TITLE OF PROJECT

CRIMINAL RECORD MANAGEMENT SYSTEM

A PROJECT REPORT

CRIMINAL RECORD MANAGEMENT SYSTEM

Submitted By

Tanisha Jain(24MCI10047)

Jaspreet Kaur(24MCI10062)

Name of Candidate

Tanisha Jain

Jaspreet Kaur

In partial fulfilment for the award of the degree of

Masters of Computers Application (AI&ML)

Name of the degree

Masters of Computers Application (AI&ML)

IN

Branch of Study

MCA(AI&ML)

Chandigarh University

Month & Year

October&2024-25

BONAFIDE CERTIFICATE

Certified that this project report "Criminal Record Management System **TITLE OF THE PROJECT** "Criminal Record Manag is the bonafide work of "**NAME OF THE CANDIDATE**(S)Tanisha Jain & Jaspreet Kaur who carried out the project work under my/our supervision.

<<Signature of HOD>> <<Signature of Supervisor>>

Signature **Signature**

<<Name of the head of the Department>> <<Name>>

Head of Department **<<Academic Designer>>**

<<Department>> << Department>>

Submitted for the project viva – voice examination held on

Internal Examiner **External examiner**

Table of Contents

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this **Criminal Record Management System** project. First and foremost, I am thankful to my mentor Dr. Deeksha Baweja for their invaluable guidance, encouragement, and constructive feedback throughout the development of this project.

I am also deeply grateful to my institution, Chandigarh University, Ghauran Mohali, for providing the necessary resources and support to carry out this project. The knowledge and insights gained during this academic journey have greatly enhanced my skills and understanding of criminal record management and system implementation.

Finally, I would like to acknowledge the developers and contributors of the various libraries and frameworks, such as Python and Tkinter, that were instrumental in the development of this system. Without their work, this project would not have been possible.

ABSTRACT

The Criminal Record Management System is a software application developed to streamline and automate the process of managing criminal records for law enforcement agencies. The system is designed to store, retrieve, and update criminal records efficiently, providing an easy-to-use interface for authorized personnel to access critical information. Built using Python's Tkinter for the graphical user interface, the system includes authentication features to ensure that only authorized users can interact with the data, maintaining security and privacy.

The system allows users to input detailed records of individuals involved in criminal activities, including personal information, criminal history, and case status. It supports functionalities such as adding, editing, and deleting records, along with searching and filtering data to facilitate quick access to relevant information. The implementation focuses on providing an intuitive, user-friendly interface with robust authentication mechanisms to prevent unauthorized access.

This project aims to replace traditional paper-based systems, reduce human error, and enhance the efficiency of law enforcement operations. By integrating secure login features and easy navigation, the system ensures reliable and quick access to important data, ultimately improving the management of criminal records.

Chapter -1

i. Introduction

In today's world, the management of criminal records is a crucial task for law enforcement agencies to ensure public safety and effective judicial processes. Traditional methods of managing criminal records, such as paper-based systems, are often inefficient, prone to errors, and difficult to maintain. With the increasing complexity of cases and the need for secure and accessible data, it has become essential to digitize and streamline the record-keeping process.

The Criminal Record Management System is a software solution designed to address these challenges by automating the storage, retrieval, and management of criminal records. This system allows law enforcement personnel to record, update, and access detailed information about criminals and their offenses, including personal details, criminal history, and case status. The system enhances the efficiency of managing criminal records and ensures data accuracy while maintaining strict security measures to prevent unauthorized access.

Developed using Python's Tkinter for the graphical user interface (GUI), the system integrates secure authentication to protect sensitive data and ensure that only authorized users can access the records. This modern approach to criminal record management not only improves data accessibility but also reduces the time spent on manual data handling, minimizing the risk of data loss or errors.

ii. OBJECTIVE

The primary objective of the Criminal Record Management System is to develop a secure, efficient, and user-friendly software application to manage and maintain criminal records. This system aims to replace traditional manual methods with an automated platform that enhances data accuracy, reduces errors, and improves access to vital information for law enforcement personnel. The specific objectives include:

1. Efficient Data Management: To enable the systematic storage, retrieval, and update of criminal records, ensuring quick and easy access to relevant information.

2. Enhanced Security: To implement robust authentication features to restrict access to authorized users only, ensuring the confidentiality and integrity of sensitive criminal records.

3. Error Reduction: To minimize human errors associated with manual data entry and record management through automated processes.

4. Improved Accessibility: To allow law enforcement personnel to search, filter, and view criminal records based on various criteria, ensuring efficient data

5. Scalability and Maintenance: To provide a flexible system that can handle a growing number of criminal records while being easy to maintain and update as needed.

6. User-Friendly Interface: To develop an intuitive and easy-to-navigate interface using Tkinter, ensuring that the system can be effectively used by law enforcement personnel without requiring extensive technical knowledge.

iii. SCOPE OF THE PROJECT

The Criminal Record Management System is designed to automate the storage, retrieval, and management of criminal records, providing law enforcement agencies with a comprehensive and secure solution for handling sensitive data. The scope of this project covers the following key areas:

1. Record Management:

- The system will allow authorized users to add, edit, delete, and update criminal records, including personal information, crime details, case status, and criminal history.

- It will support the storage of various types of records such as personal identification details, fingerprints, photos, and details about criminal activities.

2. User Authentication and Security:

- The system will include robust authentication mechanisms to ensure that only authorized users (e.g., law enforcement officers) can access and manage the records.

- Secure login features with password protection will be implemented to maintain the confidentiality of the criminal data.

3. Data Retrieval and Search Capabilities:

- The system will provide an easy-to-use interface to search and filter criminal records based on multiple parameters such as name, crime type, or case status.

- The search function will allow for efficient data retrieval, reducing time spent on manual searches.

4. Reporting and Analysis:

- The system will enable users to generate basic reports summarizing criminal data, case statuses, and other relevant statistics.

- This feature will assist law enforcement agencies in decision-making and trend analysis.

5. Scalability:

- The system is designed to handle a large volume of records and can be scaled to accommodate growing data needs.

- Future enhancements may include the integration of advanced features like biometric data management, case tracking, and integration with national databases.

6. User Interface:

- A simple and user-friendly interface will be developed using Python's Tkinter, ensuring ease of use for non-technical users such as law enforcement officers.

- The system will feature intuitive navigation, making it easy for users to perform key functions without extensive training.

7. Maintenance and Support:

- The system will be designed to be easily maintainable, with the ability to update or modify features as required.

- Documentation will be provided to support future enhancements and troubleshooting.

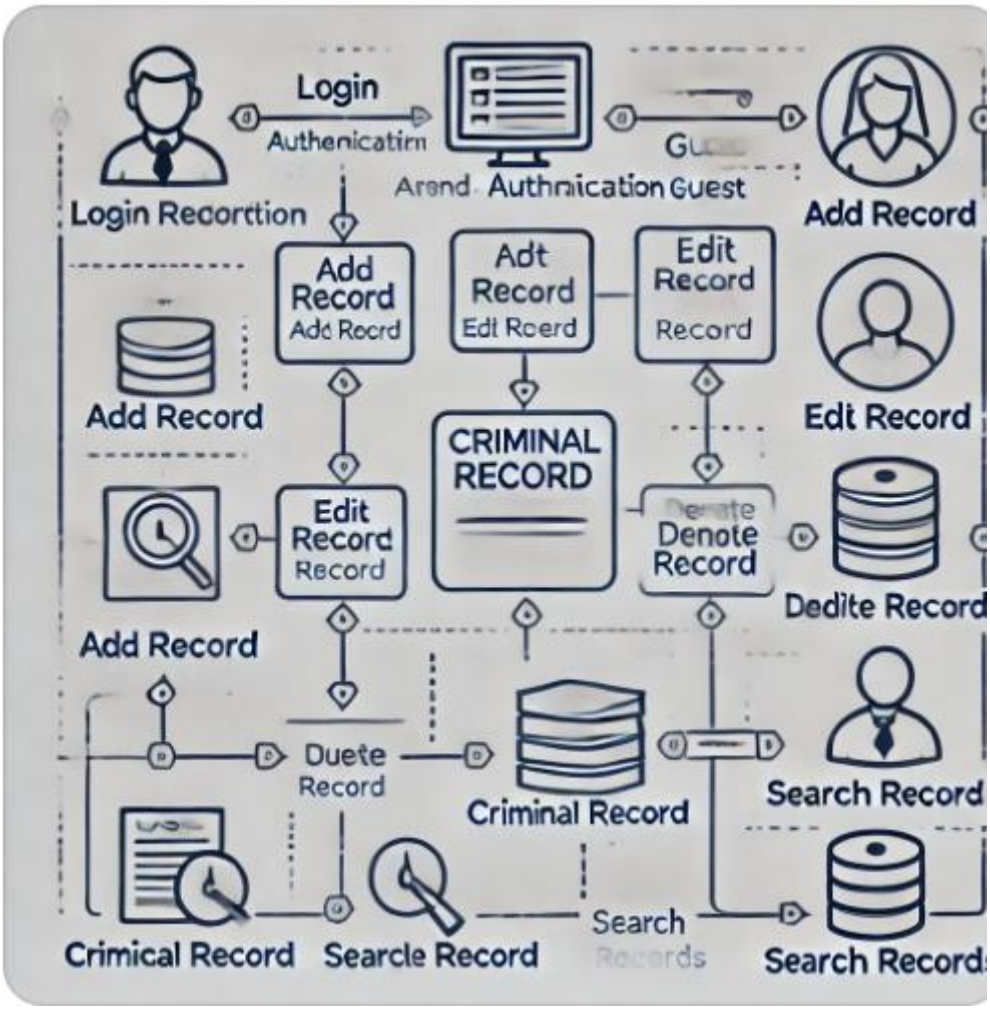
CHAPTER-2- DESIGNING

UML (UNIFIED MODELLING LANGUAGE)

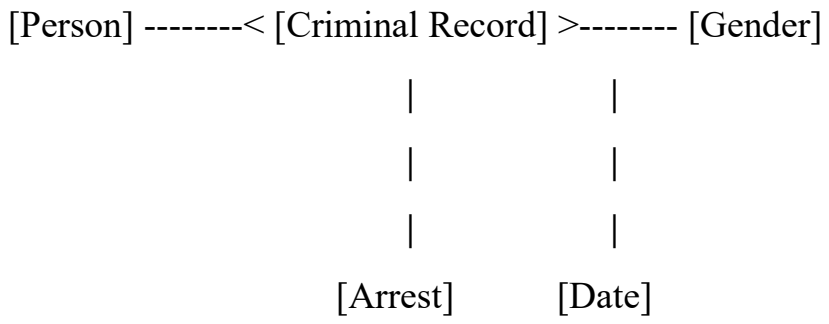
To create a **UML (Unified Modeling Language)** diagram for your **Criminal Record Management System**, there are several types of UML diagrams you can consider based on what aspect of the system you want to represent. Commonly used diagrams for such systems include

1. **Use Case Diagram** – Illustrates the interactions between users (actors) and the system's functions.
2. **Class Diagram** – Shows the structure of the system, the different classes, their attributes, methods, and relationships.
3. **Activity Diagram** – Describes the flow of activities within the system.
4. **Sequence Diagram** – Depicts the sequence of interactions between objects over time for a particular functionality.

Data Flow Diagram



ER DIAGRAM



CHAPTER -3

CODING AND SCREENSHOTS

```
import tkinter as tk
from tkinter import messagebox, simpledialog, ttk
import sqlite3

class CriminalRecordManagement:
    def __init__(self, root):
        self.root = root
        self.root.title("Criminal Record Management System")
        self.root.configure(bg='skyblue')

        # Database connection
        self.conn = sqlite3.connect("criminal_records.db")  # Create/connect to the
SQLite database
        self.cursor = self.conn.cursor()

        # Create table if not exists
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS records (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            name TEXT NOT NULL,
                            age INTEGER NOT NULL,
                            gender TEXT NOT NULL,
                            crime TEXT NOT NULL,
                            year TEXT NOT NULL,
                        description TEXT NOT NULL)''')
self.conn.commit()
```

```
# UI elements...
self.records = []
self.is_authenticated = False

self.label = tk.Label(root, text="Criminal Record Management", font=("Times
New Roman", 18), bg='white')
self.label.pack(pady=10)

self.tree = ttk.Treeview(root, columns=("Name", "Age", "Gender", "Crime",
"Year", "Description"), show='headings')
self.tree.heading("Name", text="Name")
self.tree = ttk.Treeview(root, columns=("Name", "Age", "Gender", "Crime",
"Year", "Description"), show='headings')
        self.tree.heading("Name", text="Name")
        self.tree.heading("Age", text="Age")
        self.tree.heading("Gender", text="Gender")
        self.tree.heading("Crime", text="Crime")
        self.tree.heading("Year", text="Year")
        self.tree.heading("Description", text="Description")
        self.tree.pack(pady=10)

        self.add_button = tk.Button(root, text="Add Record",
command=self.add_record, font=("Times New Roman", 12), bg='white')
        self.add_button.pack(pady=5)

        self.view_button = tk.Button(root, text="View Records",
command=self.view_records, font=("Times New Roman", 12), bg='white')
        self.view_button.pack(pady=5)

        self.delete_button = tk.Button(root, text="Delete Record",
command=self.delete_record, font=("Times New Roman", 12), bg='white')
        self.delete_button.pack(pady=5)

        self.search_button = tk.Button(root, text="Search Record",
command=self.search_record, font=("Times New Roman", 12), bg='white')
        self.search_button.pack(pady=5)

        self.logout_button = tk.Button(root, text="Logout", command=self.logout,
```

```
font=("Times New Roman", 12), bg='white')
        self.logout_button.pack(pady=5)

    def authenticate(self):
        username = simpledialog.askstring("Username", "Enter your username:")
        password = simpledialog.askstring("Password", "Enter your password:",
show='*')

        if username == "jaspreet" and password == "tanisha":
            self.is_authenticated = True
            messagebox.showinfo("Success", "Login successful!")
        else:
            messagebox.showerror("Error", "Invalid username or password.")

    def add_record(self):
        if not self.is_authenticated:
            messagebox.showwarning("Warning", "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring("Input", "Enter Criminal Name:")
        if name:
            age = simpledialog.askinteger("Input", "Enter Age:")
            gender = simpledialog.askstring("Input", "Enter Gender:")
            crime = simpledialog.askstring("Input", "Enter Crime:")
            year = simpledialog.askstring("Input", "Enter Year of Crime:")
            description = simpledialog.askstring("Input", "Enter Crime
Description:")

            self.cursor.execute('''INSERT INTO records (name, age, gender, crime,
year, description)
                                   VALUES (?, ?, ?, ?, ?, ?)''',
                                (name, age, gender, crime, year, description))
            self.conn.commit()
            self.view_records()  # Refresh the view after adding
            messagebox.showinfo("Success", "Record Added Successfully")

    def view_records(self):
        if not self.is_authenticated:
            messagebox.showwarning("Warning", "Please login first.")
```

```
            self.authenticate()
            return

        for item in self.tree.get_children():
            self.tree.delete(item)

        self.cursor.execute('''SELECT name, age, gender, crime, year, description
FROM records''')
        records = self.cursor.fetchall()

        for record in records:
            self.tree.insert("", tk.END, values=record)

    def delete_record(self):
        if not self.is_authenticated:
            messagebox.showwarning("Warning", "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring("Input", "Enter Criminal Name to
Delete:")
        if name:
            self.cursor.execute('''DELETE FROM records WHERE name = ?''',
(name,))
            self.conn.commit()
            self.view_records()
            messagebox.showinfo("Success", "Record Deleted Successfully")

    def search_record(self):
        if not self.is_authenticated:
            messagebox.showwarning("Warning", "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring("Input", "Enter Criminal Name to
Search:")
        if name:
            self.cursor.execute('''SELECT * FROM records WHERE name = ?''',
(name,))
            record = self.cursor.fetchone()
```

```
        if record:
            messagebox.showinfo("Record Found",
                f"Name: {record[1]}\nAge: {record[2]}\nGender: {record[3]}\n"
                f"Crime: {record[4]}\nYear: {record[5]}\nDescription:
{record[6]}"
            )
        else:
            messagebox.showwarning("Warning", "Record not found.")

    def logout(self):
        self.is_authenticated = False
        self.tree.delete(*self.tree.get_children())  # Clear the table on logout
        messagebox.showinfo("Logout", "You have been logged out.")
        self.conn.close()

if __name__ == "__main__":
    root = tk.Tk()
    app = CriminalRecordManagement(root)
    app.authenticate()  # Prompt for authentication at startup
    root.mainloop()
```

Process

```
import tkinter as tk
from tkinter import messagebox, simpledialog, ttk
import sqlite3

class CriminalRecordManagement:
    def __init__(self, root):
        self.root = root
        self.root.title("Criminal Record Management System")
        self.root.configure(bg='skyblue')

        # Database connection
        self.conn = sqlite3.connect("criminal_records.db")  # Create/connect to the SQLite database
        self.cursor = self.conn.cursor()

        # Create table if not exists
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS records (
                               id INTEGER PRIMARY KEY AUTOINCREMENT,
                               name TEXT NOT NULL,
                               age INTEGER NOT NULL,
                               gender TEXT NOT NULL,
                               crime TEXT NOT NULL,
                               year TEXT NOT NULL,
                               description TEXT NOT NULL)''')
        self.conn.commit()

        # UI elements...
        self.records = []
        self.is_authenticated = False

        self.label = tk.Label(root, text="Criminal Record Management", font=("Times New Roman", 18), bg='white')
        self.label.pack(pady=10)
```

```
    self.label = tk.Label(root, text="Criminal Record Management", font=("Times New Roman", 18), bg='white')
    self.label.pack(pady=10)

    self.tree = ttk.Treeview(root, columns=("Name", "Age", "Gender", "Crime", "Year", "Description"), show='headings')
    self.tree.heading("Name", text="Name")
    self.tree.heading("Age", text="Age")
    self.tree.heading("Gender", text="Gender")
    self.tree.heading("Crime", text="Crime")
    self.tree.heading("Year", text="Year")
    self.tree.heading("Description", text="Description")
    self.tree.pack(pady=10)

    self.add_button = tk.Button(root, text="Add Record", command=self.add_record, font=("Times New Roman", 12), bg='white')
    self.add_button.pack(pady=5)

    self.view_button = tk.Button(root, text="View Records", command=self.view_records, font=("Times New Roman", 12), bg='white')
    self.view_button.pack(pady=5)

    self.delete_button = tk.Button(root, text="Delete Record", command=self.delete_record, font=("Times New Roman", 12), bg='white')
    self.delete_button.pack(pady=5)

    self.search_button = tk.Button(root, text="Search Record", command=self.search_record, font=("Times New Roman", 12), bg='white')
    self.search_button.pack(pady=5)

    self.logout_button = tk.Button(root, text="Logout", command=self.logout, font=("Times New Roman", 12), bg='white')
    self.logout_button.pack(pady=5)

def authenticate(self):
    username = simpledialog.askstring( title: "Username",  prompt: "Enter your username:")
    password = simpledialog.askstring( title: "Password",  prompt: "Enter your password:", show='*')
```

```
        if username == "jaspreet" and password == "tanisha":
            self.is_authenticated = True
            messagebox.showinfo( title: "Success",  message: "Login successful!")
        else:
            messagebox.showerror( title: "Error",  message: "Invalid username or password.")

    def add_record(self):
        if not self.is_authenticated:
            messagebox.showwarning( title: "Warning",  message: "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring( title: "Input",  prompt: "Enter Criminal Name:")
        if name:
            age = simpledialog.askinteger( title: "Input",  prompt: "Enter Age:")
            gender = simpledialog.askstring( title: "Input",  prompt: "Enter Gender:")
            crime = simpledialog.askstring( title: "Input",  prompt: "Enter Crime:")
            year = simpledialog.askstring( title: "Input",  prompt: "Enter Year of Crime:")
            description = simpledialog.askstring( title: "Input",  prompt: "Enter Crime Description:")

            self.cursor.execute( __sql: '''INSERT INTO records (name, age, gender, crime, year, description)
                                VALUES (?, ?, ?, ?, ?, ?)''',
                                __parameters: (name, age, gender, crime, year, description))
            self.conn.commit()
            self.view_records()  # Refresh the view after adding
            messagebox.showinfo( title: "Success",  message: "Record Added Successfully")

    def view_records(self):
        if not self.is_authenticated:
            messagebox.showwarning( title: "Warning",  message: "Please login first.")
rdManagement › view_records() › if not self.is_authenticated
```

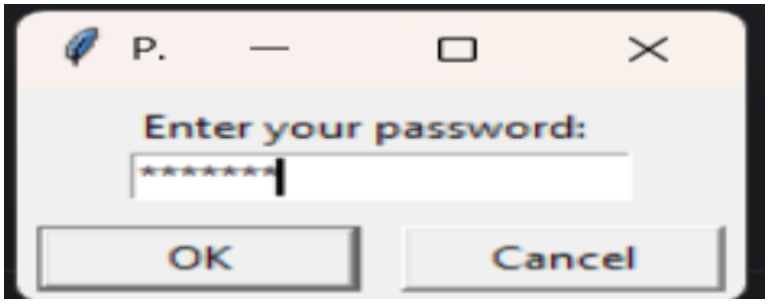
```
            return

        for item in self.tree.get_children():
            self.tree.delete(item)

        self.cursor.execute('''SELECT name, age, gender, crime, year, description FROM records''')
        records = self.cursor.fetchall()

        for record in records:
            self.tree.insert( parent: "", tk.END, values=record)

    def delete_record(self):
        if not self.is_authenticated:
            messagebox.showwarning( title: "Warning",  message: "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring( title: "Input",  prompt: "Enter Criminal Name to Delete:")
        if name:
            self.cursor.execute( __sql: '''DELETE FROM records WHERE name = ?''',  __parameters: (name,))
            self.conn.commit()
            self.view_records()
            messagebox.showinfo( title: "Success",  message: "Record Deleted Successfully")

    def search_record(self):
        if not self.is_authenticated:
            messagebox.showwarning( title: "Warning",  message: "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring( title: "Input",  prompt: "Enter Criminal Name to Search:")
```

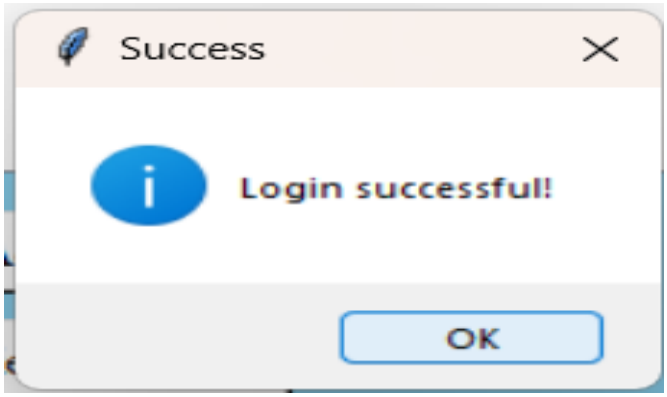
```
            return

        for item in self.tree.get_children():
            self.tree.delete(item)

        self.cursor.execute('''SELECT name, age, gender, crime, year, description FROM records''')
        records = self.cursor.fetchall()

        for record in records:
            self.tree.insert( parent: "", tk.END, values=record)

    def delete_record(self):
        if not self.is_authenticated:
            messagebox.showwarning( title: "Warning",  message: "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring( title: "Input",  prompt: "Enter Criminal Name to Delete:")
        if name:
            self.cursor.execute( __sql: '''DELETE FROM records WHERE name = ?''',  __parameters: (name,))
            self.conn.commit()
            self.view_records()
            messagebox.showinfo( title: "Success",  message: "Record Deleted Successfully")

    def search_record(self):
        if not self.is_authenticated:
            messagebox.showwarning( title: "Warning",  message: "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring( title: "Input",  prompt: "Enter Criminal Name to Search:")
```

```
        if not self.is_authenticated:
            messagebox.showwarning( title: "Warning",  message: "Please login first.")
            self.authenticate()
            return

        name = simpledialog.askstring( title: "Input",  prompt: "Enter Criminal Name to Search:")
        if name:
            self.cursor.execute( __sql: '''SELECT * FROM records WHERE name = ?''',  __parameters: (name,))
            record = self.cursor.fetchone()

            if record:
                messagebox.showinfo( title: "Record Found",
                    message: f"Name: {record[1]}\nAge: {record[2]}\nGender: {record[3]}\n"
                    f"Crime: {record[4]}\nYear: {record[5]}\nDescription: {record[6]}"
                )
            else:
                messagebox.showwarning( title: "Warning",  message: "Record not found.")

    def logout(self):
        self.is_authenticated = False
        self.tree.delete(*self.tree.get_children())  # Clear the table on logout
        messagebox.showinfo( title: "Logout",  message: "You have been logged out.")
        self.conn.close()

if __name__ == "__main__":
    root = tk.Tk()
    app = CriminalRecordManagement(root)
    app.authenticate()  # Prompt for authentication at startup
    root.mainloop()
```

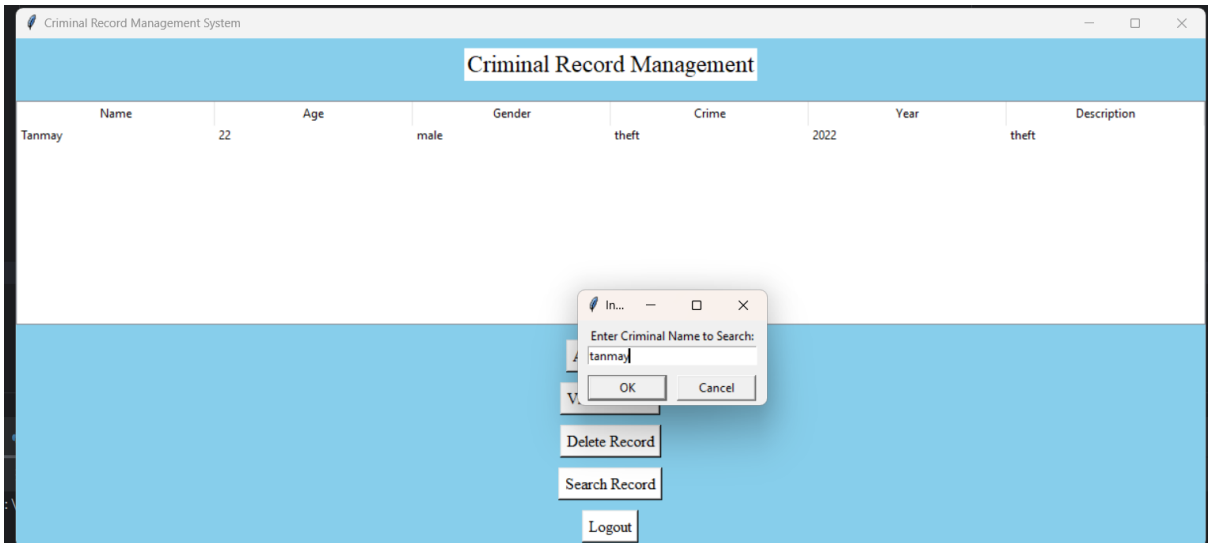
Output.

Display

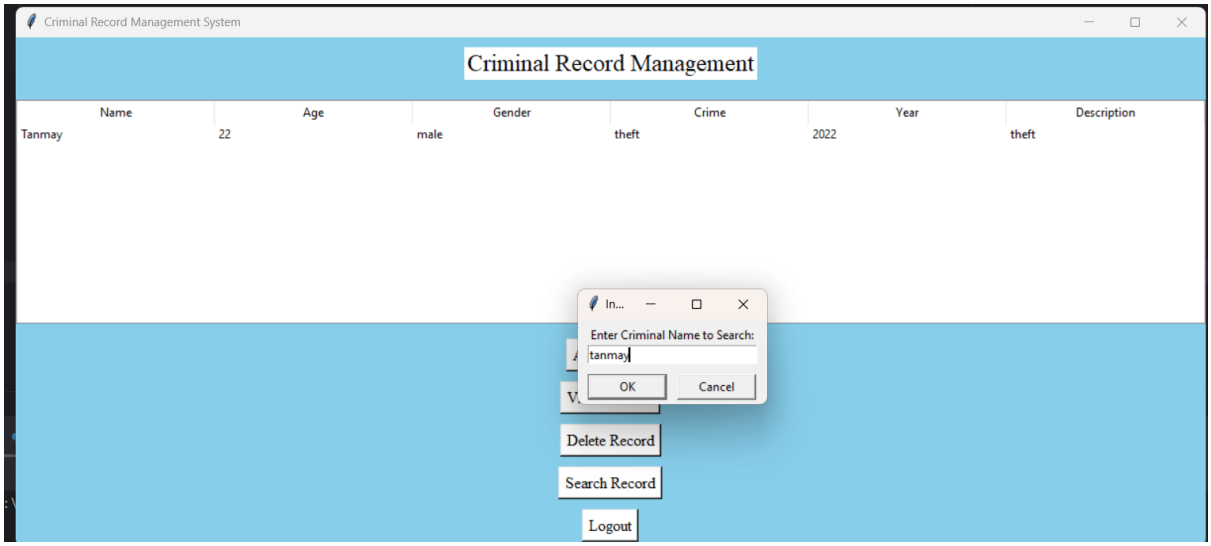




Add the details:

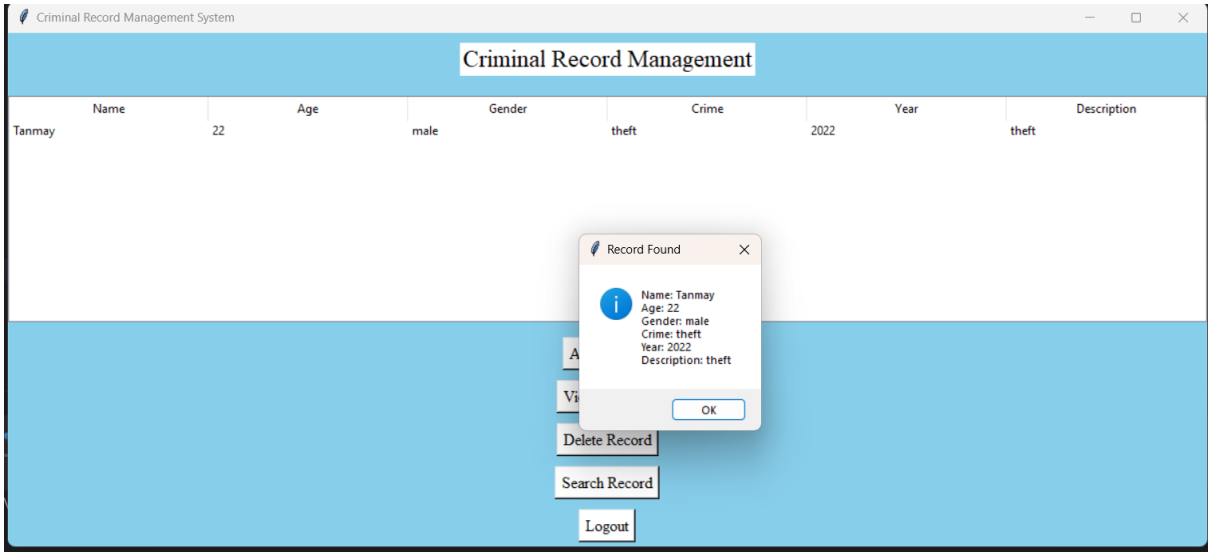


View the record:



Delete the record

Search the Record



Chapter-4

Conculsion

The Criminal Record Management System has been developed to provide a robust, efficient, and secure solution for managing criminal records. This system automates the manual process of handling criminal data, making it easier for law enforcement agencies to store, update, and retrieve information. By incorporating features such as user authentication, record management, and search functionality, the system significantly enhances the accuracy, security, and accessibility of sensitive data.

Through the use of Python and Tkinter, the system offers a user-friendly interface, ensuring that both technical and non-technical users can navigate the platform easily. The project has achieved its objective of improving operational efficiency, minimizing human error, and providing a scalable framework that can handle the increasing volume of criminal records.