SAINT MICHAEL COLLEGE OF CARAGA

BUENAVISTA TRAFFIC MANAGEMENT OFFICE REGULATION AND RECORD MANAGEMENT SYSTEM

A Capstone Project Presented To
The Faculty of
College of Computing and Information Sciences

In Partial Requirements for the Degree
Bachelor of Science in Information Technology

By

MARIA ALLYSA D. CAPAGNGAN
JOHN B. CATALAN
KHYLE NENON EGOS
DEBORAH GRACE U. JUMAWAN
JENNIFER M. ROSALES

ADVISER: RONNEL A. FALO

APRIL 2025

SAINT MICHAEL COLLEGE OF CARAGA
College Of Computing and Information Sciences

CERTIFICATE OF RESEARCH APPROVAL

This capstone entitled **"BUENAVISTA TRAFFIC MANAGEMENT OFFICE REGULATION AND RECORD MANAGEMENT SYSTEM"** presented and submitted by **MARIA ALLYSA D. CAPAGNGAN, JOHN B. CATALAN, KHYLE NENON EGOS, DEBORAH GRACE U. JUMAWAN,** and **JENNIFER M. ROSALES** in partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE AND INFORMATION TECHNOLOGY** is hereby accepted and recommended for Oral Examination.

RONNEL A. FALO
Adviser

LEALIL I. PALACIO, MSIT (CAR)
Member

MARISOL S. ROSARIO
Member

MICHELLE ANN G. LUCINO
Member

DAISA O. GUPIT, MIT
Member

MARLON JUHN M. TIMOGAN, MIT
Member

KENNETH IAN B. BARRERA, MA
Chairperson

ACCEPTED as partial fulfillment of the requirements for the degree of Bachelor of Science and Information Technology.

APPROVED by the tribunal at the Oral Examination with the grade of _____.

RONNEL A. FALO
Adviser

LEALIL I. PALACIO, MSIT (CAR)
Member

MARISOL S. ROSARIO
Member

MICHELLE ANN G. LUCINO
Member

DAISA O. GUPIT, MIT
Member

MARLON JUHN M. TIMOGAN, MIT
Member

KENNETH IAN B. BARRERA, MA
Chairperson

COLLEGE OF COMPUTING AND INFORMATION SCIENCES

ABSTRACT

TITLE:	Buenavista Traffic Management Office Regulation And Record Management System
AUTHOR:	Maria Allysa D. Capagngan, John B. Catalan, Khyle Nenon Egos, Deborah Grace U. Jumawan, Jennifer M. Rosales
DEGREE:	Bachelor of Science in Information Technology
ADVISER:	Ronnel A. Falo
PLACE OF PUBLICATION:	Saint Michael College of Caraga
DATE	April 2025
PAGES	202

DEVELOPMENTAL RESEARCH

I. OBJECTIVES

This study aimed to develop and implement a Web and Mobile-Based Traffic Regulation and Record Management System (TRRMS) for the Buenavista Traffic Management Office (BTMO). Specifically, it aimed to:

1. automate the recording and management of traffic violations, vehicle registrations, and driver details;
2. generate real-time reports and visual statistics to support efficient decision-making; and
3. enable enforcers to issue tickets using a mobile app integrated with Bluetooth printing.

II. METHODOLOGY

The system was developed using the Systems Development Life Cycle (SDLC) methodology, ensuring a systematic approach from planning to deployment. Unified Modeling Language (UML) tools such as use case, activity, and sequence diagrams guided the system's design, while an Entity-Relationship Diagram (ERD) structured the database for data integrity and relational efficiency. Development was carried out using PHP and MySQL, with Visual Studio Code

as the primary development environment. The resulting system featured a responsive, intuitive interface with efficient data handling and streamlined user interactions tailored to meet the study's objectives.

III. FINDINGS

The implementation of TRRMS significantly enhanced BTMO operations by replacing inefficient manual methods with a centralized, digital platform. Evaluation using ISO 25010 standards revealed that the system was highly rated in terms of functional suitability (3.75), performance efficiency (3.88), and usability, leading to a grand mean of 3.81. Enforcers and staff reported faster processing, better access to records, and reduced paperwork. Mobile and web-based features worked seamlessly in synchronizing data and issuing real-time violation reports.

IV. RECOMMENDATIONS

To further enhance the system's functionality, it is recommended to integrate offline capabilities for enforcers operating in remote areas and add features like SMS or email notifications for violators. Additionally, expanding user role management and considering cloud-based hosting can improve scalability, accessibility, and data security. Regular staff training should also be conducted to ensure effective use of the system.

KEYWORDS: *Traffic Management System, Web Application, Mobile App, Record Automation.*

DEDICATION

With heartfelt gratitude, this capstone project is dedicated to the individuals and institutions who have guided, supported, and inspired the researchers throughout their academic journey.

To their families, sincere thanks are extended for the endless love, patience, and encouragement. The unwavering belief of their loved ones served as a source of strength through every obstacle and milestone encountered during the project.

To their adviser, Mr. Ronnel A. Falo, the researchers express deep appreciation for his consistent support and guidance. His expertise, insightful feedback, and mentorship were instrumental in shaping the direction and success of the study. His dedication has left a meaningful impact on their academic and personal growth.

To the College of Computing and Information Sciences (CCIS) of Saint Michael College of Caraga, the researchers are truly grateful for the education, training, and opportunities provided. The college's commitment to excellence in information technology and innovation empowered them to complete this project with confidence and purpose.

This work also reflects the collective effort, collaboration, and shared commitment of the research team. Each member's contribution played a vital role in bringing the system to life, with the goal of making a practical and positive impact on the community.

This capstone project would not have been possible without the support and encouragement of all who stood beside the researchers. Their involvement is sincerely appreciated as part of this meaningful achievement.

ACKNOWLEDGEMENTS

First and foremost, the researchers extend their deepest gratitude and praise to God for His constant guidance, wisdom, and strength throughout their journey. It was through His grace that this capstone project reached completion. His presence sustained the researchers through every challenge and led them toward success.

The researchers sincerely thank their capstone adviser, Mr. Ronnel A. Falo, for his exceptional mentorship, patience, and unwavering support. His valuable insights, constructive feedback, and encouragement were instrumental in the successful completion of this study and contributed greatly to the researchers' academic and personal growth.

Gratitude is also expressed to the dedicated faculty and staff of the College of Computing and Information Sciences (CCIS) at Saint Michael College of Caraga. Their expertise, guidance, and commitment provided the foundation and inspiration for this academic endeavor.

To their families, the researchers offer heartfelt thanks for their endless love, understanding, and support. The sacrifices and encouragement of their loved ones inspired them to persevere and give their best throughout the project.

Sincere appreciation is also extended to the Tertiary Education Subsidy (TES) program for the financial assistance that enabled the researchers to focus on their academic pursuits without additional burdens.

The researchers also express their heartfelt thanks to their friends, classmates, and mentors who offered assistance, encouragement, and companionship throughout the process. The kindness and support shown by these individuals meant more than words can express.

Lastly, to all who contributed in any way—thank you. This accomplishment is shared with everyone who offered their support, and the researchers are proud to celebrate this milestone with them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Project Context

The Buenavista Traffic Management Office (BTMO) is the main center for traffic safety and regulation in Buenavista. It manages traffic law enforcement, keeps vehicle and driver records, and handles registration and documentation. Traffic enforcers are essential to this work, interacting with drivers, issuing tickets, and managing traffic flow [1]. However, the current reliance on manual methods, including Excel, for recording violations, vehicle registrations, and driver information results in time-consuming processes, difficulty in maintaining accurate records, and a higher risk of errors.

M. E. Matlala et al. highlight that digital recordkeeping can significantly improve public service. By making data more accurate and reducing paperwork, digital systems help with administrative tasks, lower the risk of lost documents, and improve workflows. This results in faster, more reliable services, increased citizen satisfaction, and a more responsive public service system [2].

Technology is essential for smoother operations, better enforcement, and more accurate traffic regulation in today's traffic management. Manually recording violations makes it difficult for the traffic office to track offenses and organize citations, often leading to inconsistent results [1].

Over the years, the Buenavista Traffic Management Office (BTMO) has used manual recordkeeping for traffic tasks, which heavily relies on physical documents and extensive data entry. This method has proven to be slow, as processing and retrieving information often require significant time and effort. Additionally, it is prone to errors, such as misplaced files, inaccurate entries, and difficulty in tracking updates. These challenges hinder the efficiency of traffic management operations and create delays in decision-making and enforcement.

To address these challenges at BTMO, the ideal solution is to create an automated Traffic Regulation and Record Management System (TRRMS). This system would include a web platform and a mobile app to simplify traffic management. Features like auto-generated reports and data visualizations would improve efficiency and accuracy. By reducing manual work, the system would make data management more accessible and offer more flexibility and accessibility. Ultimately, it would significantly improve BTMO's ability to manage traffic regulations and records more effectively.

The Buenavista Traffic Management Office (BTMO) would implement and improve a centralized record management system to address current traffic issues. A web and app-based platform can enhance BTMO's performance by providing a single system to record and manage driver details, vehicle registrations, and traffic violations. This user-friendly interface simplified operations, improved data accuracy, and ensured greater efficiency and reliability in traffic management processes.

1.2 Objective of the Study

This study aimed to design, develop, and implement an automated Traffic Regulation and Record Management System (TRRMS) for the Buenavista Traffic Management Office (BTMO). The system included both a web platform and a mobile app.

Specifically, the study aimed to:

1. automate the recording and management of traffic-related activities;
2. provide auto-generated statistical reports on pending and resolved violations, with visual charts and weekly or monthly summaries for easy presentation and documentation; and
3. enable enforcers to issue citations efficiently through a mobile application with pre-configured ticket templates.

1.3 Scope and Limitations

The BTMO-TRRMS project aimed to develop a digital solution for the Buenavista Traffic Management Office (BTMO), replacing manual recordkeeping with a streamlined, database-driven system. Its goal was to simplify traffic-related tasks, including vehicle registration, license issuance, and traffic enforcement. The system would be accessible only to authorized BTMO users and would operate within the office's local area network (LAN) to ensure data security and reliability.

The system would include a web-based platform and an Android mobile app. BTMO staff, including administrators and authorized personnel, would use the web platform to generate reports, manage records, and update system information. Traffic enforcers would use the mobile app to access real-time data and handle enforcement tasks.

The system has several limitations related to its software and hardware requirements. The mobile app requires a minimum of 4GB RAM for smooth performance, with potential degradation during heavy data usage or simultaneous multi-user access. Testing using XAMPP may not accurately reflect live performance, particularly if the server infrastructure is not scalable. Additionally, the mobile thermal receipt printer relies on Bluetooth connectivity and battery life, which can pose challenges in remote areas or during prolonged usage.

1.4 Definition of Terms

The following terms explain the core concepts and components that shaped the design and development of the BTMO Traffic Regulation and Record Management System. They show how these concepts are applied within the context of this research.

API (Application Programming Interface) – A set of protocols and tools that allows different software applications to communicate with each other. APIs will enable smooth data exchange between the web-based system and the mobile app used by traffic enforcers and motorists.

App-Based Platform – A mobile application designed for traffic enforcers to record and manage motorists' personal and vehicle information and traffic violations directly from their mobile devices.

Buenavista Traffic Management Office (BTMO) - Is a branch of the local government responsible for regulating and overseeing traffic-related activities, including traffic enforcement, vehicle registration, driver compliance, and road safety initiatives to maintain order and efficiency on the roads.

Centralized Database System - A unified system stores records of violations, vehicle registrations, and other related data in a single database. This was expected to improve data management by reducing manual work and ensuring more efficient record handling.

Data Interoperability - The ability of different software programs and systems to exchange and use data seamlessly. For BTMO, this means that information would flow smoothly across multiple platforms, improving collaboration and productivity within the system.

End-User - The people who will use and interact with the BTMO system. Primary users include traffic enforcers who log violations and manage records, while secondary users are motorists who access the system for vehicle registration and fine payments.

LAN (Local Area Network) – A network that connects computers and devices within a limited area, such as the BTMO office. It allows users to access the system's database and executable files without an internet connection.

SMTP (Simple Mail Transfer Protocol) – An internet standard protocol for sending and receiving email messages between servers. In BTMO, SMTP will be used for notifications, alerts, and reports on traffic violations or system updates to keep stakeholders informed.

Traffic Management Analytics Report – A structured report generated by the BTMO system that provides data on traffic violations, vehicle registrations, and other relevant metrics over a specified period.

User Acceptance Testing (UAT) – A phase where users test the system to ensure it meets their needs and requirements. This testing is done before the system is released to the public and helps identify any remaining issues or user satisfaction concerns.

Web-Based System – A software application that users can access through a web browser. It allows users to interact with the system from any device with internet access, typically involving a central database for storing and managing data.

CHAPTER II

REVIEW OF RELATED LITERATURE

This chapter presents a review of relevant literature related to the study. It focuses on the need to implement a digital solution for traffic management, its benefits, and the challenges it addresses. The literature is sourced from electronic materials, e-books, websites, articles, and journals.

Web-Based Traffic Regulation and Record Management System

The increasing challenges in traffic management, especially in developing countries, have led to the development of innovative solutions like the Web-Based Expert System (TRSys). This system supports traffic experts by offering guidelines and procedures for selecting effective traffic policies, drawing on knowledge from various sources to address traffic-related issues [3]. Here is the revised version with the numbers included:

Traditional methods, like writing down incidents in case books, showed the need for better systems. This led to the development of solutions like the Traffic Case Management System in Bangladesh [4]. The "correlation recording system" also helps assess automated traffic enforcement by evaluating violations and their impact on accidents [5].

Web-based crime record systems highlight the importance of accurate recordkeeping and smooth information sharing between agencies, essential for informed decision-making [6]. The design of traffic enforcement facilities also focuses on improving safety and preventing accidents [7]. Despite ongoing government efforts to reduce traffic accidents, rising rule violations like running red lights have led to new penalties to increase compliance and improve road safety [8].

The introduction of a Web-Based Vehicle Monitoring System at MSU-LNAC aims to replace the outdated and error-prone manual logbook system [9]. This shift is part of a more significant trend in traffic management, demonstrated by successful collaborations like the one in Phuket,

where multiple government agencies joined forces to create a sustainable traffic law enforcement system. This initiative has enhanced law compliance, improved road safety, and offers potential for future expansion [10]. A study on a Commercial Vehicle Ticket Reservation System using the CodeIgniter Framework further highlights the importance of evaluating the performance of web-based systems [11].

Proposed web-based systems for traffic police make it easier to collect and analyze data, allowing authorized personnel to access traffic information and generate reports quickly. These systems also help educate drivers about road rules and signs [12]. Additionally, using the Mivar system with driver assistance technology can help monitor traffic violations and warn drivers, which may reduce accidents [13].

Automating traffic offense detection makes it easier for law enforcement to manage traffic, freeing up resources for other tasks. As traffic management becomes more complex, web-based systems can improve efficiency and compliance. Systems like the Traffic Regulation and Record Management System help organize records, make data analysis more accessible, and help experts create better safety policies. Automated enforcement and driver assistance technologies also help law enforcement track violations, improving compliance and reducing accidents.

Mobile Application Utilization in Traffic Enforcement

Mobile apps are becoming more critical in traffic enforcement, helping police improve ticketing systems and monitor violations in real time [15]. These apps use technology to detect offenses continuously, making it easier for law enforcement to manage traffic efficiently. As smartphones and apps become more widespread, they collect data, which can help mobile apps help police understand traffic patterns better and make smarter decisions [16]. The World Health Organization (WHO) says that even a small 5% reduction in average speed can lower fatal accidents by 30%. This shows the importance of using mobile apps to encourage safety habits like

wearing helmets and seatbelts, which can reduce serious injuries and deaths [17]. However, the growing traffic violations make it harder for police to keep up. Fines and penalties lose effectiveness if civilians and officers do not follow the rules [18].

To solve these problems, integrated enforcement systems have been created to ensure violations are identified, tickets are issued, payments are tracked, and everything is linked to vehicle registrations and other records. This makes enforcement more accessible, faster, and transparent [19].

As the number of traffic fines, tickets, and arrests increases, along with stricter penalties, there is a need for better solutions. One solution is a mobile app for issuing traffic tickets in Ilocos Norte, which could make the process easier and faster [20]. Another solution is the Electronic Traffic Law Enforcement (ETLE) system used by the Traffic Directorate of Polda Metro Jaya. This system helps by recording traffic violations electronically, making it faster to respond, keeping things in order, and improving safety [21].

With more traffic violations, especially on highways, causing traffic jams and accidents, the E-Tilang service helps by allowing quicker ticketing and making enforcement more transparent [22]. The shift to digital is clear as paper-based ticketing is replaced with electronic systems, letting officers quickly record violations and issue tickets [23].

Using automated traffic enforcement technologies is an important step in improving traffic management. These systems help monitor violations and allow real-time action, making it easier to manage traffic offenses effectively [24].

To sum up, mobile apps are becoming a vital tool for traffic enforcement. By using smartphones, police can collect necessary data and improve how they enforce rules, like encouraging helmets and seatbelts. However, with the rise in traffic violations and enforcement

issues, it is essential to have systems that make ticketing easier and help people follow the rules. These changes are crucial for making traffic law enforcement more efficient and transparent.

Environmental Impact of Reducing Paperwork in Public Offices

This paper seeks to address the gap in research on the benefits and challenges of adopting a paperless approach across various sectors. The review provides an objective analysis of the shift to paperless systems [25]. Findings suggest that understanding the environmental impact of products and technology, particularly in reducing paper consumption, is crucial [26]. Data shows that global paper usage has stabilized and is declining in many regions due to technological advancements such as smartphones and mobile Internet and the time needed for individuals and organizations to adapt to these changes [27].

A qualitative study examined how organizations handle documents to assess paper waste and identify opportunities for savings through new technologies [28]. In Sri Lanka, like in many countries, the heavy reliance on paper-based communication in government offices led the government to invest in Information and Communication Technology (ICT) systems. This shift aims to reduce paper use, address environmental concerns, and improve services for citizens [29].

A paperless office reduces paper-based processes by using digital documents and systems. A Document Management System (DMS), a web application that streamlines document flow and automates tasks, is crucial in supporting a paperless environment [30].

This paper fills the gap in the existing literature by exploring the benefits and challenges of adopting a paperless approach in different sectors. It highlights the importance of looking at the entire lifecycle of products and technologies to understand their environmental impacts. Advances in digital technologies, such as smartphones, are leading to a global decrease in paper usage. Additionally, initiatives like Sri Lanka's shift from paper-based communication to integrated

ICT systems demonstrate a commitment to reducing environmental impact while improving public services through better automation and document management.

CHAPTER III

SOFTWARE REQUIREMENTS AND DESIGN SPECIFICATION

This chapter provides a comprehensive analysis of the technical aspects of designing and developing the proposed system, covering both software and hardware requirements. It also presents an overview of the core functionalities of the Buenavista Traffic Management Office (BTMO) Traffic Regulation and Record Management System (TRRMS).

3.1 System Architecture

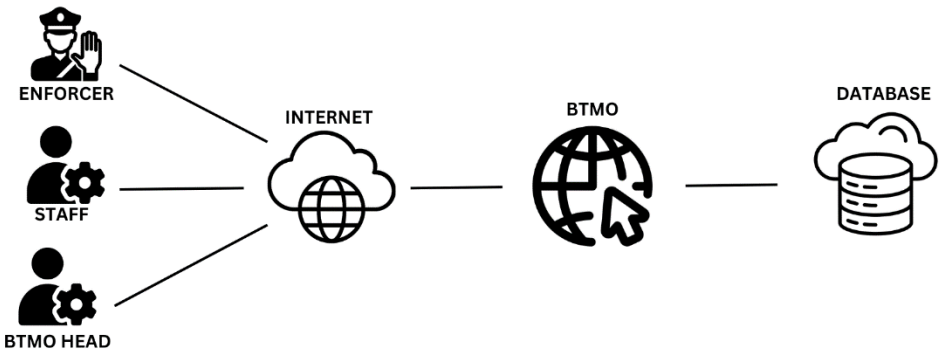


Figure 1. System Architecture

Figure 1 illustrates the process for accessing the Traffic Regulation and Record Management System (TRRMS), which supports three user roles: Enforcers, Staff, and the BTMO Head. Enforcers can log in and access the homepage to view violators and issue citations. Staff members can access the dashboard and manage traffic records. The BTMO Head holds the highest level of access, with the authority to manage user accounts, roles, and all system functionalities.

3.2 Conceptual Diagram

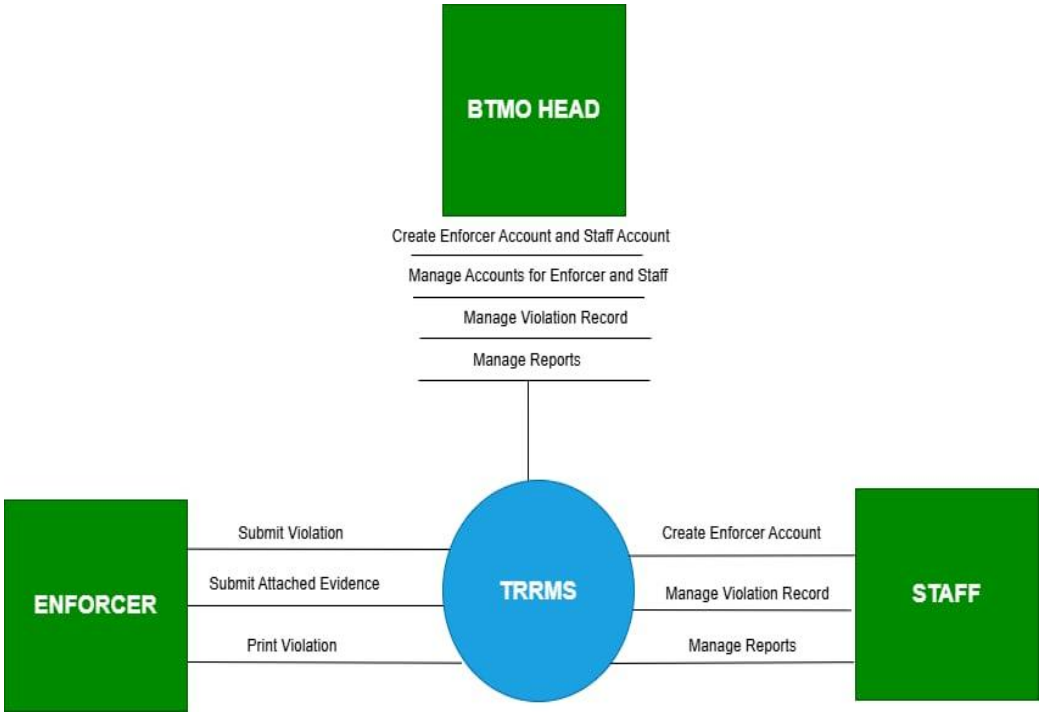


Figure 2. Conceptual Diagram

Figure 2 shows how the TRRMS architecture assigns responsibilities to different users. The BTMO Head has full system access, managing Staff accounts, creating Enforcer accounts, and overseeing all functions. Staff handle traffic records and administrative tasks. Enforcers are responsible for printing and submitting violation tickets.

3.3 Use Case Diagram

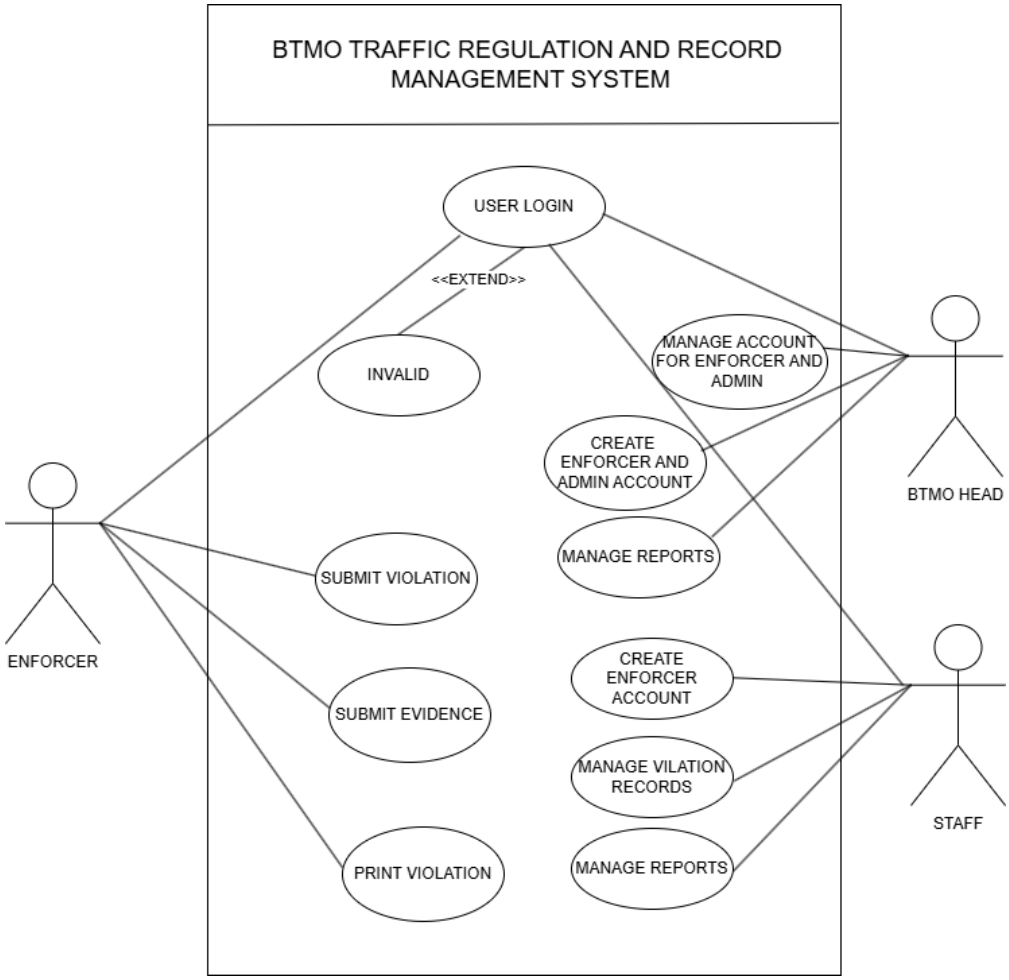


Figure 3. Use Case Diagram

Figure 3 highlights the main features and roles for each user type within the system. All users must log in, and the staff is responsible for registering all users except for themselves. Enforcers can submit violations, upload evidence, and print violation tickets. The BTMO Head has full administrative authority, including managing staff accounts, creating enforcer accounts, and overseeing all system functions.

3.4 Activity Diagram

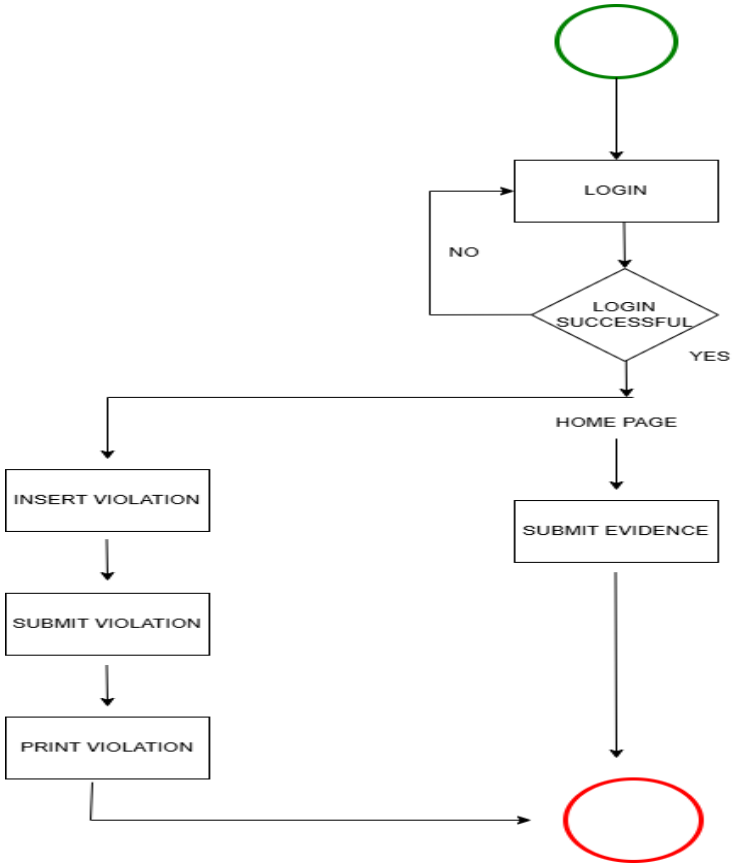


Figure 4. Activity Diagram – Enforcer's

Figure 4 outlines the activities enforcers can perform within the system. After reaching the login page, enforcers can log in successfully. If they forget their username or password, they can contact the staff for help. If they do not have an account, they can request to the staff to create one. Upon logging in, enforcers are directed to the TRRMS homepage, where they can enter, submit, and print violations and upload evidence. Enforcers can also log out when necessary.

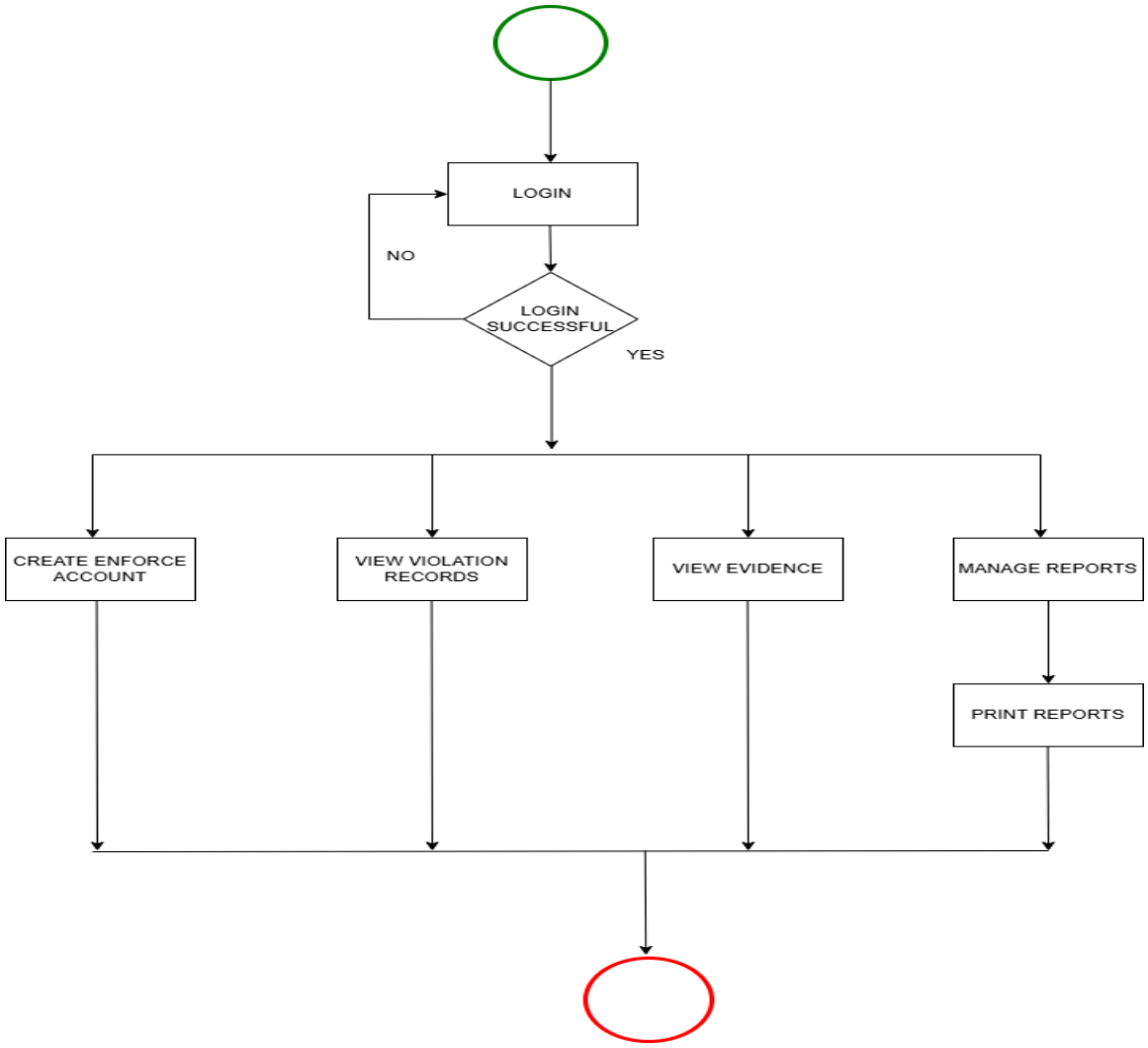


Figure 5. Activity Diagram – Staff

Figure 5 illustrates the tasks available to the staff. After successful verification, the staff is directed to the homepage, where their primary responsibilities include creating and viewing enforcer accounts. Additionally, Staff can view violation records, view evidence, manage reports, and print reports.

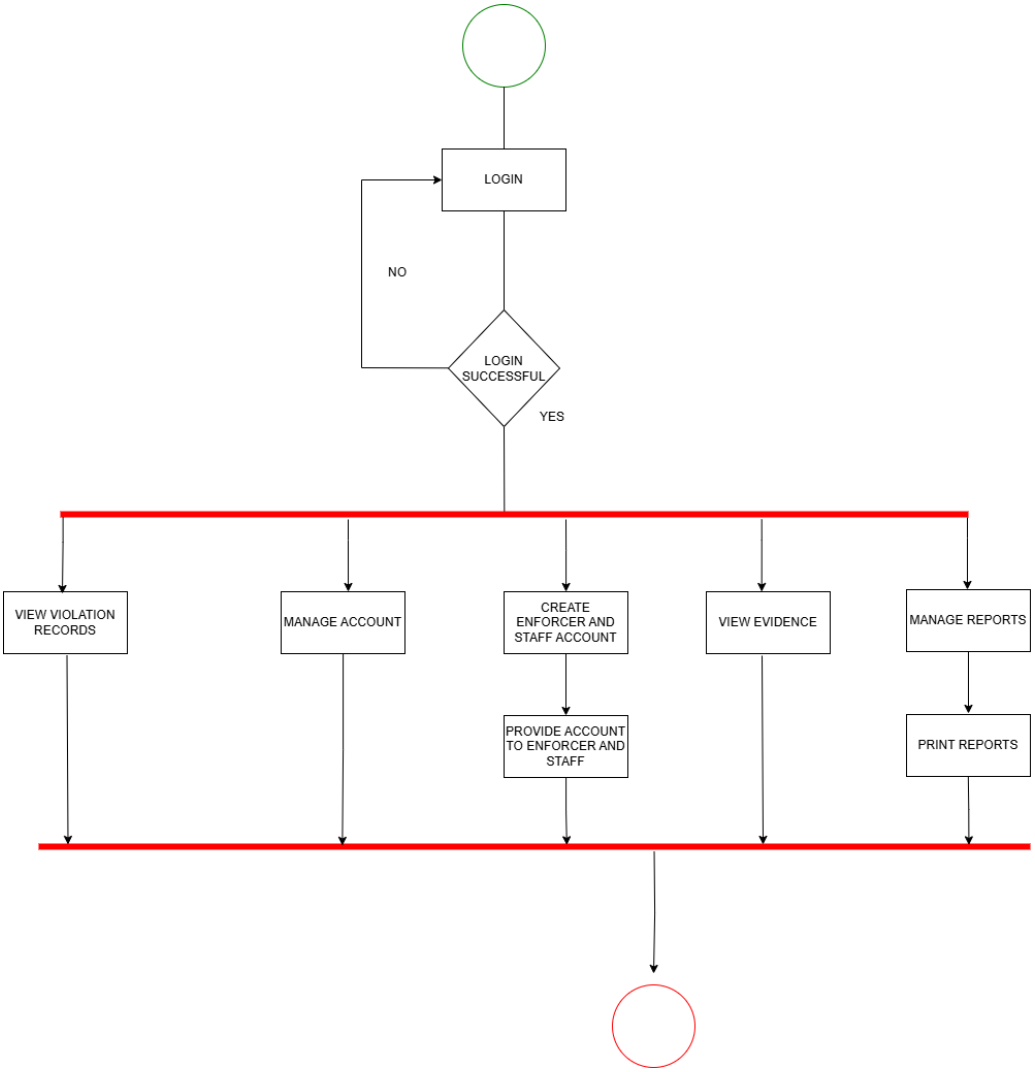


Figure 6. Activity Diagram – BTMO Head

Figure 6 outlines the activities available to the BTMO Head within the Regulation and Record Management System (RRMS). The BTMO Head is responsible for managing user accounts, including creating and managing enforcer and staff accounts, as well as overseeing all system functions. Additionally, the BTMO Head has full administrative control over the system's operations.

3.5 Sequence Diagram

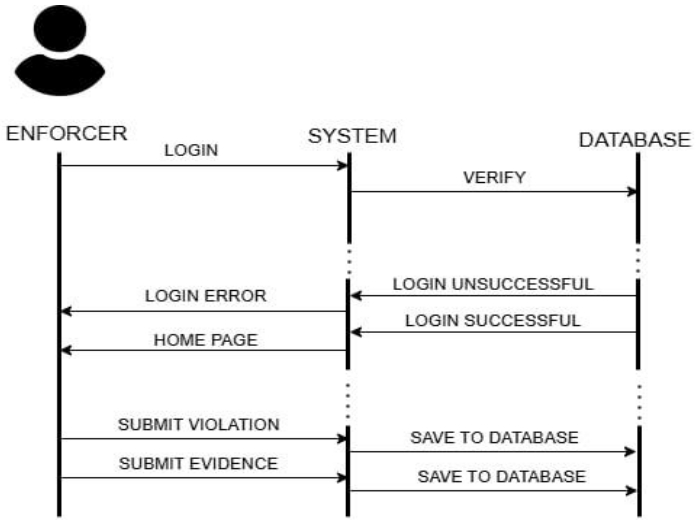


Figure 7. Sequence Diagram – Enforcer's

Figure 7 describes the steps for enforcers. Starting at the login interface, enforcers complete the verification process. Once logged in, they can submit violations, upload evidence, and store data within the system.

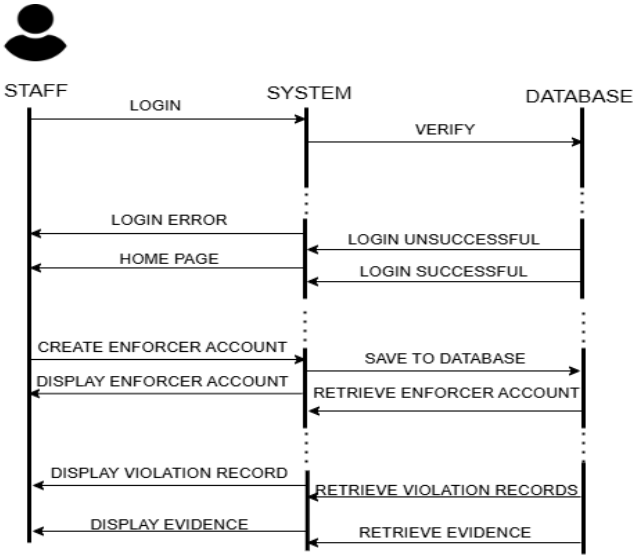


Figure 8. Sequence Diagram – Staff

Figure 8 presents the actions available to the staff. The staff can use the login interface to verify their identity, create and view enforcer accounts, manage violation records, and display evidence.

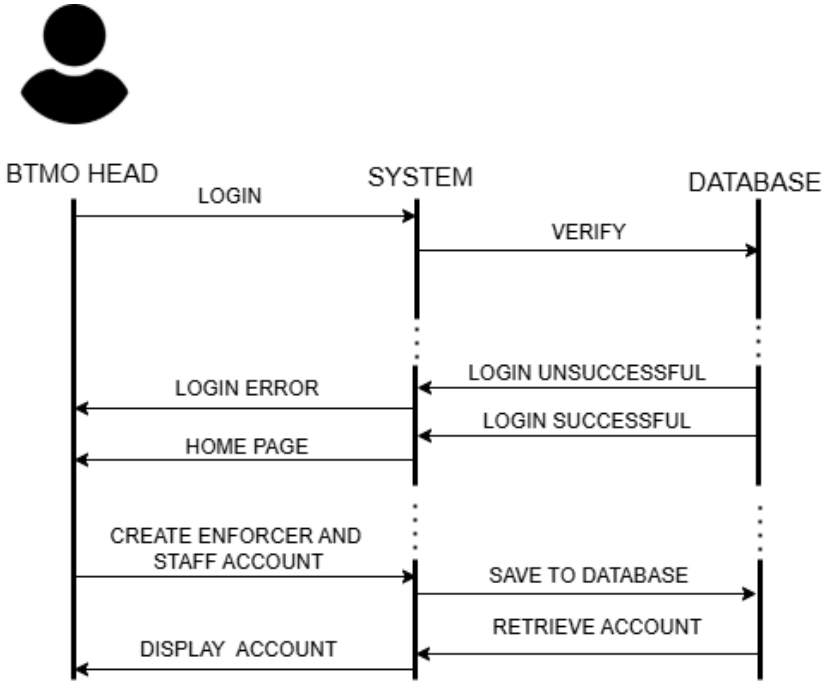


Figure 9. Sequence Diagram – BTMO Head

Figure 9 outlines the BTMO Head's actions within the Traffic Regulation and Record Management System (TRRMS). This diagram details the steps for logging in, verifying identity, managing user accounts, including creating and managing Enforcer and Staff accounts, and overseeing the system's operations. These actions ensure proper administration and functionality of the system.

3.6 ERD (Database Design)

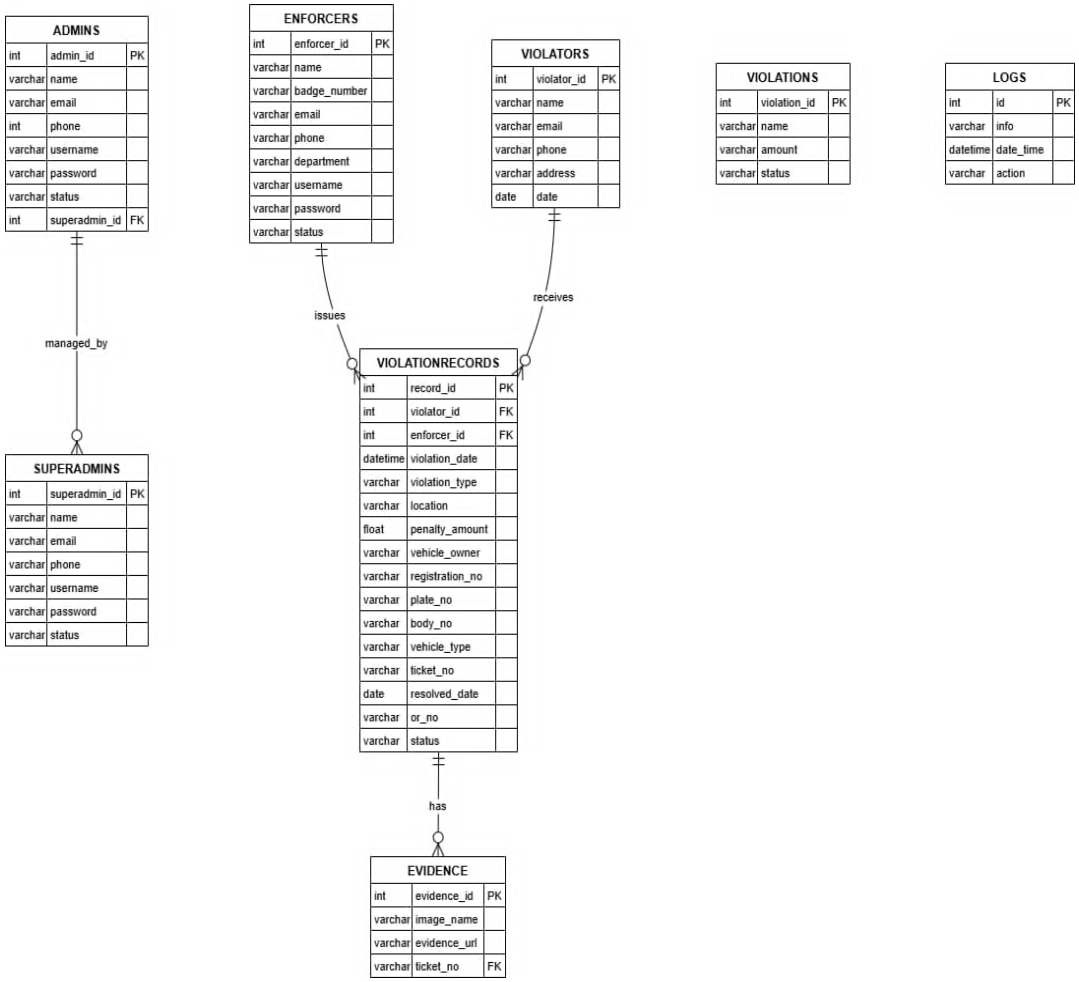


Figure 10. Entity Relationship Diagram – Database Design

Figure 10 displays the entity-relationship diagram (ERD) for TRRMS. The ERD highlights the key entities and their relationships, including BTMO Head, Staff, Enforcers, Violation Records, and Evidence. The BTMO Head manages Staff accounts, while Enforcers are responsible for issuing Violation Records, which contain Evidence. Violators are associated with Violation Records, each of which contains relevant Evidence. This structure enables efficient management and retrieval of violation data, user information, and supporting evidence, ensuring secure recordkeeping and comprehensive tracking.

3.7 User Interface Design (Prototype)

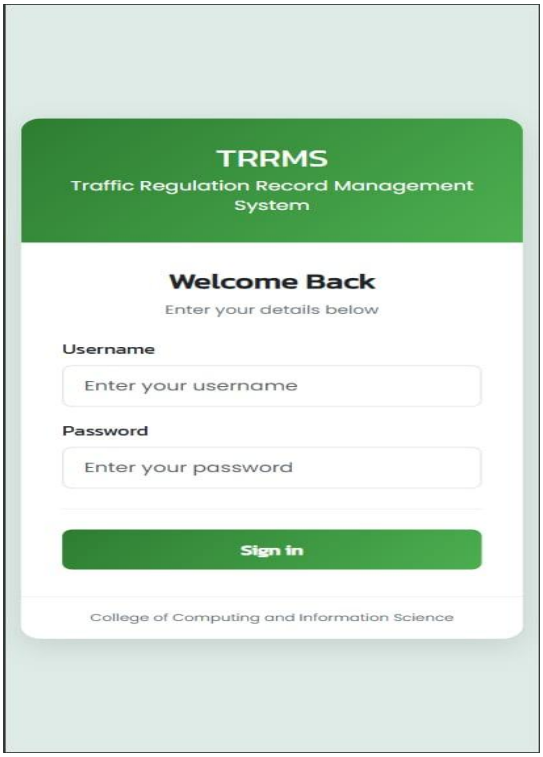


Figure 11. Enforcer Side – Login Page

Figure 11 shows the enforcer-side login page of the app-based Traffic Regulation and Record Management System (TRRMS). This interface requires enforcers to enter their username and password, ensuring secure access to the system for managing violation records, issuing citations and manage enforcement data efficiently.

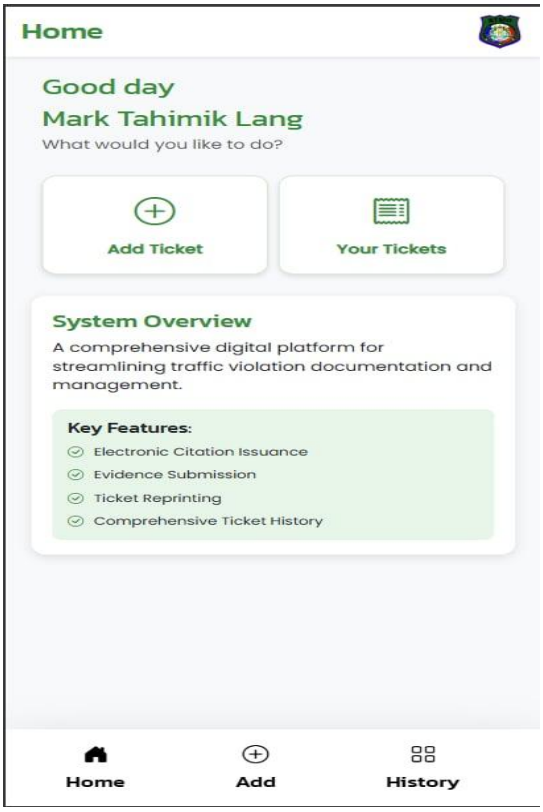


Figure 12. Enforcer Side – Homepage

Figure 12 shows the interface of the homepage that provides an overview of the system and its key features, allowing enforcers to efficiently manage violation records. It includes an "Add Ticket" button for issuing new violations and a "Your Tickets" button for tracking previously recorded cases.

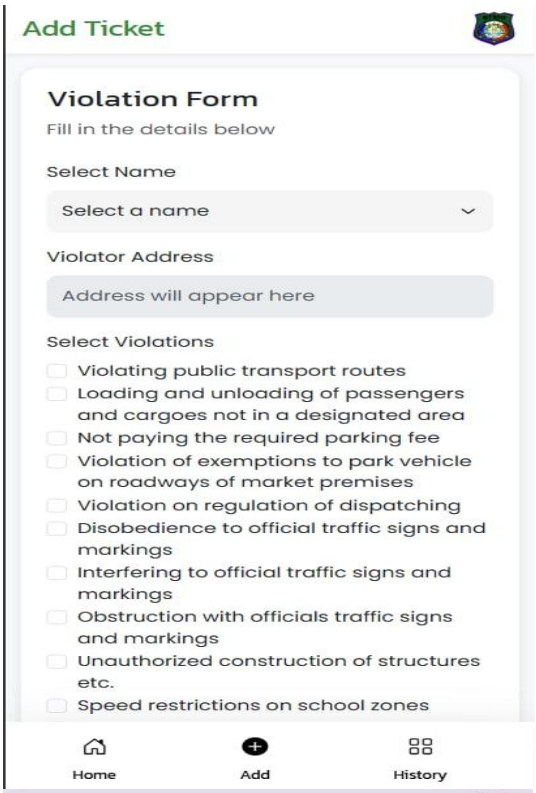


Figure 13. Enforcer Side – Violation Form

Figure 13 shows the process after tapping the "Add Ticket" button, where the system connects to a printer and displays the Violation Form. The enforcer can then select the violator's name, enter their address, and choose the type of violation from a dropdown menu for accurate and efficient record-keeping.

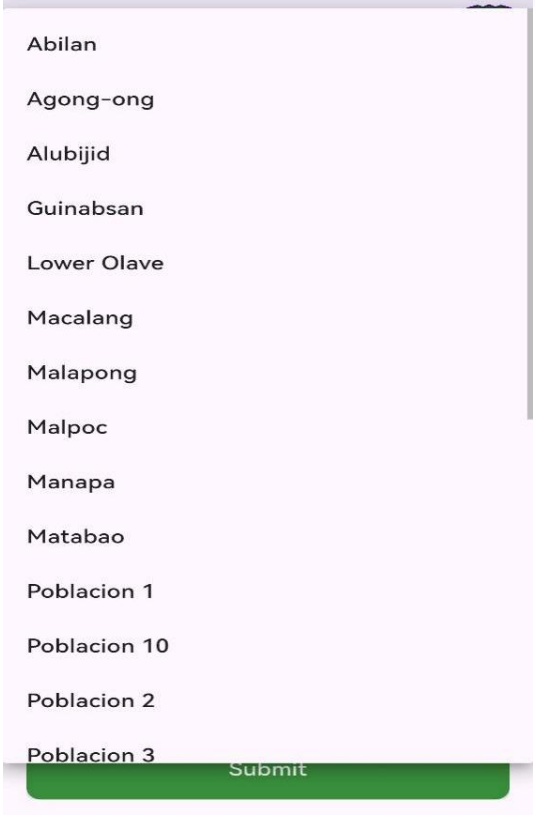


Figure 14. Enforcer Side – Select Barangay of Violation

Figure 14 shows a dropdown menu listing all 25 barangays within the Municipality of Buenavista. Enforcers select the specific barangay where the violation occurred, ensuring accurate location tracking for issued citations.

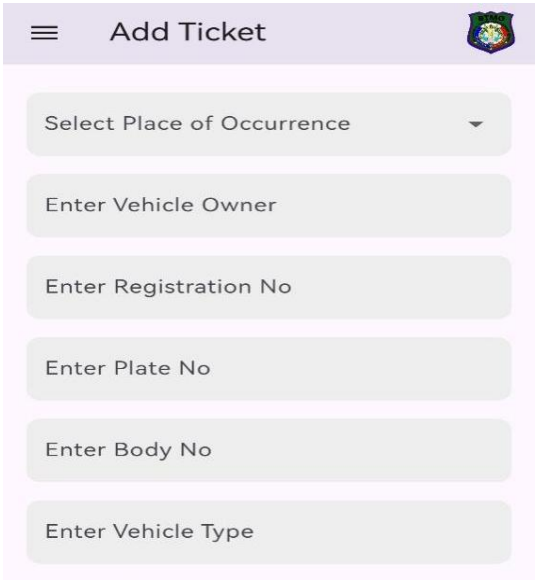


Figure 15. Enforcer Side – Vehicle and Owner Details

Figure 15 captures the essential information, including the vehicle owner's name, registration number, plate number, body number, and vehicle type, ensuring accurate identification and record-keeping for issued citations.

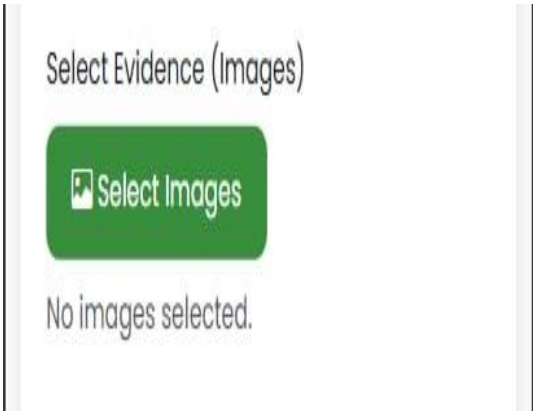


Figure 16. Enforcer Side – Select Evidence Button

Figure 16 shows the Select Evidence interface. Tapping the gallery icon redirects enforcers to the device's gallery, allowing them to choose a specific picture related to the violation for accurate documentation.

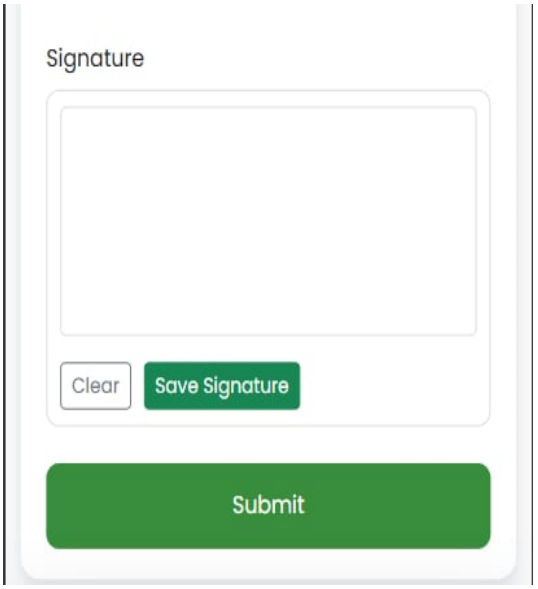


Figure 17. Enforcer Side – Violator Signature

Figure 17 shows the Violator Signature interface, a feature of the app that allows violators to write their signature directly on the phone screen. The interface includes Clear and Save Changes options, enabling accurate and verifiable documentation of the violation.

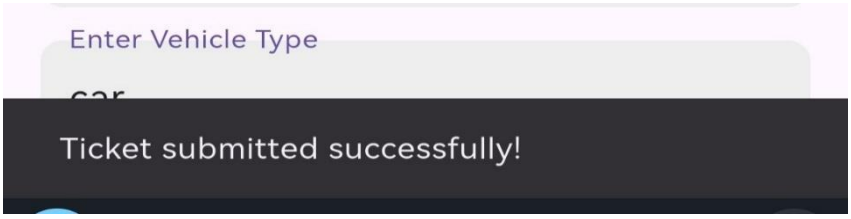


Figure 18. Enforcer Side – Ticket Submitted Notification

Figure 18 shows the "Ticket Submitted Successfully" notification. After entering all details, the system confirms submission and automatically prints the citation ticket, ensuring efficient record-keeping and enforcement.


Republic of the Philippines
PROVINCE OF AGUSAN DEL NORTE
Municipality of Buenavista

Buenavista Traffic
Management Office

Ticket No: 0019

TRAFFIC CITATION TICKET

VIOLATOR DETAILS
Name : Chavit
Address : Sikret
Total Amount: 1000.00

VIOLATIONS & OFFENSES
Disobedience to official traffic signs and markings : 1st
Amount: 500.00

Using cellphones and similar devices while driving : 1st
Amount: 500.00

TOTAL AMOUNT: 1000.00

VEHICLE DETAILS
Owner : Iyyy
Reg. No : 6262
Plate No : 2626
Body No : 7226
Type : car

INCIDENT DETAILS
Date : Mar 28, 2025
Time : 10:01 AM
Location : Abilan

ENFORCER DETAILS
Name : Mark Tahimik Lang
Badge : 0928

Violator Signature

NOTICE:
You are requested to appear to the Office of the Treasurer within (3) days from the date of this notice and provide a photocopy of the citation ticket for your records

Figure 19. Enforcer Side – Sample Generated Print

Figure 19 shows the sample generated print containing all the submitted details. After entering the required information, the system confirms submission and automatically prints the citation ticket, ensuring accurate record-keeping and enforcement.

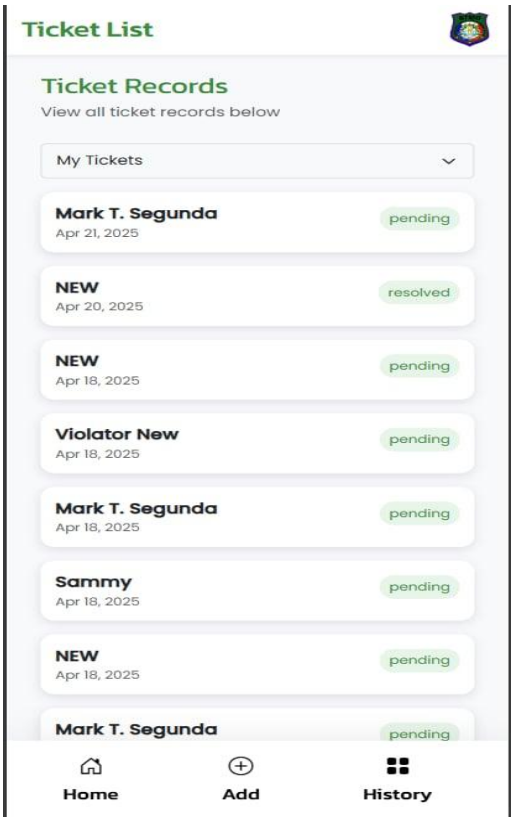


Figure 20. Enforcer Side – Tickets List Page

Figure 20 shows the Tickets List page, which displays the history of previously issued citation tickets. This page allows enforcers to review past violations, ensuring efficient tracking and record-keeping of citation ticket outputs.

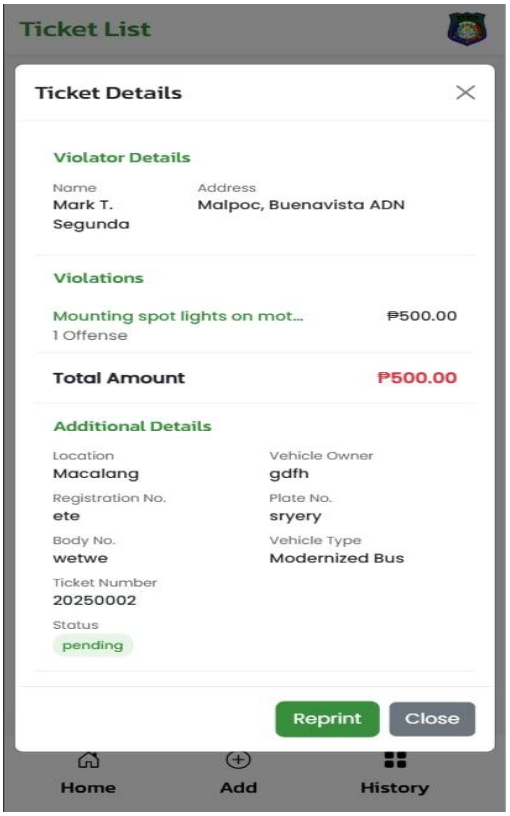


Figure 21. Enforcer Side – Ticket Details View

Figure 21 shows the detailed view of a selected ticket from the Tickets List. Tapping on a ticket displays the violator's details (name, address, total amount), violations & offenses (type of violation, amount, offense count), vehicle details (owner name, registration number, plate number, body number, type), and incident details. The interface also includes Reprint and Close options for convenience.

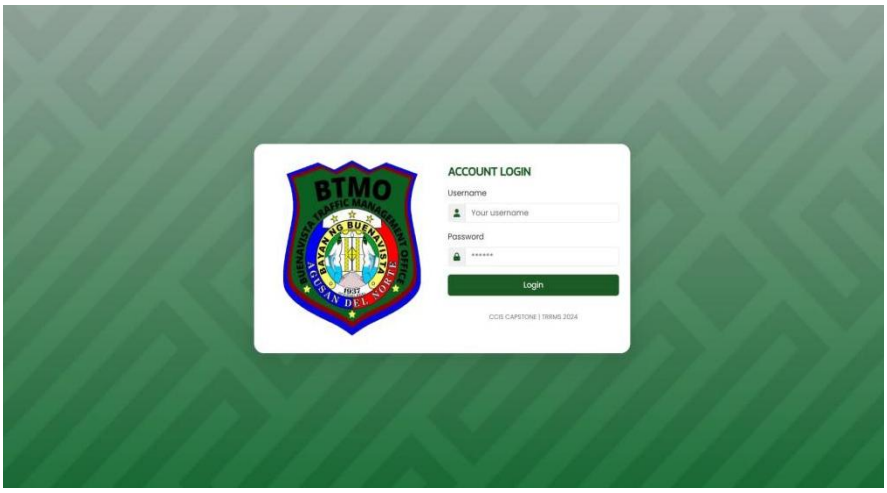


Figure 22. BTMO Head/Staff Side – Login Page

Figure 22 shows the BTMO Head and Staff Login Page, where authorized users enter their username and password to access the system. In case of forgotten credentials, staff can seek assistance from the BTMO Head for account recovery, ensuring secure and controlled access.

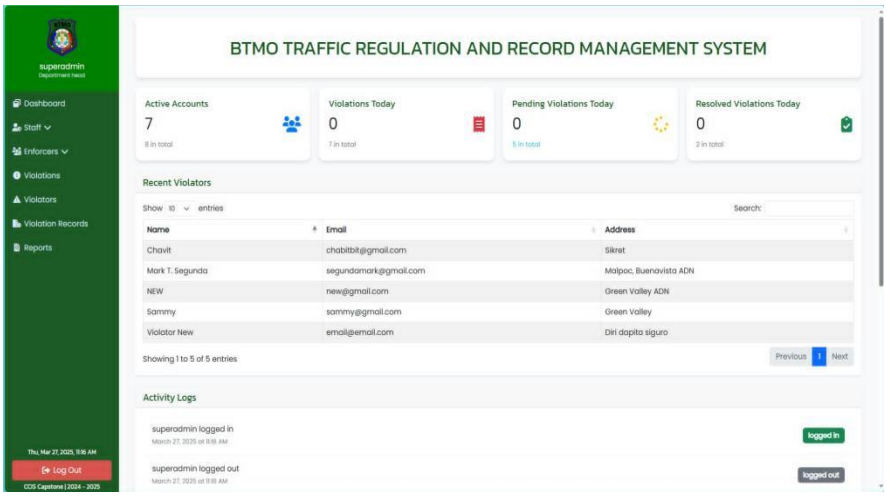


Figure 23. BTMO Head/Staff Side – Dashboard

Figure 23 displays the BTMO Head and Staff Dashboard, showing key enforcement metrics like active accounts, today's violations, pending and resolved cases. It includes a recent violations list and activity logs for real-time monitoring, enhancing data tracking, accountability, and decision-making.

Figure 24. BTMO Head Side – View Staff Accounts

Figure 24 displays the staff account management interface, accessible to the BTMO Head. It allows oversight of staff accounts by showing details like name, email, phone number, username, and status, with options to update, activate, or deactivate accounts for effective administration.

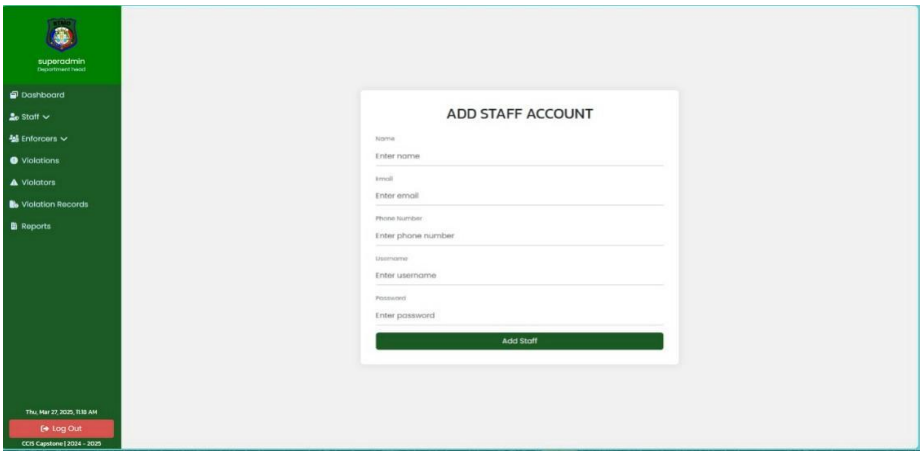


Figure 25. BTMO Head Side – Add Staff Account

Figure 25 showcases the Add Staff Account page, exclusively accessible to the BTMO Head. This interface allows the BTMO Head to register new staff accounts by entering essential details, including name, email, phone number, username and password. Additionally, it features an "Add Staff" button to finalize the registration process, ensuring secure and controlled staff account management.

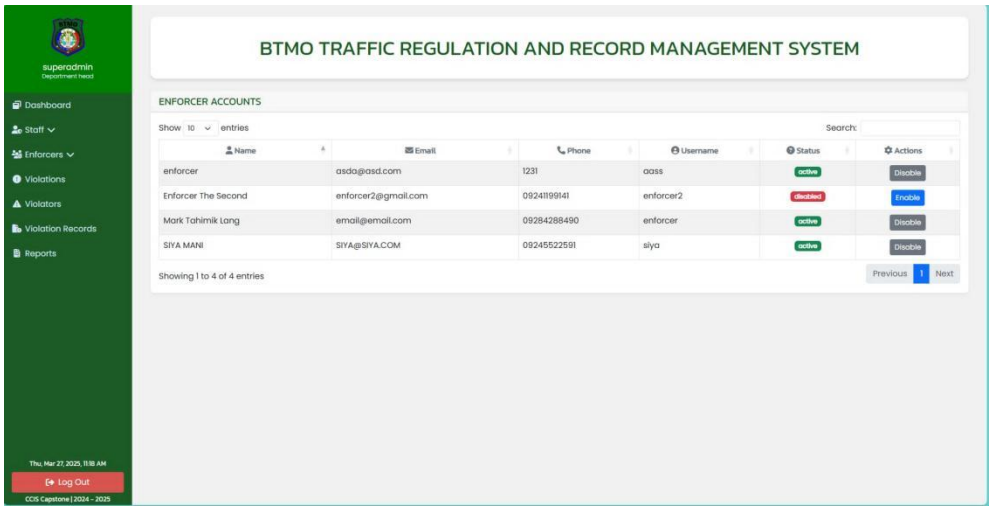


Figure 26. BTMO Head/Staff Side – View Enforcer Accounts

Figure 26 presents the enforcer account management interface, accessible to both the BTMO Head and staff. This page allows them to view enforcer accounts, displaying essential details such as name, email, phone number, username, and account status. Additionally, it provides management options for monitoring and maintaining enforcer account records.

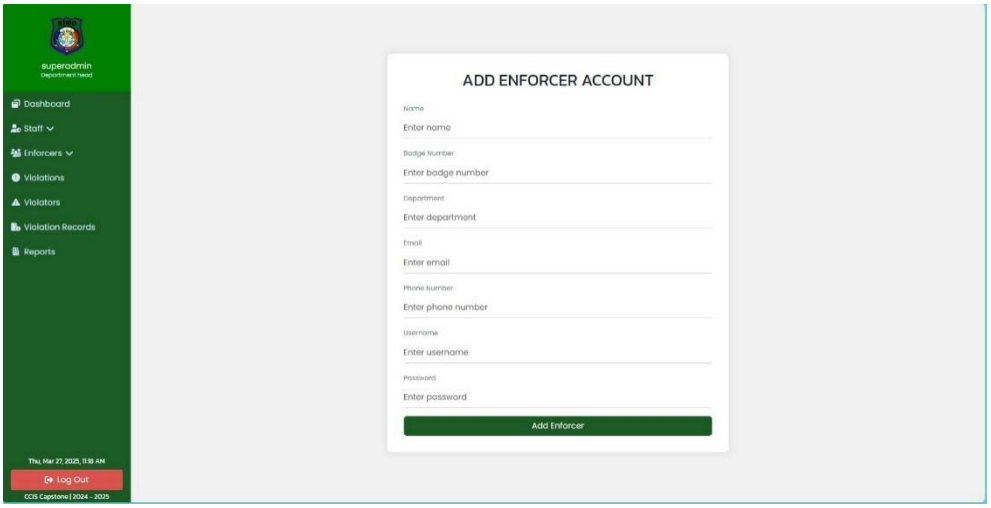


Figure 27. BTMO Head/Staff Side – Add Enforcer Account

Figure 27 showcases the Add Enforcer Account page, accessible to both the BTMO Head and staff. This interface allows them to register new enforcer accounts by entering essential details, including name, badge number, department, email, phone number, username, and

password. Additionally, it includes an "Add Enforcer" button to complete the registration, ensuring proper record-keeping and account management.

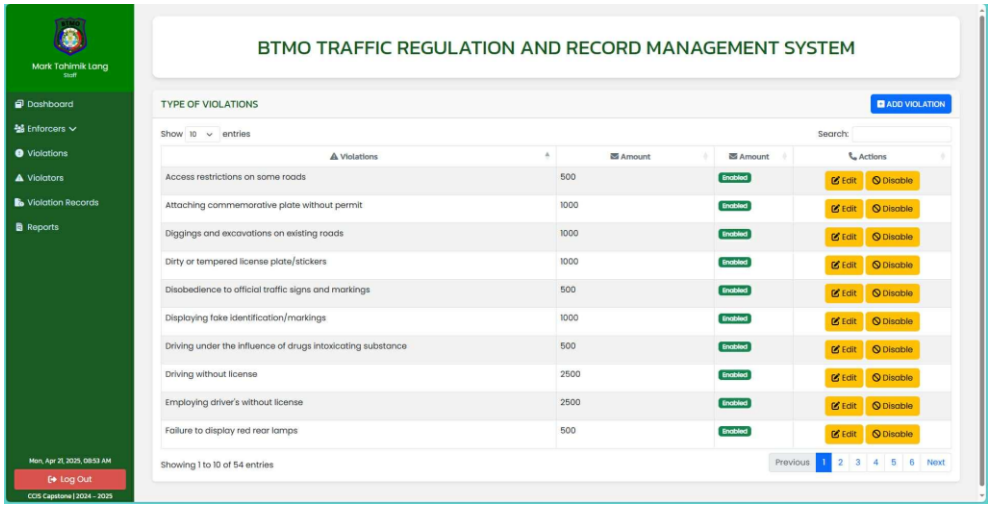


Figure 28. BTMO Head/Staff Side – Violations Page

Figure 28 displays the Violations page, which lists all types of violations along with their corresponding amounts. This interface allows enforcers to manage violations efficiently, featuring options to add new violations, edit existing entries, and delete records as needed.

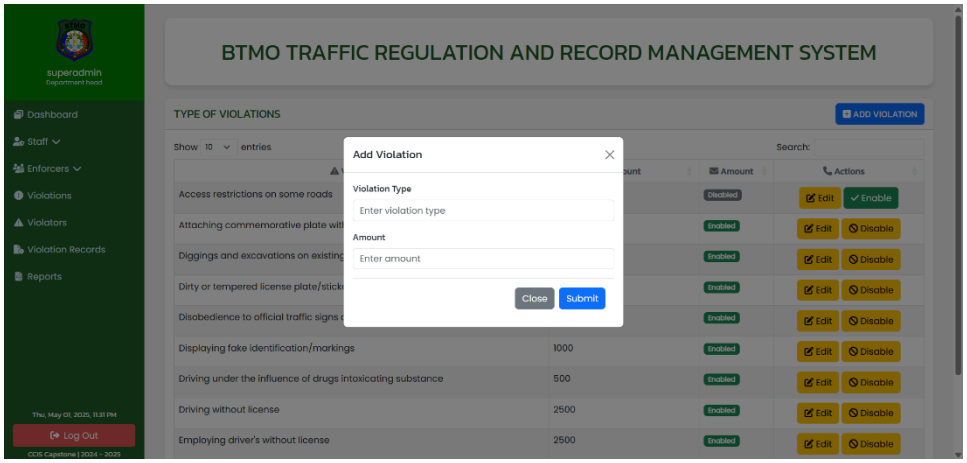


Figure 29. BTMO Head/Staff Side – Add Violation

Figure 29 displays the Violations page, where both the BTMO Head and staff can add new violations. By tapping the Add Violation button, they can enter a violation type and its

corresponding amount. The interface includes Submit and Cancel buttons for finalizing or discarding the entry.

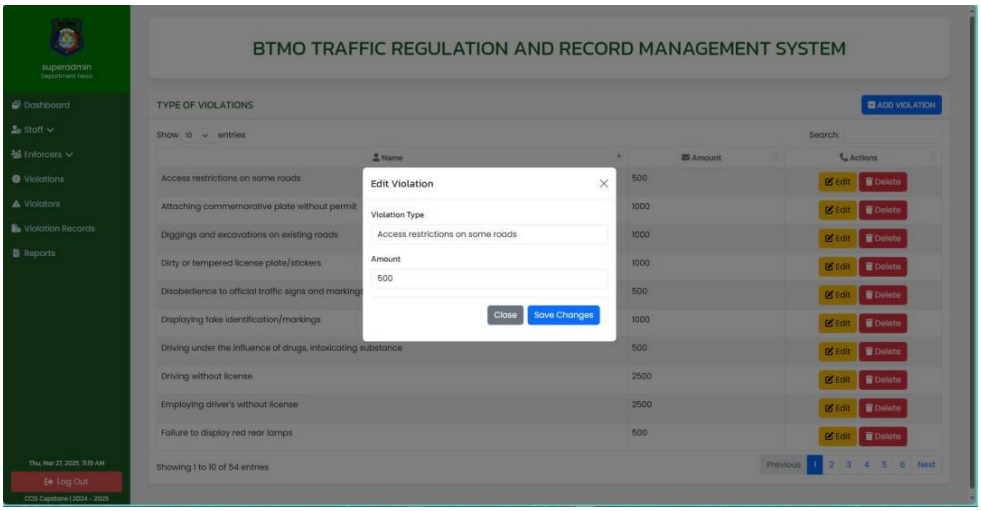


Figure 30. BTMO Head/Staff Side – Edit Violation

Figure 30 displays the Edit Violation interface, allowing modifications to violation details. By tapping the Edit icon, users can update the violation type and its amount. The interface includes Close and Save Changes buttons for canceling or confirming the updates.

Figure 31. BTMO Head/Staff Side – Violations Page

Figure 31 displays the Violators page, which lists violator records. This page includes details such as name, email, phone number, and address, along with a search function for quick record retrieval and management.

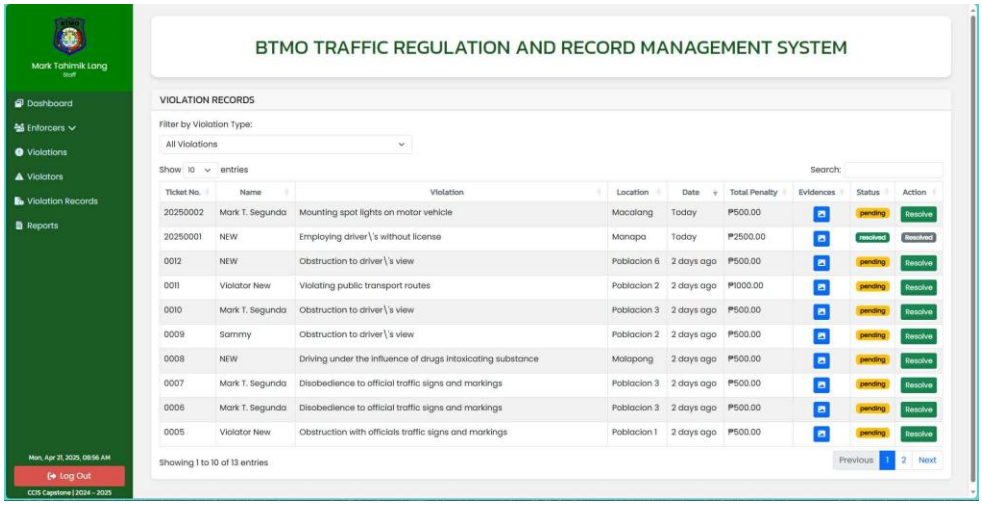


Figure 32. BTMO Head/Staff Side – Violation Records Page

Figure 32 displays the Violation Records page, which contains all recorded violations. This page shows details such as violator's name, type of violation, number of offenses, location, date, penalty amount, enforcer's name, evidence, status, and resolution options. A search feature is available for quick and efficient record lookup.

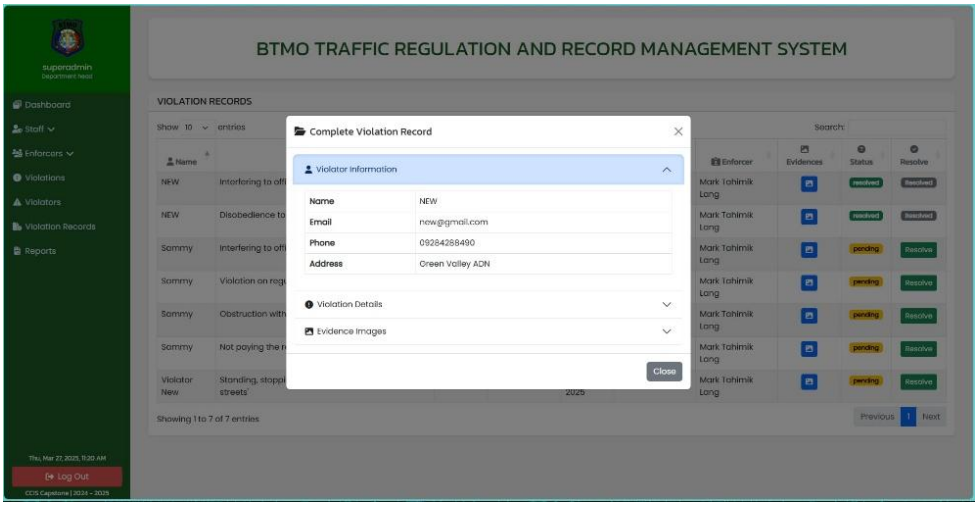


Figure 33. BTMO Head/Staff Side – Violation Information

Figure 33 displays the Violation Records page, where tapping any record reveals the complete violation details. Additionally, selecting the Violator Information section displays the violator's name, email, phone number, and address for a more detailed view.

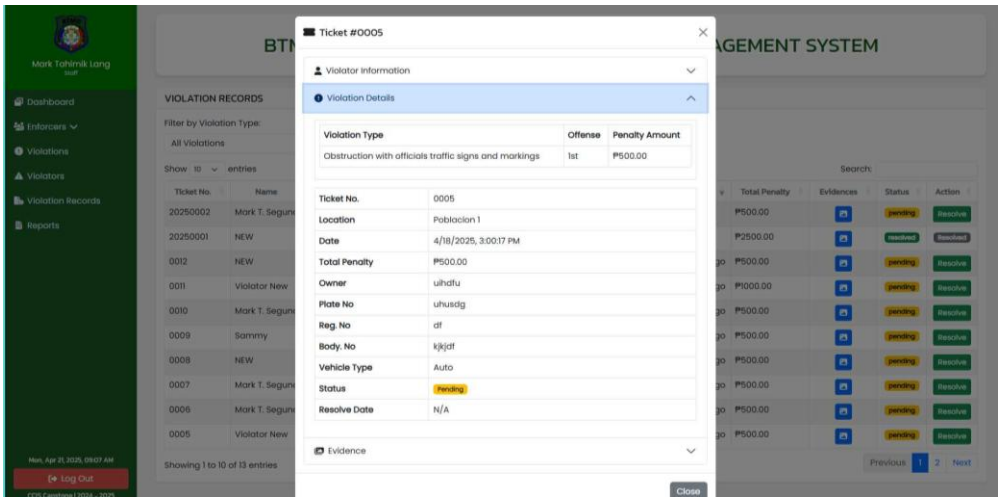


Figure 34. BTMO Head/Staff Side – Violation Details

Figure 34 presents the Violation Records page, allowing users to access complete violation details by selecting a record. Below the Violator Information section, the Violation Details are shown, covering violation type, location, penalty amount, vehicle owner, registration number, plate number, body number, vehicle type, ticket number, status, and resolved date. This structured layout ensures that enforcers can efficiently review and manage violation records for accurate enforcement and reporting.

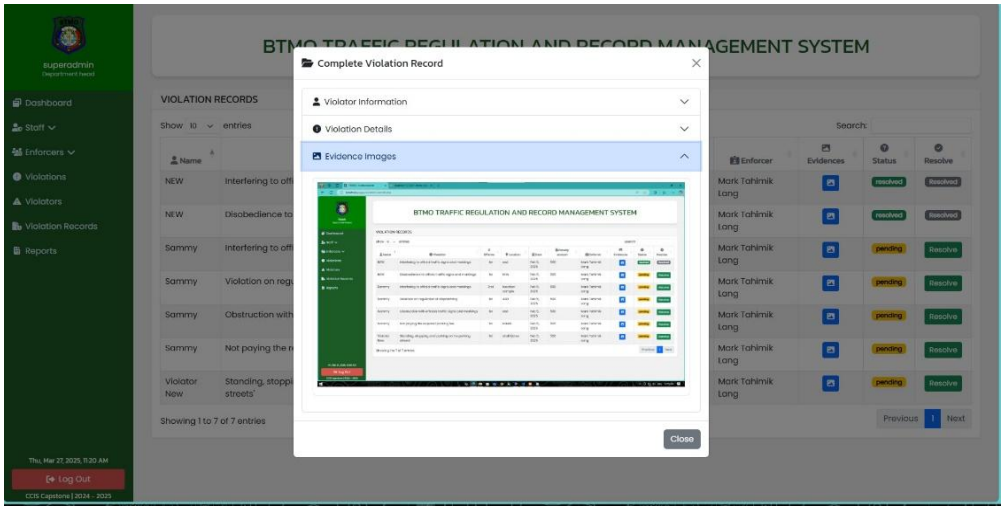


Figure 35. BTMO Head/Staff Side – Evidence Images

Figure 35 displays the Evidence Images section under the Violation Records page. After viewing the Violation Details, BTMO Head and Staff can tap on the evidence images to view the

attached proof provided by the enforcer. This feature enhances record accuracy and ensures proper documentation of each violation.

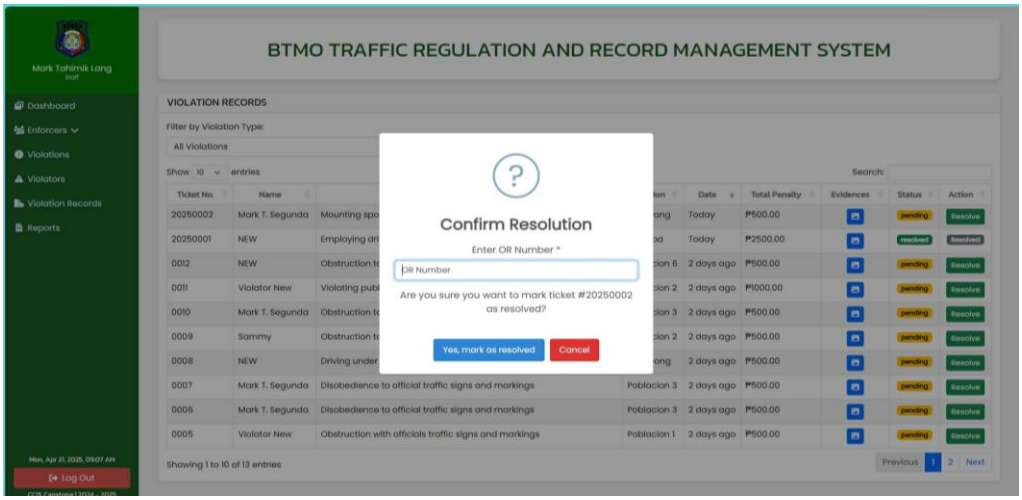


Figure 36. BTMO Head/Staff Side – Status Update with OR Confirmation

Figure 36 displays the Status Update feature under the Violation Records page. Before a case is officially marked as resolved, the system prompts the user to input the Official Receipt (OR) number as confirmation that the violator has paid the fine at the Office of the Treasurer. Once the OR number is entered, users can choose from options such as "Yes, mark as resolved" or "Cancel," allowing for accurate and verified status updates.

Figure 37. BTMO Head/Staff Side – Reports Page

Figure 37 displays the Reports Page, which presents violation records along with activity logs. Users

can filter reports by selecting weekly, monthly, or all violations. Additionally, an Export PDF button is available for printing reports, ensuring efficient documentation and record-keeping.

Figure 38. BTMO Head/Staff Side – Export PDF Report

Figure 38 displays the Print Report feature under the Reports page. After tapping Export PDF, the system automatically generates a printable report summarizing violation records. The report includes the total citations, period covered, and a tabulated list with details such as violator's name, violation type, location, date, penalty amount, enforcer, and status. This feature ensures accurate documentation, allowing authorities to efficiently track, analyze, and manage violation trends for better enforcement and decision-making.

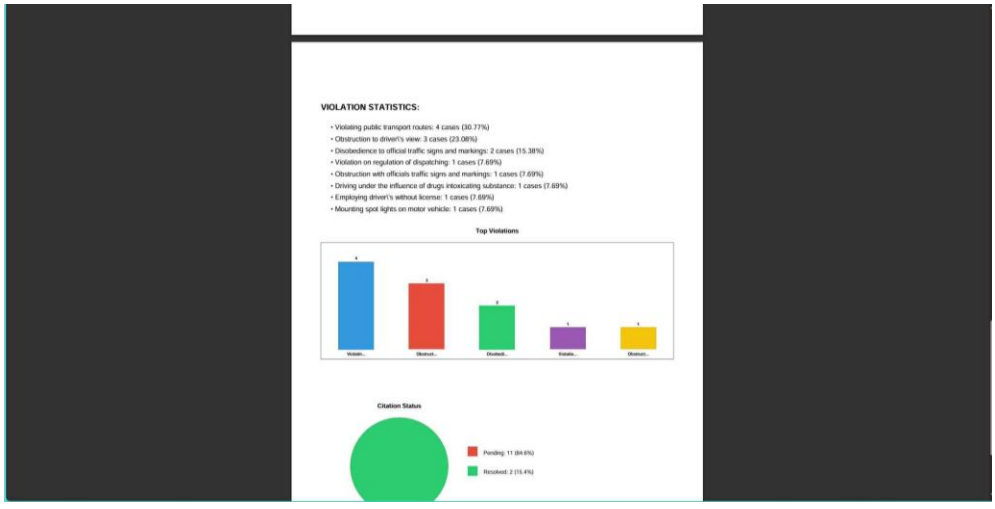


Figure 39. BTMO Head/Staff Side – Print Report with Violation Statistics Visualization

Figure 39 showcases the Print Report functionality within the Reports page. By selecting Export PDF, the system instantly generates a printable report summarizing violation records. An additional section of the report includes visual data representation through a bar chart, which displays the top types of violations along with the number of cases for each, making it easier to identify the most common infractions. A pie chart is also included to illustrate the citation status, categorizing violations into Pending and Resolved cases for a quick overview of enforcement outcomes. Additionally, the printed report contains statistical summaries, listing each type of violation along with its corresponding percentage, providing a detailed breakdown for more accurate monitoring and analysis. This feature enhances data-driven decision-making and helps authorities implement more targeted and effective enforcement strategies.

3.8 Software Platforms, Development Environment and Tools

Table 1

Software Requirements

Components	Specification	Usage
Internet Browser	Any	This means flexibility with platforms. The application needs to be able to be applied to multiple environments-web or mobile-and more so, for its greater reach. Compatibility across the different devices and platforms, as such, is quite crucial for the end users such as traffic officers and administrators.
Mobile Platform	Android	Android would be the main access system through the mobile where BTMO personnel can access this through portable devices. The traffic enforcers or administrators can log traffic violations through an application on their Android mobile and access data in real-time; they may even update some records directly from the field, thereby improving accessibility and near-real-time entry of information.
Mobile Application	Flutter	Flutter will be used to develop the mobile application of the system. It enables cross-platform development. The mobile app provide user-friendly interface for BTMO Traffic Enforcers, allowing them to record violations, scan vehicle details, capture photos, and send real-time updates directly from the field.
Front-end	HTML	The HTML is the basic language used in structuring the web interface of the application. It can be the backbone of the front end of the system that would allow BTMO staff to navigate pages, view records on traffic violations, and manage traffic data on a web-based platform.
	CSS	This will style the HTML using CSS, which will make the UI look attractive and user-friendly. The traffic management system demands an understandable and accessible output, and thus, it may help in usability with CSS, giving structure and a meaningful layout for the staff to work efficiently.
	JavaScript	Javascript can therefore be used to allow functions related to interactivity that otherwise cannot be achieved without JavaScript, such as dynamic validation of forms, search functionality or any other information which needs real time updating. In this case, therefore, JavaScript can include more functions such as sorting the record of traffic or filter through date and place and

		update real-time a dashboard in providing information on current traffic violations.
	JQuery	jQuery simplifies HTML document traversal and manipulation, event handling, and animation. It will enhance user interactions within the application, allowing for smoother transitions and dynamic content updates without extensive coding.
	SweetAlert	SweetAlert will be used to create visually appealing alert messages that notify users of important actions or errors within the application. This enhances user experience by providing clear and attractive feedback.
	Data AOS	Data AOS will be utilized to improve the user interface by adding animations to elements as they enter the viewport. This will make the application more engaging and visually appealing for users navigating through traffic data.
	Data Tables	DataTables will enable sorting, searching, and pagination of traffic violation records within the application. This functionality is essential for traffic officers and administrators to efficiently manage and analyze large datasets.
	Bootstrap	Bootstrap JS will be employed to create a responsive design that ensures the application is accessible on various devices. It provides pre-built components that streamline development while maintaining a consistent look and feel.
Back-end	MySQL (Database)	In the central database management system, MySQL would store all records of traffic violation reports, officer details, registered vehicles, and information of the traffic regulation. The high volume of data could thus be stored security with efficiency besides support queries and retrieval of information based on the needs for the BTMO staff.
	PHP	The server-side scripting language may be PHP, which would later be used for processing requests, handling data submissions, and interacting with the MySQL database to process traffic violation entries, update the traffic status, and enable secure login processes for BTMO administrators.

Server	Apache	XAMPP can offer a local server environment for development and testing, bundling PHP, MySQL, and Apache server capabilities. This enables developers to simulate a server on local machines to test the system's functions and ensure compatibility before deployment to a live environment.

3.9 Hardware Requirements

Table 2

Hardware Requirements

Components	Specification	Usage
Device	RAM	There should be sufficient RAM to multi-task efficiently, with minimal delay in application performance in the event of massive amounts of traffic data or real-time field processing. The minimum is 4 GB RAM, but 8 GB or more is preferred for complex database management, real-time data entry, and easy access to historical records without lag if the system has multiple users logging in at the same time.
	Processor	This should be mighty to complete various operations without a slowdown. For a desktop application, a quad-core processor that would be about Intel i5 or more suited would be perfect, whereas for mobiles, speed is crucial, and the best would be something as fast as Snapdragon 600 series or above, hence suited for efficient operation and processing in the application being used on the mobile device.
	Storage	Storage is needed for application files, database records, and perhaps a cache of traffic data for access in offline mode. The system should have at least 128 GB storage to keep records; however, more room can be provided with 256 GB or more, allowing for the long-term retention of records such as images for evidence of traffic violations and also backups. If the system is expected to increase in volume over time, then the server or central database location must be designed with expandable storage solutions.
Printer	Mobile Thermal Receipt Printer	The portable thermal receipt printer is ideal for field-based printing applications. With this device, a traffic officer can print violation tickets on the spot for motorists, including a description of the infraction. It can

		also produce receipts or transaction confirmations when fines are paid or notices are issued. This printer is compatible with Android devices and connects via Bluetooth, enabling enforcers to print directly from their mobile devices through the BTMO system application without the need for wires. This ensures quick, efficient, and durable receipts that can withstand outdoor conditions and the high frequency of use in traffic management.

3.10 Ethical Standard

ETHICAL STANDARD

In the research entitled "Buenavista Traffic Management Office Regulation and Record Management System," several ethical considerations were addressed to ensure that the study met the required standards for research integrity and protection of participants.

A. Protection of Intellectual Property Rights (IPR)

The Web-Based Traffic Regulation and Record Management System (TRRMS) incorporated innovative processes to streamline traffic management and regulation. All materials produced during the project, including system documentation, source code, and technical designs, were safeguarded through copyright protection to ensure proper attribution and prevent unauthorized use. The logo of the Buenavista Traffic Management Office (BTMO) associated with the system was protected under applicable trademark and copyright laws to prevent misuse. Permissions were secured to ensure compliance with intellectual property standards.

B. Informed Consent

The system's users, including the BTMO Head, staff, and traffic officers, were thoroughly briefed on its intended purpose and features. Prior to implementation and testing phases, written consent was obtained from stakeholders to confirm their agreement to participate in the system and provide feedback. This consent process involved clear communication regarding the system's objectives, scope, and possible implications, ensuring all participants had a complete understanding before engaging with the project. Furthermore, stakeholders were informed of their freedom to withdraw from participation at any point without penalties or repercussions, ensuring voluntary involvement throughout the project.

C. Data Privacy and Confidentiality

The system ensured data protection through strong security measures, such as encryption, secure login protocols, and Role-Based Access Control (RBAC). Only authorized staff had access to sensitive information, including personal details of drivers, vehicles, and traffic violation records. To further protect the identities of individuals, sensitive information was anonymized, ensuring that personal identifiers were removed or masked in data storage and reporting. Measures were implemented to ensure compliance with data protection laws, such as the Data Privacy Act (RA 10173) of the Philippines, safeguarding user privacy and maintaining confidentiality.

D. Voluntary Participation and Freedom to Withdraw

All participants involved in interviews, surveys, or testing participated voluntarily. They were fully informed of their right to decline or withdraw at any stage without adverse consequences. Clear communication about the study's objectives and participants' rights was provided to ensure respect and transparency. Ethical standards related to the use of animals were not applicable, as the study focused solely on human participants and their engagement with the system.

E. Minimization of Harm and Risk Management

The system was developed and tested to reduce potential risks to users, such as operational errors or data loss. The researchers implemented strong security measures and provided comprehensive training to users to prevent misuse or harm. A user-friendly manual was distributed, ensuring that participants could easily understand and navigate the system's features. Additionally, support services were made available to assist users in case of stress or challenges, offering prompt guidance and solutions. Regular consultations with stakeholders were conducted to address concerns and ensure a smooth experience during system development and deployment.

F. Beneficence and Contribution to Knowledge

The development of the TRRMS positively impacted the local community of Buenavista by improving traffic management efficiency, enhancing transparency, and reducing errors in traffic violations and vehicle registration. The system streamlined processes for both traffic officers and citizens, making the experience more convenient, secure, and efficient. The findings of the study were made available to participants and stakeholders upon request and were presented through community forums, detailed reports, and digital platforms to ensure transparency and understanding. This initiative contributed to safer roads, better enforcement of traffic laws, and increased public satisfaction.

G. Justice and Fair Participant Selection

Participants were selected based on fair and transparent inclusion/exclusion criteria, ensuring no discrimination based on background, status, or profession. The selection process focused on stakeholders relevant to the study, including drivers, traffic officers, and local government officials, providing equal opportunities for participation and input. The study did not involve animals, focusing exclusively on human participants and their roles in traffic regulation and management. This approach ensured fairness, inclusivity, and relevance to the system's objectives.

H. Data Integrity and Accuracy

The system was designed with rigorous standards to ensure data accuracy and integrity. Best practices in data validation and automated checks were implemented to minimize errors in traffic-related records. Potential biases in data collection or processing were documented, and mitigation strategies were employed to ensure fairness and reliability. The research acknowledged and addressed possible errors and limitations, with regular testing and updates conducted to maintain functionality.

I. Transparency and Honesty in Reporting

All findings, including system performance metrics and user feedback, were reported truthfully and transparently. Potential conflicts of interest were disclosed to ensure integrity in the research process. The use of AI tools like ChatGPT and Grammarly was acknowledged, as they assisted in documentation and content enhancement. All third-party tools and software used in the project were properly credited. The development process, including challenges and solutions, was thoroughly documented to uphold transparency and credibility.

J. Use of Patented or Copyrighted Materials

The TRRMS was an original project conceptualized to meet the specific needs of the Buenavista Traffic Management Office. While third-party software, tools, or libraries were used to enhance functionality, they were carefully selected and used in compliance with copyright laws. Proper licenses and permissions were obtained, and all copyrighted or patented materials were cited and acknowledged to uphold intellectual property standards and ethical practices.

K. Ethical Considerations for Animal and Human Trials

The project did not involve animal or human trials. It focused on developing and implementing a digital system for traffic regulation and record management. While there were no direct threats to participants, the researchers ensured the protection of data and user privacy. Strong security protocols were implemented. Participants benefited from improved traffic management and efficient services, while also contributing feedback to enhance the system.

L. Responsible Use of AI and Other Related Technologies

To maintain documentation quality, AI tools like Grammarly and ChatGPT were used to enhance writing, generate ideas, and ensure clarity. Grammarly assisted with proofreading and grammar, while ChatGPT helped with writing and improving sections of the study. All AI-generated

content was reviewed by the researchers to maintain ethical standards and accuracy. These tools contributed to the professionalism and clarity of the documentation.

M. Ethical Clearance and Institutional Approval

Ethical approval was obtained from the appropriate Institutional Review Board (IRB) or ethics committee of Saint Michael College of Caraga. This ensured that the project adhered to ethical guidelines, respected participants' rights, maintained data privacy, and protected intellectual property throughout development and implementation. The researchers followed all institutional protocols to demonstrate ethical responsibility.

CHAPTER IV

SOFTWARE DEVELOPMENT AND TESTING

This chapter describes the creation and testing phases of the Buenavista Traffic Management Office Regulation and Record Management System, including the approaches, frameworks, and technologies used for its development and optimization. It also explains the measures taken to ensure the system's functionality, usability, and efficiency in handling traffic-related records, along with the testing methodologies used to assess its performance, security, and dependability to meet the requirements of the Buenavista Traffic Management Office (BTMO).

4.1 Development Process

The systematic software development process of the Web-Based Traffic Regulation and Record Management System for the Buenavista Traffic Management Office (BTMO) is illustrated through the input-process-output diagram shown in Figure 40.

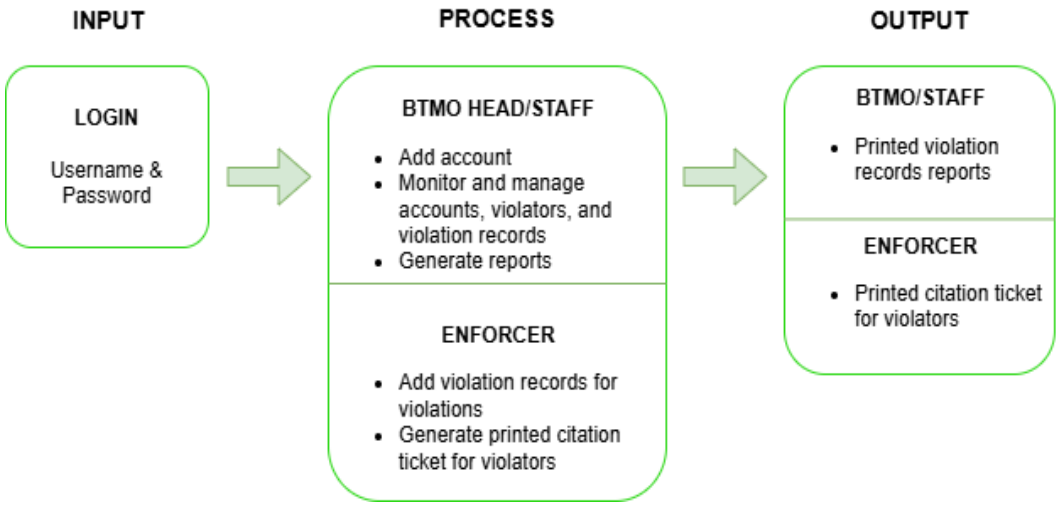


Figure 40. Input-Process-Output Diagram

Figure 40 shows the input-process-output diagram and illustrates the orderly software development process of the Traffic Regulation and Record Management System (TRRMS).

LOG IN

The BTMO Head and Staff have access to the system's essential functions. The secure login mechanism allows them to authenticate using a unique username and password. Developed with industry-standard encryption and validation protocols, this system ensures credential protection, enforces strict access controls, and prevents unauthorized access. After creating an account for an enforcer, they will log in using their credentials (Username and Password), and the system will automatically redirect them to the dashboard of the app-based TRRMS. They will remain logged in unless they manually log out, clear app data, or uninstall the application.

BTMO HEAD & STAFF

The process phase serves as the system's backbone, enabling key functions for both the BTMO Head and Staff. The BTMO Head holds top administrative control, allowing them to add staff and enforcers, ensuring proper delegation. Staff members can also add enforcers, promoting a structured enforcement system. Both roles have full access to monitor violator records, track violations, and manage accounts, ensuring accountability. Additionally, the system generates detailed reports, providing insights into traffic violations and enforcement efficiency. This functionality streamlines workflows, enhances regulation, and improves traffic management.

ENFORCER

The enforcers utilizing the app-based TRRMS have the capability to record violations in real time, ensuring accurate and up-to-date documentation of offenses. They can input detailed violation records, including the type of offense, location, date, and violator information, streamlining the enforcement process. Additionally, the system allows enforcers to generate and print official citation tickets on-site, providing violators with immediate documentation of their infractions. This feature enhances efficiency, minimizes manual paperwork, and ensures proper enforcement and record-keeping within the traffic management system.

PRINTED VIOLATION RECORDS REPORTS

The system provides weekly and monthly printed violation records reports along with statistical summaries to ensure comprehensive tracking and analysis of traffic violations. BTMO Head and staff can generate, review, and validate these reports, maintaining accurate record-keeping and compliance with reporting standards. Additionally, statistical reports offer insights into trends, enforcement effectiveness, and violation frequencies.

PRINTED CITATION TICKET FOR VIOLATORS

Enforcers provide printed citation tickets to violators, ensuring immediate documentation of offenses and compliance with enforcement protocols. This feature streamlines the citation process, reducing manual paperwork and enhancing record accuracy. By issuing printed tickets on-site, enforcers can efficiently inform violators of their infractions while maintaining organized and reliable violation records.

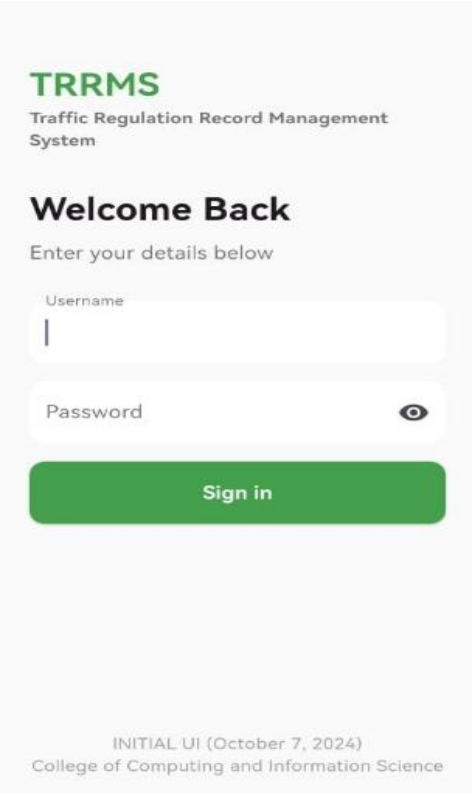


Figure 41. Enforcer Side¬–Login Page

The Enforcer Login Page serves as the access point for enforcers to enter the app-based Traffic Regulation and Record Management System (TRRMS). It features input fields for username and password, ensuring secure authentication. Enforcers must enter their designated credentials and click the "Sign In" button to gain access to the system, allowing them to perform enforcement duties efficiently.

```
Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      const SizedBox(height: 32),
                      const Text('TRRMS',
                          style: TextStyle(
                              fontSize: 28,
                              fontWeight: FontWeight.bold,
                              color: Colors.green)),
                      const Text(
                          'Traffic Regulation Record Management System',
                          style: TextStyle(
                              fontWeight: FontWeight.bold,
                              color: Colors.black54)),
                      const SizedBox(height: 32),
                      const Text('Welcome Back',
                          style: TextStyle(
                              fontSize: 28, fontWeight: FontWeight.bold)),
                      const SizedBox(height: 8),
                      const Text('Enter your details below',
                          style:
                              TextStyle(color: Colors.black54, fontSize: 16)),
                      const SizedBox(height: 32),
                      TextField(
                          controller: _usernameController,
                          decoration: InputDecoration(
                              labelText: 'Username',
                              labelStyle:
                                  const TextStyle(color: Colors.black54),
                              border: OutlineInputBorder(
                                  borderRadius: BorderRadius.circular(12),
                                  borderSide: BorderSide.none),
                              filled: true,
                              fillColor: Colors.white)),
                      const SizedBox(height: 16),
                      TextField(
                          controller: _passwordController,
                          obscureText: !_isPasswordVisible,
                          decoration: InputDecoration(
                              labelText: 'Password',
                              labelStyle:
                                  const TextStyle(color: Colors.black54),
                              border: OutlineInputBorder(
                                  borderRadius: BorderRadius.circular(12),
                                  borderSide: BorderSide.none),
                              filled: true,
                              fillColor: Colors.white,
                              suffixIcon: IconButton(
                                  icon: Icon(_isPasswordVisible
                                      ? Icons.visibility_off
                                      : Icons.visibility),
                                  onPressed: () {
                                    setState(() {
                                      _isPasswordVisible =
                                          !_isPasswordVisible;
                                    });
                                  }))),
                      const SizedBox(height: 16),
```

Figure 42 code snippet for Login Screen UI Layout - login.dart

Figure 42 shows the login interface of the App-Based Traffic Regulation Record Management System (TRRMS), developed using Flutter. The layout is structured with a Column widget, displaying system details, a welcome message, and user input fields. The TextFields for

username and password feature a clean design with rounded borders and a filled background. A crucial feature is the password visibility toggle, controlled by `_isPasswordVisible`, which updates via `setState()`. The sign-in button, styled in green, triggers `_checkPermissions()` for authentication.

```
Future<void> _login() async {
   const String apiUrl = "$baseUrl/login.php";

   final response = await http.post(Uri.parse(apiUrl),
       headers: {"Content-Type": "application/json"},
       body: json.encode({
         "username": _usernameController.text.trim(),
         "password": _passwordController.text
       }));

   final data = json.decode(response.body);
```

Figure 43 code snippet for Login API Request

Figure 43 shows the login function of the App-Based TRRMS, which sends an HTTP POST request to `login.php` for authentication. The request includes a JSON-encoded body containing the trimmed username and password, with `Content-Type` set to `application/json`. The server response is then decoded for further processing. This function ensures secure credential transmission and serves as the core authentication mechanism for the system.

```
if (data['status'] == "success") {
      final prefs = await SharedPreferences.getInstance();

      await prefs.setBool('isLoggedIn', true);
      await prefs.setString('enforcer_id', data['enforcer_id'].toString());
      await prefs.setString('username', data['username']);
      await prefs.setString('name', data['name']);
      await prefs.setString('badge_number', data['badge_number']);
      await prefs.setString('email', data['email']);
      await prefs.setString('phone', data['phone']);
      await prefs.setString('department', data['department']);

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(" welcome back! ${data['name']}.")),
      );

      Navigator.of(context).pushAndRemoveUntil(
        MaterialPageRoute(builder: (BuildContext context) => const Dashboard()),
        (route) => false,
      );
    } else {
      ScaffoldMessenger.of(context)
          .showSnackBar(SnackBar(content: Text(data['message'])));
```

Figure 44 code snippet for Handle Login Success and Navigation

Figure 44 shows the login success handling in App-Based TRRMS, where SharedPreferences stores user details such as enforcer_id, username, and badge_number for session persistence. If authentication is successful, the app displays a welcome message via Snackbar and navigates to the Dashboard, replacing all previous routes. If login fails, an error message is shown using Snackbar, ensuring clear feedback to the user.

```
$stmt = $conn->prepare("SELECT * FROM enforcers WHERE username = ? AND status = 'active'");
$stmt->bind_param("s", $user);
$stmt->execute();
$result = $stmt->get_result();
```

Figure 45 code snippet for Fetching Active Enforcer by Username – login.php

Figure 45 shows the login verification process in App-Based TRRMS (Backend). The system queries the `enforcers` table to check if a user with the given username exists and has an 'active' status. It uses prepared statements `($stmt->bind_param)` to prevent SQL injection and securely retrieve user credentials. If a match is found, authentication continues; otherwise, login fails.

```
if ($result->num_rows > 0) {
    $row = $result->fetch_assoc();
    if (password_verify($pass, $row['password'])) {
echo json_encode([
            "status" => "success",
            "message" => "Login successful.",
            "enforcer_id" => $enforcer_id,
            "name" => $name,
            "email" => $email,
            "badge_number" => $badge_number,
            "phone" => $phone,
            "department" => $department,
            "username" => $user
        ]);
    } else {
        echo json_encode(["status" => "error", "message" => "Invalid password."]);
    }
} else {
    echo json_encode(["status" => "error", "message" => "User not found or inactive."]);
}
```

Figure 46 code snippet for Enforcer Login Authentication and Response – login.php

Figure 46 shows the password authentication process in App-Based TRRMS (Backend). If a user with the given username is found, the system verifies the password using `password_verify()` to compare the entered password with the hashed password stored in

the database. If it matches, a JSON response with user details is returned, allowing access to the system. Otherwise, it sends an error message indicating an invalid password or inactive account.

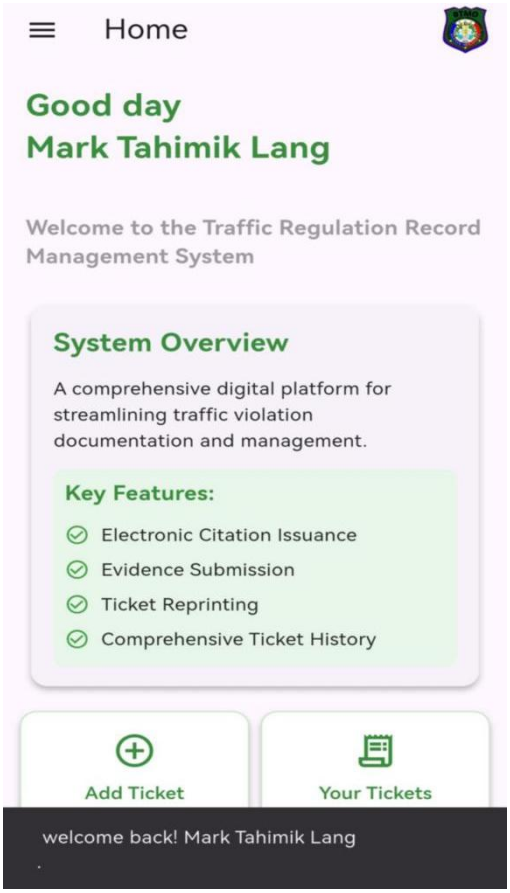


Figure 47. Enforcer Side¬–Homepage

The Enforcer Homepage in the Traffic Regulation and Record Management System (TRRMS) serves as the central hub for enforcers. It provides an overview of the system, highlighting key features such as real-time violation tracking and report management. The page includes an "Add Ticket" button for issuing new violation tickets and a "Your Tickets" button to manage and review previously recorded violations.

```
Future<void> _fetchRecentTickets() async {
    try {
      final response = await http.get(
        Uri.parse('$baseUrl/get.php?table=violationrecords'),
      );
      if (response.statusCode == 200) {
        try {
          final List<dynamic> tickets = json.decode(response.body);

          tickets.sort((a, b) {
            return DateTime.parse(b['violation_date'])
                .compareTo(DateTime.parse(a['violation_date']));
          });
          setState(() {
            _recentTickets = tickets.take(10).toList();
            _isLoading = false;
          });
        } catch (parseError) {
          setState(() {
            _isLoading = false;
            _recentTickets = [];
          });
        }
      } else {
        setState(() {
```

```
          _isLoading = false;
          _recentTickets = [];
        });
      }
    } catch (e) {
      setState(() {
        _isLoading = false;
        _recentTickets = [];
      });
    }
  }
```

Figure 48 code snippet for Fetch and Display Recent Violation Tickets

Figure 48 shows the implementation of the `_fetchRecentTickets()` function, which retrieves violation records from the backend and displays them in a list. It sends an HTTP GET request to `get.php` with the parameter `table=violationrecords`, checks if the response is successful (status code 200), and decodes the JSON data into a list of tickets. The tickets are then sorted by `violation_date` in descending order, ensuring the most recent violations appear first. The function updates the UI by storing the 10 most recent records in `_recentTickets`. Proper error handling is implemented to manage failed requests or parsing errors, ensuring smooth system operation for enforcers.

```
Row(
                  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                  children: [
                    SizedBox(
                        width: 160,
                        height: 90,
                        child: ElevatedButton(
                            style: ElevatedButton.styleFrom(
                                backgroundColor: Colors.white,
                                foregroundColor: Colors.black,
                                elevation: 6,
                                shadowColor: Colors.black26,
                                shape: RoundedRectangleBorder(
                                    borderRadius: BorderRadius.circular(15),
                                    side: BorderSide(
                                        color: Colors.grey.shade200,
                                        width: 1.5))),
                            onPressed: () {
                              Navigator.of(context).push(MaterialPageRoute(
                                  builder: (BuildContext context) {
                                return const ConnectDevice();
                              }));
                            },
                            child: const Column(
                                mainAxisAlignment: MainAxisAlignment.center,
                                children: [
                                  Icon(Icons.add_circle_outline_outlined,
                                      color: Colors.green, size: 40),
                                  SizedBox(height: 8),
                                  Text('Add Ticket',
                                      style: TextStyle(
```

```
                                    color: Colors.green,
                                    fontWeight: FontWeight.w600,
                                    fontSize: 14))
                            ])),
                    SizedBox(
                        width: 160,
                        height: 90,
                        child: ElevatedButton(
                            style: ElevatedButton.styleFrom(
                                backgroundColor: Colors.white,
                                foregroundColor: Colors.black,
                                elevation: 6,
                                shadowColor: Colors.black26,
                                shape: RoundedRectangleBorder(
                                    borderRadius: BorderRadius.circular(15),
                                    side: BorderSide(
                                        color: Colors.grey.shade200,
                                        width: 1.5))),
                            onPressed: () {
                              Navigator.of(context).push(MaterialPageRoute(
                                  builder: (BuildContext context) {
                                return const TicketList();
                              }));
                            },
                            child: Column(
                                mainAxisAlignment: MainAxisAlignment.center,
                                children: [
                                  Icon(Icons.receipt_long_outlined,
                                      color: Colors.green.shade700, size: 40),
                                  const SizedBox(height: 8),
                                  Text('Your Tickets',
                                      style: TextStyle(
                                          color: Colors.green.shade700,
                                          fontWeight: FontWeight.w600,
                                          fontSize: 14))
                                ]))
                    ]),
```

Figure 49 code snippet for Ticket Management Action Buttons

Figure 49 shows the implementation of two primary buttons on the enforcer's homepage in the TRRMS app: "Add Ticket" and "Your Tickets." The "Add Ticket" button navigates to the Bluetooth connection page `(ConnectDevice)` when pressed, allowing enforcers to connect to a Bluetooth-enabled printer or device before issuing a violation ticket. The "Your Tickets" button redirects users to the `TicketList` page, where they can view a list of previously issued violation tickets. Both buttons feature a white background with green icons and text, ensuring clear visibility, and utilize an `ElevatedButton` with rounded borders for a modern and user-friendly interface.

```
if ($table === 'violationrecords') {
            $sql = "SELECT vr.*, v.name AS violator_name
                    FROM $table vr
                    LEFT JOIN violators v ON vr.violator_id = v.violator_id
                    ORDER BY vr.violation_date";
            $result = $conn->query($sql);

            $records = [];
```

```php
        if ($result->num_rows > 0) {
            while ($row = $result->fetch_assoc()) {
                $row['name'] = $row['violator_name'];
                $records[] = $row;
            }
        }

        $groupedData = [];
        foreach ($records as $record) {
            $ticketNo = $record['ticket_no'];
            if (!isset($groupedData[$ticketNo])) {
                $groupedData[$ticketNo] = [
                    'ticket_no' => $ticketNo,
                    'name' => $record['name'],
                    'location' => $record['location'],
                    'violation_date' => $record['violation_date'],
                    'status' => $record['status'],
                    'vehicle_owner' => $record['vehicle_owner'],
                    'registration_no' => $record['registration_no'],
                    'plate_no' => $record['plate_no'],
                    'vehicle_type' => $record['vehicle_type'],
                    'body_no' => $record['body_no'],
                    'enforcer_id' => $record['enforcer_id'],
                    'violator_id' => $record['violator_id'],
                    'violations' => []
                ];
            }
            $groupedData[$ticketNo]['violations'][] = [
                'violation_type' => $record['violation_type'],
                'penalty_amount' => $record['penalty_amount']
            ];
        }

        foreach ($groupedData as &$ticket) {
            $violatorId = $ticket['violator_id'];

            foreach ($ticket['violations'] as &$violation) {
                $violationType = $violation['violation_type'];
                $currentDate = $ticket['violation_date'];

                $offenseCount = 0;
                foreach ($records as $record) {
                    if ($record['violator_id'] == $violatorId &&
                        $record['violation_type'] == $violationType &&
                        $record['violation_date'] <= $currentDate) {
                        $offenseCount++;
                    }
                }

                $violation['offense_count'] = $offenseCount;
            }

            $totalAmount = 0;
            foreach ($ticket['violations'] as $violation) {
                $totalAmount += (float)$violation['penalty_amount'];
            }
            $ticket['total_penalty_amount'] = $totalAmount;
        }

        $data = array_values($groupedData);
    }
```

Figure 50 code snippet for Violation Records Grouping and Processing – get.php

Figure 50 illustrates the backend logic for retrieving and structuring violation records in the TRRMS system. The `get.php` script executes an SQL query to fetch violation records, joining

them with the `violators` table to include violators' names. It groups records by `ticket_no` to organize multiple violations under the same ticket and calculates the total penalty amount by summing `penalty_amount` values. Additionally, it determines the offense count for each violation type by checking how many times the violator has committed the same offense before the current violation date. The final structured data is then formatted into a JSON response for efficient display in the app.

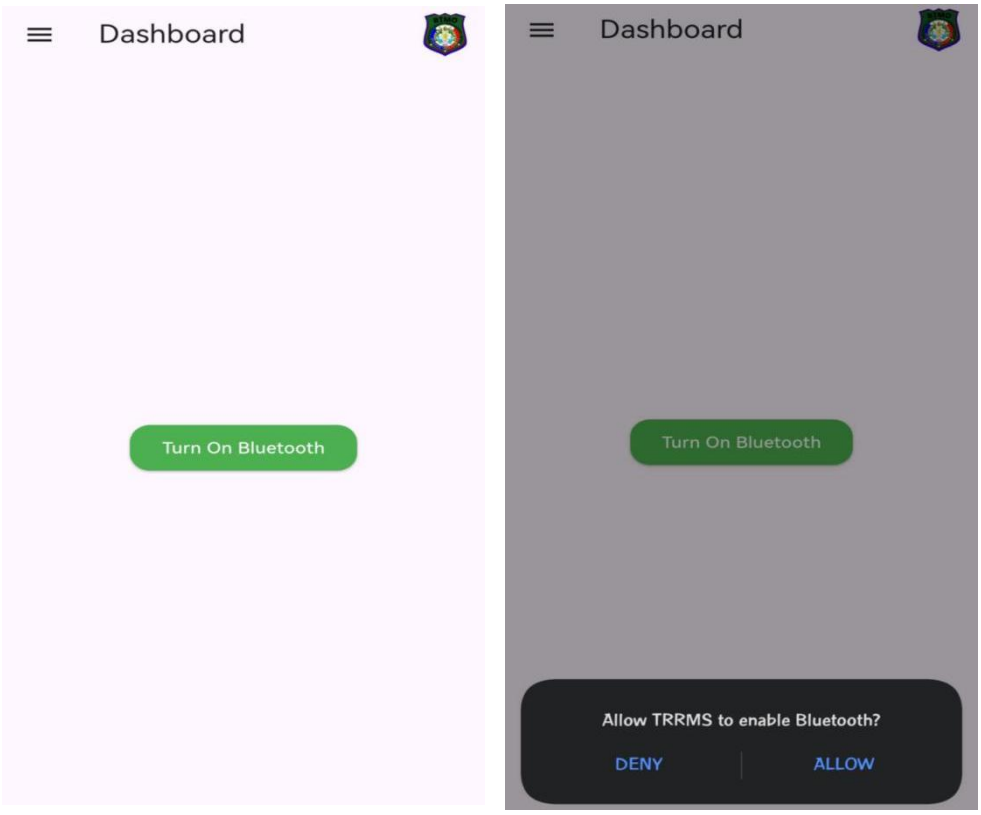


Figure 51. Enforcer Side – Connect to Bluetooth

The Connect to Bluetooth page in the Traffic Regulation and Record Management System (TRRMS) allows enforcers to establish a Bluetooth connection with a thermal printer for ticket printing. Upon accessing this page, the system prompts the enforcer with a permission request: "Allow TRRMS to enable Bluetooth?" with options to Deny or Allow. Granting permission enables

seamless communication between the app and the printer, ensuring that violation tickets can be printed instantly for proper documentation and enforcement.

```
ElevatedButton(
                            style: ElevatedButton.styleFrom(
                                backgroundColor: Colors.green,
                                shape: const RoundedRectangleBorder(
                                    borderRadius:
                                        BorderRadius.all(Radius.circular(15)))),
                            onPressed: _turnOnBluetooth,
                            child: const Text('Turn On Bluetooth',
                                style: TextStyle(color: Colors.white)))
```

Figure 52 code snippet for Bluetooth Button UI

Figure 52 shows a button that allows the enforcer to turn on Bluetooth when it is currently disabled. The button is styled with a green background and rounded corners to maintain a user-friendly interface. When tapped, it triggers the `_turnOnBluetooth` function, which attempts to enable Bluetooth on the device. This feature is essential for establishing a connection with the thermal printer, ensuring that violation tickets can be printed seamlessly within the Traffic Regulation and Record Management System (TRRMS).

```
ElevatedButton(
                        style: ElevatedButton.styleFrom(
                            backgroundColor: Colors.green,
                            shape: const RoundedRectangleBorder(
                                borderRadius:
                                    BorderRadius.all(Radius.circular(15)))),
                        onPressed: requestBluetoothPermissions,
                        child: const Text('Request Permissions',
                            style: TextStyle(color: Colors.white)))
```

Figure 53 code snippet for Request Bluetooth Permissions Button

Figure 53 shows a button that prompts the enforcer to request Bluetooth permissions if they have not been granted yet. The button is styled with a green background and rounded corners for a visually appealing and intuitive interface. When tapped, it calls the `requestBluetoothPermissions` function, which triggers the system's permission request dialog.

```
 IconButton(
                 icon: const Icon(Icons.refresh), onPressed: _startScan),
             if (_connectedDevice != null)
               IconButton(
                   icon: const Icon(Icons.bluetooth_disabled),
                   onPressed: _disconnectFromDevice)
           ])
```

Figure 54 code snippet for Bluetooth Action Buttons with Conditional Rendering

Figure 54 shows two action buttons for managing Bluetooth connectivity within the Traffic Regulation and Record Management System (TRRMS). The refresh button, represented by a refresh icon, triggers the `_startScan` method to search for nearby Bluetooth devices, allowing enforcers to discover and connect to a thermal printer. The disconnect button, which only appears if a device is currently connected, is represented by a Bluetooth-disabled icon and calls the `_disconnectFromDevice` function to terminate the active connection. These buttons ensure seamless device management for efficient ticket printing.

```
SizedBox(
                        width: double.infinity,
                        child: ElevatedButton(
                            style: ElevatedButton.styleFrom(
                                backgroundColor: Colors.green[700],
                                padding:
                                    const EdgeInsets.symmetric(vertical: 15),
                                shape: RoundedRectangleBorder(
                                    borderRadius: BorderRadius.circular(10))),
                            onPressed: () {
                              _submitTicket();
                            },
                            child: const Text('Submit',
                                style: TextStyle(
                                    fontSize: 16, color: Colors.white))))
```

Figure 55 code snippet for Submit Button with Styling

Figure 55 shows the ticket submission button on the "Add Ticket" page of the Traffic Regulation and Record Management System (TRRMS). This button, styled with a green background and rounded corners for a user-friendly interface, spans the full width of the screen to ensure easy accessibility. When pressed, it triggers the `_submitTicket()` method, which validates the form inputs, submits the ticket data to the backend, and processes the server response.

```
ElevatedButton(
                        style: ElevatedButton.styleFrom(
                            backgroundColor: Colors.green[700],
                            shape: RoundedRectangleBorder(
                                borderRadius: BorderRadius.circular(10))),
                        onPressed: _pickImages,
                        child: const Icon(Icons.image, color: Colors.white)),
```

Figure 56 code snippet for Image Picker Button

Figure 56 shows the upload image button on the "Add Ticket" page of the Traffic Regulation and Record Management System (TRRMS). This button, styled with a green background and rounded corners for a modern look, allows enforcers to attach multiple images as evidence for a violation. When pressed, it triggers the `_pickImages()` method, which utilizes the ImagePicker package to open the device's gallery for image selection. This feature ensures that photographic proof is securely uploaded and associated with each violation record.

```
Future<void> fetchNames() async {
  final response = await http.get(
    Uri.parse('$baseUrl/get.php?table=violators'),
  );
  if (response.statusCode == 200) {
    final List<dynamic> data = json.decode(response.body);
    setState(() {
      names = data.map((item) => item['name'].toString()).toList();
      violatorsData = data;
      names.add('Add new Violator');
    });
  }
}
```

Figure 57 code snippet for Fetch and Set Violator Names

Figure 57 shows the `fetchNames()` method, which retrieves the list of violators from the backend by making an HTTP GET request to `get.php?table=violators`. If the request is successful (`statusCode == 200`), the response data is decoded from JSON format and mapped into a list of violator names. The method then updates the UI using `setState()`, ensuring that the list is dynamically refreshed. Additionally, an "Add new Violator" option is appended to the list, allowing users to manually input a new violator if they are not found in the existing records.

```
_buildDropdown(
                          label: 'Select Name',
                          value: selectedName,
                          items: names,
                          onChanged: (value) {
                            setState(() {
                              selectedName = value;

                              if (value == 'Add new Violator') {
                                _showNewNameDialog();
                                violatorAddress.text = '';
                              } else {
                                final selectedViolator =
                                    violatorsData.firstWhere(
                                        (item) =>
```

```
                                    item['name'].toString() == value,
                                orElse: () => null);

                        violatorAddress.text = selectedViolator != null
                            ? selectedViolator['address'].toString()
                            : '';
                    }
                });
            }),
```

Figure 58 code snippet for Dropdown Selection for Violator Name

Figure 58 shows a dropdown menu that displays the list of violators retrieved from the backend using the `fetchNames()` function. When a user selects a violator from the dropdown, their corresponding address is automatically populated in the address field. If the "Add new Violator" option is chosen, a dialog `(_showNewNameDialog())` appears, allowing the user to input a new violator's details. The logic also ensures that if no matching violator is found, the address field remains empty. This implementation enhances usability by streamlining data entry and reducing manual input errors.

```
Future<void> _submitTicket() async {
   if (selectedName == null ||
       selectedViolations.isEmpty ||
       selectedLocation == null) {
     ScaffoldMessenger.of(context).showSnackBar(
         const SnackBar(content: Text('Please fill in all required fields')));
     return;
   }
_printReceipt(
         violatorname: selectedName,
         violation: combinedViolations,
         location: selectedLocation,
         amount: totalPenaltyAmount,
         offenseCount: offenseCounts.values.first,
         violationOffenseCounts: offenseCounts,
         violationAmounts: violationAmounts,
         vehicleOwner: vehicleOwner.text,
         registrationNo: registrationNo.text,
         plateNo: plateNo.text,
         vehicleType: vehicleType.text,
         violatorAddress: violatorAddress.text,
         bodyNumber: bodyNumber.text,
         ticketNo: ticketNo);
```

Figure 59 code snippet for Ticket Submission and Receipt Printing

Figure 59 shows the `_submitTicket()` function, which is triggered when the submit button is pressed. This function first validates whether all required fields—violator's name, selected violations, and location—are filled. If any field is missing, a snackbar notification prompts the user to complete the form. Upon successful validation, the function calls

`_printReceipt()` to generate and print a receipt containing details such as the violator's name, violations, location, total penalty amount, offense counts, and vehicle information. This ensures that every issued ticket is properly documented and printed for reference.

```
Future<void> _pickImages() async {
  final ImagePicker _picker = ImagePicker();
  final List<XFile>? images = await _picker.pickMultiImage();

  if (images != null) {
    setState(() {
      _selectedImages = images;
    });
  }
}
```

Figure 60 code snippet for Image Picker for Multiple Image Selection

Figure 60 shows the `_pickImages()` method, which enables users to select multiple images using the ImagePicker package. When triggered, it opens the device's gallery and allows the selection of multiple images. The selected images are then stored in the `_selectedImages` list. If the user successfully picks images, the UI is updated using `setState()`, ensuring that the selected images are displayed within the application.

```
Widget _buildImagePreview() {
  if (_selectedImages == null || _selectedImages!.isEmpty) {
    return const Text('No images selected.',
        style: TextStyle(color: Colors.grey));
  }

  return Wrap(
      spacing: 8.0,
      runSpacing: 8.0,
      children: _selectedImages!.map((image) {
        return Image.file(File(image.path),
            width: 100, height: 100, fit: BoxFit.cover);
      }).toList());
}
```

Figure 61 code snippet for Image Preview Builder

Figure 61 shows the `_buildImagePreview()` widget, which displays a preview of the selected images. If no images have been selected, it returns a placeholder text indicating "No images selected." Otherwise, it arranges the selected images using a `Wrap` widget for a responsive layout. Each image is displayed with a fixed size of 100x100 pixels, ensuring a consistent preview. The images are loaded using the `Image.file` widget, maintaining their aspect ratio with `BoxFit.cover`.

```
else {
            $sql = "SELECT * FROM $table";
            $result = $conn->query($sql);

            $data = [];
            if ($result->num_rows > 0) {
                while ($row = $result->fetch_assoc()) {
                    $data[] = $row;
                }
            }
        }
```

Figure 62 code snippet for Fetch Data from Table – get.php

Figure 62 shows a segment of the `get.php` script responsible for fetching data from the specified database table and returning it as a JSON response. This script executes a `SELECT * FROM $table` query to retrieve all records from the requested table. The results are stored in an array, which is then encoded into JSON format. The Flutter app utilizes this data to dynamically populate dropdown menus and lists, ensuring an updated and responsive user interface.

```
    (ticket_no, violator_id, enforcer_id, violation_date, violation_type, location,
penalty_amount, vehicle_owner, registration_no, plate_no, vehicle_type, body_no, status)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, 'pending')");

$stmt->bind_param(
    "siisssdssss",
    $ticketNo,
    $data['violator_id'],
    $data['enforcer_id'],
    $violationDate,
    $data['violation_type'],
    $data['location'],
    $data['penalty_amount'],
    $data['vehicle_owner'],
    $data['registration_no'],
    $data['plate_no'],
    $data['vehicle_type'],
    $data['body_no']
);
```

Figure 63 code snippet for Insert Violation Record – add_ticket.php

Figure 63 shows the `add_ticket.php` script responsible for inserting a new violation ticket into the `violationrecords` table. The script utilizes prepared statements to prevent SQL injection, ensuring secure data handling. The ticket's status is set to "pending" by default upon creation. The `bind_param` method securely binds the provided ticket details, including the violator's and enforcer's IDs, violation type, location, penalty amount, and vehicle details, before

executing the insertion query. This backend functionality enables the seamless recording of traffic violations within the system.

```php
$file_name = basename($_FILES['image']['name']);
$file_path = $upload_dir . $file_name;

if (move_uploaded_file($_FILES['image']['tmp_name'], $file_path)) {
    $stmt = $conn->prepare("INSERT INTO evidence (ticket_no, image_name) VALUES (?, ?)");
    $stmt->bind_param("ss", $ticket_no, $file_name);
```

Figure 64 code snippet for Upload Image and Insert Evidence Record – upload_image.php

Figure 64 shows the `upload_image.php` script, which manages the upload of images related to a violation ticket. The script processes the image file by moving it to a designated server directory and then records its association with the ticket in the `evidence` table. The `move_uploaded_file` function ensures the file is securely stored, while a prepared statement prevents SQL injection when inserting the image details into the database. This functionality allows enforcers to attach photographic evidence to tickets, enhancing documentation and enforcement accuracy within the system.

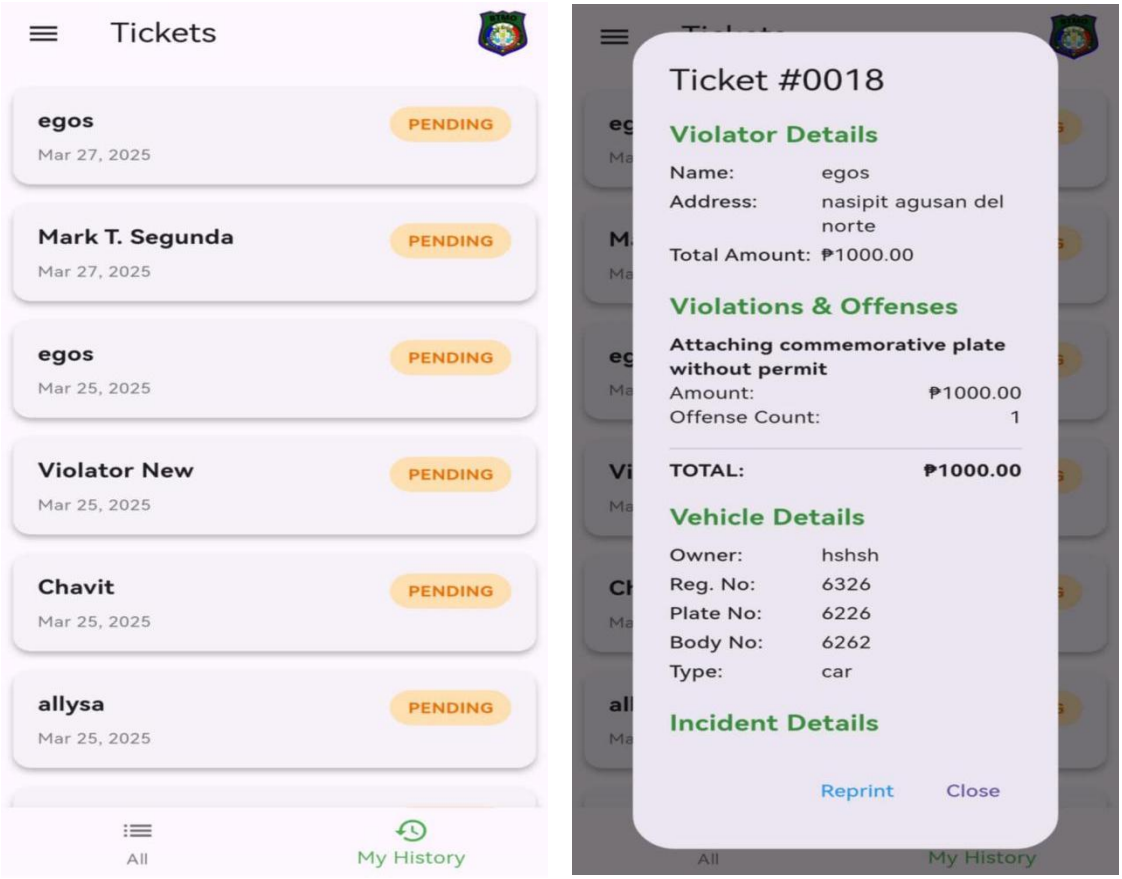


Figure 65. Enforcer Side – Tickets List Page

The Tickets List page in the Traffic Regulation and Record Management System (TRRMS) provides enforcers with a complete history of issued tickets. Each ticket entry can be tapped to display its corresponding details, including violator information (name, address, total penalty amount), violations and offenses (offense type, penalty amount), vehicle details (owner, registration number, plate number, body number, vehicle type), and incident details. The page also includes options for reprinting the ticket for documentation purposes and a close button to exit the details view. This ensures that enforcers have quick access to past violations for record-keeping and enforcement efficiency.

```
if (isMyTicket)
                    TextButton(
                        onPressed: () {
                          _reprintTicket(ticket);
                        },
                        child: const Text('Reprint',
                            style: TextStyle(color: Colors.blue))),
```

Figure 66 code snippet for Reprint Ticket Button

Figure 66 shows the TICKETS LIST PAGE, which displays a history of issued tickets and allows enforcers to view ticket details. The `Reprint` button is conditionally shown only if the logged-in enforcer is the one who originally submitted the ticket. This is determined by the `isMyTicket` condition, ensuring that only authorized enforcers can reprint their own tickets. When tapped, the button triggers the `_reprintTicket(ticket)` function, which initiates the reprinting process. Styled as a `TextButton` with blue-colored text, the button provides a clear and accessible way for enforcers to generate a duplicate copy of a previously issued ticket while maintaining security and accountability in the system.

```
Future<void> _fetchTickets() async {
    setState(() {
      isLoading = true;
    });

    try {
      print("Attempting to fetch tickets with enforcer_id: $enforcerId");
      print(
          "Constructed URL for my tickets:
$baseUrl/get_recent_tickets.php?enforcer_id=${enforcerId.toString()}");

      final allTicketsResponse =
          await http.get(Uri.parse('$baseUrl/get_recent_tickets.php'));
      print("All tickets response status: ${allTicketsResponse.statusCode}");

      final myTicketsResponse = await http.get(Uri.parse(
          '$baseUrl/get_recent_tickets.php?enforcer_id=${enforcerId.toString()}'));
      print("My tickets response status: ${myTicketsResponse.statusCode}");

      if (allTicketsResponse.statusCode == 200 &&
          myTicketsResponse.statusCode == 200) {
        var allData = json.decode(allTicketsResponse.body);
        var myData = json.decode(myTicketsResponse.body);

        if (myData.containsKey('debug')) {
          print("Debug info from API: ${myData['debug']}");
        }

        if (allData['success'] && myData['success']) {
          setState(() {
            allTickets = allData['tickets'];
            myTickets = myData['tickets'];
            isLoading = false;
          });
        } else {
          print(
              "API reported failure: allData success=${allData['success']}, myData
success=${myData['success']}");
          setState(() {
            isLoading = false;
          });
        }
```

```
      } else {
        print(
            "HTTP request failed: allTickets status=${allTicketsResponse.statusCode}, myTickets
status=${myTicketsResponse.statusCode}");
        setState(() {
          isLoading = false;
        });
      }
    } catch (e) {
      print("Error fetching tickets: $e");
      setState(() {
        isLoading = false;
      });
    }
  }
```

Figure 67 code snippet for Fetch Tickets Function

Figure 67 shows the `_fetchTickets()` method, which retrieves ticket data from the backend using HTTP GET requests. It first fetches all tickets from `get_recent_tickets.php` and then retrieves tickets specifically issued by the logged-in enforcer using `get_recent_tickets.php?enforcer_id=....` The responses are decoded from JSON and stored in `allTickets` and `myTickets`. Debugging logs are included to track the request process, ensuring successful data retrieval. If both requests succeed, the state is updated to display the tickets; otherwise, error handling mechanisms prevent the UI from freezing by setting `isLoading` to `false`.

```
ListView.builder(
                            padding: const EdgeInsets.all(8.0),
                            itemCount: displayTickets.length,
                            itemBuilder: (context, index) {
                              var ticket = displayTickets[index];
                              bool isMyTicket =
                                  ticket['enforcer_id'].toString() ==
                                      enforcerId.toString();

                              return Card(
                                  elevation: 3,
                                  margin:
                                      const EdgeInsets.symmetric(vertical: 8.0),
                                  child: InkWell(
                                      onTap: () => _showTicketDetails(
                                          ticket, isMyTicket),
                                      child: Padding(
                                          padding: const EdgeInsets.all(16.0),
                                          child: Row(
                                              crossAxisAlignment:
                                                  CrossAxisAlignment.start,
                                              children: [
                                                Expanded(
                                                    child: Column(
                                                        crossAxisAlignment:
                                                            CrossAxisAlignment
                                                                .start,
```

```
                                        children: [
                                      Text(
                                          ticket[
                                              'violator_name'],
                                          style: const TextStyle(
                                              fontSize: 16,
                                              fontWeight:
                                                  FontWeight
                                                      .bold)),
                                      const SizedBox(height: 8),
                                      Text(
                                          _formatDate(ticket[
                                              'violation_date']),
                                          style: TextStyle(
                                              fontSize: 12,
                                              color: Colors.grey
                                                  .shade600))
                                  ])),
                              Container(
                                  padding:
                                      const EdgeInsets.symmetric(
                                          horizontal: 12,
                                          vertical: 6),
                                  decoration: BoxDecoration(
                                      color: ticket['status'] ==
                                              'pending'
                                          ? Colors
                                              .orange.shade100
                                          : Colors
                                              .green.shade100,
                                      borderRadius:
                                          BorderRadius.circular(
                                              20)),
                                  child: Text(ticket['status'].toUpperCase(),
                                      style: TextStyle(
                                          color: ticket['status'] ==
                                                  'pending'
                                              ? Colors.orange
                                                  .shade800
                                              : Colors.green.shade800,
                                          fontWeight: FontWeight.bold,
                                          fontSize: 12)))
                          ])));
                  })
```

Figure 68 code snippet for Ticket List View Builder

Figure 68 shows the `ListView.builder` widget, which dynamically generates a list of violation tickets. Each ticket is displayed as a `Card` containing key details such as the violator's name, violation date, and ticket status (e.g., pending, paid). When a ticket is tapped, it triggers the `_showTicketDetails()` method, opening a dialog with the ticket's full details. The UI visually distinguishes ticket statuses using colored labels—orange for pending tickets and green for resolved ones. Additionally, the system checks whether the logged-in enforcer issued the ticket, ensuring personalized interactions and access control.

```
void _showTicketDetails(dynamic ticket, bool isMyTicket) {
```

```
    print(
        "Ticket enforcer data: ${ticket['enforcer_id']} - ${ticket['enforcer_name']} -
${ticket['enforcer_badge_number']}");
    showDialog(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
              title: Text('Ticket #${ticket['ticket_no']}'),
              content: SingleChildScrollView(
                  child: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      mainAxisSize: MainAxisSize.min,
                      children: [
                    _buildSectionTitle('Violator Details'),
                    _buildDetailRow('Name', ticket['violator_name']),
                    _buildDetailRow('Address', ticket['violator_address']),
                    _buildDetailRow('Total Amount',
                        '₱${ticket['total_amount'].toStringAsFixed(2)}'),
                    const SizedBox(height: 16),
                    _buildSectionTitle('Violations & Offenses'),
                    ...List.generate(ticket['violations'].length, (index) {
                      var violation = ticket['violations'][index];
                      return Padding(
                          padding: const EdgeInsets.only(bottom: 8.0),
                          child: Column(
                              crossAxisAlignment: CrossAxisAlignment.start,
                              children: [
                                Text(violation['violation_type'],
                                    style: const TextStyle(
                                        fontWeight: FontWeight.bold)),
                                Row(
                                    mainAxisAlignment:
                                        MainAxisAlignment.spaceBetween,
                                    children: [
                                      Text('Amount:'),
                                      Text(
                                          '₱${violation['penalty_amount'].toStringAsFixed(2)}')
                                    ]),
                                const Row(
                                    mainAxisAlignment:
                                        MainAxisAlignment.spaceBetween,
                                    children: [
                                      Text('Offense Count:'),
                                      Text('1')
                                    ])
                              ]));
                    }),
                    const Divider(),
                    Row(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          const Text('TOTAL:',
                              style: TextStyle(fontWeight: FontWeight.bold)),
                          Text('₱${ticket['total_amount'].toStringAsFixed(2)}',
                              style:
                                  const TextStyle(fontWeight: FontWeight.bold))
                        ]),
                    const SizedBox(height: 16),
                  ])),
              actions: [
                if (isMyTicket)
                  TextButton(
                      onPressed: () {
                        _reprintTicket(ticket);
                      },
                      child: const Text('Reprint',
                          style: TextStyle(color: Colors.blue))),
                TextButton(
```

```
                    onPressed: () {
                      Navigator.of(context).pop();
                    },
                    child: const Text('Close'))
                ]);
        });
}
```

Figure 69 code snippet for Show Ticket Details Dialog

Figure 69 shows the `_showTicketDetails` method, which displays a dialog containing comprehensive details about a violation ticket. The dialog presents structured sections, including violator details (name, address, total amount), violations & offenses (list of offenses with penalty amounts), vehicle details (owner, registration number, plate number), and incident details (date, time, location). It also displays enforcer details, such as the name and badge number of the issuing officer. If the logged-in enforcer issued the ticket, a "Reprint" button appears, allowing them to generate another copy. The dialog ensures clarity and accessibility of ticket records for efficient enforcement.

```
$sql = "SELECT vr.*, v.name as violator_name, v.email as violator_email,
        v.phone as violator_phone, v.address as violator_address
        FROM violationrecords vr
        INNER JOIN violators v ON vr.violator_id = v.violator_id";

if ($enforcerId) {
    $sql .= " WHERE vr.enforcer_id = '$enforcerId'";
    error_log("Applying filter for enforcer_id: $enforcerId");
}

$sql .= " ORDER BY vr.violation_date DESC";
error_log("Executing SQL query: " . $sql);
```

Figure 70 code snippet for Fetch Violation Records with Enforcer Filter – get_recent_ticket.php

Figure 70 shows the `get_recent_tickets.php` script, which retrieves violation tickets from the `violationrecords` table. The script joins the `violators` table to include additional violator details, such as name, email, phone number, and address. If an `enforcer_id` is provided, the query filters results to include only tickets issued by that specific enforcer. The results are sorted in descending order based on the violation date, ensuring that the most recent tickets appear first. Error logs are included to track query execution and filtering, aiding in debugging and monitoring backend operations.

```php
$tempTickets[$ticketNo] = [
                'ticket_no' => $ticketNo,
                'violator_id' => $row['violator_id'],
                'violator_name' => $row['violator_name'],
                'violator_email' => $row['violator_email'],
                'violator_phone' => $row['violator_phone'],
                'violator_address' => $row['violator_address'],
                'enforcer_id' => $rowEnforcerId,
                'enforcer_name' => $enforcerName,
                'enforcer_badge_number' => $enforcerBadge,
                'violation_date' => $row['violation_date'],
                'location' => $row['location'],
                'vehicle_owner' => $row['vehicle_owner'],
                'registration_no' => $row['registration_no'],
                'plate_no' => $row['plate_no'],
                'body_no' => $row['body_no'],
                'vehicle_type' => $row['vehicle_type'],
                'status' => $row['status'],
                'violations' => [],
                'total_amount' => 0
            ];
  $offenseCount = 1;
       $key = $row['violator_id'] . '_' . $row['violation_type'];

       if (isset($offenseCounts[$key])) {
           foreach ($offenseCounts[$key] as $index => $offense) {
               if ($offense['ticket_no'] == $ticketNo) {
                   $offenseCount = $index + 1;
                   break;
               }
           }
       }
```

Figure 71 code snippet for Organize Violation Ticket Data with Offense Count

Figure 71 shows the ticket grouping logic in the backend, where multiple violations are consolidated under a single `ticket_no`. The script organizes ticket data by assigning violator details (such as name and address), enforcer information, and vehicle details to each ticket. Additionally, it maintains an array to store associated violations, ensuring that multiple offenses committed by the same violator are grouped together. The `total_amount` field is dynamically updated to reflect the sum of all violation penalties. An offense counter tracks repeated violations by checking existing records, allowing accurate identification of offense counts within a ticket.

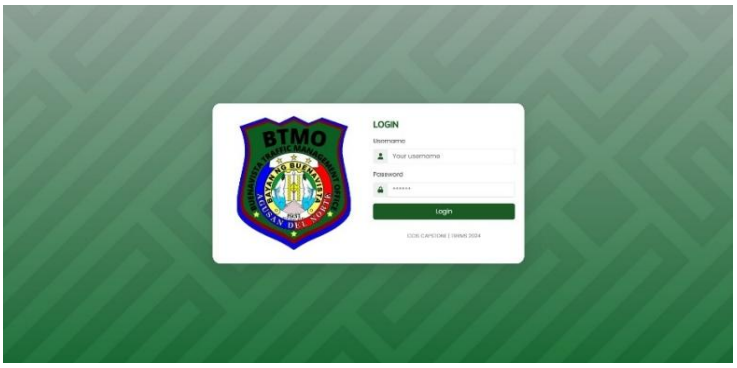


Figure 72. BTMO Head/Staff – Login Page

This section displays the login page for the BTMO Head and Staff, serving as their first point when accessing the TRRMS. This interface includes input fields where they can enter their assigned Username and Password to log into their accounts securely.

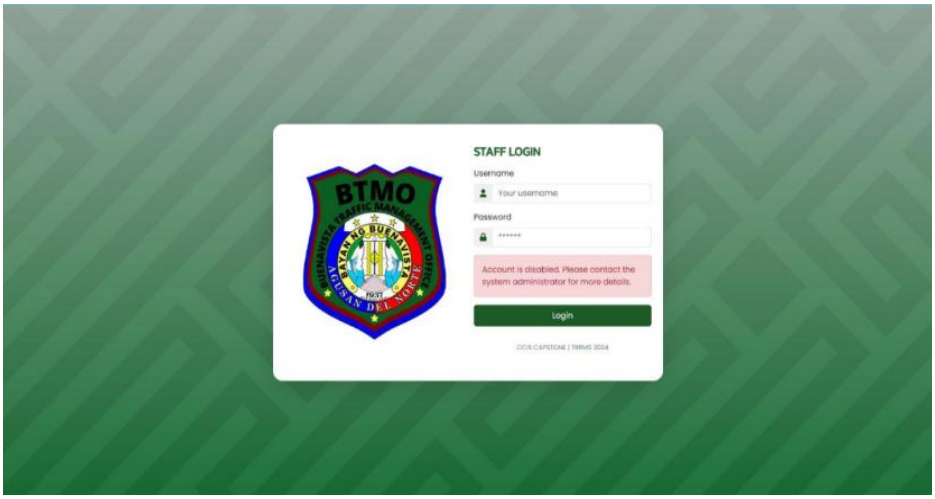


Figure 73. BTMO Head/Staff–Login Error Notification

The system displays an error modal when an account is disabled or declined, or if the provided Username and Password do not match any records in the database. The notification alerts the user that their credentials are incorrect or their account access is restricted.

```
<?php
include 'controllers/login.php';
?>
 <form method="POST" action="">
<input type="text" class="form-control" id="username" name="username" placeholder="Your
username"required>
 <input type="password" class="form-control" id="password" name="password" placeholder="******"
required>
<?php if ($error_message != "") { ?>
              <div class="alert alert-danger" role="alert"><?php echo $error_message; ?></div>
            <?php } ?>
<button type="submit" class="btn btn-green w-100 py-2 text-light">
              Login
            </button>
          </form>
```

Figure 74 Code snippet for BTMO Head/Staff – Login Page

Figure 74 shows the code snippet for the login page, where users enter their credentials to access the system. The form uses the POST method to send login data, including the username and password fields, to `controllers/login.php` for authentication. The PHP `include` statement at the beginning imports the `login.php` file, which processes the

login logic, validating user credentials against stored records. If an authentication error occurs, the `$error_message` variable stores the error message, which is then displayed inside a Bootstrap alert box, providing immediate feedback to users regarding failed login attempts.

The form consists of input fields for the username and password, both marked as required to ensure users provide the necessary credentials before submission. The login button, styled with Bootstrap classes, allows users to submit the form. If an error occurs during authentication, the system dynamically displays an error alert using PHP. This approach enhances the user experience by providing real-time feedback. Upon successful submission, the entered credentials are processed by `controllers/login.php`, which verifies user authenticity and grants access to the system if valid.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
```

Figure 75 code snippet for submit – login.php

The line `if ($_SERVER["REQUEST_METHOD"] == "POST")` { ensures that the login process runs only when the form is submitted via the POST method, preventing unauthorized access and ensuring secure data transmission.

```
$user = mysqli_real_escape_string($conn, $_POST['username']);
$pass = mysqli_real_escape_string($conn, $_POST['password']);
```

Figure 76 code snippet for Sanitizing for user input – login.php

These lines use `mysqli_real_escape_string()` to sanitize user input by escaping special characters, protecting the database from SQL injection attacks. This ensures that malicious SQL code cannot be injected through the `username` and `password` fields, preserving data integrity and system security.

```
$sql_superadmin = "SELECT * FROM superadmins WHERE username = '$user'";
$result_superadmin = $conn->query($sql_superadmin);

$sql_admin = "SELECT * FROM admins WHERE username = '$user'";
$result_admin = $conn->query($sql_admin);
```

Figure 77 code snippet for User Authentication Query for Multiple Roles – login.php

These SQL queries retrieve user records from the `superadmins` and `admins` tables to check if the entered username exists in either. The `query()` function executes these SQL statements, and the results determine whether the provided username matches a stored record. This step is crucial for identifying the user's role and verifying credentials during the authentication process. However, using prepared statements instead of directly embedding user input into SQL queries would enhance security by preventing SQL injection attacks.

```
if ($result_superadmin->num_rows > 0) {
   $row = $result_superadmin->fetch_assoc();
   if (password_verify($pass, $row['password'])) {

 elseif ($result_admin->num_rows > 0) {
        $row = $result_admin->fetch_assoc();
        if (password_verify($pass, $row['password'])) {
```

Figure 78 code snippet for Password Hash Verification – login.php

This code verifies the user's credentials by checking if the entered username exists in either the `superadmins` or `admins` table. If a match is found, it retrieves the account details and validates the entered password using `password_verify()`, which compares it against the securely hashed password stored in the database. This approach enhances security by ensuring that only authenticated users with valid credentials can gain access.

```
if ($row['status'] == 'disabled') {
                    $error_message = "Account is disabled. Please contact the system administrator
for more details.";
```

Figure 79 code snippet for Account Status Check in Authentication – login.php

This condition checks if the retrieved user account has a status of `disabled`. If so, it prevents login by setting an error message, informing the user that their account is restricted. This security measure ensures that inactive or suspended accounts cannot access the system, maintaining administrative control and preventing unauthorized use.

```
$_SESSION['username'] = $row['username'];
$_SESSION['name'] = $row['name'];
$_SESSION['phone'] = $row['phone'];
$_SESSION['email'] = $row['email'];
```

Figure 80 code snippet for User Session Initialization – login.php

Upon successful authentication, these lines store the user's details in session variables. This allows the system to maintain user-specific data across multiple pages, ensuring seamless access to personalized information without requiring reauthentication.

```
$name = $_SESSION['name'];
$info = $name . " logged in";
$date_time = date('Y-m-d H:i:s');
$action = "logged in";

$log_sql = "INSERT INTO logs (info, date_time, action) VALUES ('$info', '$date_time', '$action')";
if (!$conn->query($log_sql)) {
   error_log("Failed to insert log: " . $conn->error);
}
```

Figure 81 code snippet for User Activity Logging – login.php

This code logs user activity by recording the login event in the `logs` table. It captures the user's name, the current date and time, and the action performed. If the database query fails, an error message is logged for debugging purposes.

```
header("Location: /pages/dashboard.php");
exit();
```

Figure 82 code snippet for Page Redirection – Login.php

After verifying the user's credentials, the system redirects them to the dashboard page using `header("Location: /pages/dashboard.php");`. The `exit();` function immediately stops script execution to ensure a smooth transition without processing any additional code.

```
 $error_message = "Invalid username or password.";
```

Figure 83 code snippet for Login Error Message – login.php

If the entered username or password is incorrect, the system assigns an error message to `$error_message`, informing the user that their login credentials are invalid. This helps provide feedback and prompts the user to re-enter the correct details.

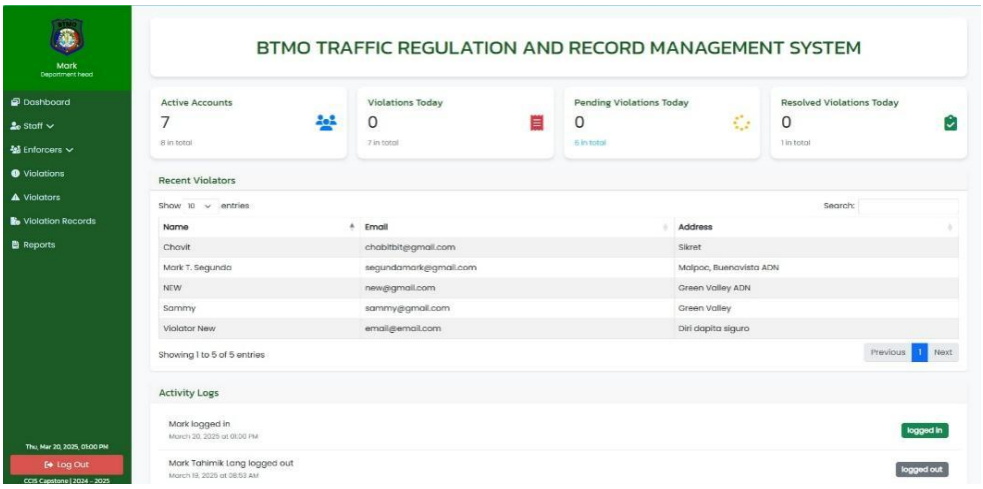


Figure 84. BTMO Head/Staff–Dashboard

The BTMO Head Dashboard provides an overview of key enforcement metrics, including active accounts, violations today, pending violations today, and resolved violations today. It also features a recent violators list and activity logs, ensuring real-time monitoring and efficient management of enforcement activities. This streamlined interface enhances data tracking, accountability, and decision-making within the traffic regulation system.

```
$query = "SELECT COUNT(*) as active_count FROM enforcers WHERE status = 'active'";
$result = $conn->query($query);
$active_enforcers = $result->fetch_assoc()['active_count'];

$query = "SELECT COUNT(*) as active_count FROM admins WHERE status = 'active'";
$result = $conn->query($query);
$active_admins = $result->fetch_assoc()['active_count'];

$response['active_accounts'] = $active_enforcers + $active_admins;

$query = "SELECT COUNT(*) as total_count FROM enforcers";
$result = $conn->query($query);
$total_enforcers = $result->fetch_assoc()['total_count'];

$query = "SELECT COUNT(*) as total_count FROM admins";
$result = $conn->query($query);
$total_admins = $result->fetch_assoc()['total_count'];

$response['total_accounts'] = $total_enforcers + $total_admins;

date_default_timezone_set('Asia/Manila');
$current_date = date('Y-m-d');
$query = "SELECT COUNT(*) as violations_today FROM violationrecords WHERE violation_date =
'$current_date'";
$result = $conn->query($query);
$response['violations_today'] = $result->fetch_assoc()['violations_today'];

$query = "SELECT COUNT(*) as total_violations FROM violationrecords";
$result = $conn->query($query);
$response['total_violations'] = $result->fetch_assoc()['total_violations'];
```

```
$query = "SELECT COUNT(*) as pending_violations_today FROM violationrecords WHERE violation_date =
'$current_date' AND status = 'pending'";
$result = $conn->query($query);
$response['pending_violations_today'] = $result->fetch_assoc()['pending_violations_today'];

$query = "SELECT COUNT(*) as total_pending_violations FROM violationrecords WHERE status =
'pending'";
$result = $conn->query($query);
$response['total_pending_violations'] = $result->fetch_assoc()['total_pending_violations'];

$query = "SELECT COUNT(*) as resolved_violations_today FROM violationrecords WHERE status =
'resolved' AND resolved_date = '$current_date'";
$result = $conn->query($query);
$response['resolved_violations_today'] = $result->fetch_assoc()['resolved_violations_today'];

$query = "SELECT COUNT(*) as total_resolved_violations FROM violationrecords WHERE status =
'resolved'";
$result = $conn->query($query);
$response['total_resolved_violations'] = $result->fetch_assoc()['total_resolved_violations'];
```

Figure 85 code snippet Dashboard Statistics Query – get_counts.php

Figure 85 shows the code snippet for `get_counts.php`, which retrieves statistical data for the dashboard. This backend script queries the database to count active accounts, total accounts, and violation records. The script first counts active enforcers and admins by querying the `enforcers` and `admins` tables where the `status` is `'active'`. These counts are stored in `$response['active_accounts']`. Similarly, the total number of enforcers and admins is retrieved and stored in `$response['total_accounts']`. To track violations, the script sets the timezone to `'Asia/Manila'` and retrieves today's date using the `date()` function. It then queries the `violationrecords` table to count the number of violations recorded today, storing the result in `$response['violations_today']`. Additionally, the script fetches the total number of violations, pending violations (both today and overall), and resolved violations (both today and overall). Each query result is stored in the `$response` array, which can be used to display real-time data in the dashboard. This approach ensures efficient data retrieval and enhances system monitoring by providing up-to-date information on active users and violation records.

```
<div class="dashboard-cards">
                    <div class="dashboard-card">
                        <div class="card-content">
                            <h4 class="fg-green kanit">
                                Active Accounts
                            </h4>
```

```
                    <h2 class="fg-green">
                        32
                    </h2>
                    <small class="text-secondary">
                        50 in total
                    </small>
                </div>
                <i class="fas fa-users card-icon"></i>
            </div>
            <div class="dashboard-card">
                <div class="card-content">
                    <h4 class="fg-green kanit">
                        Violations Today
                    </h4>
                    <h2 class="fg-green">
                        127
                    </h2>
                    <small class="text-secondary">
                        1000 in total
                    </small>
                </div>
                <i class="fas fa-receipt card-icon text-danger"></i>
            </div>
            <div class="dashboard-card">
                <div class="card-content">
                    <h4 class="fg-green kanit">
                        Pending Violations Today
                    </h4>
                    <h2 class="fg-green">
                        12
                    </h2>
                    <small class="text-info">
                        20 in total
                    </small>
                </div>
                <i class="fas fa-spinner card-icon text-warning"></i>
            </div>
            <div class="dashboard-card">
                <div class="card-content">
                    <h4 class="fg-green kanit">
                        Resolved Violations Today
                    </h4>
                    <h2 class="fg-green">
                        3
                    </h2>
                    <small class="text-secondary">
                        100 in total
                    </small>
                </div>
                <i class="fas fa-clipboard-check card-icon text-success"></i>
            </div>
```

Figure 86 code snippet for Dashboard Summary Cards – dashboard.php

Figure 86 shows the code snippet for the dashboard cards in `dashboard.php`, which visually presents key statistics retrieved from the backend. These cards provide an overview of system activity, including the number of active accounts, violations recorded today, pending violations, and resolved violations. Each card is structured using a `<div class="dashboard-card">` container that houses the title, numerical data, and an icon

representing the metric. The first card displays the number of Active Accounts, showing the count of currently active users alongside the total number of accounts in the system. The second card highlights Violations Today, presenting the number of violations recorded within the current day, with the total count displayed below. The third card represents Pending Violations Today, displaying unresolved violations for the day, helping administrators monitor cases that need action. The last card shows Resolved Violations Today, tracking violations that have been addressed. Each card uses Font Awesome icons (`fa-users`, `fa-receipt`, `fa-spinner`, and `fa-clipboard-check`) for better visualization, with color-coded text styles (`text-danger`, `text-warning`, and `text-success`) to indicate urgency and status. This structured layout enhances the user experience by providing real-time insights into system activity, allowing administrators to make informed decisions efficiently.

```
 $.getJSON('../controllers/get_counts.php', function (data) {
                if (data.error) {
                    console.error('Error fetching counts:', data.error);
                } else {
                    $('.dashboard-card:nth-child(1) h2').text(data.active_accounts);
                    $('.dashboard-card:nth-child(1) small').text(data.total_accounts + ' in
total');

                    $('.dashboard-card:nth-child(2) h2').text(data.violations_today);
                    $('.dashboard-card:nth-child(2) small').text(data.total_violations + ' in
total');

                    $('.dashboard-card:nth-child(3) h2').text(data.pending_violations_today);
                    $('.dashboard-card:nth-child(3) small').text(data.total_pending_violations + '
in total');

                    $('.dashboard-card:nth-child(4) h2').text(data.resolved_violations_today);
                    $('.dashboard-card:nth-child(4) small').text(data.total_resolved_violations +
' in total');
                }
            });
```

Figure 87 code snippet for Dashboard Data Fetching with AJAX

Figure 87 presents the JavaScript code snippet responsible for dynamically updating the dashboard cards in `dashboard.php` by retrieving real-time data from `get_counts.php` via an AJAX request. Using the `$.getJSON()` function, the script fetches JSON-formatted data, logging errors to the console if retrieval fails. Otherwise, it updates

the dashboard by selecting each card using the `nth-child` selector and modifying its content dynamically. The first card displays Active Accounts, showing both active and total users, while the second card presents Violations Today, reflecting the number of violations recorded for the current day and the total count. The third card highlights Pending Violations Today, indicating unresolved cases alongside the total pending violations, whereas the fourth card represents Resolved Violations Today, showing successfully addressed violations with their cumulative total. This method enables seamless real-time updates, allowing administrators to monitor system activities efficiently and make data-driven decisions.

```
case 'violators':
            $sql = "SELECT * FROM $tableName";
            if ($id) {
                $sql .= " WHERE violator_id = " . intval($id);
            }
            break;
```

Figure 88 code snippet for switch case statement – get.php

Figure 88 shows the backend code responsible for retrieving data for the Recent Violators table. This code dynamically generates an SQL query to fetch violator records from the specified database table. The case `'violators'`: condition ensures that the query is executed only when the request is specifically for violator data. The query starts with `SELECT * FROM $tableName`, which retrieves all records from the designated table. If a specific `violator_id` is provided, the query appends a `WHERE` clause using `intval($id)`, ensuring that only the record with the matching ID is fetched. The `intval()` function is used for security purposes, converting the input into an integer to prevent SQL injection.

```
<div class="card shadow-sm mb-4">
                    <div class="card-header">
                        <h5 class="card-title mb-0 kanit fg-green">
                            Activity Logs
                        </h5>
                    </div>
                    <div class="card-body">
                        <ul id="activityLogsList" class="list-group">
                        </ul>
                    </div>
                </div>
```

Figure 89 code snippet for Activity Logs Card – Dashboard.php

Figure 89 shows the HTML structure for displaying the Activity Logs in a card-styled component. The card is designed with a shadow effect (`shadow-sm`) and margin spacing (`mb-4`) to enhance visual separation from other elements on the page. The card's header (`card-header`) contains a title, "Activity Logs", styled with the `kanit` font and a green foreground color (`fg-green`), ensuring consistency with the system's theme. Inside the card body (`card-body`), an unordered list (`ul`) with the ID `activityLogsList` is present. This list serves as a placeholder for dynamically inserted log entries, which are fetched from the backend and displayed as list items.

```
$.getJSON(`../controllers/get.php?table=${tableName}`, function (data) {
                if (data.error) {
                    alert(data.error);
                } else if (data.length > 0) {
                    data.sort((a, b) => new Date(b.date) - new Date(a.date));

                    const limitedData = data.slice(0, 5);

                    let rows = '';
                    limitedData.forEach(item => {
                        rows += `
                <tr>
                    <td>${item.name}</td>
                    <td>${item.email}</td>
                    <td>${item.address}</td>
                </tr>
            `;
                    });
                    $('#violatorsData').html(rows);
                } else {
                    $('#violatorsData').html(`<tr><td colspan='6'>No data found</td></tr>`);
                }

                $('#violatorsTable').DataTable();
            });
```

Figure 90 code snippet for Fetch and Display Violators Data – time.js

Figure 90 shows the JavaScript code responsible for retrieving data from the backend and dynamically populating the Violators Table. The script uses `` `$.getJSON()` `` to fetch data from the `` `get.php` `` controller, passing the specified table name as a query parameter. If an error is encountered in the response, an alert displays the error message. Otherwise, if the retrieved data contains records, it is sorted in descending order based on the date, ensuring the most recent entries appear first. The script then selects the latest five records and constructs corresponding table rows (`` `<tr>` ``) using a `` `forEach` `` loop, inserting violator details such as name, email,

and address into the table. If no records are found, a message `"No data found"` is displayed across all columns to inform users of the absence of data. Finally, the script initializes the DataTables plugin (`$('#violatorsTable').DataTable();`) to enhance the table with sorting, searching, and pagination functionalities.

```
$sql = "SELECT * FROM $tableName";
            break;
    }

    $result = $conn->query($sql);

    $data = [];
    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            $data[] = $row;
        }
    }
```

Figure 91 code snippet for Fetch All Records from a Table – get.php

Figure 91 shows the backend code responsible for retrieving data for the Activity Logs from the database. The script constructs an SQL query (`SELECT * FROM $tableName`) to fetch all records from the specified table. This query is executed using `$conn->query($sql)`, retrieving the data from the database. Once the query runs, an empty array `$data` is initialized to store the fetched records. If the query returns results (`$result->num_rows > 0`), a `while` loop iterates through each row using `$result->fetch_assoc()`, converting the retrieved records into an associative array format. The fetched data is then stored in the `$data` array, preparing it for further processing or conversion into JSON format for frontend use.

```
<div class="card shadow-sm mb-4">
                    <div class="card-header">
                        <h5 class="card-title mb-0 kanit fg-green">
                            Activity Logs
                        </h5>
                    </div>
                    <div class="card-body">
                        <ul id="activityLogsList" class="list-group">
                        </ul>
                    </div>
                </div>
```

Figure 92 code snippet for Activity Logs Card – dashboard.php

Figure 92 shows the HTML structure for displaying Activity Logs in a list format. The code defines a Bootstrap-styled card component that contains a header and a body section. The `<div class="card-header">` element holds the title Activity Logs, styled using the `kanit` and `fg-green` classes for a distinct appearance. The `<div class="card-body">` contains an unordered list `(<ul>)` with the `id="activityLogsList"` and the list-group class, which is used to dynamically populate the activity logs retrieved from the backend.

```
function fetchLogs() {
                $.ajax({
                    url: '../controllers/get.php',
                    type: 'GET',
                    data: { table: 'logs' },
                    dataType: 'json',
                    success: function (data) {
                        if (data.error) {
                            console.error('Error fetching logs:', data.error);
                        } else {
                            renderLogs(data);
                        }
                    },
                    error: function (xhr, status, error) {
                        console.error('AJAX Error:', status, error);
                    }
                });
            }
```

```
function renderLogs(logs) {
                const logsList = $('#activityLogsList');
                logsList.empty();

                logs.sort((a, b) => new Date(b.date_time) - new Date(a.date_time));

                const limitedLogs = logs.slice(0, 10);

                if (limitedLogs.length > 0) {
                    limitedLogs.forEach(log => {
                        const formattedDate = formatDateTime(log.date_time);
                        const listItem = `
                <li class="list-group-item d-flex justify-content-between align-items-center">
                    <div>
                        <span>${log.info}</span>
                        <br>
                        <span class="text-secondary small">${formattedDate}</span>
                    </div>
                    <span class="badge bg-${getBadgeClass(log.action)}">${log.action}</span>
                </li>
            `;
                        logsList.append(listItem);
                    });
                } else {
                    logsList.append('<li class="list-group-item">No logs available</li>');
                }
            }
```

Figure 93 code snippet for Fetch and Render Activity Logs – dashboard.php

Figure 93 shows the JavaScript code responsible for retrieving and displaying Activity Logs in the system. The function `fetchLogs()` makes an AJAX request to `../controllers/get.php`, specifying the logs table as the data source. The retrieved data is expected in JSON format. If an error occurs, it is logged to the console for debugging. Upon a successful response, the `renderLogs()` function is called to process and display the logs. The `renderLogs()` function first clears any existing entries in the `<ul id="activityLogsList">` element before sorting the logs in descending order based on the `date_time` field. It then selects the latest 10 logs for display. Each log entry is formatted with the `formatDateTime()` function and dynamically added to the list using the `<li>` element. The `getBadgeClass(log.action)` function assigns a Bootstrap badge class to visually categorize the log action.

Figure 94. BTMO Head–View Staff Accounts

The BTMO Head has exclusive access to view staff accounts, including names, email, phone number, username, status, and actions. The dashboard also features a search engine button, allowing for quick and efficient account management. This ensures streamlined oversight, accountability, and easy retrieval of staff information within the system.

```
<table id="table" class="table table-striped table-bordered">
                            <thead>
                                <tr>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-user"></i>
                                        Name
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-envelope"></i>
                                        Email
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-phone"></i>
                                        Phone
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-circle-user"></i>
                                        Username
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-circle-question"></i>
                                        Status
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-gear"></i>
                                        Actions
                                    </th>
                                </tr>
                            </thead>
                            <tbody id="adminAccounts">

                            </tbody>
                        </table>
```

Figure 95 code snippet for Admin Accounts Table Structure – admins.php

Figure 95 shows the table structure used to display the list of staff accounts in the system. The table is implemented using the `<table>` element with Bootstrap classes `table-striped` and `table-bordered` to enhance readability and styling. The table includes six columns: Name, Email, Phone, Username, Status, and Actions, each represented with an appropriate Font Awesome icon for better visual clarity. The `<thead>` section defines the table headers, ensuring that column labels remain fixed at the top for easy navigation. The headers are styled with `text-secondary`, `text-center`, and `kanit` classes to improve alignment and aesthetics. The `<tbody>` section, identified by `id="adminAccounts"`, serves as a dynamic placeholder where JavaScript will inject the list of staff accounts fetched from the backend.

```
$(document).ready(function () {
          const tableName = 'admins';
          $.getJSON(`../controllers/get.php?table=${tableName}`, function (data) {
              if (data.error) {
                  alert(data.error);
              } else if (data.length > 0) {
```

```
                    let rows = '';
                    data.forEach(item => {
                        const action = item.status === 'active' ? 'Disable' : 'Enable';
                        const nextStatus = item.status === 'active' ? 'disable' : 'enable';

                        rows += `
                    <tr>
                        <td>${item.name}</td>
                        <td>${item.email}</td>
                        <td>${item.phone}</td>
                        <td>${item.username}</td>
                        <td class="text-center"><span class="badge bg-${item.status === 'active' ?
'success' : 'danger'}">${item.status}</span></td>
                        <td class="text-center">
                                            <button class='btn btn-sm btn-${nextStatus === 'disable'
? 'secondary' : 'primary'}' onclick="confirmStatusChange(${item.admin_id},
'${nextStatus}')">${action}</button>
                        </td>
                    </tr>
                `;
                    });
                    $('#adminAccounts').html(rows);
                } else {
                    $('#adminAccounts').html(`<tr><td colspan='6'>No data found</td></tr>`);
                }

                $('#table').DataTable();
            });
        });
```

Figure 96 code snippet for Admin Accounts Table Data Fetching and Rendering – admins.php

Figure 96 shows a JavaScript/jQuery snippet responsible for dynamically populating an admin accounts table by fetching data from the backend using an AJAX request (`$.getJSON()`) to `get.php`. When the page loads, the script retrieves data from the `admins` table and displays admin details such as name, email, phone, username, and status. If an error occurs, an alert is triggered. If data is available, it iterates through the results, constructing table rows that display each admin's information. The status of each admin is visually represented using badges, where active accounts are labeled in green (`success`), and inactive accounts in red (`danger`). Additionally, the script includes a toggle button for enabling or disabling an admin account. The button dynamically changes based on the admin's current status—if active, it displays "Disable" with a secondary color; if inactive, it shows "Enable" with a primary color. Clicking the button calls the `confirmStatusChange()` function, passing the admin's ID and the corresponding next status for processing. The populated table rows are then inserted into the `#adminAccounts` element, and the `DataTable()` plugin is initialized to enhance

table functionality, including sorting and pagination, ensuring a user-friendly and efficient admin management interface.

```
function confirmStatusChange(id, action) {
            Swal.fire({
                title: 'Are you sure?',
                text: `You want to ${action} this account?`,
                icon: 'warning',
                confirmButtonColor: '#3085d6',
                cancelButtonColor: '#d33',
                showCancelButton: true,
                confirmButtonText: `Yes, ${action} it!`,
                cancelButtonText: 'No, cancel!',
                reverseButtons: true
            }).then((result) => {
                if (result.isConfirmed) {
                    updateStatus(id, action);
                }
            });
        }
```

Figure 97 code snippet for Status Confirmation Modal – admins.php

Figure 97 shows the confirmation alert before updating an account's status. The function `confirmStatusChange(id, action)` utilizes SweetAlert2 `(Swal.fire)` to prompt the admin with a warning message, ensuring deliberate modifications. The alert includes confirm `(Yes, <action> it!)` and cancel `(No, cancel!)` buttons. If the admin confirms, the function calls `updateStatus(id, action)` to proceed with the status update.

```
function updateStatus(id, action) {
            $.ajax({
                url: '../controllers/update.php',
                type: 'POST',
                data: { id, table: 'admins', action },
                success: function (response) {
                    const res = JSON.parse(response);
                    if (res.success) {
                        Swal.fire('Success', 'The status has been updated.', 'success');
                        setTimeout(() => location.reload(), 1000);
                    } else {
                        Swal.fire('Error', res.message || 'Failed to update status.', 'error');
                    }
                },
                error: function () {
                    Swal.fire('Error', 'An unexpected error occurred.', 'error');
                }
            });
        }
```

Figure 98 code snippet for Status Update via AJAX

Figure 98 shows the function `updateStatus(id, action)`, which sends a POST request to `update.php` to modify an account's status. The function passes the id, table (admins),

and action as data. Upon success, it parses the JSON response and displays a SweetAlert2 notification indicating whether the update was successful or not. If successful, the page reloads after 1 second to reflect the changes. If an error occurs, an alert notifies the admin of the failure.

```
default:
            $sql = "SELECT * FROM $tableName";
            break;
    }

    $result = $conn->query($sql);

    $data = [];
    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            $data[] = $row;
        }
    }
```

Figure 99 code snippet for Fetching Data from Database – get.php

Figure 99 shows a PHP code snippet that retrieves data from the database based on the requested table `(admins)`. The script constructs a SQL query using `SELECT * FROM $tableName`, where `$tableName` is dynamically set in the request. The query is executed using `$conn->query($sql)`, and the retrieved data is stored in an array. If results are found `($result->num_rows > 0)`, each row is fetched using `$result->fetch_assoc()` and added to the `$data` array.

```
else if ($table === 'admins') {
                $status = $action === 'disable' ? 'disabled' : 'active';
                $stmt = $conn->prepare("UPDATE $table SET status = ? WHERE admin_id = ?");
                $stmt->bind_param('si', $status, $id);

                $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' ' . ($action ===
'disable' ? 'disabled' : 'activated') . ' an account' :
                    'Unknown user ' . ($action === 'disable' ? 'disabled' : 'activated') . ' an
account';
                $logAction = $action === 'disable' ? 'disabled account' : 'activated account';
            }
```

Figure 100 code snippet for Status update in PHP – update.php

Figure 100 shows a PHP code snippet in `update.php` that handles updating the status of staff accounts in the `admins` table. The script checks if the request targets admins, then determines the new status (`disabled or active`) based on the action (`disable or enable`). Using a prepared statement `($stmt = $conn->prepare())`, it updates the

status field in the database for the specified `admin_id`. Additionally, it logs the action by capturing the name of the user performing the update (`$_SESSION['name']`), recording whether the account was `disabled` or `activated`.

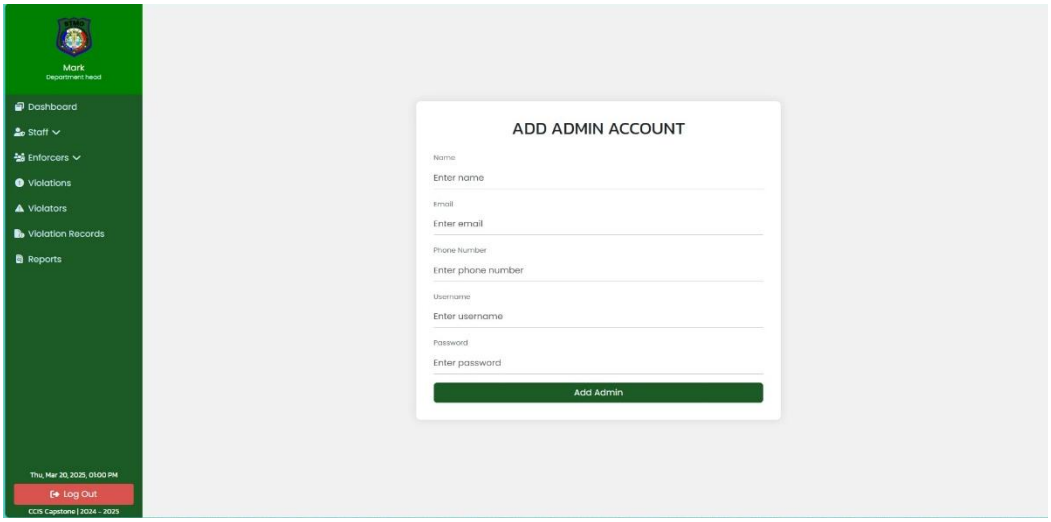


Figure 101. BTMO Head–Add Staff Account

The BTMO Head can add new staff accounts by assigning designated credentials, including name, email, phone number, username, and password. The interface features an "Add Staff" button, allowing for seamless account creation and management. This ensures efficient user registration, proper delegation of roles, and streamlined administrative control.

```
<form id="addAccount" method="POST">
                    <input type="hidden" name="table" value="admins">
                    <div class="mb-3">
                        <label for="name" class="form-label small text-secondary">
                            Name
                        </label>
                        <input type="text" class="form-control" id="name" name="name"
placeholder="Enter name" required>
                    </div>
                    <div class="mb-3">
                        <label for="email" class="form-label small text-secondary">
                            Email
                        </label>
                        <input type="email" class="form-control" id="email" name="email"
placeholder="Enter email"
                            required>
                    </div>
                    <div class="mb-3">
                        <label for="phone" class="form-label small text-secondary">
                            Phone Number
                        </label>
                        <input type="tel" class="form-control" id="phone" name="phone"
placeholder="Enter phone number"
                            required>
```

```
                </div>
                <div class="mb-3">
                    <label for="username" class="form-label small text-secondary">
                        Username
                    </label>
                    <input type="text" class="form-control" id="username" name="username"
                        placeholder="Enter username" required>
                </div>
                <div class="mb-3">
                    <label for="password" class="form-label small text-secondary">
                        Password
                    </label>
                    <input type="password" class="form-control" id="password" name="password"
                        placeholder="Enter password" required>
                </div>
                <div class="d-grid gap-2">
                    <button type="submit" class="btn btn-green text-light">
                        Add Staff
                    </button>
                </div>
            </form>
```

Figure 102 code snippet for add accountform – add_admin.php

Figure 102 shows an HTML form for adding a new staff account to the "admins" table. The form uses the POST method and includes input fields for collecting essential details such as name, email, phone number, username, and password, all marked as required to ensure completeness. A hidden input field specifies the target table, while placeholders in each field enhance user experience. The form is styled with Bootstrap classes for a clean and responsive layout. Upon submission, the data is sent to a backend script for processing and storage in the database. A green "Add Staff" button provides a clear call to action for submitting the form.

```
$(document).ready(function () {
    $('#addAccount').on('submit', function (e) {
        e.preventDefault();

        $.ajax({
            url: '../controllers/add_account.php',
            type: 'POST',
            data: $(this).serialize(),
            success: function (response) {
                Swal.fire({
                    title: 'Success!',
                    text: 'Account added successfully!',
                    icon: 'success',
                    confirmButtonText: 'OK'
                }).then(() => {
                    $('#addAccount')[0].reset();
                    location.reload();
                });
            },
            error: function () {
                Swal.fire({
                    title: 'Error!',
                    text: 'There was an issue adding the account. Please try again.',
                    icon: 'error',
```

```
                    confirmButtonText: 'OK'
                });
            }
        });
    });
});
```

Figure 103 code snippet for AJAX Form Submission for Account Creation – add-account

Figure 103 shows the `add-account.js script`, which sends form data to the backend without refreshing the page and handles responses dynamically. When the "Add Staff" form `(#addAccount)` is submitted, an AJAX request is sent to `add_account.php` using the POST method. The form data is serialized and processed in the backend. If the request is successful, a SweetAlert popup confirms that the account was added. The form is then reset, and the page reloads to reflect the changes. If an error occurs, a SweetAlert notification informs the user of the issue.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
 $name = htmlspecialchars(trim($_POST['name']));
    $email = htmlspecialchars(trim($_POST['email']));
    $phone = htmlspecialchars(trim($_POST['phone']));
    $username = htmlspecialchars(trim($_POST['username']));
    $password = htmlspecialchars(trim($_POST['password']));
    $table = htmlspecialchars(trim($_POST['table']));
    $hashed_password = password_hash($password, PASSWORD_DEFAULT);
    $status = 'active';

$sql = "INSERT INTO $table (name, email, phone, username, password, status) VALUES (?, ?, ?, ?, ?,
?)";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("ssssss", $name, $email, $phone, $username, $hashed_password, $status);

        if ($stmt->execute()) {
            $info = $_SESSION['name'] . " inserted new admin account";
            $date_time = date('Y-m-d H:i:s');
            $action = "added account";
            $log_sql = "INSERT INTO logs (info, date_time, action) VALUES (?, ?, ?)";
            $log_stmt = $conn->prepare($log_sql);
            $log_stmt->bind_param("sss", $info, $date_time, $action);
            $log_stmt->execute();
            $log_stmt->close();

            echo "success";
        } else {
            http_response_code(500);
            echo "Error: " . $stmt->error;
        }
```

Figure 104 code snippet for adding account to database – add_account.php

Figure 104 shows the `add_account.php script`, which processes form submissions for adding staff accounts. When a POST request is received from `add-admin.php`, the script sanitizes input data to prevent security risks. The password is hashed using `password_hash()` before storing it in the database. The SQL query is executed using prepared statements to protect against SQL injection. If the insertion is successful, an activity log is recorded, and a `"success"` message is returned to trigger a SweetAlert confirmation. If an error occurs, an error response is sent.

Figure 105. BTMO Head/Staff–View Enforcer Accounts

The BTMO Head and Staff can view enforcer accounts, including relevant details such as name, email, phone number, username, status, and actions. The dashboard also features a search engine button for quick and efficient account management, ensuring effective monitoring and organized record-keeping of enforcer information.

```
<table id="table" class="table table-striped table-bordered">
                            <thead>
                                <tr>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-user"></i>
                                        Name
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-envelope"></i>
                                        Email
                                    </th>
```

```
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-phone"></i>
                                    Phone
                                </th>
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-circle-user"></i>
                                    Username
                                </th>
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-circle-question"></i>
                                    Status
                                </th>
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-gear"></i>
                                    Actions
                                </th>
                            </tr>
                        </thead>
                        <tbody id="enforcerAccounts">

                        </tbody>
                    </table>
```

Figure 106 code snippet for enforcer account table – enforcers.php

Figure 106 shows the `enforcers.php` page, which contains a table for displaying the list of enforcers. The table includes columns for Name, Email, Phone, Username, Status, and Actions. The `<tbody>` element with the `id="enforcerAccounts"` serves as a placeholder where enforcer data will be dynamically populated using JavaScript. Each column header includes an icon for better visual representation. The Actions column will later be used for performing operations such as editing or updating an enforcer's account.

```
$(document).ready(function () {
            const tableName = 'enforcers';
            $.getJSON(`../controllers/get.php?table=${tableName}`, function (data) {
                if (data.error) {
                    alert(data.error);
                } else if (data.length > 0) {
                    let rows = '';
                    data.forEach(item => {
                        const action = item.status === 'active' ? 'Disable' : 'Enable';
                        const nextStatus = item.status === 'active' ? 'disable' : 'enable';

                        rows += `
                    <tr>
                        <td>${item.name}</td>
                        <td>${item.email}</td>
                        <td>${item.phone}</td>
                        <td>${item.username}</td>
                        <td class="text-center"><span class="badge bg-${item.status === 'active' ?
'success' : 'danger'}">${item.status}</span></td>
                        <td class="text-center">
                            <button class='btn btn-sm btn-${nextStatus === 'disable' ? 'secondary'
: 'primary'}' onclick="confirmStatusChange(${item.enforcer_id},
'${nextStatus}')">${action}</button>
                        </td>
                    </tr>
                `;
```

```
                });
                $('#enforcerAccounts').html(rows);
            } else {
                $('#enforcerAccounts').html(`<tr><td colspan='6'>No data found</td></tr>`);
            }

            $('#table').DataTable();
        });
    });
```

Figure 107 code snippet for dynamic table population with status toggle – enforcers.php

Figure 107 shows the JavaScript code responsible for retrieving and displaying enforcer accounts dynamically. The script sends an AJAX request to `get.php` to fetch enforcer data from the `enforcers` table. If data is retrieved successfully, it generates HTML rows dynamically and updates the `<tbody>` with `id="enforcerAccounts"`. Each row includes details such as name, email, phone, username, and status. The status column displays a badge (`Active` in green or `Disabled` in red). The Actions column contains a button that allows toggling the account's status between Enable and Disable using the `confirmStatusChange()` function. Finally, the script initializes the DataTables plugin on the table for better user interaction.

```
$tables = ['admins', 'enforcers', 'logs', 'violators', 'violationrecords', 'violations'];
default:
            $sql = "SELECT * FROM $tableName";
            break;
    }

    $result = $conn->query($sql);

    $data = [];
    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            $data[] = $row;
        }
    }
```

Figure 108 code snippet for fetch data from specific tables– get.php

Figure 108 shows the PHP script `(get.php)` that retrieves data from the database based on the requested table. The script first checks if the requested table exists in the predefined list `(admins, enforcers, logs, violators, violationrecords, violations)`. If a valid table is specified, it executes an SQL query `(SELECT * FROM $tableName)` to fetch all records. If data is found, it loops through each record using

`fetch_assoc()` and stores it in an array. Finally, the script returns the data in JSON format, which is later processed in JavaScript for dynamic content updates.

```
function confirmStatusChange(id, action) {
            Swal.fire({
                title: 'Are you sure?',
                text: `You want to ${action} this account?`,
                icon: 'warning',
                confirmButtonColor: '#3085d6',
                cancelButtonColor: '#d33',
                showCancelButton: true,
                confirmButtonText: `Yes, ${action} it!`,
                cancelButtonText: 'No, cancel!',
                reverseButtons: true
            }).then((result) => {
                if (result.isConfirmed) {
                    updateStatus(id, action);
                }
            });
        }
```

```
function updateStatus(id, action) {
            $.ajax({
                url: '../controllers/update.php',
                type: 'POST',
                data: { id, table: 'enforcers', action },
                success: function (response) {
                    const res = JSON.parse(response);
                    if (res.success) {
                        Swal.fire('Success', 'The status has been updated.', 'success');
                        setTimeout(() => location.reload(), 1000);
                    } else {
                        Swal.fire('Error', res.message || 'Failed to update status.', 'error');
                    }
                },
                error: function () {
                    Swal.fire('Error', 'An unexpected error occurred.', 'error');
                }
            });
        }
```

Figure 109 code snippet for Confirm and Update Account Status

Figure 109 shows the JavaScript code responsible for updating the status of enforcer accounts dynamically. The script first displays a confirmation alert (SweetAlert JS) when an admin attempts to change an account's status. If confirmed, it calls `updateStatus(id, action)`, which sends an AJAX request to `update.php` with the `id`, `table: 'enforcers'`, and `action (enable or disable)`. If the status update is successful, a success alert appears, and the page reloads after 1 second. Otherwise, an error alert is shown.

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $id = isset($_POST['id']) ? $_POST['id'] : 0;
    $table = isset($_POST['table']) ? $_POST['table'] : '';
    $action = isset($_POST['action']) ? $_POST['action'] : '';
```

```php
if ($table === 'enforcers') {
                $status = $action === 'disable' ? 'disabled' : 'active';
                $stmt = $conn->prepare("UPDATE $table SET status = ? WHERE enforcer_id = ?");
                $stmt->bind_param('si', $status, $id);

                $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' ' . ($action === 
'disable' ? 'disabled' : 'activated') . ' an account' :
                    'Unknown user ' . ($action === 'disable' ? 'disabled' : 'activated') . ' an 
account';
                $logAction = $action === 'disable' ? 'disabled account' : 'activated account';
            }
```

Figure 110 code snippet for Update enforcer status – update.php

Figure 110 shows the PHP code responsible for updating the status of enforcer accounts. The script receives a POST request, checks if the table is `enforcers`, and determines whether to set the status as `active` or `disabled` based on the `action` parameter. It then prepares an SQL UPDATE query to modify the `status` field in the `enforcers` table and executes it using prepared statements to prevent SQL injection. Additionally, the action is logged, recording who performed the status change. A success message is returned, triggering a SweetAlert success notification on the page.

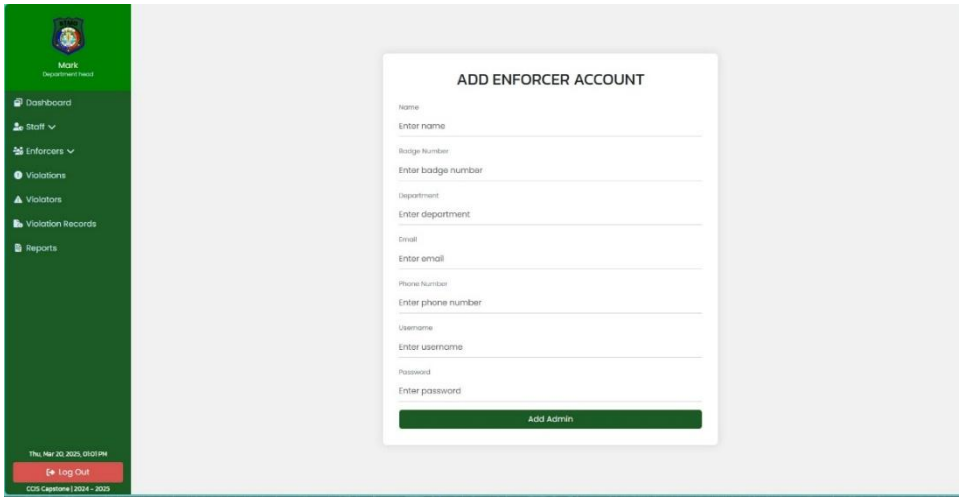


Figure 111. BTMO Head/Staff–Add Enforcer Account

The BTMO Head and Staff can add new enforcer accounts by assigning designated credentials, including name, badge number, department, email, phone number, username, and password. The interface features an "Add Enforcer" button, enabling seamless account creation

and management. This ensures efficient user registration, proper role delegation, and streamlined enforcement operations.

```
<form id="addAccount" method="POST">
                    <input type="hidden" name="table" value="enforcers">
                    <div class="mb-3">
                        <label for="name" class="form-label small text-secondary">
                            Name
                        </label>
                        <input type="text" class="form-control" id="name" name="name"
placeholder="Enter name" required>
                    </div>
                    <div class="mb-3">
                        <label for="badge" class="form-label small text-secondary">
                            Badge Number
                        </label>
                        <input type="text" class="form-control" id="badge" name="badge"
placeholder="Enter badge number"
                            required>
                    </div>
                    <div class="mb-3">
                        <label for="department" class="form-label small text-secondary">
                            Department
                        </label>
                        <input type="text" class="form-control" id="department" name="department"
                            placeholder="Enter department" required>
                    </div>
                    <div class="mb-3">
                        <label for="email" class="form-label small text-secondary">
                            Email
                        </label>
                        <input type="email" class="form-control" id="email" name="email"
placeholder="Enter email"
                            required>
                    </div>
                    <div class="mb-3">
                        <label for="phone" class="form-label small text-secondary">
                            Phone Number
                        </label>
                        <input type="tel" class="form-control" id="phone" name="phone"
placeholder="Enter phone number"
                            required>
                    </div>
                    <div class="mb-3">
                        <label for="username" class="form-label small text-secondary">
                            Username
                        </label>
                        <input type="text" class="form-control" id="username" name="username"
                            placeholder="Enter username" required>
                    </div>
                    <div class="mb-3">
                        <label for="password" class="form-label small text-secondary">
                            Password
                        </label>
                        <input type="password" class="form-control" id="password" name="password"
                            placeholder="Enter password" required>
                    </div>
                    <div class="d-grid gap-2">
                        <button type="submit" class="btn btn-green text-light">
                            Add Enforcer
                        </button>
                    </div>
                </form>
```

Figure 112 code snippet for Add Enforcer Form – add-enforcer.php

Figure 112 shows the HTML form designed for adding a new enforcer account. This form allows users to input necessary details such as the enforcer's name, badge number, department, email, phone number, username, and password. A hidden input field `(table)` is included with a predefined value of `enforcers`, ensuring that the submitted data is stored in the correct database table. Each field is accompanied by a descriptive label and marked as required to enforce data completeness. The password input field ensures security by masking user input. Upon submission, the form sends a POST request to the backend, where the provided details are validated and stored in the database. If the operation is successful, a success alert is displayed, confirming the creation of the new enforcer account.

```
$(document).ready(function () {
    $('#addAccount').on('submit', function (e) {
        e.preventDefault();

        $.ajax({
            url: '../controllers/add_account.php',
            type: 'POST',
            data: $(this).serialize(),
            success: function (response) {
                Swal.fire({
                    title: 'Success!',
                    text: 'Account added successfully!',
                    icon: 'success',
                    confirmButtonText: 'OK'
                }).then(() => {
                    $('#addAccount')[0].reset();
                    location.reload();
                });
            },
            error: function () {
                Swal.fire({
                    title: 'Error!',
                    text: 'There was an issue adding the account. Please try again.',
                    icon: 'error',
                    confirmButtonText: 'OK'
                });
            }
        });
    });
});
```

Figure 113 code snippet for Enforcer Account Add – add-account

Figure 113 shows the JavaScript code responsible for handling the submission of the "Add Enforcer" form asynchronously. The script listens for the form's submit event and prevents the default page reload using `e.preventDefault()`. It then sends the form data to

`add_account.php` via an AJAX POST request. If the request is successful, a SweetAlert notification appears, confirming that the account was added. The form is then reset, and the page reloads to reflect the updated enforcer list. If an error occurs, another SweetAlert notification is displayed, informing the user that the operation failed.

```php
if ($table === "enforcers") {
        $department = htmlspecialchars(trim($_POST['department']));
        $badge = htmlspecialchars(trim($_POST['badge']));

        $sql = "INSERT INTO $table (name, email, phone, username, password, department,
badge_number, status) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("ssssssss", $name, $email, $phone, $username, $hashed_password,
$department, $badge, $status);

        if ($stmt->execute()) {
            $info = $_SESSION['name'] . " inserted new enforcer account";
            $date_time = date('Y-m-d H:i:s');
            $action = "added account";
            $log_sql = "INSERT INTO logs (info, date_time, action) VALUES (?, ?, ?)";
            $log_stmt = $conn->prepare($log_sql);
            $log_stmt->bind_param("sss", $info, $date_time, $action);
            $log_stmt->execute();
            $log_stmt->close();

            echo "success";
        } else {
            http_response_code(500);
            echo "Error: " . $stmt->error;
        }
    }
```

Figure 114 code snippet for Add Enforcer Account– add_account.php

Figure 114 shows the PHP backend code responsible for handling the addition of a new enforcer account. The script receives POST data from `add-enforcer.php`, sanitizes inputs using `htmlspecialchars(trim())`, and hashes the password using `password_hash()`. A prepared statement is used to insert the enforcer's details into the `enforcers` table, preventing SQL injection. If the insertion is successful, an activity log is recorded in the `logs` table, and a success message is returned, triggering a SweetAlert confirmation on the frontend. If an error occurs, the script returns a 500 response with an error message.

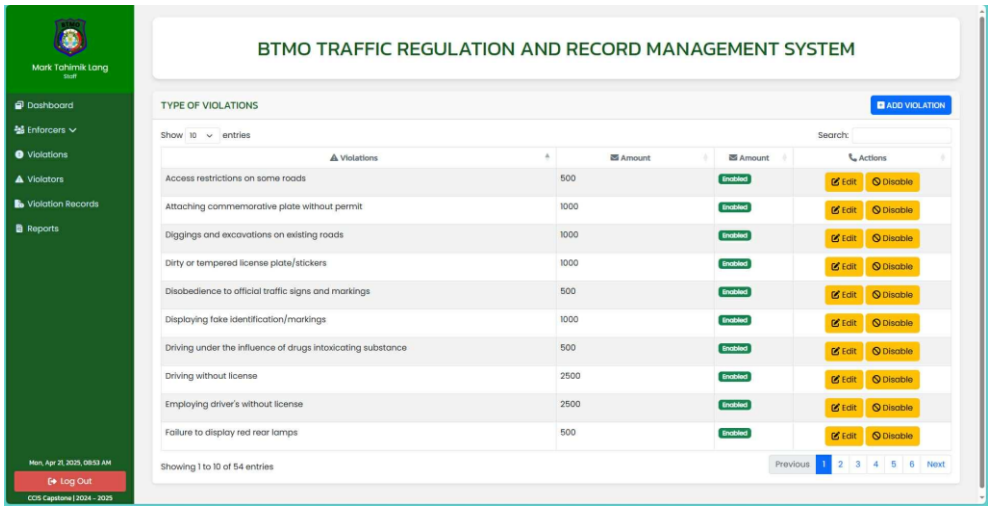


Figure 115. BTMO Head/Staff–Violations Page

The Violations Page showcases the type of violation, penalty amount, and corresponding actions. The interface also includes an "Add Violation" button, allowing users to record new violations efficiently.

```
 <table id="table" class="table table-striped table-bordered">
                            <thead>
                                <tr>
                                    <th style="width: 60%;" class="text-secondary text-center
kanit">
                                        <i class="fa-solid fa-triangle-exclamation"></i>
                                        Violations
                                    </th>
                                    <th style="width: 20%;" class="text-secondary text-center
kanit">
                                        <i class="fa-solid fa-envelope"></i>
                                        Amount
                                    </th>
                                    <th style="width: 20%;" class="text-secondary text-center
kanit">
                                        <i class="fa-solid fa-phone"></i>
                                        Actions
                                    </th>
                                </tr>
                            </thead>
                            <tbody id="violations">

                            </tbody>
                        </table>
```

Figure 116 code snippet for Violations List Table – violations.php

Figure 116 shows the `violations.php` file, which displays a table listing traffic violations. It consists of three columns: Violations, Amount, and Actions. The `<tbody>` with `id="violations"` is dynamically updated via JavaScript to fetch real-time data from the backend, allowing enforcers to manage violations efficiently.

```
$.getJSON(`../controllers/get.php?table=${tableName}`, function (data) {
                if (data.error) {
                    alert(data.error);
                } else if (data.length > 0) {
                    let rows = '';
                    data.forEach(item => {
                        rows += `
                        <tr>
                            <td>${item.name}</td>
                            <td>${item.amount}</td>
                            <td class="text-center">
                                <button class="btn btn-warning btn-edit"
                                        data-id="${item.violation_id}"
                                        data-name="${item.name}"
                                        data-amount="${item.amount}">
                                    <i class="fa fa-edit"></i> Edit
                                </button>
                                <button class="btn btn-danger btn-delete"
                                        data-id="${item.violation_id}">
                                    <i class="fa fa-trash"></i> Delete
                                </button>
                            </td>
                        </tr>
                    `;
                    });
                    $('#violations').html(rows);
                } else {
                    $('#violations').html(`<tr><td colspan='3'>No data found</td></tr>`);
                }

                $('#table').DataTable();
            });
```

Figure 117 code snippet for Fetch and Display Violations with Actions

Figure 117 shows the JavaScript code responsible for retrieving and displaying violations dynamically. It sends an AJAX request to `get.php` to fetch all violations from the database. If data is retrieved successfully, it generates table rows containing the violation name, amount, and action buttons for editing and deleting. The script then populates the `<tbody>` with `id="violations"`, ensuring real-time updates. If no data is found, it displays a message indicating an empty table. Finally, it initializes the DataTable plugin for better table functionality.

```
  $(document).on('click', '.btn-edit', function () {
                const id = $(this).data('id');
                const name = $(this).data('name');
                const amount = $(this).data('amount');

                $('#editViolationId').val(id);
                $('#editViolationType').val(name);
                $('#editAmount').val(amount);

                $('#editViolationModal').modal('show');
            });
```

Figure 118 code snippet for Edit Violation Modal Trigger

Figure 118 shows the JavaScript code responsible for handling the edit functionality for violations. When the Edit button is clicked, it retrieves the violation's ID, name, and amount from the button's `data` attributes. These values are then assigned to the corresponding input fields in the edit form. Finally, the Edit Violation modal `(#editViolationModal)` is displayed, allowing users to modify the violation details.

```
if (isset($_GET['table']) && in_array($_GET['table'], $tables)) {
    $tableName = $_GET['table'];
 $sql = "SELECT * FROM $tableName";
```

Figure 119 code snippet for Dynamic Table Data Retrieval – get.php

Figure 119 shows the PHP script responsible for retrieving records from the database. It first verifies if the requested table exists in the predefined list of allowed tables to prevent unauthorized access. If valid, it constructs an SQL query to fetch all records from the specified table, such as the `violations` table, ensuring secure and efficient data retrieval.

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $id = $_POST['id'];
    $violationType = $_POST['violationType'];
    $amount = $_POST['amount'];
```

Figure 120 code snippet for Handle Violation Data Submission – get.php

Figure 120 shows the PHP script responsible for handling edit requests. It first checks if the request method is `POST` to ensure proper data submission. Then, it retrieves and validates the `id`, `violationType`, and `amount` from the form input, preparing the data for further processing, such as updating the `violations` table in the database.

```
                $status = $action === 'disable' ? 'disabled' : 'enabled';
                $stmt = $conn->prepare("UPDATE $table SET status = ? WHERE violation_id = ?");
                if (!$stmt) {
                    throw new Exception("Prepare failed: " . $conn->error);
                }
                $stmt->bind_param('si', $status, $id);

                $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' ' . ($action ===
'disable' ? 'disabled' : 'enabled') . ' a violation type' :
                    'Unknown user ' . ($action === 'disable' ? 'disabled' : 'enabled') . ' a
violation type';
                $logAction = $action === 'disable' ? 'disabled violation type' : 'enabled
violation type';
            }
```

Figure 121 code snippet for Update Violation Data – update.php

Figure 121 shows the SQL query used to update a violation record in the database. The script utilizes prepared statements to prevent SQL injection, ensuring secure data handling. It updates the `name` and `amount` fields in the `violations` table where the `violation_id` matches the provided ID. The `bind_param()` function securely binds the values before execution.

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $id = $_POST['id'];
    $table = $_POST['table'];
 $sql = "DELETE FROM $table WHERE violation_id = ?";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("i", $id);
        $stmt->execute();
```

Figure 122 code snippet for Delete Violation from Database– delete.php

Figure 122 shows the SQL query responsible for deleting a violation record from the database. The script first verifies that the request method is `POST`, then retrieves the `id` and `table` parameters. Using a prepared statement to prevent SQL injection, it executes a `DELETE` query to remove the record from the specified table where the `violation_id` matches the provided ID.

Figure 123. BTMO Head/Staff–Violators Page

The Violators Page displays a comprehensive record of violators' names, violation types, number of offenses, locations, dates, penalty amounts, assigned enforcer, photo evidence, status,

and resolution. It also features a search function for quick data retrieval, ensuring efficient tracking and management of violation records.

```
<table id="table" class="table table-striped table-bordered">
                            <thead>
                                <tr>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-user"></i>
                                        Name
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-envelope"></i>
                                        Email
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-phone"></i>
                                        Phone
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-circle-user"></i>
                                        Address
                                    </th>
                                </tr>
                            </thead>
                            <tbody id="violators">

                            </tbody>
                        </table>
```

Figure 124 code snippet for Displaying Violator Information– violators.php

Figure 124 shows a code snippet for the violator account table in violators.php. This HTML table is designed to display a list of violators with columns for Name, Email, Phone, and Address. The `<thead>` section defines the table headers, using Font Awesome icons for better visual representation. The `<tbody>` with the ID `violators` serves as a placeholder where data will be dynamically inserted, likely through JavaScript or AJAX. The table is styled with Bootstrap classes `(table-striped, table-bordered)` for a clean and structured appearance.

```
  $(document).ready(function () {
            const tableName = 'violators';
            $.getJSON(`../controllers/get.php?table=${tableName}`, function (data) {
                if (data.error) {
                    alert(data.error);
                } else if (data.length > 0) {
                    let rows = '';
                    data.forEach(item => {
                        rows += `
                <tr>
                    <td>${item.name}</td>
                    <td>${item.email}</td>
                    <td>${item.phone}</td>
                    <td>${item.address}</td>
                </tr>
                `;
                    });
```

```
                $('#violators').html(rows);
            } else {
                $('#violators').html(`<tr><td colspan='6'>No data found</td></tr>`);
            }

            $('#table').DataTable();
        });
    });
```

Figure 125 code snippet for Display Violator Data in Table

Figure 125 shows a jQuery code snippet in violators.php that retrieves and displays violator data from the database. The script initializes when the document is ready and sends an AJAX request using `$.getJSON()` to fetch data from `get.php` with the specified table name `(violators)`. If an error occurs, an alert notifies the user. If data is available, it dynamically generates table rows containing the violator's name, email, phone number, and address, appending them to the `<tbody>` with the `ID violators`. If no data is found, a message is displayed instead. Finally, the `$('#table').DataTable();` function is called to enhance the table with sorting and pagination features.

```
$tables = ['admins', 'enforcers', 'logs', 'violators', 'violationrecords', 'violations'];

if (isset($_GET['table']) && in_array($_GET['table'], $tables)) {
 default:
            $sql = "SELECT * FROM $tableName";
            break;
    }

    $result = $conn->query($sql);

    $data = [];
    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            $data[] = $row;
        }
    }
```

Figure 126 code snippet for Dynamic SQL Query Execution for Multiple Tables – get.php

Figure 126 shows a PHP code snippet in get.php that retrieves violator data from the `violators` table. The script first defines an array of allowed tables, including `admins`, `enforcers`, `logs`, `violators`, `violationrecords`, and `violations`. If a valid table name is provided via a `GET` request, the script constructs a SQL query to select all records from the specified table. The query is executed using `$conn->query($sql)`, and if records are found,

they are fetched and stored in an array. This data is then returned as a response for further processing on the frontend.

Figure 127. BTMO Head/Staff–Violation Records Page

The Violation Records Page provides a detailed overview of violation records, including the violator's name, type of violation, location, date, penalty amount, assigned enforcer, and status. It features an "All" button for quick access to all records, and an "Export PDF" option for report generation. Additionally, an activity log tracks enforcer logins and logouts with timestamps, ensuring accountability. A search function enables quick retrieval of specific records for efficient violation management.

```
<table id="table" class="table table-striped table-bordered">
                        <thead>
                            <tr>
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-id-card-clip"></i>
                                    Ticket No.
                                </th>
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-user"></i>
                                    Name
                                </th>
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-circle-exclamation"></i>
                                    Violation
                                </th>
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-hashtag"></i>
                                    Offense
                                </th>
                                <th class="text-secondary text-center kanit">
                                    <i class="fa-solid fa-location-pin"></i>
                                    Location
```

```
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-calendar-days"></i>
                                        Date
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-file-invoice-dollar"></i>
                                        Penalty amount
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-image"></i>
                                        Evidences
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-circle-question"></i>
                                        Status
                                    </th>
                                    <th class="text-secondary text-center kanit">
                                        <i class="fa-solid fa-circle-check"></i>
                                        Resolve
                                    </th>
                                </tr>
                            </thead>
                            <tbody id="violation-reocrds"></tbody>
                        </table>
```

Figure 128 code snippet for Violation Records Table– violation-records.php

Figure 128 shows the HTML structure for displaying violation records in a tabular format. The table includes columns for Ticket Number, Violator's Name, Violation Type, Offense Count, Location, Date, Penalty Amount, Evidences, Status, and a Resolve Action. The `<tbody>` section with `id="violation-records"` is dynamically populated with data retrieved from the database, ensuring real-time updates. Each row represents a recorded violation, allowing users to view key details and take necessary actions.

```
<div class="modal fade" id="evidenceModal" tabindex="-1" aria-labelledby="evidenceModalLabel"
aria-hidden="true">
        <div class="modal-dialog modal-dialog-centered">
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title kanit" id="evidenceModalLabel">Complete Violation
Record</h5>
                    <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
                </div>
                <div class="modal-body" id="modalBody">
                    <div class="text-center">
                        <div class="spinner-border" role="status">
                            <span class="visually-hidden">Loading...</span>
                        </div>
                    </div>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Close</button>
                </div>
            </div>
        </div>
```

```
    </div>
```

Figure 129 code snippet for Table Evidence Modal Structure– violation-records.php

Figure 129 shows the modal used for displaying complete violation record details, including the violator's name, violation type, and evidences. The modal is triggered dynamically and contains a header with the title "Complete Violation Record," a close button, and a body section (`id="modalBody"`) where the violation details are loaded. A loading spinner is initially displayed while retrieving data. The modal ensures a user-friendly way to review violation records without navigating away from the page.

```
$.getJSON(`../controllers/get.php?table=${tableName}`, function (data) {
                if (data.error) {
                    alert(data.error);
                } else if (data.length > 0) {
                    const violationCounts = {};

                    data.forEach(item => {
                        const violations = item.violation_type.split(',').map(v => v.trim());
                        const violatorId = item.violator_id;

                        violations.forEach(violation => {
                            const key = `${violatorId}_${violation}`;

                            if (!violationCounts[key]) {
                                violationCounts[key] = 0;
                            }
                            violationCounts[key]++;
                        });
                    });

                    data.sort((a, b) => new Date(a.violation_date) - new Date(b.violation_date));

                    let rows = '';
                    let expandedData = [];

                    data.forEach(item => {
                        const violations = item.violation_type.split(',').map(v => v.trim());

                        violations.forEach(violation => {
                            const key = `${item.violator_id}_${violation}`;

                            expandedData.push({
                                ...item,
                                original_violation_type: item.violation_type,
                                violation_type: violation,
                                violationCount: violationCounts[key]
                            });
                        });
                    });

                    expandedData.forEach(item => {
                        const rowClass = item.violationCount >= 3 ? 'table-danger' : '';

                        const statusBadge = item.status === 'pending'
                            ? `<span class="badge text-bg-warning">${item.status}</span>`
                            : `<span class="badge text-bg-success">${item.status}</span>`;
```

```
                        const resolveButton = item.status === 'pending'
                            ? `<button class="btn btn-sm btn-success resolve-btn" data-
id="${item.record_id}">Resolve</button>`
                            : `<span class="badge text-bg-secondary">Resolved</span>`;

                        const evidenceButton = `<button class="btn btn-sm btn-primary evidence-
btn" data-bs-toggle="modal" data-bs-target="#evidenceModal" data-id="${item.record_id}" data-
violator-id="${item.violator_id}" data-violation="${item.violation_type}" data-all-
violations="${item.original_violation_type}"><i class="fa-solid fa-image"></i></button>`;

                        const formattedDate = new
Date(item.violation_date).toLocaleDateString('en-US', {
                            month: 'short',
                            day: 'numeric',
                            year: 'numeric',
                        });

                        rows += `
        <tr class="${rowClass}">
            <td>${item.ticket_no}</td>
            <td>${item.violator_name}</td>
            <td>${item.violation_type}</td>
            <td class="text-center">${getOrdinal(item.violationCount)}</td>
            <td>${item.location}</td>
            <td>${formattedDate}</td>
            <td>${item.penalty_amount}</td>
            <td class="text-center">${evidenceButton}</td>
            <td class="text-center">${statusBadge}</td>
            <td class="text-center">${resolveButton}</td>
        </tr>
        `;
                    });

                    $('#violation-reocrds').html(rows);
                } else {
                    $('#violation-reocrds').html(`<tr><td colspan='10'>No data found</td></tr>`);
                }
```

Figure 130 code snippet for Violation Records Table Generation with Violator Details

Figure 130 presents the JavaScript code responsible for retrieving and displaying the list of violations dynamically. It sends an AJAX request to `get.php` to fetch violation records and processes the data to count multiple offenses per violator. The data is sorted by violation date and expanded to ensure each violation is displayed individually. The script generates table rows, assigning a red background `(table-danger)` if a violator has three or more offenses. It also formats the violation date, displays status badges, and provides action buttons for resolving violations and viewing evidence via a modal. If no records are found, it displays a message indicating an empty table.

```
const violationCounts = {};

                    data.forEach(item => {
                        const violations = item.violation_type.split(',').map(v => v.trim());
                        const violatorId = item.violator_id;
```

```
                violations.forEach(violation => {
                    const key = `${violatorId}_${violation}`;

                    if (!violationCounts[key]) {
                        violationCounts[key] = 0;
                    }
                    violationCounts[key]++;
                });
            });
```

Figure 131 code snippet for Violation Count Aggregation by Violator

Figure 131 illustrates the JavaScript code responsible for counting violations and tracking repeat offenders. It processes violation data by analyzing how many times each violator has committed specific violations. Using an object `(violationCounts)`, it iterates through the dataset, splitting multiple violations and associating them with the violator's ID. Each unique violation-violator pair is stored as a key, incrementing its count upon recurrence. This logic enables the system to track frequent offenders and categorize violations accordingly.

```
$(document).on('click', '.resolve-btn', function () {
                const ticketNo = $(this).closest('tr').find('td:first').text();
                Swal.fire({
                    title: 'Are you sure?',
                    text: "You are about to mark this record as resolved.",
                    icon: 'warning',
                    showCancelButton: true,
                    confirmButtonColor: '#3085d6',
                    cancelButtonColor: '#d33',
                    confirmButtonText: 'Yes, resolve it!'
                }).then((result) => {
                    if (result.isConfirmed) {
                        $.ajax({
                            url: '../controllers/update.php',
                            type: 'POST',
                            data: {
                                id: ticketNo,
                                table: 'violationrecords',
                                action: 'resolve'
                            },
                            success: function (response) {
                                const res = JSON.parse(response);
                                if (res.success) {
                                    Swal.fire({
                                        icon: 'success',
                                        title: 'Success!',
                                        text: "Violation resolved.",
                                        showConfirmButton: true
                                    }).then(() => {
                                        location.reload();
                                    });
                                } else {
                                    Swal.fire({
                                        icon: 'error',
                                        title: 'Error',
                                        text: res.message
                                    });
                                }
```

```
                            },
                            error: function () {
                                Swal.fire({
                                    icon: 'error',
                                    title: 'Error',
                                    text: 'An error occurred while processing your request.'
                                });
                            }
                        });
                    }
                });
            });
```

Figure 132 code snippet for Resolve Violation Record with Confirmation

Figure 132 presents the JavaScript function for updating a violation record's status from "pending" to "resolved." When the resolve button is clicked, it retrieves the ticket number and prompts the user with a confirmation dialog using SweetAlert. If confirmed, an AJAX request is sent to `update.php`, updating the violation status in the database. Upon a successful update, a success message appears, and the page refreshes to reflect the changes. If an error occurs, an error message is displayed, ensuring proper feedback for the user.

```
$(document).on('click', '.evidence-btn', function () {
                    const ticketNo = $(this).closest('tr').find('td:first').text();
                    const violatorId = $(this).data('violator-id');
                    const specificViolation = $(this).data('violation');
                    const allViolations = $(this).data('all-violations');

                    $('#modalBody').html('<div class="text-center"><div class="spinner-border"
role="status"><span class="visually-hidden">Loading...</span></div></div>');
 });
                });
```

Figure 133 code snippet for Display Evidence Modal with Loading Spinner

Figure 133 illustrates the JavaScript function that handles the clicking of the evidence button (image icon) under the "Evidences" column. When clicked, it retrieves the ticket number, violator ID, specific violation, and all violations associated with the violator. It then updates the modal body with a loading spinner to indicate that the evidence data is being fetched. This ensures a smooth user experience while retrieving and displaying the necessary violation details.

```
 case 'violationrecords':
            $sql = "SELECT vr.*, v.name AS violator_name, e.name AS enforcer_name
                    FROM violationrecords vr
                    LEFT JOIN violators v ON vr.violator_id = v.violator_id
                    LEFT JOIN enforcers e ON vr.enforcer_id = e.enforcer_id";

            if ($id) {
                $sql .= " WHERE vr.record_id = " . intval($id);
            } else if ($filter === 'this_week') {
```

```
                $sql .= " WHERE YEARWEEK(vr.violation_date, 1) = YEARWEEK(CURDATE(), 1)";
            } elseif ($filter === 'this_month') {
                $sql .= " WHERE MONTH(vr.violation_date) = MONTH(CURDATE()) AND
YEAR(vr.violation_date) = YEAR(CURDATE())";
            }
            break;
```

Figure 134 code snippet for Fetching Violation Records with Filters–get.php

Figure 134 displays the backend PHP script `(get.php)` responsible for retrieving violation records from the database. The script dynamically constructs an SQL query based on the requested table `(violationrecords)`. It performs `LEFT JOIN` operations to combine data from related tables, including `violators` for violator details and enforcers for `enforcer` details.

```
$stmt = $conn->prepare("UPDATE $table SET status = 'resolved', resolved_date = ?, or_no = ? WHERE
ticket_no = ?");
                if (!$stmt) {
                    throw new Exception("Prepare failed: " . $conn->error);
                }
                $stmt->bind_param('sss', $current_date, $or_no, $id);

                $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' resolved a violation
with OR#: ' . $or_no : 'Unknown user resolved a violation with OR#: ' . $or_no;
                $logAction = 'resolved violation';
```

Figure 135 code snippet for Update Violation Record Status to Resolved–update.php

Figure 135 illustrates the function responsible for updating a violation's status to `resolved`. The function ensures data integrity and security by utilizing prepared statements to prevent SQL injection. When executed, it sets the `status` to `resolved` and records the `resolved_date` using the current timestamp. Additionally, it logs the action with details of the user performing the update. If a session is active, the log captures the user's name; otherwise, it defaults to "Unknown user." This implementation enhances accountability and ensures accurate tracking of resolved violations in the system.

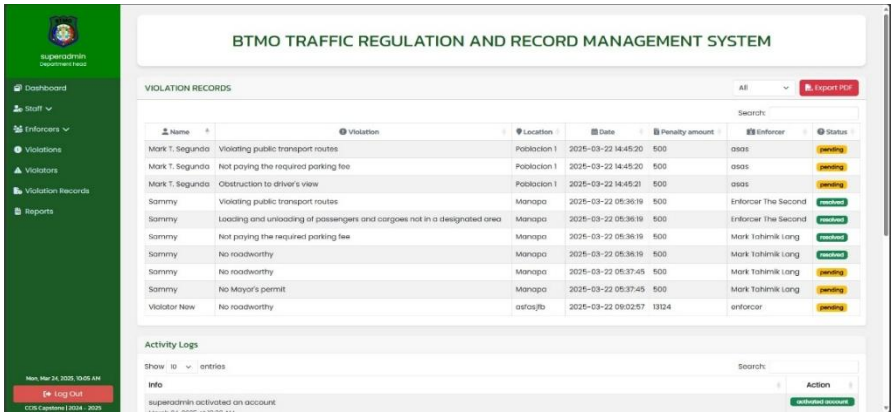


Figure 136. BTMO Head/Staff–Reports Page

The Reports Page provides a detailed and structured overview of recorded traffic violations, offering essential insights into enforcement activities. The report is displayed in a tabular format, showcasing key information such as the violator's name, type of violation, location, date, penalty amount, assigned enforcer, and status (pending or resolved). Additionally, the system includes an export to PDF feature, enabling users to generate a Traffic Violation Report for documentation and analysis. The report also integrates violation statistics, presenting a percentage-based breakdown of different violation types.

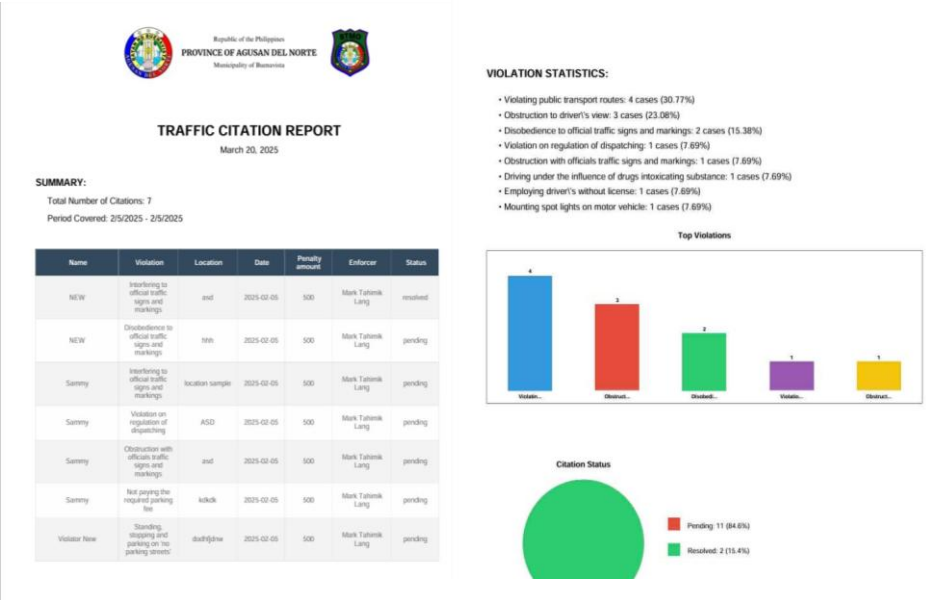


Figure 137. BTMO Head/Staff–Document Pdf Viewer of Reports

The Document PDF Viewer serves as a comprehensive reporting tool for traffic violations, providing a detailed Traffic Citation Report that includes a summary of the total number of citations issued within a specific period. The report is structured in a table format, displaying violator names, violation types, locations, dates, penalty amounts, assigned enforcers, and status updates. Additionally, the system integrates violation statistics, offering a percentage-based tally of each citation type. This feature enhances data analysis, enforcement evaluation, and strategic decision-making, ensuring accurate record-keeping and streamlined traffic regulation management.

```
<div class="d-flex align-items-center">
                            <select id="filterDropdown" class="form-select w-auto me-2">
                                <option value="all">All</option>
                                <option value="this_week">This Week</option>
                                <option value="this_month">This Month</option>
                            </select>
                            <button id="pdfExportBtn" class="btn btn-danger">
                                <i class="fa-solid fa-file-pdf"></i> Export PDF
                            </button>
                        </div>
```

Figure 138 code snippet for Filter and PDF Export UI - reports.php

Figure 138 shows a code snippet in reports.php that provides filtering and PDF export functionality for violation reports. The script includes a dropdown menu (`<select>`) that allows users to filter records based on "All," "This Week," or "This Month." This selection dynamically updates the displayed violation data. Additionally, a PDF export button (`<button>`) enables users to generate and download a report of the filtered data. The button features a Font Awesome PDF icon (`<i class="fa-solid fa-file-pdf"></i>`) for visual clarity.

```
 <table id="table" class="table table-striped table-bordered">
                            <thead>
                                <tr>
                                    <th class="text-secondary text-center kanit"><i class="fa-
solid fa-user"></i> Name
                                    </th>
                                    <th class="text-secondary text-center kanit"><i
                                            class="fa-solid fa-circle-exclamation"></i>
Violation</th>
                                    <th class="text-secondary text-center kanit"><i
                                            class="fa-solid fa-location-pin"></i> Location</th>
                                    <th class="text-secondary text-center kanit"><i
                                            class="fa-solid fa-calendar-days"></i> Date</th>
                                    <th class="text-secondary text-center kanit"><i
```

```
                                        class="fa-solid fa-file-invoice-dollar"></i> Penalty
amount</th>
                                    <th class="text-secondary text-center kanit"><i
                                        class="fa-solid fa-id-card-clip"></i> Enforcer</th>
                                    <th class="text-secondary text-center kanit"><i
                                        class="fa-solid fa-circle-question"></i> Status</th>
                                </tr>
                            </thead>
                            <tbody></tbody>
                        </table>
```

Figure 139 code snippet for Violation Records Table

Figure 139 shows a code snippet in reports.php that defines a table for displaying violation records. The table is structured with a striped and bordered format (`class="table table-striped table-bordered"`) to enhance readability. The table headers include columns for violator name, violation type, location, date, penalty amount, assigned enforcer, and status. Each column is visually represented with Font Awesome icons (`<i class="fa-solid ..."></i>`) for better user experience. The `<tbody>` element is left empty as it is dynamically populated with violation data from the backend.

```
 <div class="card shadow-sm mb-4">
                    <div class="card-header">
                        <h5 class="card-title mb-0 kanit fg-green">
                            Activity Logs
                        </h5>
                    </div>
                    <div class="card-body">
                        <table id="activityLogsTable" class="table table-striped table-bordered"
style="width:100%">
                            <thead>
                                <tr>
                                    <th>Info</th>
                                    <th class="text-center">Action</th>
                                    <th style="display: none;">DateTime</th>
                                </tr>
                            </thead>
                            <tbody></tbody>
                        </table>
                    </div>
                </div>
```

Figure 140 code snippet for Activity Logs Table

Figure 140 shows a code snippet in logs.php that defines a table for displaying user activity logs. The table is enclosed within a card component (`<div class="card shadow-sm mb-4">`) for a structured and visually appealing layout. The card header (`<div class="card-header">`) includes a title labeled "Activity Logs" to indicate the purpose of the table. The table

itself (id="activityLogsTable") is styled with striped and bordered formatting (class="table table-striped table-bordered") for enhanced readability. The table consists of three columns: Info, Action, and DateTime, where the DateTime column is initially hidden (style="display: none;"). The <tbody> element is left empty to be dynamically populated with user activity logs from the backend, ensuring real-time tracking of user actions within the system.

```
function initializeDataTable(data) {
                if (dataTable) {
                    dataTable.destroy();
                }

                dataTable = $('#table').DataTable({
                    data: data,
                    columns: [
                        { data: 'violator_name' },
                        { data: 'violation_type' },
                        { data: 'location' },
                        { data: 'violation_date' },
                        { data: 'penalty_amount' },
                        { data: 'enforcer_name' },
                        {
                            data: 'status',
                            render: function (data) {
                                return data === 'pending'
                                    ? `<span class="badge text-bg-warning">${data}</span>`
                                    : `<span class="badge text-bg-success">${data}</span>`;
                            }
                        }
                    ],
                    paging: false,
                    lengthChange: false,
                    info: false
                });
            }
```

Figure 141 code snippet for DataTable Initialization and Rendering

Figure 141 shows a JavaScript function in violations.js that initializes and configures a DataTable for displaying violation records. The function initializeDataTable(data) first destroys any existing instance of the DataTable to prevent duplication, then creates a new DataTable using $('#table').DataTable({}), where it maps data fields such as violator name, violation type, location, date, penalty amount, and enforcer name to table columns. The status column is customized to display a yellow badge for "pending" violations and a green badge for "resolved" violations. To maintain a streamlined interface, pagination, length change, and info

display are disabled. This script ensures dynamic and efficient rendering of violation records for improved data management.

```
function fetchData(filter) {
                const tableName = 'violationrecords';
                $.getJSON(`../controllers/get.php?table=${tableName}&filter=${filter}`, function
(data) {
                    if (data.error) {
                        alert(data.error);
                        return;
                    }

                    if (data.length > 0) {
                        initializeDataTable(data);
                    } else {
                        initializeDataTable([]);
                        $('#table tbody').html('<tr><td colspan="8" class="text-center">No data
found</td></tr>');
                    }
                }).fail(function (jqXHR, textStatus, errorThrown) {
                    console.error("Error fetching data:", textStatus, errorThrown);
                    alert("Failed to fetch violation records");
                });
            }

            fetchData('all');
```

Figure 142 code snippet for Fetch and Display Violation Records

Figure 142 shows a JavaScript function in violations.js that retrieves violation records from get.php based on the selected filter, with the default set to 'all'. The function `fetchData(filter)` sends an AJAX request using `$.getJSON()` to fetch records from the `violationrecords` table, appending the filter parameter to the request URL. If an error occurs in the response, an alert notifies the user. If records are found, the `initializeDataTable(data)` function is called to display them; otherwise, an empty table with a "No data found" message is shown. Additionally, the function includes an error-handling mechanism using `.fail()` to log errors and alert the user in case of a request failure. The script ensures dynamic retrieval and filtering of violation records, enhancing data accessibility.

```
$('#filterDropdown').on('change', function () {
                const selectedFilter = $(this).val();
                fetchData(selectedFilter);
            });
```

Figure 143 code snippet for Filter Dropdown Change Handler

Figure 143 shows a JavaScript snippet that updates the violation records table when the filter dropdown value changes. The `$('#filterDropdown').on('change', function () { ... })` event listener detects user selection from the dropdown menu. It retrieves the selected filter value using `$(this).val()` and then calls the `fetchData(selectedFilter)` function to retrieve and display the filtered violation records. This functionality allows users to dynamically update the table based on selected time filters, improving data accessibility and user interaction.

```
document.getElementById('pdfExportBtn').addEventListener('click', function () {
            const { jsPDF } = window.jspdf;
            const doc = new jsPDF({
                orientation: 'portrait',
                unit: 'mm',
                format: 'a4'
            });
```

Figure 144 code snippet for PDF Export Button Click Handler

Figure 144 shows a JavaScript snippet that implements the export-to-PDF functionality for violation records. When the `#pdfExportBtn` button is clicked, it initializes a new jsPDF instance with an A4 portrait layout. The script sets up page formatting, including margins and title placement. It then generates a structured table displaying violation records and appends violation statistics at the bottom, showing the percentage of each violation type. Finally, the formatted document is saved as a downloadable PDF, enabling users to generate reports efficiently.

```
function fetchLogs() {
                $.ajax({
                    url: '../controllers/get.php',
                    type: 'GET',
                    data: { table: 'logs' },
                    dataType: 'json',
                    success: function (data) {
                        if (data.error) {
                            console.error('Error fetching logs:', data.error);
                        } else {
                            renderLogs(data);
                        }
                    },
                    error: function (xhr, status, error) {
                        console.error('AJAX Error:', status, error);
                    }
                });
            }
```

Figure 145 code snippet for Fetch Logs via AJAX

Figure 145 shows a JavaScript function, `fetchLogs`, that retrieves data for the activity logs table using an AJAX request. The function sends a GET request to `get.php` with the parameter `table=logs`, expecting a JSON response. If the request is successful and no errors are returned, the retrieved data is passed to the `renderLogs` function for display. In case of an error, it logs the issue to the console, ensuring debugging information is available for troubleshooting.

```
function renderLogs(logs) {
                const logsTable = $('#activityLogsTable').DataTable();
                logsTable.clear();

                logs.sort((a, b) => new Date(b.date_time) - new Date(a.date_time));

                if (logs.length > 0) {
                    logs.forEach(log => {
                        const formattedDate = formatDateTime(log.date_time);
                        const badgeClass = getBadgeClass(log.action);
                        logsTable.row.add([
                            `<span>${log.info}<br><small class="text-
muted">${formattedDate}</small></span>`,
                            `<span class="badge bg-${badgeClass} d-flex justify-content-
center">${log.action}</span>`,
                            log.date_time
                        ]).draw(false);
                    });
                }
            }
```

Figure 146 code snippet for Render Logs to DataTable

Figure 146 shows a JavaScript function, `renderLogs`, responsible for displaying activity logs in the `activityLogsTable` using DataTables. The function first clears existing table data before sorting the logs in descending order based on the `date_time` field. If logs are available, it iterates through each log, formats the timestamp using `formatDateTime`, determines the badge styling for the `action` type using `getBadgeClass`, and then dynamically adds rows to the table. This approach ensures that logs are displayed in a structured and visually distinct manner, making it easier to track user activities.

```
function getBadgeClass(action) {
                switch (action.toLowerCase()) {
                    case 'logged in':
                        return 'success';
                    case 'added account':
                        return 'success';
                    case 'added violation':
                        return 'success';
```

```
                case 'resolved violation':
                    return 'success';
                case 'activated account':
                    return 'success';
                case 'updated violation':
                    return 'warning';
                case 'deleted violation':
                    return 'danger';
                case 'logged out':
                    return 'secondary';
                case 'disabled account':
                    return 'secondary';
                default:
                    return 'primary';
            }
        }
```

Figure 147 code snippet for Get Badge Class for Log Action

Figure 147 shows the JavaScript function `getBadgeClass`, which assigns an appropriate Bootstrap class based on the action type. The function converts the action string to lowercase and uses a `switch` statement to match different cases. Actions such as "logged in," "added account," "added violation," "resolved violation," and "activated account" return the `success` class, while "updated violation" returns `warning`, "deleted violation" returns `danger`, and "logged out" or "disabled account" return `secondary`. If the action does not match any predefined cases, it defaults to the `primary` class. This ensures consistent styling for log entries based on their significance.

```
function formatDateTime(dateTime) {
            const date = new Date(dateTime);
            return date.toLocaleString('en-US', {
                month: 'long',
                day: '2-digit',
                year: 'numeric',
                hour: '2-digit',
                minute: '2-digit',
                hour12: true
            });
        }
```

Figure 148 code snippet for Format DateTime for Logs

Figure 148 shows the JavaScript function `formatDateTime`, which formats a given date and time string for display in the activity logs. The function converts the input `dateTime` into a JavaScript `Date` object and uses `toLocaleString` to format it in a user-friendly manner. The

output follows the "Month Day, Year, HH:MM AM/PM" format (e.g., March 23, 2025, 10:30 PM). This ensures consistency and readability when displaying timestamps in the activity logs.

```
$('#activityLogsTable').DataTable({
                columns: [
                    { title: "Info" },
                    { title: "Action" },
                    { title: "DateTime" }
                ],
                order: [[2, 'desc']],
                columnDefs: [
                    { targets: 2, visible: false },
                    { targets: 0, width: '90%' },
                    { targets: 1, width: '10%' }
                ]
            });

            fetchLogs();
```

Figure 149 code snippet for Initialize Activity Logs Table

Figure 149 shows the JavaScript code that configures the Activity Logs Table using the DataTables plugin. The table is initialized with three columns: "Info," "Action," and "DateTime", where the "DateTime" column is hidden for sorting purposes. The table rows are ordered in descending order based on the "DateTime" column, ensuring the most recent logs appear first. Additionally, column widths are adjusted to optimize layout, with the "Info" column occupying 90% and the "Action" column 10% of the table width. Once the table configuration is set, the `fetchLogs()` function is called to retrieve and populate log data dynamically.

```
$tables = ['admins', 'enforcers', 'logs', 'violators', 'violationrecords', 'violations'];

if (isset($_GET['table']) && in_array($_GET['table'], $tables)) {
    $tableName = $_GET['table'];
    $filter = $_GET['filter'] ?? 'all';
```

Figure 150 code snippet for Table Selection and Filter Handling – get.php

Figure 150 shows a PHP code snippet in get.php that validates whether the requested table is allowed by checking it against a predefined list of permitted tables: `admins`, `enforcers`, `logs`, `violators`, `violationrecords`, and `violations`. If a valid table name is received via a GET request, it is assigned to the `$tableName` variable. Additionally, the script retrieves an optional filter parameter, `$filter`, which defaults to `'all'` if no filter is

provided. This ensures that only authorized tables can be queried, enhancing security and data integrity.

```
case 'violationrecords':
            $sql = "SELECT vr.*, v.name AS violator_name, e.name AS enforcer_name
                    FROM violationrecords vr
                    LEFT JOIN violators v ON vr.violator_id = v.violator_id
                    LEFT JOIN enforcers e ON vr.enforcer_id = e.enforcer_id";

            if ($id) {
                $sql .= " WHERE vr.record_id = " . intval($id);
            } else if ($filter === 'this_week') {
                $sql .= " WHERE YEARWEEK(vr.violation_date, 1) = YEARWEEK(CURDATE(), 1)";
            } elseif ($filter === 'this_month') {
                $sql .= " WHERE MONTH(vr.violation_date) = MONTH(CURDATE()) AND
YEAR(vr.violation_date) = YEAR(CURDATE())";
            }
            break;
```

Figure 151 code snippet for SQL Query for Violation Records with Filters – get.php

Figure 151 shows a PHP query in get.php that retrieves violation records from the `violationrecords` table while joining data from the `violators` and `enforcers` tables to include the violator's and enforcer's names. If a specific record ID `($id)` is provided, the query fetches only the corresponding record. Otherwise, it applies a filter based on the selected time range: "this_week" retrieves violations from the current week using `YEARWEEK()`, while "this_month" fetches records from the current month using `MONTH()` and `YEAR()`. This query ensures dynamic data retrieval based on user-defined filters.

4.2 TESTING PROCESS

The testing process outlines the various procedures conducted to assess the system's accuracy, functionality, and overall performance. To evaluate its effectiveness, efficiency, and reliability, the researchers utilized a survey instrument based on the System Usability, Functionality, and Efficiency Scale test. This instrument was adapted from the ISO 25010 Software Product Quality Standards. Respondents rated the system using a four-point scale, where 4 represented the highest rating and 1 the lowest.

Data were analyzed and interpreted based on the following parameters:

Table 3

Functional Suitability

Adjectival Rating	Scale Range (Mean)	Verbal Interpretation
4	3.50 – 4.0	Very Functional
3	2.50 – 3.49	Functional
2	1.50 – 2.49	Moderately Functional
1	1.0 – 1.49	Poor Functional

Table 3 presents the criteria used to evaluate a product or system's functional suitability according to the ISO survey.

Table 4

Performance Efficiency

Adjectival Rating	Scale Range (Mean)	Verbal Interpretation
4	3.50 – 4.0	Very Efficient
3	2.50 – 3.49	Efficient
2	1.50 – 2.49	Moderately Efficient
1	1.0 – 1.49	Poor Efficient

Table 4 outlines the scoring parameters used to assess the system's overall performance efficiency in executing its functions and tasks.

Table 5

Usability

Adjectival Rating	Scale Range (Mean)	Verbal Interpretation
4	3.50 – 4.0	Very Usable
3	2.50 – 3.49	Usable
2	1.50 – 2.49	Moderately Usable
1	1.0 – 1.49	Poor Usable

Table 5 presents the scoring parameters used to evaluate how effectively, efficiently, and satisfactorily a system or product enables specific users to achieve their objectives within a given context of use.

Table 6

Respondent's Distribution

Buenavista Traffic Management Office (BTMO)		
Position	**N (Population)**	**Percentage (%)**
BTMO Head	1	12.5%
Staff	1	12.5%
Enforcer	6	75%
Total	**8**	**100**

The respondents for the study were the Head, Staff, and an Enforcer from the Buenavista Traffic Management Office (BTMO). The researchers used purposive sampling to choose the participants for the survey. Information was collected directly from the respondents through a

survey form, which they filled out and returned. The data was then studied and used to help develop the project.

To make sure the results were accurate, the researchers also held a group discussion and a project demonstration. During this, the participants tested and gave feedback on the system.

Descriptive Statistics Result

Table 7

Functional Suitability

A.	**Functional Suitability**	**MEAN**	**VERBAL INTERPRETATION**
1.	Functional completeness - Degree to which the set of functions covers all the specified tasks and user objectives	3.75	VF
2.	Functional correctness - Degree to which a product or system provides the correct results with the needed degree of precision.	3.63	VF
3.	Functional appropriateness - Degree to which the functions facilitate the accomplishment of specified tasks and objectives.	3.88	VF
	Weighted Mean	**3.75**	**VF**

Legend:

VF – Very Functional (3.50 – 4.0) **MF** – Moderately Functional (1.50 – 2.49)

F – Functional (2.50 – 3.49) **PF** – Poorly Functional (1.0 – 1.49)

Table 7 presents the mean distribution and verbal interpretation for Functional Suitability, showing a total weighted mean of 3.75, which is described as "Very Functional". This indicates that the system effectively meets user needs in terms of functionality.

Among the criteria, Functional appropriateness received the highest score of 3.88, suggesting that the system efficiently helps users complete their tasks and objectives. Functional completeness and Functional correctness also scored high, with means of 3.75 and 3.63, respectively, further supporting the system's reliability in delivering accurate results and covering all specified tasks. Overall, these results demonstrate that the system is user-friendly, accessible, and effective in generating data and reports, making it highly functional as perceived by the users.

Table 8

Performance Efficiency

B.	Performance Efficiency	MEAN	VERBAL INTERPRETATION
1.	Time behaviour - Degree to which the response and processing times and throughput rates of a system, when performing its functions, meet requirements.	3.75	VE
2.	Resource utilization - Degree to which the amounts and types of resources used by a system, when performing its functions, meet requirements.	4.0	VE

3.	Capacity - Degree to which the maximum limits of a product or system parameter meet requirements.	3.88	VE
	Weighted Mean	**3.88**	**VE**

Legend:

VE – Very Efficient (3.50 – 4.0) **ME** – Moderately Efficient (1.50 – 2.49)

E – Efficient (2.50 – 3.49) **PE** – Poorly Efficient (1.0 – 1.49)

Table 8 presents the mean distribution and verbal interpretation for Performance Efficiency, with a total weighted mean of 3.88, described as "Very Efficient". This indicates that the system effectively meets performance expectations in terms of response time, resource usage, and capacity. Among the criteria, Resource utilization received the highest score of 4, showing the system's ability to manage resources efficiently. Meanwhile, Capacity scored 3.88, and Time behaviour scored 3.75, both reflecting the system's effectiveness in handling workloads and meeting processing speed requirements. Overall, the results confirm that the system performs efficiently and effectively as perceived by the users.

Table 9

Usability

C.	**Usability**	**MEAN**	**VERBAL INTERPRETATION**
1.	Appropriateness recognizability - Degree to which users can recognize whether a system is appropriate for their needs.	3.88	VU

2.	Learnability - Degree to which a system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.	3.75	VU
3.	Operability - Degree to which a system has attributes that make it easy to operate and control.	3.88	VU
4.	User error protection - Degree to which a system protects users against making errors.	3.75	VU
5.	User interface aesthetics - Degree to which a user interface enables pleasing and satisfying interaction for the user.	3.88	VU
6.	Accessibility - Degree to which a system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.	3.75	VU
	Weighted Mean	**3.81**	**VU**

Legend:

VU – Very Usable (3.50 – 4.0) **MU** – Moderately Usable (1.50 – 2.49)

U – Usable (2.50 – 3.49)	**PU** – Poorly Usable (1.0 – 1.49)

Table 9 presents the mean distribution and verbal interpretation for Usability, with a total weighted mean of 3.81, described as "Very Usable". This indicates that the system effectively meets usability standards, providing a positive user experience. Several criteria, including Appropriateness recognizability, Operability, and User interface aesthetics, received the highest score of 3.88, highlighting the system's ease of use, pleasing design, and clear purpose for users. Meanwhile, Learnability, User error protection, and Accessibility each scored 3.75, further supporting the system's effectiveness in helping users learn, avoid mistakes, and accommodate individuals with diverse needs. Overall, the results confirm that the system is highly usable and user-friendly.

Table 10

Summary Table of the Over-all Mean and Grand Distribution of the Acceptability Level

Acceptability Level of the System in terms of:	**Over-all Mean**	**Rating**
Functional Suitability	3.75	SA
Performance Efficiency	3.88	SA
Usability	3.81	SA
Grand Mean	**3.81**	**SA**

Legend:

SA – Strongly Acceptable (3.50 – 4.0)	**U** – Unacceptable (1.50 – 2.49)
A – Acceptable (2.50 – 3.49)	**SU** – Strongly Unacceptable (1.0 – 1.49)

Table 10 provides a summary of the overall acceptability of the system, combining the results from functional suitability, performance efficiency, and usability. The system has a generally favorable acceptance, with all evaluated aspects rated as "Strongly Acceptable". Functional Suitability scored 3.75, Performance Efficiency scored 3.88, and Usability scored 3.81, all reflecting a high level of user satisfaction. The Grand Mean of 3.81 further confirms that the system effectively meets user expectations in terms of functionality, performance, and usability. These results indicate that the system is well-received, though further improvements may enhance user experience and overall efficiency even more.

CHAPTER V

SUMMARY, CONCLUSION, AND RECOMMENDATION

This chapter outlines the key findings of the research, presents conclusions derived from the results, and offers recommendations for future improvements or further studies. It highlights the project's overall success, considers its impact, and provides practical suggestions for enhancing the system and expanding its scope of application.

5.1. Summary of Findings

Based on the testing process conducted, the Buenavista Traffic Management Office (BTMO) Traffic Regulation and Record Management System was evaluated for its accuracy, functionality, and overall performance using a survey instrument adapted from the ISO 25010 Software Product Quality Standards. The evaluation focused on aspects such as functional suitability, performance efficiency, and usability.

The results revealed that the system is "Very Functional" with a mean score of 3.75, indicating that it effectively meets the specific tasks and objectives of managing and regulating traffic-related records. The system excelled in functional appropriateness, ensuring that it facilitates the accomplishment of specified tasks and objectives, receiving the highest score of 3.88.

In terms of performance efficiency, the system was rated as "Very Efficient," with a mean score of 3.88. It performed exceptionally in resource utilization, scoring 4.0, demonstrating the system's capability to efficiently manage resources while handling operational demands. However, there is room for improvement in enhancing time behavior and capacity management, particularly during peak processing times.

The usability of the system was rated as "Very Usable," with a mean score of 3.81, performing best in operability, appropriateness recognizability, and user interface aesthetics, all scoring 3.88. This suggests that the system is intuitive and accessible to users, ensuring ease of navigation and interaction. Though the system is user-friendly, improvements in learnability and accessibility may further enhance the user experience.

Overall, the BTMO Traffic Regulation and Record Management System achieved a "Strongly Acceptable" rating with a mean score of 3.81, with the highest rating in performance efficiency. This suggests that the system aligns well with the needs of its users, providing an effective tool for traffic regulation, record management, and enforcement processes.

5.2. Conclusion

In conclusion, the system demonstrates strong functional suitability, excellent performance efficiency, and high usability, making it a reliable tool for the Buenavista Traffic Management Office. All project objectives were successfully achieved, as the system effectively fulfilled its intended purpose by ensuring accurate data recording and efficient management of traffic-related activities. While opportunities remain to further improve certain aspects, such as enhancing its capacity to manage larger data volumes and refining usability features like learnability and operability, the system's overall performance and accessibility confirm its potential to significantly streamline BTMO operations and enhance traffic management efficiency.

5.3. Recommendation/s

To further enhance the system's impact, it is recommended that the BTMO allocate sufficient funding to support ongoing development and maintenance, particularly given that the application requires a stable internet connection to function optimally. Funding can also support user training programs tailored to traffic enforcers who may struggle with technology, have

limited experience using gadgets, or face challenges such as impaired vision. Ensuring that the system is inclusive and accessible to all personnel will maximize its effectiveness and adoption.

For future researchers, the integration of offline functionality is strongly advised to ensure uninterrupted system access even in areas with unstable internet connectivity. Implementing a local caching mechanism would allow data to be stored temporarily and synchronized automatically once connectivity is restored, improving reliability during field operations. Additionally, developing a lightweight version optimized for low-bandwidth environments will enhance accessibility, especially in remote or underserved regions.

To optimize performance, refining database queries and minimizing unnecessary data loads is essential for maintaining responsiveness during peak usage. Adaptive design improvements—such as auto-save functionality and asynchronous data processing—will help prevent data loss due to disconnection and improve overall user experience.

Lastly, creating a dedicated mobile application for iOS is recommended to extend full system functionality to Apple users. Including features such as push notifications and real-time synchronization will promote user engagement and increase operational efficiency across departments.

By implementing these enhancements and recommendations, the Traffic Regulation and Record Management System (TRRMS) can evolve into a more robust, efficient, and inclusive platform capable of addressing diverse user needs and maintaining reliable service across various operational contexts.

REFERENCES

[1] O. L. Allen, "Traffic Management Centers: Challenges, Best Practices, and Future Plans Contract # DTRT12GUTC12 with USDOT Office of the Assistant Secretary for Research and Technology (OST-R) Final Report National Center for Transportation Systems Productivity and Management," 2014. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/36968

[2] M. E. Matlala, T. R. Ncube, and S. Parbanath, "The state of digital records preservation in South Africa's public sector in the 21st century: a literature review," Jun. 01, 2022, Emerald Group Holdings Ltd. [Online]. Available: https://openscholar.dut.ac.za/server/api/core/bitstreams/4ff6867a-5bfc-4f6c-8c8b-48c822edb90b/content

[3] Abduraouf. B. Z. Alshetwi, R. Atiq Abdullah O.K. Rahmat, M. Nazri Borhan, S. Ismael Albrka Ali, H. Imhimmed Mohamed Irtema, and A. Y. Y. Alfakhria, "Web-Based Expert System for Optimizing of Traffic Road in Developing Countries," International Journal of Engineering & Technology, vol. 7, no. 2.29, p. 876, May 2018. [Online]. Available: https://www.researchgate.net/profile/Abduraoufbz-Alshetwi/publication/326727672_Web-Based_Expert_System_for_Optimizing_of_Traffic_Road_in_Developing_Countries/links/5bcd425ba6fdcc03c79accca/Web-Based-Expert-System-for-Optimizing-of-Traffic-Road-in-Developing-Countries.pdf

[4] S. Saha, "AN SMS AND WEB-BASED TRAFFIC CASE MANAGEMENT SYSTEM IN BANGLADESH," 2006. [Online]. Available: https://core.ac.uk/download/pdf/61800147.pdf

[5] M. Kerimov, A. Marusin, A. Marusin, and I. Danilov, "Methodological aspects of building mathematical model to evaluate efficiency of automated vehicle traffic control systems," in Transportation Research Procedia, Elsevier B.V., 2020, pp. 253–261. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352146520307778?via%3Dihub

[6] O. S. Ufuoma, “DESIGN AND IMPLEMENTATION OF A WEB BASED CRIME RECORD INFORMATION SYSTEM A PROJECT PRESENTED TO.” [Online]. Available: https://www.projects.calebuniversity.edu.ng/caleb_uploads/2024/04/12-Onafete-shadrachs-project.pdf

[7] M. Kerimov, S. Evtiukov, and A. Marusin, “Model of multi-level system managing automated traffic enforcement facilities recording traffic violations,” in Transportation Research Procedia, Elsevier B.V., 2020, pp. 242–252. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352146520307766?via%3Dihub

[8] V. Kumar and R. Konduru, “ADVANCED TRAFFIC CONTROL AND VIOLATION SYSTEM WITH WEB SERVER,” 2023. [Online]. Available: https://www.ijnrd.org/papers/IJNRD2303306.pdf

[9] A. M. Mimbala, N. U. Macre, N. C. Maunga, and A. C. Acala, “WEB-BASED VEHICLE MONITORING SYSTEM OF MINDANAO STATE UNIVERSITY LANAO DEL NORTE AGRICULTURAL COLLEGE PSYCHOLOGY AND EDUCATION: A MULTIDISCIPLINARY JOURNAL Web-Based Vehicle Monitoring System of Mindanao State University Lanao Del Norte Agricultural College,” Psych Educ, vol. 2024, no. 7, pp. 2024–2105. [Online]. Available: https://scimatic.org/storage/journals/11/pdfs/3301.pdf

[10] S. Limsoonthrakul et al., “Design and implementation of a highly scalable, low-cost distributed traffic violation enforcement system in Phuket, Thailand,” Sustainability (Switzerland), vol. 13, no. 3, pp. 1–23, Feb. 2021. [Online]. Available: https://www.mdpi.com/2071-1050/13/3/1210

[11] A. A. Kayode and A. O. Alabi, “Design and Implementation of a Simplified CodeIgniter Framework for Commercial Vehicles Ticket Reservation System,” Asian Journal of Research in Computer Science, pp. 1–12, Feb. 2021. [Online]. Available:

https://www.researchgate.net/publication/349651856_Design_and_Implementation_of_a_Simplified_CodeIgniter_Framework_for_Commercial_Vehicles_Ticket_Reservation_System

[12] T. G. Bhathiya, "Decision Support System for Road Traffic Violations," 2018. [Online]. Available: http://dl.lib.mrt.ac.lk/handle/123/15944

[13] D. V. Aladin, O. O. Varlamov, D. A. Chuvikov, V. M. Chernenkiy, E. A. Smelkova, and A. V. Baldin, "Logic-based artificial intelligence in systems for monitoring the enforcing traffic regulations," in IOP Conference Series: Materials Science and Engineering, Institute of Physics Publishing, Jun. 2019. [Online]. Available: https://iopscience.iop.org/article/10.1088/1757-899X/534/1/012025/pdf

[14] R. Kaffardeen Mohammed, "International Journal of Advanced Research and Publications Traffic Squad-Smart Traffic Violation Detection System". [Online]. Available: https://www.ijarp.org/published-research-papers/jun2023/Traffic-Squad-Smart-Traffic-Violation-Detection-System.pdf

[15] R. S. Dewanto, "Citation : Rezkhy Satya Dewanto, Applying Information Technology In Realizing An Orderly Traffic Community (A Case Study On Electronic Traffic Law Enforcement Program Conducted By Traffic Directorate Of Jakarta Metropolitan Police Region In," Management Technology and Security International Journal, pp. 243–259, 2019. [Online]. Available: https://www.ctrs-stik-mtsij.ac.id/wp-content/uploads/2021/08/APPLYING-INFORMATION-TECHNOLOGY-IN-REALIZING-AN.pdf

[16] E. Balkovich, D. Prosnitz, S. C. Isley, A. Boustead, and B. Triezenberg, "Helping Law Enforcement Use Data from Mobile Applications: A Guide to the Prototype Mobile Information and Knowledge Ecosystem (MIKE) Tool," 2017. [Online]. Available: https://www.rand.org/content/dam/rand/pubs/research_reports/RR1400/RR1482/RAND_RR1482.pdf

[17] M. D. Foroutaghe, A. M. Moghaddam, and V. Fakoor, "Impact of law enforcement and increased traffic fines policy on road traffic fatality, injuries and offenses in Iran: Interrupted time series analysis," PLoS One, vol. 15, no. 4, Apr. 2020. [Online]. Available: https://doi.org/10.1371/journal.pone.0231182

[18] S. E. and M. Iksan, "The Benefits of the E-Traffic Ticketing (E-Tilang) System in the Settlement of Traffic Violation in Indonesia," 2019. [Online]. Available: https://doi.org/10.2991/icils-19.2019.22

[19] S. Macan and S. Paunović, "Use of video surveillance systems for detecting seat belt usage, mobile device usage, or vehicle registration-penalty or prevention". [Online]. Available: https://tttp-au.com/wp-content/uploads/2024/06/TTTP-Vol-9_No-1_2024-WEB.pdf#page=22

[20] M. T. Marie Mariano and T. Domingo Palaoag, "MATIKETAN Utilization of IT Infrastructure to Support the Development of Mobile Applications for Road Traffic Violation Ticketing," 2024. [Online]. Available: https://journal.esrgroups.org/jes/article/view/4472/3301

[21] E. K. Rahmadany and F. Windiyastuti, "Assessment of Electronic Ticket Law (ETLE) Against Four-Wheeled Road Users in the Area of the Traffic Directorate of Polda Metro Jaya." [Online]. Available: https://www.legal.isha.or.id/index.php/legal/article/view/237

[22] M. Lutfi and M. Takdir, "IMPLEMENTATION OF ELECTRONIC TICKET SANCTIONS FOR TRAFFIC VIOLATORS IN BONE DISTRICT," 2024. [Online]. Available: https://ojs.transpublika.com/index.php/POLRI/article/view/926

[23] S. E. Wahyuningsih and M. Iksan, "The Benefits of the E-Traffic Ticketing (E-Tilang) System in the Settlement of Traffic Violation in Indonesia," 2019. [Online]. Available: https://www.atlantis-press.com/proceedings/icils-19/125922712

[24] M. Kerimov, S. Evtiukov, and A. Marusin, "Model of multi-level system managing automated traffic enforcement facilities recording traffic violations," in Transportation Research Procedia, Elsevier B.V., 2020, pp. 242–252. [Online]. Available: https://www.researchgate.net/publication/346621380_Model_of_multi-level_system_managing_automated_traffic_enforcement_facilities_recording_traffic_violations

[25] E. Prasetyo, G. Damaraji, and S. Kusumawardhani, "A Review of The Challenges of Paperless Concept in the Society 5.0". [Online]. Available: https://ojs.uajy.ac.id/index.php/IJIEEM/article/view/3755

[26] X. Chen, M. Despeisse, and B. Johansson, "Environmental sustainability of digitalization in manufacturing: A review," Dec. 02, 2020, MDPI. [Online]. Available: https://www.researchgate.net/publication/347519070_Environmental_Sustainability_of_Digitalization_in_Manufacturing_A_Review

[27] M. D. Briscoe, "The paperless office twenty years later: Still a myth?," Sustainability: Science, Practice, and Policy, vol. 18, no. 1, pp. 837–845, 2022. [Online]. Available: https://www.tandfonline.com/doi/pdf/10.1080/15487733.2022.2146370

[28] S. Hussein and _____ A.-Q., "The Paperless Organization Improved processes and reduction in paper usage through wider use of electronic documents and tablet computers," 2012. [Online]. Available: https://researchcommons.waikato.ac.nz/server/api/core/bitstreams/3b5be595-acb8-4911-8d44-92a58e5e346a/content

[29] "Proceedings of the 14 th International Conference on Sustainable Built Environment (ICSBE) 2023 Vision." [Online]. Available: https://www.researchgate.net/profile/Nuskiya-Hassan/publication/379661413_SPATIO-TEMPORAL_ANALYSIS_OF_MODIS-

BASED_LAND_SURFACE_TEMPERATURE_USING_GOOGLE_EARTH_ENGINE_GEE_IN_AMPARA_DISTRICT/links/6613ff1666ba7e2359b6695d/SPATIO-TEMPORAL-ANALYSIS-OF-MODIS-BASED-LAND-SURFACE-TEMPERATURE-USING-GOOGLE-EARTH-ENGINE-GEE-IN-AMPARA-DISTRICT.pdf

[30] S. Sunuwar, "ROLE OF DOCUMENT MANAGEMENT SYSTEM IN PAPERLESS OFFICE AT ONE HEART WORLDWIDE, KATHMANDU, NEPAL". [Online]. Available: https://www.researchgate.net/profile/Shreejana-Sunuwar/publication/382650220_ROLE_OF_DOCUMENT_MANAGEMENT_SYSTEM_IN_PAPERLESS_OFFICE_AT_ONE_HEART_WORLDWIDE_KATHMANDU_NEPAL/links/66a7c6b34433ad480e845ec7/ROLE-OF-DOCUMENT-MANAGEMENT-SYSTEM-IN-PAPERLESS-OFFICE-AT-ONE-HEART-WORLDWIDE-KATHMANDU-NEPAL.pdf

APPENDICES

A. System Sources Code

MOBILE APPLICATION

add_ticket.php

```php
<?php
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: POST');
header('Access-Control-Allow-Headers: Access-Control-Allow-Headers, Content-Type, Access-Control-Allow-Methods, Authorization, X-Requested-With');

include '../conn.php';

date_default_timezone_set('Asia/Manila');

Pieces: Comment | Pieces: Explain
function countViolations($conn, $violatorId, $violationType)
{
    $sql = "SELECT COUNT(*) as count FROM violationrecords WHERE violator_id = ? AND violation_type = ?";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("is", $violatorId, $violationType);
    $stmt->execute();
    $result = $stmt->get_result();
    $row = $result->fetch_assoc();
    return $row['count'];
}

$data = json_decode(file_get_contents("php://input"), true);

$requiredFields = [
    'violator_id',
    'enforcer_id',
    'violation_type',
    'location',
    'penalty_amount',
    'vehicle_owner',
    'registration_no',
    'plate_no',
    'vehicle_type',
    'body_no',
    'ticket_no'  // Now required as it's generated separately
];
```

```php
$missingFields = [];
foreach ($requiredFields as $field) {
    if (!isset($data[$field])) {
        $missingFields[] = $field;
    }
}

if (!empty($missingFields)) {
    http_response_code(400);
    echo json_encode([
        'status' => 'error',
        'message' => 'Missing required fields: ' . implode(', ', $missingFields)
    ]);
    exit();
}

// Check if the violator has already reached the 3rd offense
$violationCount = countViolations($conn, $data['violator_id'], $data['violation_type']);
if ($violationCount >= 3) {
    http_response_code(400);
    echo json_encode([
        'status' => 'error',
        'message' => 'This violator has already reached the 3rd offense for this violation type.'
    ]);
    exit();
}

$ticketNo = $data['ticket_no']; // Use provided ticket number
$violationDate = date('Y-m-d H:i:s');

$stmt = $conn->prepare("INSERT INTO violationrecords
    (ticket_no, violator_id, enforcer_id, violation_date, violation_type, location, penalty_amount, vehicle_owner, registration_no, plate_no, vehic
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, 'pending')");

$stmt->bind_param(
    "siisssdsssss",
    $ticketNo,
```

```
    $data['violator_id'],
    $data['enforcer_id'],
    $violationDate,
    $data['violation_type'],
    $data['location'],
    $data['penalty_amount'],
    $data['vehicle_owner'],
    $data['registration_no'],
    $data['plate_no'],
    $data['vehicle_type'],
    $data['body_no']
);

try {
    $result = $stmt->execute();

    if ($result) {
        http_response_code(201);
        echo json_encode([
            'status' => 'success',
            'message' => 'Violation ticket added successfully',
            'ticket_id' => $conn->insert_id,
            'ticket_no' => $ticketNo
        ]);
    } else {
        http_response_code(500);
        echo json_encode([
            'status' => 'error',
            'message' => 'Failed to add violation ticket: ' . $stmt->error
        ]);
    }
} catch (Exception $e) {
    http_response_code(500);
    echo json_encode([
        'status' => 'error',
        'message' => 'Exception occurred: ' . $e->getMessage()
    ]);
}

$stmt->close();
$conn->close();
?>
```

add.php

```php
<?php
include '../conn.php';
date_default_timezone_set("Asia/Manila");

$name = $_POST['name'] ?? '';
$email = $_POST['email'] ?? '';
$phone = $_POST['phone'] ?? '';
$address = $_POST['address'] ?? '';
$date = date('Y-m-d');

$sql = "INSERT INTO violators (name, email, phone, address, date)
        VALUES (?, ?, ?, ?, ?)";

$stmt = $conn->prepare($sql);
$stmt->bind_param("sssss", $name, $email, $phone, $address, $date);

if ($stmt->execute()) {
    echo json_encode(["status" => "success", "message" => "Violator added successfully"]);
} else {
    echo json_encode(["status" => "error", "message" => "Error: " . $stmt->error]);
}

$stmt->close();
$conn->close();
?>
```

fech_violation_records.php

```php
<?php
header('Content-Type: application/json');

include '../conn.php';

// Fetch all violation records
$sql = "SELECT * FROM violationrecords ORDER BY violation_date";
$result = $conn->query($sql);

$violationRecords = array();

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        $violationRecords[] = $row;
    }
}

// Fetch all violators
$sql = "SELECT name, address, violator_id FROM violators";
$result = $conn->query($sql);

$violators = array();

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        $violators[] = $row;
    }
}

// Group violations by ticket_no
$groupedViolations = array();
foreach ($violationRecords as $record) {
    $ticketNo = $record['ticket_no'];
    if (!isset($groupedViolations[$ticketNo])) {
        $groupedViolations[$ticketNo] = array(
            'ticket_no' => $ticketNo,
            'name' => $record['name'],
            'location' => $record['location'],
            'violation_date' => $record['violation_date'],
            'status' => $record['status'],
            'vehicle_owner' => $record['vehicle_owner'],
            'registration_no' => $record['registration_no'],
            'plate_no' => $record['plate_no'],
            'vehicle_type' => $record['vehicle_type'],
            'body_no' => $record['body_no'],
            'enforcer_id' => $record['enforcer_id'],
            'violator_id' => $record['violator_id'],
            'violations' => array()
        );
    }
    $groupedViolations[$ticketNo]['violations'][] = array(
        'violation_type' => $record['violation_type'],
        'penalty_amount' => $record['penalty_amount']
    );
}

// Calculate offense counts for each violation type per violator
foreach ($groupedViolations as &$ticket) {
    $violatorId = $ticket['violator_id'];

    // For each violation in this ticket
    foreach ($ticket['violations'] as &$violation) {
        $violationType = $violation['violation_type'];

        // Count how many times this violator has committed this violation type before this date
        $offenseCount = 0;
        $currentDate = $ticket['violation_date'];

        foreach ($violationRecords as $record) {
            if ($record['violator_id'] == $violatorId &&
                $record['violation_type'] == $violationType &&
                $record['violation_date'] <= $currentDate) {
                $offenseCount++;
            }
```

```php
        }

        $violation['offense_count'] = $offenseCount;
    }

    // Calculate total penalty amount
    $totalAmount = 0;
    foreach ($ticket['violations'] as $violation) {
        $totalAmount += (float)$violation['penalty_amount'];
    }
    $ticket['total_penalty_amount'] = $totalAmount;
}

// Convert to indexed array
$groupedViolationsArray = array_values($groupedViolations);

$response = array(
    'violationRecords' => $violationRecords,
    'violators' => $violators,
    'groupedViolations' => $groupedViolationsArray
);

echo json_encode($response);

$conn->close();
?>
```

generate_ticket_number.php

```php
<!--
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET');
header('Access-Control-Allow-Headers: Access-Control-Allow-Headers, Content-Type, Access-Control-Allow-Methods, Authorization, X-Requested-With');

include '../conn.php';

date_default_timezone_set('Asia/Manila');

function generateTicketNumber($conn)
{
    $sql = "SELECT ticket_no FROM violationrecords ORDER BY ticket_no DESC LIMIT 1";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        $row = $result->fetch_assoc();
        $lastTicketNo = $row['ticket_no'];
        $nextTicketNo = intval($lastTicketNo) + 1;
    } else {
        $nextTicketNo = 1;
    }

    $formattedTicketNo = str_pad($nextTicketNo, 4, '0', STR_PAD_LEFT);
    return $formattedTicketNo;
}

try {
    $ticketNo = generateTicketNumber($conn);

    http_response_code(200);
    echo json_encode([
        'status' => 'success',
        'ticket_no' => $ticketNo
    ]);
} catch (Exception $e) {
    http_response_code(500);
    echo json_encode([
        'status' => 'error',
        'message' => 'Exception occurred: ' . $e->getMessage()
    ]);
}

$conn->close();

 -->

 <?php
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET');
header('Access-Control-Allow-Headers: Access-Control-Allow-Headers, Content-Type, Access-Control-Allow-Methods, Authorization, X-Requested-With');

include '../conn.php';

date_default_timezone_set('Asia/Manila');

Pieces: Comment | Pieces: Explain
function generateTicketNumber($conn)
{
    $current_year = date('Y');

    // Get the highest ticket number for the current year
    $sql = "SELECT MAX(SUBSTRING(ticket_no, 5)) as max_ticket_seq
            FROM violationrecords
            WHERE ticket_no LIKE '{$current_year}%'";

    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        $row = $result->fetch_assoc();
        $max_ticket_seq = empty($row['max_ticket_seq']) ? 0 : intval($row['max_ticket_seq']);
        $next_ticket_seq = $max_ticket_seq + 1;
    } else {
```

```
        $next_ticket_seq = 1;
    }

    $next_ticket_no = $current_year . sprintf('%04d', $next_ticket_seq);
    return $next_ticket_no;
}

try {
    $ticketNo = generateTicketNumber($conn);

    http_response_code(200);
    echo json_encode([
        'status' => 'success',
        'ticket_no' => $ticketNo
    ]);
} catch (Exception $e) {
    http_response_code(500);
    echo json_encode([
        'status' => 'error',
        'message' => 'Exception occurred: ' . $e->getMessage()
    ]);
}

$conn->close();
?>

```

get_recent_tickets.php

```
<?php
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET');
header('Access-Control-Allow-Headers: Access-Control-Allow-Headers, Content-Type, Access-Control-Allow-Methods, Authorization, X-Requested-With');

include '../conn.php';

$enforcerId = isset($_GET['enforcer_id']) ? $_GET['enforcer_id'] : null;
error_log("Received enforcer_id: " . ($enforcerId ? $enforcerId : "null"));

if ($enforcerId === "") {
    error_log("Warning: Empty enforcer_id received in request");
}

$enforcerLookup = [];
$enforcerSql = "SELECT enforcer_id, name, badge_number FROM enforcers";
$enforcerResult = $conn->query($enforcerSql);

if ($enforcerResult->num_rows > 0) {
    while ($enfRow = $enforcerResult->fetch_assoc()) {
        $enforcerLookup[$enfRow['enforcer_id']] = [
            'name' => $enfRow['name'],
            'badge_number' => $enfRow['badge_number']
        ];
    }
}

$offenseCountSql = "SELECT violator_id, violation_type, violation_date, ticket_no
                    FROM violationrecords
                    ORDER BY violator_id, violation_type, violation_date ASC";

$offenseResult = $conn->query($offenseCountSql);
$offenseCounts = [];

if ($offenseResult->num_rows > 0) {
    while ($row = $offenseResult->fetch_assoc()) {
```

```php
        $key = $row['violator_id'] . '_' . $row['violation_type'];
        if (!isset($offenseCounts[$key])) {
            $offenseCounts[$key] = [];
        }
        $offenseCounts[$key][] = [
            'ticket_no' => $row['ticket_no'],
            'violation_date' => $row['violation_date']
        ];
    }
}

$sql = "SELECT vr.*, v.name as violator_name, v.email as violator_email,
        v.phone as violator_phone, v.address as violator_address
        FROM violationrecords vr
        INNER JOIN violators v ON vr.violator_id = v.violator_id";

if ($enforcerId) {
    $sql .= " WHERE vr.enforcer_id = '$enforcerId'";
    error_log("Applying filter for enforcer_id: $enforcerId");
}

$sql .= " ORDER BY vr.violation_date DESC";
error_log("Executing SQL query: " . $sql);

$result = $conn->query($sql);

if ($result->num_rows > 0) {
    $ticketsArray = [];
    $tempTickets = [];

    while ($row = $result->fetch_assoc()) {
        $ticketNo = $row['ticket_no'];
        $rowEnforcerId = $row['enforcer_id'];

        $enforcerName = isset($enforcerLookup[$rowEnforcerId]) ? $enforcerLookup[$rowEnforcerId]['name'] : 'Unknown';
        $enforcerBadge = isset($enforcerLookup[$rowEnforcerId]) ? $enforcerLookup[$rowEnforcerId]['badge_number'] : 'Unknown';
```

```php
        if (!isset($tempTickets[$ticketNo])) {
            $tempTickets[$ticketNo] = [
                'ticket_no' => $ticketNo,
                'violator_id' => $row['violator_id'],
                'violator_name' => $row['violator_name'],
                'violator_email' => $row['violator_email'],
                'violator_phone' => $row['violator_phone'],
                'violator_address' => $row['violator_address'],
                'enforcer_id' => $rowEnforcerId,
                'enforcer_name' => $enforcerName,
                'enforcer_badge_number' => $enforcerBadge,
                'violation_date' => $row['violation_date'],
                'location' => $row['location'],
                'vehicle_owner' => $row['vehicle_owner'],
                'registration_no' => $row['registration_no'],
                'plate_no' => $row['plate_no'],
                'body_no' => $row['body_no'],
                'vehicle_type' => $row['vehicle_type'],
                'status' => $row['status'],
                'violations' => [],
                'total_amount' => 0
            ];
        }

        $offenseCount = 1;
        $key = $row['violator_id'] . '_' . $row['violation_type'];

        if (isset($offenseCounts[$key])) {
            foreach ($offenseCounts[$key] as $index => $offense) {
                if ($offense['ticket_no'] == $ticketNo) {
                    $offenseCount = $index + 1;
                    break;
                }
            }
        }

        $tempTickets[$ticketNo]['violations'][] = [
```

```php
                'violation_type' => $row['violation_type'],
                'penalty_amount' => (float) $row['penalty_amount'],
                'offense_count' => $offenseCount
            ];

            $tempTickets[$ticketNo]['total_amount'] += (float) $row['penalty_amount'];
        }

        foreach ($tempTickets as $ticket) {
            $ticketsArray[] = $ticket;
        }

        echo json_encode([
            "success" => true,
            "tickets" => $ticketsArray,
            "debug" => [
                "received_enforcer_id" => $enforcerId,
                "filter_applied" => ($enforcerId ? true : false),
                "ticket_count" => count($ticketsArray)
            ]
        ]);
    } else {
        echo json_encode([
            "success" => true,
            "tickets" => [],
            "debug" => [
                "received_enforcer_id" => $enforcerId,
                "filter_applied" => ($enforcerId ? true : false),
                "ticket_count" => 0
            ]
        ]);
    }

    $conn->close();
    ?>
```

get.php

```php
<?php
include '../conn.php';

$allowedTables = ['violationrecords', 'tickets', 'violators', 'violations'];

if (isset($_GET['table']) && in_array($_GET['table'], $allowedTables)) {
    $table = $_GET['table'];

    if ($table === 'violationrecords' && isset($_GET['violator_id']) && isset($_GET['violation_type'])) {
        $violatorId = $_GET['violator_id'];
        $violationType = $_GET['violation_type'];

        $sql = "SELECT COUNT(*) AS violation_count
                FROM $table
                WHERE violator_id = ? AND violation_type = ?";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("is", $violatorId, $violationType);
        $stmt->execute();
        $result = $stmt->get_result();

        if ($result->num_rows > 0) {
            $row = $result->fetch_assoc();
            $data = ['violation_count' => $row['violation_count']];
        } else {
            $data = ['violation_count' => 0];
        }

        $stmt->close();
    } else {
        if ($table === 'violationrecords') {
            $sql = "SELECT vr.*, v.name AS violator_name
                    FROM $table vr
                    LEFT JOIN violators v ON vr.violator_id = v.violator_id
                    ORDER BY vr.violation_date";
            $result = $conn->query($sql);

            $records = [];
```

```php
            if ($result->num_rows > 0) {
                while ($row = $result->fetch_assoc()) {
                    $row['name'] = $row['violator_name'];
                    $records[] = $row;
                }
            }

            $groupedData = [];
            foreach ($records as $record) {
                $ticketNo = $record['ticket_no'];
                if (!isset($groupedData[$ticketNo])) {
                    $groupedData[$ticketNo] = [
                        'ticket_no' => $ticketNo,
                        'name' => $record['name'],
                        'location' => $record['location'],
                        'violation_date' => $record['violation_date'],
                        'status' => $record['status'],
                        'vehicle_owner' => $record['vehicle_owner'],
                        'registration_no' => $record['registration_no'],
                        'plate_no' => $record['plate_no'],
                        'vehicle_type' => $record['vehicle_type'],
                        'body_no' => $record['body_no'],
                        'enforcer_id' => $record['enforcer_id'],
                        'violator_id' => $record['violator_id'],
                        'violations' => []
                    ];
                }
                $groupedData[$ticketNo]['violations'][] = [
                    'violation_type' => $record['violation_type'],
                    'penalty_amount' => $record['penalty_amount']
                ];
            }

            foreach ($groupedData as &$ticket) {
                $violatorId = $ticket['violator_id'];

                foreach ($ticket['violations'] as &$violation) {
```

```
                    $violationType = $violation['violation_type'];
                    $currentDate = $ticket['violation_date'];

                    $offenseCount = 0;
                    foreach ($records as $record) {
                        if ($record['violator_id'] == $violatorId &&
                            $record['violation_type'] == $violationType &&
                            $record['violation_date'] <= $currentDate) {
                            $offenseCount++;
                        }
                    }

                    $violation['offense_count'] = $offenseCount;
                }

                $totalAmount = 0;
                foreach ($ticket['violations'] as $violation) {
                    $totalAmount += (float)$violation['penalty_amount'];
                }
                $ticket['total_penalty_amount'] = $totalAmount;
            }

            $data = array_values($groupedData);
        } elseif ($table === 'violations') {
            $sql = "SELECT * FROM $table WHERE status = 'enabled'";
            $result = $conn->query($sql);

            $data = [];
            if ($result->num_rows > 0) {
                while ($row = $result->fetch_assoc()) {
                    $data[] = $row;
                }
            }
        } else {
            $sql = "SELECT * FROM $table";
            $result = $conn->query($sql);

            $data = [];
            if ($result->num_rows > 0) {
                while ($row = $result->fetch_assoc()) {
                    $data[] = $row;
                }
            }
        }
    }

    $conn->close();

    header('Content-Type: application/json');
    echo json_encode($data);
} else {
    http_response_code(400);
    echo json_encode(['error' => 'Invalid or missing table parameter']);
}
```

login.php

```php
<?php
header('Content-Type: application/json');

include '../conn.php';

date_default_timezone_set('Asia/Manila');

$data = json_decode(file_get_contents("php://input"), true);
$user = $data['username'] ?? '';
$pass = $data['password'] ?? '';

if (empty($user) || empty($pass)) {
    echo json_encode(["status" => "error", "message" => "Username or password is missing."]);
    exit;
}

$stmt = $conn->prepare("SELECT * FROM enforcers WHERE username = ? AND status = 'active'");
$stmt->bind_param("s", $user);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    $row = $result->fetch_assoc();
    if (password_verify($pass, $row['password'])) {
        $enforcer_id = $row['enforcer_id'];
        $name = $row['name'];
        $email = $row['email'];
        $badge_number = $row['badge_number'];
        $phone = $row['phone'];
        $department = $row['department'];

        $info = "$name has logged in";
        $action = "logged in";
        $date_time = date("Y-m-d H:i:s");

        $log_stmt = $conn->prepare("INSERT INTO logs (info, date_time, action) VALUES (?, ?, ?)");
        $log_stmt->bind_param("sss", $info, $date_time, $action);
        $log_stmt->execute();
        $log_stmt->close();

        echo json_encode([
            "status" => "success",
            "message" => "Login successful.",
            "enforcer_id" => $enforcer_id,
            "name" => $name,
            "email" => $email,
            "badge_number" => $badge_number,
            "phone" => $phone,
            "department" => $department,
            "username" => $user
        ]);
    } else {
        echo json_encode(["status" => "error", "message" => "Invalid password."]);
    }
} else {
    echo json_encode(["status" => "error", "message" => "User not found or inactive."]);
}

$stmt->close();
$conn->close();
?>
```

upload_image.php

```php
<?php
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: POST');
header('Access-Control-Allow-Headers: Access-Control-Allow-Headers, Content-Type, Access-Control-Allow-Methods, Authorization, X-Requested-With');

include '../conn.php';

if (!isset($_POST['ticket_no']) || empty($_POST['ticket_no'])) {
    http_response_code(400);
    echo json_encode([
        'status' => 'error',
        'message' => 'Ticket number is required'
    ]);
    exit();
}

$ticket_no = $_POST['ticket_no'];
$upload_dir = '../../uploads/';

if (!file_exists($upload_dir)) {
    mkdir($upload_dir, 0777, true);
}

if (!isset($_FILES['image']) || $_FILES['image']['error'] != UPLOAD_ERR_OK) {
    http_response_code(400);
    echo json_encode([
        'status' => 'error',
        'message' => 'Image upload failed'
    ]);
    exit();
}

$file_ext = pathinfo($_FILES['image']['name'], PATHINFO_EXTENSION);
$file_name = uniqid('img_', true) . '.' . $file_ext;
$file_path = $upload_dir . $file_name;
```

```php
if (move_uploaded_file($_FILES['image']['tmp_name'], $file_path)) {
    $stmt = $conn->prepare("INSERT INTO evidence (ticket_no, image_name) VALUES (?, ?)");
    $stmt->bind_param("ss", $ticket_no, $file_name);

    if ($stmt->execute()) {
        http_response_code(200);
        echo json_encode([
            'status' => 'success',
            'message' => 'Image uploaded and record created successfully'
        ]);
    } else {
        http_response_code(500);
        echo json_encode([
            'status' => 'error',
            'message' => 'Failed to create evidence record: ' . $stmt->error
        ]);
    }

    $stmt->close();
} else {
    http_response_code(500);
    echo json_encode([
        'status' => 'error',
        'message' => 'Failed to move uploaded file'
    ]);
}

$conn->close();
?>
```

WEB SYSTEM

add-account.php

```php
<?php
ini_set('display_errors', 0);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

header('Content-Type: application/json');

$response = [
    'success' => false,
    'message' => '',
    'error' => [
        'message' => '',
        'validation' => [],
        'database' => null,
        'debug' => []
    ],
    'debug_info' => []
];

try {
    session_name('trrms_user');
    session_start();

    if ($_SERVER["REQUEST_METHOD"] !== "POST") {
        throw new Exception("Invalid request method. Only POST is allowed.");
    }

    if (!isset($_SESSION['name'])) {
        throw new Exception("Session validation failed. User not authenticated.");
    }

    include 'conn.php';
    if (!$conn) {
        throw new Exception("Failed to connect to database: " . mysqli_connect_error());
    }

    date_default_timezone_set('Asia/Manila');
```

```php

    $requiredFields = [
        'name' => 'Name is required',
        'email' => 'Email is required',
        'phone' => 'Phone number is required',
        'username' => 'Username is required',
        'password' => 'Password is required',
        'table' => 'Account type is required'
    ];

    $validationErrors = [];
    $inputValues = [];

    foreach ($requiredFields as $field => $errorMsg) {
        if (!isset($_POST[$field])) {
            $validationErrors[$field] = "Field '$field' is missing in the request";
        } elseif (empty(trim($_POST[$field]))) {
            $validationErrors[$field] = $errorMsg;
        }
    }

    if (!empty($validationErrors)) {
        $response['error']['validation'] = $validationErrors;
        throw new Exception("Form validation failed");
    }

    $name = htmlspecialchars(trim($_POST['name']));
    $email = htmlspecialchars(trim($_POST['email']));
    $phone = htmlspecialchars(trim($_POST['phone']));
    $username = htmlspecialchars(trim($_POST['username']));
    $password = htmlspecialchars(trim($_POST['password']));
    $table = htmlspecialchars(trim($_POST['table']));

    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $validationErrors['email'] = "Invalid email format";
    }

```

```php
    if (strlen($password) < 8) {
        $validationErrors['password'] = "Password must be at least 8 characters";
    }

    if (!empty($validationErrors)) {
        $response['error']['validation'] = $validationErrors;
        throw new Exception("Form validation failed");
    }

    $hashed_password = password_hash($password, PASSWORD_DEFAULT);
    $status = 'active';

    $allowedTables = ['admins', 'enforcers', 'violations'];
    if (!in_array($table, $allowedTables)) {
        throw new Exception("Invalid table specified: '$table'");
    }

```

```php
    $conn->begin_transaction();

    try {
        if ($table === "enforcers") {
            $enforcerRequired = [
                'department' => 'Department is required',
                'badge' => 'Badge number is required'
            ];

            foreach ($enforcerRequired as $field => $errorMsg) {
                if (!isset($_POST[$field])) {
                    $validationErrors[$field] = "Field '$field' is missing in the request";
                } elseif (empty(trim($_POST[$field]))) {
                    $validationErrors[$field] = $errorMsg;
                }
            }

            if (!empty($validationErrors)) {
                $response['error']['validation'] = $validationErrors;
                throw new Exception("Enforcer validation failed");
            }

            $department = htmlspecialchars(trim($_POST['department']));
            $badge = htmlspecialchars(trim($_POST['badge']));

            $sql = "INSERT INTO $table (name, email, phones, username, password, department, badge_number, status)
                    VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
            $stmt = $conn->prepare($sql);
            if (!$stmt) {
                throw new Exception("Prepare failed: " . $conn->error . " (SQL: $sql)");
            }

            $bindResult = $stmt->bind_param(
                "ssssssss",
                $name,
                $email,
                $phone,
                $username,
                $hashed_password,
                $department,
                $badge,
                $status
            );
            if (!$bindResult) {
                throw new Exception("Bind param failed: " . $stmt->error);
            }

            if (!$stmt->execute()) {
                throw new Exception("Execution failed: " . $stmt->error);
            }

            $stmt->close();

            logAction($conn, $_SESSION['name'] . " inserted new enforcer account", "added account");

            $response['message'] = 'Enforcer account added successfully!';
        } elseif ($table === "violations") {
            $violationRequired = [
                'violationType' => 'Violation type is required',
                'amount' => 'Amount is required'
            ];

            foreach ($violationRequired as $field => $errorMsg) {
                if (!isset($_POST[$field])) {
                    $validationErrors[$field] = "Field '$field' is missing in the request";
                } elseif (empty(trim($_POST[$field]))) {
                    $validationErrors[$field] = $errorMsg;
                }
            }

            if (!empty($validationErrors)) {
                $response['error']['validation'] = $validationErrors;
                throw new Exception("Violation validation failed");
            }

            $violationType = htmlspecialchars(trim($_POST['violationType']));
            $amount = htmlspecialchars(trim($_POST['amount']));

            if (!is_numeric($amount) || $amount <= 0) {
                $validationErrors['amount'] = "Amount must be a positive number";
                $response['error']['validation'] = $validationErrors;
                throw new Exception("Violation validation failed");
            }

            $sql = "INSERT INTO $table (name, amount) VALUES (?, ?)";
            $stmt = $conn->prepare($sql);
            if (!$stmt) {
                throw new Exception("Prepare failed: " . $conn->error . " (SQL: $sql)");
            }

            $bindResult = $stmt->bind_param("sd", $violationType, $amount);
            if (!$bindResult) {
                throw new Exception("Bind param failed: " . $stmt->error);
            }
```

```php
            if (!$stmt->execute()) {
                throw new Exception("Execution failed: " . $stmt->error);
            }

            $stmt->close();

            logAction($conn, $_SESSION['name'] . " inserted new violation", "added violation");

            $response['message'] = 'Violation added successfully!';
            $response['reload'] = false;
        } else {
            $sql = "INSERT INTO $table (name, email, phone, username, password, status)
                    VALUES (?, ?, ?, ?, ?, ?)";
            $stmt = $conn->prepare($sql);
            if (!$stmt) {
                throw new Exception("Prepare failed: " . $conn->error . " (SQL: $sql)");
            }

            $bindResult = $stmt->bind_param(
                "ssssss",
                $name,
                $email,
                $phone,
                $username,
                $hashed_password,
                $status
            );
            if (!$bindResult) {
                throw new Exception("Bind param failed: " . $stmt->error);
            }

            if (!$stmt->execute()) {
                throw new Exception("Execution failed: " . $stmt->error);
            }

            $stmt->close();
```

```php
            logAction($conn, $_SESSION['name'] . " inserted new admin account", "added account");

            $response['message'] = 'Admin account added successfully!';
        }

        $conn->commit();
        $response['success'] = true;

    } catch (Exception $e) {
        $conn->rollback();
        throw $e;
    }

} catch (Exception $e) {
    $response['error']['message'] = $e->getMessage();
    $response['error']['debug'] = [
        'file' => $e->getFile(),
        'line' => $e->getLine(),
        'trace' => $e->getTraceAsString()
    ];

    error_log("Add Account Error: " . print_r($response['error'], true));
    error_log("POST Data: " . print_r($_POST, true));

    http_response_code(400);
}

if (isset($conn) && $conn) {
    $conn->close();
}

echo json_encode($response);

Pieces: Comment | Pieces: Explain
function logAction($conn, $info, $action)
{
```

```php
    $date_time = date('Y-m-d H:i:s');
    $log_sql = "INSERT INTO logs (info, date_time, action) VALUES (?, ?, ?)";
    $log_stmt = $conn->prepare($log_sql);

    if (!$log_stmt) {
        throw new Exception("Log prepare failed: " . $conn->error);
    }

    $log_stmt->bind_param("sss", $info, $date_time, $action);
    if (!$log_stmt->execute()) {
        throw new Exception("Log execution failed: " . $log_stmt->error);
    }

    $log_stmt->close();
}
?>
```

conn.php

```php
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "trrms";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

delete.php

```php
<?php
ini_set('display_errors', 0);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

header('Content-Type: application/json');

$response = [
    'success' => false,
    'message' => '',
    'error' => [
        'message' => '',
        'debug' => []
    ]
];

try {
    session_name('trrms_user');
    session_start();
    date_default_timezone_set('Asia/Manila');

    if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
        throw new Exception("Invalid request method. Only POST is allowed.");
    }

    include 'conn.php';
    if (!$conn) {
        throw new Exception("Database connection failed: " . mysqli_connect_error());
    }

    if (!isset($_POST['id']) || !isset($_POST['table'])) {
        throw new Exception("Missing required parameters (id or table).");
    }

    $id = intval($_POST['id']);
    $table = htmlspecialchars($_POST['table'], ENT_QUOTES);

    if ($id <= 0) {
        throw new Exception("Invalid ID provided.");
    }

    $allowedTables = ['violations', 'enforcers', 'admins'];
    if (!in_array($table, $allowedTables)) {
        throw new Exception("Invalid table specified: '$table'. Allowed: " . implode(', ', $allowedTables));
    }

    $conn->begin_transaction();

    try {
        $sql = "DELETE FROM $table WHERE id = ?";
        $stmt = $conn->prepare($sql);

        if (!$stmt) {
            throw new Exception("Prepare statement failed: " . $conn->error);
        }

        $stmt->bind_param("i", $id);

        if (!$stmt->execute()) {
            throw new Exception("Execution failed: " . $stmt->error);
        }

        if ($stmt->affected_rows === 0) {
            throw new Exception("No matching record found to delete.");
        }

        $logSql = "INSERT INTO logs (infos, date_times, action) VALUES (?, NOW(), ?)";
        $logStmt = $conn->prepare($logSql);
        if (!$logStmt) {
            throw new Exception("Log statement preparation failed: " . $conn->error);
        }

        $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . " deleted a record from $table" : 'Unknown user deleted a record';
        $action = "deleted record";
```

```
        $logStmt->bind_param("ss", $logInfo, $action);

        if (!$logStmt->execute()) {
            throw new Exception("Log execution failed: " . $logStmt->error);
        }

        $conn->commit();
        $response['success'] = true;
        $response['message'] = "Record deleted successfully.";

    } catch (Exception $e) {
        $conn->rollback();
        throw $e;
    }

} catch (Exception $e) {
    $response['error']['message'] = $e->getMessage();
    $response['error']['debug'] = [
        'file' => $e->getFile(),
        'line' => $e->getLine(),
        'trace' => $e->getTraceAsString()
    ];

    error_log("Delete Error: " . print_r($response['error'], true));
    error_log("POST Data: " . print_r($_POST, true));

    http_response_code(400);
}

echo json_encode($response);

if (isset($stmt)) $stmt->close();
if (isset($logStmt)) $logStmt->close();
$conn->close();
?>
```

edit.php

```
<?php
session_name('trrms_user');
session_start();
date_default_timezone_set('Asia/Manila');
include 'conn.php';

header('Content-Type: application/json');

// ERROR REPORTING
error_reporting(E_ALL);
ini_set('display_errors', 1);

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $id = $_POST['id'] ?? null;
    $violationType = $_POST['violationType'] ?? null;
    $amount = $_POST['amount'] ?? null;

    error_log("Received data - ID: $id, ViolationType: $violationType, Amount: $amount");

    if (empty($id) || empty($violationType) || empty($amount)) {
        http_response_code(400);
        echo json_encode([
            'status' => 'error',
            'message' => 'Missing required fields.',
            'debug' => [
                'id' => $id,
                'violationType' => $violationType,
                'amount' => $amount
            ]
        ]);
        exit;
    }

    if (!is_numeric($amount)) {
        http_response_code(400);
        echo json_encode(['status' => 'error', 'message' => 'Amount must be a number.']);
        exit;
    }

    $conn->begin_transaction();

    try {
        if (!isset($violationType)) {
            throw new Exception("Variable 'violationType' is not set. Check for typos.");
        }

        $stmt = $conn->prepare("UPDATE violations SET name = ?, amount = ? WHERE violation_id = ?");
        if (!$stmt) {
            throw new Exception("Prepare failed: " . $conn->error);
        }

        error_log("Binding params: name=$violationType, amount=$amount, id=$id");

        $stmt->bind_param("sdi", $violationType, $amount, $id);
        $executed = $stmt->execute();

        if (!$executed) {
            throw new Exception("Execute failed: " . $stmt->error);
        }

        if ($stmt->affected_rows === 0) {
            $checkCurrent = $conn->prepare("SELECT 1 FROM violations WHERE violation_id = ? AND name = ? AND amount = ?");
            $checkCurrent->bind_param("isd", $id, $violationType, $amount);
            $checkCurrent->execute();
            $checkCurrent->store_result();

            if ($checkCurrent->num_rows > 0) {
                echo json_encode(['status' => 'success', 'message' => 'Values unchanged']);
                exit;
            } else {
                throw new Exception("No records updated. Violation ID may not exist.");
            }
        }

```

```php
        $logStmt = $conn->prepare("INSERT INTO logs (info, date_time, action) VALUES (?, NOW(), ?)");
        if (!$logStmt) {
            throw new Exception("Prepare failed for logs: " . $conn->error);
        }

        $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' updated violation details' : 'Unknown user updated violation details';
        $action = 'updated violation';
        $logStmt->bind_param("ss", $logInfo, $action);
        $logStmt->execute();

        $conn->commit();
        echo json_encode(['status' => 'success']);

    } catch (Exception $e) {
        $conn->rollback();
        http_response_code(500);

        $errorData = [
            'status' => 'error',
            'message' => $e->getMessage(),
            'error_details' => [
                'file' => $e->getFile(),
                'line' => $e->getLine(),
                'trace' => $e->getTraceAsString()
            ],
            'sql_error' => $conn->error ?? null,
            'debug_info' => [
                'variables' => get_defined_vars()
            ]
        ];

        error_log("Error in edit.php: " . print_r($errorData, true));
        echo json_encode($errorData);
    } finally {
        if (isset($stmt))
            $stmt->close();
        if (isset($logStmt))
            $logStmt->close();
        $conn->close();
    }
} else {
    http_response_code(405);
    echo json_encode(['status' => 'error', 'message' => 'Invalid request method.']);
}
?>
```

get_counts.php

```php
<?php
session_start();
require_once 'conn.php';

ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

Pieces: Comment | Pieces: Explain
function executeQuery($conn, $query, $errorLocation)
{
    $result = $conn->query($query);
    if (!$result) {
        return [
            'error' => true,
            'error_details' => $conn->error,
            'error_query' => $query,
            'error_location' => $errorLocation
        ];
    }
    return $result;
}

$response = [];

try {
    if (!$conn) {
        throw new Exception("Database connection failed");
    }

    $query = "SELECT COUNT(*) as active_count FROM enforcers WHERE status = 'active'";
    $result = executeQuery($conn, $query, 'active_enforcers_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $active_enforcers = $result->fetch_assoc()['active_count'];
```

```php
    $query = "SELECT COUNT(*) as active_count FROM admins WHERE status = 'active'";
    $result = executeQuery($conn, $query, 'active_admins_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $active_admins = $result->fetch_assoc()['active_count'];

    $response['active_accounts'] = $active_enforcers + $active_admins;

    $query = "SELECT COUNT(*) as total_count FROM enforcers";
    $result = executeQuery($conn, $query, 'total_enforcers_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $total_enforcers = $result->fetch_assoc()['total_count'];

    $query = "SELECT COUNT(*) as total_count FROM admins";
    $result = executeQuery($conn, $query, 'total_admins_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $total_admins = $result->fetch_assoc()['total_count'];

    $response['total_accounts'] = $total_enforcers + $total_admins;

    date_default_timezone_set('Asia/Manila');
    $current_date = date('Y-m-d');

    $query = "SELECT COUNT(*) as violations_today FROM violationrecords WHERE violation_date = '$current_date'";
    $result = executeQuery($conn, $query, 'violations_today_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $response['violations_today'] = $result->fetch_assoc()['violations_today'];

    $query = "SELECT COUNT(*) as total_violations FROM violationrecords";
    $result = executeQuery($conn, $query, 'total_violations_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $response['total_violations'] = $result->fetch_assoc()['total_violations'];

    $query = "SELECT COUNT(*) as pending_violations_today FROM violationrecords WHERE violation_date = '$current_date' AND status = 'pending'";
    $result = executeQuery($conn, $query, 'pending_violations_today_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $response['pending_violations_today'] = $result->fetch_assoc()['pending_violations_today'];

    $query = "SELECT COUNT(*) as total_pending_violations FROM violationrecords WHERE status = 'pending'";
    $result = executeQuery($conn, $query, 'total_pending_violations_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $response['total_pending_violations'] = $result->fetch_assoc()['total_pending_violations'];

    $query = "SELECT COUNT(*) as resolved_violations_today FROM violationrecords WHERE status = 'resolved' AND resolved_date = '$current_date'";
    $result = executeQuery($conn, $query, 'resolved_violations_today_query');
    if (is_array($result) && isset($result['error'])) {
        echo json_encode($result);
        exit;
    }
    $response['resolved_violations_today'] = $result->fetch_assoc()['resolved_violations_today'];

    $query = "SELECT COUNT(*) as total_resolved_violations FROM violationrecords WHERE status = 'resolved'";
    $result = executeQuery($conn, $query, 'total_resolved_violations_query');
        echo json_encode($result);
        exit;
    }
    $response['total_resolved_violations'] = $result->fetch_assoc()['total_resolved_violations'];

} catch (Exception $e) {
    $response = [
        'error' => true,
        'error_details' => $e->getMessage(),
        'error_location' => $e->getFile() . ' (line ' . $e->getLine() . ')',
        'error_trace' => $e->getTraceAsString()
    ];
}

header('Content-Type: application/json');
echo json_encode($response);
?>
```

get_evidence.php

```php
<?php

session_name('trrms_user');
session_start();

if (!isset($_SESSION['username']) || empty($_SESSION['username'])) {
    header('Location: /index.php');
    exit();
}

include 'conn.php';

$ticket_no = $_GET['ticket_no'];

$sql = "SELECT * FROM evidence WHERE ticket_no = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("s", $ticket_no);
$stmt->execute();
$result = $stmt->get_result();

$evidenceData = [];

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        $evidenceData[] = [
            'image_url' => '../uploads/' . $row['image_name']
        ];
    }
} else {
    $evidenceData['error'] = 'No evidence found for this ticket.';
}

echo json_encode($evidenceData);

$stmt->close();
$conn->close();
?>
```

get.php

```php
<?php
include 'conn.php';

// ERR REPORTINGG
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

header('Content-Type: application/json');

// ALLOWED TABLES
$tables = ['admins', 'enforcers', 'logs', 'violators', 'violationrecords', 'violations'];

// RESPONSE ARRAY
$response = [
    'success' => false,
    'data' => null,
    'error' => null,
    'debug_info' => []
];

try {
    if (!isset($_GET['table'])) {
        throw new Exception('Missing table parameter');
    }

    $tableName = $_GET['table'];
    $response['debug_info']['requested_table'] = $tableName;

    if (!in_array($tableName, $tables)) {
        throw new Exception("Invalid table: '$tableName' is not an allowed table");
    }

    $filter = $_GET['filter'] ?? 'all';
    $id = $_GET['id'] ?? null;
    $sql = "";

```

```php
    $response['debug_info']['filter'] = $filter;
    $response['debug_info']['id'] = $id;

    switch ($tableName) {
        case 'violationrecords':
            $sql = "SELECT vr.*,
                    v.name, v.email, v.phone, v.address,
                    e.name AS enforcer_name
                    FROM violationrecords vr
                    LEFT JOIN violators v ON vr.violator_id = v.violator_id
                    LEFT JOIN enforcers e ON vr.enforcer_id = e.enforcer_id";

            if ($id) {
                $id = intval($id);
                $sql .= " WHERE vr.record_id = $id";
                $response['debug_info']['query_type'] = 'single_record';
            } else if ($filter === 'this_week') {
                $sql .= " WHERE YEARWEEK(vr.violation_date, 1) = YEARWEEK(CURDATE(), 1)";
                $response['debug_info']['query_type'] = 'this_week_records';
            } elseif ($filter === 'this_month') {
                $sql .= " WHERE MONTH(vr.violation_date) = MONTH(CURDATE()) AND YEAR(vr.violation_date) = YEAR(CURDATE())";
                $response['debug_info']['query_type'] = 'this_month_records';
            } else {
                $response['debug_info']['query_type'] = 'all_records';
            }

            if (isset($_GET['ticket_no'])) {
                $ticket_no = $conn->real_escape_string($_GET['ticket_no']);
                $sql = preg_replace('/\sWHERE\s.*$/i', '', $sql);
                $sql .= " WHERE vr.ticket_no = '$ticket_no'";
                $response['debug_info']['query_type'] = 'ticket_records';
            }
            break;

        case 'violators':
            $sql = "SELECT * FROM $tableName";
            if ($id) {
                $id = intval($id);
                $sql .= " WHERE violator_id = $id";
                $response['debug_info']['query_type'] = 'single_violator';
            } else {
                $response['debug_info']['query_type'] = 'all_violators';
            }
            break;

        default:
            $sql = "SELECT * FROM $tableName";
            $response['debug_info']['query_type'] = 'full_table_scan';
            break;
    }

    $response['debug_info']['generated_sql'] = $sql;

    if (!$result = $conn->query($sql)) {
        throw new Exception("Database query error: " . $conn->error);
    }

    $data = [];
    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            $data[] = $row;
        }
    }

    $response['success'] = true;
    $response['data'] = $data;
    $response['debug_info']['row_count'] = count($data);

} catch (Exception $e) {
    $response['error'] = [
        'message' => $e->getMessage(),
        'file' => $e->getFile(),
        'line' => $e->getLine(),
        'trace' => $e->getTraceAsString()
    ];

    // LOGGER FOR DETAILED ERR INFO...
    error_log("API Error: " . print_r($response['error'], true));
    error_log("Debug Info: " . print_r($response['debug_info'], true));
}

if (isset($conn) && $conn) {
    $conn->close();
}

echo json_encode($response);
?>
```

login.php

```php
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);

session_name('trrms_user');
if (!session_start()) {
    die(json_encode(['error' => 'Session initialization failed']));
}

include 'conn.php';
if (!$conn) {
    die(json_encode(['error' => 'Database connection failed: ' . mysqli_connect_error()]));
}

Pieces: Comment | Pieces: Explain
function console_error($error)
{
    echo '<script>console.error(' . json_encode($error) . ');</script>';
}

$error_message = "";
date_default_timezone_set('Asia/Manila');

if (isset($_SESSION['username']) && !empty($_SESSION['username'])) {
    header("Location: /pages/dashboard.php");
    exit();
}

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (isset($_POST['username']) && isset($_POST['password'])) {
        $user = mysqli_real_escape_string($conn, $_POST['username']);
        $pass = mysqli_real_escape_string($conn, $_POST['password']);

        try {
            $sql_superadmin = "SELECT * FROM superadmins WHERE username = '$user'";
            $result_superadmin = $conn->query($sql_superadmin);
            if ($result_superadmin === false) {
                throw new Exception("SUPERADMIN QUERY ERROR: " . $conn->error);
            }

            $sql_admin = "SELECT * FROM admins WHERE username = '$user'";
            $result_admin = $conn->query($sql_admin);
            if ($result_admin === false) {
                throw new Exception("ADMIN QUERY ERROR: " . $conn->error);
            }

            if ($result_superadmin->num_rows > 0) {
                $row = $result_superadmin->fetch_assoc();
                if (password_verify($pass, $row['password'])) {
                    if ($row['status'] == 'disabled') {
                        $error_message = "Account is disabled. Please contact the system administrator.";
                    } else {
                        $_SESSION['username'] = $row['username'];
                        $_SESSION['name'] = $row['name'];
                        $_SESSION['phone'] = $row['phone'];
                        $_SESSION['email'] = $row['email'];
                        $_SESSION['role'] = "Department head";

                        $name = $_SESSION['name'];
                        $info = $name . " logged in";
                        $date_time = date('Y-m-d H:i:s');
                        $action = "logged in";

                        $log_sql = "INSERT INTO logs (info, date_time, action) VALUES ('$info', '$date_time', '$action')";
                        if (!$conn->query($log_sql)) {
                            console_error("LOG ENTRY FAILED: " . $conn->error);
                        }

                        header("Location: /pages/dashboard.php");
                        exit();
                    }
                } else {
                    $error_message = "Invalid username or password.";
```

```
            } elseif ($result_admin->num_rows > 0) {
                $row = $result_admin->fetch_assoc();
                if (password_verify($pass, $row['password'])) {
                    if ($row['status'] == 'disabled') {
                        $error_message = "Account is disabled. Please contact the system administrator.";
                    } else {
                        $_SESSION['username'] = $row['username'];
                        $_SESSION['name'] = $row['name'];
                        $_SESSION['phone'] = $row['phone'];
                        $_SESSION['email'] = $row['email'];
                        $_SESSION['role'] = "Staff";

                        $name = $_SESSION['name'];
                        $info = $name . " logged in";
                        $date_time = date('Y-m-d H:i:s');
                        $action = "logged in";

                        $log_sql = "INSERT INTO logs (info, date_time, action) VALUES ('$info', '$date_time', '$action')";
                        if (!$conn->query($log_sql)) {
                            console_error("LOG ENTRY FAILED: " . $conn->error);
                        }

                        header("Location: /pages/dashboard.php");
                        exit();
                    }
                } else {
                    $error_message = "Invalid username or password.";
                }
            } else {
                $error_message = "No account found with that username.";
            }
        } catch (Exception $e) {
            console_error($e->getMessage());
            $error_message = "A system error occurred. Please try again later.";
        }
    } else {
        $error_message = "Please enter both username and password.";
    }
}

$conn->close();
?>
```

logout.php

```
<?php
session_name('trrms_user');
session_start();

include 'conn.php';

date_default_timezone_set('Asia/Manila');

if (isset($_SESSION['username'])) {
    $name = $_SESSION['name'];

    $info = $name . " logged out";
    $date_time = date('Y-m-d H:i:s');
    $action = "logged out";

    $log_sql = "INSERT INTO logs (info, date_time, action) VALUES ('$info', '$date_time', '$action')";
    if (!$conn->query($log_sql)) {
        error_log("Failed to insert log: " . $conn->error);
    }
}

$_SESSION = [];
session_destroy();

header('Location: /');
exit();
?>
```

Update.php

```php
<?php
session_name('trrms_user');
session_start();
date_default_timezone_set('Asia/Manila');

ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

Pieces: Comment | Pieces: Explain
function handleDBError($conn, $e, $query = '')
{
    $errorDetails = [
        'success' => false,
        'message' => 'Database operation failed',
        'error_details' => $e->getMessage(),
        'error_code' => $e->getCode(),
        'error_location' => $e->getFile() . ' (line ' . $e->getLine() . ')',
        'sql_error' => $conn->error,
        'sql_query' => $query
    ];

    error_log("TRRMS Error: " . json_encode($errorDetails));

    return $errorDetails;
}

header('Content-Type: application/json');

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    try {
        include 'conn.php';

        if (!$conn) {
            throw new Exception("Database connection failed");
        }

        if (!isset($_POST['id']) || empty($_POST['id'])) {
            echo json_encode(['success' => false, 'message' => 'Missing record ID']);
            exit;
        }

        if (!isset($_POST['table']) || empty($_POST['table'])) {
            echo json_encode(['success' => false, 'message' => 'Missing table name']);
            exit;
        }

        if (!isset($_POST['action']) || empty($_POST['action'])) {
            echo json_encode(['success' => false, 'message' => 'Missing action parameter']);
            exit;
        }

        $id = $_POST['id'];
        $table = $_POST['table'];
        $action = $_POST['action'];

        $tables = ['violationrecords', 'enforcers', 'admins', 'violations'];
        $current_date = date('Y-m-d');

        if (!in_array($table, $tables)) {
            echo json_encode(['success' => false, 'message' => 'Invalid table name: ' . $table]);
            exit;
        }

        $conn->begin_transaction();

        try {
            $stmt = null;
            $logInfo = '';
            $logAction = '';

            if ($table === 'enforcers') {
                $status = $action === 'disable' ? 'disabled' : 'active';
                $stmt = $conn->prepare("UPDATE $table SET status = ? WHERE enforcer_id = ?");
```

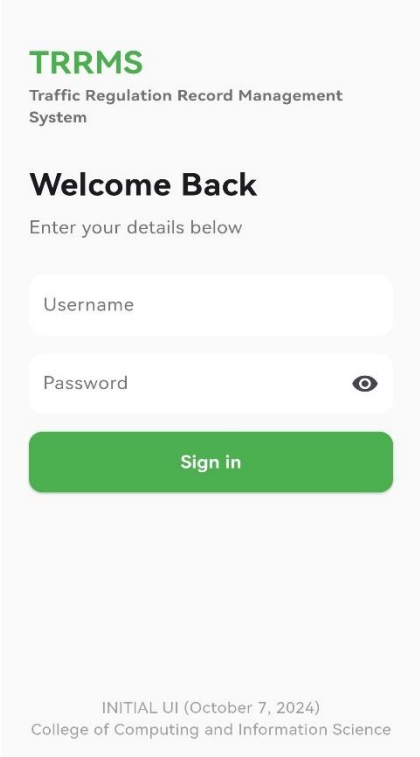
```
        if (!$stmt) {
            throw new Exception("Prepare failed: " . $conn->error);
        }
        $stmt->bind_param('si', $status, $id);

        $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' ' . ($action === 'disable' ? 'disabled' : 'activated') . ' an account' :
            'Unknown user ' . ($action === 'disable' ? 'disabled' : 'activated') . ' an account';
        $logAction = $action === 'disable' ? 'disabled account' : 'activated account';
    } else if ($table === 'admins') {
        $status = $action === 'disable' ? 'disabled' : 'active';
        $stmt = $conn->prepare("UPDATE $table SET status = ? WHERE admin_id = ?");
        if (!$stmt) {
            throw new Exception("Prepare failed: " . $conn->error);
        }
        $stmt->bind_param('si', $status, $id);

        $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' ' . ($action === 'disable' ? 'disabled' : 'activated') . ' an account' :
            'Unknown user ' . ($action === 'disable' ? 'disabled' : 'activated') . ' an account';
        $logAction = $action === 'disable' ? 'disabled account' : 'activated account';
    } else if ($table === 'violations') {
        $status = $action === 'disable' ? 'disabled' : 'enabled';
        $stmt = $conn->prepare("UPDATE $table SET status = ? WHERE violation_id = ?");
        if (!$stmt) {
            throw new Exception("Prepare failed: " . $conn->error);
        }
        $stmt->bind_param('si', $status, $id);

        $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' ' . ($action === 'disable' ? 'disabled' : 'enabled') . ' a violation type'
            'Unknown user ' . ($action === 'disable' ? 'disabled' : 'enabled') . ' a violation type';
        $logAction = $action === 'disable' ? 'disabled violation type' : 'enabled violation type';
    } else if ($table === 'violationrecords') {
        $or_no = isset($_POST['or_no']) ? $_POST['or_no'] : '';

        if (empty($or_no)) {
            echo json_encode(['success' => false, 'message' => 'OR Number is required']);
            exit;
        }
```

```
        $stmt = $conn->prepare("UPDATE $table SET status = 'resolved', resolved_date = ?, or_no = ? WHERE ticket_no = ?");
        if (!$stmt) {
            throw new Exception("Prepare failed: " . $conn->error);
        }
        $stmt->bind_param('sss', $current_date, $or_no, $id);

        $logInfo = isset($_SESSION['name']) ? $_SESSION['name'] . ' resolved a violation with OR#: ' . $or_no : 'Unknown user resolved a violati
        $logAction = 'resolved violation';
    }

    if (!$stmt->execute()) {
        throw new Exception("Execute failed: " . $stmt->error);
    }

    if ($stmt->affected_rows == 0) {
        throw new Exception("No records were updated. The record ID may not exist: " . $id);
    }

    $logStmt = $conn->prepare("INSERT INTO logs (info, date_time, action) VALUES (?, NOW(), ?)");
    if (!$logStmt) {
        throw new Exception("Log prepare failed: " . $conn->error);
    }
    $logStmt->bind_param("ss", $logInfo, $logAction);

    if (!$logStmt->execute()) {
        throw new Exception("Log execute failed: " . $logStmt->error);
    }

    $conn->commit();
    echo json_encode(['success' => true]);
} catch (Exception $e) {
    $conn->rollback();
    $errorDetails = handleDBError($conn, $e);
    echo json_encode($errorDetails);
}
```

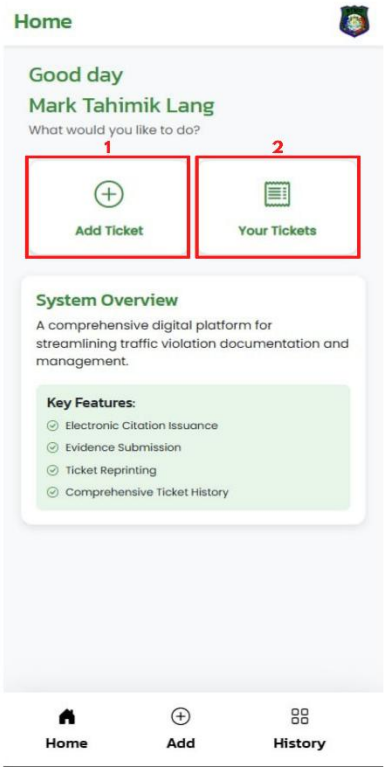
```
    if (isset($stmt) && $stmt) {
        $stmt->close();
    }
    if (isset($logStmt) && $logStmt) {
        $logStmt->close();
    }
atch (Exception $e) {
    echo json_encode([
        'success' => false,
        'message' => 'System error: ' . $e->getMessage(),
        'error_location' => $e->getFile() . ' (line ' . $e->getLine() . ')',
        'error_trace' => $e->getTraceAsString()
    ]);


(isset($conn) && $conn) {
    $conn->close();

{
o json_encode(['success' => false, 'message' => 'Invalid request method. Only POST is allowed.']);
```

B. User's Manual

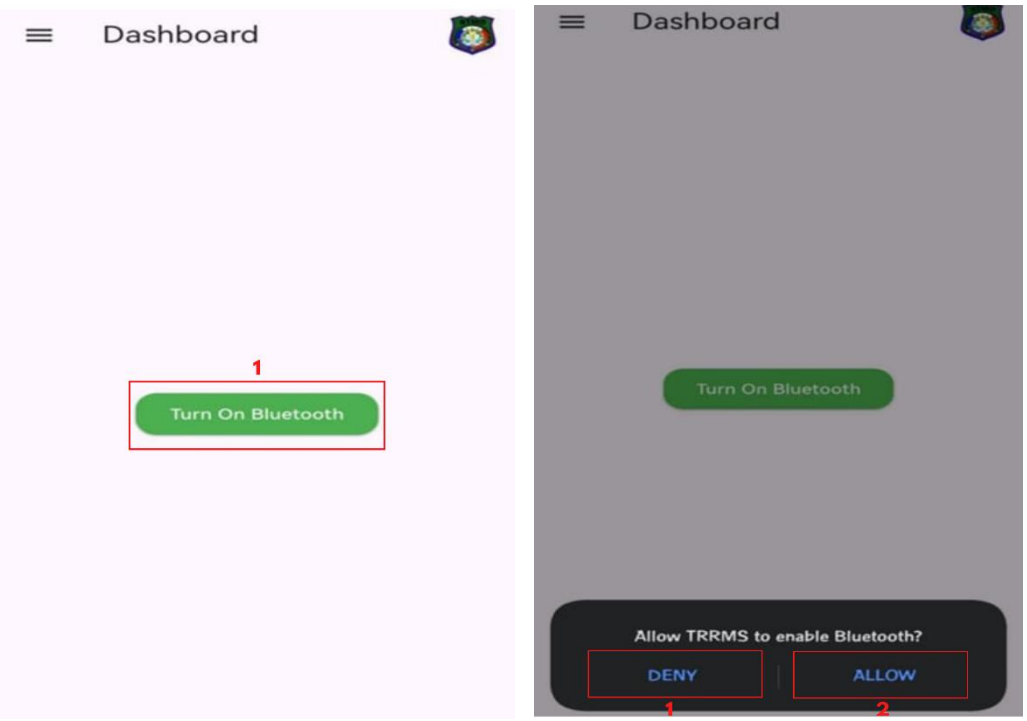
MOBILE APPLICATION



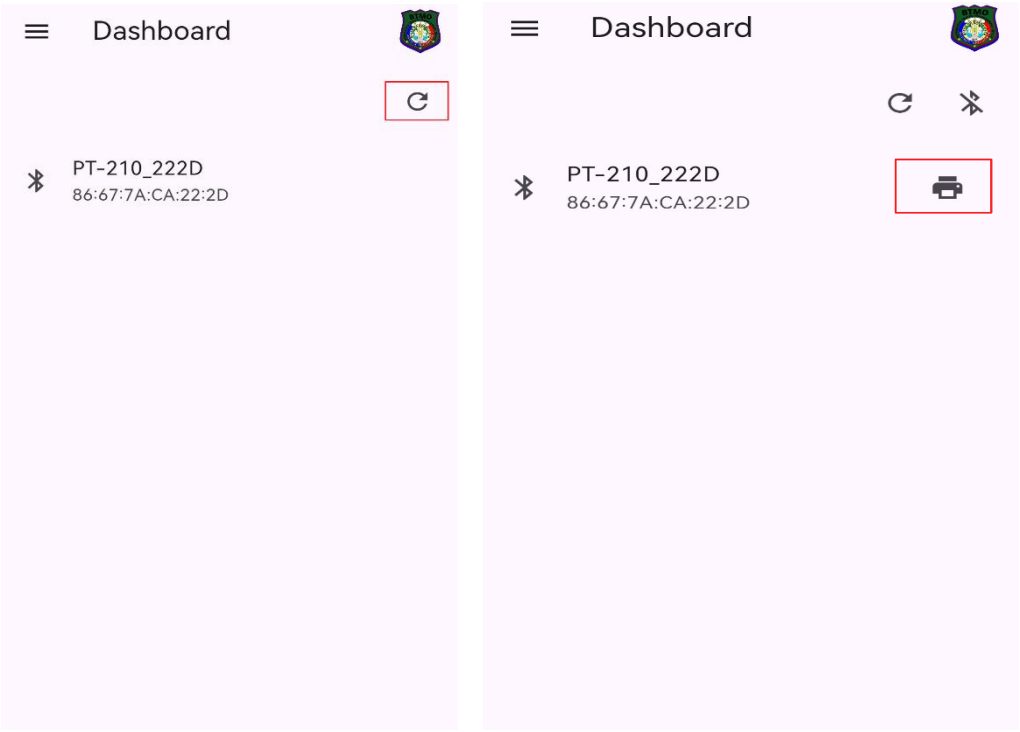
To begin the login process, launch the application on the device. The enforcer must enter the assigned username and password into the appropriate fields. Once the credentials are entered, click the "Sign In" button. If the information is valid, the enforcer will be granted access to the system.



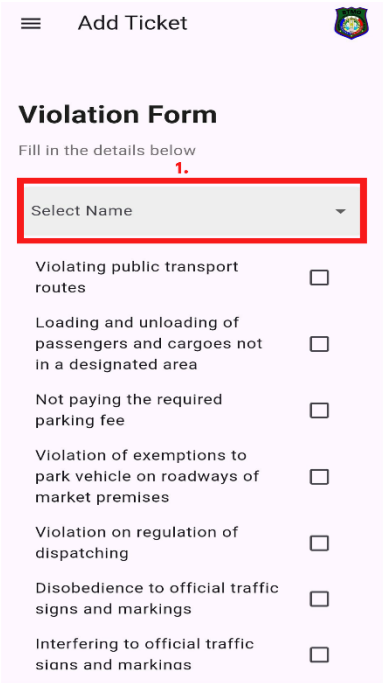
The homepage provides two main functions for the enforcer. The (1) Add Ticket feature allows the enforcer to create and submit a new violation record. The (2) Your Tickets section enables the enforcer to view and manage previously issued tickets. These tools help the enforcer efficiently monitor and handle violation reports.



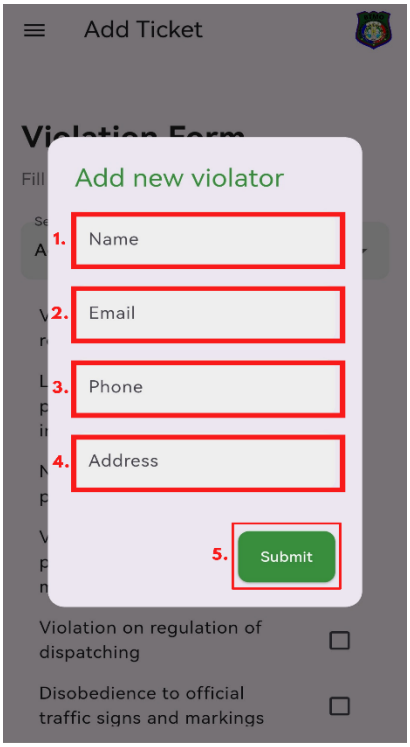
To begin the process, first ensure that Bluetooth is turned on. Once Bluetooth is enabled, a connection notification will appear on your device. This notification will present two options: (1) Deny and (2) Allow. Select Allow to grant permission for the connection to the printer or other device.



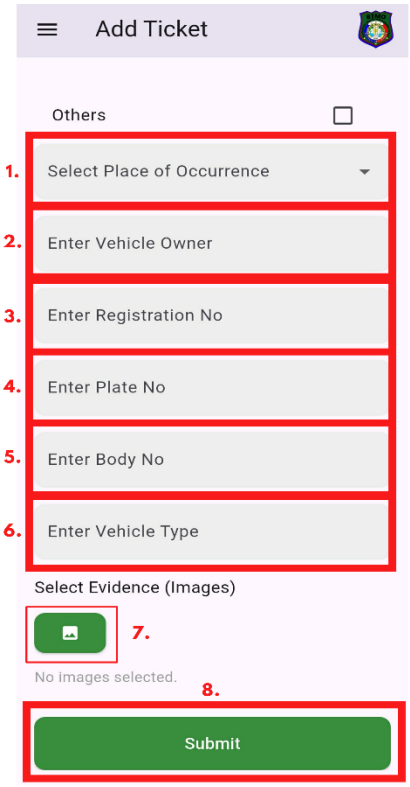
After selecting Allow to connect to TRRMS, a refresh icon will appear on the screen. Tap the refresh icon to update the connection status. Once the refresh is complete, the print icon will become visible, indicating that the printer is ready for use, and you can proceed with printing the violation ticket.



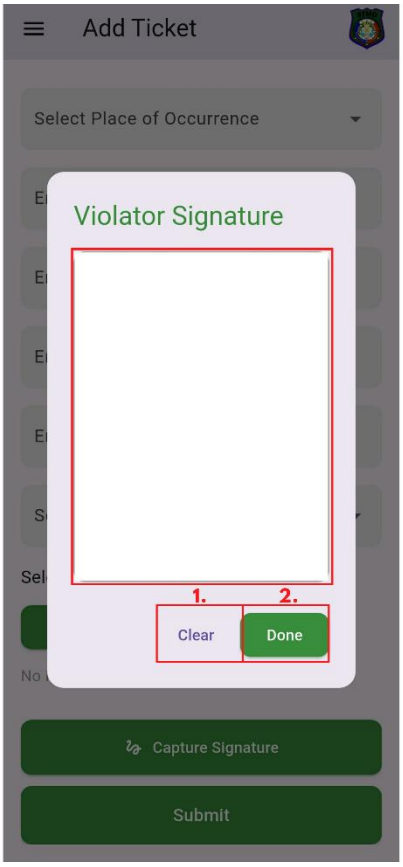
After connecting to the printer, the violation form will be displayed for the enforcer to fill out. The first step is to (1) Select Name, where the enforcer chooses the name of the violator. This can be either an existing violator from the list or a new violator, depending on the situation. Following this, the enforcer will choose the appropriate type of violation from the available dropdown options.



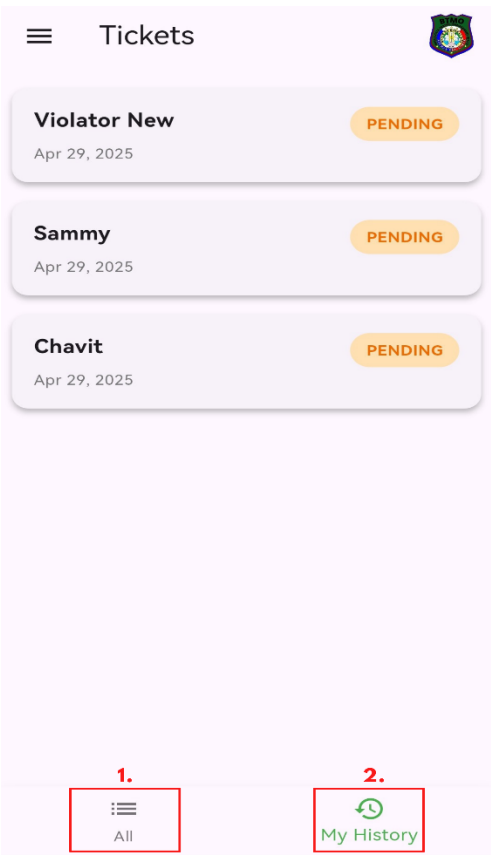
If the violator is new, the enforcer will be prompted to fill out the Add New Violator form. The enforcer must enter the following details: (1) the violator's full name, (2) email address, (3) phone number, and (4) residential address. After all fields are completed, (5) click the Submit button to save the new violator's information.



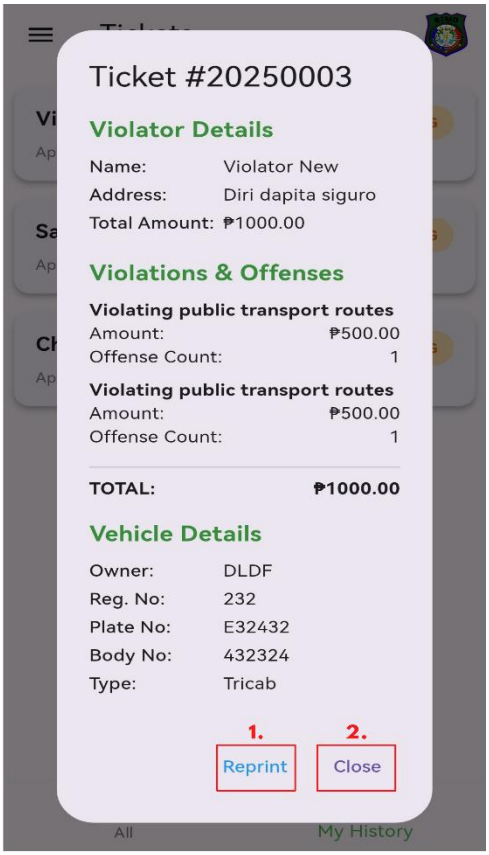
After selecting the type of violation, the enforcer must complete the remaining fields in the form. (1) Select Place of Occurrence from the dropdown list, which includes all barangays in Buenavista. Then, (2) enter the Vehicle Owner's name, (3) the Registration Number, (4) the Plate Number, (5) the Body Number, and (6) the Vehicle Type. Next, (7) select and upload a photo as Evidence of the violation. Finally, (8) click the Submit button to finalize and record the violation ticket.



Before tapping the Submit button, the violator is required to provide their signature as confirmation. A signature field will appear, allowing the violator to sign using a touchscreen or input device. If needed, (1) the Clear button can be used to erase and re-enter the signature. Once the signature is complete, (2) tap Done to proceed and finalize the submission of the violation ticket.

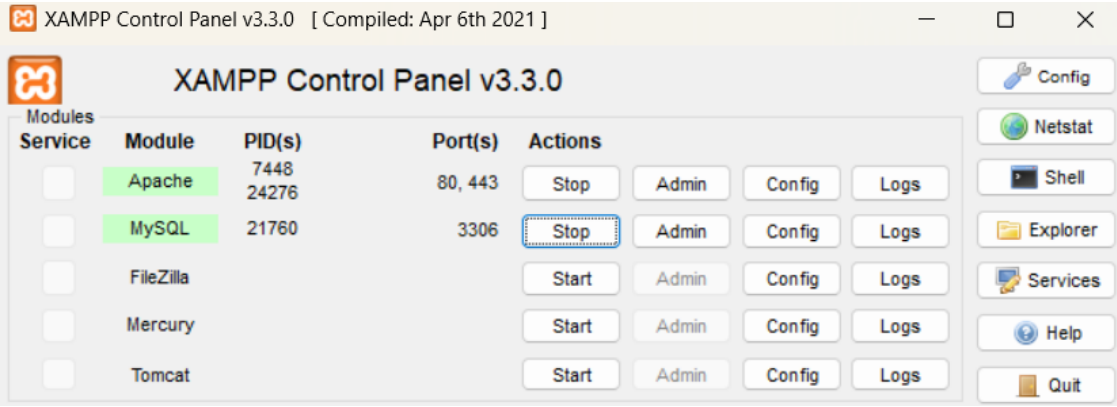


From the Your Tickets section on the homepage displayed after signing in, the enforcer can view ticket records. Tapping on this section will show two options: (1) All, which displays all violation tickets issued, and (2) My History, which shows only the tickets personally issued by the logged-in enforcer.

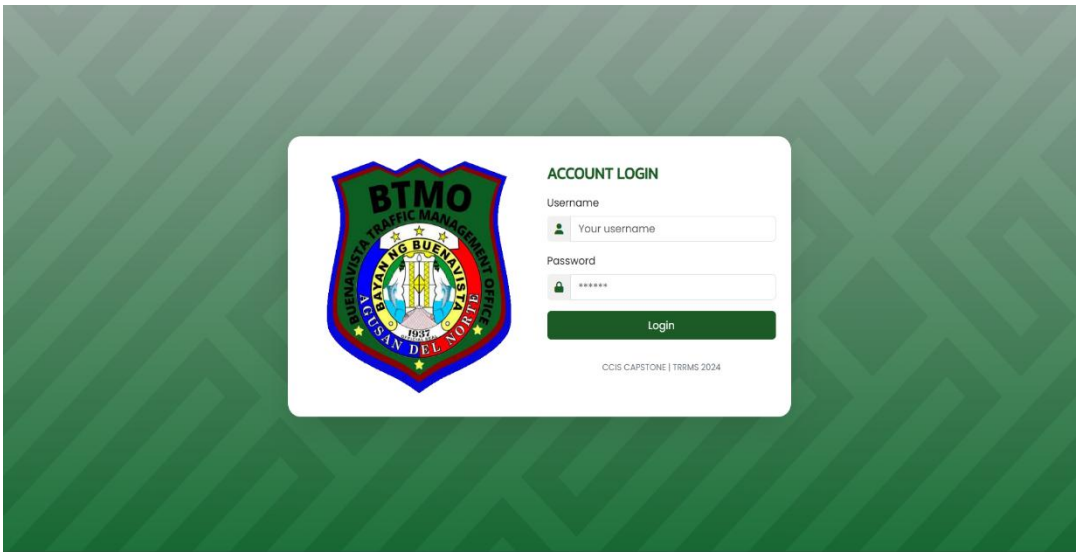


Within the My History section, tapping on a specific ticket will display its full details. This includes the Violator Details, Violations & Offenses, and Vehicle Details related to the issued ticket. At the bottom of the screen, the enforcer will have two options: (1) Reprint, to print the ticket again if needed, and (2) Close, to exit the ticket view and return to the previous screen.

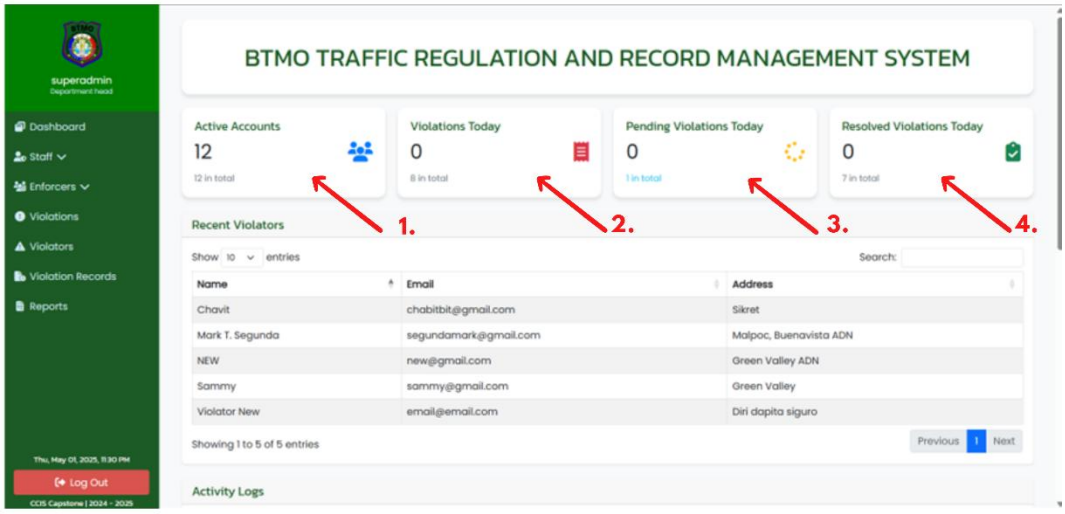
WEB SYSTEM



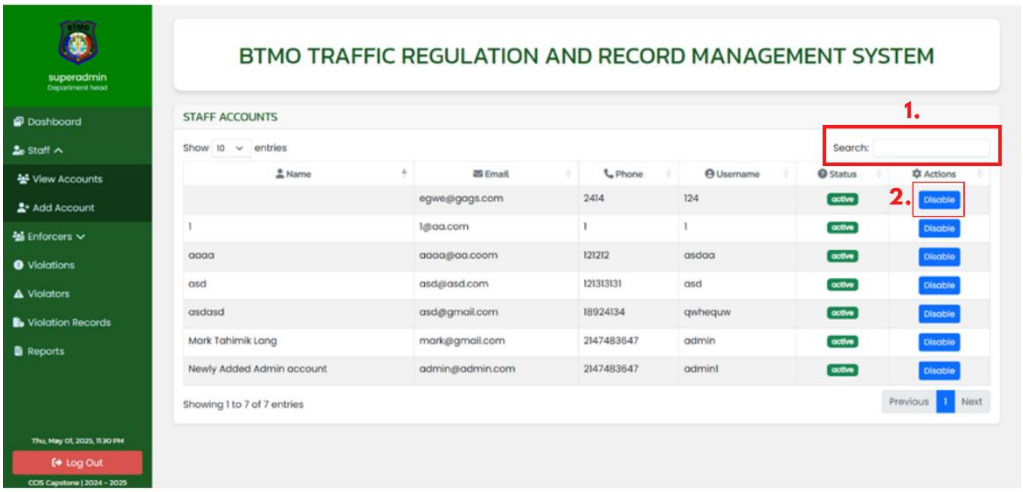
Ensure that both Apache and MySQL in XAMPP are running before launching TRRMS via the desktop shortcut or by entering http://localhost/ in your default browser. When the stop buttons for Apache and MySQL are visible, it means they are active. Once confirmed, the system and database are ready for use.



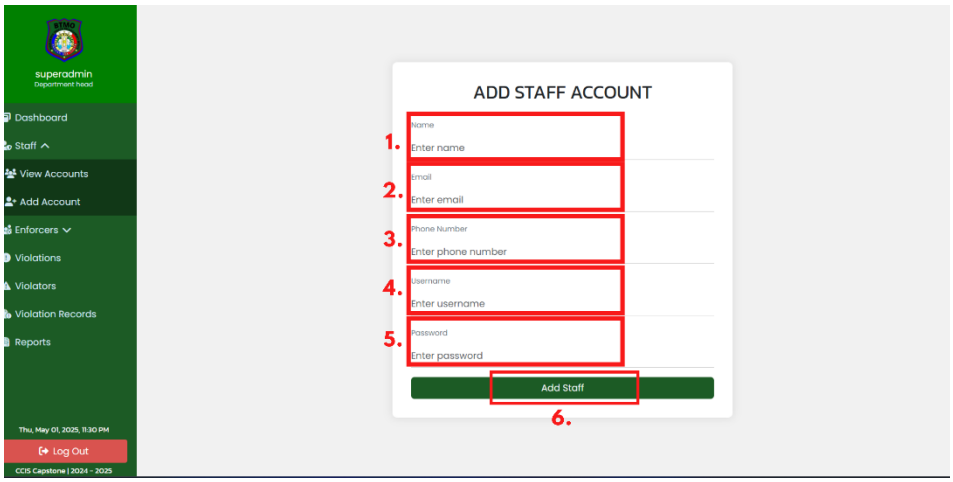
This is the login page of both BTMO Head and the Staff. The staff can only access the system if they have been provided an account by the BTMO Head. This ensures that only authorized individuals can access the organization's data on TRRMS.



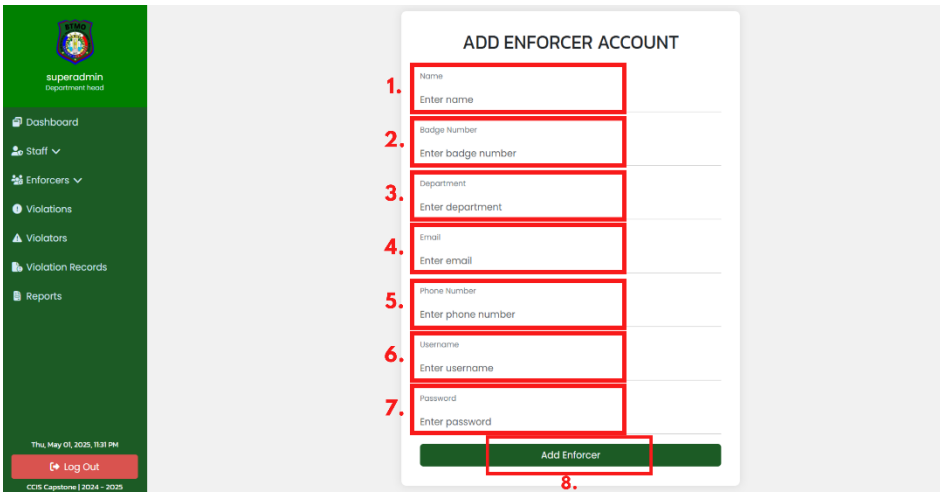
After logging in, the user will be automatically redirected to the Dashboard page. The (1) Active Accounts section displays the total number of active accounts, (2) Violations Today shows the total number of violations reported for the day, (3) Pending Violations Today tracks unresolved violations, (4) Resolved Violations Today shows the number of violations that have been addressed.



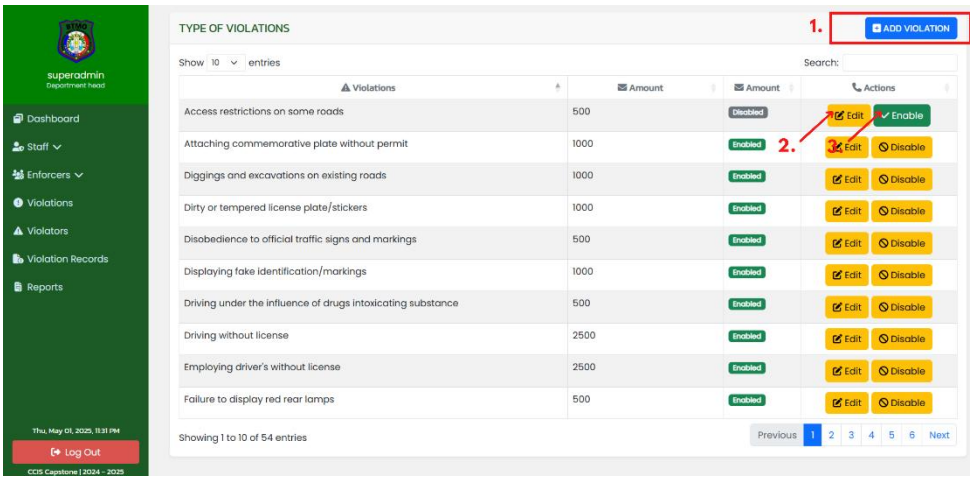
The View Staff Account page displays a list of all registered staff accounts. This page allows for easy management of staff information. (1) The Search feature enables quick filtering of accounts by name or other details. (2) The Disable button is used to deactivate a specific staff account when necessary.



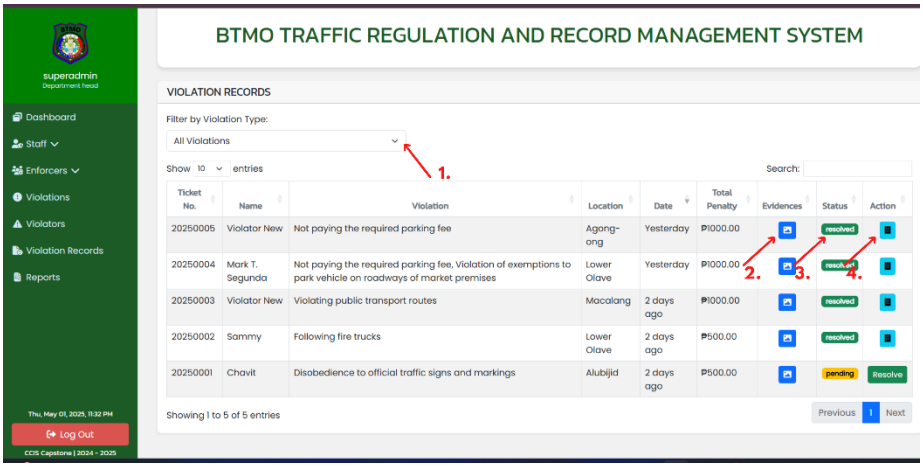
This is the UI for adding a staff account, which can only be done by the BTMO Head. To create a staff account: (1) Enter the staff member's name, (2) Enter their email address, (3) Enter their phone number, (4) Enter a unique username, (5) Enter a secure password, and (6) Click the "Add Staff" button to automatically create the account.



This is the UI for adding an enforcer account, which can be done by both the BTMO Head and staff. To add an enforcer account: (1) Enter the enforcer's name, (2) Enter their badge number, (3) Select the department, (4) Enter their email address, (5) Enter their phone number, (6) Enter a username, (7) Enter a password, and (8) Click the "Add Enforcer" button to create the account.

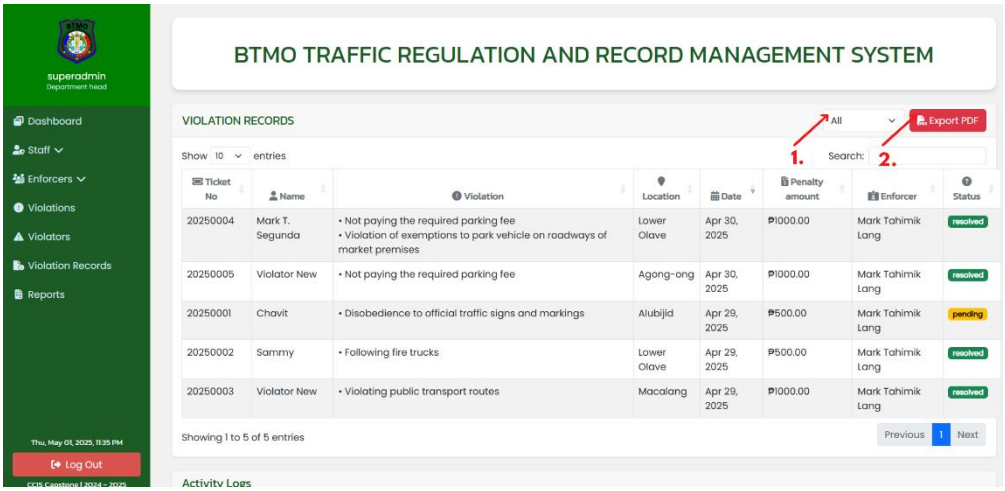


The Violations page displays a list of all defined types of violations within the system. It serves as a central area for managing violation records. (1) The Add Violation button allows the user to create a new type of violation. (2) The Edit option is used to modify existing violation details, and (3) the Enable/Disable toggle allows users to activate or deactivate specific violations as needed.



The Violation Records page provides a detailed overview of all recorded violations. (1) The All Violations filter allows users to sort and view records based on specific violation types. (2) The View Evidence option lets users open and review the uploaded images related to each violation. (3) The Status column indicates whether a violation is marked as Resolved or Pending. (4) Under

Action, users can enter the OR Number (Official Receipt) to update and mark a violation as resolved.



The Reports page serves as a summary dashboard for viewing compiled violation records. (1) Users can view violation data as Weekly or Monthly summaries for easier tracking and analysis. (2) The Export PDF button allows users to generate and download a PDF version of the report for documentation or printing purposes.

C. Letter of Permission

Saint Michael College of Caraga
Brgy. 4, Nasipit, Agusan del Norte, Philippines
Tel. Nos. +63 085 343-3251 / +63 085 283-3113 Fax No. +63 085 808-0892
www.smccnasipit.edu.ph

December 12, 2024

Cherlou C. Tomoling
Head Designate
Buenavista Traffic Management Office
Barangay 3 Poblacion, Buenavista, Agusan del Norte

Dear Sir,

Greetings!

We, the undersigned, are currently working on our capstone project titled **"Buenavista Traffic Management Office Regulation and Record Management System"** as part of the requirements for the Bachelor of Science in Information Technology degree. This study aims to comprehensively design, develop, and implement an automated Traffic Regulation and Record Management System (TRRMS).

In line with this, we respectfully seek your permission **to conduct interviews with you, staff, and traffic enforcers of BTMO. These interviews aim to gather insights into the challenges associated with the manual recordkeeping process and explore how these issues can be addressed through automation and paperless solutions.**

We believe your approval of this request will significantly contribute to the success of our research and, ultimately, help in providing a valuable solution to streamline this process.

Thank you for considering our request. We are hopeful for your favorable response.

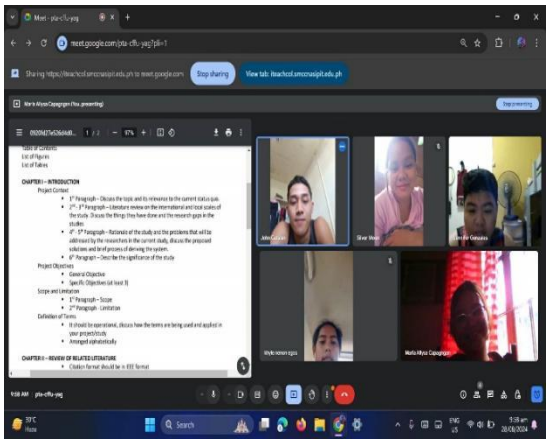
Sincerely,
Maria Allyssa D. Capagngan
Research Leader

Received by:
Cherlou C. Tomoling
Head Designate

D. Documented Undertakings

Researchers passed the title, “Buenavista Traffic Management Office (BTMO) Traffic Regulation and Record Management System, during the Title hearing.

Researchers gained knowledge by attending the seminar-workshop titled 'Empowering Minds & Writing Skills: A Seminar-Workshop on Integrating Mendeley in Writing a Review of Related Literature.



Researchers during their online meeting by discussing and formulating the Chapters 1-3.

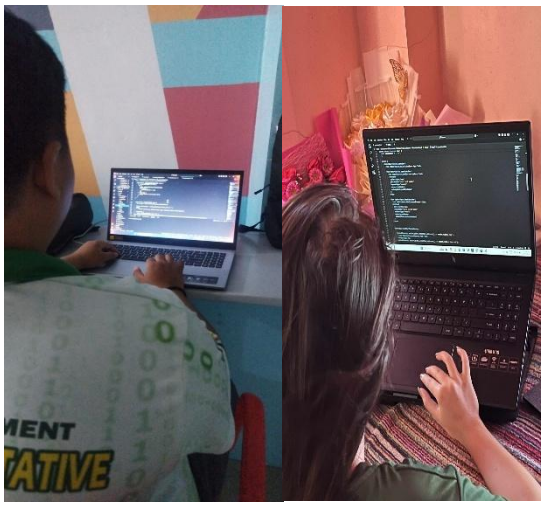
Researchers conducted a presentation run-through and a thorough discussion the night before the scheduled Title Proposal.

The researchers successfully presented and defended Chapters 1 to 3 during the Proposal Defense, demonstrating a clear understanding of the project's background, objectives, and significance. They effectively explained the scope and limitations of the study, and addressed questions and feedback from the panel, laying a strong foundation for the development of the system.

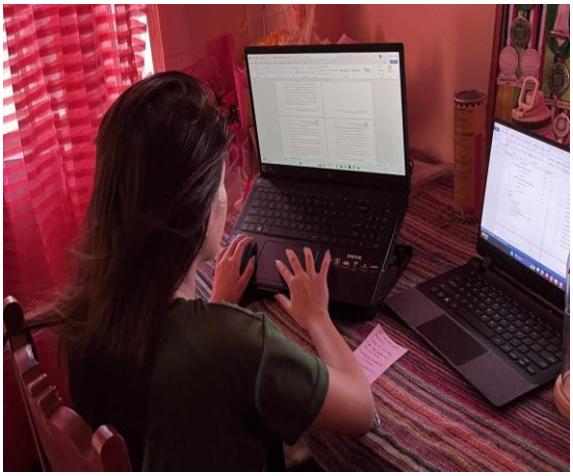
The researchers, together with their esteemed research adviser, Mr. Ronnel Falo, are actively engaged in discussions regarding the process for implementing necessary revisions and the proper routing of documents. These discussions aim to ensure that all corrections are addressed thoroughly and that the updated research components follow the correct submission and approval procedures.

The researchers are currently focused on revising the manuscript to incorporate the feedback and suggestions provided during the evaluation process. This step is essential to enhance the overall quality and clarity of the research. Once the revisions are completed, the manuscript will be prepared for proper routing through the required channels for review and approval, ensuring that all academic and institutional standards are met.

During the routing and revision process, the researchers are coordinating closely to ensure timely revisions and smooth communication within the group. Regular discussions help keep everyone aligned and ensure tasks are completed efficiently. This collaborative approach ensures that all feedback is addressed and the project moves forward without delays.



In the process of developing the app and web system, the researchers were responsible for designing the overall structure, coding and integrating the core features, and managing database connections. They also created technical diagrams and conducted thorough testing and refinement to ensure the system met all functional requirements.



During the development of Chapter 4, the researchers structured the code, included detailed discussions, and recorded the results from user evaluations. They analyzed the system's performance and shared insights based on the feedback gathered from end users.

The researchers, together with their adviser, Mr. Ronnel A. Falo, regularly reviewed the progress of the system to ensure it was on track. During these evaluations, Mr. Falo provided valuable feedback and suggestions to help improve the system and guide the researchers in enhancing its overall functionality and performance.

The researchers, in coordination with their target beneficiary led by BTMO Head Mr. Cherlou C. Tomoling, conducted a review and evaluation of the system to ensure its alignment with the operational needs of the office. This collaborative assessment focused on verifying that the system supports field and office efficiency, particularly in managing traffic-related activities and streamlining daily workflows.

The researchers visited the Buenavista Traffic Management Office (BTMO) to assess the system's performance in real-world conditions and gather feedback from the staff for further improvements. This visit helped ensure the system aligns with the office's operational needs and efficiency.

The researchers conducted pilot testing of the system with 6 enforcers, 1 staff member, and 1 head from the Buenavista Traffic Management Office (BTMO). This testing allowed them to evaluate the system's functionality and gather feedback from key users to identify any issues and areas for improvement.

The researchers gathered to study and review their project, focusing on key elements to ensure they were fully prepared for the upcoming final defense. They worked together to refine their presentation and address any remaining details before the final presentation.

The researchers successfully passed and defended their capstone research during the final oral defense, presenting their system to the panel for assessment. They also demonstrated the debugging process, addressing any technical issues and ensuring the system met the required standards before receiving approval.

After the final defense, the researchers began the process of revising their manuscript based on the panel's recommendations and refining the system for routing to address identified limitations. This included enhancing system functionality, optimizing performance, and ensuring the design better aligned with real-world operational requirements.

The researchers consulted with their adviser, Sir Falo, to seek guidance and feedback on both the manuscript and the routing system. During the meeting, they discussed necessary revisions, clarified technical aspects, and ensured that both components met academic standards and practical requirements.

The researchers began the routing process of their revised manuscript and system to the other panel members for further review and approval.

The researchers sought guidance from the SMCC Research and Instructional Innovation Department (RIILD) to enhance the efficiency of their system and to have their final manuscript reviewed for accuracy and completeness.

The researchers deployed their system at the Electronic Data Processing (EDP) Office of Saint Michael College of Caraga, where it was officially acknowledged and signed by John Louie R. Abobo (MIS In-Charge) and Engr. Marisol S. Rosario (EDP Head).

The researchers finally deployed and implemented their system to their Capstone system beneficiary, the Buenavista Traffic Management Office (BTMO), headed by Mr. Cherlou C. Tomoling (BTMO Head Designate). The deployment was conducted in coordination with the CCIS-CES Coordinator, Mr. Jessie Mahinay, as part of the community extension initiative.

Mr. Cherlou C. Tomoling (BTMO Head Designate) evaluated the system using the official evaluation form to rate and assess its overall functionality, usability, and effectiveness. This assessment aimed to determine how well the deployed system addressed the operational needs of the Buenavista Traffic Management Office (BTMO).

Mrs. Wendell B. Gonzaga took on the responsibility of reviewing and proofreading the final manuscript due to the absence of Ms. Felmarie Manlunas, the designated human grammarian. Her contribution was essential in ensuring the manuscript's grammatical accuracy and overall quality.

E. Certificate of Implementation and Extension

Republic of the Philippines
Province of Agusan del Norte
BUENAVISTA TRAFFIC MANAGEMENT OFFICE
Municipality of Buenavista

CERTIFICATE
OF IMPLEMENTATION

This certificate is presented to:

Maria Allysa D. Capagngan
Jennifer M. Rosales
John B. Catalan
Khyle Nenon Egos
Deborah Grace Jumawan

Implemented their Capstone Project titled
"Buenavista Traffic Management Office Traffic Regulation And Record Management System"

Given this 16Th day of May in the year 2025, at Buenavista Traffic Management Office, Buenavista Agusan del Norte

CHERLOU C. TOMOLING
BTMO HEAD

Republic of the Philippines
Province of Agusan del Norte
BUENAVISTA TRAFFIC MANAGEMENT OFFICE
Municipality of Buenavista

CERTIFICATE
OF EXTENSION

This certificate is presented to:

Maria Allysa D. Capagngan
Jennifer M. Rosales
John B. Catalan
Khyle Nenon Egos
Deborah Grace Jumawan

For extending their Capstone Project titled
"Buenavista Traffic Management Office Traffic Regulation And Record Management System"
at Buenavista Traffic Management Office(BTMO), Buenavista, Agusan del Norte for School Year 2024-2025.

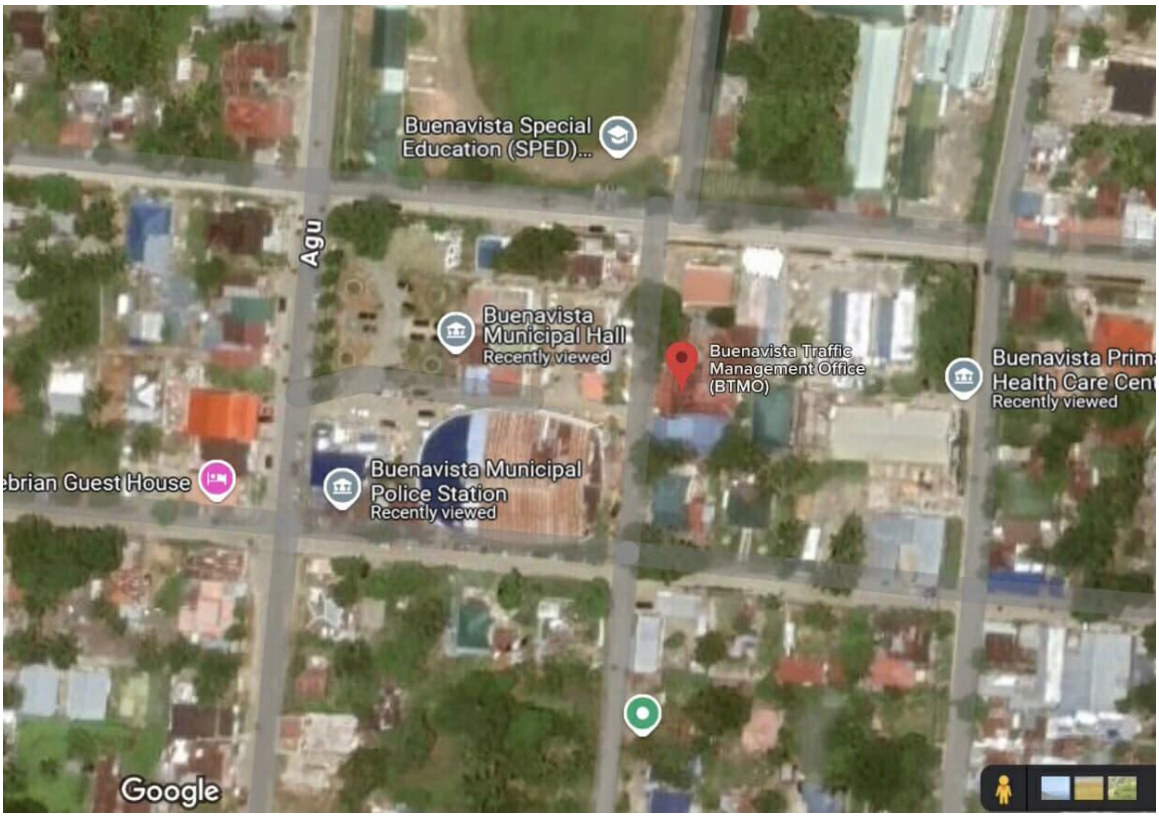
This certificate is issued upon their request for whatever purpose it may serve.
Done this 16th day of May in the year 2025, at Buenavista Traffic Management Office. Buenavista, Agusan del Norte

JESSIE S. MAHINAY
CCIS- CES Coordinator

CHERLOU C. TOMOLING
BTMO HEAD

F. Map of the Research Locale

BUENAVISTA TRAFFIC MANAGEMENT OFFICE (BTMO)



G. CURRICULUM VITAE

NAME: Maria Allysa D. Capagngan

ADDRESS: D-5, Barangay Sacol, Buenavista, ADN

EMAIL: mariaallysa_capagngan@smccnasipit.edu.ph

Personal Information

Sex : Female

Date of birth : April, 22, 2004

Height : 154 cm

Weight : 54 kg

Civil Status : Single

Educational Attainment

Elementary : Sacol Elementary School

Secondary : Saint James High School Buenavista Agusan Inc.

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELD
Michaelinian Cybersecurity Guild (MCG)	Member

G. CURRICULUM VITAE

NAME: John B. Catalan

ADDRESS: Barangay 4, Nasipit, ADN

EMAIL: john_catalan@smccnasipit.edu.ph

Personal Information

Sex : Male

Date of birth : February 24, 2004

Height : 168 cm

Weight : 56 kg

Civil Status : Single

Educational Attainment

Elementary : Las Nieves Elementary School

Secondary : Las Nieves National High School

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELD
N/A	

G. CURRICULUM VITAE

NAME: Khyle Nenon Egos

ADDRESS: District Asucena, Brgy. 7, Nasipit, ADN

EMAIL: khylenenon_egos@smccnasipit.edu.ph

Personal Information

Sex : Male

Date of birth : March 29, 2003

Height : 166 cm

Weight : 60 kg

Civil Status : Single

Educational Attainment

Elementary : Nasipit Central Elementary School

Secondary : Nasipit National Vocational School

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELD
N/A	

G. CURRICULUM VITAE

NAME: Deborah Grace U. Jumawan

ADDRESS: Gemelina Talisay, Nasipit, ADN

EMAIL: deborahgrace_jumawan@smccnasipit.edu.ph

Personal Information

Sex : Female

Date of birth : June 24, 1999

Height : 161 cm

Weight : 73 kg

Civil Status : Single

Educational Attainment

Elementary : Talisay Elementary School

Secondary : Saint Michael College of Caraga

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELD
N/A	

G. CURRICULUM VITAE

NAME: Jennifer M. Rosales

ADDRESS: Brgy 4, Nasipit Agusan Del Norte

EMAIL: jennifer_rosales@smccnasipit.edu.ph

Personal Information

Sex : Female

Date of birth : July 23, 2004

Height : 152 cm

Weight : 72 kg

Civil Status : Single

Educational Attainment

Elementary : San Pedro Elementary School

Secondary : San Vicente National High School

Tertiary : Saint Michael College of Caraga

Membership/Affiliations

ORGANIZATION	POSITIONS HELD
CCISLSG	3rd Year Representative

CERTIFICATE OF TECHNOLOGY BASED ASSESSMENT

Saint Michael College of Caraga
Brgy. 4, Nasipit, Agusan del Norte, Philippines
Tel. Nos. +63 085 343-3251 / +63 085 283-3113
www.smccnasipit.edu.ph

TEST RESULT

TECHNOLOGY-BASED QUALITY ASSURANCE

Date of Test	MAY 15, 2025	**Control number**	**RBSIT124009**	
Research Title	**BUENAVISTA TRAFFIC MANAGEMENT OFFICE (BTMO) TRAFFIC REGULATION AND RECORD MANAGEMENT SYSTEM**			
Author (s)	1	CAPAGNGAN MARIA ALLYSA	**ORCID No.**	0009-0001-4820-5887
	2	ROSALES JENNIFER	**ORCID No.**	0009-0002-3080-3114
	3	CATALAN JOHN	**ORCID No.**	0009-0008-6306-9183
	4	JUMAWAN DEBORAH GRACE	**ORCID No.**	0009-0002-6275-8127
Co-author (s)	1	FALO RONNEL	**ORCID No.**	0000-0003-1876-5173

Contact Person	CAPAGNGAN MARIA ALLYSA
Address	D-5 CABULAKAN, BRGY. SACOL, BUENAVISTA, AGUSAN DEL NORTE
Mobile Number	09129197140

Test subject	Chapters 1-3				
Total number of words	**10,689**	**Sentences**	**703**	**Characters**	**73,794**

	Test Area	Score	Required	Remarks
	(1) Plagiarism	98	90%	PASSED
	(2) Grammar	97	95%	PASSED
	(3) Readability Score	25	25	PASSED
	(4) Reliability *(Test Instrument only)*	N/A	N/A	

FINDING(S) & RECOMMENDATION(S)

SEE HUMAN GRAMMARIAN FOR THE REVISE SENTENCES

Evaluated by:	Approved by:
MICHELLE ANN G. LUCINO	**KENNETH IAN B. BARRERA, MA**
Signature over Printed Name	Head of Research

Member: info@smccnasipit.edu.ph

Saint Michael College of Caraga
Atupan St., Nasipit, Agusan del Norte, Philippines

Certificate of Human Grammarian

This is to certify that the research titled

Buenavista Traffic Management Office Regulation and Record Management System

with authors

Maria Allysa D. Capagngan, Jenniper M. Rosales, John B. Catalan, Khyle Neron Egos and Deborah Grace U. Jumawan

had passed the evaluation by the undersigned as the **HUMAN GRAMMARIAN** and recommended for approval.

This certificate is given on the 16th day of May 2025 at Research and Instructional Innovation Department.

WENDELL B. CONTRERAS
Human Grammarian
Saint Michael College of Caraga

Member: PHILIPPINE SOCIETY FOR QUALITY, INC.

CERTIFICATE OF AUTHENTIC AUTHORSHIP

This is to certify that I/we, the undersigned author(s), have solely and honestly written the entire paper titled: "**Buenavista Traffic Management Office Regulation and Record Management System**" All contents, ideas, data, and analysis presented in this paper are the results of my/our own diligent work and intellectual effort. The paper has been crafted to the best of my/our abilities, in full compliance with academic integrity standards and ethical research practices.

I/We further declare that:

1. This paper is free from any form of plagiarism or copyright infringement.
2. All referenced materials have been properly cited, and due credit has been given to the original authors and sources.
3. The work does not contain any falsified or fabricated data or information.

I/We acknowledge that if any rules or laws related to copyright, plagiarism, or other related legal or ethical standards are violated, Saint Michael College of Caraga shall not be held accountable for my/our actions. Any legal, academic, or ethical consequences arising from such violations shall solely be my/our responsibility.

Signed this
Date: May 1, 2025
Location: Saint Michael College of Caraga

Author(s):

Maria Allysa D. Capagngan
Author

John B. Catalan
Author

Khyle Nenon Egos
Author

Deborah Grace U. Jumawan
Author

Jennifer M. Rosales
Author