

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Santhibastawad Road, Machhe Belagavi - 590018, Karnataka, India



FILE STRUCTURES LABORATORY WITH MINI PROJECT (18ISL67) REPORT

ON

“FIR MANAGEMENT SYSTEM”

Submitted in the partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING IN INFORMATION SCIENCE AND ENGINEERING

For the Academic Year 2022-2023

Submitted by

Namratha Prakash
Neha Subrahmanya Hegde

1JS20IS054
1JS20IS056

Under the Guidance of

Mrs. Sahana V
Assistant Professor
Dept. of ISE, JSSATEB



2022-2023

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING
JSS ACADEMY OF TECHNICAL EDUCATION

JSS Campus, Dr. Vishnuvardhan Road, Bengaluru-560060

JSS MAHAVIDYAPEETHA, MYSURU
JSS ACADEMY OF TECHNICAL EDUCATION
JSS Campus, Dr.Vishnuvardhan Road, Bengaluru-560060

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING



CERTIFICATE

This is to certify that FILE STRUCTURES LABORATORY WITH MINI PROJECT (18ISL67) Report entitled “**FIR Management System**” is a Bonafide work carried out by **Namratha Prakash [1JS20IS054], Neha Subrahmanya Hegde [1JS20IS056]** in partial fulfillment for the award of degree of Bachelor of Engineering in Information Science and Engineering of Visvesvaraya Technological University Belagavi during the year 2022- 2023.

Signature of the Guide
Mrs. Sahana V
Assistant Professor
Dept. of ISE, JSSATEB

Signature of the HOD
Dr. Rekha PM
Professor & HOD
Dept. of ISE, JSSATEB

External Viva

Name of the Examiners

Signature and Date

1. .

2.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	1
ABSTRACT	2
CHAPTER 1	3
INTRODUCTION	3
1.1 OVERVIEW	3
1.2 PROBLEM STATEMENT	3
1.3 ORGANIZATION OF THE REPORT	4
CHAPTER 2	5
INTRODUCTION TO FILE STRUCTURES	5
2.1 INTRODUCTION	5
2.2 HISTORY	6
2.3 TYPES OF FILES	7
2.4 OPERATIONS ON FILE STRUCTURES	8
2.5 ADVANTAGES OF FILE STRUCTURES	10
2.6 DISADVANTAGES OF FILE STRUCTURES	11
CHAPTER 3	12
INTRODUCTION TO INDEXING	12
3.1 TYPES OF INDEXING	12
3.2 PURPOSE OF INDEXING	14
3.3 ADVANTAGES OF INDEXING	15
3.4 DISADVANTAGES OF INDEXING	17
CHAPTER 4	18
INTRODUCTION TO HASHING	18
4.1 SHA-1 HASHING ALGORITHM	18
CHAPTER 5	20
SYSTEM SPECIFICATION	20
5.1 HARDWARE SPECIFICATION	20
5.2 SOFTWARE SPECIFICATION	20
CHAPTER 6	26
SYSTEM FUNCTIONALITIES	26
CHAPTER 7	28
SYSTEM IMPLEMENTATION	28
7.1 PSEUDO CODES	28

CHAPTER 8	39
SYSTEM TESTING	39
8.1 UNIT TESTING	39
8.2 INTEGRATION TESTING	39
8.3 SYSTEM TESTING	42
CHAPTER 9	44
RESULTS AND DISCUSSIONS	44
CHAPTER 10	55
CONCLUSION AND FUTURE ENHANCEMENTS	55
10.1 CONCLUSION	55
10.2 FUTURE ENHANCEMENTS	56
BIBLIOGRAPHY	57
BOOK REFERENCES	57
WEB REFERENCES	57

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible. So with gratitude, we acknowledge all those whose guidance and encouragement crowned our efforts with success.

First and foremost, we would like to thank his **Holiness Jagadguru Sri Shivarathri Deshikendra Mahaswamiji** for his divine blessings, without which the work wouldn't have been possible.

It's my pleasure in thanking our principal **Dr. Bhimasen Soragaon**, Principal, JSSATE, Bangalore for providing an opportunity to carry out the Project Work as a part of our curriculum in the partial fulfilment of the degree course.

We express our sincere gratitude for our beloved Head of the department, **Dr. Rekha P.M.**, for her co-operation and encouragement at all the moments of our approach.

It is our pleasant duty to place on record our deepest sense of gratitude to our respected guide, **Mrs. Sahana V**, Assistant Professor, Dept. of Information Science and Engineering for the constant encouragement, valuable help and assistance in every possible way.

Namratha Prakash[1JS20IS054]

Neha Subrahmanya Hegde[1JS20IS056]

ABSTRACT

The FIR Management System is a file-based mini-project designed to simplify the process of managing FIR records. It helps all Police stations across the country and specifically looks into the subject of FIR Records Management.

FIR Management System offers essential functionalities for adding, deleting, modifying, searching and displaying the FIR record information. The system efficiently stores the FIR data in files, allowing users to easily add new records by providing a FIR ID. Jailor can log in as a user and can add the details into FIR record like name of accused and victim, date, time, status and register a FIR by providing FIR ID to the record as well as modify the details of existing records, and can view all the records in the file, ensuring accurate and up-to-date information.

Anyone who knows about the commission of a cognizable offence can file an FIR. It is not necessary that only the victim of the crime should file an FIR. A police officer who comes to know about a cognizable offence can file an FIR himself/herself.

It is proposed to centralize Information Management in Crime for the purposes of fast and efficient sharing of important information across all Police Stations. The project has been planned to be having the view of distributed architecture.

CHAPTER 1

INTRODUCTION

OVERVIEW

The project titled FIR Management System is a file-based system that provides storage, manipulating, extraction of records and to view information. First Information Report (FIR) is a written document prepared by the police when they receive information about the commission of a cognizable offence. It is a report of information that reaches the police first in point of time and that is why it is called the First Information Report. It is generally a complaint lodged with the police by the victim of a cognizable offence or by someone on his/her behalf.

An FIR is a very important document as it sets the process of criminal justice in motion. It is only after the FIR is registered in the police station that the police takes up investigation of the case.

The records include incident and accident details, status, accused and victim details and other related records. FIR management System does not address the functions such as court proceedings finances, payroll and law enforcement personnel records etc.

PROBLEM STATEMENT

Develop a FIR Management System using file structures to address the challenges of manual method. The system should allow the user to register, delete, modify, search, and display the FIR details efficiently.

The entire manual process is time consuming as the complainant has to physically go to the police station numerous times. The same also consumes a whole lot of money and energy. Previously, this has been a paper-based process, and paper records were easily manipulated or lost.

By implementing this system, the aim is to design a well-organized FIR management system that helps to maintain and access the records effectively and providing a user-friendly software solution that enables users to register new complaint, delete records, modify, search for specific record, and display FIR details.

ORGANIZATION OF THE REPORT

Chapter 1 provides the Overview of the project and explains the Problem statement. In **Chapter 2**, we discuss the File Structures concepts, types, Operations, Advantages and Disadvantages. In **Chapter 3**, we discuss about Indexing and it's types, Purpose, Advantages and Disadvantages of Indexing. In **Chapter 4**, we brief about Hashing and discuss the SHA-1 Algorithm. In **Chapter 5**, we discuss the software and hardware requirements to design and implement the above concepts in our project. **Chapter 6**, gives the idea of the system design and Functionalities. **Chapter 7** gives a clear picture about the project and its actual implementation. In **Chapter 8**, we discuss the system testing. **Chapter 9** discusses the results and discussions of the program. **Chapter 10** concludes by giving the direction for future enhancements.

CHAPTER 2

INTRODUCTION TO FILE STRUCTURES

INTRODUCTION

File structures, also known as data structures for files, are methods or techniques used to organize and manage data stored in computer files. They define how data is stored, accessed, and modified within a file, aiming to optimize data storage efficiency and retrieval performance.

In computing, files are used to store and organize data on various storage media such as hard drives, solid-state drives (SSDs), or other types of storage devices. A file structure provides a logical representation of the data within a file, allowing for efficient storage, retrieval, and manipulation of information.

File structures encompass different techniques and data organization methods, including:

Sequential Files: In a sequential file structure, data is stored in a sequential manner, one after another. Each record in the file contains a fixed number of fields, and the records are accessed sequentially from the beginning to the end. Sequential files are simple and suitable for applications that require frequent access to all records in a specific order.

Indexed Files: Indexed file structures introduce an index, a separate data structure that allows direct access to specific records within a file. The index contains key-value pairs, where the key is typically a field or a combination of fields from the records. The index facilitates quick searches and retrieval of records based on the specified key.

Hashed Files: Hashed file structures use a hash function to map keys to specific locations within the file. The hash function generates a hash value based on the key, which is used to determine the storage location. Hashed files provide rapid access to records but may suffer from collisions (multiple records mapped to the same location) if not properly handled.

B-Trees: B-Trees are balanced tree structures that allow efficient insertion, deletion, and searching operations in large files. B-Trees are particularly useful for managing large amounts of data, as they maintain balance and keep the tree height relatively small, resulting in fast access times.

Directories: Directories are file structures used to organize and manage files within a file system. They provide a hierarchical structure, allowing files to be organized into directories and subdirectories for better organization and management. Directories typically maintain metadata about the files, such as names, sizes, and permissions.

File structures play a crucial role in data management, providing efficient and optimized methods for storing and retrieving data from files. The choice of file structure depends on the specific requirements of an application, including the type and size of data, access patterns, and desired performance character.

HISTORY

Early work with files presumed that files were on tape, since most files were. Access was sequential and the cost of access grew in direct proportion to the size of the file. The indexes made it possible to keep a list of keys and pointers in a smaller file that could be searched more quickly. With the key and pointer, the user had direct access to the large, primary file.

Unfortunately, simple indexes had some of the same sequential flavour as the data files, and as the indexes grew, they too became difficult to manage, especially for dynamic files in which the set of keys changes. Then in the early 1960's, the idea of applying tree structures emerged.

Unfortunately, trees can grow very unevenly as records are added and deleted.

In 1963 researchers developed the tree, an elegant, self-adjusting binary tree structure, called AVL tree, for data in memory. The problem was that even with a balanced binary tree, dozens of accesses were required to find a record in even moderate sized files.

It took nearly ten more years of design work before a solution emerged in form of the B-tree. AVL trees grow from top down as records are added, B-trees grow from the bottom up.

B-trees provided excellent access performance, but there was a cost: no longer could a file be accessed sequentially with efficiency. Fortunately, this problem was solved almost immediately by adding a linked list structure at the bottom level of the B-tree. The combination of a B-tree and a sequential linked list is called a B+ tree.

Being able to retrieve information with just three or four accesses is pretty good. But the goal of being able to get what we want with a single request was satisfied by hashing. From early on, hashed indexes were used to provide fast access to file. After the developed of B-trees, researchers turned to work on systems for extendible, dynamic hashing that could retrieve information with one or, at most, two disk accesses no matter how big the file became.

TYPES OF FILES

As we know that computers are used for storing the information for a permanent time or the files are used for storing the Data of the users for a long time period. And the files can contain any type of information which means that they can store text, any images or pictures or data in any format. So that there must be some mechanism which are used for storing the information.

Accessing the information and also performing some operations on the files. When we store a file in the system, then we have to specify the name and the type of file. The name of file will be any valid name and type means the application with which the file is linked. When we say that every file has some type, it means that every file belongs to a special type of application software. When we provide a name to a file, we also specify the extension of the file because the system will retrieve the contents of that file into application software.

1. **Ordinary file or Simple file:** Ordinary file may belong to any type of application. For example, notepad, paint, C program, music[mpeg] etc. All the files that are created by a user are ordinary files. Ordinary files are used for storing information about the user programs. With the help of ordinary files, we can store the information which contain text, database, images or any other type of information.

2. **Directory File:** Directory files are nothing but a place/area/location where details of files are stored. It contains details about file names, ownership, file size and time when they are created and last modified.
3. **Special Files:** The special files are also called as device files. The special files are those which are not created by the user or the files which are necessary to run a system. These are the files that are created by the system. All the files of an operating system are referred to as special files. There are many types of special files and they are, system files or windows files, input/output files. System files are stored using .sys extension.
4. **FIFO Files:** The first in first Out Files are used by the system for executing the processes in a particular order. Which means that the files which comes first, will be executed first and the system maintains a order known as Sequence order. When a user requests for a service from the system, then the requests of the users are arranged into some files and all the requests will be performed by the system.

OPERATIONS ON FILE STRUCTURES

File structures support various operations that enable efficient management of data stored within files. These operations include:

1. **Creation:** The creation operation involves establishing a new file structure on a storage medium. During this process, the file structure is initialized, and necessary metadata structures, such as file headers or indexes, are set up. This operation prepares the file structure for subsequent data storage and retrieval operations.
2. **Opening and Closing:** Opening a file structure involves establishing a connection or access point to an existing file on the storage medium. Opening a file grants the ability to perform read, write, and other operations on the file's data. After completing operations on a file, it is essential to close the file to release system resources and ensure data integrity.

3. **Reading:** The reading operation allows retrieving data from a file structure. It involves accessing specific records or blocks of data within the file and transferring the data into the application's memory space. Reading operations may involve sequential access, direct access using indexes, or hashed access based on keys.
4. **Writing:** The writing operation involves adding new data or modifying existing data within a file structure. It requires specifying the location or position in the file where the data should be stored or updated. Writing operations may involve appending data at the end of the file, overwriting existing data, or inserting new records at specific positions.
5. **Updating:** The updating operation involves modifying the existing data within a file structure. It allows changing specific fields or attributes of records without completely rewriting the entire record. Updating operations are typically performed by locating the desired record within the file and applying the necessary changes to the identified fields.
6. **Deleting:** The deleting operation involves removing data or records from a file structure. Deletion can be performed on individual records or multiple records at once. Deleting operations may involve marking the deleted record as inactive or physically removing the record from the file structure, depending on the specific file organization method.
7. **Searching:** The searching operation enables locating specific records or data within a file structure based on given criteria. Searching operations may involve sequential scanning of records, direct access using indexes, or utilizing search algorithms such as binary search or hashing techniques for efficient retrieval of data.
8. **Indexing:** Indexing operations involve creating and maintaining index structures to facilitate faster access to data within a file structure. Indexes are used to map key values to specific locations in the file, allowing for direct access to records without the need for sequential scanning.
9. **File Maintenance:** File maintenance operations involve tasks such as reorganizing the file structure to improve performance, resizing the file to accommodate more data, compressing or

decompressing data, and performing integrity checks to ensure data consistency.

These operations collectively enable effective data management within file structures, providing the necessary functionalities for storing, accessing, updating, and manipulating data in a structured manner. The specific operations and their implementation depend on the chosen file structure and the requirements of the application.

ADVANTAGES OF FILE STRUCTURES

1. **Simplicity:** File structures are relatively simple and easy to understand compared to complex database management systems. They provide a straight-forward way of organizing and managing data within files.
2. **Direct Access:** Certain file structures, such as indexed files or hashed files, offer direct access to specific records based on key values. This allows for quick retrieval of data without the need for sequential scanning, resulting in faster access times.
3. **Low Overhead:** File structures typically have low overhead in terms of memory and processing resources. They require minimal system resources to manage and maintain the file organization, making them efficient for handling small to moderate amounts of data.
4. **Flexibility:** File structures provide flexibility in terms of data formats. They can handle a variety of data types, including text, numbers, images, and more, allowing for diverse data storage and retrieval requirements.
5. **Portability:** File structures are portable across different platforms and systems. Files can be easily transferred or shared between different computers or software applications, ensuring data interoperability.

DISADVANTAGES OF FILE STRUCTURES

1. **Limited Data Integrity:** File structures do not offer built-in mechanisms for enforcing data integrity rules and constraints. It is the responsibility of the application or user to ensure data consistency and accuracy. Without proper validation and error-checking mechanisms, data integrity can be compromised.
2. **Lack of Concurrent Access:** File structures often lack built-in support for concurrent access by multiple users or processes. Simultaneous access to the same file by multiple users can lead to data inconsistencies or conflicts unless explicit locking mechanisms are implemented.
3. **Lack of Query Language:** Unlike database management systems, file structures do not provide a dedicated query language for complex data retrieval and manipulation. Retrieving specific data requires manual implementation of search algorithms or iteration over the file contents.
4. **Limited Scalability:** File structures may face challenges in handling large volumes of data efficiently. As the size of the file increases, sequential scanning and direct access operations may become slower, impacting overall performance.
5. **Data Redundancy:** File structures can result in data redundancy if the same information is stored in multiple files or multiple locations within a file. This redundancy can lead to increased storage requirements and potential inconsistencies if updates are not properly synchronized.
6. **Lack of Data Sharing and Integration:** File structures do not inherently support data sharing or integration between different applications or systems. Sharing data stored in file structures often requires manual data export/import processes or file transfer mechanisms.

It's worth noting that while file structures have certain limitations compared to more advanced database systems, they still serve as effective solutions for managing data in smaller-scale applications or scenarios where simplicity and direct access are prioritized over complex data management capabilities.

CHAPTER 3

INTRODUCTION TO INDEXING

In the context of file structures, indexing refers to the technique of organizing and accessing data within a file based on specific attributes or keys. It involves creating additional data structures or indexes that allow efficient searching and retrieval of data from the file.

File indexing is commonly used in databases and file systems to improve query performance and optimize data access. It helps in speeding up search operations by reducing the need to scan the entire file sequentially. Instead, indexes provide a way to locate specific data quickly by utilizing key-value pairs or other indexing mechanisms.

TYPES OF INDEXING

1. Primary Indexing

In primary indexing, a separate index is created that contains the key values of the records along with their corresponding disk block addresses. The index is typically stored in a separate file or data structure. This technique allows for direct access to specific records based on their key values. Primary indexing is a technique used in file structures to facilitate efficient retrieval of records based on their primary key values. The primary key is a unique identifier for each record in a file and is typically chosen from one of the fields in the record.

In primary indexing, an index is created that maps the primary key values to their corresponding disk block addresses. The index is typically stored separately from the actual data file and contains key-address pairs. Each key represents a unique primary key value, and its associated address points to the disk block where the record with that key value is stored.

The primary index allows for direct access to specific records based on their primary key values, eliminating the need for sequential searching through the entire file. When a search or retrieval

operation is performed using a primary key, the index is consulted to quickly locate the disk block address of the desired record. This direct access significantly improves the performance of data retrieval operations.

2. Secondary Indexing

Secondary indexing involves creating an index on a non-key attribute of the records. The secondary index contains the attribute values along with their associated disk block addresses. It provides an alternate way to access records based on non-key attributes, enabling efficient searching and retrieval.

Secondary indexing is a technique used in file structures to facilitate efficient retrieval of records based on attributes other than the primary key. While primary indexing focuses on direct access to records using their primary key values, secondary indexing allows for retrieval based on non-key attributes.

In secondary indexing, an additional index is created that maps the values of the chosen attribute to their corresponding disk block addresses. This secondary index is separate from the primary index and provides an alternate access path to the records based on the selected attribute.

The secondary index typically consists of key-value pairs, where the key represents the attribute value, and the value points to the disk block address where the record(s) with that attribute value are stored. Multiple records with the same attribute value can be associated with a single entry in the secondary index.

The secondary index allows for efficient retrieval of records that match a specific attribute value. When a search operation is performed using the secondary attribute, the index is consulted to locate the disk block addresses of the relevant records. This enables quick access to the desired records without the need for sequential scanning of the entire file.

PURPOSE OF INDEXING

The purpose of indexing in data management and retrieval systems is to improve the efficiency and performance of data access operations. Indexing accomplishes this by providing a data structure or mechanism that enables faster searching, retrieval, and manipulation of specific data based on key values or attributes. The key purposes of indexing include:

1. **Improved Data Retrieval Speed:** Indexing allows for quick and direct access to specific data, eliminating the need for sequential scanning of the entire dataset. By using indexes, data can be located and retrieved in a fraction of the time compared to unindexed structures. This speeds up data retrieval operations and improves system responsiveness.
2. **Efficient Search Operations:** Indexing facilitates efficient search operations by reducing the search space. Instead of examining every data record, indexes narrow down the search to a subset of data that matches specific criteria or key values. This dramatically improves search performance, especially in large datasets, and enables faster query execution.
3. **Optimized Query Performance:** Indexes optimize query performance by enabling the query optimizer to choose efficient access paths and join strategies. By utilizing indexes, the optimizer can select the most appropriate index or combination of indexes to minimize disk I/O operations and reduce query execution time. This enhances overall system performance and scalability.
4. **Enhanced Data Integrity:** Indexing can help enforce data integrity by supporting unique key constraints. By creating a unique index on a key attribute, duplicate values are automatically prevented, ensuring that each record has a distinct key value. This promotes data integrity and prevents data inconsistencies.
5. **Efficient Data Modification:** Although indexing primarily focuses on data retrieval, it can also enhance data modification operations. With indexes, the system can quickly locate and modify specific data without scanning the entire dataset. This improves the performance of data insertion, deletion, and updates.

6. **Flexibility in Data Access:** Indexing provides flexibility in accessing data based on different criteria. By creating indexes on various attributes, users can efficiently retrieve data using different search patterns, sorting orders, or filtering conditions. This enhances the system's versatility and usability.
7. **Concurrency and Scalability:** Indexing supports efficient concurrent access to data by allowing multiple users or processes to access different portions of the dataset simultaneously. By providing direct access paths through indexes, concurrency and scalability can be achieved without contention and performance degradation.

ADVANTAGES OF INDEXING

Indexing offers several benefits in data storage and retrieval systems. Here are some key benefits of indexing:

1. **Improved Search Performance:** Indexing significantly improves search performance by reducing the number of disk I/O operations required to locate specific data. Instead of scanning the entire dataset, indexes provide a direct access path to relevant data based on key values or attributes. This speeds up search queries and enhances overall system responsiveness.
2. **Faster Data Retrieval:** With indexing, the time required to retrieve specific data is significantly reduced. Indexes serve as lookup tables, allowing for quick identification of data locations based on key values or attributes. This eliminates the need for time-consuming sequential scans, enabling faster retrieval of desired data.
3. **Efficient Query Execution:** Indexes optimize query execution by facilitating the use of query optimization techniques. Query optimizers can utilize indexes to determine the most efficient access paths, join strategies, and filtering mechanisms for query processing. This results in faster query execution and improved system performance.
4. **Reduced Disk I/O Operations:** By providing direct access paths to data, indexes reduce the number of disk I/O operations needed for data retrieval. Instead of scanning the entire dataset, the system can retrieve specific data directly using index entries. This minimizes disk read/write operations, reducing disk latency and improving overall system efficiency.

5. **Enhanced Concurrency:** Indexing allows for efficient concurrent data access. Multiple users or processes can simultaneously access different portions of the dataset through appropriate indexes, reducing contention and improving system concurrency. This enables better scalability and responsiveness in multi-user or multi-threaded environments.
6. **Optimal Disk Space Utilization:** Indexing can optimize disk space utilization by reducing data duplication. Instead of storing duplicate copies of data, indexes provide references or pointers to the actual data locations. This saves disk space and allows for more efficient storage management.
7. **Flexibility in Data Retrieval:** Indexing provides flexibility in retrieving data based on different criteria. By creating indexes on various attributes, the system can support diverse search and retrieval patterns. Users can efficiently retrieve data based on key values, range queries, sorting orders, and other specific criteria, enhancing the system's flexibility and usability.
8. **Efficient Data Modification:** While indexes primarily benefit data retrieval, they can also improve data modification operations. Though updates may require additional index maintenance, efficient indexes enable faster identification and modification of specific data. This enhances overall system performance during data insertion, deletion, and updates.

Overall, indexing plays a vital role in improving search performance, data retrieval speed, and system efficiency. It enables faster query execution, reduces disk I/O operations, enhances concurrency, optimizes disk space utilization, and provides flexibility in data retrieval. By leveraging appropriate indexing strategies, systems can deliver faster and more efficient data storage and retrieval capabilities.

DISADVANTAGES OF INDEXING

1. **Increased Storage Overhead:** Indexes require additional storage space to store the index structures and associated metadata. This can lead to increased storage requirements, especially when dealing with large datasets or multiple indexes. The additional storage overhead should be considered when designing and managing indexes.
2. **Increased Insertion and Update Time:** Indexes need to be updated whenever new data is inserted, updated, or deleted in the underlying dataset. Maintaining the index structures can introduce additional overhead, causing a slight delay in insertion and update operations. The extent of this overhead depends on the size and complexity of the indexes.
3. **Increased Disk Space Usage:** Indexes consume disk space, which can be a concern when working with limited storage resources or very large datasets. Having multiple indexes or indexes with high cardinality attributes can significantly increase disk space usage. Efficient index design and management strategies should be employed to mitigate excessive disk space usage.
4. **Index Maintenance Overhead:** Indexes require ongoing maintenance to remain synchronized with the underlying dataset. This maintenance includes updates, rebalancing, and reorganizing the indexes as the dataset evolves. The maintenance operations can consume system resources and introduce additional overhead, particularly for frequently updated datasets.

CHAPTER 4

INTRODUCTION TO HASHING

Hashing is an algorithm that calculates a fixed-size bit string value from a file. A file basically contains blocks of data. Hashing transforms this data into a far shorter fixed-length value or key which represents the original string. The hash value can be considered the distilled summary of everything within that file.

The transformation of a search key into a number by means of mathematical Calculations. A good hashing algorithm would exhibit a property called the avalanche effect, where the resulting hash output would change significantly or entirely even when a single bit or byte of data within a file is changed. A hash function that does not do this is considered to have poor randomization, which would be easy to break by hackers.

A hash is usually a hexadecimal string of several characters. Hashing is also a unidirectional process so you can never work backwards to get back the original data. A good hash algorithm should be complex enough such that it does not produce the same hash value from two different inputs. If it does, this is known as a hash collision. A hash algorithm can only be considered good and acceptable if it can offer a very low chance of collision.

SHA-1 HASHING ALGORITHM

SHA-1 (Secure Hash Algorithm-1) is a cryptographic hash function that generates a 160-bit (20-byte) hash value known as a message digest. It was developed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) in 1995.

The primary purpose of SHA-1 is to ensure data integrity and verify the authenticity of the data. It takes an input message of arbitrary length and produces a fixed-size hash value that is unique to the input data. Even a small change in the input message will result in a significantly different hash value. The hash value is typically represented as a hexadecimal string.

STEPS IN SHA-1 ALGORITHM

General steps involved in SHA-1 algorithm:

1. **Padding:** The input message is padded to ensure it can be divided into blocks of 512 bits. The padding includes adding a '1' bit followed by '0' bits to the message, along with a representation of the original message length.
2. **Message Expansion:** The padded message is divided into blocks of 512 bits each. If the message is longer than 512 bits, multiple blocks are created.
3. **Initialization:** The SHA-1 algorithm initializes five 32-bit variables, known as registers or buffers, with specific initial values. These registers are denoted as A, B, C, D, and E.
4. **Processing of Message Blocks:** Each message block is processed in a series of rounds. Each round involves various logical functions, bitwise operations, and modular arithmetic.
 - a. **Message Block Division:** Each 512-bit message block is divided into sixteen 32-bit words, denoted as W[0] to W[15].
 - b. **Expansion of Message Words:** Additional words (W[16] to W[79]) are generated through a specific expansion process, using bitwise operations and a cyclic left shift.
 - c. **Round Operations:** In each round, the values of the registers are updated based on the current message words and the values from the previous round.
 - d. **Final Hash Value:** After processing all the message blocks, the final hash value is obtained by concatenating the values of the registers A, B, C, D, and E. Each register represents a portion of the final hash value.
 - e. **Output:** The final hash value, typically represented as a hexadecimal string, serves as the output of the SHA-1 algorithm. It represents the unique fingerprint of the input message.

CHAPTER 5

SYSTEM SPECIFICATION

HARDWARE SPECIFICATION

- Processor: intel 3 or AMD 3
- RAM: 4 GB RAM minimum
- Hard Disk: 20 GB or grater
- Monitor: VGA/SVGA
- Keyboard: 104 keys standard
- Mouse: 2/3 button. Optical/Mechanical

SOFTWARE SPECIFICATION

- Operating System: Microsoft Windows (64 bit)
- Tech Stach: Python
- IDE: VS CODE (or any other Python IDE)
- File: NOTEPAD (.txt File, .py File, .idx File)

Python

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation via the off-side rule.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in

2000. Python 3.0, released in 2008, was a major revision not completely backward-compatible with earlier versions. Python 2.7.18, released in 2020, was the last release of Python 2.

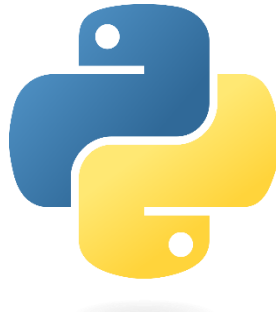


Fig 4.1: Python logo

Python consistently ranks as one of the most popular programming languages.

Key features of Python

The following are some of the features in Python that are discussed below:

1. Easy to Code

Python is a very high-level programming language, yet it is effortless to learn. Anyone can learn to code in Python in just a few hours or a few days. Mastering Python and all its advanced concepts, packages and modules might take some more time. However, learning the basic Python syntax is very easy, as compared to other popular languages like C, C++, and Java.

2. Easy to Read

Python code looks like simple English words. There is no use of semicolons or brackets, and the indentations define the code block. You can tell what the code is supposed to do simply by looking at it.

3. Free and Open-Source

Python is developed under an OSI-approved open source license. Hence, it is completely free to use, even for commercial purposes. It doesn't cost anything to download Python or to include it in your application. It can also be freely modified and re-distributed. Python can be downloaded from the official Python website.

4. Robust Standard Library

Python has an extensive standard library available for anyone to use. This means that programmers don't have to write their code for every single thing unlike other programming languages. There are libraries for image manipulation, databases, unit-testing, expressions and a lot of other functionalities. In addition to the standard library, there is also a growing collection of thousands of components, which are all available in the Python Package Index.

5. Interpreted

When a programming language is interpreted, it means that the source code is executed line by line, and not all at once. Programming languages such as C++ or Java are not interpreted, and hence need to be compiled first to run them. There is no need to compile Python because it is processed at runtime by the interpreter.

6. Portable

Python is portable in the sense that the same code can be used on different machines. Suppose you write a Python code on a Mac. If you want to run it on Windows or Linux later, you don't have to make any changes to it. As such, there is no need to write a program multiple times for several platforms.

7. Object-Oriented and Procedure-Oriented

A programming language is object-oriented if it focuses design around data and objects, rather than functions and logic. On the contrary, a programming language is procedure-oriented if it focuses more on functions (code that can be reused). One of the critical Python features is that it supports both object-oriented and procedure-oriented programming.

8. Extensible

A programming language is said to be extensible if it can be extended to other languages. Python code can also be written in other languages like C++, making it a highly extensible language.

9. Expressive

Python needs to use only a few lines of code to perform complex tasks. For example, to display Hello World, you simply need to type one line - `print("Hello World")`. Other languages like Java or C would take up multiple lines to execute this.

10. Support for GUI

One of the key aspects of any programming language is support for GUI or Graphical User Interface. A user can easily interact with the software using a GUI. Python offers various toolkits, such as Tkinter, wxPython and JPython, which allows for GUI's easy and fast development.

11. Dynamically Typed

Many programming languages need to declare the type of the variable before runtime. With Python, the type of the variable can be decided during runtime. This makes Python a dynamically typed language.

12. High-level Language

Python is a high-level programming language because programmers don't need to remember the system architecture, nor do they have to manage the memory. This makes it super programmer-friendly and is one of the key features of Python.

13. Simplify Complex Software Development

Python can be used to develop both desktop and web apps and complex scientific and numerical applications. Python's data analysis features help you create custom big data solutions without so much time and effort. You can also use the Python data visualization libraries and APIs to present data in a more appealing way. Several advanced software developers use Python to accomplish high-end AI and natural language processing tasks.

14. Other Advanced Programming Features

Python contains several advanced programming features such as generators (used to create iterators with a different approach than most other languages) and list comprehensions (used to

create new lists from other iterables). Python also has automatic memory management eliminating the need to manually allocate and free memory in the code.

VS CODE

VS Code, short for Visual Studio Code, is a popular source code editor developed by Microsoft. It is designed to be lightweight, highly customizable, and optimized for modern programming languages and workflows.



Fig 4.2: VS Code Logo

Here is a brief overview of VS Code:

1. Features and Functionality

VS Code provides a rich set of features to enhance the coding experience. It offers support for syntax highlighting, code completion, code navigation, and debugging capabilities. It also includes built-in Git integration, terminal access, and a wide range of extensions that can be installed to extend its functionality further.

2. Cross-Platform Compatibility

VS Code is available for Windows, macOS, and Linux, making it a versatile choice for developers working on different platforms. It maintains consistent performance and functionality across these operating systems.

3. Lightweight and Fast

VS Code is built to be lightweight and optimized for speed. It starts up quickly and consumes minimal system resources, ensuring a smooth and responsive coding experience even when working with large codebases.

4. Customization and Extensions

One of the standout features of VS Code is its extensive customization options. Users can personalize the editor's appearance, keyboard shortcuts, and settings according to their preferences. Additionally, the VS Code marketplace offers a vast selection of extensions created by the community, enabling users to enhance their coding environment with additional language support, productivity tools, and integrations with various development frameworks and technologies.

5. Integrated Development Environment (IDE) Features

While VS Code is primarily a code editor, it provides several IDE-like features to streamline development workflows. These include integrated terminal support, built-in Git version control, debugging tools, task automation, and more. These features help developers manage their entire development process within a single environment.

6. Community and Support

VS Code has a large and active community of developers who contribute to its development, create extensions, and provide support through forums and online resources. This vibrant community ensures that users have access to a wide range of learning materials, troubleshooting guides, and community-driven improvements to the editor.

Overall, VS Code offers a powerful, customizable, and lightweight coding environment that caters to the needs of developers across different programming languages and platforms. Its extensive feature set, flexibility, and active community make it a popular choice for developers worldwide.

CHAPTER 6

SYSTEM FUNCTIONALITIES

System functionalities refer to the capabilities and features provided by a computer system or software application to perform specific tasks or operations. These functionalities enable users to interact with the system, perform desired actions, and achieve their objectives.

The Login records are stored in the file using SHA-1 algorithm and the FIR records are stored using the indexing. The indexing is done by considering the FIR no. as the primary key.

FIR management has the following functionalities:

1. Signup and Login

A user can login to the FIR system if he/she is already having an account otherwise they can signup by providing SSN, First name, Last name and setting an username and password. Using SHA-1 the unique hash value will be generated for the username and password.

2. Register FIR

A FIR can be registered by using providing the FIR no., Accused name, victim name, case date, case time, case description and case status. The record will be stored in the victim file.

3. Search FIR

A FIR can be searched by entering the unique FIR no. The details of the record are displayed if the record exists otherwise displays the message- “No such record exist”.

4. Modify FIR

User can modify the status of the case by entering the FIR number, after modification the details of the modified will be displayed on the screen.

5. Delete FIR

User can enter the FIR number of the particular FIR which he/she wants to delete. The record will be deleted from the victim file and the index file will be modified. This ensures the system's flexibility by the deletion of specific record, providing user with efficient data management capabilities.

6. Display FIR

This operation is used to display all the existing records stored in the victim file to ensure the updation in file after each operation. The Display FIR feature presents the complete list of FIR records in a structured format, enabling users to review and analyse the existing records easily.

CHAPTER 7

SYSTEM IMPLEMENTATION

Implementation is the stage where the theoretical design is turned into a working system. The most crucial stage in achieving a new successful system and in giving confidence on the new system for the users that it will work efficiently and effectively.

The system can be implemented only after thorough testing is done and if it is found to work according to the specification. It involves careful planning, investigation of existing systems and its constraints on implementations, design of methods to achieve the change. Two major tasks of preparing the implementation are education and training of the users and testing of the system.

The implementation phase consists of several activities. The required hardware and software acquisition is carried out. The system may require the need to develop a software. For this purpose, programs are written and tested.

PSEUDO CODES

HASHING

```
def wordret(wa,wb,wc,wd):
    a=int(wa,2)^int(wb,2)^int(wc,2)^int(wd,2)
    return '{0:032b}'.format(a)
def F1(S2,S3,S4):
    return (S2&S3)|(~S2&S4)
def F2(S2,S3,S4):
    return S2^S3^S4
def F3(S2,S3,S4):
    return (S2&S3)|(S2&S4)|(S3&S4)
def F4(S2,S3,S4):
    return S2^S3^S4
```

```
binmessage= sys.argv[1]
```



```
print(binmessage + " ...hello")
```

```
def process(chunk):
```

```
    words=[]
```

```
    words=textwrap.wrap(chunk, 32)
```

```
    for i in range(16,80):
```

```
        print(words[i-3])
```

```
        words.append(wordret(words[i-3],words[i-8],words[i-14],words[i-16]))
```

```
    return words
```

```
def compress(words):
```

```
    k1=0x5A827999
```

```
    k2=0x6ED9EBA1
```

```
    k3=0x8F1BBCDC
```

```
    k4=0xCA62C1D6
```

```
    s1 = 0x67452301
```

```
    s2 = 0xEFCDAB89
```

```
    s3 = 0x98BADCFE
```

```
    s4 = 0x10325476
```

```
    s5 = 0xC3D2E1F0
```

```
    h1=s1
```

```
    h2=s2
```

```
    h3=s3
```

```
    h4=s4
```

```
    h5=s5
```

```
    for i in range(0,20):
```

```
        temp=s5+(s1<<5)+F1(s2,s3,s4)+k1+int(words[i],2)
```

```
        s5=s4
```

```
        s4=s3
```

```
        s3=s2
```

```
        s2=s1
```

```
        s1=temp
```

```
    for i in range(20,40):
```

```
        temp=s5+(s1<<5)+F2(s2,s3,s4)+k2+int(words[i],2)
```

```
        s5=s4
```

```
        s4=s3
```

```
        s3=s2
```

```
        s2=s1
```

```
        s1=temp
```

```
    for i in range(40,60):
```

```
        temp=s5+(s1<<5)+F3(s2,s3,s4)+k3+int(words[i],2)
```

```
        s5=s4
```

```

        s4=s3
        s3=s2
        s2=s1
        s1=temp
    for i in range(60,80):
        temp=s5+(s1<<5)+F4(s2,s3,s4)+k4+int(words[i],2)
        s5=s4
        s4=s3
        s3=s2
        s2=s1
        s1=temp
    h1 = h1 + s1 & 0xffffffff
    h2 = h2 + s2 & 0xffffffff
    h3 = h3 + s3 & 0xffffffff
    h4 = h4 + s4 & 0xffffffff
    h5 = h5 + s5 & 0xffffffff
    return h1,h2,h3,h4,h5
length=len(binmessage)
low=0
high=448
while(length>448):
    chunk=binmessage[low:high]
    binlength='{0:064b}'.format(448)
    chunk+=binlength
    obj1=process(chunk)
    length=length-448
    low=low+448
    high=high+448
    obj2=compress(obj1)
if(length==447):
    binmessage+='1'
    length='{0:064b}'.format(length)
    binmessage+=length
    obj1=process(binmessage)
    obj2=compress(obj1)
elif(length<447):
    binmessage+='1'
    length=len(binmessage)
    for i in range(length,448):
        binmessage+='0'
    length='{0:064b}'.format(length)
    binmessage+=length

```

```

        obj1=process(binmessage)
        obj2=compress(obj1)
hand=open('hashcontent','a')
hand.write('%08x%08x%08x%08x%08x'%(obj2[0],obj2[1],obj2[2],obj2[3],obj2[4])+
"+binmessage+"\n")

```

INSERT RECORD (REGISTER FIR)

```

case_desc1=""
txt_filename="victim.txt"
txt_fil = open(txt_filename, "a")
txt_indexname="index_file.idx"
n=len(sys.argv)
FIR_no=sys.argv[1]
vic_name=sys.argv[2]
acc_name=sys.argv[3]
case_date=sys.argv[4]
case_time=sys.argv[5]
case_stat=sys.argv[6]
for i in range(7,n):
    case_desc1=case_desc1+sys.argv[i]+" "
entry=FIR_no+'
'+vic_name+'|'+acc_name+'|'+case_date+'|'+case_time+'|'+case_desc1+'|'+case_stat+'|'+'\n'
txt_fil.write(entry)
txt_fil.close()
def index_text_file(txt_filename, idx_filename,
    delimiter_chars=".,:;!?"):
    txt_fil = open(txt_filename, "r")
    word_occurrences = dict()
    line_num = 0
    for lin in txt_fil:
        line_num += 1
        # Split the line into words delimited by whitespace.
        #words = lin.split()
        words=re.findall('^F...',lin)
        # Remove unwanted delimiter characters adjoining words.
        words2 = [ word.strip(delimiter_chars) for word in words ]
        # Find and save the occurrences of each word in the line.
        for word in words2:
            if word in word_occurrences:
                word_occurrences[word].append(line_num)
            else:
                word_occurrences[word] = [ line_num ]

```

```

        if (line_num < 1):
            print("No lines found in text file, no index file created.")
            txt_fil.close()
            sys.exit(0)

# Display results.
word_keys=list()
word_keys = list(word_occurrences.keys())
print(word_keys)
#print "{ } unique words found.".format(len(word_keys))
#word_keys = word_occurrences.keys()
# Sort the words in the word_keys list.
word_keys.sort()
# Create the index file.
idx_fil = open(idx_filename, "w")
for word in word_keys:
    line_nums = word_occurrences[word]
    idx_fil.write(word + " ")
    for line_num in line_nums:
        idx_fil.write(str(line_num) + " ")
    idx_fil.write("\n")

txt_fil.close()
idx_fil.close()
index_text_file(txt_filename,txt_indexname)

```

SEARCH FIR

```

txt_file="victim.txt"
idx_file="index_file.idx"
key=sys.argv[1]
record=[]
record=open(txt_file).readlines()
def search(txt_file,idx_file,key):
    flag=0
    idx_f=open(idx_file,"r")
    for line in idx_f:
        if re.match(key,line):
            flag=1
            l=line.split()
            n=len(l)
            txt_f=open(txt_file,"r")
            for i in range (1,n):
                c=int(l[i])

```

```

        #print(record[c-1])
    #return l
    l2=record[c-1].split('|')
    print("\nFIR number:"+l2[0])
    print("Victim name:"+l2[1])
    print("Accused name:"+l2[2])
    print("Case date:"+l2[3])
    print("Case time:"+l2[4])
    print("Case description:"+l2[5])
    print("Case status:"+l2[6)+"\n")
    txt_f.close()
    if(flag==0):
        print("No such record exist")
    idx_f.close()
search(txt_file,idx_file,key)

```

MODIFY FIR

```

def index_text_file(txt_filename, idx_filename,
    delimiter_chars=".,:;!?" ):
    txt_fil = open(txt_filename, "r")
    word_occurrences = { }
    line_num = 0
    for lin in txt_fil:
        line_num += 1
        # Split the line into words delimited by whitespace.
        #words = lin.split()
        words=re.findall('F...',lin)
        # Remove unwanted delimiter characters adjoining words.
        words2 = [ word.strip(delimiter_chars) for word in words ]
        # Find and save the occurrences of each word in the line.
        for word in words2:
            if word_occurrences == word:
                word_occurrences[word].append(line_num)
            else:
                word_occurrences[word] = [ line_num ]
    if line_num < 1:
        print("No lines found in text file, no index file created.")
        txt_fil.close()
        sys.exit(0)

# Display results.
word_keys = word_occurrences.keys()

```

```

#print "{ } unique words found.".format(len(word_keys))
word_keys = list(word_occurrences.keys())
# Sort the words in the word_keys list.
word_keys.sort()
# Create the index file.
idx_fil = open(idx_filename, "w")
for word in word_keys:
    line_nums = word_occurrences[word]
    idx_fil.write(word + " ")
    for line_num in line_nums:
        idx_fil.write(str(line_num) + " ")
    idx_fil.write("\n")
txt_fil.close()
idx_fil.close()
record=[]
record=open("victim.txt").readlines()
def search(txt_file,idx_file,key):
    flag=0
    idx_f=open(idx_file,"r")
    for line in idx_f:
        if re.match(key,line):
            flag=1
            l=line.split()
            n=len(l)
            txt_f=open(txt_file,"r")
            for i in range (1,n):
                c=int(l[i])
                l2=record[c-1].split('|')
            txt_f.close()
            n=int(l[1])
            l3=record[n-1].split('|')
            l3[6]=sys.argv[2]
            file1=open("victim.txt","w")
            file_size=len(record)
            record2=[]
            for i in range(1,file_size+1):
                if(i!=n):
                    record2.append(record[i-1])
            file1.writelines(record2)
            file1.close()
            txt_fil=open("victim.txt","a")
            FIR_no=l3[0]

```

```

        vic_name=l3[1]
        acc_name=l3[2]
        case_date=l3[3]
        case_time=l3[4]
        case_desc=l3[5]
        case_stat=l3[6]

entry=FIR_no+'|'+vic_name+'|'+acc_name+'|'+case_date+'|'+case_time+'|'+case_desc+'|'+case_stat+'|'+'\n'

        txt_fil.write(entry)
        print("\nFIR number:"+l3[0])
        print("Victim name:"+l3[1])
        print("Accused name:"+l3[2])
        print("Case date:"+l3[3])
        print("Case time:"+l3[4])
        print("Case description:"+l3[5])
        print("Case status:"+l3[6]+"\\n")
        print("\\nRecord modified.\\n")
        txt_fil.close()
        index_text_file("victim.txt","index_file.idx")
if(flag==0):
    print("No such record exist")
    return -1
    #return 1
idx_f.close()
search("victim.txt","index_file.idx",sys.argv[1])

```

DELETE FIR

```

def index_text_file(txt_filename, idx_filename,
    delimiter_chars=".,:;!"):
    txt_fil = open(txt_filename, "r")
    word_occurrences = { }
    line_num = 0
    for lin in txt_fil:
        line_num += 1
        # Split the line into words delimited by whitespace.
        #words = lin.split()
        words=re.findall('F...',lin)
        # Remove unwanted delimiter characters adjoining words.
        words2 = [ word.strip(delimiter_chars) for word in words ]
        # Find and save the occurrences of each word in the line.
        for word in words2:

```

```

        if word in word_occurrences:
            word_occurrences[word].append(line_num)
        else:
            word_occurrences[word] = [ line_num ]

if line_num < 1:
    print("No lines found in text file, no index file created.")
    txt_fil.close()
    sys.exit(0)
# Display results.
#word_keys = word_occurrences.keys()
#print "{ } unique words found.".format(len(word_keys))
word_keys = list(word_occurrences.keys())
# Sort the words in the word_keys list.
word_keys.sort()
# Create the index file.
idx_fil = open(idx_filename, "w")
for word in word_keys:
    line_nums = word_occurrences[word]
    idx_fil.write(word + " ")
    for line_num in line_nums:
        idx_fil.write(str(line_num) + " ")
    idx_fil.write("\n")
txt_fil.close()
idx_fil.close()
record=[]
record=open("victim.txt").readlines()
def search(txt_file,idx_file,key):
    flag=0
    idx_f=open(idx_file,"r")
    for line in idx_f:
        if re.match(key,line):
            flag=1
            l=line.split()
            n=len(l)
            txt_f=open(txt_file,"r")
            #print(record)
            for i in range (1,n):
                c=int(l[i])
                #print(record[c-1])
                l2=record[c-1].split('|')

```



```

        print("\nFIR number:"+l2[0])
        print("Victim name:"+l2[1])
        print("Accused name:"+l2[2])
        print("Case date:"+l2[3])
        print("Case time:"+l2[4])
        print("Case description:"+l2[5])
        print("Case status:"+l2[6]+"\n")
    txt_f.close()
    n=len(l)
    l2=[]
    for i in range(1,n):
        l2.append(int(l[i]))
    file1=open("victim.txt","w")
    n=len(record)
    record2=[]
    for i in range(1,n+1): #line number in the original file(1 to ...)
        if i not in l2:
            record2.append(record[i-1])
    print("Record deleted.\n")
    file1.writelines(record2)
    file1.close()
    index_text_file("victim.txt","index_file.idx")

if(flag==0):
    print("No such record exist")
idx_f.close()

search("victim.txt","index_file.idx",sys.argv[1])

```

DISPLAY FIR

```

def display_fir_records():
    # Read the records from the victim.txt file
    with open("victim.txt", "r") as file:
        records = file.readlines()
    # Create a Tkinter window
    window = tk.Tk()
    window.title("FIR Records")
    window.configure(bg="#b1dee3") # Set background color
    # Create a Treeview widget
    treeview = ttk.Treeview(window)
    treeview["columns"] = ("FIR No.", "Victim Name", "Accused Name", "Case Date", "Case
Time", "Case Details", "Case Status")

```

```

treeview.heading("#0", text="Record")
treeview.column("#0", width=70, stretch=tk.NO)
# Define column alignment
alignments = ("center", "center", "center", "center", "center", "center", "center")
for column, alignment in zip(treeview["columns"], alignments):
    treeview.heading(column, text=column)
    treeview.column(column, width=150, anchor=alignment)
# Insert the records into the Treeview
for i, record in enumerate(records, start=1):
    record = record.strip().split("|")
    fir_no = record[0]
    victim_name = record[1]
    accused_name = record[2]
    case_date = record[3]
    case_time = record[4]
    case_details = record[5]
    case_status = record[6]
    treeview.insert("", "end", text=str(i), values=(fir_no, victim_name, accused_name,
case_date, case_time, case_details, case_status))
# Pack the Treeview widget
treeview.pack(padx=10, pady=10)
# Start the Tkinter event loop
window.mainloop()
# Call the display_fir_records function to display the records
display_fir_records()

```

CHAPTER 8

SYSTEM TESTING

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is defect free.

UNIT TESTING

It focuses on smallest unit of software design. In this we test an individual unit or group of inter related units. It is often done by programmer by using sample input and observing its corresponding outputs.

Example:

- a. In a program we are checking if loop, method or if a function is working correctly.
- b. Incorrect or misunderstood arithmetic procedures.
- c. Incorrect initialization.

In this project we have implemented and tested for different inputs like FIR No., Username, Password, Victim and accused names, Case date, time, description etc.

INTEGRATION TESTING

When a software test case covers more than one unit, it is considered an integration test. When developing a software test case, the lines between units can quickly evolve into integration tests. The dependency itself will not need to be tested and the integration to it will be mocked or faked. It is of two types: (i) Top down (ii) Bottom up

Example:

- a. **Black Box testing** –It is used for validation. Here we ignore internal working mechanism and focus on what the output would be.
- b. **White Box testing** –It is used for verification. Here we focus on internal mechanism i.e., how the output is obtained.

Login Page:

SL.NO	Test Cases	Expected Result	Test Result
1.	On Clicking Login	The user's username and password will be authenticated with the data in the hashcontent.txt .	Successful
2.	On Clicking SignUp	The user informations are taken and stored in hashcontent.txt.	Successful

Table 8.1 Login Page**Main Menu:**

SL.NO	Test Cases	Expected Result	Test Result
1.	On Clicking Register FIR	Should redirect to Add FIR page	Successful
2.	On Clicking Modify FIR	Should redirect to Modify FIR page	Successful
3.	On Clicking Search FIR	Should redirect to Search FIR page	Successful
4.	On Clicking Delete FIR	Should redirect to Delete FIR page	Successful
5.	On clicking Display FIR	Should display all the records in a tabular form	Successful

Table 8.2 Main Menu**Add FIR:**

SL.NO	Test cases	Expected Result	Test Result
1.	On Clicking Add FIR	Should display the Fields for inserting various information	Successful
2.	On Clicking Go To Main Menu	Should redirect to Home page	Successful

Table 8.3 Add FIR

Modify FIR:

SL.NO	Test cases	Expected Result	Test Result
1.	On Clicking Modify FIR	Should display the Fields for inserting FIR no. and New Status or display No record Exist for incorrect FIR.	Successful
2.	On Clicking Go To Main Menu	Should redirect to Home page	Successful

Table 8.4 Modify FIR**Search FIR:**

SL.NO	Test cases	Expected Result	Test Result
1.	On Clicking Search FIR	Should display the record for the given FIR no. or display No record Exist for incorrect FIR	Successful
2.	On Clicking Go To Main Menu	Should redirect to Home page	Successful

Table 8.5 Search FIR**Delete FIR:**

SL.NO	Test cases	Expected Result	Test Result
1.	On Clicking Delete FIR	Should delete the specified FIR no. or display No record Exist for incorrect FIR	Successful
2.	On Clicking Go To Main Menu	Should redirect to Home page	Successful

Table 8.6 Delete FIR

SYSTEM TESTING

In this the software is tested such that it works fine for different operating systems. It is covered under the black box testing technique. In this we focus on required inputs and outputs without focusing on internal working.

Sl. No	Functionality	Action	Expected Result	Actual Result	Test Result
1	Inserting User Credentials to Hash content file	Signing up by entering SSN, First name, Lastname, Username and Password.	Should insert record to hash content file and render the Login page to Main menu.	Inserts record to hash content file and renders Login page to main menu.	PASS
2	Inserting FIR details into the file.	Entering FIR No, accused, victim, case-date, case-time, case-description, case-status.	Should insert record in victim file	Inserts record Into the victim file	PASS
3	Searching of records from thefiles	Search the record according to the entered FIR No.	Should display the result after searching	Display the result accordingly.	PASS
4	Retrieval of all records from thefiles	Displays all records fromthe victim files	Should display all records from the victim files.	Displays all records from the victim files.	PASS
5	Modifying Index file after insertion or deletion	Changing index of the record	Should change the index.	Changes the index after insertion or deletion	PASS

6	Modifying of records from thefiles	Modify the record according to the entered FIR No.	Should modify the Case_status after searching	Modify the record accordingly.	PASS
7	Deleting of records from thefiles	Delete the record according to the entered FIR No.	Should Delete the record after searching	Delete the record accordingly.	PASS
8	Returning to Main menu on clicking 'Go to Main menu'	Clicking on 'Go to main menu' button to return to Main menu	Should Render the Current view to Main menu	Renders Current view to Main menu	PASS

Table 8.7 System Testing

CHAPTER 9

RESULTS AND DISCUSSIONS

1. Signup screen of FIR Management System:



FIR PORTAL



SIGN-IN

SSN

First Name

Last Name

Username

Password

Submit

Login

Quit

Fig 8.1: Signup Page



FIR PORTAL



SIGN-IN

SSN

First Name

Last Name

Username

Password

Submit

Login

Quit

Fig 8.2: Successful signup

The above Figure 8.1 and Figure 8.2 shows the Signup screen where user can Signup by entering the SSN, First Name, Last Name, Username and Password where Username is unique and Password must contain

2. Login screen of FIR Management System

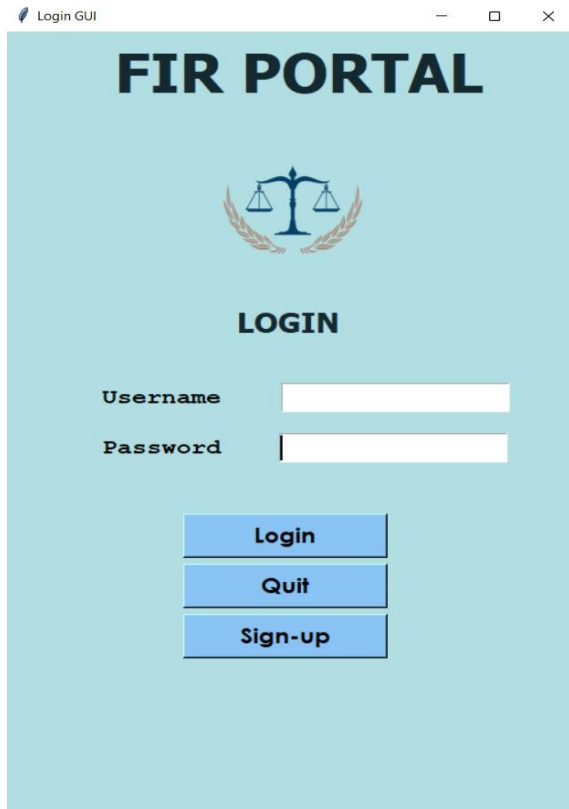


Fig 8.3: Login page

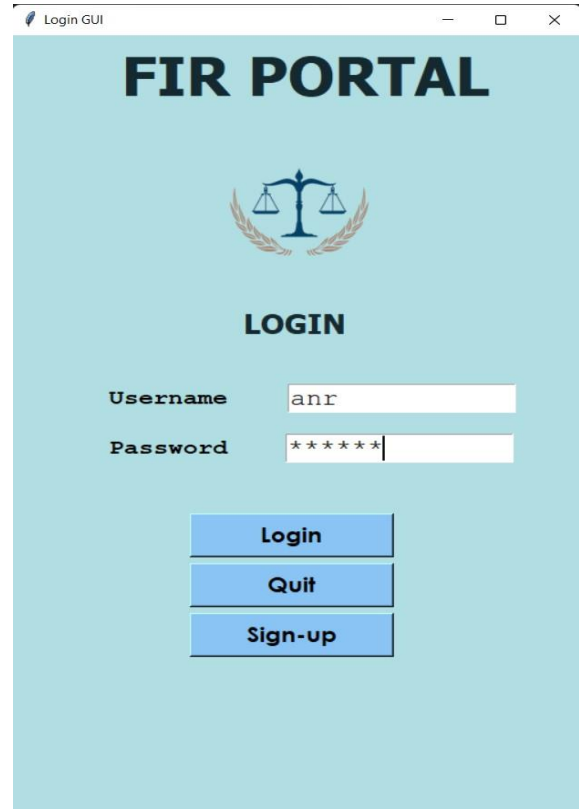


Fig 8.4: Successful Login

The Fig 8.2 and Fig 8.3 shows the Login screen where the user can gain access to the FIR Portal after the successful Signup by entering the correct Username and Password.

3. Main Menu of FIR Portal

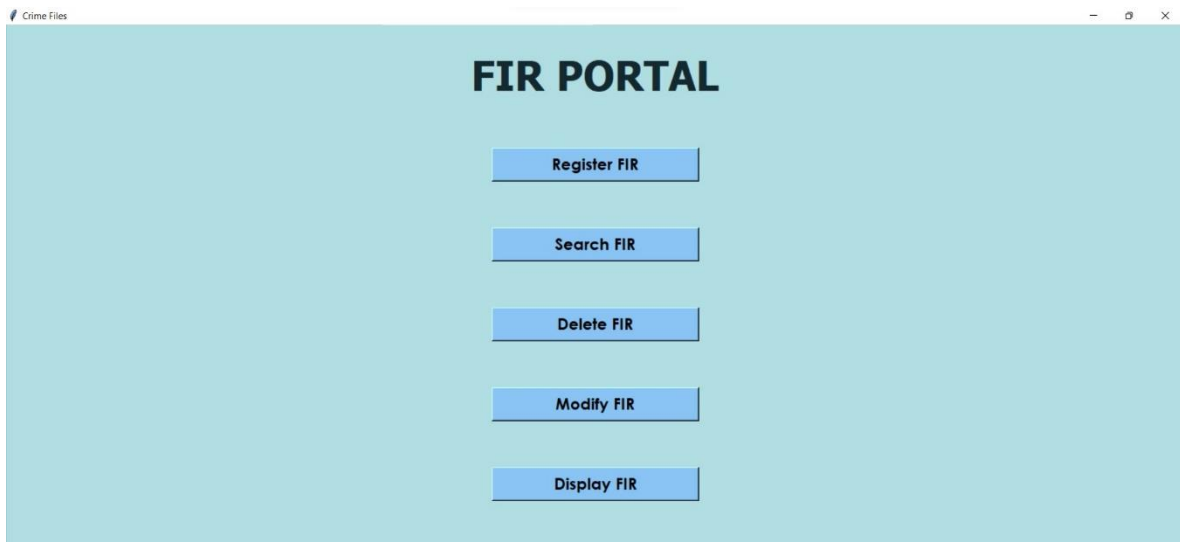


Fig 8.5: Main Menu of FIR Portal

The Fig 8.5 shows the Main Menu of FIR Portal where user can choose any of the displayed options to perform the operation on the file.

4. Register FIR

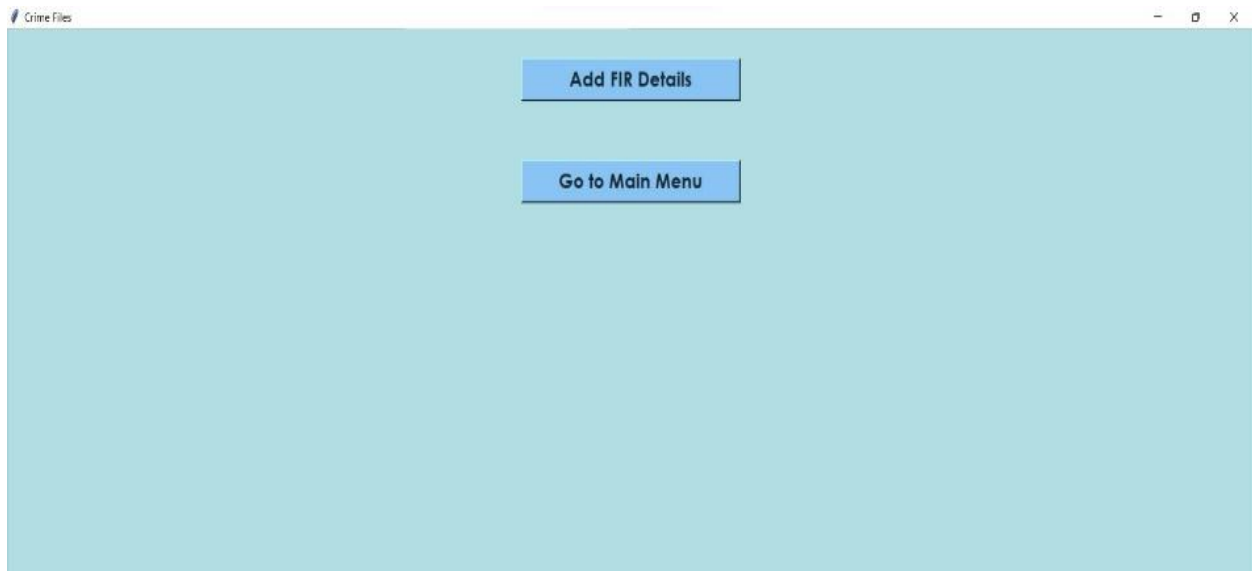


Fig 8.6: To register FIR

On choosing the 'Register FIR' option from the Main Menu the User can Register a FIR by clicking on 'Add FIR Details' button as shown in the Fig 8.6.


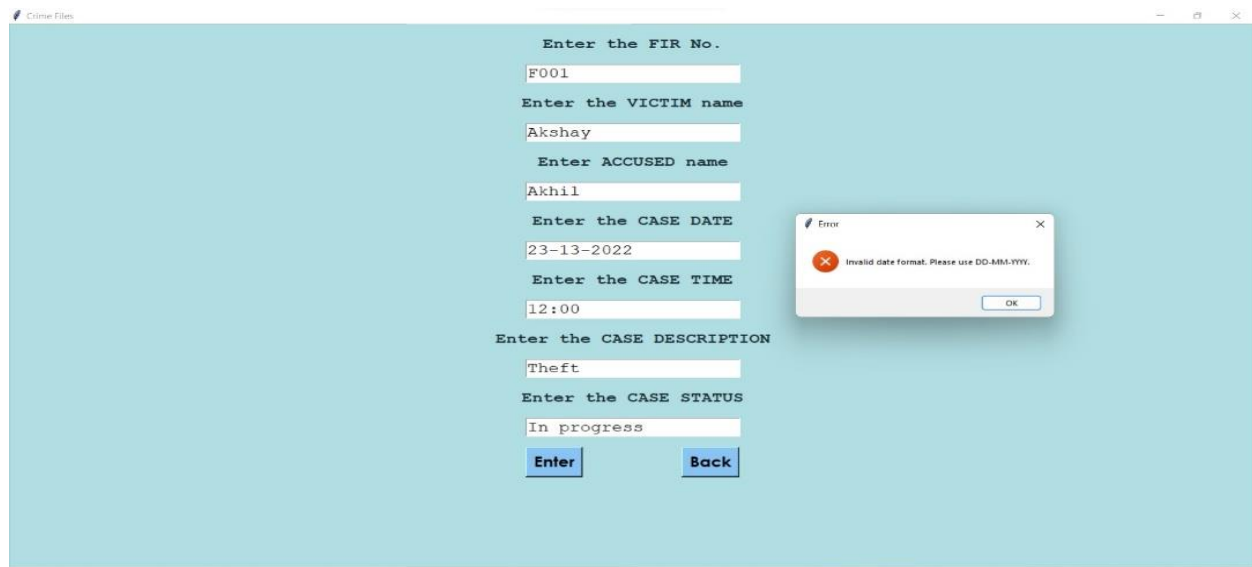
A screenshot of a web application window titled 'Crime Files'. The window has a light blue background. It displays a form for registering an FIR. The form consists of several text input fields with labels above them: 'Enter the FIR No.' (with 'F001' entered), 'Enter the VICTIM name' (with 'Akshay' entered), 'Enter ACCUSED name' (with 'Akhil' entered), 'Enter the CASE DATE' (with '23-11-2022' entered), 'Enter the CASE TIME' (with '12:00' entered), 'Enter the CASE DESCRIPTION' (with 'Theft' entered), and 'Enter the CASE STATUS' (with 'In progress' entered). At the bottom of the form, there are two blue buttons: 'Enter' and 'Back'. The window has standard OS controls (minimize, maximize, close) in the top right corner.

Fig 8.7: FIR Details

5. Failed Registration

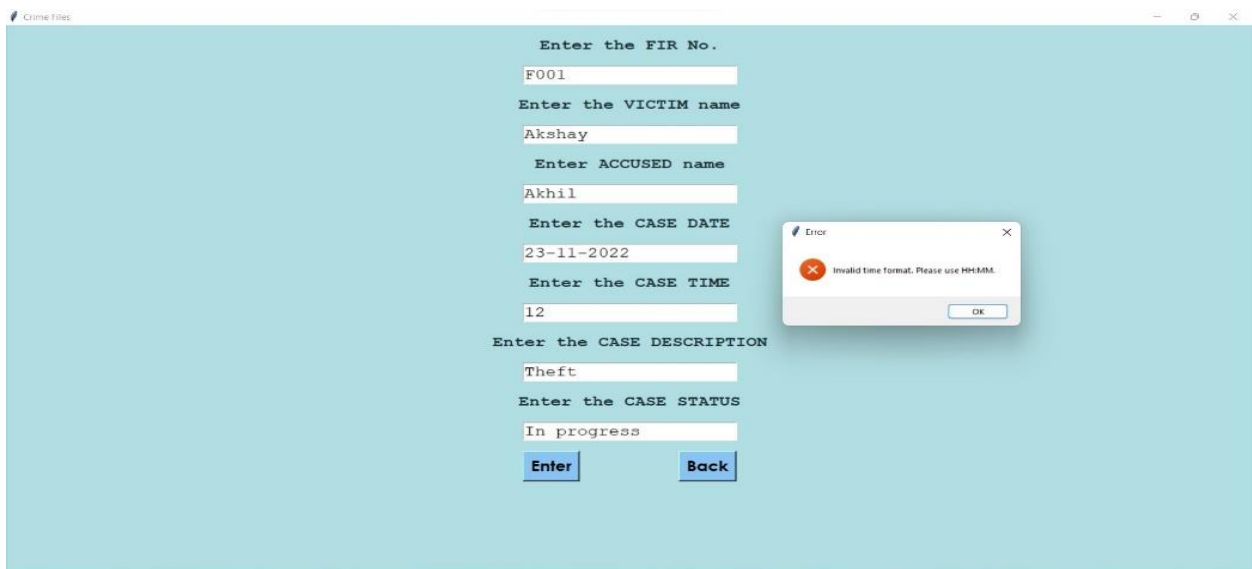


The screenshot shows the 'Crime Files' application window. The form contains the following fields and values:

- Enter the FIR No.: F001
- Enter the VICTIM name: Akshay
- Enter ACCUSED name: Akhil
- Enter the CASE DATE: 23-13-2022
- Enter the CASE TIME: 12:00
- Enter the CASE DESCRIPTION: Theft
- Enter the CASE STATUS: In progress

At the bottom of the form are 'Enter' and 'Back' buttons. An error dialog box is displayed on the right side of the window with the message: "Invalid date format. Please use DD-MM-YYYY." and an 'OK' button.

Fig 8.8: Invalid Date format



The screenshot shows the 'Crime Files' application window. The form contains the following fields and values:

- Enter the FIR No.: F001
- Enter the VICTIM name: Akshay
- Enter ACCUSED name: Akhil
- Enter the CASE DATE: 23-11-2022
- Enter the CASE TIME: 12
- Enter the CASE DESCRIPTION: Theft
- Enter the CASE STATUS: In progress

At the bottom of the form are 'Enter' and 'Back' buttons. An error dialog box is displayed on the right side of the window with the message: "Invalid time format. Please use HH:MM." and an 'OK' button.

Fig 8.9: Invalid Time format

The Error message dialog box appears on entering the invalid Date or Time format as shown in the above figures – Fig 8.8 and Fig 8.9.

6. Search FIR

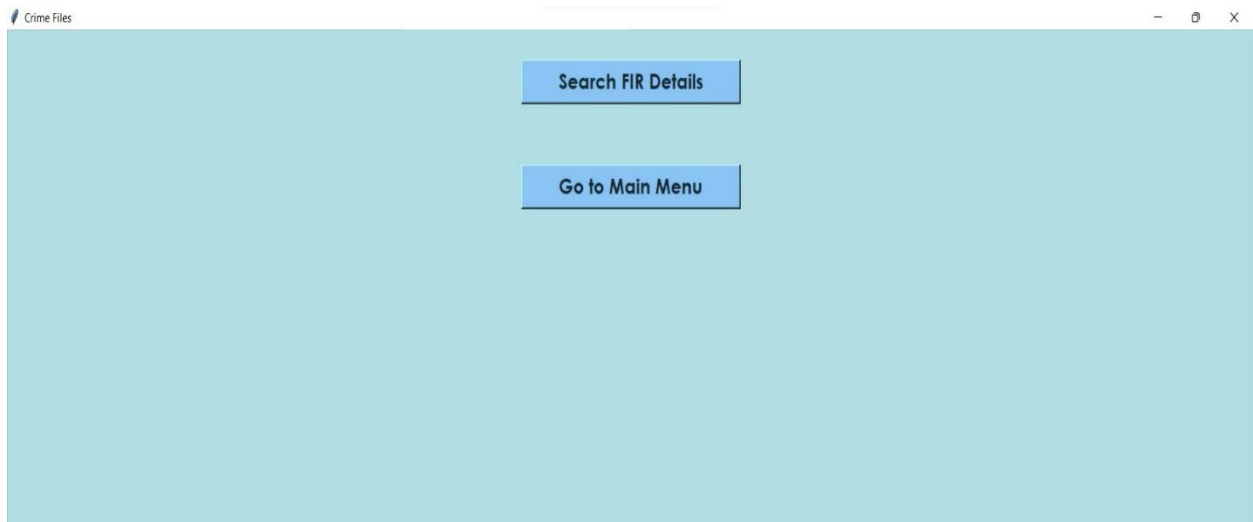


Fig 8.10 To Search FIR

On choosing the Search FIR option from the Main Menu the user can search a FIR by clicking on “Search FIR Details” button as shown in the Fig 8.10.

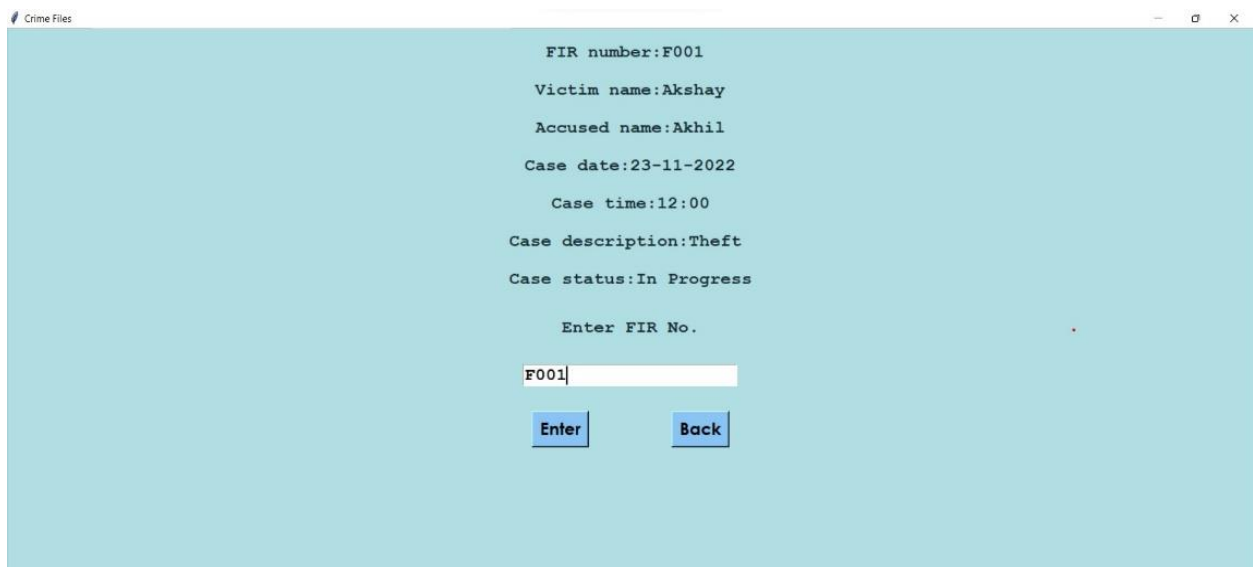


Fig 8.11 Successful Search

The FIR Details of the requested FIR are displayed on the screen as shown in the above Fig 8.11.



Fig 8.12: Failed Search

The error message is displayed on entering incorrect FIR No. as shown in the Fig 8.12.

7. Modify FIR

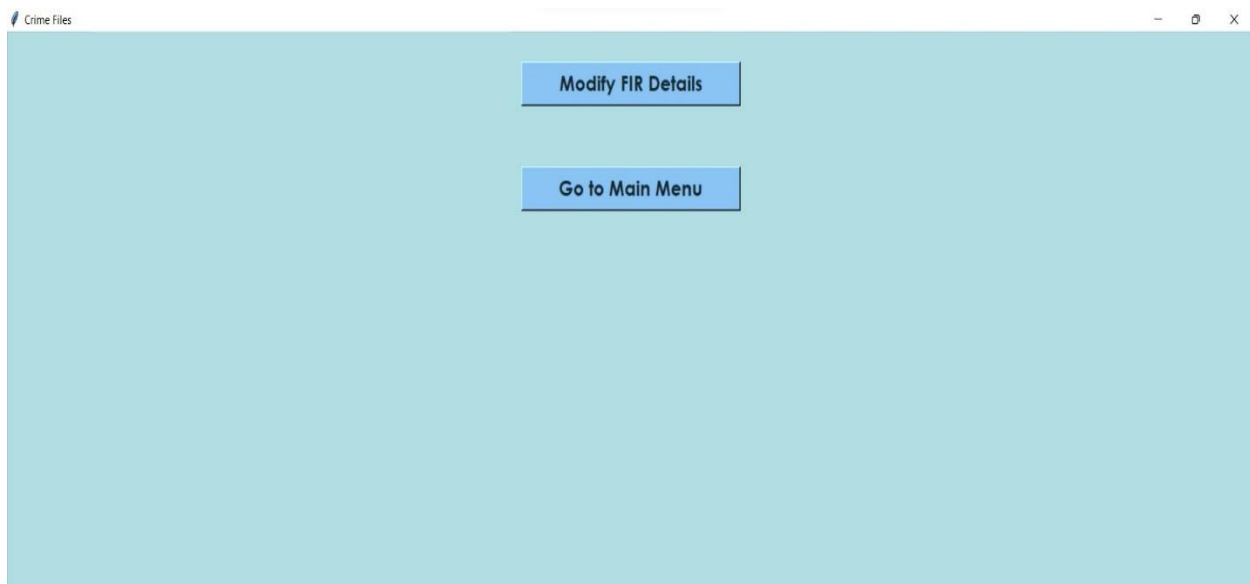


Fig 8.13: Modify FIR

On choosing the Modify FIR option from the Main Menu the user can Modify a FIR by clicking on “Modify FIR Details” button as shown in the Fig 8.13.



The screenshot shows a web application window titled "Crime Files". The main content area has a light blue background and displays the following text in a monospaced font:

```
FIR number:F001
Victim name:Akshay
Accused name:Akhil
Case date:23-11-2022
Case time:12:00
Case description:Theft
Case status:Solved

Record modified.

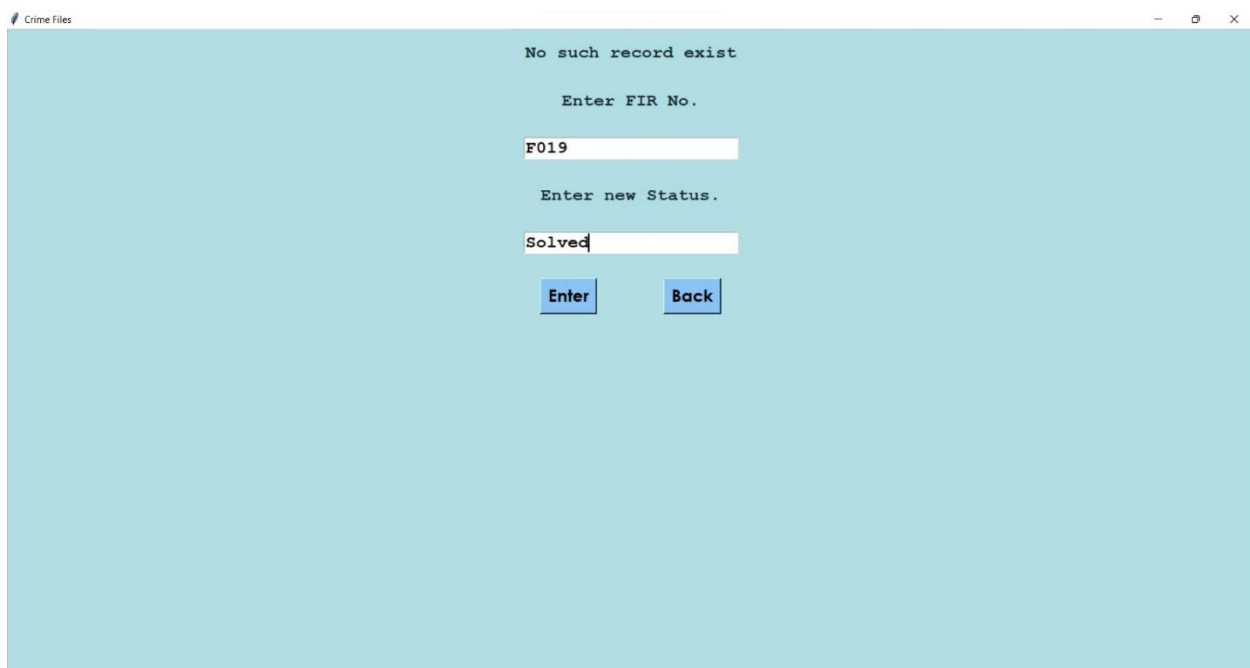
Enter FIR No.
F001

Enter new Status.
Solved

[Enter] [Back]
```

Fig 8.14: Successful Modification

The status of the FIR is updated and the FIR details are displayed after the Modification as shown in Fig 8.14.



The screenshot shows the same "Crime Files" application window. The main content area displays the following text in a monospaced font:

```
No such record exist

Enter FIR No.
F019

Enter new Status.
Solved

[Enter] [Back]
```

Fig 8.15: Failed attempt

The error message is displayed on entering incorrect FIR No. as shown in the Fig 8.15.

8. Display FIR Records

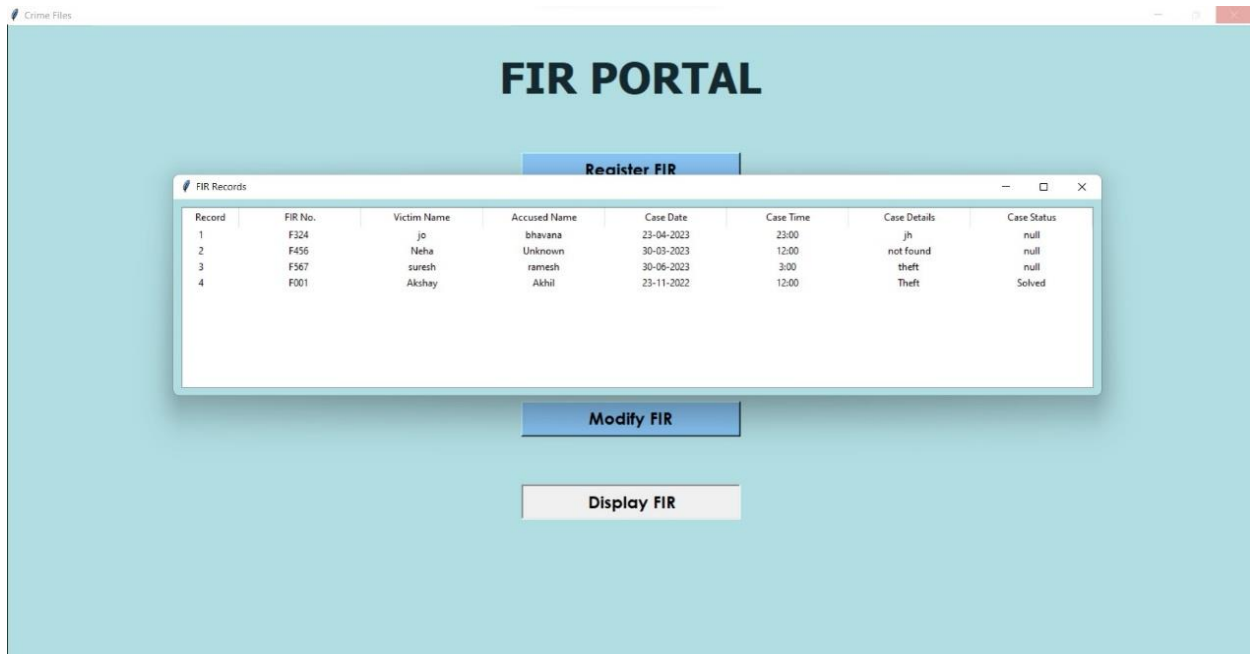


Fig 8.16: FIR Records Display

On choosing the Display FIR option from the Main Menu the user can view all existing records on the screen as shown in the Fig 8.16.

9. Files after Modification

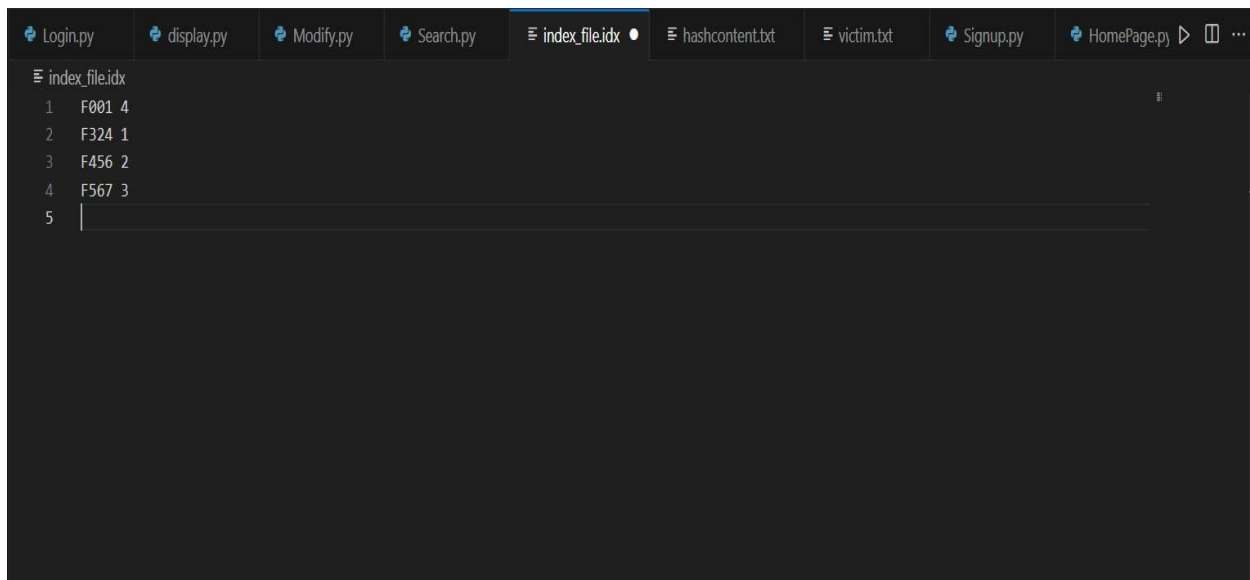
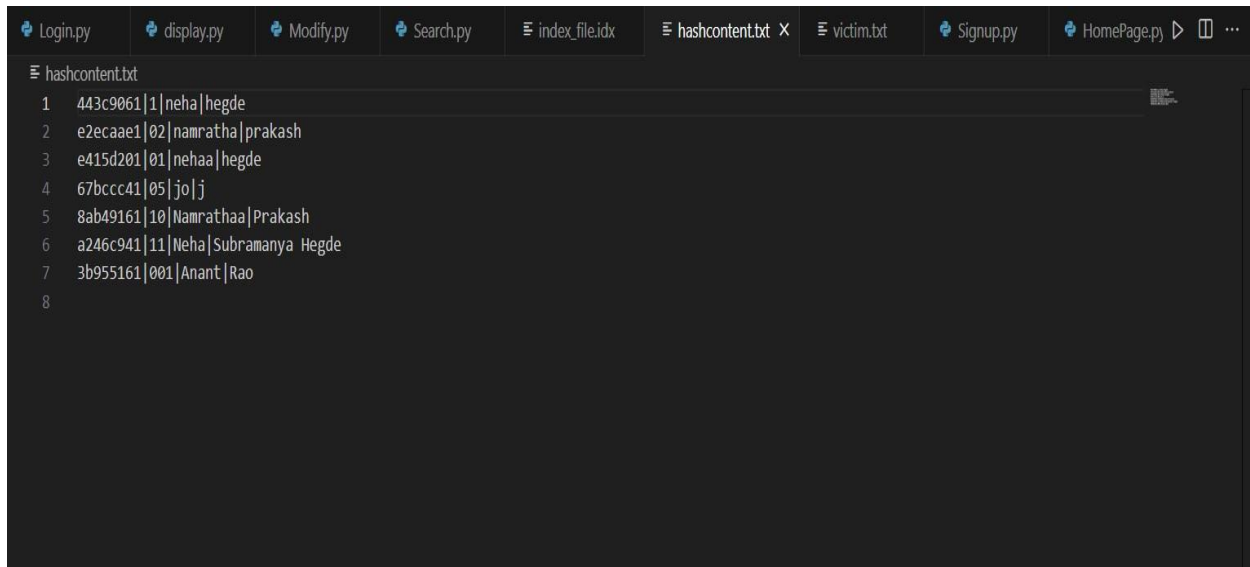
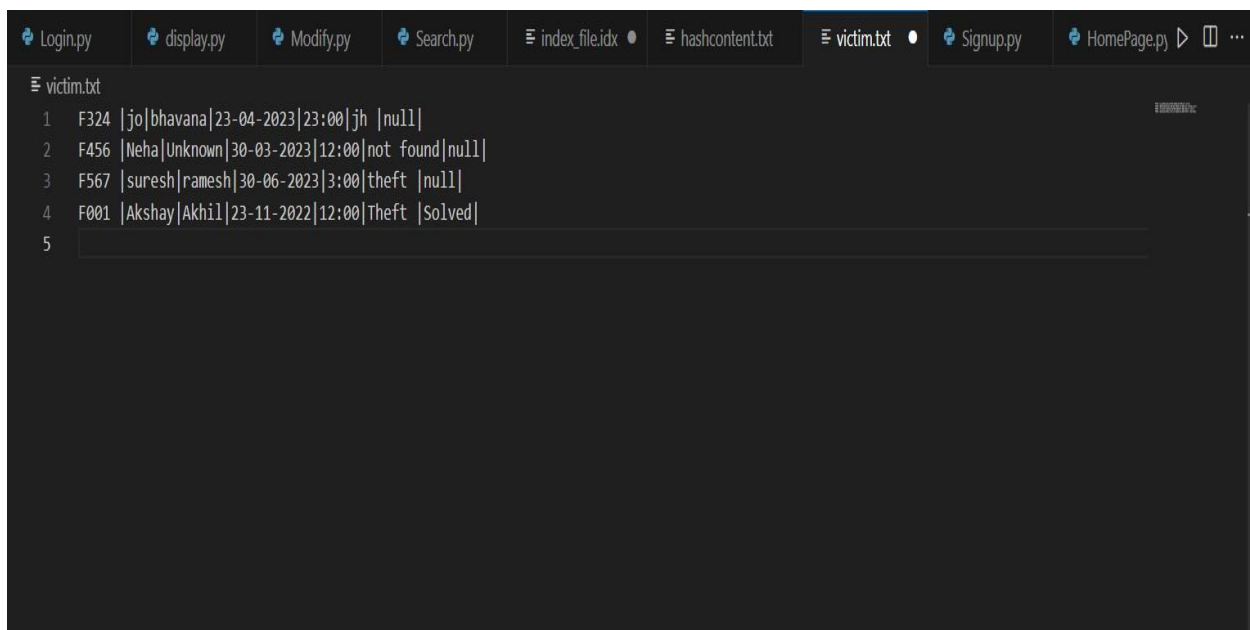


Fig 8.17: Index File



```
Login.py display.py Modify.py Search.py index_file.idx hashcontent.txt X victim.txt Signup.py HomePage.py
hashcontent.txt
1 443c9061|1|neha|hegde
2 e2ecaae1|02|namratha|prakash
3 e415d201|01|nehaa|hegde
4 67bcc41|05|jo|j
5 8ab49161|10|Namrathaa|Prakash
6 a246c941|11|Neha|Subramanya Hegde
7 3b955161|001|Anant|Rao
8
```

Fig 8.18: Hash contents File

```
Login.py display.py Modify.py Search.py index_file.idx hashcontent.txt victim.txt Signup.py HomePage.py
victim.txt
1 F324 |jo|bhavana|23-04-2023|23:00|jh |null|
2 F456 |Neha|Unknown|30-03-2023|12:00|not found|null|
3 F567 |suresh|ramesh|30-06-2023|3:00|theft |null|
4 F001 |Akshay|Akhil|23-11-2022|12:00|Theft |Solved|
5
```

Fig 8.19: Victims File

The Index, Hash contents and Victims files are auto-generated as shown in Fig 8.17, Fig 8.18 and Fig 8.19 respectively.

10. Delete FIR

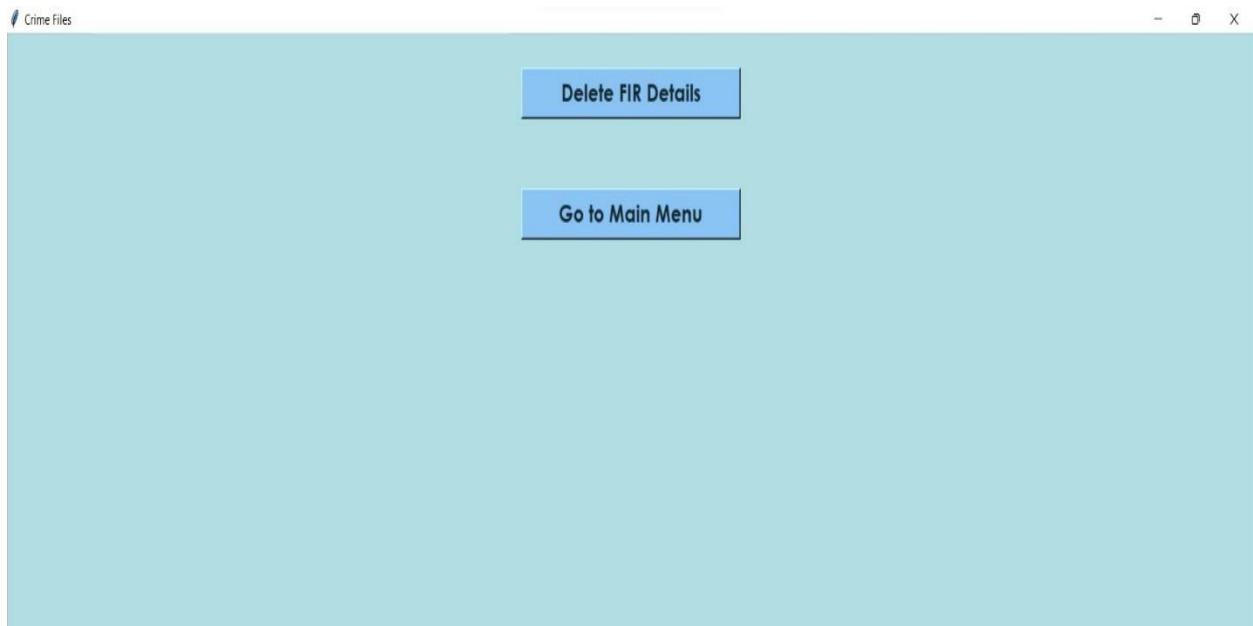


Fig 8.20 Delete FIR

On choosing the Delete FIR option from the Main Menu the user can Delete a record by clicking on “Delete FIR Details” button as shown in the Fig 8.20.



Fig 8.21 Successful Deletion

The details of the deleted FIR are displayed on the screen after the successful deletion as shown in the above Fig 8.21.

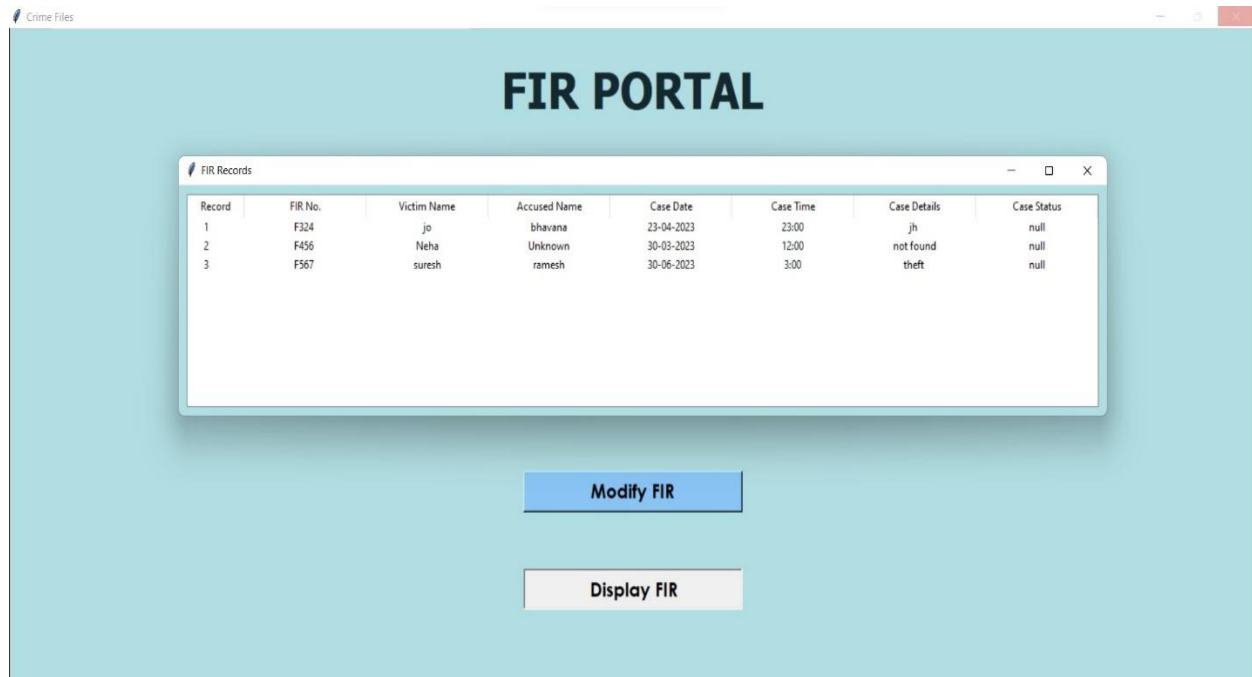


Fig 8.22 Updated display after the deletion

CHAPTER 10

CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION

The mini project entitled “FIR Management System” presents a easy to use computerized version of FIR data that will not only help to overcome the difficulties of manual process but will also help in ease of administration. In this entire process, the project ensures that data stored is easy to access and available at all times. The security and maintenance of data is ensured.

By understanding and implementing file structures concepts, the system is designed to provide various functionalities such as registering FIR, searching for specific records, modifying existing records, deleting and displaying the data to ensure the updation in file after each operation.

The system's utilization of file structures allows for organized and optimized data storage and retrieval. User can easily add FIR details for registration by entering relevant information, ensuring accurate and up-to-date records. The system's flexibility enables the deletion of specific record, providing user with efficient data management capabilities.

Modifying and updating the status of a FIR record within the system is straightforward, enabling user to update the FIR status after the progress in the case, ensuring data accuracy and flexibility. The search functionality allows for quick and precise retrieval, facilitating efficient access to specific FIR details. The Display FIR feature presents the complete list of FIR records in a structured format, enabling users to review and analyse the existing records easily.

Overall, the FIR Management System using file structures offers an intuitive and efficient way of storing and organising FIR records. The system's functionalities enable streamlined record management, accurate record-keeping, and easy access to records stored.

FUTURE ENHANCEMENTS

- **Hierarchical Folder Structure:** Developing a well-defined hierarchical folder structure within the FIR Management system can facilitate the organization and categorization of files. This structure can be based on different criteria, such as incident type, geographical location, or time period, making it easier for users to navigate and locate relevant files.
- **File Versioning:** Introducing file versioning capabilities allows for better management of revisions and updates to FIR-related files. This feature enables tracking changes made to a file over time, preserving previous versions, and providing an audit trail for accountability purposes.
- **Data Compression:** Implementing data compression techniques can optimize storage space within the FIR Management system. Compressing files, particularly for large-sized documents or multimedia files, can reduce storage requirements and enhance system performance without compromising data integrity.
- **Integration with Document Management Systems:** Integrating the FIR Management system with document management systems (DMS) can enhance file collaboration, version control, and workflow management capabilities. This integration allows seamless transfer of files between systems, streamlines document approval processes, and ensures proper document lifecycle management.
- **Backup and Disaster Recovery:** Establishing robust backup and disaster recovery mechanisms ensures the preservation and availability of FIR-related files in the event of system failures, data corruption, or natural disasters. Regular backups, off-site storage, and redundant systems can help in quickly restoring files and minimizing data loss.

These enhancements aim to improve the organization, accessibility, security, and scalability of file management within the FIR Management system, thereby enhancing the overall efficiency and effectiveness of crime reporting and management processes.

BIBLIOGRAPHY

BOOK REFERENCES

1. File Structures - an Object-Oriented Approach with C++, Micheal J Folk, Bill Zoellick, Greg Riccardi, Third edition, Pearson Education, 1998.
2. File Structures - Theory and Practice, Wendell Odom
3. PROGRAMMING IN PYTHON – Learn the Powerful Object-Oriented Programming, Pooja Sharma, BPB Publications.

WEB REFERENCES

1. <https://www.tutorialspoint.com>
2. <https://www.geeksforgeeks.org>
3. <https://www.database.guide/advantages-and-disadvantages-of-indexes>
4. <https://www.javatpoint.com/system-testing>
5. <https://brilliant.org/wiki/secure-hashing-algorithms>