accurate and up to date filing system
Archive, store and destroy the obsolete documents

Figure 13. Documentation

3. **Technical writing**: It includes organizing material and write process documents, editing, standardizing or making changes to the documents, preparing the necessary charts, graphs, and diagrams. Ensuring technical ideas are conveyed in simple language.

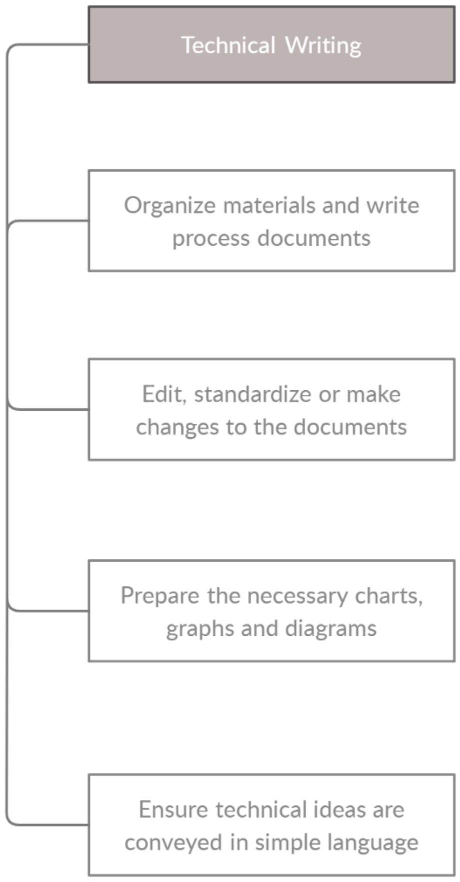

Technical Writing
Organize materials and write process documents
Edit, standardize or make changes to the documents
Prepare the necessary charts, graphs and diagrams
Ensure technical ideas are conveyed in simple language

Figure 14. Technical writing

4.5.3 NAMING STANDARDS

The naming standards are a set of rules applied when creating text scripts to select the character string that will be used for identifiers that define variables, types, functions, and other parts in source code because well-chosen identifiers make it easier to understand what the system is doing and how to fix or extend the source code to apply new needs. There are several reasons for using naming standards:

- To reduce the effort needed to understand the source code.
- To ensure that code reviews focus on more important issues than discussing syntax and naming standards.
- To ensure that code quality review tools focus their reporting on important issues other than syntax and style preferences.

In this project, there are several rules applied in naming standards for different parts and are followed:

- **File naming:**
  - Files are named consistently.
  - File names are short but descriptive.
  - Special characters or spaces are avoided in a file name.
  - Relevant components are used in file names to provide description and context.
  - Capitals and underscores are used instead of periods, or spaces or slashes.
  - Common date format is used.
- **File Structures:** Attention was paid to the hierarchical file structures because they add additional organization to the files.
- **File formats:** File formats have a direct impact on the ability to open and access those files at a later date; these formats are:
  - Non-proprietary.
  - Unencrypted.
  - Uncompressed.
  - In common usage by the research community.
  - Interoperable among different platforms and applications.
  - Conforms to an open, documented standard.

- **Naming standards in code:**
  - **Package:** The prefix of a unique package name must always be written in all-lowercase ASCII letters, and if the name contains multiple words, it should be separated by dots.
  - **Class:** It should start with an uppercase letter, be a noun, and use appropriate words instead of acronyms.
  - **Interface:** It should start with an uppercase letter, be an adjective, and use appropriate words instead of acronyms.
  - **Method:** It should start with a lowercase letter and be a verb.

- **Variable:** variable names must be meaningful. The choice of a variable name should be mnemonic. It should start with a lowercase letter and should not start with special characters; using one-character variables must be avoided.
- **Constant:** It should be in uppercase letters; if the name contains multiple words, it should be separated by an underscore, and it may contain digits but not as the first letter.

4.5.4 PROGRAMMING STANDARDS

This project is completed with Object-Oriented Java programming language since it is including natural processing language and safe; it also comes with better high-level concurrency tools such as:

- **Lock objects:** they support locking idioms that simplify many concurrent applications.
- **Executors:** they define a high-level API for launching and managing threads.
- **Concurrent collections:** They make it easier to manage large data collections, and they reduce the need for synchronization.
- **Atomic variables:** They have features that minimize synchronization and help avoid memory consistency errors.
- **ThreadLocalRandom:** provides efficient generation of pseudorandom numbers from multiple threads.

There are specific points standardized, these points are:

a) Modularity and structuring

Modularity is a general concept that applies to software development that allows individual modules to be developed and offers benefits in returning margins to scale. A module must contain a module descriptor which is the metadata that specifies the module's dependencies, and it is the compiled version of a module declaration; each module declarations must begin with the keyword module, followed by a unique module name and a module body enclosed in braces. In this project, the Java 9 module is used, and the main components of it are one module, module name, module descriptor, set of packages, and set of types and resources.

b) Headers and commenting

Styling is important to make the code elegant and readable, and these are standards of styling that are followed:

- Including a header comment at the top of the class containing the "main" method, and this header should contain
    - The name.
    - The course number.
    - The name of the assignment.
    - The date of last modification to the file.
- Using inline comments.
- Writing self-documenting code:
    - Choosing meaningful names for all variables, parameters, classes, and methods.
    - Using named constants if it is used more than once.
    - Avoiding repetitious code by writing an appropriately named procedure and calling it multiple times.
- Following standard formatting conventions.

c) Indenting and layout:

Two parts that are standardized:

- **Line length:** Avoiding lines longer than eighty characters since they are not handled well by many terminals and tools **[12]**.
- **Wrapping lines:** When an expression will not fit on a single line, breaking it must be according to the general principles:
    - Breaking after the comma.
    - Breaking before an operator.
    - Preferring higher-level breaks to lower-level breaks.
    - Align the new line with the beginning of the expression at the same level on the previous line **[13]**.

d) Library routines used:

The library routines that are used; are the standard library methods that are built-in methods in Java; these standard libraries come along with the Java Class library that is presented in Java Archive (*.jar) file with Java Virtual Machine (JVM) and Java Runtime Environment (JRE). Also, to access a library routine from a java program, the Java Software Development Kit (SDK)'s Java Native Interface (JNI), gives compile and run-time support for calling native code if exists.

e) Language constructs used and avoided:

Standards of writing software are determined and followed; these are the standards of language constructs that used and avoided:

- Writing the source code. The source code must be flexible and understandable in order not to get confused. Here are the technique standards:

  - Making an app run for multiple users on multiple machines and multiple OS types.
  - Abstracting common functionality.
  - Designing a switchable code.
  - Using accessor functions.
  - Avoiding the use of literals, use constants.
  - Separating data and code.
  - Avoiding static variables if it is not necessary.
  - Using least specific parameter types.
  - Reporting errors.
  - Keeping method side effects to a minimum.

  Also, some rules in the coding part that taken care of:

  - **Getter Methods:** It should be a public method; the method name should be prefixed with "get", and it should not take any argument.
  - **Setter Methods:** It should be a public method; the return type should be void, the method name should be prefixed with "set", and it should take some argument.
  - **Coding convention for Listeners:** To register a Listener, the method name should be prefixed with add, and to unregister a Listener, the method name should be prefixed with remove.

- Testing the program and eliminating errors and design flows by debugging.

- Testing with real-world users.

- Writing final documentation.

CHAPTER 5: IMPLEMENTATION

5 CHAPTER 5: IMPLEMENTATION

This chapter explains the system's architecture design and implementation output; the issues and challenges that occurred while implementing will also be discussed.

5.1 SYSTEM ARCHİTECTURE

In this part, the structures, behaviors, and views of the system will be explained. A system architecture can consist of system components, and the sub-systems develop that will work together to implement the overall system **[14]**. There are several parts to mention because each part has its methods and architecture individually.

5.1.1 LOGIN AND AUTHENTICATION ARCHITECTURE

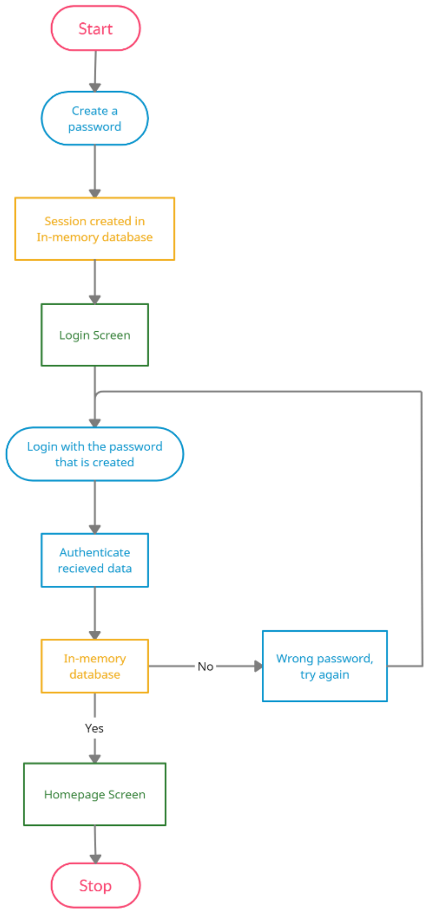

Start
Create a password
Session created in In-memory database
Login Screen
Login with the password that is created
Authenticate recieved data
In-memory database
No
Wrong password, try again
Yes
Homepage Screen
Stop

Figure 15. Login And Authentication Architecture

As we can see from **Figure 15** above, when the user enters his password from the GUI screen, the data will be received and scanned in the In-Memory Database; after the authentication, the user will be able to access the homepage of the search engine.

5.1.2 SEARCH ARCHITECTURE

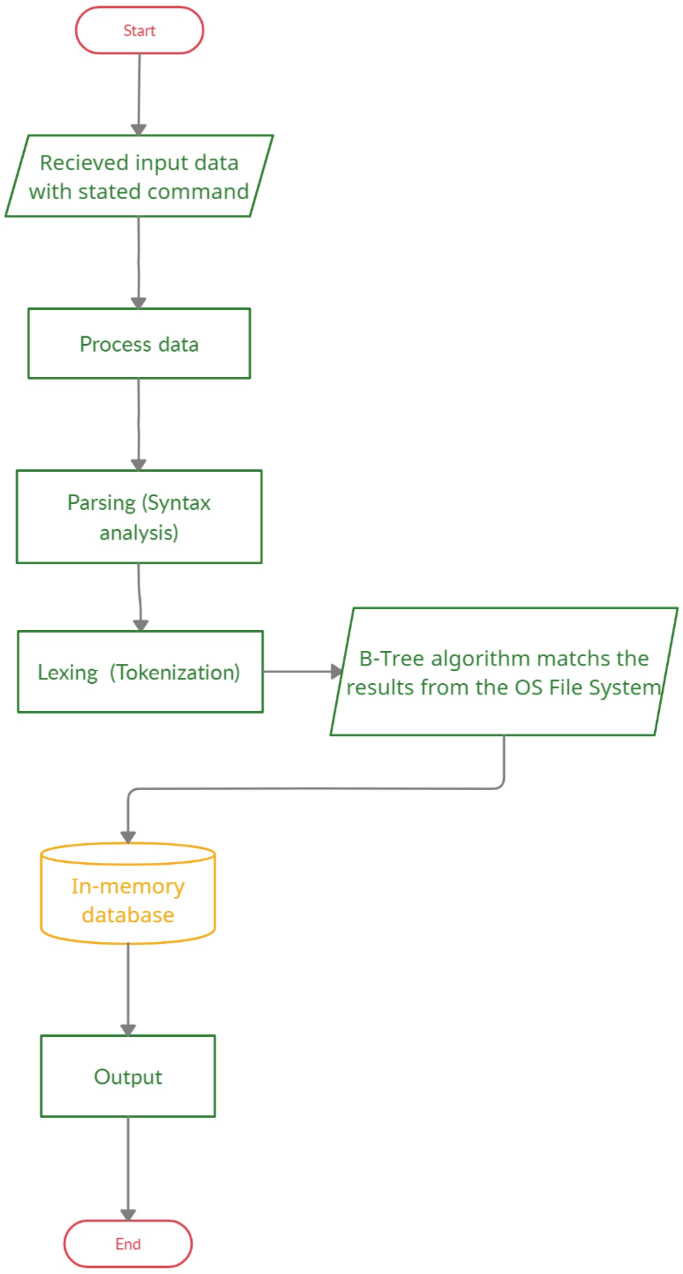

Start
Recieved input data with stated command
Process data
Parsing (Syntax analysis)
Lexing (Tokenization)
B-Tree algorithm matchs the results from the OS File System
In-memory database
Output
End

Figure 16. Search Architecture

Here in **Figure 16** above, we can see how the search method is applied inside the application; at first, when the user enters the keyword input, the program will receive the data and process it, the parsing process, also called (Syntax analysis) will analyze a string of symbols that received. Then the lexing process, also called (Tokenization) will convert a sequence of characters into a sequence of tokens; after that, that the search tree algorithm will be applied to match the key from the key-value pair to find the result, the data, and information will be fetched from the in-memory database and the output will be displayed.

5.1.3 SUPERFICIAL ARCHITECTURE OF THE SYSTEM

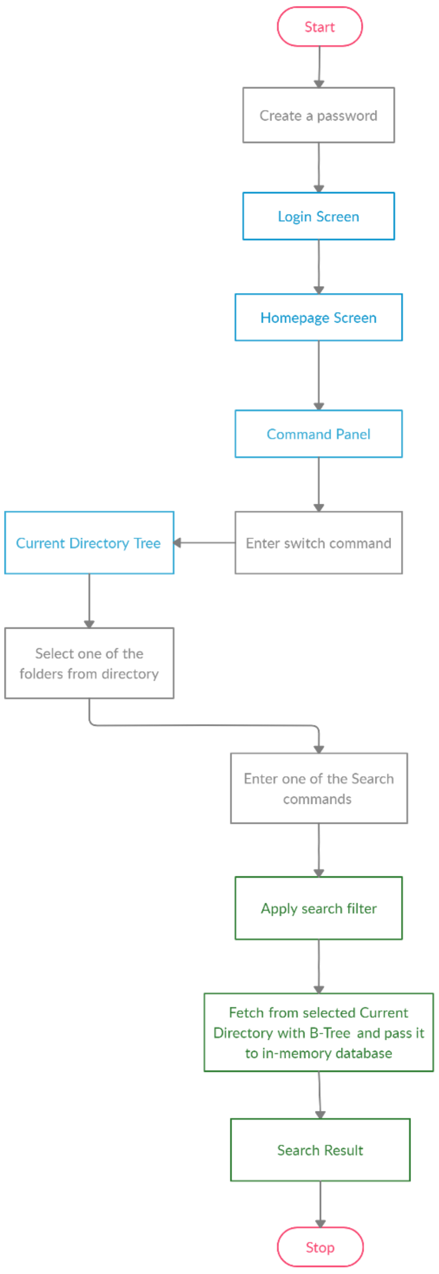

Start
Create a password
Login Screen
Homepage Screen
Command Panel
Enter switch command
Current Directory Tree
Select one of the folders from directory
Enter one of the Search commands
Apply search filter
Fetch from selected Current Directory with B-Tree and pass it to in-memory database
Search Result
Stop

Figure 17. Superficial Architecture Of The System

As we can see in **Figure 17**, the user must create a password for security standards; after that, the user enters the password to the login screen, and then the user can access the homepage screen of the application. There are three separated screens in the homepage screen, the command panel which shows the options and stated commands to the user, when the user enters the “switch” command, he can switch between files in the directory, when the user selects a file, it will be shown in current directory tree part, and when the user selects one of the search commands and types it with a string or a regular expression, by clicking enter the search algorithm of B-tree will match the results and pass them to the in-memory database, after that the user will see the matched results.

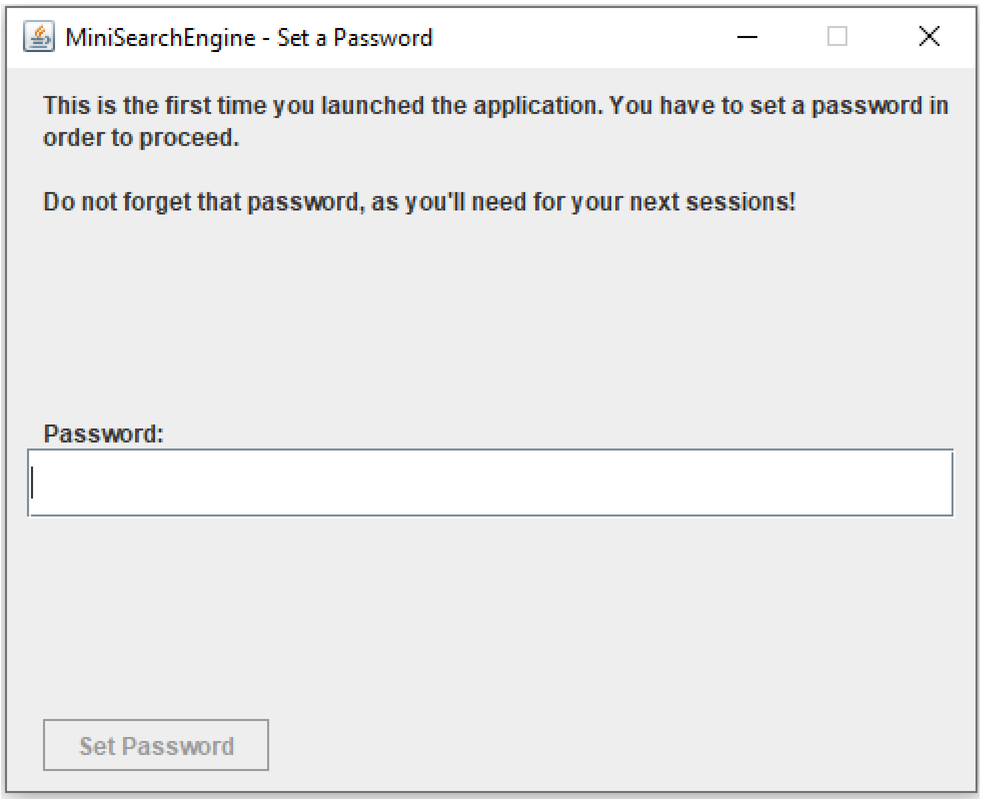
5.2 BUILDING SOFTWARE

When the requirements are met, a software desktop application was built, it was challenging to complete the building of the software, the time considered as a cost when trying to build such a project; also I had some troubles with the application and code, so I searched a lot to get it done, thus the mini search engine application was ready.

5.3 IMPLEMENTATION OUTPUT

The project output and processes step by step will be shown below.

- **First time launching the application (Setting Password and Logging In).**

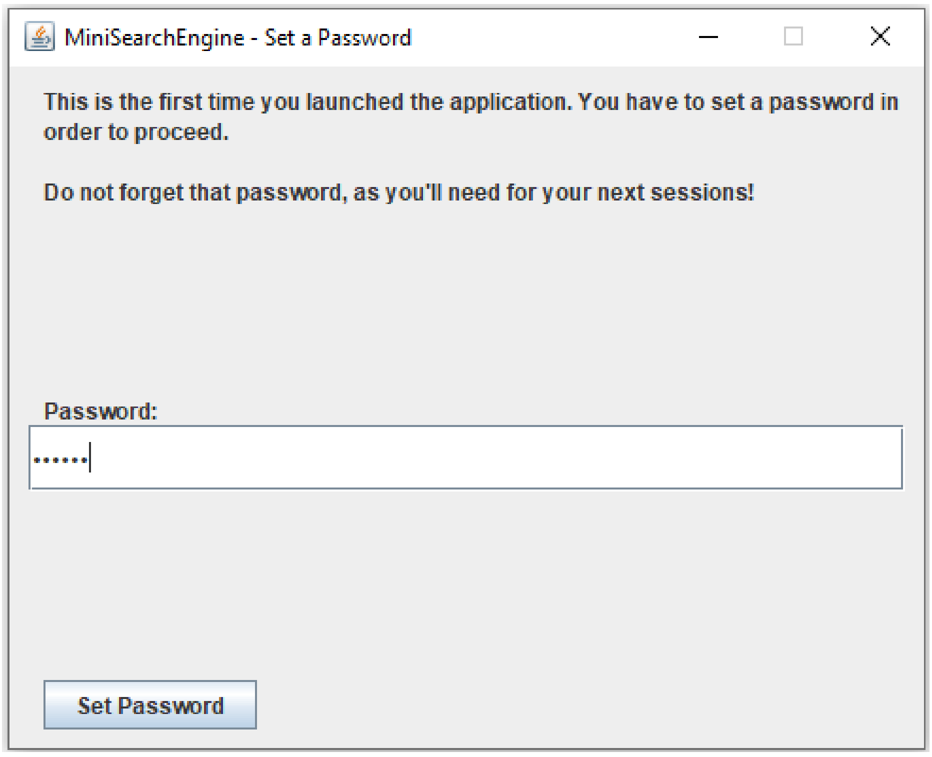

MiniSearchEngine - Set a Password

This is the first time you launched the application. You have to set a password in order to proceed.

Do not forget that password, as you'll need for your next sessions!

Password:

When the user launches the application for the first time, the application informs the user to set a password in order to proceed; and the "Set Password" button will be unclickable.



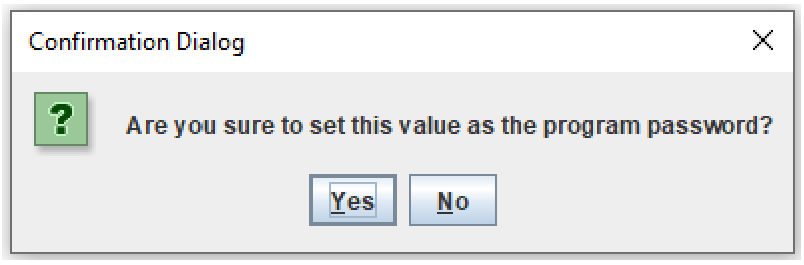
This is the first time you launched the application. You have to set a password in order to proceed.

Do not forget that password, as you'll need for your next sessions!

Password:

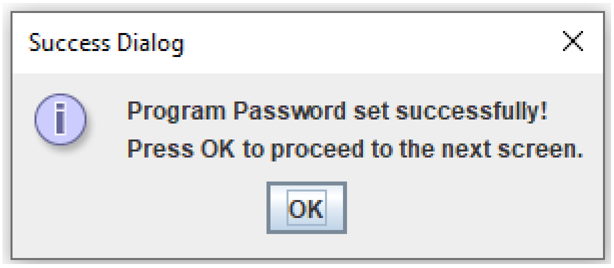
••••••

When the user types any keyword, the “Set Password” button will become clickable to set the password.



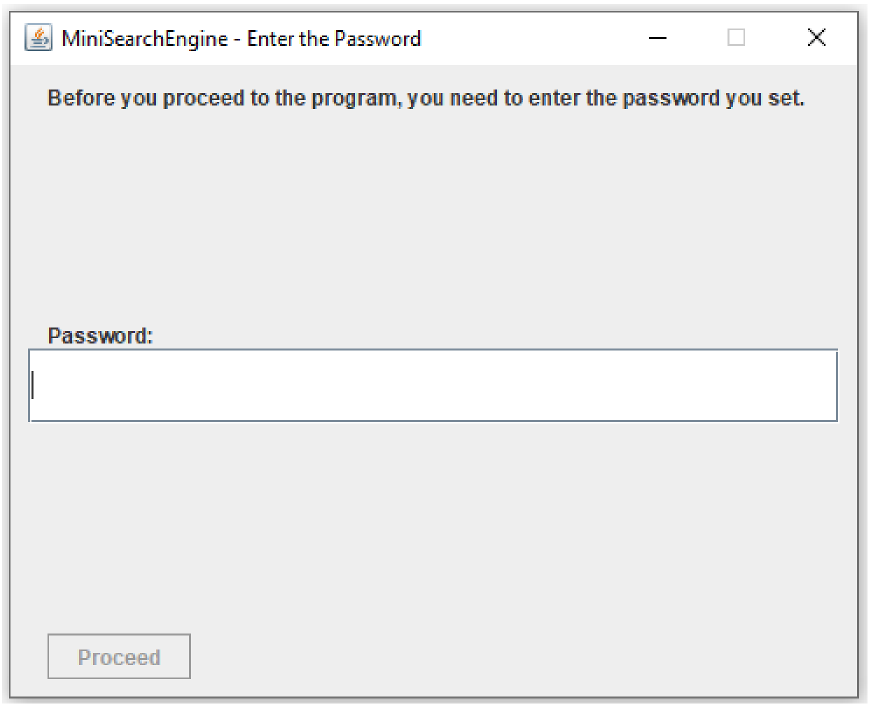
Are you sure to set this value as the program password?

When the user clicks the “Set Password” button, a “Confirmation Dialog” window will arise; this window will ask the user to confirm the value entered as a password. If the user clicks the “No” button, then the window will be closed, and the user will still be in the setting password screen. If the user clicks yes, a “Success Dialog” window will arise.



Program Password set successfully!
Press OK to proceed to the next screen.

When the user clicks the “Ok” button in the “Success Dialog” window, he will be directed to the “Enter the Password” window in order to access the “Search Console” screen of the search engine.

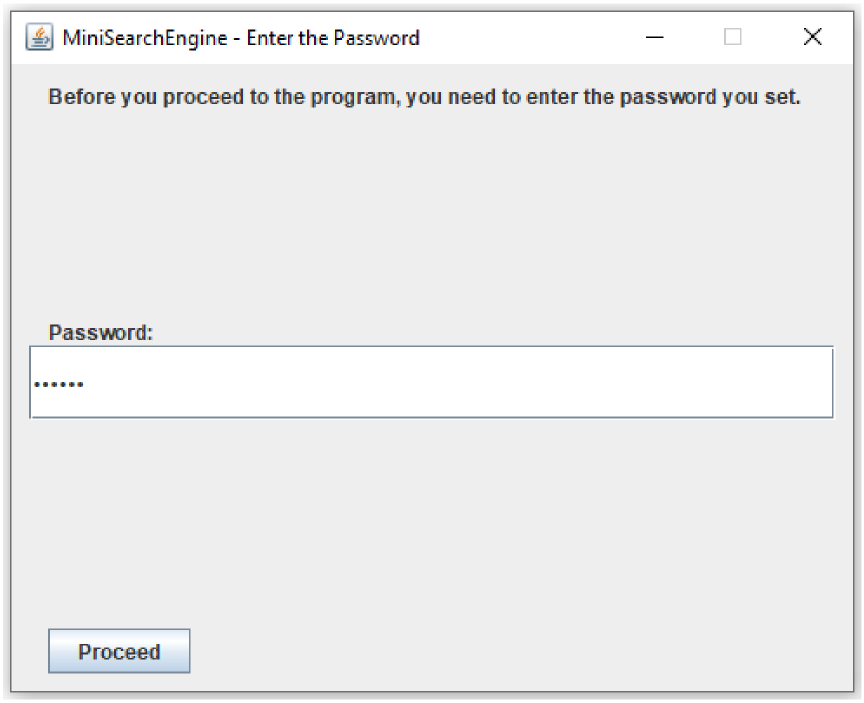


MiniSearchEngine - Enter the Password

Before you proceed to the program, you need to enter the password you set.

Password:

When the user types any keyword, the “Proceed” button will become clickable to access the “Search Console” screen of the search engine.

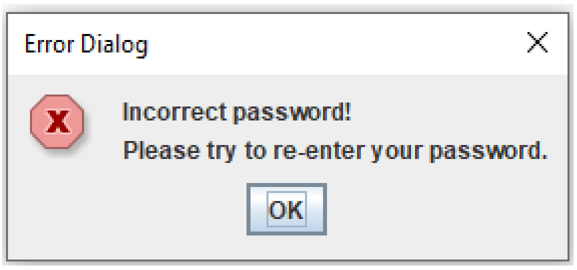


MiniSearchEngine - Enter the Password

Before you proceed to the program, you need to enter the password you set.

Password: ••••••

If the user types a wrong password and clicks the “Proceed Button,” and “Error Dialog” screen will be shown to the user to inform the user that he entered a wrong password.



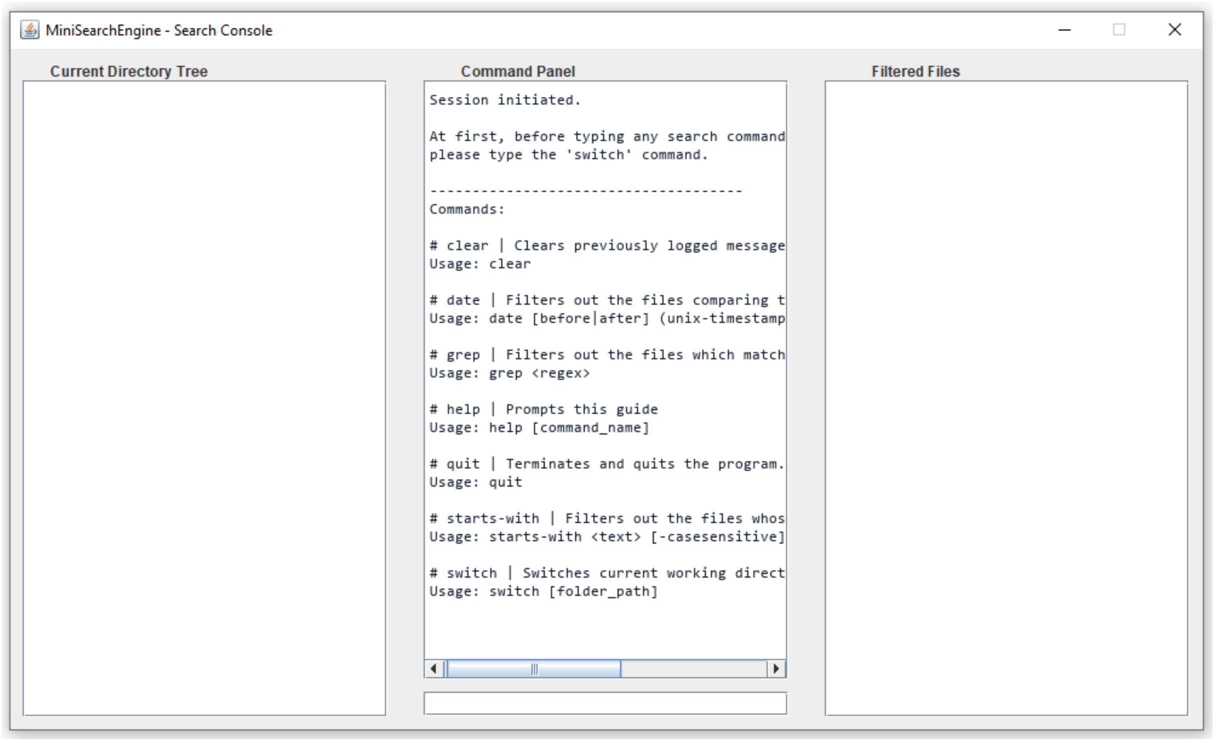
Error Dialog

Incorrect password!
Please try to re-enter your password.

When the user clicks the "OK" button, the "Error Dialog" screen will be closed to rewrite the password in the correct form.

When the user enters the correct password, he will be directed to the "Search Console" screen of the mini search engine.

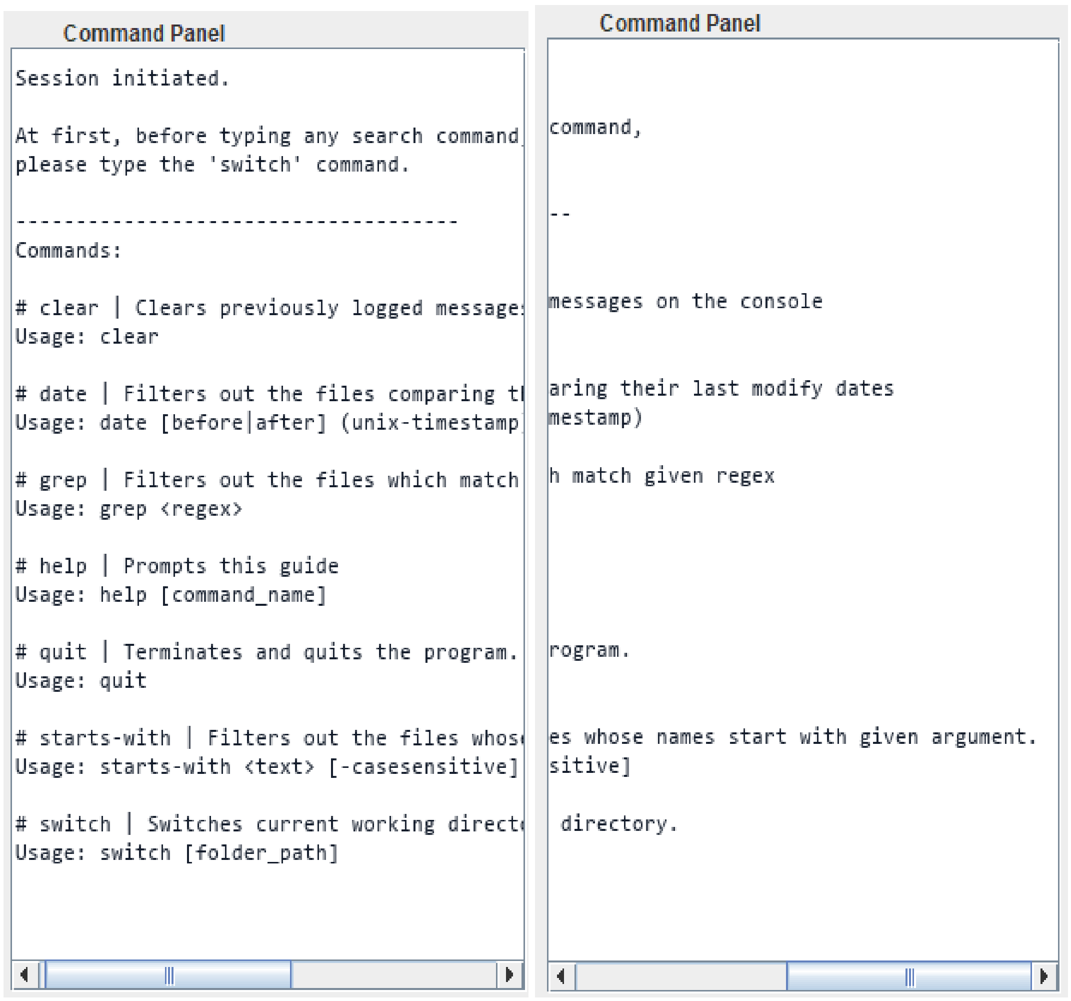
- **Mini Search Engine "Search Console".**



```
Session initiated.

At first, before typing any search command
please type the 'switch' command.

-------------------------------------
Commands:

# clear | Clears previously logged message
Usage: clear

# date | Filters out the files comparing t
Usage: date [before|after] (unix-timestamp

# grep | Filters out the files which match
Usage: grep <regex>

# help | Prompts this guide
Usage: help [command_name]

# quit | Terminates and quits the program.
Usage: quit

# starts-with | Filters out the files whos
Usage: starts-with <text> [-casesensitive]

# switch | Switches current working direct
Usage: switch [folder_path]
```

In the "Search Console," there are three different panels (Command Panel, Current Directory Tree, and Filtered Files); each panel has its own duty, and there is a "Search Bar" that takes the search keyword inputs.

❖ Command Panel.



```
Session initiated.

At first, before typing any search command
please type the 'switch' command.

-------------------------------------
Commands:

# clear | Clears previously logged message
Usage: clear

# date | Filters out the files comparing tl
Usage: date [before|after] (unix-timestamp

# grep | Filters out the files which match
Usage: grep <regex>

# help | Prompts this guide
Usage: help [command_name]

# quit | Terminates and quits the program.
Usage: quit

# starts-with | Filters out the files whos
Usage: starts-with <text> [-casesensitive]

# switch | Switches current working direct
Usage: switch [folder_path]
```

```
command,

--

messages on the console

aring their last modify dates
mestamp)

h match given regex

rogram.

es whose names start with given argument.
sitive]

 directory.
```

When the user launches the application, he will see every information that he needs in the "Command Panel," at first, the user must type the "switch" command before any command else in order to access a folder/file from the file system of the directory.

There are seven different commands, each has a separate task, and these commands are:

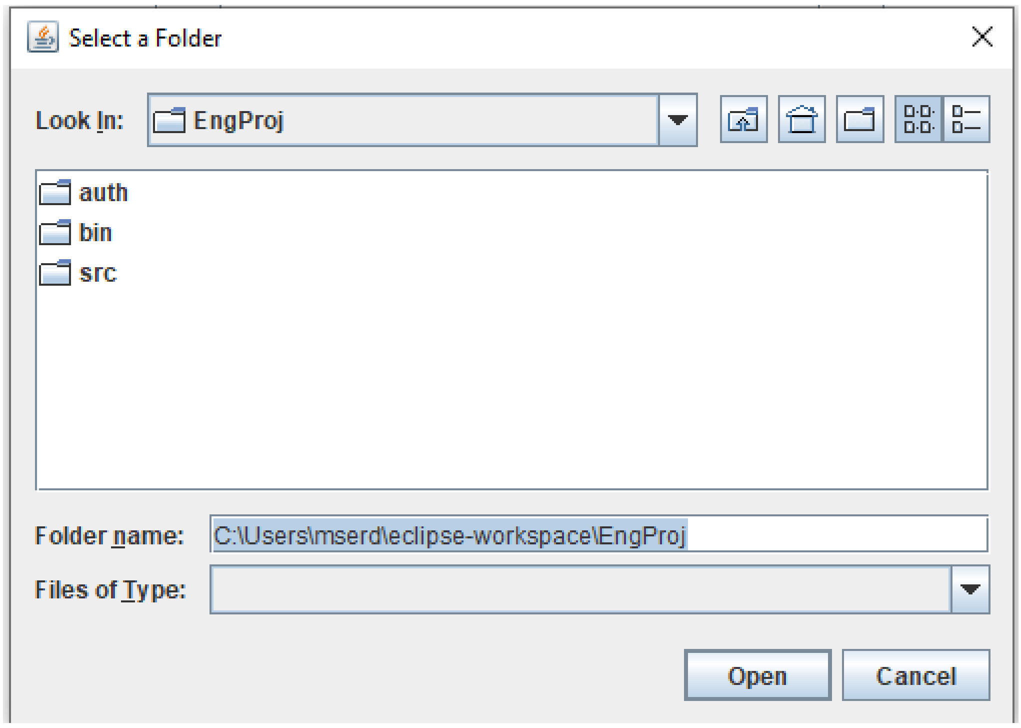
- **clear:** Clears previously logged messages on the console. The usage of this command is just typing the command itself "clear."
- **date:** Filters out the files comparing their last modified dates. The usage of this command is typing "date (before|after) Unix-time (Epoch Time)."
- **grep:** Filters out the files that match the given regex (Regular Expression), and it will accept any valid regex. The usage of this command is typing "grep <regex>"; it will accept any valid regular expression.
- **help:** Prompts the guide**.** The usage of this command is typing "help" or "help 'command.'"
- **quit:** Terminates and quits the program. The usage of this command is just typing the command itself "quit."
- **starts-with:** Filters out the files whose names start with the given argument. The usage of these commands is typing "starts-with <text>." It is case-sensitive.

- **switch:** Switches the current directory The usage of this command is just typing the command itself "switch."

The "Command Panel" also displays the entered commands with a confirmation if it's true; it also warns when the user enters the wrong usage of a command.

➢ **"switch" command.**

When the user types the "switch" command to the search bar.

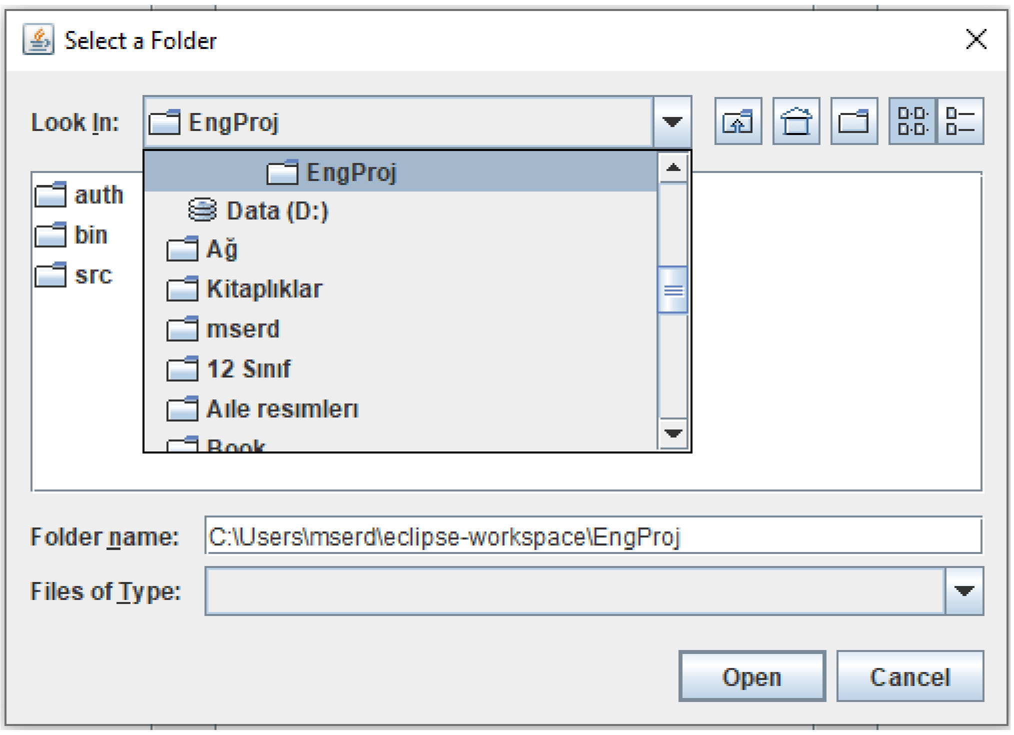


C:\Users\mserd\eclipse-workspace\EngProj

A "Select a Folder" screen will arise; it is created by JFileChooser; via this screen, the user can navigate inside the file system and directories and choose one folder.

There are five different buttons on the top right corner of the "Select a Folder" screen; these buttons will be listed from left to right.

- **First button: Up one level.** It is for turning back to the upper root hierarchically.
- **Second button. Home.** It is for turning back to the main menu.
- **Third button. Create New Folder.**
- **Fourth button. List.** It is to change the content of the information screen to list content (It is in List choice in these screenshots).
- **Fİfth content. Details.** It is to change the content of the information screen to details content to see the detailed information.


Select a Folder

Look In: EngProj

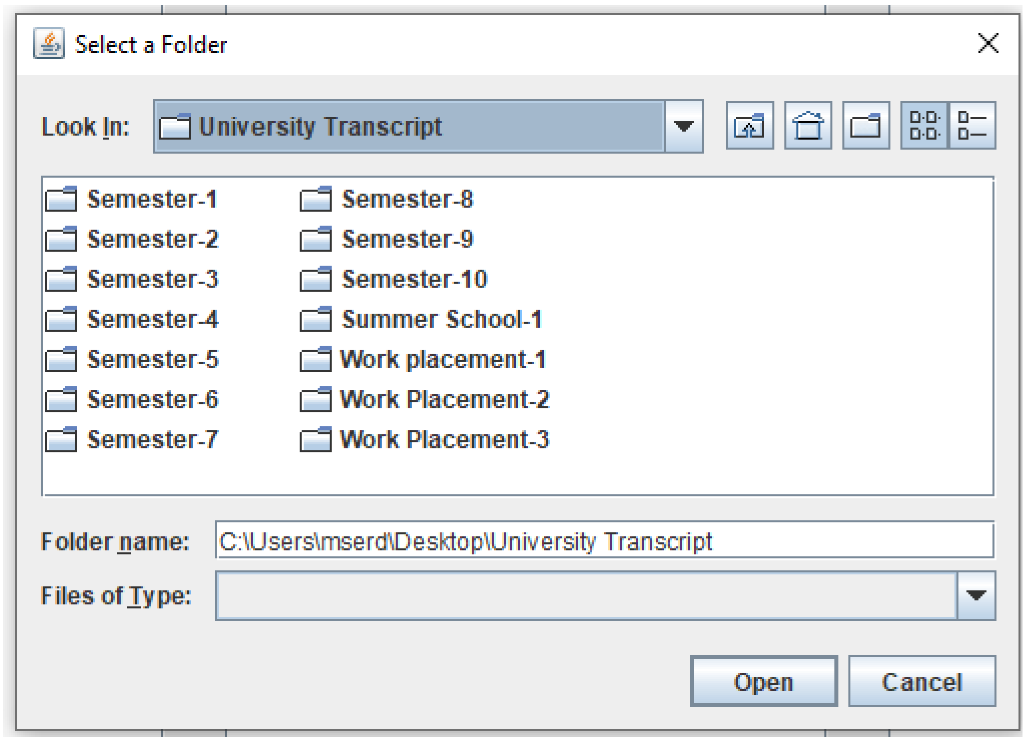
auth
bin
src

EngProj
Data (D:)
Ağ
Kitaplıklar
mserd
12 Sınıf
Aıle resımlerı
Book

Folder name: C:\Users\mserd\eclipse-workspace\EngProj

Files of Type:

When the user clicks in the “Look In” arrow, he will navigate inside the file system and choose a folder to search the files inside.


Select a Folder

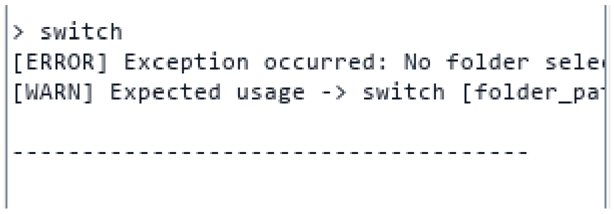
Look In: University Transcript

Semester-1
Semester-2
Semester-3
Semester-4
Semester-5
Semester-6
Semester-7
Semester-8
Semester-9
Semester-10
Summer School-1
Work placement-1
Work Placement-2
Work Placement-3

Folder name: C:\Users\mserd\Desktop\University Transcript

Files of Type:

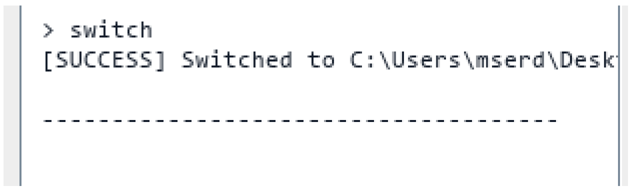
When the user clicks the “Cancel” button, he will receive an error message in the command panel.



```
> switch
[ERROR] Exception occurred: No folder sele
[WARN] Expected usage -> switch [folder_pa

------------------------------------
```

This message tells the error and also corrects the expected usage in case if there is something forgotten.

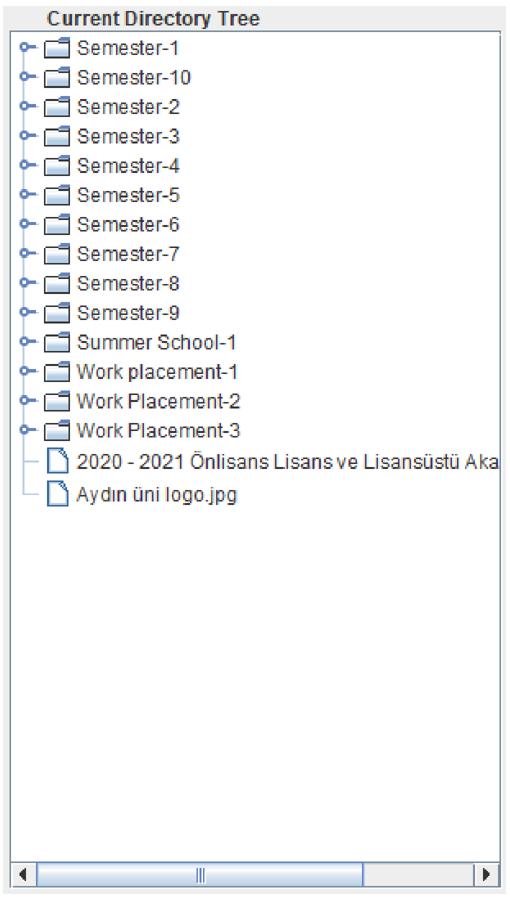
When the user clicks the “Open” button, he will be inside that folder. Also, he will receive a success message.



```
> switch
[SUCCESS] Switched to C:\Users\mserd\Desk

------------------------------------
```

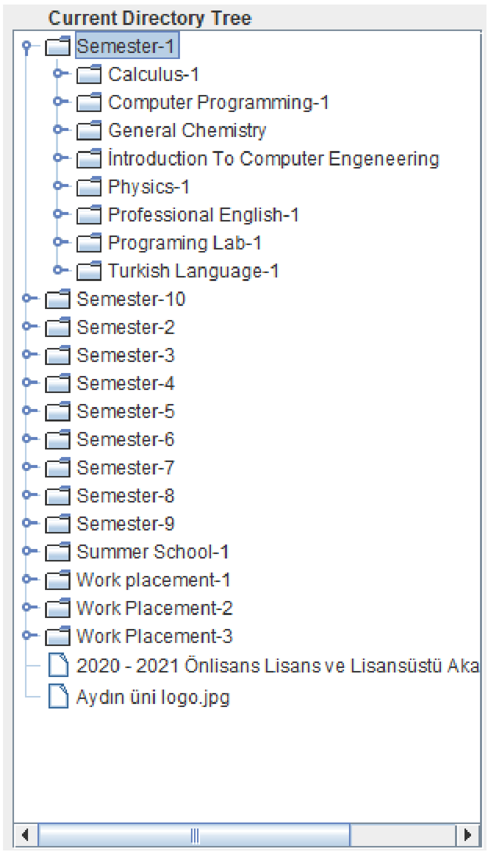
- **Current Directory Tree.**

The chosen folder will be shown in detail in the “Current Directory Tree” panel. Now the user can write one of the stated search commands and search whatever he needs.



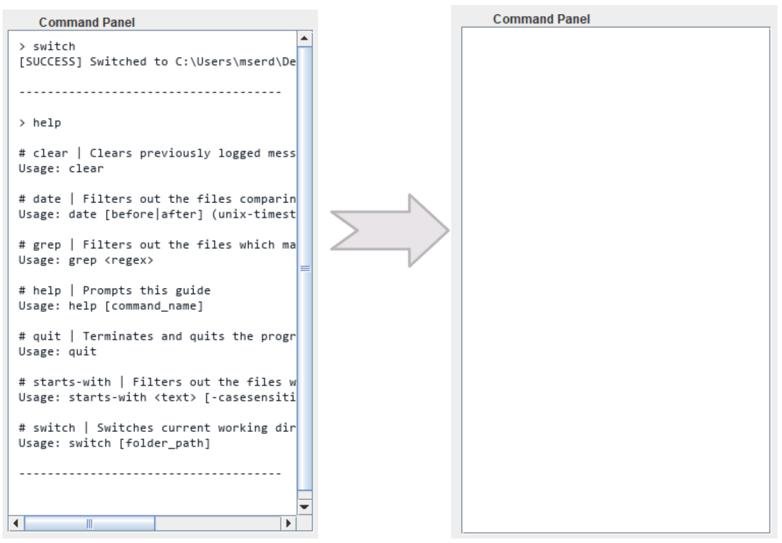
```
Current Directory Tree
Semester-1
Semester-10
Semester-2
Semester-3
Semester-4
Semester-5
Semester-6
Semester-7
Semester-8
Semester-9
Summer School-1
Work placement-1
Work Placement-2
Work Placement-3
2020 - 2021 Önlisans Lisans ve Lisansüstü Aka
Aydın üni logo.jpg
```

When the user clicks on any folder in the “Current Tree Directory” panel, he can see its subclasses.



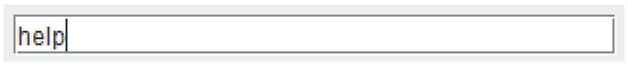
➢ **“clear” command.**

When the user types the “clear” command to the search bar, he clears previous actions and information.



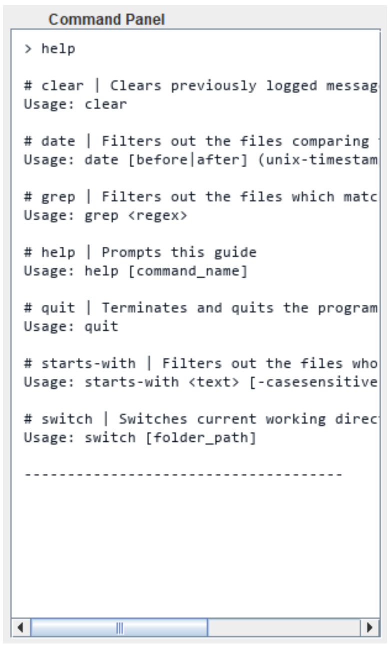
```
> switch
[SUCCESS] Switched to C:\Users\mserd\De

------------------------------------

> help

# clear | Clears previously logged mess
Usage: clear

# date | Filters out the files comparin
Usage: date [before|after] (unix-timest

# grep | Filters out the files which ma
Usage: grep <regex>

# help | Prompts this guide
Usage: help [command_name]

# quit | Terminates and quits the progr
Usage: quit

# starts-with | Filters out the files w
Usage: starts-with <text> [-casesensiti

# switch | Switches current working dir
Usage: switch [folder_path]

------------------------------------
```

- **“help” command.**



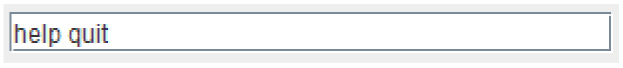
```
help
```

When the user types the “help” command to the search bar, he can see the information that he cleared in the “Command Panel.”

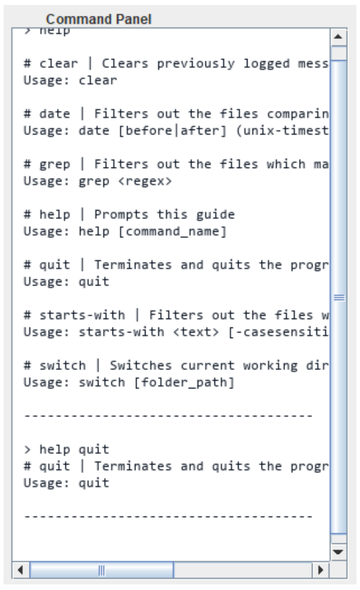


```
> help

# clear | Clears previously logged messag
Usage: clear

# date | Filters out the files comparing
Usage: date [before|after] (unix-timestam

# grep | Filters out the files which matc
Usage: grep <regex>

# help | Prompts this guide
Usage: help [command_name]

# quit | Terminates and quits the program
Usage: quit

# starts-with | Filters out the files who
Usage: starts-with <text> [-casesensitive

# switch | Switches current working direc
Usage: switch [folder_path]

-------------------------------------
```

Also, there is another way of using the “help” command; the user can type beside “help” any one of the given commands in the “Command Panel” to see its property.



```
help quit
```



```
> help

# clear | Clears previously logged mess
Usage: clear

# date | Filters out the files comparin
Usage: date [before|after] (unix-timest

# grep | Filters out the files which ma
Usage: grep <regex>

# help | Prompts this guide
Usage: help [command_name]

# quit | Terminates and quits the progr
Usage: quit

# starts-with | Filters out the files w
Usage: starts-with <text> [-casesensiti

# switch | Switches current working dir
Usage: switch [folder_path]

-------------------------------------

> help quit
# quit | Terminates and quits the progr
Usage: quit

-------------------------------------
```

As we can see above, the information of the “quit” command is listed.

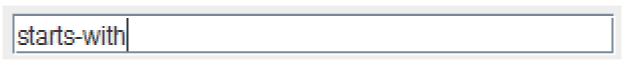
❖ **Search commands.**

There are three different command filters for searching, and they will be listed and explained below.

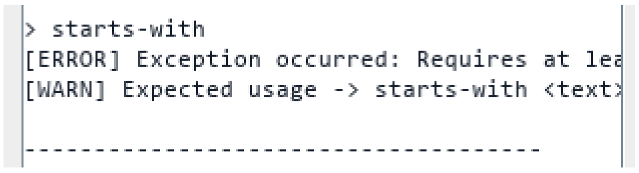
Note: The application can not search for files that the system is not unauthorized to access.

➢ **“starts-with” command.**

When the user types the “starts-with” command to the search bar, he will receive an error because it should be a valid argument beside it.



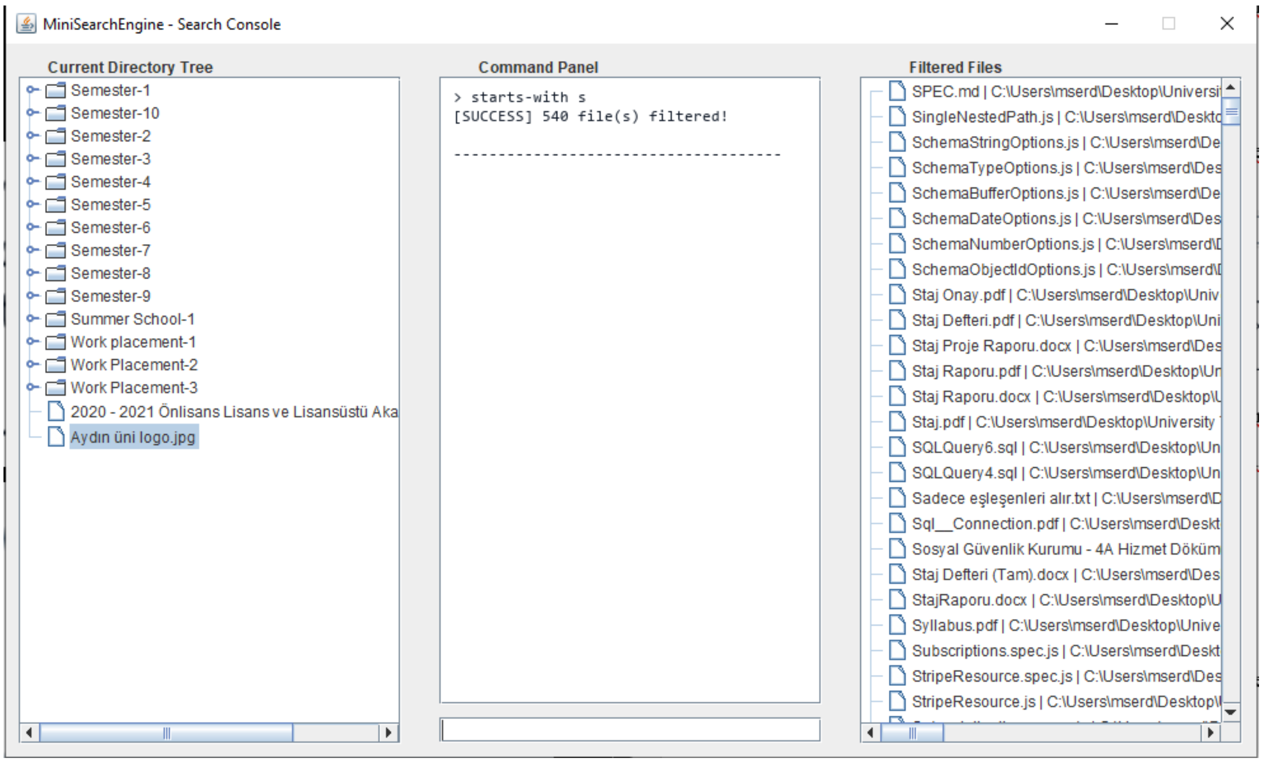
```
starts-with
```



```
> starts-with
[ERROR] Exception occurred: Requires at lea
[WARN] Expected usage -> starts-with <text>

------------------------------------
```

So, one of the valid arguments that should be beside the “starts-with” command is:

- **starts-with s,** this will list every file that starts with s in the chosen folder.



Current Directory Tree:

```
Semester-1
Semester-10
Semester-2
Semester-3
Semester-4
Semester-5
Semester-6
Semester-7
Semester-8
Semester-9
Summer School-1
Work placement-1
Work Placement-2
Work Placement-3
2020 - 2021 Önlisans Lisans ve Lisansüstü Aka
Aydın üni logo.jpg
```

Command Panel:

```
> starts-with s
[SUCCESS] 540 file(s) filtered!

------------------------------------
```

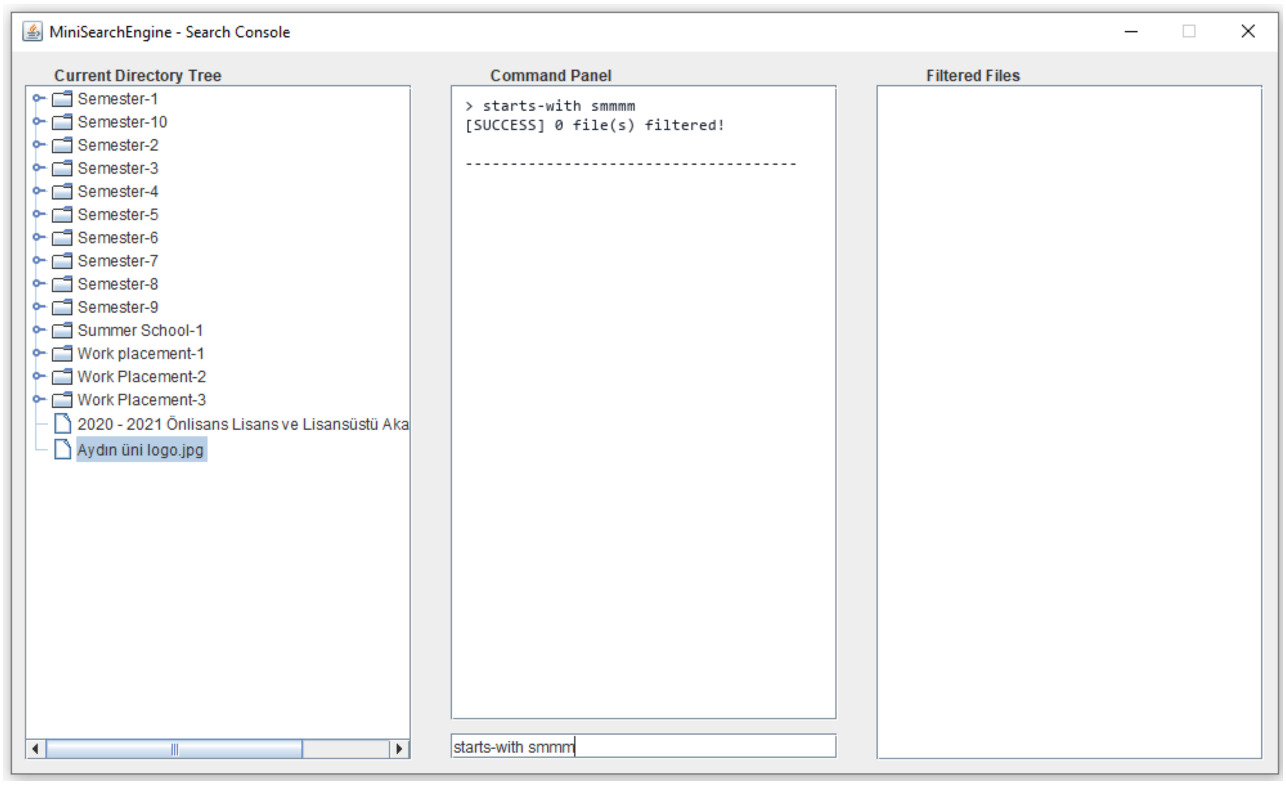
Filtered Files:

```
SPEC.md | C:\Users\mserd\Desktop\Universi
SingleNestedPath.js | C:\Users\mserd\Deskto
SchemaStringOptions.js | C:\Users\mserd\De
SchemaTypeOptions.js | C:\Users\mserd\Des
SchemaBufferOptions.js | C:\Users\mserd\De
SchemaDateOptions.js | C:\Users\mserd\Des
SchemaNumberOptions.js | C:\Users\mserd\
SchemaObjectIdOptions.js | C:\Users\mserd\
Staj Onay.pdf | C:\Users\mserd\Desktop\Univ
Staj Defteri.pdf | C:\Users\mserd\Desktop\Uni
Staj Proje Raporu.docx | C:\Users\mserd\Des
Staj Raporu.pdf | C:\Users\mserd\Desktop\Ur
Staj Raporu.docx | C:\Users\mserd\Desktop\U
Staj.pdf | C:\Users\mserd\Desktop\University
SQLQuery6.sql | C:\Users\mserd\Desktop\Un
SQLQuery4.sql | C:\Users\mserd\Desktop\Un
Sadece eşleşenleri alır.txt | C:\Users\mserd\D
Sql__Connection.pdf | C:\Users\mserd\Deskt
Sosyal Güvenlik Kurumu - 4A Hizmet Döküm
Staj Defteri (Tam).docx | C:\Users\mserd\Des
StajRaporu.docx | C:\Users\mserd\Desktop\U
Syllabus.pdf | C:\Users\mserd\Desktop\Unive
Subscriptions.spec.js | C:\Users\mserd\Deskt
StripeResource.spec.js | C:\Users\mserd\Des
StripeResource.js | C:\Users\mserd\Desktop\
```

As we can see, every available file that starts with the “s” letter is listed in the “Filtered Files” panel, and in the “Command Panel,” the user received a success message with the number of files that starts with “s” letter information.

So there are a lot of arguments that can be written and searched, including numbers. Example: first two letters that start with **“sm,”** first three letters that start with **“sm2”** and etc.

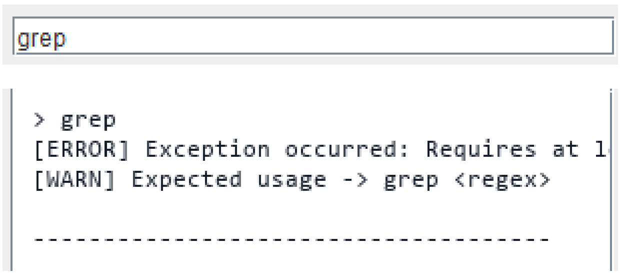
If the given argument with the “starts-with” command is correct and there is no such file, the “Filtered Files” panel will stay empty, and the user receives a success message with the number of files information.



```
> starts-with smmmm
[SUCCESS] 0 file(s) filtered!

------------------------------------
```

➢ **“grep” command.**

When the user types the “grep” command to the search bar. He will receive an error because it should be a valid Regex (Regular Expression) beside it.

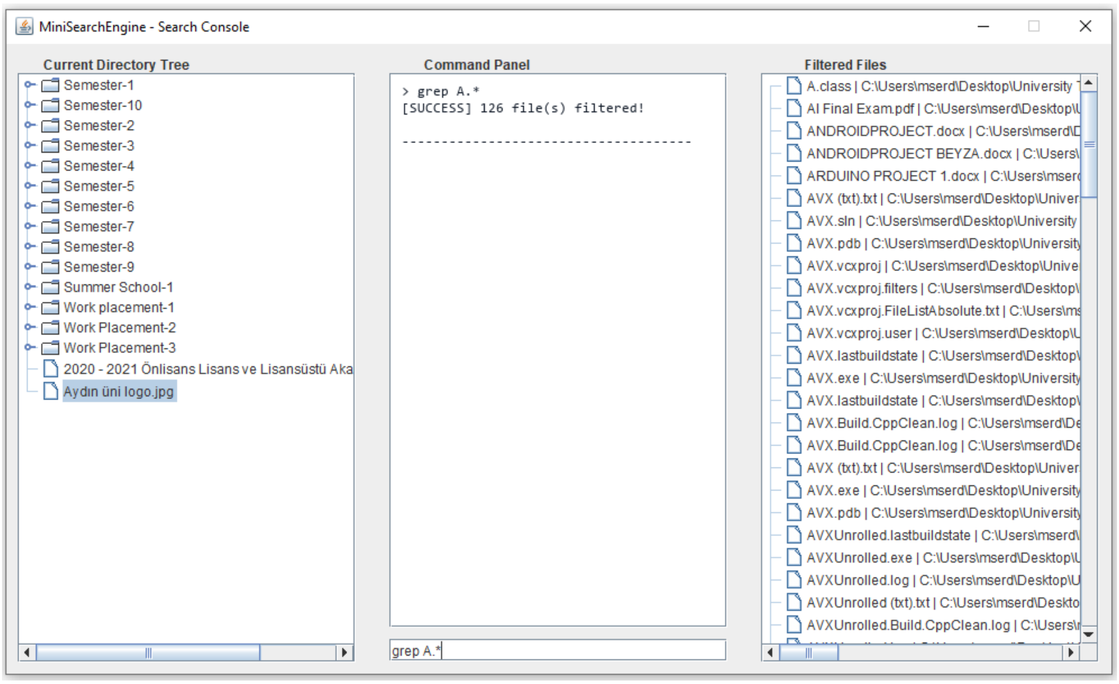


```
grep
```

```
> grep
[ERROR] Exception occurred: Requires at l
[WARN] Expected usage -> grep <regex>

-------------------------------------
```

So, there are unlimited regular expressions that can be searched and matched with the “grep” command. Here are some regular expression search examples.

- **grep A.***, this command means search every file that starts with the “A” letter in the current directory.


MiniSearchEngine - Search Console

Current Directory Tree

```
Semester-1
Semester-10
Semester-2
Semester-3
Semester-4
Semester-5
Semester-6
Semester-7
Semester-8
Semester-9
Summer School-1
Work placement-1
Work Placement-2
Work Placement-3
2020 - 2021 Önlisans Lisans ve Lisansüstü Aka
Aydın üni logo.jpg
```

Command Panel

```
> grep A.*
[SUCCESS] 126 file(s) filtered!

-------------------------------------
```

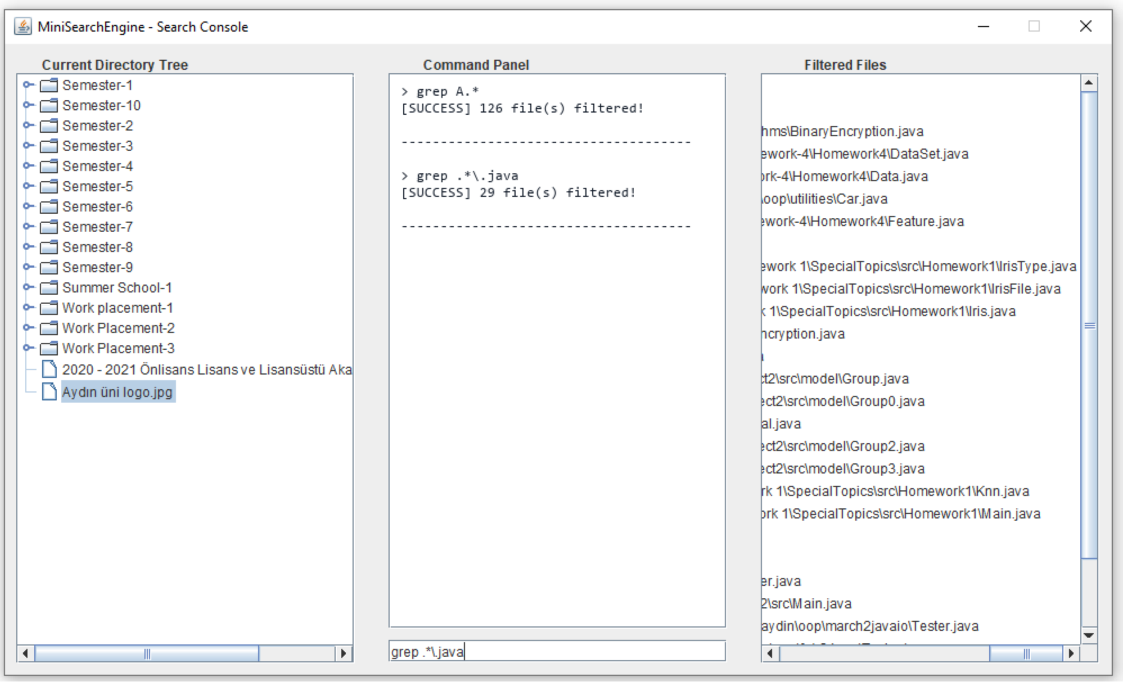
```
grep A.*
```

Filtered Files

```
A.class | C:\Users\mserd\Desktop\University
AI Final Exam.pdf | C:\Users\mserd\Desktop\U
ANDROIDPROJECT.docx | C:\Users\mserd\D
ANDROIDPROJECT BEYZA.docx | C:\Users\
ARDUINO PROJECT 1.docx | C:\Users\mserd
AVX (txt).txt | C:\Users\mserd\Desktop\Univer
AVX.sln | C:\Users\mserd\Desktop\University
AVX.pdb | C:\Users\mserd\Desktop\University
AVX.vcxproj | C:\Users\mserd\Desktop\Unive
AVX.vcxproj.filters | C:\Users\mserd\Desktop\
AVX.vcxproj.FileListAbsolute.txt | C:\Users\ms
AVX.vcxproj.user | C:\Users\mserd\Desktop\U
AVX.lastbuildstate | C:\Users\mserd\Desktop\
AVX.exe | C:\Users\mserd\Desktop\University
AVX.lastbuildstate | C:\Users\mserd\Desktop\
AVX.Build.CppClean.log | C:\Users\mserd\De
AVX.Build.CppClean.log | C:\Users\mserd\De
AVX (txt).txt | C:\Users\mserd\Desktop\Univer
AVX.exe | C:\Users\mserd\Desktop\University
AVX.pdb | C:\Users\mserd\Desktop\University
AVXUnrolled.lastbuildstate | C:\Users\mserd\
AVXUnrolled.exe | C:\Users\mserd\Desktop\U
AVXUnrolled.log | C:\Users\mserd\Desktop\U
AVXUnrolled (txt).txt | C:\Users\mserd\Deskto
AVXUnrolled.Build.CppClean.log | C:\Users\r
```

As we can see, every available file that starts with the "A" letter is listed in the "Filtered Files" panel, and in the "Command Panel," the user received a success message with the number of files that starts with "A" letter information.

- **grep .*\.java**, this command means to search every file that ends with ".java" in the current directory.


MiniSearchEngine - Search Console

Current Directory Tree

```
Semester-1
Semester-10
Semester-2
Semester-3
Semester-4
Semester-5
Semester-6
Semester-7
Semester-8
Semester-9
Summer School-1
Work placement-1
Work Placement-2
Work Placement-3
2020 - 2021 Önlisans Lisans ve Lisansüstü Aka
Aydın üni logo.jpg
```

Command Panel

```
> grep A.*
[SUCCESS] 126 file(s) filtered!

-------------------------------------

> grep .*\.java
[SUCCESS] 29 file(s) filtered!

-------------------------------------
```

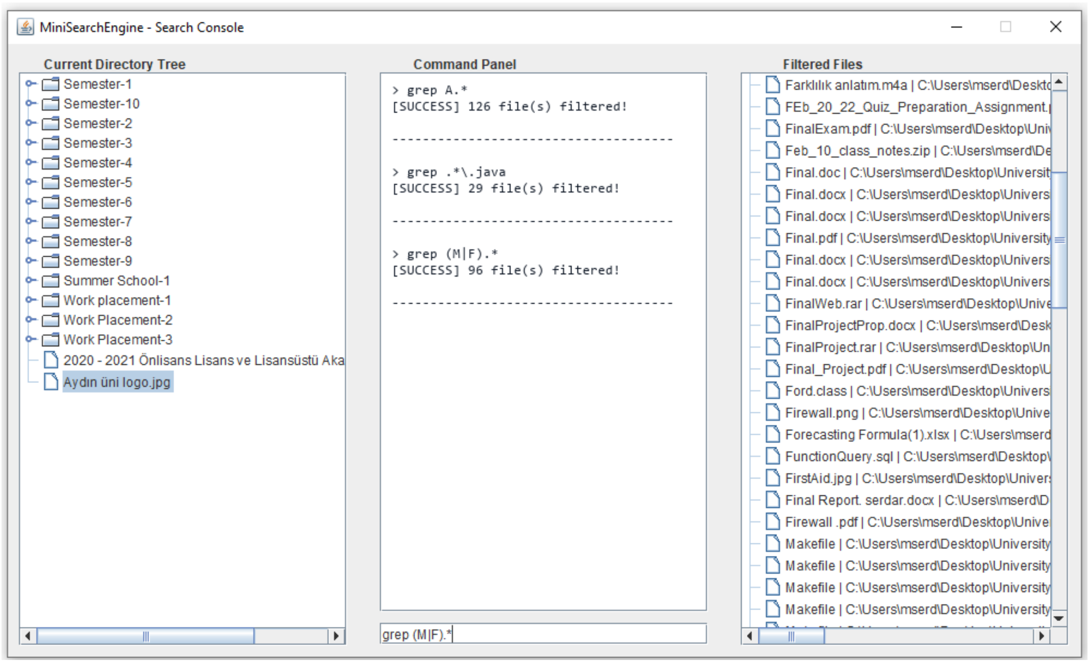
```
grep .*\.java
```

Filtered Files

```
hms\BinaryEncryption.java
ework-4\Homework4\DataSet.java
rk-4\Homework4\Data.java
oop\utilities\Car.java
ework-4\Homework4\Feature.java

ework 1\SpecialTopics\src\Homework1\IrisType.java
work 1\SpecialTopics\src\Homework1\IrisFile.java
 1\SpecialTopics\src\Homework1\Iris.java
ncryption.java

t2\src\model\Group.java
ect2\src\model\Group0.java
al.java
ect2\src\model\Group2.java
ect2\src\model\Group3.java
rk 1\SpecialTopics\src\Homework1\Knn.java
rk 1\SpecialTopics\src\Homework1\Main.java


er.java
2\src\Main.java
aydin\oop\march2javaio\Tester.java
```

As we can see, every available file that ends with ".java" is listed in the "Filtered Files" panel, and in the "Command Panel," the user received a success message with the number of files that end with ".java" information.

- **grep (M|F).***, this command means search every file that starts with "M or F" letters in the current directory.



MiniSearchEngine - Search Console

Current Directory Tree

- Semester-1
- Semester-10
- Semester-2
- Semester-3
- Semester-4
- Semester-5
- Semester-6
- Semester-7
- Semester-8
- Semester-9
- Summer School-1
- Work placement-1
- Work Placement-2
- Work Placement-3
- 2020 - 2021 Önlisans Lisans ve Lisansüstü Aka
- Aydın üni logo.jpg

Command Panel

```
> grep A.*
[SUCCESS] 126 file(s) filtered!

------------------------------------

> grep .*\.java
[SUCCESS] 29 file(s) filtered!

------------------------------------

> grep (M|F).*
[SUCCESS] 96 file(s) filtered!

------------------------------------
```

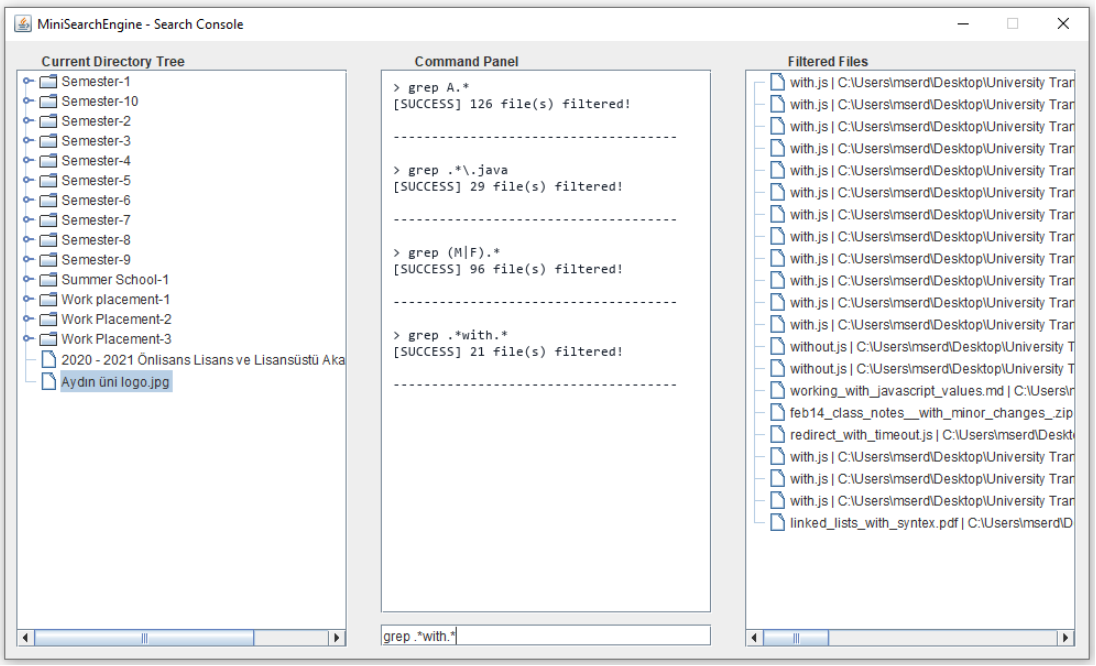
```
grep (M|F).*
```

Filtered Files

```
Farklılık anlatım.m4a | C:\Users\mserd\Deskto
FEb_20_22_Quiz_Preparation_Assignment.
FinalExam.pdf | C:\Users\mserd\Desktop\Uni
Feb_10_class_notes.zip | C:\Users\mserd\De
Final.doc | C:\Users\mserd\Desktop\Universit
Final.docx | C:\Users\mserd\Desktop\Univers
Final.docx | C:\Users\mserd\Desktop\Univers
Final.pdf | C:\Users\mserd\Desktop\University
Final.docx | C:\Users\mserd\Desktop\Univers
Final.docx | C:\Users\mserd\Desktop\Univers
FinalWeb.rar | C:\Users\mserd\Desktop\Unive
FinalProjectProp.docx | C:\Users\mserd\Desk
FinalProject.rar | C:\Users\mserd\Desktop\Un
Final_Project.pdf | C:\Users\mserd\Desktop\U
Ford.class | C:\Users\mserd\Desktop\Univers
Firewall.png | C:\Users\mserd\Desktop\Unive
Forecasting Formula(1).xlsx | C:\Users\mserd
FunctionQuery.sql | C:\Users\mserd\Desktop\
FirstAid.jpg | C:\Users\mserd\Desktop\Univer
Final Report. serdar.docx | C:\Users\mserd\D
Firewall .pdf | C:\Users\mserd\Desktop\Unive
Makefile | C:\Users\mserd\Desktop\University
Makefile | C:\Users\mserd\Desktop\University
Makefile | C:\Users\mserd\Desktop\University
Makefile | C:\Users\mserd\Desktop\University
```

As we can see, every available file that starts with "M or F" letters is listed in the "Filtered Files" panel, and in the "Command Panel," the user received a success message with the number of files that starts with "M or F" letters information.

- **grep .*with.*,** this command means search every file that contains "with" sentence in the current directory.



MiniSearchEngine - Search Console

Current Directory Tree

- Semester-1
- Semester-10
- Semester-2
- Semester-3
- Semester-4
- Semester-5
- Semester-6
- Semester-7
- Semester-8
- Semester-9
- Summer School-1
- Work placement-1
- Work Placement-2
- Work Placement-3
- 2020 - 2021 Önlisans Lisans ve Lisansüstü Aka
- Aydın üni logo.jpg

Command Panel

```
> grep A.*
[SUCCESS] 126 file(s) filtered!

------------------------------------

> grep .*\.java
[SUCCESS] 29 file(s) filtered!

------------------------------------

> grep (M|F).*
[SUCCESS] 96 file(s) filtered!

------------------------------------

> grep .*with.*
[SUCCESS] 21 file(s) filtered!

------------------------------------
```

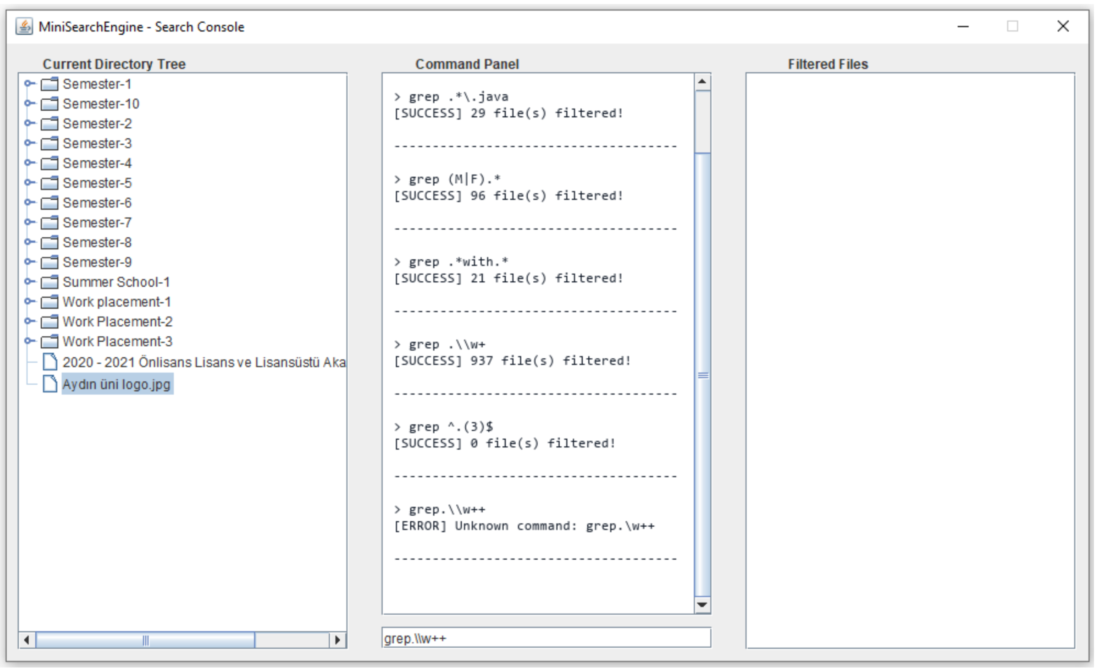
```
grep .*with.*
```

Filtered Files

```
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
without.js | C:\Users\mserd\Desktop\University T
without.js | C:\Users\mserd\Desktop\University T
working_with_javascript_values.md | C:\Users\r
feb14_class_notes__with_minor_changes_.zip
redirect_with_timeout.js | C:\Users\mserd\Deskt
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
with.js | C:\Users\mserd\Desktop\University Trar
linked_lists_with_syntex.pdf | C:\Users\mserd\D
```

As we can see, every available file that contains the "with" sentence is listed in the "Filtered Files" panel, and in the "Command Panel," the user received a success message with the number of files that contains "with" sentences information.

When the user types an unknown regex beside the "grep" command, he will receive an error message in "Command Panel."

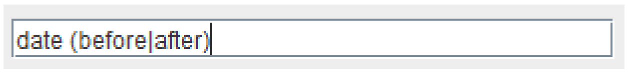


```
> grep .*\.java
[SUCCESS] 29 file(s) filtered!

-------------------------------------

> grep (M|F).*
[SUCCESS] 96 file(s) filtered!

-------------------------------------

> grep .*with.*
[SUCCESS] 21 file(s) filtered!

-------------------------------------

> grep .\\w+
[SUCCESS] 937 file(s) filtered!

-------------------------------------

> grep ^.(3)$
[SUCCESS] 0 file(s) filtered!

-------------------------------------

> grep.\\w++
[ERROR] Unknown command: grep.\w++

-------------------------------------
```

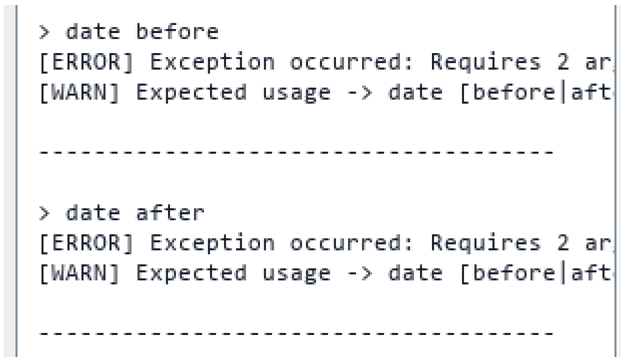
```
grep.\\w++
```

- **“date” command.**

When the user types the “date” command to the search bar, he will receive an error because it should be a valid Unix-time (Epoch time) beside it.



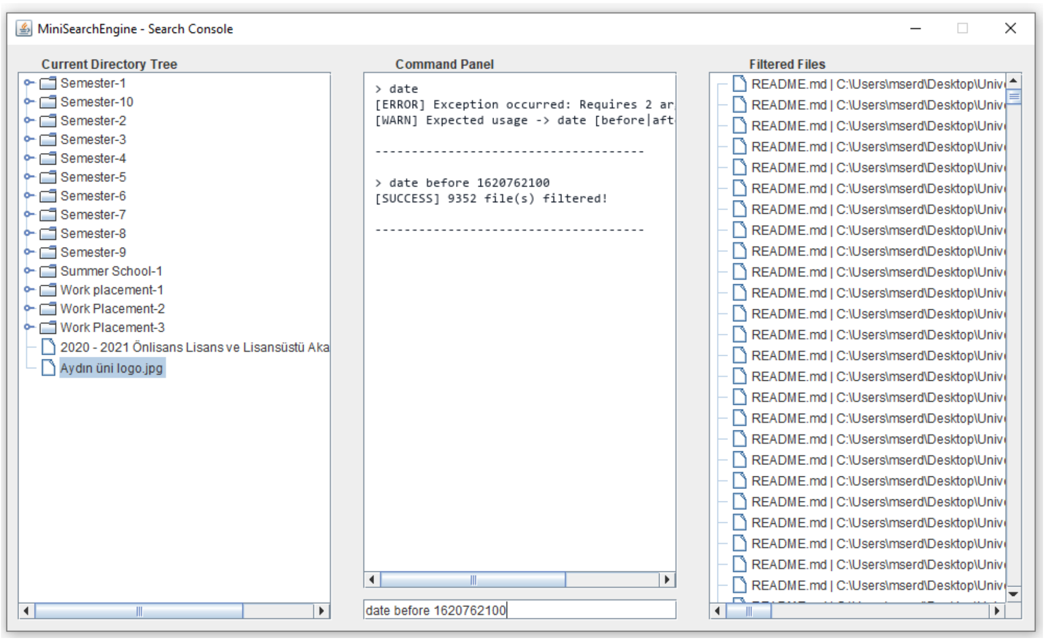
```
date (before|after)
```



```
> date before
[ERROR] Exception occurred: Requires 2 ar
[WARN] Expected usage -> date [before|aft

-------------------------------------

> date after
[ERROR] Exception occurred: Requires 2 ar
[WARN] Expected usage -> date [before|aft

-------------------------------------
```

So, there are unlimited Unix-time values that can be searched and matched with the “date” command. Here are some Unix-time search examples.

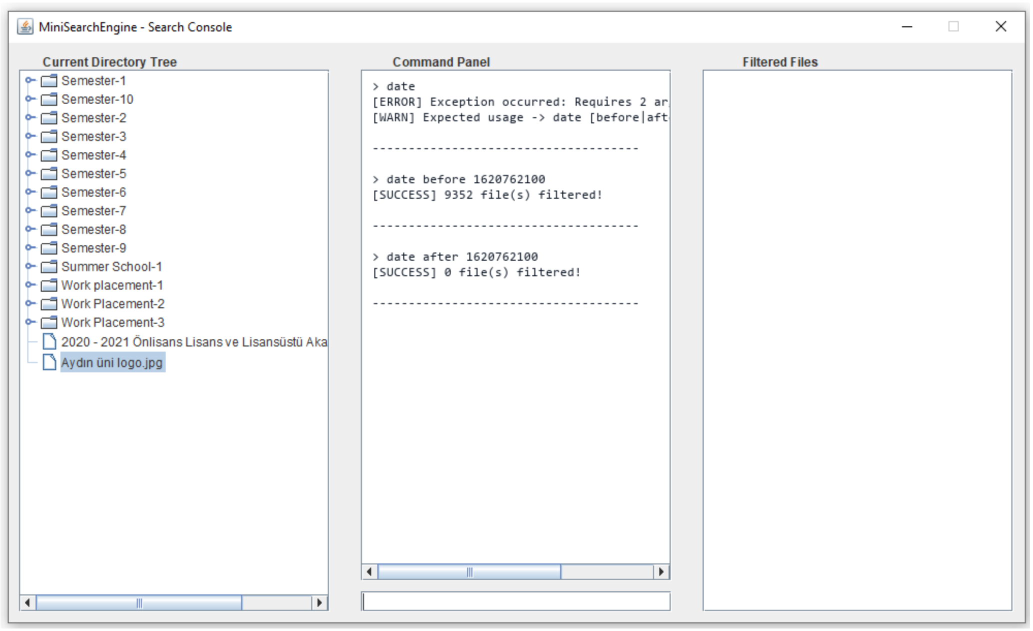
- **date before 'Unix-Time'.** It shows whatever file was before the entered date and time.



```
> date
[ERROR] Exception occurred: Requires 2 ar
[WARN] Expected usage -> date [before|aft

------------------------------------

> date before 1620762100
[SUCCESS] 9352 file(s) filtered!

------------------------------------
```

As we can see, every available file is listed; and the user received a success message with the number of files information.

- **date after 'Unix-Time'.** It shows whatever file was after the entered date and time.



```
> date
[ERROR] Exception occurred: Requires 2 ar
[WARN] Expected usage -> date [before|aft

------------------------------------

> date before 1620762100
[SUCCESS] 9352 file(s) filtered!

------------------------------------

> date after 1620762100
[SUCCESS] 0 file(s) filtered!

------------------------------------
```

As we can see, nothing is listed; and the user received a success message with the number of files information.

➢ **"quit" command.**

When the user types the “quit” command, the mini search engine application will be closed.

5.4 ISSUES AND CHALLENGES

I have faced many challenges and received errors while coding; these challenges are:

- Finding the suitable encryption library for the project and Linking the encryption method into the encryption and hashing algorithm library “Sha 256”.
- Implementing what I researched on B-Tree was challenging; it took more time than the scheduled program that I created at first.
- While creating a traverse method on B-Tree, I needed to create a consumer, and finding out the Lambda Function, which is considered as one of many java utilities, was challenging also.
- When the FileManifest abstraction that I created reads the entire page, the memory swells, so I developed a POJO (Plain Old Java Objects) to keep the lightweight form so the memory does not swell; creating the POJO was also challenging for me.
- While creating the command method, I used a Flag because I needed to modify the operation of the commands; since I had problems in execution when the system cannot separate the entered sentence correctly, finding out the existence of Flags was also challenging for me.
- When I decided to create a command line, I faced many errors and problems since I needed to implement a lower-level programming language.
- When I needed to link the given string with the command object that I created, I used Mapping, and some errors have occurred since it was the first time that I am using it.
- Various object creation problems have occurred since the Object-Oriented programming language is used. So the Builder design pattern is used to separate the construction of a complex object from its representation; this part was also challenging.
- Problems have occurred while integrating the JFileChooser API since it is a ready class of swing library in Java; thankfully, the errors were solved.
- Also, other debugging issues took a bit of time to solve.

CHAPTER 6: TOOLS AND COMPONENTS

6 CHAPTER 6: TOOLS AND COMPONENTS

This chapter will illustrate the tools phase of the project. Also, this chapter will handle the software components.

6.1 SOFTWARE DEVELOPMENT TOOLS

Software development tools are a set of computer programs that are used to create, maintain, debug and support other applications and programs. Development tools can be of different types like compilers, code editors, GUI designers, and debuggers. There are several software development tools used in this project:

- **Operating System (OS):** An operating system is a software that manages hardware and software resources and provides services for software programs; it acts as an interface between the user and computer and lets the user communicate with the computer, it is not possible to use any computer device without having an operating system
    - **Microsoft Windows OS:** It is a graphical operating system that provides a way to store files, run the software, and connect to the internet. It also provides the working environment of this project.
    - **Linux/Unix OS**: Linux and Unix are some of the most popular open-source operating systems on the planet because of their longevity, maturity, stability, flexibility, and security features, and this project might be tested in LINUX or UNIX also.
- **Eclipse Desktop Integrated Development Environment (IDE):** It is well known for its large community and wealth of plugins which make the platform endlessly extensible.
- **Google search:** It is a search engine with extra tools such as google translate, google docs, etc.., and it is mainly used in this project as a search engine to reach resources and information.
- **Stack Overflow:** used for learning and sharing knowledge.
- **Github:** It is a code hosting platform for version control; it simplifies the process of working with peoples by making them work together on projects from anywhere; it makes working on files and merging the changes with the master branch of the project easily.
  There are multiple benefits of GitHub since it is the world’s largest software development platform:
    - Provides cloud storage for source code.

- Supports all popular programming languages.
- Streamlines the iteration process.

- **Microsoft Word:** It is a word processor developed by Microsoft and used for documentation.

- **Draw.io and Creately:** They are web application that allows creating diagrams of all kinds.

6.2 COMPONENT DESCRIPTION

In programming, a system is divided into components, and a component is a part of a larger program. A component provides particular or group-related functions; it is important because the code will not work without them because the IDE will not see it.

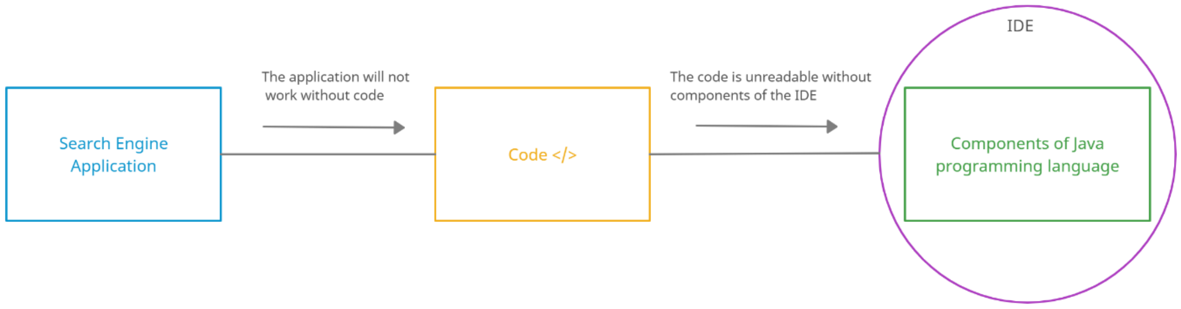

Search Engine Application
The application will not work without code
Code </>
The code is unreadable without components of the IDE
IDE
Components of Java programming language

Figure 18. Component Description

In Object-Oriented programming, a component is a program building block that can be combined with other components. The component description is used for program mining needs to meet the following requests:

- To provide the description of component functions in order to help understand component behavior.

- To provide information on the component organization, classification, and application, to conveniently facilitate establishing relations between user requirements and components.

- To provide a precise description of semantic information regarding component behavior dependency.

- To support traditional keyword-based component searching.

- To provide information on component location and resources.

There are different components used in implementation, and these components are:

- **Client-Based Components:** They allow flexible widget designer implementation that reflects various User Interface (UI) requirements of widget designers. It contains these components for this project:
    - **Abstract Window Toolkit (AWT).**
    - **Swing.**
- **Implementation Components:** Implementation components take place in the context of configuration management on the project, and here are the components that are used in this project:
    - **Java Development Kit (JDK).**
    - **Java Runtime Environment (JRE).**
    - **Java Virtual Machine (JVM):** It contains three components:
        - **ClassLoader Subsystem.**
        - **Runtime Data Area:** This part is divided into five major components:
            - Method Area.
            - Heap Area.
            - Stack Area.
            - PC Registers.
            - Native Method Stacks.
        - **Execution Engine:** It contains the following:
            - Interpreter.
            - JIT Compiler.
            - Garbage Collector.
            - Java Native Interface (JNI).
            - Native Method Library.
    - **Java development tools (JDT):** It contains the following:
        - JDT APT.
        - JDT Core.
        - JDT Debug.
        - JDT Text.
        - JDT UI.
    - **Java programming language compiler (Javac).**

All these components will be explained in detail below.

6.2.1 COMPONENT IDENTIFIER

All Java components require names used for classes, methods, interfaces, and variables; these are called Identifiers. Some rules should be used:

- The first character must be a lowercase letter, an uppercase letter, or an underscore.
- After the first character, all other characters are a combination of digits, lowercase letters, uppercase letters, and underscores.
- Java keywords cannot be used as identifiers.

6.2.1.1 TYPE

This section will describe the type of components that are used in this project.

- **Java Package:** It is the main package that contains other sub-packages hierarchically. These sub-packages of java contain classes.

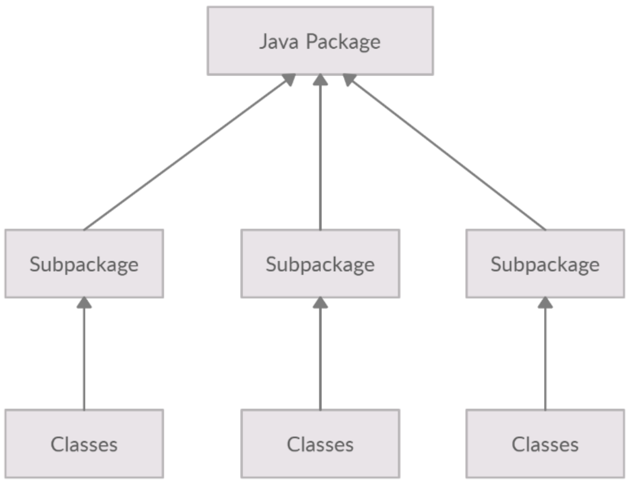


Figure 19. Java Package

- **Abstract Window Toolkit (AWT):** It is considered as a subpackage of the Java package, and it is a built-in package that includes components which are elementary GUI entities and containers that are used to hold components in a specific layout, and these contains classes such as:

  - GUI Component classes “Button and label.”
  - GUI Container classes “Frame and Panel.”
  - Layout managers “FlowLayout and BorderLayout.”
  - Custom graphics classes “Color and Font.”

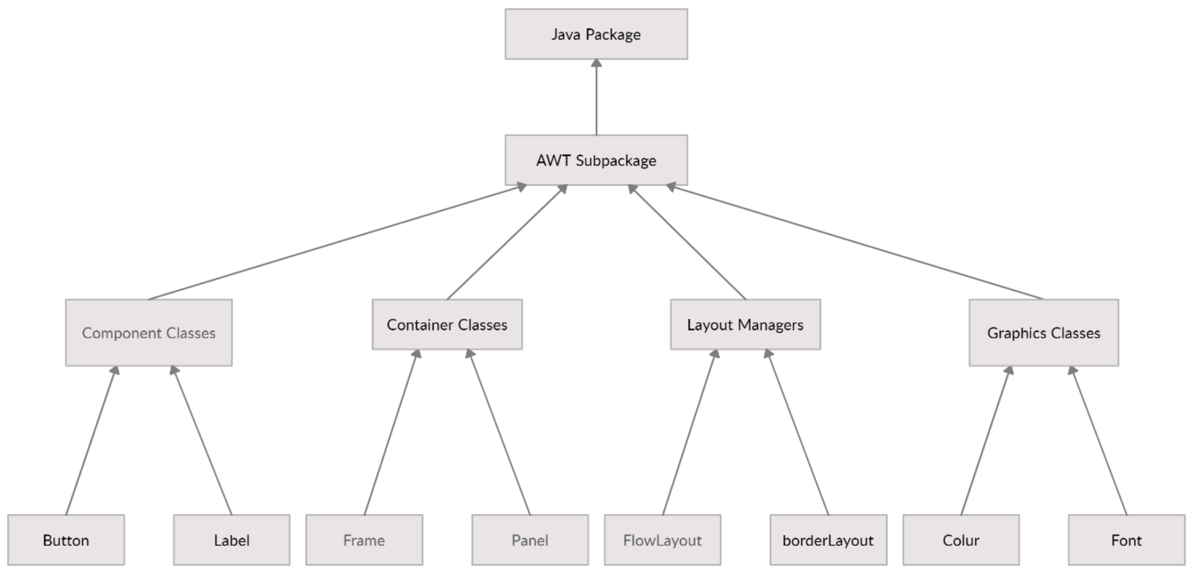


Figure 20. AWT Subpackage

- **Swing:** It is a much more comprehensive graphics library that enhances the AWT, it also stands under the Java package as a subpackage, and in comparison with AWT component classes, it begins with a prefix “J,” such as “JButton and JLabel.”

- **Java Development Kit (JDK):** It is considered a file since it contains a collection of tools and libraries for developing java applications. For windows, the Java Development Kit (JDK) extension directory is located at <Java_Home>\jre\lib\ext.

- **Java Runtime Environment (JRE):** It is considered as a file inside Java Development Kit (JDK); the Java runtime environment (JRE) is the on-disk system that takes the code, combines with necessary libraries, and executes it. The Java runtime environment (JRE) contains libraries and software that java programs need to run.

- **Java Virtual Machine (JVM):** It is a program whose purpose is to execute other programs.

- **Java development tools (JDT):** This package contains utilities for annotation processing in command-line builds and ant scripts.

- **Java programming language compiler (Javac):** It is a compiler for the programming language java.

6.2.1.2 PURPOSE

In this section, the purpose of each component will be explained and handled briefly and carefully.

- **Abstract Window Toolkit (AWT):** It creates components which are elementary GUI entities and containers that are used to hold components by calling subroutines of native platforms, and it contains swing.
- **Swing:** It is a GUI widget toolkit for Java, it is an API for providing a Graphical User Interface (GUI) for Java programs, it is built on top of the Abstract Window Toolkit (AWT), and it is consist of the following items:
    - The default swing components.
    - Some enhanced components are not available within the swing component library.
    - Layout manager that does arrange components **[15]**.
- **Java Development Kit (JDK):** It is a software development kit that contains a collection of tools and libraries for developing Java applications; it converts the source code into a format the Java Runtime Environment (JRE) can execute, and other implementation components will be run inside the Java Development Kit (JDK).
- **Java Runtime Environment (JRE):** It is software the runs other software, it contains the Java class libraries, the Java class loader, and the Java Virtual Machine (JVM), and all these are inside Java Development Kit (JDK). The Java Runtime Environment (JRE) takes the Java code, combines it with necessary libraries, and starts the Java Virtual Machine (JVM) to execute it.
- **Java Virtual Machine (JVM):** It is a running software system that provides a runtime environment that is responsible for executing live Java programs by converting bytecode into machines language, and it contains three components:
    - **ClassLoader Subsystem:** It handles Java’s dynamic class loading functionality; it loads, links, and initializes the class file when it refers to a class for the first time at runtime, not compile time. And these are the steps:
        - Loading.
        - Linking.
        - Initialization.
    - **Runtime Data Area:** This part is divided into five major components:
        - Method Area.
        - Heap Area.
        - Stack Area.
        - PC Registers.

        - Native Method Stacks.

    - **Execution Engine:** The bytecode assigned to the Runtime Data Area will be executed by the execution engine; it contains:
        - Interpreter.
        - JIT Compiler.
        - Garbage Collector.
    - **Java Native Interface (JNI):** Interacts with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.
    - **Native Method Library:** Collection of the Native Libraries, which is required for the Execution Engine.

- **Java Development Tools (JDT):** It provides the tool plug-ins that implement a Java IDE supporting the development of any Java application; it allows Eclipse IDE to be a development environment for itself. And Java Development Tools (JDT) contains:
    - **JDT APT:** Adds annotation processing support to Java projects in Eclipse and provides the following features:
        - Support for running annotation processors written for the APT tool.
        - Contribution of comment-based compilation constructs during incremental compilation.
        - Contribution of problem markers for annotated problems.
    - **JDT Core:** It defines the non-UI infrastructure and has no built-in JDK version dependencies; it includes:
        - An incremental Java builder.
        - A Java Model that provides API for navigating the Java element tree. The Java element tree defines a Java-centric view of a project. It surfaces elements like package fragments, compilation units, binary classes, types, methods, fields **[16]**.
        - Code select support.
        - An indexed-based search infrastructure that is used for searching, code assist, type hierarchy computation, and refactoring **[17]**.
        - Evaluation support.
    - **JDT Debug:** Implements Java debugging support and works with any Java Virtual Machine (JVM), and provides the following debugging features:
        - Launching of a Java Virtual Machine (JVM) in either run or debug mode.
        - Attaching to running Java Virtual Machine (JVM).
        - Notebook pages for interactive Java code evaluation.
        - Dynamic class reload when supported by Java Virtual Machine (JVM).

    - **JDT Text:** It provides the following features:

        - Keyword and syntax coloring.
        - Code assist and code select.
        - Method level edit.
        - Margin annotations for problems and breakpoints.
        - Outliner updating.
        - Import assistance automatically creates import declarations.
        - Code formatting.

    - **JDT UI:** Implements Java-Specific workbench contributions:

        - Package Explorer.
        - Type Hierarchy View.
        - Java Outline View.
        - Wizards for creating java elements.

- **Java programming language compiler (Javac):** It is a tool that reads class and interfaces definitions that are written in the java programming language and compiles them into bytecode class files.

6.2.1.3 FUNCTION, PROCESSING, INTERFACES AND DATA

In this section, there will be a description of what the components do to be engaged; processing will be defined by control and data flow within it by defining how the component will be executed and how it is terminated, Techniques of the process of a component will be shown in diagrams and the logical and physical structure of the components will be shown by relationships between the elements.

- **Abstract Window Toolkit (AWT):** The classes which are included in Abstract Window Toolkit (AWT) will work in a hierarchical way that will be shown below:

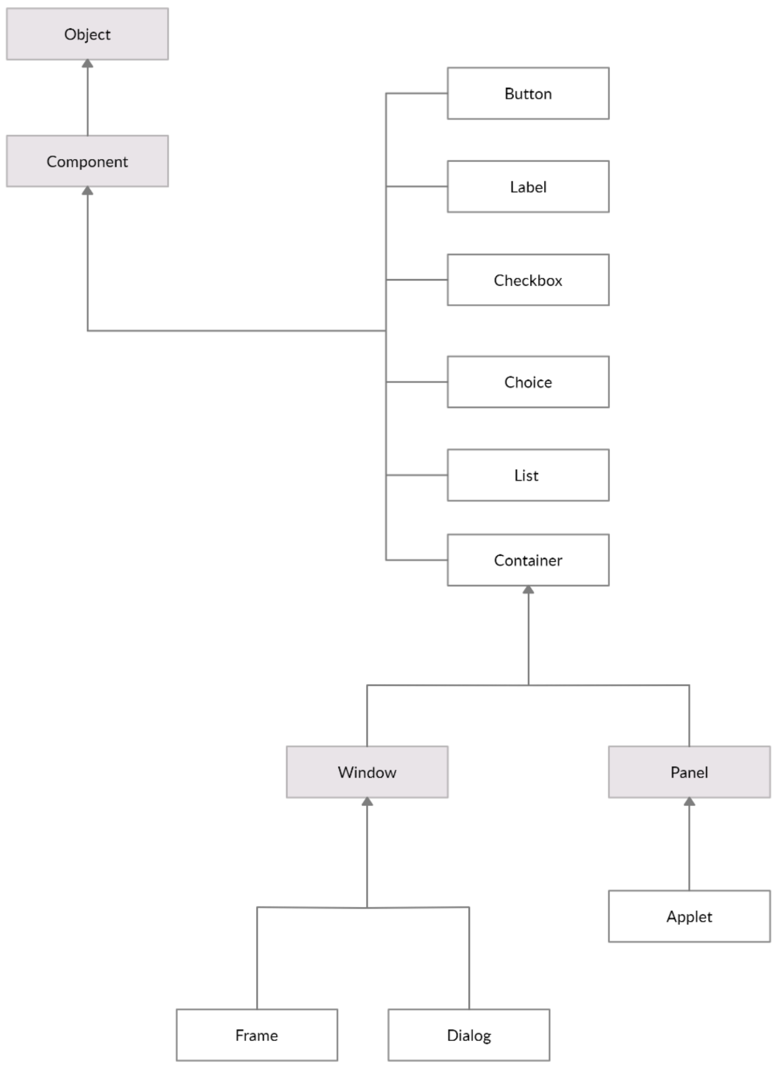


Figure 21. Java AWT Hierarchy

- **Swing:** The classes which are included in Swing will work in a hierarchical way that will be shown below:

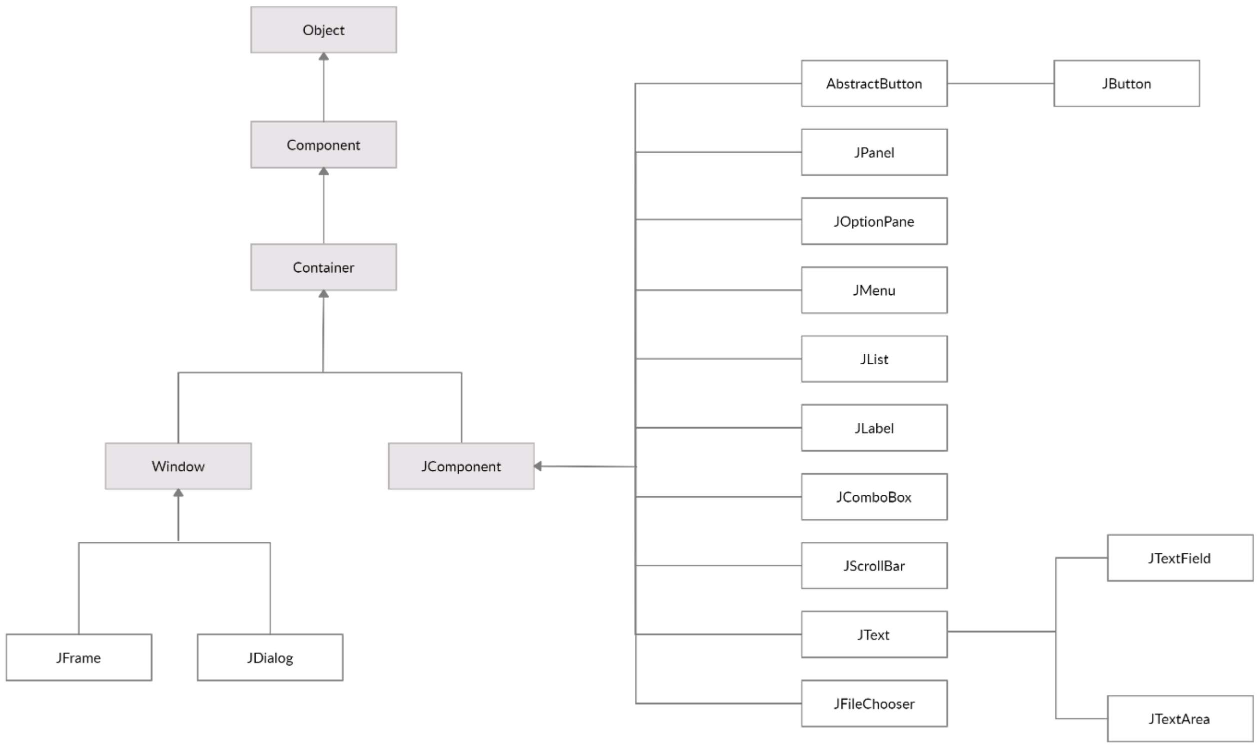


Figure 22. Swing Hierarchy

- **Java Development Kit (JDK):** It contains Java Runtime Environment (JRE) and development resources; it has resources to finish the development of a Java application.

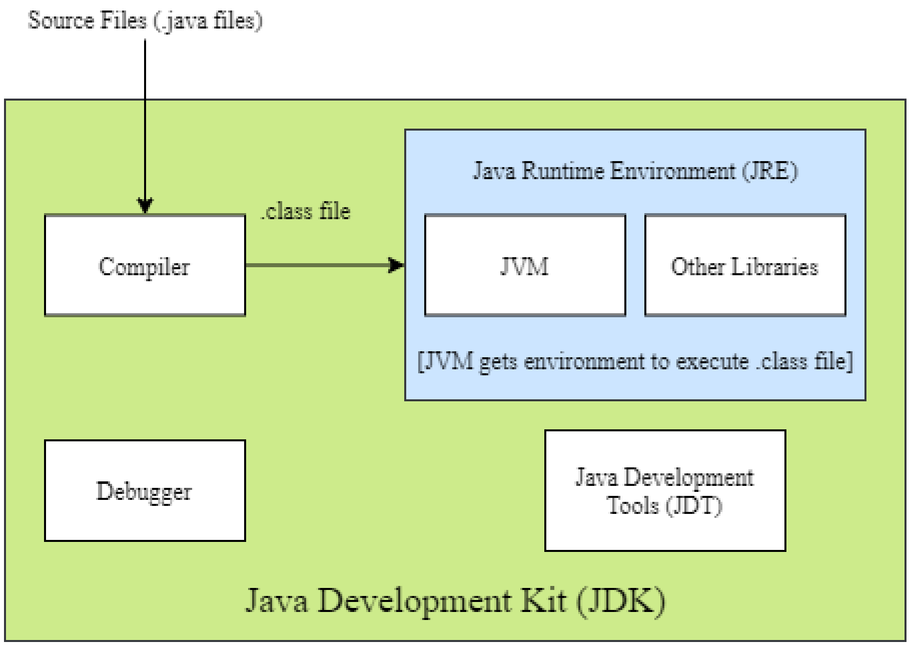

Source Files (.java files)
Compiler
.class file
Java Runtime Environment (JRE)
JVM
Other Libraries
[JVM gets environment to execute .class file]
Debugger
Java Development Tools (JDT)
Java Development Kit (JDK)

Figure 23. JDK Process

- **Java Runtime Environment (JRE):** The Java Runtime Environment (JRE) contains libraries and software that Java programs need to run, the Java class loader is part of the Java Runtime Environment (JRE), this important piece of software loads compiled Java code into memory and connects the code to the appropriate Java class libraries, and the Java Virtual Machine (JVM) is created by Java runtime Environment (JRE).

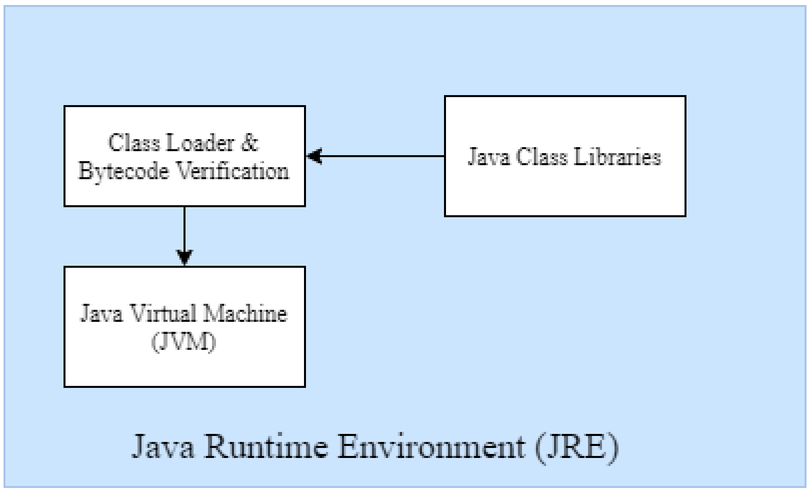


Figure 24. JRE Process

- **Java Virtual Machine (JVM):** The JVM is divided into three main subsystems:
  - **ClassLoader Subsystem:** these are the steps of the ClassLoader Subsystem:
    - **Loading:** Classes will be loaded by this component. BootStrap ClassLoader, Extension ClassLoader, and Application ClassLoader are the three ClassLoaders that will help in achieving it. Types of ClassLoaders are:
      - **Bootstrap ClassLoader:** Responsible for loading classes from the bootstrap classpath, the highest priority will be given to this loader.
      - **Extension Class Loader:** Responsible for loading classes which are inside the ext folder (jre).
      - **Application Class Loader:** Responsible for loading Application Level Classpath.
    - **Linking:** It contains Verify, Prepare and Resolve.
      - **Verify:** Verifies whether the generated bytecode is proper or not.
      - **Prepare:** For all static variables, memory will be allocated and assigned with default values.
      - **Resolve:** All symbolic memory references are replaced with the original references from Method Area.

    - **Initialization:** All static variables will be assigned with the original values, and the static block will be executed.

- **Runtime Data Area:** This part is divided into five major components:
    - **Method Area:** All the class-level data will be stored here, including static variables.
    - **Heap Area:** All the objects and their corresponding instance variables and arrays will be stored here.
    - **Stack Area:** For every thread, a separate runtime stack will be created. The stack frame is divided into three subentities:
        - **Local Variable Array:** Number of local variables that are involved and the corresponding values will be stored in it.
        - **Operand stack:** operand stack acts as runtime workspace to perform the intermediate operation that is required to perform.
        - **Frame data:** All symbols corresponding to the method are stored in it.
    - **PC Registers:** Each thread will have separate PC Registers to hold the address of the currently executing instruction; once the instruction is executed, the PC register will be updated with the next instruction.
    - **Native Method Stacks:** Holds native method information.
- **Execution Engine:** The bytecode that is assigned to the Runtime Data Area will be executed by the execution engine; it contains:
    - **Interpreter:** The interpreter interprets the bytecode faster but executes slowly.
    - **JIT Compiler:** The Execution Engine will be using the help of the interpreter in converting byte code to native code.
        - **Intermediate Code Generator:** Produces intermediate code.
        - **Code Optimizer:** Responsible for optimizing the intermediate code generated.
        - **Target Code Generator:** Responsible for generating machine code or native code.
        - **Profiler:** Responsible for finding hotspots.
    - **Garbage Collector:** Collects and removes unreferenced objects.
- **Java Native Interface (JNI).**
- **Native Method Library.**

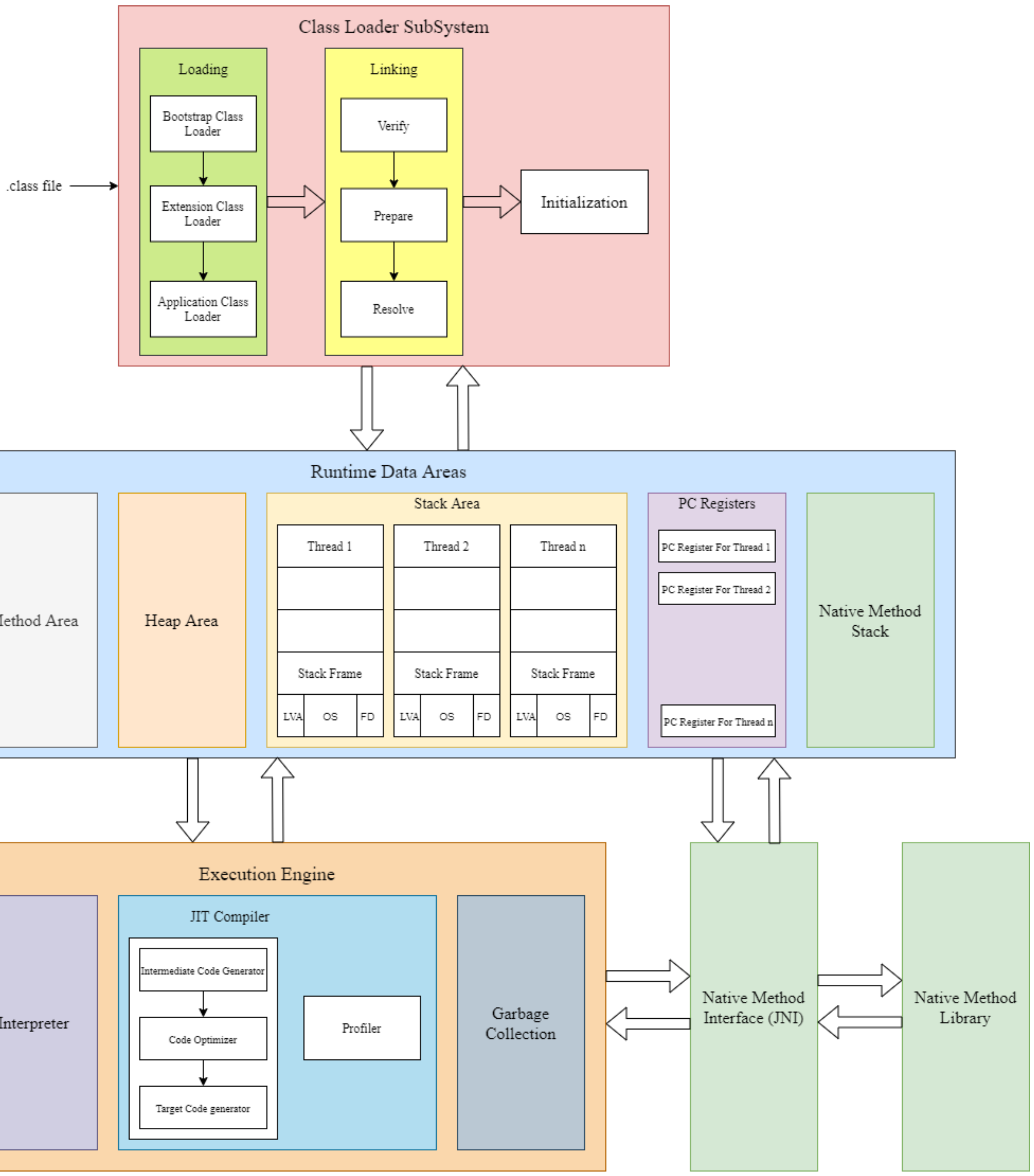


Figure 25. JVM Process

- **Java programming language compiler (Javac):** Java Virtual Machine (JVM) makes Java portable and platform-independent, the machine language is byte code, and it is the same across all operating systems.

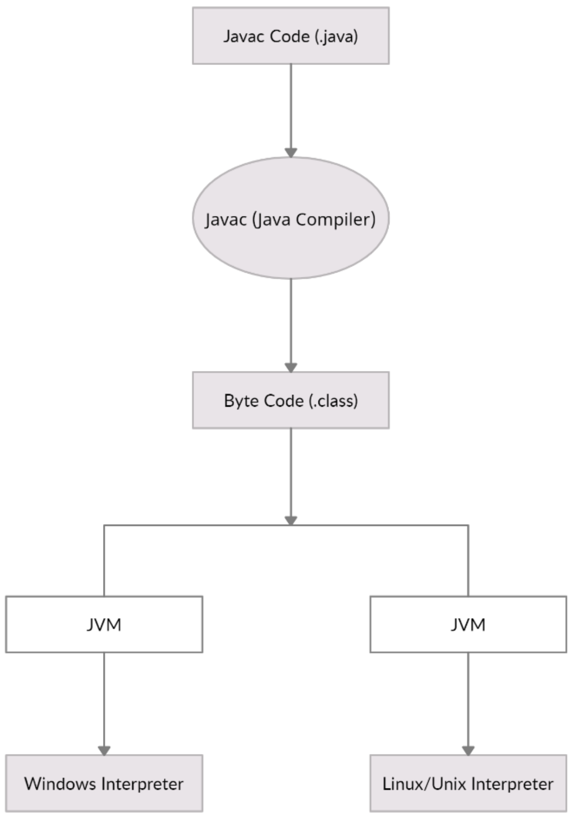


Figure 26. Javac Process

6.2.1.4 SUBORDINARIES

In this section, the modules and classes that are called by each component will be listed.

- **Abstract Window Toolkit (AWT):** Object, Component, Window, Frame, Dialog, and Applet will be called.
- **Swing:** Object, Component, Container, Window, JFrame, JDialog, JComponent, and JFileChooser will be called.
- **Java Development Kit (JDK):** Compiler, Debugger, and Java Runtime Environment (JRE) will be called.
- **Java Runtime Environment (JRE):** Java Virtual Machine, Java Libraries, ClassLoader, and Bytecode Verification will be called.
- **Java Virtual Machine (JVM):** ClassLoader Subsystem, Runtime data Area which contains Method Area, Heap Area, Stack Area, PC Registers and Native Method

Stack, and Execution Engine which includes Interpreter, JIT Compiler, Garbage Collection, Native Method Interface (JNI) and Native Method Library will be called.

- **Java development tools (JDT):** APT, Core, Debug, Text, and UI will be called.

6.2.1.5 DEPENDENCIES AND RESOURCES

In this section, the operations that have taken place, the operations that are excluded, and the components that have to be executed for each component will be discussed. Also, the needs of components from their environment to perform their function will be explained in detail.

- **Abstract Window Toolkit (AWT):** For successful engagement of Abstract Window Toolkit (AWT), at first the Operating System must be working; after that, the Eclipse IDE must be opened, Java Development Kit (JDK) will be activated, which contains the Java Runtime Environment (JRE), the Java Runtime Environment (JRE) will be activated, and it contains Java Class Libraries, one of these libraries is the Abstract Window Toolkit (AWT), all the previous steps must continue to work together in order to make the Abstract Window Toolkit (AWT) work. After that, the classes which are included in Abstract Window Toolkit will work in a hierarchical way; finally, the Java Virtual Machine (JVM) will execute the output.

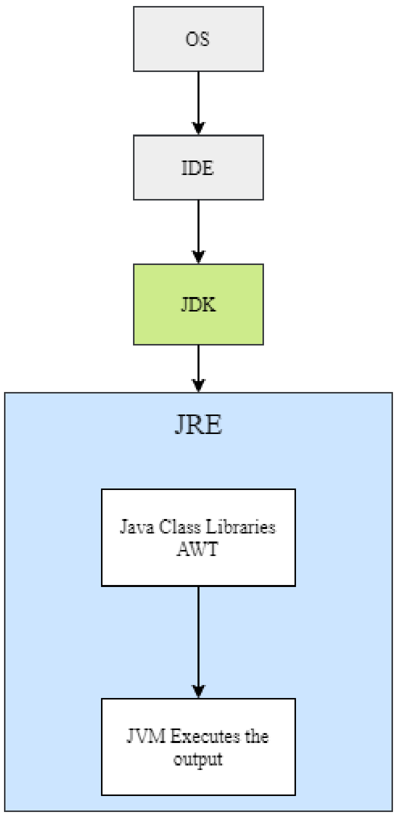


Figure 27. AWT Dependencie

- **Swing:** For successful engagement of Swing, at first the Operating System must be working; after that, the Eclipse IDE must be opened, Java Development Kit (JDK) will be activated, which contains the Java Runtime Environment (JRE), the Java Runtime Environment (JRE) will be activated, and it contains Java Class Libraries, one of these libraries is the Abstract Window Toolkit (AWT), and Swing is built on top of it, all the previous steps must continue to work together in order to make the Swing work. After that, the classes which are included in Swing will work in a hierarchical way that will be shown below; finally, the Java Virtual Machine (JVM) will execute the output.

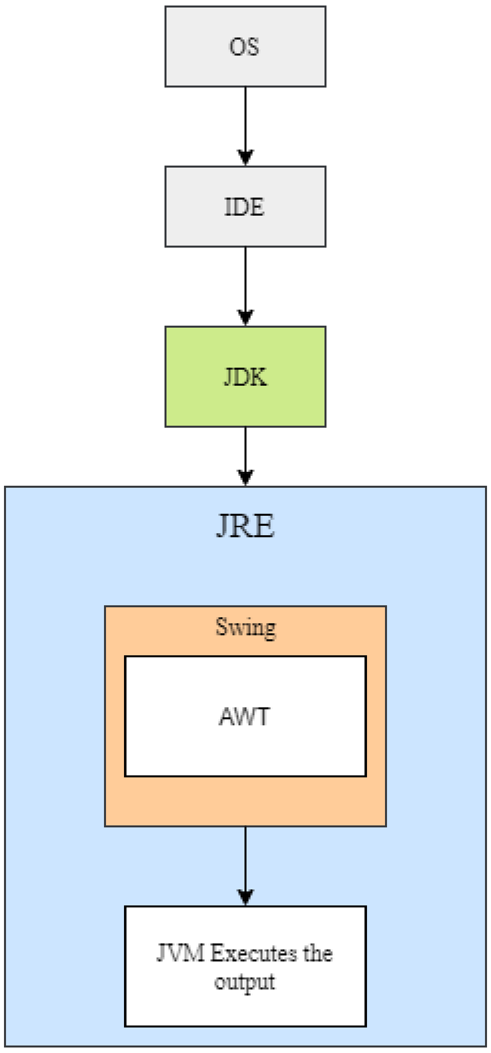


Figure 28. Swing Dependencie

- **Java Development Kit (JDK):** For successful engagement of Java Development Kit (JDK), at first, the Operating System must be working. The Java Development Kit (JDK) contains the Compiler, Debugger, and Java Runtime Environment. Without activation of Java Development Kit (JDK), the contents of it will not work.

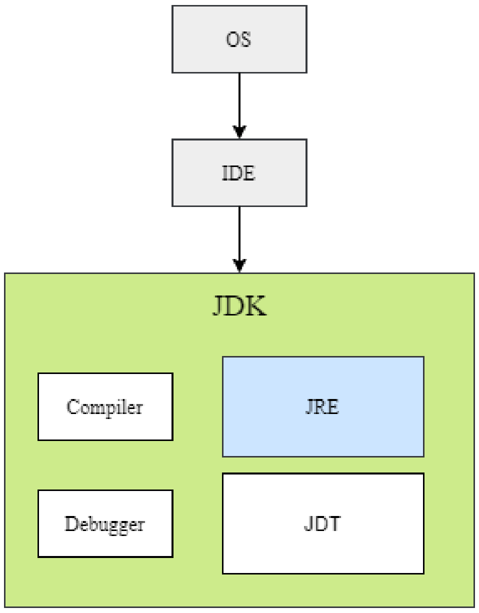


Figure 29. JDK Dependencie

- **Java Runtime Environment (JRE):** For successful engagement of Java Runtime Environment (JRE), at first, the Operating System must be working, and the Java Development Kit (JDK) must be activated. The Java Runtime Environment (JRE) contains the Java Virtual Machine (JVM) and other Java Class libraries, Also a ClassLoader and Bytecode Verification. Without activation of Java Runtime Environment (JRE), the contents of it will not work.

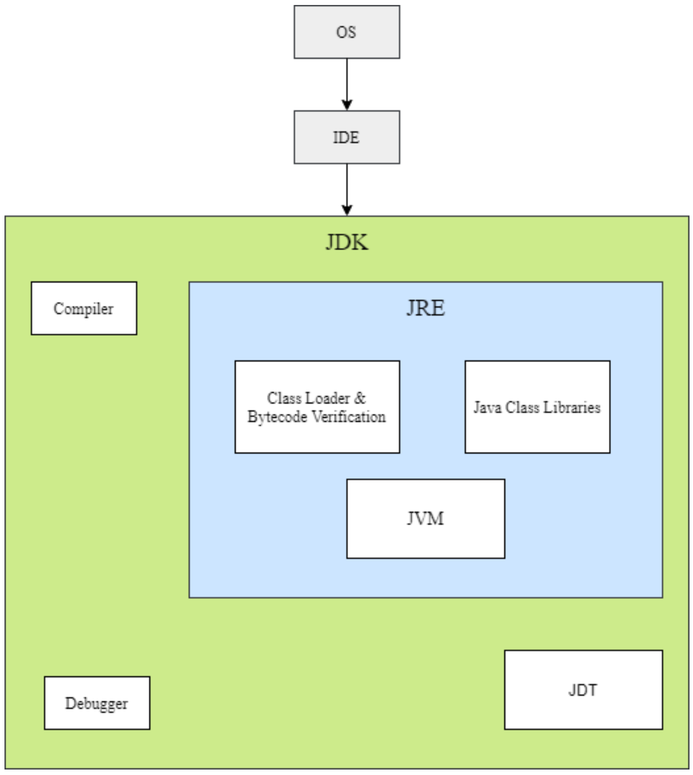


Figure 30. JRE Dependencie

- **Java Virtual Machine (JVM):** For successful engagement of Java Virtual Machine (JVM), at first, the Operating System must be working, and the Java Development Kit (JDK) must be activated as well as The Java Runtime Environment (JRE) must be working also. The Java Virtual Machine (JVM) contains ClassLoader Subsystem, Runtime data Area which contains Method Area, Heap Area, Stack Area, PC Registers and Native Method Stack, and Execution Engine which includes Interpreter, JIT Compiler, Garbage Collection, Native Method Interface (JNI) and Native Method Library. Without activation of Java Virtual Machine (JVM), the contents of it will not work.

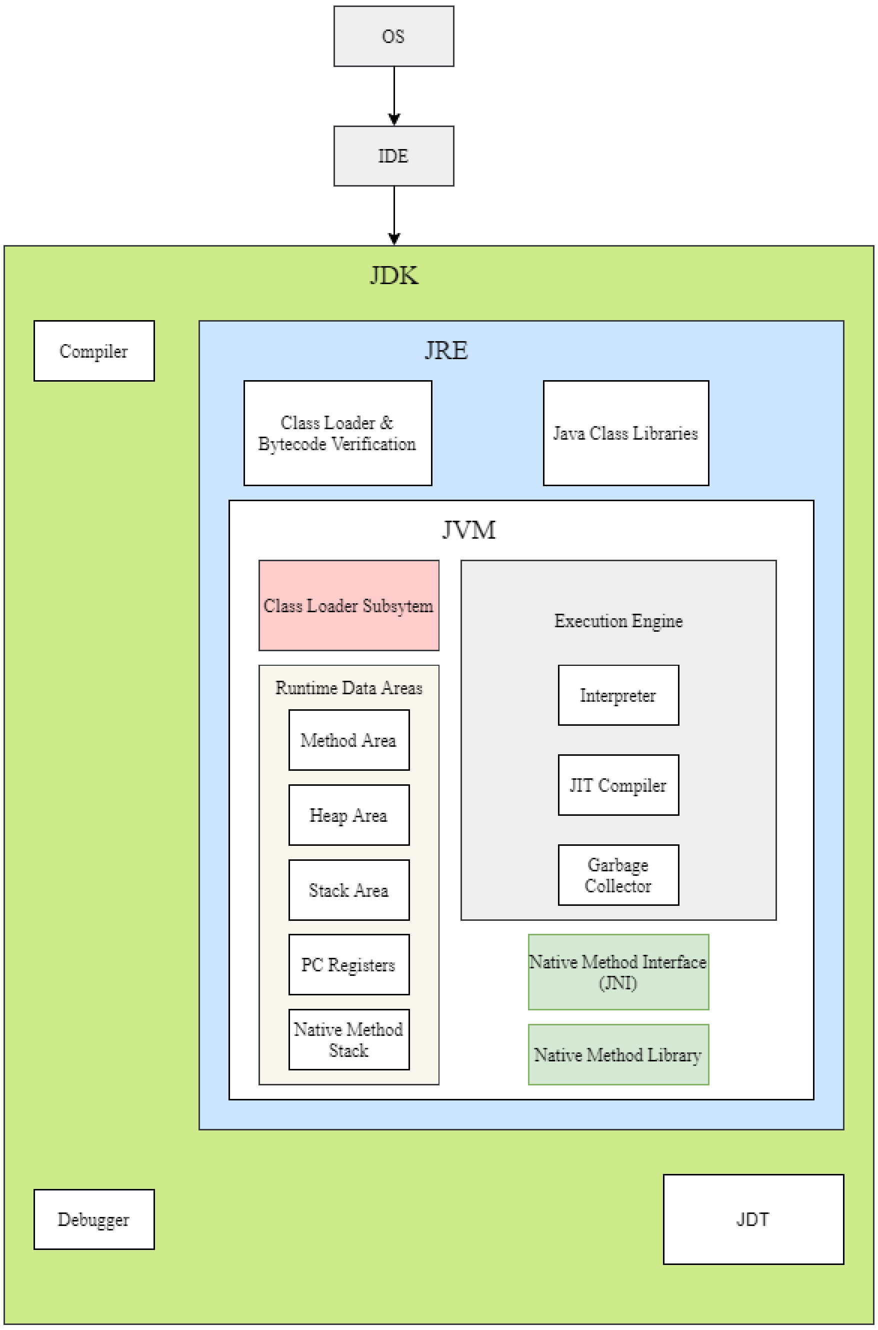


Figure 31. JVM Dependencie

- **Java Development Tools (JDT):** For successful engagement of Java Development Tools (JDT), at first, the Operating System must be working, and the Java Development Kit (JDK) must be activated. The Java Development Tools (JDT) contains APT, Core, Debug, Text, and UI. Without activation of Java Development Tools (JDT), the contents of it will not work.

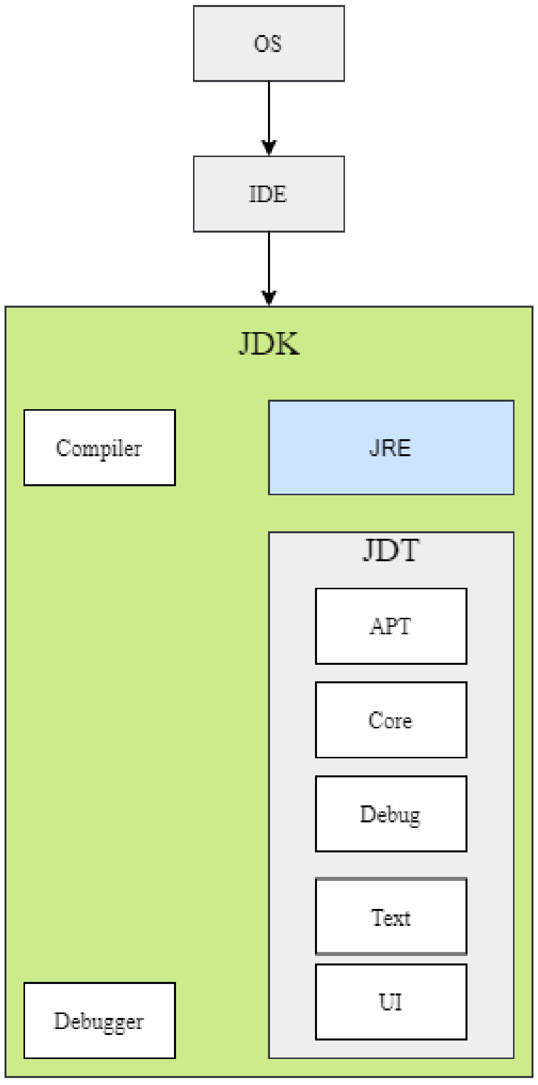


Figure 32. JDT Dependencie

- **Java programming language compiler (Javac):** For successful engagement of Java programming language compiler (Javac), at first, the Operating System must be working, and the Java Development Kit (JDK) must be activated as well as The Java Runtime Environment (JRE) must be working also. The Java programming language compiler (Javac) will be activated by engaging Java Development Tools (JDT), and after that, finally, the Java Virtual Machine (JVM) will execute the output.

CHAPTER 7: TESTING, PROCESS DURATION AND EVALUATION

7 CHAPTER 7: TESTING, PROCESS DURATION AND EVALUATION

This chapter will illustrate the testing and tools phase of the project. In the testing phase, performance testing and functionality are performed. Also, this chapter will handle the software components.

7.1 PERFORMANCE TESTİNG

After structuring the system and building the software application, I had some significant problems with running the application and searching through the files in the right way. I tested the application on several computers, the search results were correct, and all the functions of the application were operating well.

7.2 PROCESS DURATION

The project went right on schedule without any delays. The problem chosen is on an appropriate level of correspondence with ordinary skills from a graduating engineering student.

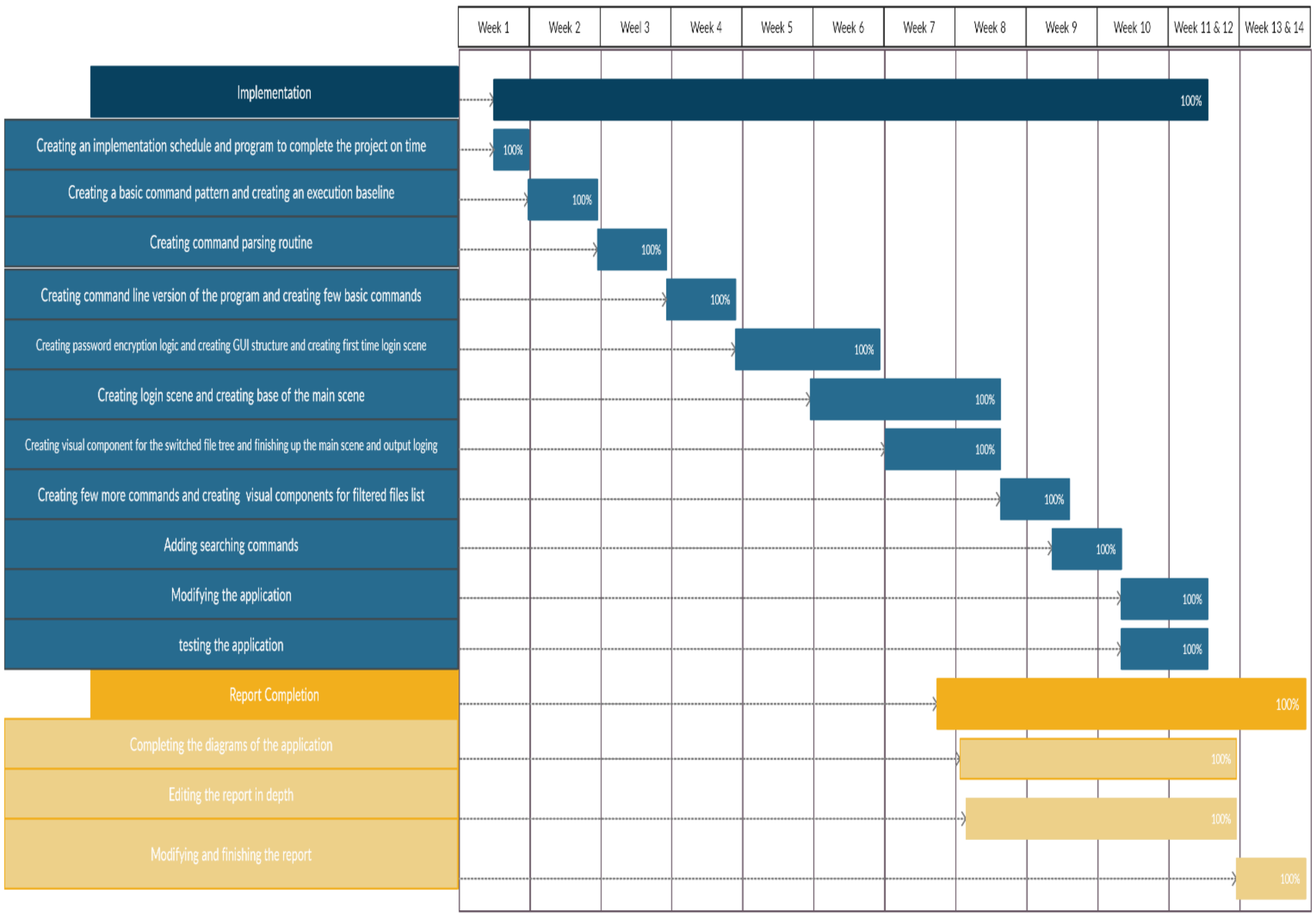


Figure 33. Time Line

As we can see from **Figure 33** above, we can notice that the processes were on time without any delay; these processes are:

Implementation phase: From first week until eleventh week.

- **Week 1:** Creating an implementation schedule and program to complete the project on time.
- **Week 2:** Creating a basic command pattern and creating an execution baseline.
- **Week 3:** Creating command parsing routine.
- **Week 4:** Creating a command-line version of the program and creating few basic commands.
- **Week 5&6:** Creating password encryption logic, creating GUI structure, and creating first-time login scene.
- **Week 6&7&8:** Creating login scene and creating the base of the main scene.
- **Week 7&8:** Creating visual component for the switched file tree and finishing up with the main scene and output logging.
- **Week 8&9:** Creating few more commands and creating visual components for filtered files list.
- **Week 9&10:** Adding searching commands.
- **Week 10&11:** Modifying the application and testing the application.

Report Completion phase: From seventh week until last week.

- **Week 8&9&10&11&12:** Completing the diagrams of the application and editing the report in-depth.
- **Week 13&14:** Finishing and modifying the report.

7.3 EVALUATION

Since the system has the software part, a sample of end-users was consulted to evaluate the application.

7.4 USABILITY EVALUATION

Usability is considered an essential attribute of software quality and is referred to as the efficiency, effectiveness, and satisfaction with which users can perform tasks with a tool. The term is used to describe the quality of a user's experience when interacting with a system, whether hardware or software. A usable system enables users to perform their job effectively and efficiently. Evaluating usability is considered an essential part of the system development process, and there are different methods to professionally support the human factors. Traditionally, the concept of usability has been defined in multiple different ways, basically on one of the following bases:

- **Semantics:** usability is equated to terms such as 'ease of use' or 'user-friendliness, without a formal definition of the properties of the construct.
- **Features:** usability is equated to the presence or absence of certain features in the user interface such as Windows, Icons, Menus, or Pointing devices.

- **Operations:** defined in terms of performance and affective levels manifest by users for a certain task and environmental scenarios **[18]**.

7.5 END-USER EVALUATION

To evaluate the system based on the customers, 16 participants were selected and asked to use the system. After they finished, they were asked to assess the system using a questionnaire designed for that purpose. The questionnaire was divided into two parts. The first part includes general information about the participants including, their gender, age, qualifications. The second part was used to evaluate the system and evaluate the participant's opinion about specific considerations about the system.

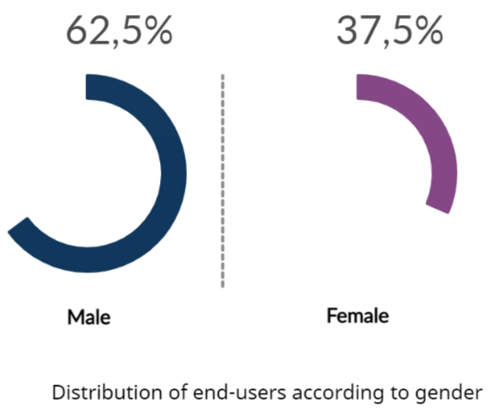


Figure 34. Distribution Of Gender

As shown in **Figure 34**, the male participants were about 62,5%, while the female participants were about 37,5%.

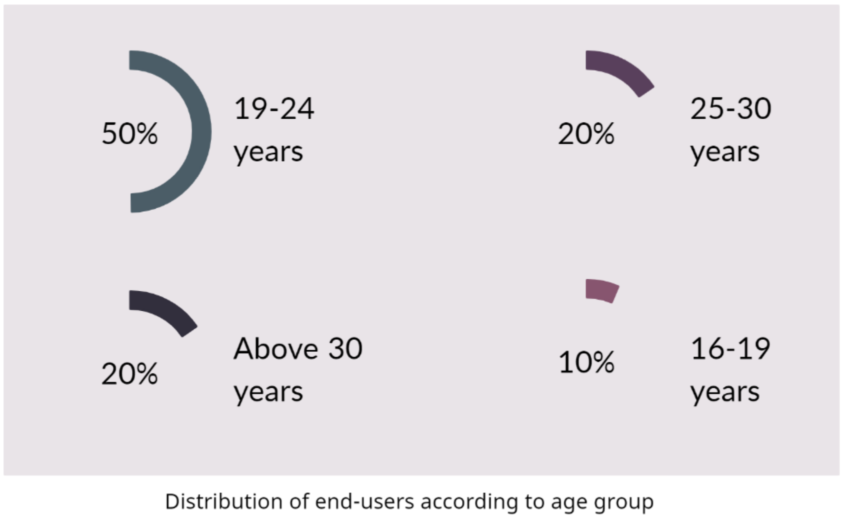


Figure 35. Distribution Of Age

As shown in **Figure 35**, 50% of the participants were between 19-24 years old, 20% between 25-30, 20% above 30, and 10% between 16-19.

As shown in **Figure 36**, Regarding the qualifications of the end-users participated in the evaluation process, 50% of them were university students, 20% of them were master students, 20% of them Hold bachelor degree, and about 10% of them were high school students. This diversity in the qualification of the users makes the evaluation process more consistent and reliable.

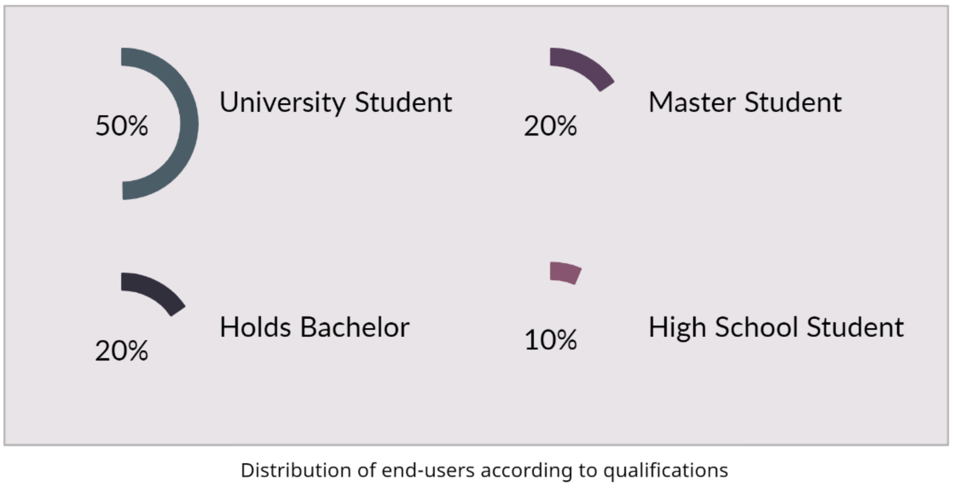


Figure 36. Distribution Of Qualification

The end-users were also chosen according to their experience in using software applications. This was considered important criteria in the selection of the users because the users of the system must be familiar with this technology.

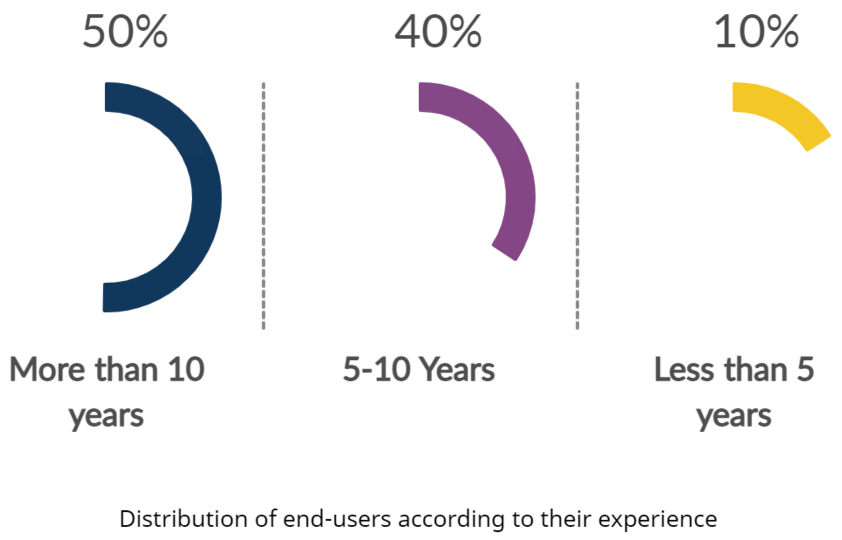


Figure 37. Distribution Of Experience

As shown in **Figure 37**, half of the participants uses software applications since 10 years. The second half of them are distributed among those whose experience between 5-10 years and less than 5 years.

Regarding the importance of the system, participants were asked to evaluate the importance of the system from their point of view and after using the system.

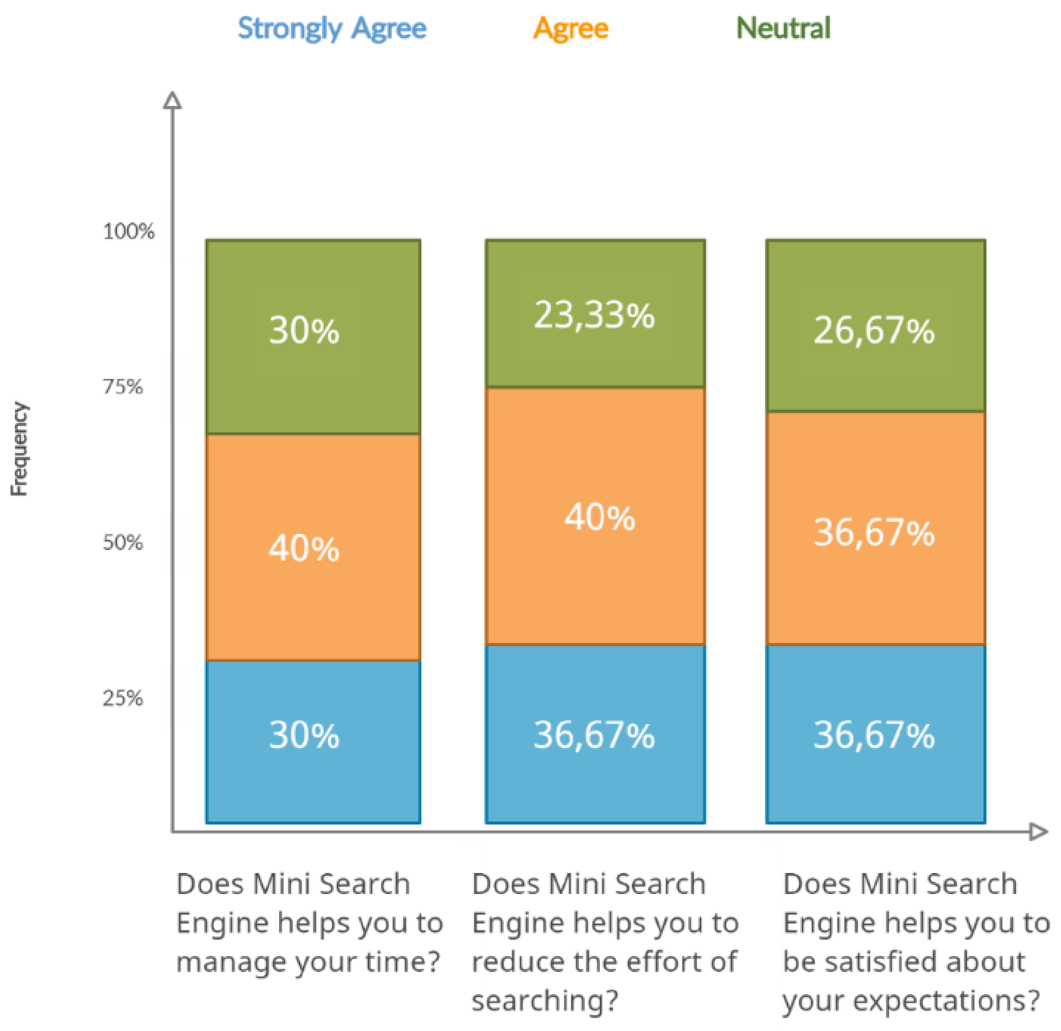


Figure 38. Importance Of The System

As we can see from **Figure 38** above, about 30% of participants strongly agree that the system would help them to save their effort and reduces time; 40% of them agree, and the rest were natural for this point. On the other hand, 36,67% of participants strongly agreed that the system would help them reduce the effort, and 40% of them agree, and the rest were neutral for this point. Also, 36,67% strongly agreed that the system would make them more satisfied with their expectations, and 36,67 of them agree, and the rest were neutral.

In relation to the easiness of the system.

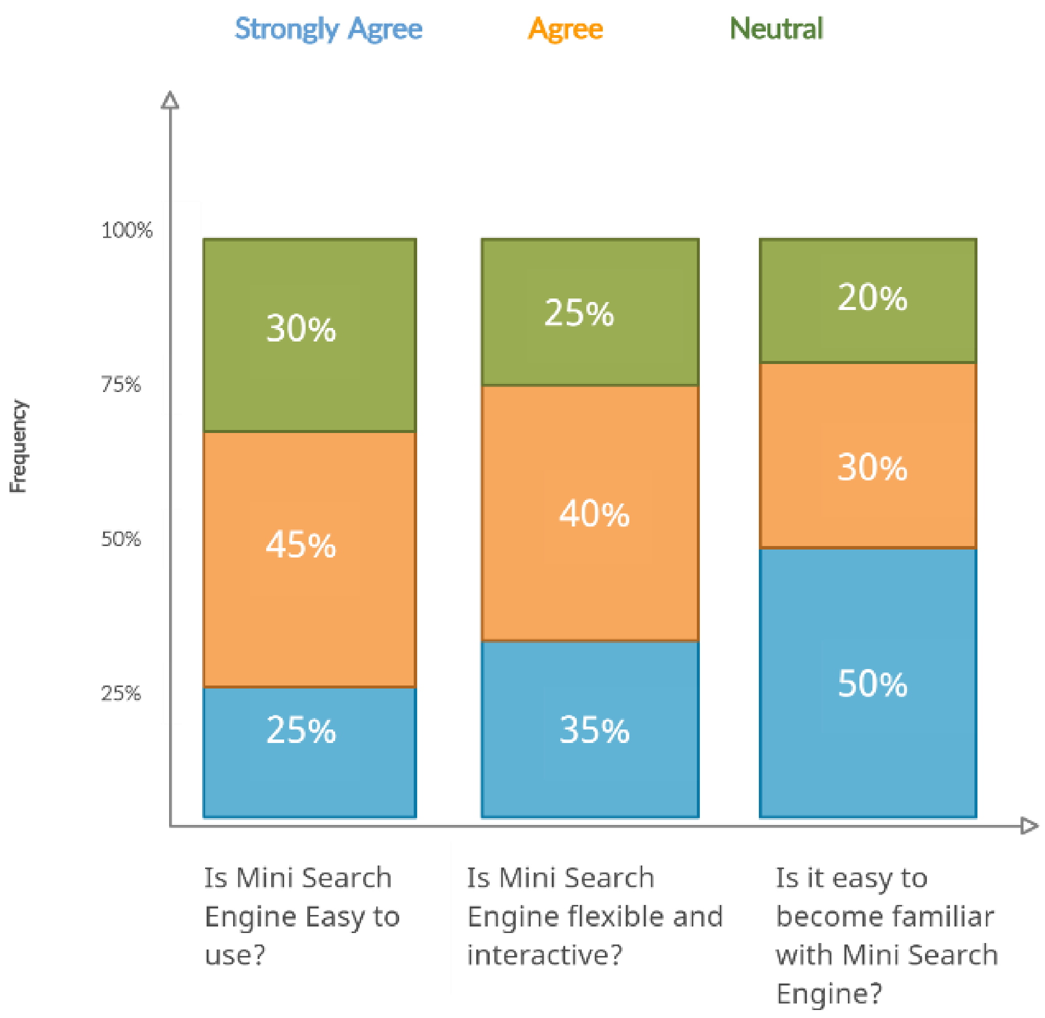


Figure 39. Easiness Of The System

As we can see from **Figure 39** above, 25% of the users strongly agreed on the point that the system is easy to use, 45% of them agree that the system is easy to use, and the rest were neutral about this issue. 35% of the users strongly agreed that the system is flexible and interactive, and 40% of the users agreed, while 25% were neutral. Further, 50% of the users strongly agreed that it is easy to become familiar with the system, 30% agreed, and the rest were neutral.

7.6 MAINTENANCE

At this stage, maintenance, improvement, and updating of the system will occur, which will involve all the errors that are met through the usage of the developed system, resulting in an updated and improved system.

7.7 CONCULSION

In recent years, with the pace of technological development, people have become more and more demanding in terms of quality of life. Today we find search engines in most operating systems, applications, businesses, and the internet; they are the central vein of research and learning.

Search engines are helpful in many ways. For instance, it reduces the time for businesses because they need to be fast and efficient while searching any file in their structure, it reduces the time for students because they don't have time to waste any second in the busy pace of life; also, it reduces the time for irregular users since they forgot where they saved that specific file. Finally, it reduces the time for any user that wants a fast search application for the files.

As technology improves, there will be new methods to use search engines to bring new hopes and new potentials to humanity.

7.8 FUTURE WORK

There is a clear vision for the future of this project; these visions are:

- Faster, more useful, and more comprehensive commands will be added since the application commands are extensible.
- The Project Objectives will be expanded to include much more than surveillance and exploration; it will realize some intelligent functions that can help more people.
- The application will be more flexible and user-friendly.
- The application's UI will be improved to make the application more usable.

REFERENCES

REFERENCES

1. Oracle Java Documentation. The Java Tutorials. https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html

2. Dasgupta, Anirban; Ghosh, Arpita; Kumar, Ravi; Olston, Christopher; Pandey, Sandeep and Tomkins, Andrew. The Discoverability of the Web. https://www.arpitaghosh.com/papers/discoverability.pdf

3. Kahn, David (1967). The Codebreakers. ISBN 978-0-684-83130-5. Retrieved from https://en.wikipedia.org/wiki/Cryptography

4. Buttcher, Stefan. (2007). Multi-User File System Search.Waterloo, Ontario, Canada. Page 3, 4. http://www.stefan-buettcher.de/papers/buettcher_phd_thesis.pdf

5. B.Cole. (2005). Search engines tackle the desktop. IEEE, Flagstaff, Arizona. https://ieeexplore.ieee.org/abstract/document/1413110/authors#authors

6. O. Gorter. (2004). Database file system - an alternative to hierarchy based file systems. Master's thesis, University of Twente.

7. J. Nielsen. (1996). The death of file systems.

8. Foo, Schubert; Hendry, Douglas. Desktop Search Engine Visualization and Evaluation. Division of Information Studies, School of Communication and Information Nanyang Technological University, Singapore 637718.

9. Huang, Jun; Wu, ShunXiang. The Research of Fast File Search Engine Based on NTFS and Its Application in Fast Electronic Document Destruction. Department of Automation, Xiamen University, Xiamen, China, 361005.

10. Computer Weekly (2002). Write once run anywhere?. https://www.computerweekly.com/feature/Write-once-run-anywhere
    Retrieved from https://en.wikipedia.org/wiki/Java_(programming_language)

11. Oracle (2013). 1.2 Design Goals of the Java Programming Language. https://web.archive.org/web/20130123204103/http://www.oracle.com/technetwork/java/intro-141325.html
    Retrieved from https://en.wikipedia.org/wiki/Java_(programming_language)

12. (September 12, 1997). Java Code Conventions. https://introcs.cs.princeton.edu/java/11style/codeconventions-150003.pdf

13. (September 12, 1997). Java Code Conventions. https://introcs.cs.princeton.edu/java/11style/codeconventions-150003.pdf

14. Clements, Paul C. (1996). A survey of architecture description languages. Proceedings of the 8th international workshop on software specification and design. IEEE Computer Society. https://en.wikipedia.org/wiki/Systems_architecture#cite_note-2

15. Dietz, Jab. (2006). Enterprise Ontology – Theory and Mehodology. Springer-Verlag Berlin Heidelberg.

16. Eclipse Foundation. Eclipse Java development tools (JDT) Overview. https://www.eclipse.org/jdt/overview.php

17. Eclipse Foundation. Eclipse Java development tools (JDT) Overview. https://www.eclipse.org/jdt/overview.php

18. Mohammed Rajab Al-Wakeel; Waseem Bassam Ayesh. Exploration Robot Controlled by an Android Application(ERCAA). Page 35. http://dspace.up.edu.ps/jspui/bitstream/123456789/197/1/Exploration%20Robot%20Controlled%20by%20an%20Android%20Application.pdf