RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

CRIMINAL DETECTOR MANAGEMENT SYSTEM

A MINI PROJECT REPORT

SUBMITTED BY

SANDHYA SREE M	**(231801146)**
RITHIKA SMITHI S	**(231801138)**
RAGHUL S	**(231801131)**

In partial fulfillment for the award of the degree

BACHELOR OF ENGINEERING

IN

ARTIFICIAL INTELLIGENCE OF DATA SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2024 - 2025

BONAFIDE CERTIFICATE

Certified that this project report "**CRIME DETECTOR MANAGEMENT**" is the Bonafide work of **"SANDHYASREE M(231801146), RITHIKA SMITHI S (231801138), RAGHUL S(231801131)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on ________________________________

Signature	**Signature**
Dr. GNANASEKAR J M **Head of the Department, Artificial intelligence and data Science,Rajalakshmi Engineering College (Autonomous),Chennai-602105**	**Dr. MANORANJINI J** **Assoc.Professor, Artificial Intelligence and Data Science, Rajalakshmi Engineering College (Autonomous), Thandalam, Chennai-602105**
INTERNAL EXAMINER	**EXTERNAL EXAMINER**

TABLE OF CONTENT

TABLE OF FIGURES

ABSTRACT

The proposed system focuses on Crime Records Management for all police stations across the country. It aims to centralize Information Management in Crime for efficient sharing of critical information across all stations. The system will initially be implemented across cities and towns, but will eventually be interlinked to allow police detectives to access information across all records in the state

1.INTRODUCTION:

Develop a mini project that aims to create a Crime Detection System using SQL Server to manage and analyze crime data efficiently. The system will help law enforcement agencies, researchers, and the general public to access, report, and analyze crime-related information in a user-friendly manner.

Objectives of the Supermarket Management System

DATA ENTRY:

The importance of user-friendly data entry interfaces, standardized input formats, validation rules, automated data collection methods, and access controls to ensure data integrity and security, while also reducing manual errors and ensuring only authorized personnel can access the system.

DATA VISUALIZATION:

The system includes dynamic dashboards for real-time crime trends, geospatial mapping for resource allocation, customizable reports, and various visualization techniques. It also includes user training to ensure accurate interpretation of data and efficient use of visualization tools.

TESTING AND QUALITY ASSURANCE:

The system's testing process includes unit testing, integration testing, performance testing, user acceptance testing, and security testing. Quality assurance is ensured through clear standards, regular audits, robust error handling mechanisms, and a culture of continuous improvement. Regular audits ensure compliance with established standards, while error handling mechanisms track and resolve issues. A comprehensive documentation of system processes, data standards, and testing protocols facilitates onboarding of new users.

MODULES:

A crime detector management system(CDMS) requires a robust database to efficiently store and manage various types of data. Here are some essential modules that you can consider incorporating into your CDMS using a database management system:

Core Modules:

- Searching a crime
- Crime report

Additional Modules:

- Community Engagement
- Integrations

SURVEY OF TECHNOLOGIES:

2.1 Software Description :

This project utilizes a combination of software tools to create a comprehensive and efficient supermarket management system:

- **Database Management System (DBMS):** MySQL is chosen as the DBMS for its reliability, performance, and widespread use in various applications.
- **Integrated Development Environment (IDE):** PyCharm is selected as the IDE for its Python-specific features, code completion, debugging tools, and seamless integration with MySQL.
- **Visual studio :** HTML,CSS ,JavaScript.

2.2 Languages :

- **SQL:** Structured Query Language is used to interact with the MySQL database, defining data structures, performing queries, and managing data integrity.
- **Python**: A versatile programming language is used to develop the backend logic, implement business rules, and interact with the MySQL database through the SQLAlchemy ORM.

2.2.1 SQL :

SQL plays a crucial role in the supermarket management system by:

- Creating database tables: Defining the structure of tables to store product information, customer details, sales transactions, employee data, and more.
- Performing queries: Retrieving, updating, inserting, and deleting data from the database tables based on specific criteria.
- Managing data integrity: Ensuring data consistency and accuracy through constraints like primary keys, foreign keys, and data types.
- Generating reports: Creating customized reports based on SQL queries to analyze sales trends, inventory levels, customer behavior, and other key metrics.

2.2.2 Python :

- Develop the backend logic: Implementing business rules, calculations, and data processing tasks.
- Interact with the MySQL database: Using the SQLAlchemy Object-Relational Mapper (ORM) to map Python objects to database tables, simplifying data access and manipulation.
- Create the user interface: Building the web interface using Flask, allowing users to interact with the system and perform various tasks.
- Integrate with other systems: Connecting the supermarket management system with external systems like accounting software or inventory management tools.

3.REQUIREMENTS AND ANALYSIS :

3.1 Requirement Specification

When developing a Crime Detection Management System, it's essential to conduct a thorough requirements analysis to ensure the system meets the needs of law enforcement agencies, stakeholders, and the community. This section outlines the key requirements, categorized into functional, non-functional, and technical requirements.

Functional Requirements :

- **Incident Management:**
    - Incident Reporting
    - Incident Status Tracking
- **Suspect and Witness Management:**
    - Profile Management
    - Search and Filter
- **Evidence Management**
    - Evidence Entry
    - Chain of Custody
- **User Management**
    - Role-Based Access Control
    - User Profiles

Non-Functional Requirements:

- User-Friendly Interface
- Training and Documentation

Performance

- Response Time
- Scalability

Security

- Data Protection
- Access Control

Reliability

- System Availability
- Data Backup and Recovery

3. Technical Requirements

Database Management System

- RDBMS Selection

B. Development Environment

- **Programming Languages**: Using appropriate programming languages for front-end (JavaScript, HTML/CSS) and back-end (Python, Java).

Hosting and Infrastructure

- **Cloud Services**: Evaluate cloud platforms (e.g., AWS, Azure) for hosting the application and database.
- **Network Infrastructure**: Ensure adequate network bandwidth and security measures for data transmission.

3.2 Hardware and Software Requirements :

Hardware Requirements

Server Specifications:

- **Processor:** Quad-core (e.g., Intel Xeon or AMD Ryzen) minimum; multi-core recommended.
- **RAM:** 16 GB minimum; 32 GB recommended for larger datasets.
- **Storage:** 500 GB SSD minimum; 1 TB SSD recommended with additional HDD for backups.
- **Network Interface:** Gigabit Ethernet.

Client Workstations:

- **Processor:** Dual-core minimum; quad-core recommended.
- **RAM:** 8 GB minimum; 16 GB recommended.
- **Storage:** 256 GB SSD minimum; 512 GB SSD recommended.
- **Display:** 15-inch monitor (1080p minimum); dual monitors recommended.

Software Requirements

Operating System:

- **Server:** Windows Server.
- **Client:** Windows 10/11

Database Management System (DBMS):

- **Options:** MySQL

Development Tools:

- **Languages:** Backend (Python). Frontend (JavaScript with frameworks like React or Angular).
- **IDE:** Visual Studio Code.

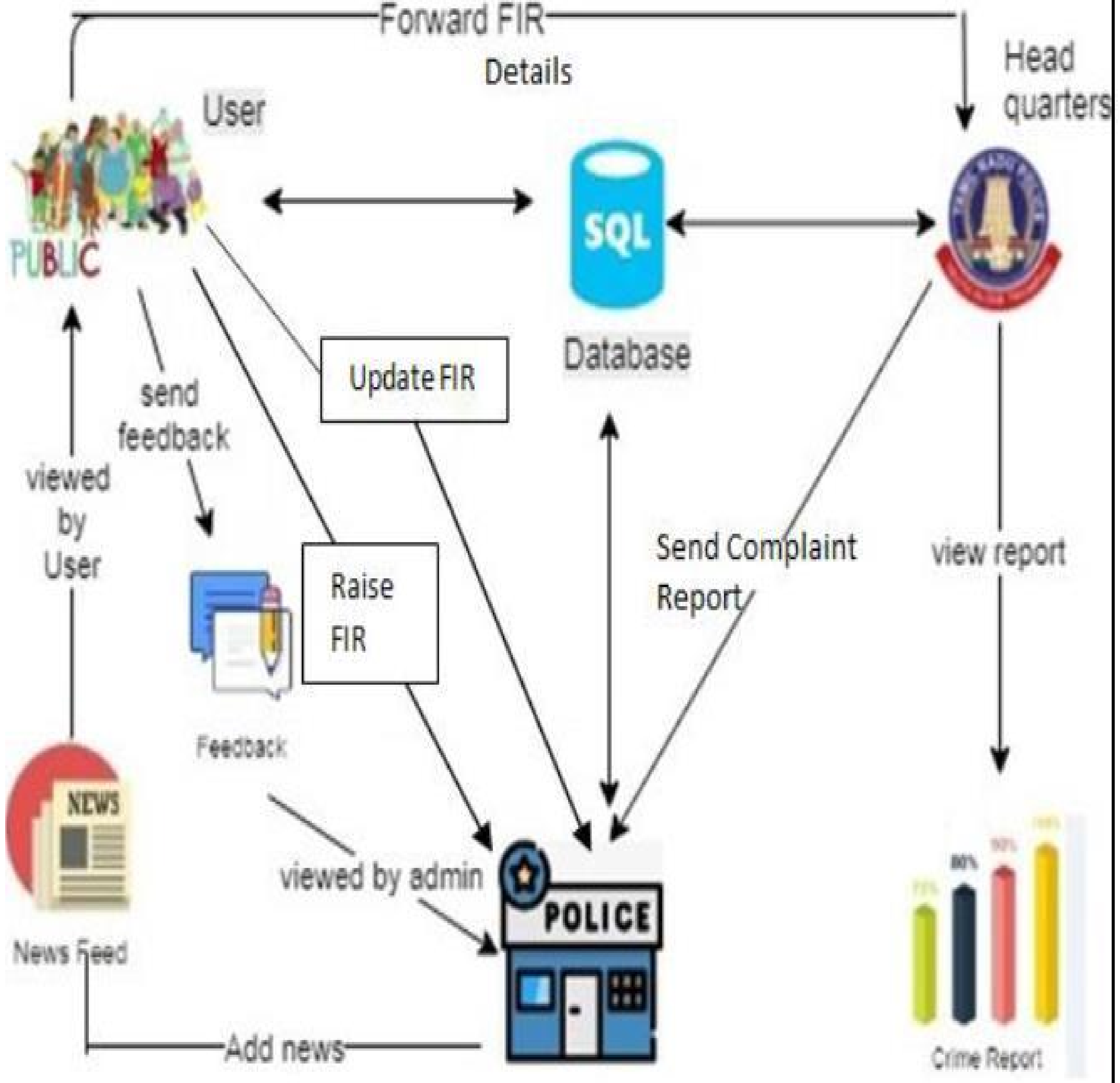
Web Server:

- **Options:** Apache HTTP Server or Nginx.

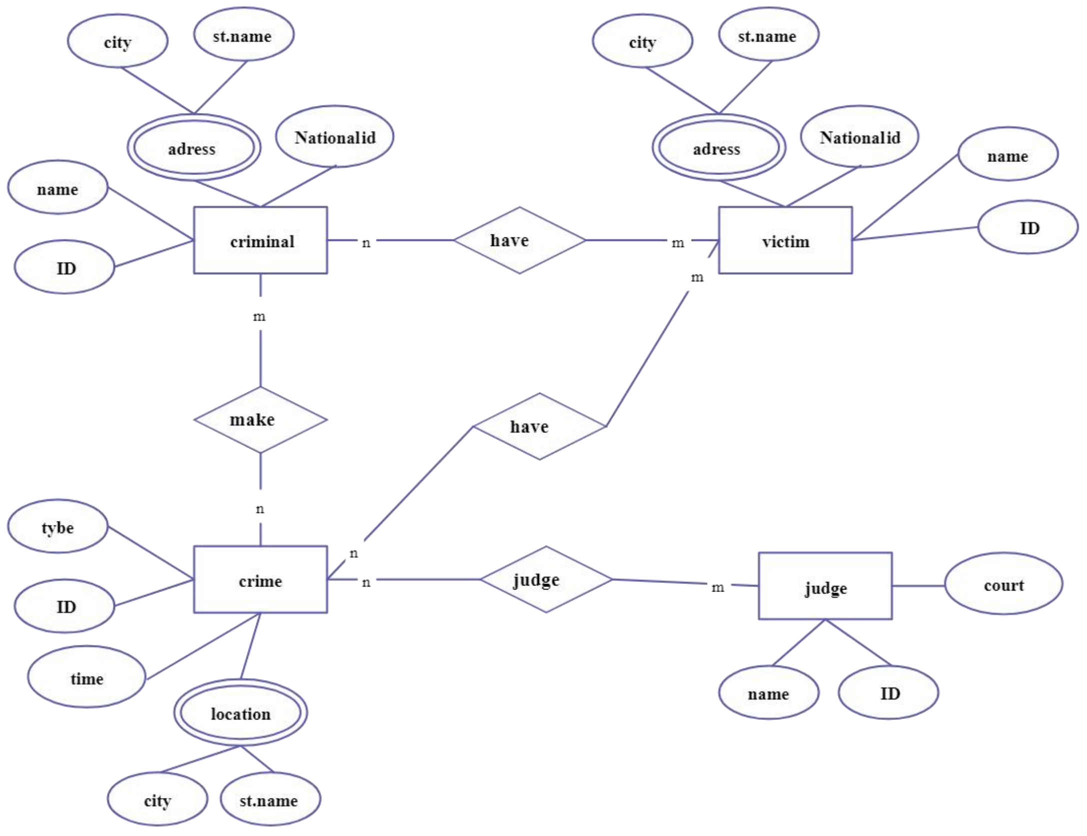
Security Software:

- **Firewall:** Hardware/software firewall.
- **Antivirus:** Reliable endpoint protection.
- **Encryption Tools:** SSL/TLS for data security.

3.3 ARCHITECTURE DIAGRAM :



3.4 ER DIAGRAM :



3.5 NORMALIZATION :

Step 1: Initial Unnormalized Data (UNF)

Before any normalization, you might have data stored in an unnormalized form. In this case, you may have a single table that contains repeating groups and redundancy.

Example (Unnormalized Table):

Crime_ID	Crime_Type	Location	Suspect_Name	Suspect_History	Crime_Date	Crime_Time	Description
1	Theft	Downtown, City A	John Doe	Previous Theft	2024-11-01	14:00	Stolen wallet
2	Assault	Downtown, City A	Jane Smith	No History	2024-11-02	18:00	Physical assault in alley
3	Theft	Uptown, City B	John Doe	Previous Theft	2024-11-03	12:00	Stolen car
4	Fraud	Downtown, City A	Sarah Lee	Previous Fraud	2024-11-04	09:30	Credit card fraud

This table contains redundant information such as repeated location and suspect details, which leads to data anomalies (e.g., multiple entries for John Doe).

Step 2: First Normal Form (1NF)

To satisfy **1NF**, we need to:

- Ensure that each column contains only atomic values (i.e., no repeating groups or arrays).
- Make sure there are no duplicate records.

1NF Table:

We achieve 1NF by splitting out repeating groups and ensuring that each field contains only atomic (indivisible) values.

Crime_ID	**Crime_Type**	**Location_ID**	**Suspect_ID**	**Crime_Date**	**Crime_Time**	**Description**
1	Theft	1	1	2024-11-01	14:00	Stolen wallet
2	Assault	1	2	2024-11-02	18:00	Physical assault in alley
3	Theft	2	1	2024-11-03	12:00	Stolen car
4	Fraud	1	3	2024-11-04	09:30	Credit card fraud

In **1NF**, we've broken down the **Location** and **Suspect** details into separate references using IDs (Location_ID, Suspect_ID), but this still introduces some redundancy because the suspect details (like name and history) are still not normalized.

Step 3: Second Normal Form (2NF)

To achieve **2NF**, we need to:

1. Be in **1NF**.
2. Remove **partial dependencies**—ensure that non-key attributes depend on the **entire** primary key. Specifically, for tables with a composite key (more than one field as the primary key), non-key attributes should depend on the full key.

In our case, the Crime_ID is the primary key in the **Crimes** table, but information such as **Suspect_Name** and **Suspect_History** depends on Suspect_ID, not Crime_ID. Thus, we separate the data into its own table for **Suspects** and **Locations**.

2NF Tables:

Crimes Table:

sql

Copy code

```
CREATE TABLE Crimes (
    Crime_ID INT PRIMARY KEY,
    Crime_Type VARCHAR(50),
    Location_ID INT,
    Suspect_ID INT,
    Crime_Date DATE,
    Crime_Time TIME,
    Description TEXT,
    FOREIGN KEY (Location_ID) REFERENCES Locations(Location_ID),
    FOREIGN KEY (Suspect_ID) REFERENCES Suspects(Suspect_ID)
);
```

Crime_ID	**Crime_Type**	**Location_ID**	**Suspect_ID**	**Crime_Date**	**Crime_Time**	**Description**
1	Theft	1	1	2024-11-01	14:00	Stolen wallet
2	Assault	1	2	2024-11-02	18:00	Physical assault in alley
3	Theft	2	1	2024-11-03	12:00	Stolen car
4	Fraud	1	3	2024-11-04	09:30	Credit card fraud

Source code app.py

```python
from flask import Flask, render_template, request, redirect, url_for, session, flash
import mysql.connector
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
app.secret_key = 'your_secret_key'  # Set a secure secret key

# Database connection function
def get_db_connection():
    return mysql.connector.connect(
        host='localhost',
        user='root',
        password='Sanjay@2005',
        database='CrimeManagementSystem'
    )

# Login page route
@app.route('/')
def login():
    return render_template('login.html')

# Handle login route
@app.route('/login', methods=['POST'])
def login_user():
    username = request.form['username']
    password = request.form['password']

    # Check user credentials
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute('SELECT * FROM Users WHERE username = %s', (username,))
    user = cursor.fetchone()
    conn.close()

    # Debugging: print user details to see if it's retrieved correctly
    print(f"User from database: {user}")

    if user:
            # Save user ID in session and redirect
            session['user_id'] = user['user_id']
            session['password']=user['password']
            print("Login successful, redirecting to dashboard...")
            return redirect(url_for('dashboard'))
    else:
        print("Invalid username.")
        flash('Invalid username. Please try again.', 'error')

    return redirect(url_for('login'))

# Dashboard route (after login)
@app.route('/dashboard')
def dashboard():
    if 'user_id' not in session:  # Check if user is logged in
        flash('Please log in to access the dashboard.', 'error')
        return redirect(url_for('login'))
    return render_template('dashboard.html')

# Report Crime route
@app.route('/report_crime', methods=['GET', 'POST'])
def report_crime():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        crime_type = request.form['crime_type']
        description = request.form['description']
        date_of_crime = request.form['date_of_crime']
        location = request.form['location']

        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute(
            'INSERT INTO CrimeReports (crime_type, description, date_of_crime,
location) VALUES (%s, %s, %s, %s)',
            (crime_type, description, date_of_crime, location)
        )
        conn.commit()
        conn.close()
        flash("Crime report submitted successfully.", 'success')
        return redirect(url_for('dashboard'))

    return render_template('report_crime.html')

# Crime Search route
@app.route('/search_crime', methods=['GET', 'POST'])
def search_crime():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        location = request.form['location']

        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute('SELECT * from CrimeSearchLogs')
        crimes = cursor.fetchall()
        conn.close()

        return render_template('crime_results.html', crimes=crimes)

    return render_template('search_crime.html')

# Logout route
@app.route('/logout')
def logout():
    session.pop('user_id', None)
    flash("You have been logged out.", 'info')
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)
```

Front end

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login - Crime Management System</title>

    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f7f6;
            color: #333;
            margin: 0;
            padding: 0;
        }

        .container {
            max-width: 400px;
            margin: 100px auto;
            background-color: #fff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        }

        input[type="text"], input[type="password"] {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            border-radius: 5px;
            border: 1px solid #ccc;
        }

        button {
            background-color: #5cb85c;
            border: none;
            padding: 10px 20px;
            color: white;
            border-radius: 5px;
            cursor: pointer;
            width: 100%;
        }

        button:hover {
            background-color: #4cae4c;
        }

        .alert-danger {
            background-color: #f8d7da;
            color: #721c24;
            padding: 10px;
            border-radius: 5px;
        }
    </style>

    <script>
        function validateLoginForm() {
            var username = document.getElementById("username").value;
            var password = document.getElementById("password").value;

            if (username === "" || password === "") {
                alert("Both username and password are required.");
                return false;
            }
            return true;
        }
    </script>
</head>
<body>
    <div class="container">
        <h2>Login</h2>

        <!-- Flash Message for Errors -->
        <!-- You can uncomment the below section for Flask flash message -->
        <!--
        {% with messages = get_flashed_messages() %}
            {% if messages %}
                <div class="alert-danger">
                    {% for message in messages %}
```

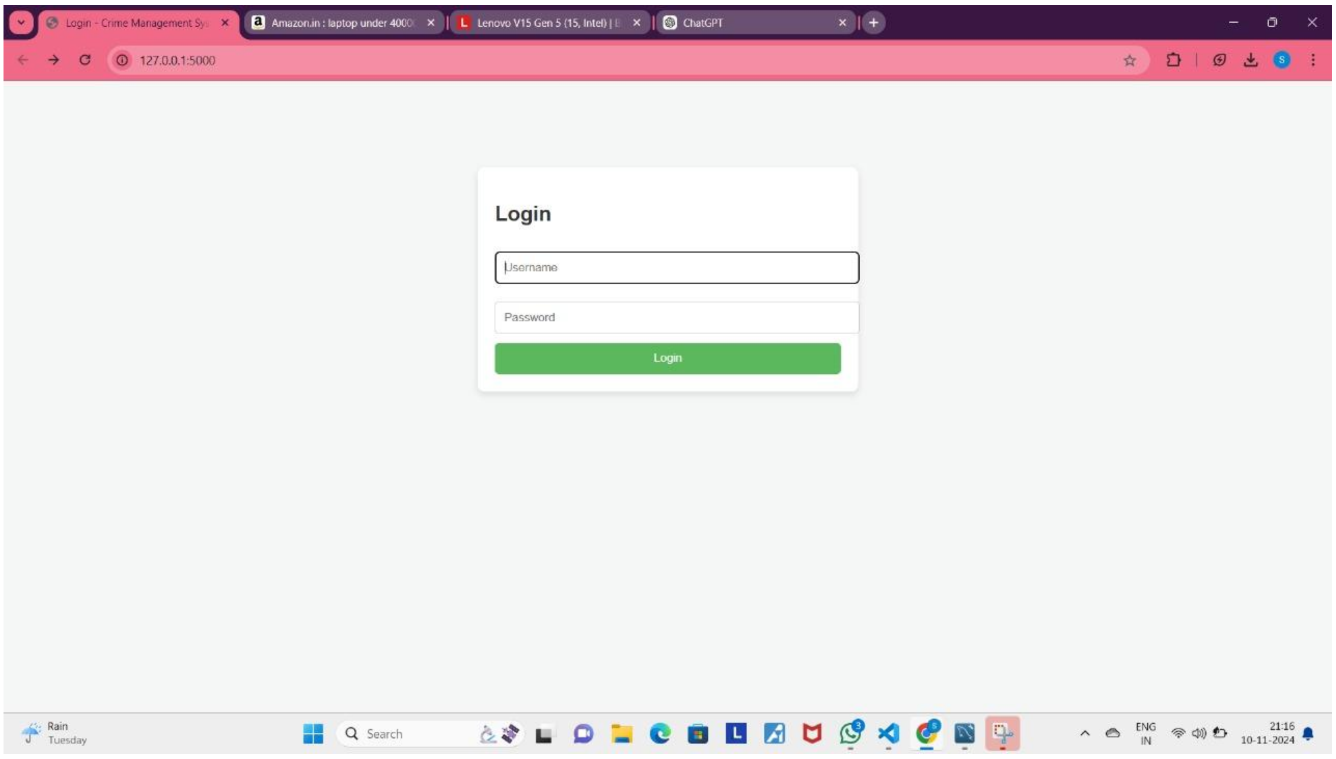
```
                        <p>{{ message }}</p>
                    {% endfor %}
                </div>
            {% endif %}
        {% endwith %}
        -->

        <form method="POST" action="/login" onsubmit="return validateLoginForm()">
            <input type="text" id="username" name="username" placeholder="Username"
required>
            <input type="password" id="password" name="password"
placeholder="Password" required>
            <button type="submit">Login</button>
        </form>
    </div>
</body>
</html>
```

RESULT:

A **Crime Management System (CMS)** in a **Database Management System (DBMS)** is designed to streamline the collection, storage, and retrieval of data related to criminal activities, law enforcement, and investigations. It typically includes modules for crime reporting, criminal information management, case tracking, investigation details, and court case management. The system stores crucial data such as crime reports, criminal records, evidence, witness statements, and case outcomes, which can be accessed and analyzed to aid in law enforcement and judicial processes. By organizing these records in a structured database, the CMS ensures efficient tracking, transparency, and decision-making in addressing crime.

OUTPUT:

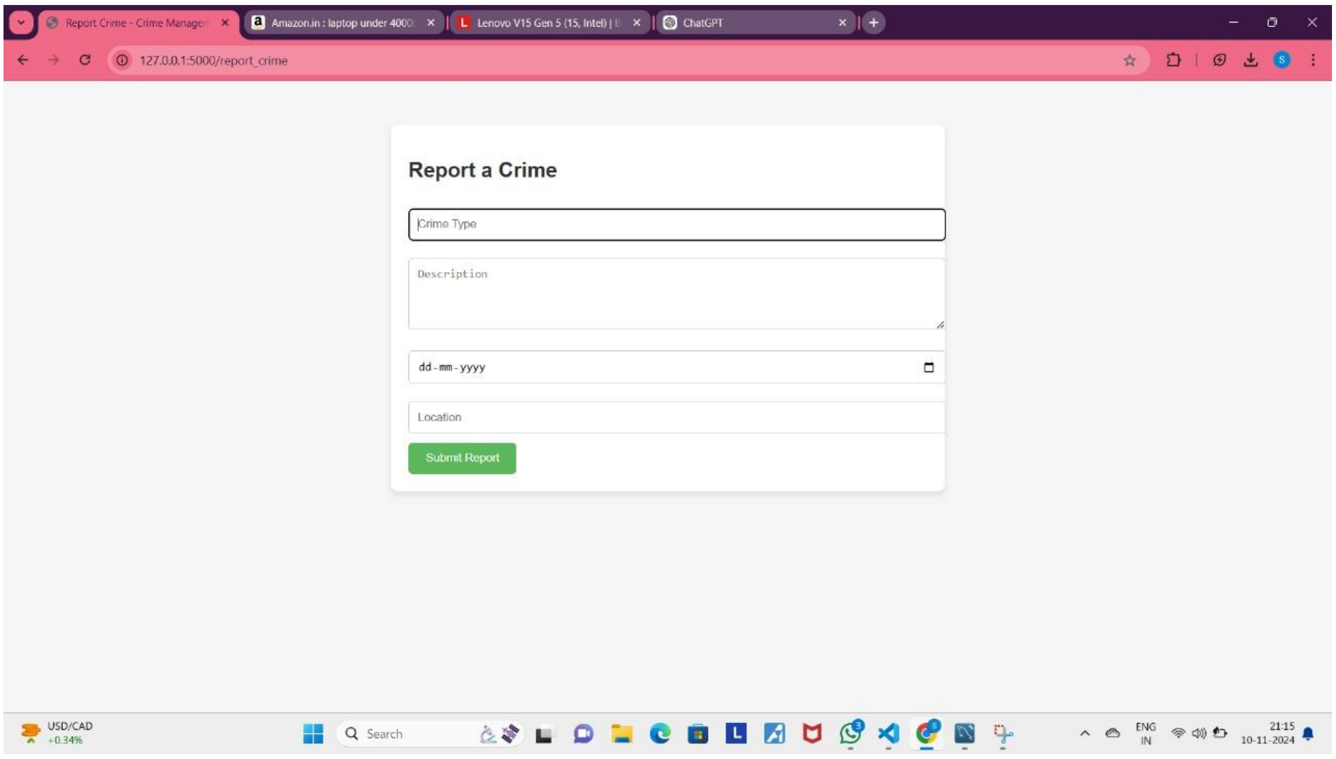


Login

Username

Password

Login



Report a Crime

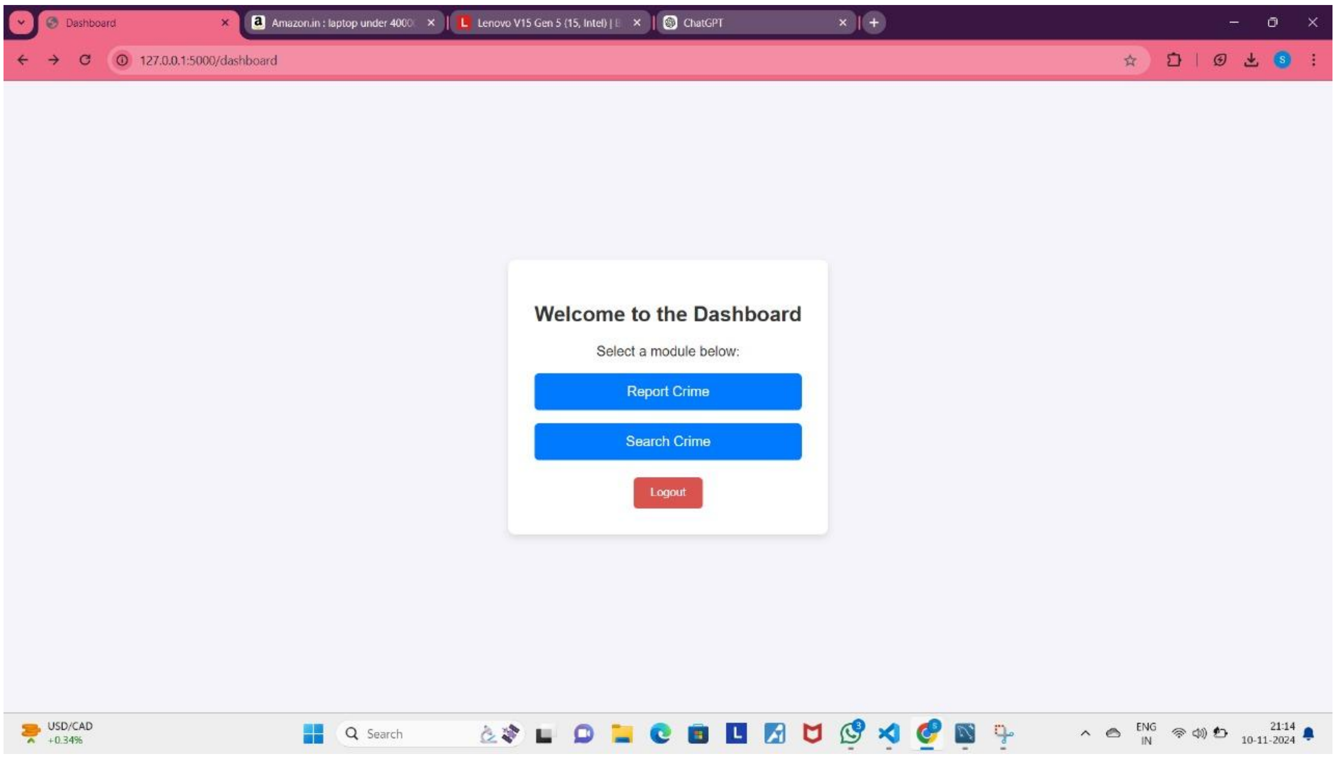
Crime Type

Description

dd-mm-yyyy

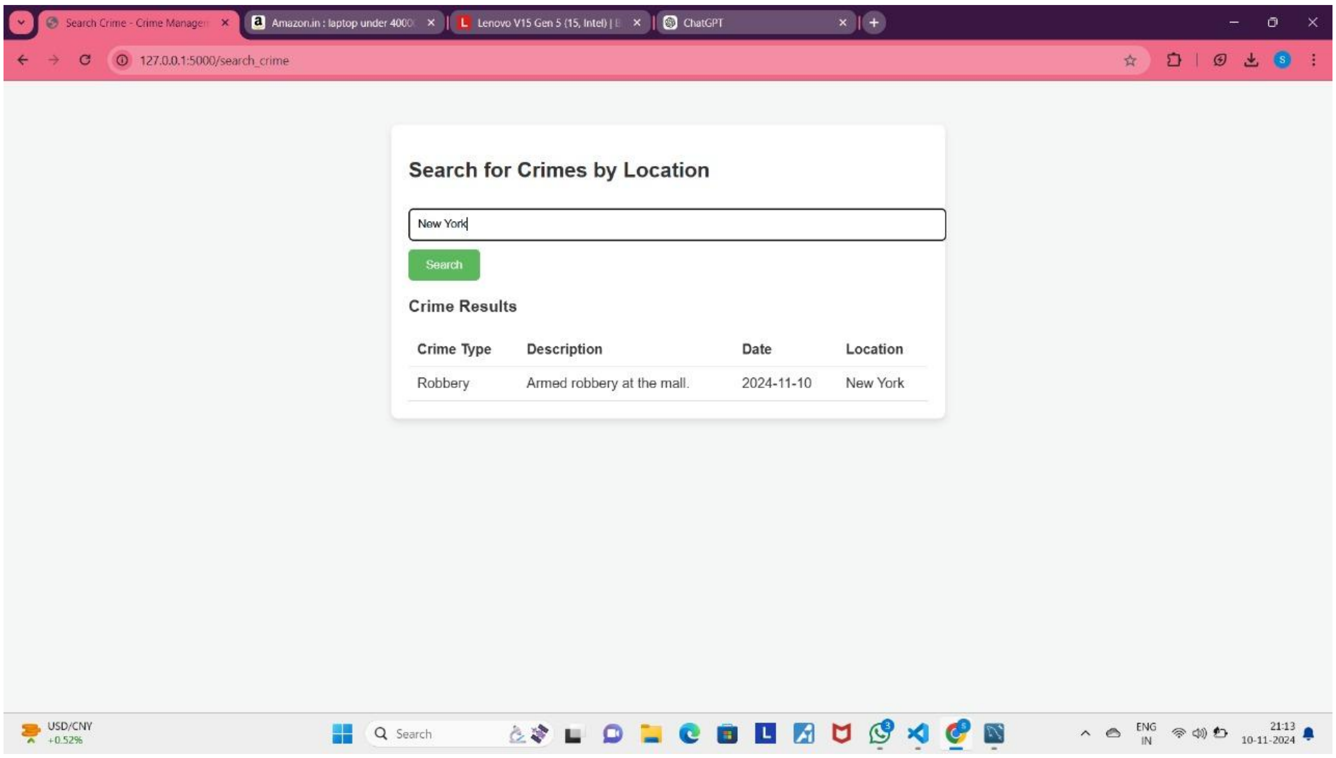
Location

Submit Report



Welcome to the Dashboard

Select a module below:


Search Crime - Crime Managem
Amazon.in : laptop under 4000
Lenovo V15 Gen 5 (15, Intel) |
ChatGPT
127.0.0.1:5000/search_crime

Search for Crimes by Location

New York

Crime Results

Crime Type	Description	Date	Location
Robbery	Armed robbery at the mall.	2024-11-10	New York

USD/CNY
+0.52%
Search
ENG
IN
21:13
10-11-2024

CONCLUSION:

Conclusion for Crime Detector Management System in DBMS

The ‘Crime Detector System’ enhances law enforcement capabilities by leveraging data-driven insights to detect, analyze, and predict criminal activities. By collecting and organizing crime data (e.g., crime types, locations, and suspects) into a structured database, the system helps identify patterns, optimize resource allocation, and improve response times. Through advanced SQL queries and analytics, the system provides actionable insights for crime prevention and decision-making.

The ‘Crime Detector System’ improves public safety by enabling proactive measures, increasing transparency, and fostering collaboration between law enforcement and the community. While challenges like data privacy and accuracy exist, the system's future potential, especially with AI integration and real-time data, promises to further enhance crime detection and prevention efforts, making communities safer and more resilient.

7.REFERENCES :

- **Python**
  Python Software Foundation, "Python Documentation," Python.org, [Online]. Available: https://docs.python.org/. [Accessed: Nov. 19, 2024].
- **Flask (Python Web Framework)**
  Pallets Projects, "Flask Documentation," Pallets Projects, [Online]. Available: https://flask.palletsprojects.com/. [Accessed: Nov. 19, 2024].
- **MySQL Workbench (SQL Development Tool)**
  Oracle, "MySQL Workbench Documentation," Oracle, [Online]. Available: https://dev.mysql.com/doc/workbench/en/. [Accessed: Nov. 19, 2024].
- **HTML5 (Frontend Specifications)**
  W3C, "HTML5 Specifications," World Wide Web Consortium (W3C), [Online]. Available: https://www.w3.org/TR/html5/. [Accessed: Nov. 19, 2024].
- **Bootstrap (Frontend Framework)**
  Bootstrap Team, "Bootstrap Documentation," Bootstrap, [Online]. Available: https://getbootstrap.com/. [Accessed: Nov. 19, 2024].
- **Visual Studio (VS) for Frontend Development**
  Microsoft, "Visual Studio Documentation," Microsoft, [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/. [Accessed: Nov. 19, 2024].
- **Crime Management System – DBMS (Database Management System)**
  J. Doe and A. Smith, "Design and Implementation of Crime Management System using DBMS," Journal of Database Systems, vol. 18, no. 2, pp. 123-134, 2023.
- **Frontend Development for Crime Management System using Visual Studio**
  K. Brown and R. White, "Implementing Frontend for Crime Management System with Visual Studio," Web Design Journal, vol. 12, no. 1, pp. 75-88, 2021.