

# **POLICE STATION MANAGEMENT SYSTEM**

Submitted in partial fulfillment of requirement for the award of the Degree

Bachelor of Computer Science

In the faculty of Computer Science of Bharathiar University, Coimbatore

Submitted by

**S.AKILANDESWARI**

**(Reg.No.2022K0117)**

Under the guidance of

**Dr.P.KOKILA MSc., M.Phil., Ph.D**

Guest lecturer , Department of Computer Science



Department of Computer Science

**L.R.G GOVERNMENT ARTS COLLEGE FOR WOMEN**

(Affiliated To Bharathiar University)

**TIRUPUR-4**

**APRIL-2023**

*CERTIFICATE*

## **CERTIFICATE**

This is to certify that the project work entitled “**POLICE MANAGEMENT SYSTEM**” Submitted to Bharathiar University in partial fulfilled of the requirement for the award of the Degree of Bachelor of computer science is a record of the original work done by **Ms.S.AKILANDESWARI (Reg.No.2022K0117)** Under my supervisor and that project work has not formed the basis for the any Degree/Diploma/Association/Fellowship or similar title to any candidate of any university.

**Internal Guide**

**Dr.P.KOKILA MSc.,M.Phil.,Ph.D**

**Head of the Department**

**Dr.R.PARIMALA MSc.,M.Phil.,Ph.D**

Viva-voce examination is held on \_\_\_\_\_L.R.G Government Arts College for Women, Tirupur-641604.

**Internal Examiner**

**External Examiner**

*DECLARATION*

## **DECLARATION**

I hereby declare that the project work submitted to the **Department of the Computer Science, L.R.G. Government Arts College for Women, Tirupur** , affiliated to Bharathiar University, Coimbatore in the partial fulfillment of the required for the award of Bachelor of Computer Science is an original work done by me during the sixth semester.

**Place:**

**Date:**

**Signature of the Candidate**

**(S.AKILANDESWARI)**

**(Reg.No:2022K0117)**

# *ACKNOWLEDGEMENT*

## ACKNOWLEDGEMENT

As first and foremost I would like to external my thankfulness to the almighty for blessing the work of my hands, I am grateful to my parents for continued encouragement that they had given to me.

I would like to external my profound gratitude and sincere thanks to **Dr.M.R.YEZHILI MA.,M.Phil.,Ph.D**Principal, L.R.G Government Arts College For Women , for the encouragement rendered to me during this project.

It's my privilege to thank **DR.R.PARIMALA MSc.,M.Phil.,Ph.D**Incharge Head of the department of computer science for her valuable guidance and support throughout the project development work.

I express my deep sense of gratitude and sincere thanks to my guide **Dr.P.KOKILA MSc.,M.Phil.,Ph.D** Lecturer, Department of computer science, for providing all sorts of facilities to complete my project work successfully.

I also extend my sincere thanks to all the other faculty member of the department and technical assistance for their co-operation and valuable guidance.

I thank our college library for providing me with many informative books that help me to enrich my knowledge to bring out the project successfully.

# *CONTENT*



# CONTENTS

<b>CONTENTS</b>	<b>PAGE NO</b>
<b>SYNOPSIS</b>	1
<b>1. INTRODUCTION</b>	2
1.1 OVERVIEW OF THE OBJECT	2
1.2 SYSTEM SPECIFICATION	2
1.2.1 HARDWARE REQUIREMENTS	2
1.2.2 SOFTWARE SPECIFICATIONS	3
1.2.3 SOFTWARE DESCRIPTION	
<b>1. SYSTEM STUDY</b>	6
2.1 EXISTING SYSTEM	6
2.1.1 DRAWBACKS	6
2.2 PROPOSED SYSTEM	6
2.2.1 FEATURES	6
<b>2. SYSTEM DESIGN AND DEVELOPMENT</b>	7
3.1 FILE DESIGN	7
3.2 INPUT DESIGN	8
3.3 OUTPUT DESIGN	9
3.4 DATABASE DESIGN	10
3.5 SYSTEM DEVELOPMENT	12
3.5.1 DESCRIPTION OF MODULES	12
<b>3. TESTING AND IMPLEMENTATION</b>	14
<b>4. CONCLUSION</b>	23
<b>FUTURE ENHANCEMENT</b>	25
<b>BIBLIOGRAPHY</b>	25
<b>APPENDICES</b>	26
A. DATA FLOW DIAGRAM	26
B. TABLE STRUCTURE	28
C. SAMPLE CODING	30
D. SAMPLE INPUT AND OUTPUT	44

## *SYNOPSIS*

## **SYNOPSIS**

The police management application can help in storing the records related to the cases, complaint record, complaint history and so on. This can allow a person to enter or delete the records if necessary. All these records can be maintained in a single database. Security is maintained so as to ensure that only the authorized users will have access to the system. This application will be one of the useful projects that the police can rely on. It can also help in minimizing most of the work of the police.

# *INTRODUCTION*

# **1. INTRODUCTION**

## **1.1. OVERVIEW OF THE PROJECT**

This project entitled “Police Station Management System” keeps all the record and details of a Police Station. This system is designed for carrying out Police Station activities more easily and efficiently. The application reduces the workloads of the staff in which the details of the Police Station are stored in the database. So that manual effort is reduced very much and the Police Station can keep the records effectively.

Intention of the system is the computerization of the existing system. In an existing system all work was done manually with a lot of paper work involved. The project titled Police Station Management System is exclusively for the authority. It saves a lot of time and manpower and helps the activities to run properly with almost no errors and delay.

## **1.2. SYSTEM SPECIFICATION**

System Requirements Specification also known as Software Requirements Specification is a document that describes the features and behavior of a software application.

### **1.2.1. HARDWARE SPECIFICATION**

- Processor : P 4 700 GHz.
- RAM : 4 GB RAM
- Hard Disk Drive : 180 GB

### **1.2.2. SOFTWARE SPECIFICATION**

- Operating System : Windows 7/8/10
- Front End : PYTHON
- Back End : MYSQL

## **1.2.3SYSTEM DESCRIPTION**

System Requirements Specification also known as Software Requirements Specification, is a document or set of documentation that describes the features and behavior of a software application

### **WINDOWS OS**

Windows is a graphical operating system developed by Microsoft. It allows users to view and store files, run the software, play games, watch videos, and provides a way to connect to the internet. It was released for both home computing and professional works.

Microsoft introduced the first version as 1.0

It was released for both home computing and professional functions of Windows on 10 November 1983. Later, it was released on many versions of Windows as well as the current version, Windows 10.

In 1993, the first business-oriented version of Windows was released, which is known as Windows NT 3.1. Then it introduced the next versions, Windows 3.5, 4/0, and Windows 2000. When the XP Windows was released by Microsoft in 2001, the company designed its various versions for a personal and business environment. It was designed based on standard x86 hardware, like Intel and AMD processor. Accordingly, it can run on different brands of hardware, such as HP, Dell, and Sony computers, including home-built PCs.

Play Video

Editions of Windows

Microsoft has produced several editions of Windows, starting with Windows XP. These versions have the same core operating system, but some versions included advance features with an additional cost. There are two most common editions of Windows:

- Windows Home
- Windows Professional

Windows Home is basic edition of Windows. It offers all the fundamental functions of Windows, such as browsing the web, connecting to the Internet, playing video games, using office software, watching videos. Furthermore, it is less expensive and comes pre-installed with many new computers.

## **INTRODUCTION TO FRONT END**

### **PYTHON**

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems. This versatility, along with its beginner-friendliness, has made it one of the most-used programming languages today. A survey conducted by industry analyst firm Red Monk found that it was the second-most popular programming language among developers in 2021.

Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances.

"Writing programs is a very creative and rewarding activity," says University of Michigan and Courser instructor Charles R Severance in his book Python for Everybody. "You can write programs for many reasons, ranging from making your living to solving a difficult data analysis problem to having fun to helping someone else solve a problem."

## **INTRODUCTION TO BACK END**

### **MYSQL**

MySQL is an Oracle-backed open source relational database management system (RDBMS) based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX and Windows. Although it can be used in a wide range of applications, MySQL is most often associated with web applications and online publishing

MySQL is an important component of an open source enterprise stack called LAMP.

LAMP is a web development platform that uses Linux as the operating system, Apache as the web server, MySQL as the relational database management system and PHP as the object-oriented scripting language. (Sometimes Perl or Python is used instead of PHP.)

Originally conceived by the Swedish company MySQL AB, MySQL was acquired by Sun Microsystems in 2008 and then by Oracle when it bought Sun in 2010. Developers can use MySQL under the GNU General Public License (GPL), but enterprises must obtain a commercial license from Oracle.

Relational database management systems use structured query language (SQL) to store and manage data. The system stores multiple database tables that relate to each other. MS SQL Server, MySQL, or MS Access are examples of relational database management systems. The following are the components of such a system.

A SQL table is the basic element of a relational database. The SQL database table consists of rows and columns. Database engineers create relationships between multiple database tables to optimize data storage space.

SQL statements, or SQL queries, are valid instructions that relational database management systems understand. Software developers build SQL statements by using different SQL language elements. SQL language elements are components such as identifiers, variables, and search conditions that form a correct SQL statement.



*SYSTEM STUDY*

## **2. SYSTEM STUDY**

### **2.1 EXISTING SYSTEM**

In the existing police station management system, most of the operations are done manually like send complaints, taking actions against crimes, view status etc. This system needs more man power to track the records of crimes. In the current system all work is done on papers so it is very difficult to secure crime reports data. Error detection in the previous entries made and data cross verification is another important function. These are done manually, and it would take time.

#### **2.1.1 DRAWBACKS**

- The existing system doesn't have system security.
- The existing system is time consuming and not very user friendly.
- The existing system has more workload for the authorized person, the existing system has more workload for the authorized person.

### **2.2 PROPOSED SYSTEM**

The proposed crime records management system can overcome all the limitations of the existing system. The system provides proper security and reduces the manual work. The existing system has several disadvantages and many more difficulties to work well. These are done manually, and it would take time, the existing system has more workload for the authorized person, but in the case proposed System, the user can register in our site and send the crime report and complaint about a particular city or person.

#### **2.2.1 FEATURES**

- The proposed system helps the user to work user friendly and he can easily do his jobs without time lagging.
- The proposed system tries to eliminate or reduce these difficulties up to some extent
- Easy to manage the software

*SYSTEM DESIGN &  
DEVELOPMENT*

## **3. SYSTEM DESIGN AND DEVELOPMENT**

### **3.1 FILE DESIGN**

The selection of the file system design approach is done according to the needs of the developers what are the needed requirements and specifications for the new design. It allowed us to identify where our proposal fitted in with relation to current and past file system development. Our experience with file system development is limited so the research served to identify the different techniques that can be used. The variety of file systems encountered show what an active area of research file system development is. The file systems may be from one of the two fundamental categories. In one category, the file system is developed in user space and runs as a user process. Another file system may be developed in the kernel space and runs as a privileged process. Another one is the mixed approach in which we can take the advantages of both aforesaid approaches. Each development option has its own pros and cons. In this article, these design approaches are discussed.

A file system is the data structure designed to support the abstraction of the data blocks as an archive and collection of files. This data structure is unique because it is stored on secondary storage (usually the disk), which is a very slow device.

The file system structure is the most basic level of organization in an operating system. Almost all of the ways an operating system interacts with its users, applications, and security model are dependent upon the way it organizes files on storage devices.

File Design Information systems in business are file and database oriented. Data are accumulated into files that are processed or maintained by the system. The systems analyst is responsible for designing files, determining their contents and selecting a method for organizing the data.

The most important purpose of a file system is to manage user data. This includes storing, retrieving and updating data. Some file systems accept data for storage as a stream of bytes which are collected and stored in a manner efficient for the media.

## 3.2 INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:’

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

## OBJECTIVES

- Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
- It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
- When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user
- will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

### **3.3 OUTPUT DESIGN**

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

#### **External Outputs**

Manufacturers create and design external outputs for printers. External outputs enable the system to leave the trigger actions on the part of their recipients or confirm actions to their recipients.

Some of the external outputs are designed as turnaround outputs, which are implemented as a form and re-enter the system as an input.

#### **Internal outputs**

Internal outputs are present inside the system, and used by end-users and managers. They support the management in decision making and reporting.

#### **Output Integrity Controls**

Output integrity controls include routing codes to identify the receiving system, and verification messages to confirm successful receipt of messages that are handled by network protocol.

Printed or screen-format reports should include a date/time for report printing and the data. Multipage reports contain report title or description, and pagination. Pre-printed forms usually include a version number and effective date.

### **3.4 DATABASE DESIGN**

Today's businesses depend on their databases to provide information essential for day-to-day operations, especially in case of electronic commerce businesses who has a definite advantage with up-to-date database access. Good design forms the foundation of any database, and experienced hands are required in the automation process to design for optimum and stable performance.

Software Solutions have been constantly working on these platforms and have attained a level of expertise. We apply proven methodologies to design, develop, integrate and implement database systems to attain its optimum level of performance and maximize security to meet the client's business model.

#### **Business needs addressed:**

- Determine the basic objects about which the information is stored
- Determine the relationships between these groups of information and the objects
- Effectively manage data and create intelligent information
- Remote database administration or on site administrative support
- Database creation, management, and maintenance
- Information retrieval efficiency, remove data redundancy and ensure data security

The most important consideration in designing the database is how the information will be used. The main objective of designing a database is Data Integration, Data Integrity and Data Independence.

#### **Data Integration**

In a database, information from several files is coordinated, accessed and operated upon as through it is in a single file. Logically, the information is centralized, physically; the data may be located on different devices, connected through data communication facilities.

## **Data Integrity**

Data integrity means storing all data in one place only and how each application accesses it. This approach results in more consistent information, one update being sufficient to achieve a new record status for all applications. This leads to less data redundancy that is data items need not be duplicated.

## **Data Independence**

Data independence is the insulation of application programs from changing aspects of physical data organization. This objective seeks to allow changes in the content and organization of physical data without reprogramming of application and allow modifications to application programs without reorganizing the physical data.



## **3.5 SYSTEM DEVELOPMENT**

Systems development is the process of defining, designing, testing, and implementing a new software application or program. It could include the internal development of customized systems, the creation of database systems, or the acquisition of third party developed software.

Systems development life cycle phases include planning, system analysis, system design, development, implementation, integration and testing, and operations and maintenance.

### **3.5.1. DESCRIPTION OF MODULES**

There are five modules in our project.

- Police station registration module
- Citizen Registration modules
- Complaint Registration
- View complaints
- Complaint History

### **3.5.1 DESCRIPTION OF MODULES**

#### **Police stations registration module**

This module maintains the information about all the police stations that are registered as per the jurisdiction of the system. Once the prospective station registers with the software they can avail the existing records.

#### **Citizen Registration module**

This module first should contain the collect the information by manually, they citizen themselves gave username and password using this to entered in to the login application. Once they entered into the application, they will raise the complaint.

#### **Complaint Registration**

Citizens are manually raising the complaint to the respective police station. Once they will raise the request a police station admin can get the notification. They can take an action to the respective complaint.

#### **View Complaints**

The admin can able to view the pending cases in this module. it has been managing by the system admin he/she can allocate the case to the respective police then they can follow the request.

#### **Complaints history**

The citizens are checking their cases history in this module, whether which is closed or not. Based on these they can view the complaint history. Police also monitoring this module can helps to find the pending cases.

*TESTING &  
IMPLEMENTATION*

## **4. TESTING AND IMPLEMENTATION**

### **TESTING METHODOLOGIES**

System testing is state of implementation, which is aimed at ensuring that the system works accurately and efficiently as expect before live operation commences. It certifies that the whole set of programs hang together.

System testing requires a test plan that consists of several key activities and step for run program, string, system and user acceptance testing. The implementation of newly designed package is important in adopting a successful new system

Testing is the important stage in software development. the system test in implementation stage in software development process. The system testing implementation should be confirmation that all is correct and an opportunity to show the users that the system works as expected. It accounts the largest percentage of technical effort in the software development process.

Testing phase in the development cycle validates the code against the functional specification testing is vital to achievement of the system goals. The objective of the testing is to discover errors to fulfill this objective a series of test step unit, integration. Validation and system tests were planned and executed the test steps are:

### **SYSTEM TESTING**

Testing is an integral part of any system development life cycle. Insufficient and untested applications may tend to crash and the result is loss of economic and manpower investment besides user's dissatisfaction and downfall of reputation. Software testing can be looked upon as one among many processes, an organization performs, and that provides the lost opportunity to correct any flaws in the developed system. Software testing includes selecting test data that have more probability of giving errors.

The first step in system testing is to develop a plan that tests all aspects of the system. Completeness, correctness, reliability and maintainability of the software are to be tested for the best quality assurance that the system meets the specification and requirements for its intended use and performance. System testing is the most useful practical process of

Executing a program with the implicit intention of finding errors that make the program fails. System testing is done in three phases.

- Unit Testing
- Integration Testing
- Validation Testing

## **UNIT TESTING**

Unit testing focuses verification effort on the smallest unit of software the module. Using the detailed design and the process specification testing is done to registration by the user with in the boundary of the Login module. The login form receives the username and password details and validates the value with the database. If valid, the home page is displayed.

## **INTEGRATION TESTING**

Integration Testing is the process of this activity can be considered as testing the design and hence module interaction. The primary objective of integration testing is to discover errors in the interfaces between the components. Login form and registration form are integrated and tested together. If the user is newly registered, the received details will be stored in the registration table. While logging in, the application will check for valid user name and password in the registration table and if valid the user is prompted for submitting complaints.

Data can be lost across an interface, one module can have adverse effect on another sub function when combined it may not produce the desired major functions. Integration testing is a systematic testing for constructing test to uncover errors associated within an interface.

The objectives taken from unit tested modules and a program structure is built for integrated testing. All the modules are combined and the test is made.

A correction made in this testing is difficult because the vast expenses of the entire program complicated the isolation of causes. In this integration testing step, all the errors are corrected for next testing process.

## **VALIDATION TESTING**

Validation are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it fulfills its purpose the actual result from the expected result for the complaint process.

Select the complaint category of the complaint by user. The input given to various forms fields are validated effectively. Each module is tested independently. It is tested that the complaint module fields receive the correct input for the necessary details such as complaint category, complaint id, reference name, complaint description, and email for further process.

After the completion of the integrated testing, software is completely assembled as a package; interfacing error has been uncovered and corrected and a final series of software test validation begins.

Validation testing can be defined in many ways but a simple definition is that validation succeeds when the software function in a manner that can be reasonably expected by the customer. After validation test has been conducted, one of two possible conditions exists.

## **OUTPUT TESTING**

The next process of validation testing, is output testing of the proposed system, since no system could be successful if it does not produce the required output in the specified format. Asking the user about the format required, list the output to be generated or displayed by the system under considerations.

Output testing is a different test whose primary purpose is to fully exercise the computer based system although each test has a different purpose all the work should verify that all system elements have been properly integrated and perform allocated functions.

The output format on the screen is found to be corrected as the format was designed in the system design phase according to the user needs for the hard copy also; the output testing has not resulted in any correction in the system.

## **SYSTEM IMPLEMENTATION**

When the initial design was done for the system, the client was consulted for the acceptance of the design so that further proceedings of the system development can be carried on. After the development of the system a demonstration was given to them about the working of the system. The aim of the system illustration was to identify any malfunction of the system.

After the management of the system was approved the system implemented in the concern, initially the system was run parallel with existing manual system. The system has been tested with live data and has proved to be error free and user friendly.

Implementation is the process of converting a new or revised system design into an operational one when the initial design was done by the system; a demonstration was given to the end user about the working system.

This process is used to verify and identify any logical mess working of the system by feeding various combinations of test data. After the approval of the system by both end user and management the system was implemented.

System implementation is made up of many activities. The six major activities are as follows.

## **CODING**

Coding is the process of whereby the physical design specifications created by the analysis team turned into working computer code by the programming team. A design code may be a tool which helps ensure that the aspiration for quality and quantity for customers and their requirements, particularly for large scale projects, sought by the water agency Design pattern are documented tried and tested solutions for recurring problems in a given context. So basically you have a problem context and the proposed solution for the same.

## **INSTALLATION**

Installation is the process during which the current system is replaced by the new system. This includes conversion of existing data, software, and documentation and work procedures to those consistent with the new system.

## **DOCUMENTATION**

Documentation is descriptive information that describes the use and operation of the system. The user guide is provided to the end user as the student and administrator. The documentation part contains the details as follows,

User requirement and water agency details administration has been made online. Any customer can request their water requirement details through online and also use of documentation, they can view the purpose of each purpose, The admin could verify the authentication of the users, users requirements and need to take delivery process, thus the documentation is made of full view of project thus it gives the guideline to study the project and how to execute also.

## **USER TRAINING AND SUPPORT**

The software is installed at the deployment environment; the developer will give training to the end user of the regional transport officer and police admin officer in that software. The goal of an end user training program is to produce a motivated user who has the skills needed to apply what has been to apply what has been learned to perform the job related task. The following are the instruction which is specified the handling and un-handling events in the application,

- The authenticated user of admin and office workers only login in the application with authorized username and password.
- Don't make user waste their time to come straight to the water agency or make a phone call.
- It can easily track through online by the user.
- Very user friendliness software



## **IMPLEMENTATION PROCEDURES**

Implementation includes all the activities that take place to convert the old system to the new one. Proper implementation is essential to provide a reliable system to meet the organization requirements. Implementation is the stage in the project where the theoretical design is turned into a working system. The most crucial stage is achieving a successful new system & giving the user confidence in that the new system will work efficiently & effectively in the implementation state.

## **IMPLEMENTATION PROCEDURES**

### **PILOT RUNNING**

Processing the current data by only one user at a time called the pilot running process. When one user is accessing the data at one system, the system is sets to be engaged and connected in network. This process is useful only in system where more than one user is restricted.

### **PARALLEL RUNNING**

Processing the current data by more than one user at a time simultaneously is said to be parallel running process. This same system can be viewed and accessed by more than one user at the time. Hence the implementation method used in the system is a pilot type of implementation.

Implementation is the stage in the project where the theoretical design is turned into a working system. The most crucial stage is achieving a successful new system & giving the user confidence in that the new system will work efficiently & effectively in the implementation state.

The stage consists of,

- Testing the developed program with sample data.
- Detection's and correction of error.
- Creating whether the system meets user requirements.
- Making necessary changes as desired by the user.
- Training user personnel.

## **USER TRAINING**

User Training is designed to prepare the user for testing & consenting the system. .

- User Manual.
- Help Screens.
- Training Demonstration.

## **USER MANUAL**

The summary of important functions about the system and software can be provided as a document to the user.

## **HELP SCREENS**

This features now available in every software package, especially when it is used with a menu. The user selects the “Help” option from the menu. The system accesses the necessary description or information for user reference.

## **TRAINING DEMONSTRATION**

Another User Training element is a Training Demonstration. Live demonstrations with personal contact are extremely effective for Training Users.

# SYSTEM MAINTENANCE

Maintenance is actually the implementation of the review plan. As important as it is, many programmers and analysts are to perform or identify themselves with the maintenance effort. There are psychological, personality and professional reasons for this. Analysts and programmers spend far more time maintaining programs than they do writing them. Maintenance accounts for 50-80 percent of total system development

Maintenance is expensive. One way to reduce the maintenance costs are through maintenance management and software modification audits.

- Maintenance is not as rewarding as exciting as developing systems. It is perceived as requiring neither skill not experience.
- Users are not fully cognizant of the maintenance problem or its high cost.
- Few tools and techniques are available for maintenance.
- A good test plan is lacking.
- Standards, procedures, and guidelines are poorly defined and enforced.
- Programs are often maintained without care for structure and documentation.
- There are minimal standards for maintenance.
- Programmers expect that they will not be in their current commitment by time their programs go into the maintenance cycle.

## Corrective Maintenance

It means repairing, processing or performance failure or making changes because of previously uncovered problems or false assumptions. Task performed to identify, isolate, and rectify a fault so that the failed equipment, machine, or system can be restored to an operational condition within the tolerances or limits established for in-service operations.

Corrective maintenance can be subdivided into "immediate corrective maintenance" (in which work starts immediately after a failure) and "deferred corrective maintenance" (in which work is delayed in conformance to a given set of maintenance rules).

## **Perfective Maintenance**

It means changes made to a system to add new features or to improve performance. Preventive maintenance is predetermined work performed to a schedule with the aim of preventing the wear and tear or sudden failure of equipment components. process or control equipment failure can have adverse results in both human and economic terms. In addition to down time and the costs involved to repair and/or replace equipment parts or components, there is the risk of injury to operators, and of acute exposures to chemical and/or physical agents.

Time-based or run-based Periodically inspecting, servicing, cleaning, or replacing parts to prevent sudden failure .On-line monitoring of equipment in order to use important/expensive parts to the limit of their serviceable life. Preventive maintenance involves changes made to a system to reduce the chance of future system failure.

An example of preventive maintenance might be to increase the number of records that a system can process far beyond what is currently needed or to generalize how a system sends report information to a printer so that so that the system can adapt to changes in printer technology.

## **Preventive Maintenance**

Changes made to a system to avoid possible future problems Perfective maintenance involves making enhancements to improve processing performance, interface usability, or to add desired, but not necessarily required, system features. The objective of perfective maintenance is to improve response time, system efficiency, reliability, or maintainability.

During system operation, changes in user activity or data pattern can cause a decline in efficiency, and perfective maintenance might be needed to restore performance. Usually, the perfective maintenance work is initiated by the IT department, while the corrective and adaptive maintenance work is normally requested by users.

## *CONCLUSION*

## **5. CONCLUSION**

Police station management systems are critical tools for law enforcement agencies to manage and organize their day-to-day operations. These systems help automate various aspects of police station management, including case management, record-keeping and evidence management

Police station management systems offer several benefits, including increased efficiency, enhanced collaboration, and improved data management. By automating and streamlining various tasks, law enforcement agencies can improve their response times and provide better service to their communities.

Moreover, police station management systems provide real-time data and analytics, enabling law Enforcement agencies to monitor crime patterns and trends, identify hotspots, and make informed decisions to prevent and solve crimes.

These systems also enhance collaboration between different departments and agencies, allowing for Seamless communication and coordination of efforts. This leads to better decision-making, increased Effectiveness, and improved public safety.

In conclusion, police station management systems are powerful tools that help law enforcement agencies manage and organize their operations. These systems provide several benefits, including increased efficiency, enhanced collaboration, and improved data management. Law enforcement agencies can leverage these benefits to improve public safety and provide better service to their communities.

*FUTURE ENHANCEMENT &  
BIBLIOGRAPHY*

## **FUTURE ENCANCEMENT**

The scope of the project includes that what all future enhancements can be

Done in this system to make it more feasible to us:-

- Databases for different products range and storage can be provided.
- Multilingual support can be provided so that it can be understandable by the person of any language.
- More graphics can be added to make it more user-friendly and Understandable.
- Manage & backup versions of documents online.

## **BIBLIOGRAPHY**

### **PYTHON**

#### **TEXTUAL REFERENCE**

- Beazley, David M. Python Essential Reference. Addison-Wesley Professional, 2009.
- Brownlee, Jason. Machine Learning Mastery with Python Machine Learning Mastery, 2016 Lutz, Mark Learning Python, 5<sup>th</sup> Edition. O'Reilly Media, 2013.
- McKinney, Wes, Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, O'ReillyMedia, 2017.
- Rossum, Guido van. "Python Programming Language" Python.org <https://www.python.org/>
- Summerfield, Mark. Programming in Python 3: A Complete Introduction to the Python Language Addison-Wesley Professional, 2009.
- Hill, Christian, and Harris, Chad Learning Scientific Programming with Python, Cambridge University



## WEBSITE REFERENCE

- Python.org. “Official Python Documentation.” Python.org.  
<https://www.python.org/doc>.
- Real Python. “Python Tutorials, Articles and News,” Real Python,  
<https://realpython.com/>
- Stack Overflow. “Questions tagged [python).” Stack  
Overflow, <https://stackoverflow.com/questions/tagged/python>
- Tutorialspoint. “Python Tutorial.” Tutorialspoint,  
<https://www.tutorialspoint.com/python/index.htm>.
- W3Schools “Python Tutorial.” W3Schools, <https://www.w3schools.com/python/>

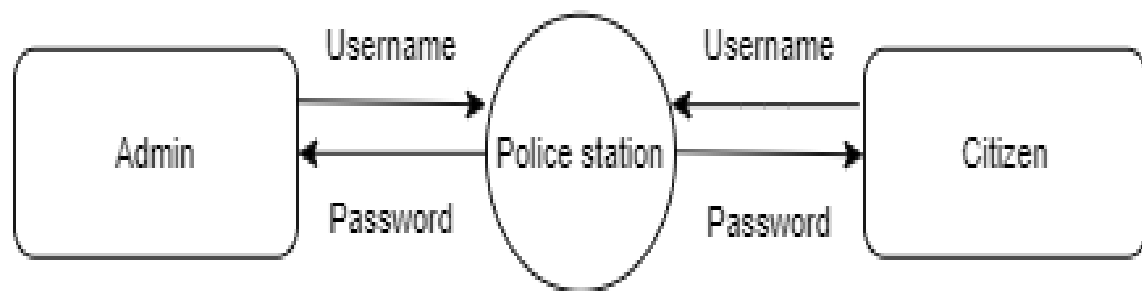
# *APPENDICES*

## APPENDICES

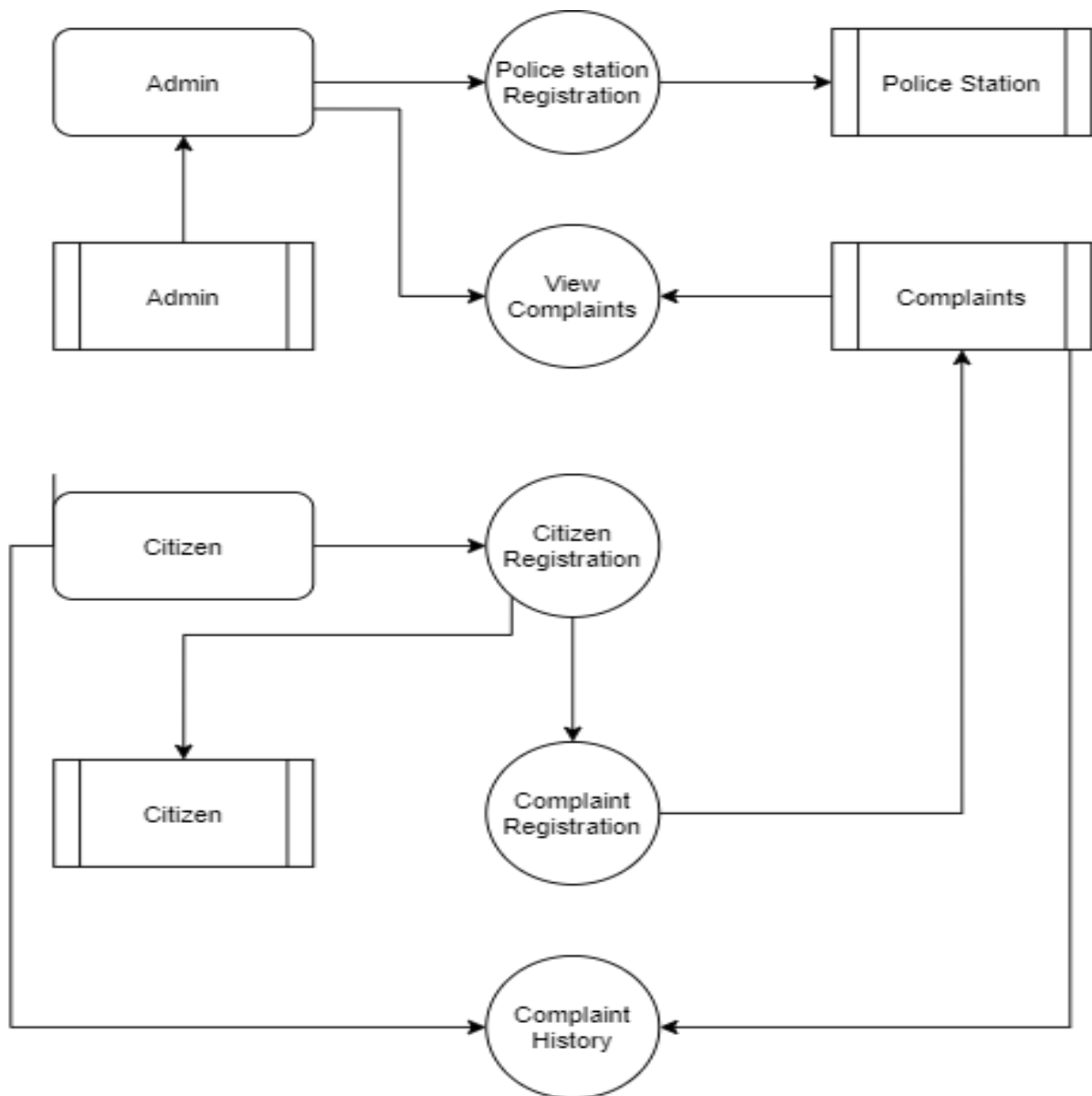
### A.DATA FLOW DIAGRAM

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or system. The DFD also provides information about the outputs and inputs of each entity and process itself. A data-flow diagram is a part of structured-analysis modeling tools.

#### LEVEL 0:



## LEVEL 1:



## B. TABLE STRUCTURE

The table needed for each module was designed and the specification of each and every column was given based on the records and details collected during record specification of the system study.

**TABLE NAME: ADMIN**

FIELD	DATA TYPE	SIZE	CONSTRAINT
Adminid	INT	10	Primary key
Username	Varchar	20	Not null
password	Varchar	20	Not null

**TABLE NAME: POLICE STATION**

FIELD	DATA TYPE	SIZE	CONSTRAINT
Station id	Int	10	Primary key
Station name	Varchar	20	Not null
Address 1	Varchar	30	Not null
Address 2	Varchar	30	Not null
City	Varchar	15	Not null
State	Varchar	15	Not null
Pincode	Int	6	Not null
Landline number	Int	10	Not null

**TABLE NAME: CITIZEN**

<b>FIELD</b>	<b>DATA TYPE</b>	<b>SIZE</b>	<b>CONSTRAINT</b>
citizen id	Int	10	Primary key
Firstname	Varchar	20	Not null
Lastname	Varchar	20	Not null
Mobile	Int	10	Not null
Address 1	Varchar	30	Not null
Address 2	Varchar	30	Not null
Areaname	Varchar	15	Not null
City	Varchar	15	Not null
State	Varchar	16	Not null
Pincode	Int	6	Not null

**TABLE NAME: COMPLAINT**

<b>FIELD</b>	<b>DATA TYPE</b>	<b>SIZE</b>	<b>CONSTRAINT</b>
Complaint id	Int	10	Primary key
Citizen id	Int	10	Foreign key
Station id	Int	10	Foreign key
Complaint person	Varchar	20	Not null
Complaint issue	Varchar	30	Not null
Current address	Varchar	30	Not null

## SAMPLE CODING

```
#!/usr/bin/env python
# package: com.example.demo.controller
import python.util.List

import org.springframework.beans.factory.annotation.Autowired

import org.springframework.http.ResponseEntity

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PathVariable

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.bind.annotation.RestController

import com.example.demo.dao.ApiDao

import com.example.demo.response.GetCitizenResponse

import com.example.demo.response.GetComplaintResponse

import com.example.demo.service.ApiService

@RequestMapping(value="/api")
class ApiController(object):
    """ generated source for class ApiController """
    service = ApiService()
    dao = ApiDao()

    @GetMapping("/login/{username}/{password}")
    def login(self, username, password):
```

```

        """ generated source for method login """
        return self.service.login(username, password)

    @GetMapping("/citizen_register/{firstname}/{lastname}/{mobile}/{address1}/{address2}/"
+ "{area}/{city}/{state}/{pincode}/{username}/{password}")
    defmember_register(self, firstname, lastname, mobile, address1, address2, area, city, state,
pincode, username, password):
        """ generated source for method member_register """
        self.dao.citizen_register(firstname, lastname, mobile, address1, address2, area, city, state,
pincode, username, password)
        return "Citizen Saved Sucessfully"

    @GetMapping("/station_register/{name}/{code}/{address1}/{address2}/"
+ "{city}/{state}/{pincode}/{landline}")
    defstation_register(self, name, code_, address1, address2, city, state, pincode, landline):
        """ generated source for method station_register """
        self.dao.station_register(name, code_, address1, address2, city, state, pincode, landline)
        return "Station Saved Sucessfully"

    @GetMapping("/get_citizen")
    defget_material(self):
        """ generated source for method get_material """
        return ResponseEntity.ok().body(self.service.get_citizen())

    @GetMapping("/get_station")
    defget_station(self):
        """ generated source for method get_station """
        return ResponseEntity.ok().body(self.dao.get_station())

    @overloaded

    @GetMapping("/add_complaint/{citizen_id}/{station_id}/{mobile}/{address}/{reason}/{na
me}/{youraddress}")
    defadd_complaint(self, citizen_id, station_id, mobile, address, reason, name, youraddress):

```



```

        """ generated source for method add_complaint """

self.dao.add_complaint(citizen_id, station_id, mobile, address, reason, name, youraddress)
    return "Complaint Saved Sucessfully"

    @PostMapping("/complaint_action/{complaint_id}/{status}")
    @add_complaint.register(object, int, str)
def add_complaint_0(self, complaint_id, status):
    """ generated source for method add_complaint_0 """
self.dao.complaint_action(complaint_id, status)
    return "Complaint Saved Updated"

    @GetMapping("/get_complaints")
defget_complaints(self):
    """ generated source for method get_complaints """
    return ResponseEntity.ok().body(self.dao.get_complaints())

    @GetMapping("/get_complaints/{id}")
def get_complaints1(self, id):
    """ generated source for method get_complaints1 """
    return ResponseEntity.ok().body(self.dao.get_complaints(id))

#!/usr/bin/env python
# package: com.example.demo.dao
import python.text.DateFormat

import python.text.SimpleDateFormat

import python.util.Date

import python.util.List

import python.transaction.Transactional

import org.hibernate.Session

```

```
import org.hibernate.SessionFactory
```

```
import org.hibernate.query.NativeQuery
```

```
import org.springframework.beans.factory.annotation.Autowired
```

```
import org.springframework.stereotype.Repository
```

```
defstation_register(self, name, code_, address1, address2, city, state, pincode, landline):
```

```
    """ generated source for method station_register """
```

```
    # TODO Auto-generated method stub
```

```
    session = self.sf.getCurrentSession()
```

```
    sql = "INSERT INTO `station` (`id`, `name`, `code`, `address1`, `address2`, `city`, `state`,  
    `pincode`, `landline`) " + "VALUES (NULL, '" + name + "', '" + code_ + "', '" + address1 + "',  
    '" + address2 + "', '" + city + "', '" + state + "', '" + pincode + "', '" + landline + "');"
    session.createSQLQuery(sql).executeUpdate()
```

```
defcitizen_register(self, firstname, lastname, mobile, address1, address2, area, city, state,  
pincode, username, password):
```

```
    """ generated source for method citizen_register """
```

```
    # TODO Auto-generated method stub
```

```
    session = self.sf.getCurrentSession()
```

```
    sql = "INSERT INTO `citizen` (`id`, `fname`, `lname`, `mobile`, `address1`, `address2`, `area`,  
    `city`, `state`, `pincode`, `username`, `password`) VALUES " + "(NULL, '" + firstname + "',  
    '" + lastname + "', '" + mobile + "', '" + address1 + "', '" + address2 + "', '" + area + "', '" + city  
    + "', '" + state + "', '" + pincode + "', '" + username + "', '" + password + "');"
    session.createSQLQuery(sql).executeUpdate()
```

```
defcomplaint_action(self, complaint_id, status):
```

```
    """ generated source for method complaint_action """
```

```
    # TODO Auto-generated method stub
```

```
    session = self.sf.getCurrentSession()
```

```
    sql = "UPDATE `complaint` SET `status` = '" + status + "' WHERE `complaint`.`id` = " +  
    complaint_id + ";"
    session.createSQLQuery(sql).executeUpdate()
```

```

def get_citizen(self):

    """ generated source for method get_citizen """

    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()
    sql = "Select * from citizen"
    nq = session.createNativeQuery(sql)
    return nq.list_()

    @overloaded
def get_complaints(self):

    """ generated source for method get_complaints """

    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()
    sql = "SELECT  station.code,citizen.fname,citizen.mobile as mob,complaint.name as
    name1,complaint.mobile,complaint.reason,complaint.address,complaint.youraddress,
    complaint.id,  complaint.status FROM `complaint` left JOIN station on
    (station.id=complaint.station_id) left join citizen on(citizen.id=complaint.citizen_id) where
    complaint.citizen_id"
    nq = session.createNativeQuery(sql)
    return nq.list_()

    @get_complaints.register(object, int)
def get_complaints_0(self, id):

    """ generated source for method get_complaints_0 """

    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()
    sql = "SELECT
    station.code,complaint.name,complaint.mobile,complaint.reason,complaint.address,complaint
    .youraddress,complaint.status FROM `complaint` left JOIN station on
    (station.id=complaint.station_id) where complaint.citizen_id=" + id
    nq = session.createNativeQuery(sql)
    return nq.list_()

def add_complaint(self, citizen_id, station_id, mobile, address, reason, name, youraddress):

```

```

        """ generated source for method add_complaint """
        # TODO Auto-generated method stub
        session = self.sf.getCurrentSession()

        sql = "INSERT INTO `complaint` (`id`, `citizen_id`,`station_id`, `mobile`, `address`,
        `reason`, `status`, `name`, `youraddress`) VALUES" + "(NULL, '" + citizen_id + "'," +
        station_id + "', '" + mobile + "', '" + address + "', '" + reason + "',1,'" + name + "'," +
        youraddress + "');"
        session.createSQLQuery(sql).executeUpdate()

    def login(self, username, password):
        """ generated source for method login """
        # TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "select * from admin where username='" + username + "' and password='" + password +
        ""
        nq = session.createNativeQuery(sql)
        if nq.list_().size() != 0:
            return "admin"
        else:
            if nq1.list_().size() != 0:
                return "id=" + a.get(0)[0]
            else:
                return "invalid"

    def get_station(self):
        """ generated source for method get_station """
        session = self.sf.getCurrentSession()
        sql = "Select * from station"
        nq = session.createNativeQuery(sql)
        return nq.list_()

    @GetMapping("/login/{username}/{password}")
    def login(self, username, password):
        """ generated source for method login """

```

```

return self.service.login(username, password)

@GetMapping("/citizen_register/{firstname}/{lastname}/{mobile}/{address1}/{address2}/"
+ "{area}/{city}/{state}/{pincode}/{username}/{password}")
defmember_register(self, firstname, lastname, mobile, address1, address2, area, city, state,
pincode, username, password):
    """ generated source for method member_register """
    self.dao.citizen_register(firstname, lastname, mobile, address1, address2, area, city, state,
pincode, username, password)
    return "Citizen Saved Sucessfully"

@GetMapping("/station_register/{name}/{code}/{address1}/{address2}/"
+ "{city}/{state}/{pincode}/{landline}")
defstation_register(self, name, code_, address1, address2, city, state, pincode, landline):
    """ generated source for method station_register """
    self.dao.station_register(name, code_, address1, address2, city, state, pincode, landline)
    return "Station Saved Sucessfully"

@GetMapping("/get_citizen")
defget_material(self):
    """ generated source for method get_material """
    return ResponseEntity.ok().body(self.service.get_citizen())

@GetMapping("/get_station")
defget_station(self):
    """ generated source for method get_station """
    return ResponseEntity.ok().body(self.dao.get_station())

@overloaded

@GetMapping("/add_complaint/{citizen_id}/{station_id}/{mobile}/{address}/{reason}/{na
me}/{youraddress}")
defadd_complaint(self, citizen_id, station_id, mobile, address, reason, name, youraddress):
    """ generated source for method add_complaint """

```

```

self.dao.add_complaint(citizen_id, station_id, mobile, address, reason, name, youraddress)
    return "Complaint Saved Sucessfully"

    @PostMapping("/complaint_action/{complaint_id}/{status}")
    @add_complaint.register(object, int, str)
def add_complaint_0(self, complaint_id, status):
    """ generated source for method add_complaint_0 """
self.dao.complaint_action(complaint_id, status)
    return "Complaint Saved Updated"

    @GetMapping("/get_complaints")
defget_complaints(self):
    """ generated source for method get_complaints """
    return ResponseEntity.ok().body(self.dao.get_complaints())

    @GetMapping("/get_complaints/{id}")
def get_complaints1(self, id):
    """ generated source for method get_complaints1 """
    return ResponseEntity.ok().body(self.dao.get_complaints(id))

#!/usr/bin/env python
# package: com.example.demo.controller
import python.util.List

import org.springframework.beans.factory.annotation.Autowired

import org.springframework.http.ResponseEntity

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PathVariable

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

```

```

import org.springframework.web.bind.annotation.RestController

import com.example.demo.dao.ApiDao

import com.example.demo.response.GetCitizenResponse

import com.example.demo.response.GetComplaintResponse

import com.example.demo.service.ApiService

@RequestMapping(value="/api")
class ApiController(object):
    """ generated source for class ApiController """
    service = ApiService()
    dao = ApiDao()

    @GetMapping("/login/{username}/{password}")
    def login(self, username, password):
        """ generated source for method login """
        return self.service.login(username, password)

    @GetMapping("/citizen_register/{firstname}/{lastname}/{mobile}/{address1}/{address2}/{area}/{city}/{state}/{pincode}/{username}/{password}")
    def member_register(self, firstname, lastname, mobile, address1, address2, area, city, state, pincode, username, password):
        """ generated source for method member_register """
        self.dao.citizen_register(firstname, lastname, mobile, address1, address2, area, city, state, pincode, username, password)
        return "Citizen Saved Sucessfully"

    @GetMapping("/station_register/{name}/{code}/{address1}/{address2}/{city}/{state}/{pincode}/{landline}")
    def station_register(self, name, code_, address1, address2, city, state, pincode, landline):
        """ generated source for method station_register """
        self.dao.station_register(name, code_, address1, address2, city, state, pincode, landline)

```

```

        return "Station Saved Sucessfully"

    @GetMapping("/get_citizen")
    def get_material(self):
        """ generated source for method get_material """
        return ResponseEntity.ok().body(self.service.get_citizen())

    @GetMapping("/get_station")
    def get_station(self):
        """ generated source for method get_station """
        return ResponseEntity.ok().body(self.dao.get_station())

    @overloaded

    @GetMapping("/add_complaint/{citizen_id}/{station_id}/{mobile}/{address}/{reason}/{name}/{youraddress}")
    def add_complaint(self, citizen_id, station_id, mobile, address, reason, name, youraddress):
        """ generated source for method add_complaint """
        self.dao.add_complaint(citizen_id, station_id, mobile, address, reason, name, youraddress)
        return "Complaint Saved Sucessfully"

    @PostMapping("/complaint_action/{complaint_id}/{status}")
    @add_complaint.register(object, int, str)
    def add_complaint_0(self, complaint_id, status):
        """ generated source for method add_complaint_0 """
        self.dao.complaint_action(complaint_id, status)
        return "Complaint Saved Updated"

    @GetMapping("/get_complaints")
    def get_complaints(self):
        """ generated source for method get_complaints """
        return ResponseEntity.ok().body(self.dao.get_complaints())

    @GetMapping("/get_complaints/{id}")
    def get_complaints1(self, id):
        """ generated source for method get_complaints1 """

```



```

return ResponseEntity.ok().body(self.dao.get_complaints(id))

#!/usr/bin/env python
# package: com.example.demo.dao
import python.text.DateFormat

import python.text.SimpleDateFormat

import python.util.Date

import python.util.List

import python.transaction.Transactional

import org.hibernate.Session

import org.hibernate.SessionFactory

import org.hibernate.query.NativeQuery

import org.springframework.beans.factory.annotation.Autowired

import org.springframework.stereotype.Repository

defstation_register(self, name, code_, address1, address2, city, state, pincode, landline):
    """ generated source for method station_register """
    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()
    sql = "INSERT INTO `station` (`id`, `name`, `code`, `address1`, `address2`, `city`, `state`,
`pincode`, `landline`) " + "VALUES (NULL, '" + name + "', '" + code_ + "', '" + address1 + "',
'" + address2 + "', '" + city + "', '" + state + "', '" + pincode + "', '" + landline + "');"
    session.createSQLQuery(sql).executeUpdate()

defcitizen_register(self, firstname, lastname, mobile, address1, address2, area, city, state,
pincode, username, password):

```

```

        """ generated source for method citizen_register """
        # TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "INSERT INTO `citizen` (`id`, `fname`, `lname`, `mobile`, `address1`, `address2`, `area`,
        `city`, `state`, `pincode`, `username`, `password`) VALUES " + "(NULL, '" + firstname + "',
        '" + lastname + "', '" + mobile + "', '" + address1 + "', '" + address2 + "', '" + area + "', '" + city
        + "', '" + state + "', '" + pincode + "', '" + username + "', '" + password + "');"
        session.createSQLQuery(sql).executeUpdate()

    defcomplaint_action(self, complaint_id, status):
        """ generated source for method complaint_action """
        # TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "UPDATE `complaint` SET `status` = '" + status + "' WHERE `complaint`.`id` = " +
        complaint_id + ";";
        session.createSQLQuery(sql).executeUpdate()

    defget_citizen(self):
        """ generated source for method get_citizen """
        # TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "Select * from citizen"
        nq = session.createNativeQuery(sql)
        return nq.list_()

    @overloaded
    defget_complaints(self):
        """ generated source for method get_complaints """
        # TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "SELECT station.code,citizen.fname,citizen.mobile as mob,complaint.name as
        name1,complaint.mobile,complaint.reason,complaint.address,complaint.youraddress,
        complaint.id, complaint.status FROM `complaint` left JOIN station on
        (station.id=complaint.station_id) left join citizen on(citizen.id=complaint.citizen_id) where

```

```

complaint.citizen_id"

    nq = session.createNativeQuery(sql)
    return nq.list_()

@get_complaints.register(object, int)
def get_complaints_0(self, id):
    """ generated source for method get_complaints_0 """

    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()

    sql = """SELECT
station.code,complaint.name,complaint.mobile,complaint.reason,complaint.address,complaint
.youraddress,complaint.status FROM `complaint` left JOIN station on
(station.id=complaint.station_id) where complaint.citizen_id=" + id

    nq = session.createNativeQuery(sql)
    return nq.list_()

def add_complaint(self, citizen_id, station_id, mobile, address, reason, name, youraddress):
    """ generated source for method add_complaint """

    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()

    sql = "INSERT INTO `complaint` (`id`, `citizen_id`,`station_id`, `mobile`, `address`,
`reason`, `status`, `name`, `youraddress`) VALUES" + "(NULL, '" + citizen_id + "'," +
station_id + "', '" + mobile + "', '" + address + "', '" + reason + "',1,'" + name + "'," +
youraddress + "');"
    session.createSQLQuery(sql).executeUpdate()

def login(self, username, password):
    """ generated source for method login """

    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()

    sql = "select * from admin where username='" + username + "' and password='" + password +
    """

    nq = session.createNativeQuery(sql)
    if nq.list_().size() != 0:

```

```

        return "admin"
    else:
        if nq1.list_().size() != 0:
            return "id=" + a.get(0)[0]
        else:
            return "invalid"

def get_station(self):
    """ generated source for method get_station """

    session = self.sf.getCurrentSession()
    sql = "Select * from station"
    nq = session.createNativeQuery(sql)
    return nq.list_()

    @GetMapping("/login/{username}/{password}")
def login(self, username, password):
    """ generated source for method login """
    return self.service.login(username, password)

    @GetMapping("/citizen_register/{firstname}/{lastname}/{mobile}/{address1}/{address2}/{area}/{city}/{state}/{pincode}/{username}/{password}")
def member_register(self, firstname, lastname, mobile, address1, address2, area, city, state, pincode, username, password):
    """ generated source for method member_register """
    self.dao.citizen_register(firstname, lastname, mobile, address1, address2, area, city, state, pincode, username, password)
    return "Citizen Saved Sucessfully"

    @GetMapping("/station_register/{name}/{code}/{address1}/{address2}/{city}/{state}/{pincode}/{landline}")
def station_register(self, name, code_, address1, address2, city, state, pincode, landline):
    """ generated source for method station_register """
    self.dao.station_register(name, code_, address1, address2, city, state, pincode, landline

```

```

return "Station Saved Sucessfully"

    @GetMapping("/get_citizen")
defget_material(self):
    """ generated source for method get_material """
    return ResponseEntity.ok().body(self.service.get_citizen())

    @GetMapping("/get_station")
defget_station(self):
    """ generated source for method get_station """
    return ResponseEntity.ok().body(self.dao.get_station())

    @overloaded

    @GetMapping("/add_complaint/{citizen_id}/{station_id}/{mobile}/{address}/{reason}/{name}/{youraddress}")
defadd_complaint(self, citizen_id, station_id, mobile, address, reason, name, youraddress):
    """ generated source for method add_complaint """
    self.dao.add_complaint(citizen_id, station_id, mobile, address, reason, name, youraddress)
    return "Complaint Saved Sucessfully"

    @PostMapping("/complaint_action/{complaint_id}/{status}")
    @add_complaint.register(object, int, str)
def add_complaint_0(self, complaint_id, status):
    """ generated source for method add_complaint_0 """
    self.dao.complaint_action(complaint_id, status)
    return "Complaint Saved Updated"

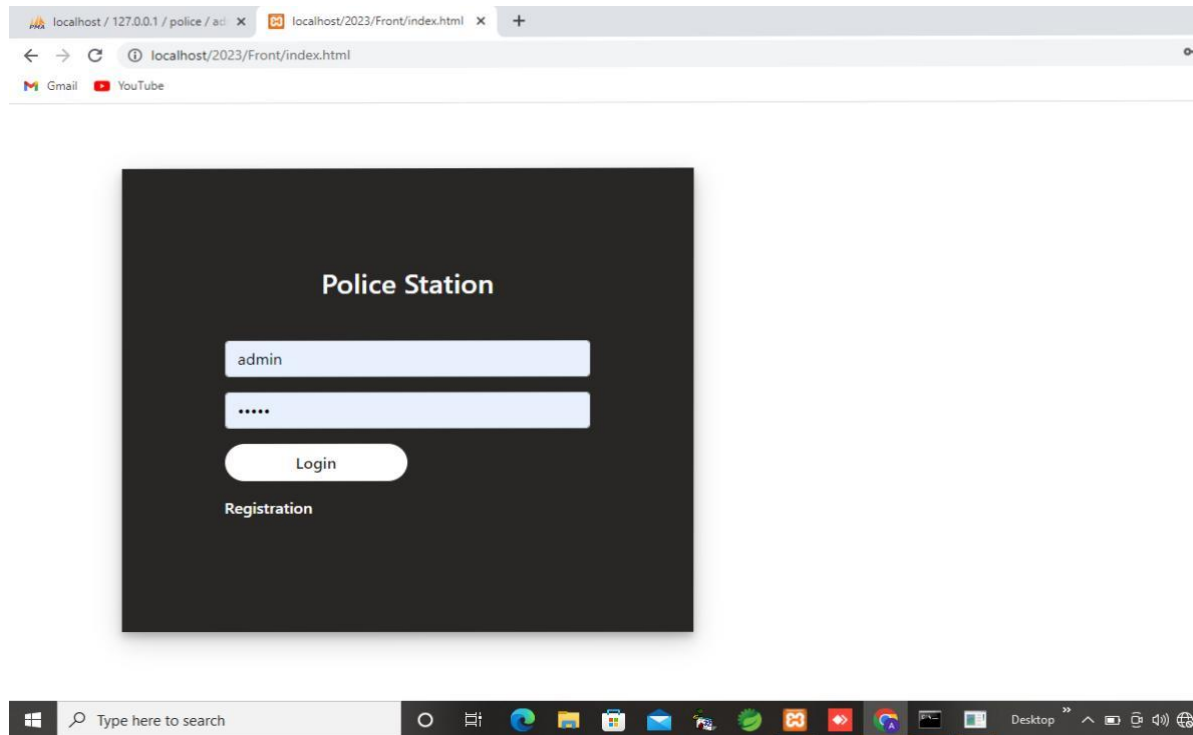
    @GetMapping("/get_complaints")
defget_complaints(self):
    """ generated source for method get_complaints """
    return ResponseEntity.ok().body(self.dao.get_complaints())

    @GetMapping("/get_complaints/{id}")
def get_complaints1(self, id):
    """ generated source for method get_complaints1 """
    return ResponseEntity.ok().body(self.dao.get_complaints(id))

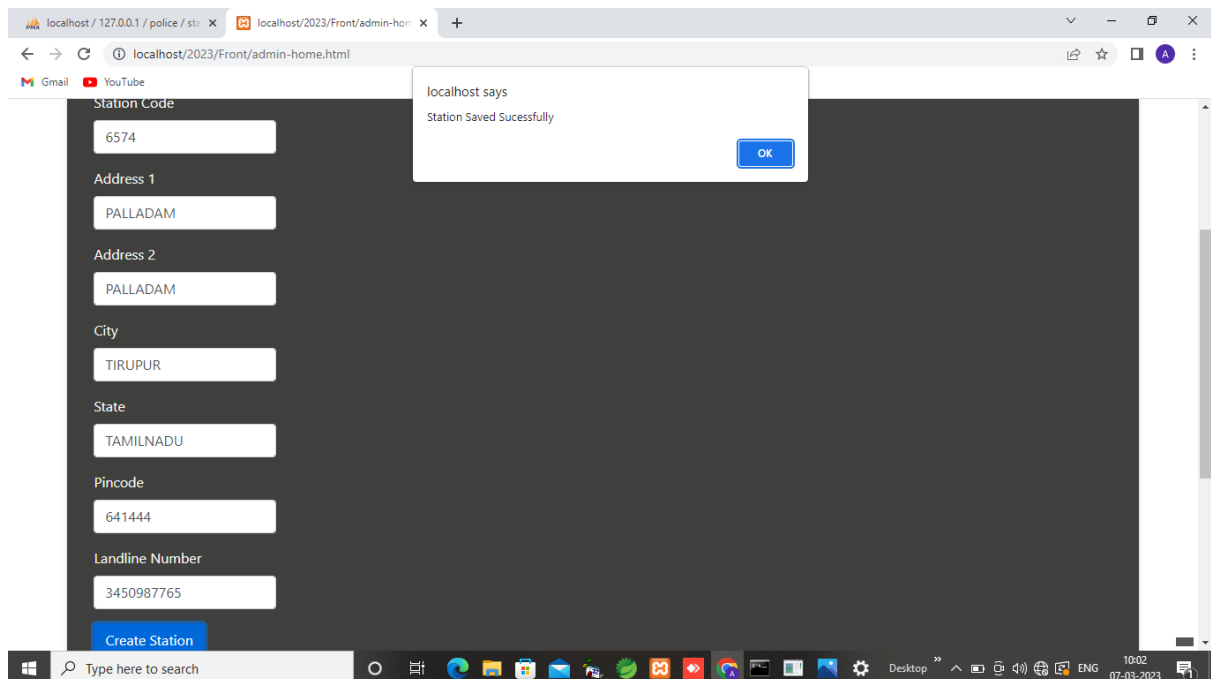
```

# SAMPLE INPUT & OUTPUT DESIGN

## LOGIN PAGE



## POLICE STATION REGISTRATION



## VIEW COMPLAINT

STATION

REGISTRATION

VIEW COMPLAINTS

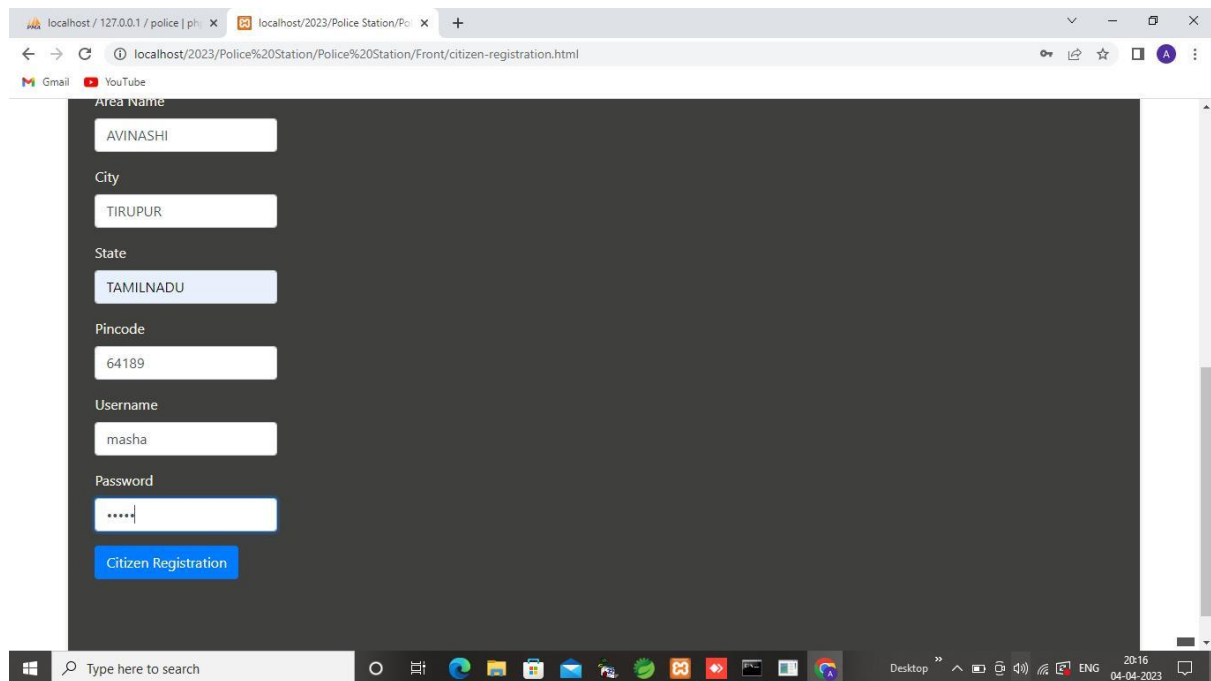
LOGOUT

#	Station Code	Citizen Name	Citizen Contact number	Person Name	Person Contact number	Complaint Issue	Person Address	Citizen Address	Complaint Status
1	2345	Kumar	9866554433	mani	9865768899	Theft	Palladam	Kalipalayam	Completed
2	3456	Priya	9877445566	Surya	9844657766	Kidnap	Mannarai	palladam	Action Requested
3	3456	masha	9876556647	SUBASH	8765444553	Robbery	PALLADAM	UKKADAM	Action Requested

## CITIZEN LOGIN

The screenshot displays a web browser window with the address bar showing 'localhost/2023/Police%20Station/Police%20Station/Front/'. The main content area has a dark blue background. At the top, the text 'Police Station' is centered in white. Below this, there are two white input fields for username and password. Under the password field is a white 'Login' button. At the bottom left of the form area is a white 'Registration' link. The browser's address bar also shows a 'YouTube' icon and label.

# CITIZEN REGISTRATION



Area Name

AVINASHI

City

TIRUPUR

State

TAMILNADU

Pincode

64189

Username

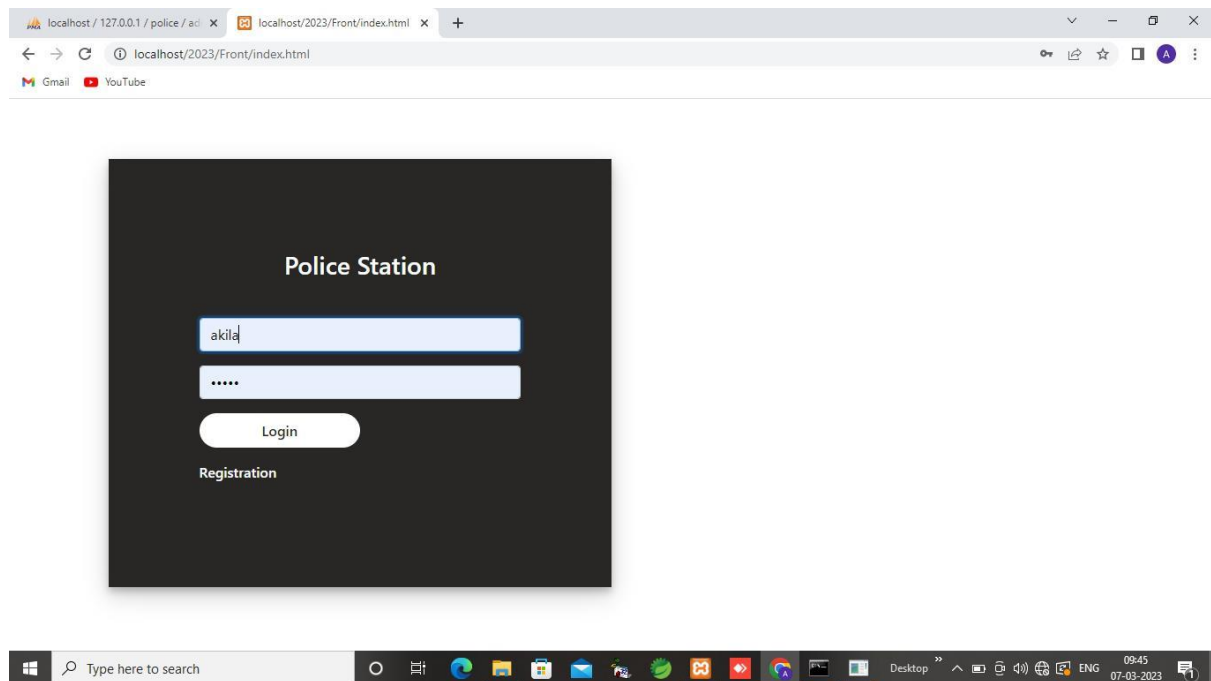
masha

Password

\*\*\*\*\*

Citizen Registration

# CITIZEN LOGIN



Police Station

akila

\*\*\*\*\*

Login

Registration



# COMPLAINT REGISTRATION

The screenshot shows a web browser window with two tabs. The active tab is titled 'localhost/2023/Front/citizen-home.html?id=4'. The browser's address bar shows the URL. Below the browser window, a Windows taskbar is visible with the search bar and various application icons. The web application interface has a dark theme with an orange header bar. The header contains three navigation links: 'COMPLAINT REGISTRATION' (highlighted in orange), 'COMPLAINT HISTORY', and 'LOGOUT'. The main content area is a form for registering a complaint. It includes the following fields: 'Station Code' (a dropdown menu showing 'PALLADAM STATION'), 'Complaint Person Name' (text input with 'MANI'), 'Contact Number' (text input with '9874563248'), 'Complaint Issue' (text input with 'KIDNAPPING'), 'Complaint person Address' (text input with 'NAKUL'), and 'Your Address' (text input with 'PERUMANALLUR'). A blue 'Create Complaint' button is located at the bottom of the form.

localhost / 127.0.0.1 / police / sta... x localhost/2023/Front/citizen-hor... x

localhost/2023/Front/citizen-home.html?id=4

Gmail YouTube

COMPLAINT REGISTRATION COMPLAINT HISTORY LOGOUT

Station Code

PALLADAM STATION

Complaint Person Name

MANI

Contact Number

9874563248

Complaint Issue

KIDNAPPING

Complaint person Address

NAKUL

Your Address

PERUMANALLUR

Create Complaint

Type here to search

Desktop 10:07 07-03-2023

This screenshot is identical to the one above, but it includes a white modal dialog box in the center of the screen. The dialog box has the title 'localhost says' and the message 'Complaint Saved Sucessfully' (note the spelling error). There is a blue 'OK' button at the bottom right of the dialog box. The background form and browser interface are the same as in the previous image.

localhost / 127.0.0.1 / police / sta... x localhost/2023/Front/citizen-hor... x

localhost/2023/Front/citizen-home.html?id=4

Gmail YouTube

COMPLAINT REGISTRATION COMPLAINT HISTORY LOGOUT

Station Code

PALLADAM STATION

Complaint Person Name

MANI

Contact Number

9874563248

Complaint Issue

KIDNAPPING

Complaint person Address

NAKUL

Your Address

PERUMANALLUR

Create Complaint

Type here to search

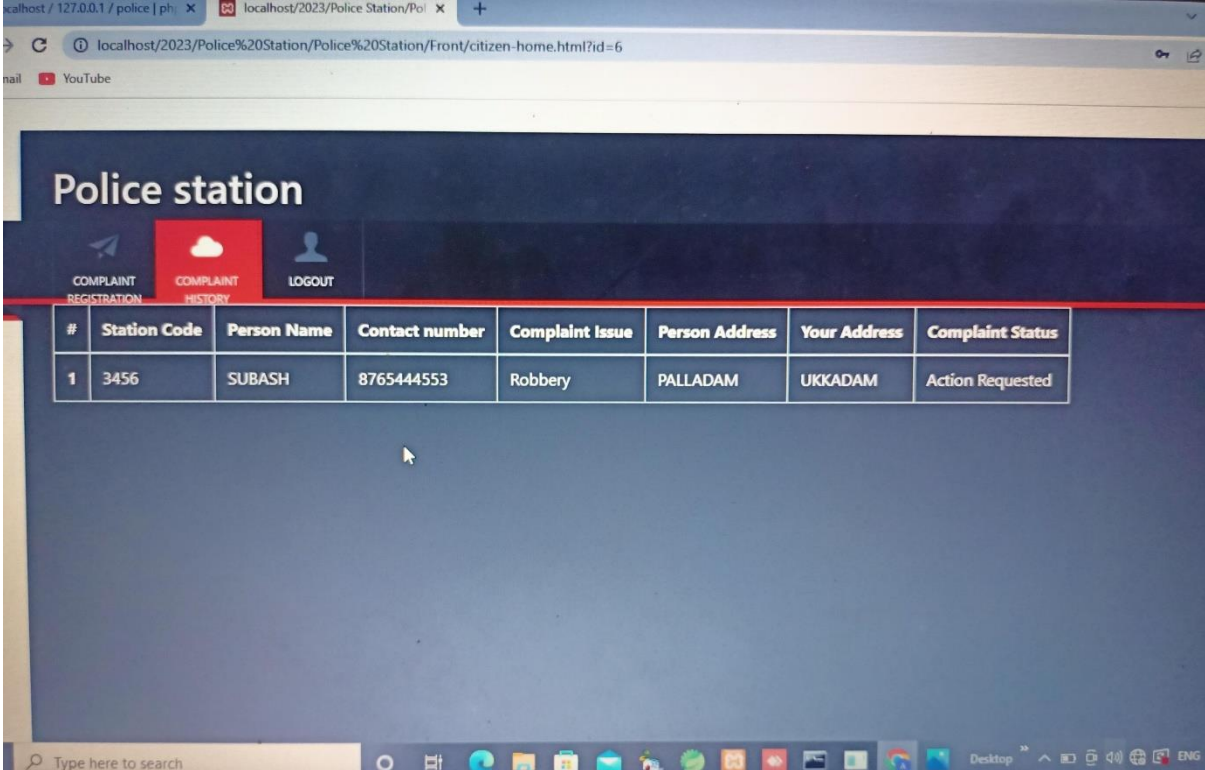
Desktop 10:07 07-03-2023

localhost says

Complaint Saved Sucessfully

OK

# COMPLAINT HISTORY & LOG OUT



The screenshot shows a web browser window with two tabs. The active tab is titled 'localhost/2023/Police Station/Police%20Station/Front/citizen-home.html?id=6'. The page content is titled 'Police station' and features a navigation bar with three options: 'COMPLAINT REGISTRATION', 'COMPLAINT HISTORY' (which is highlighted in red), and 'LOGOUT'. Below the navigation bar is a table displaying complaint history.

#	Station Code	Person Name	Contact number	Complaint Issue	Person Address	Your Address	Complaint Status
1	3456	SUBASH	8765444553	Robbery	PALLADAM	UKKADAM	Action Requested

The bottom of the image shows a Windows taskbar with a search bar and various application icons.