M.Kumarasamy
College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

A Project Report

on

E-CHALLAN GENERATOR AND TRAFFIC VIOLATION TRACKER

Submitted in partial fulfilment of requirements for the award of the course

of

CGB1201 – JAVA PROGRAMMING

Under the guidance of

Mrs.M. SARATHA., B.Tech., M.E.,

Assistant Professor/AI

Submitted By

BHUVANASRI T (927624BAM006)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

M.KUMARASAMY COLLEGE OF ENGINEERING
(Autonomous)

KARUR – 639 113

DECEMBER 2025

M.Kumarasamy College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on **"E-CHALLAN GENERATOR AND TRAFFIC VIOLATION TRACKER"** is the bonafide work of **BHUVANASRI T (927624BAM006)** who carried out the project work during the academic year 2025- 2026 under my supervision

Signature	Signature
Mrs. M.SARATHA, B.Tech.,M.E.,	**Dr. A.SELVI .,Ph.D.,**
SUPERVISOR,	**HEAD OF THE DEPARTMENT,**
Department of Artificial Intelligence,	Department of Artificial Intelligence,
M. Kumarasamy College of Engineering,	M. Kumarasamy College of
Engineering,	
Thalavapalayam, Karur -639 113.	Thalavapalayam, Karur -639 113.

Programme: B.E. CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

Vision of the Department:

To become a renowned hub for AIML technologies to producing highly talented globally recognizable technocrats to meet industrial needs and societal expectation.

Mission of the Department:

M1: To impart advanced education in AI and Machine Learning, built upon a foundation in Computer Science and Engineering.

M2: To foster Experiential learning equips students with engineering skills to tackle real-world problems.

M3: To promote collaborative innovation in AI, machine learning, and related research and development with industries.

M4: To provide an enjoyable environment for pursuing excellence while upholding strong personal and professional values and ethics.

Programme Educational Objectives (PEOs):

Graduates will be able to:

PEO 1: Excel in technical abilities to build intelligent systems in the fields of AI & ML in order to find new opportunities.

PEO 2: Embrace new technology to solve real-world problems, whether alone or as a team, while prioritizing ethics and societal benefits.

PEO 3: Accept lifelong learning to expand future opportunities in research and product development.

Mapping of Programme Educational Objectives with Mission of the Department:

PEOs / Department Mission Statements	M1	M2	M3	M4
PEO1	3	3	2	3
PEO2	3	3	2	2
PEO3	3	2	3	2

1: Slight (Low) **2: Moderate (Medium)** **3: Substantial (High)**

Programme Outcomes (POs):

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective , make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes (PSOs):

PSO1: Expertise in tailoring ML algorithms and models to excel in designated applications and fields.

PSO2: Ability to conduct research, contributing to machine learning advancements and innovations that tackle emerging societal challenges.

Mapping of Programme Educational Objectives with Programme Outcomes and Programme Specific Outcomes:

PEOs / POs & PSOs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
PEO1	3	2	2	2	3	2	3	3	1	2	3	1	3	1
PEO2	2	2	3	2	3	3	3	2	2	3	2	3	3	2
PEO3	3	3	2	3	3	2	3	3	3	2	3	3	2	3

1: Slight (Low) **2: Moderate (Medium)** **3: Substantial (High)**

ABSTRACT

The E-Challan Generator and Traffic Violation Tracker is a Java-based system designed to digitalize traffic enforcement by automating the process of issuing challans and recording violations. The application allows traffic officers to enter violation details, calculate fines automatically, generate instant challans, and maintain accurate digital records of drivers, vehicles, and payment status. By reducing manual paperwork and human error, the system ensures transparency, faster processing, and efficient tracking of unpaid challans. With a secure login system and a user-friendly interface, the project enhances road safety management and serves as a foundation for future upgrades such as ANPR camera integration, GPS tagging, and online payment support.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs	PSOs
The E-Challan Generator and Traffic Violation Tracker is a Java-based application that automates issuing traffic challans and recording violations. It allows officers to enter violation details, calculate fines instantly, generate digital challans, and track payment status. The system reduces manual work, minimizes errors, and improves transparency in traffic enforcement through a secure and user-friendly interface.	PSO2 PSO3 PSO5 PSO9 PSO10	PSO1 PSO2

SUPERVISOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

Managing traffic violations manually is often slow, error-prone, and difficult to maintain, especially as the number of vehicles continues to increase. An E-Challan Generator and Traffic Violation Tracker built using Java provides an efficient and systematic solution to automate this process. The system allows traffic officers to record violations, calculate fines based on predefined rules, and instantly generate digital challans. It maintains organized records of drivers, vehicles, violation types, and payment status, ensuring accuracy and transparency. By replacing manual paperwork with an automated application, the project reduces human error, speeds up enforcement activities, and improves consistency in penalty management. This technology-driven approach supports better traffic regulation, enhances road safety, and makes the challan management process more reliable and user-friendly.

1.2 Objective

- To develop a system that automatically generates e-challans based on predefined traffic violation rules and fine structures.
- To ensure accurate tracking of violations by maintaining organized records of drivers, vehicles, challans, and payment status.
- To create a structured data management system that supports efficient storage, retrieval, and categorization of traffic violations.
- To minimize manual effort and reduce human errors commonly seen in traditional challan issuing processes.
- To provide traffic authorities with a reliable, fast, and user-friendly tool .

1.3 JAVA concepts used

- **Object-Oriented Programming (OOPS):**
  The project is built entirely on OOP principles. Classes such as Driver, Vehicle, Violation, and Challan Generator help organize the system into clear modules, making the application easier to maintain, extend, and scale.
- **Collections Framework**:
  This forms the backbone of storing and managing challan records. Lists, Maps, and ArrayLists are used to maintain data such as violation logs, driver details, and payment status for quick retrieval and efficient tracking.
- **File Handling or Database Connectivity (JDBC):**
  Used to store and retrieve challan details, violation types, and user records. Whether the application uses files or a database, these concepts handle all important read/write operations that keep the system's data updated and accessible.
- **Exception Handling:**
  Ensures smooth functioning of the system. Issues like invalid inputs, missing data, or incorrect file access are handled gracefully without crashing the application, improving reliability during real-time usage by traffic officers.
- **Java Swing (GUI Framework):**
  Used to design the user interface for officers. Swing components like JFrame, JPanel, JButton, JTextField, and JTable help create a clean, interactive interface for issuing challans and viewing records.
- **Event Handling :**

  Manages actions such as button clicks, form submissions, and login operations. This allows the system to respond instantly to user inputs.

CHAPTER 2
PROJECT METHODOLOGY

2.1 Requirement Analysis

Objective

To understand the needs of traffic authorities and identify the functional and non-functional requirements of the E-Challan Generator and Traffic Violation Tracker system.

Tasks

- Interact with traffic officers to understand challenges in manual challan issuing
- Identify required violation categories, fine structures, and record-keeping needs
- Analyze existing challan formats, enforcement procedures, and documentation methods
- Document all user expectations, system constraints, and workflow limitations

Output

- A clearly defined requirements specification document
- List of features such as challan generation, violation tracking, payment status, and secure login
- Strong foundation for system design and implementation

2.1.1.System Design

Objective

To create a structured and modular architecture for the e-challan system to ensure efficient data handling and smooth user interaction.

Tasks

- Prepare class diagrams, flowcharts, and system block diagrams
- Define interaction between User (Officer), Violation Module, Challan Generator, and Record Management modules
- Choose appropriate data structures (Lists, HashMaps) for storing violations, vehicles, and challan information
- Decide the format for storing data (File handling or JDBC database connectivity)
- Design UI structure using Java Swing components

Output

- Complete system architecture layout
- Module-wise design documentation outlining component interactions
- A blueprint for coding, validation, and integration

2.1.2 . Violation Database creation:

Objective

To build a well-organized and searchable repository of traffic violations and fine details.

Tasks

- Create the question data model (text, marks, difficulty, topic)
- Add all standard traffic violations to the system (speeding, no helmet, signal jump, etc.)
- Categorize violations based on severity, type, and applicable fine rules
- Implement methods for add, edit, delete, and search violations
- Ensure proper mapping between vehicle, driver, and violation records

Output

- A structured and fully functional Violation Database
- Easy retrieval of violation details based on rule categories
- Reliable backend for generating accurate and consistent e-challans

2.1.3 . Challan Rule Configuration

Objective

To define the rules and parameters required for generating e-challans accurately based on traffic violations.

Tasks

- Create fields for violation categories, fine amounts, severity levels, and penalty descriptions
- Allow users (traffic admin) to add, update, or remove violation rules
- Validate inputs to ensure correct fine mapping (e.g., fines must match the violation severity and category)
- Enable saving and loading of rule templates for future use
- Ensure rule updates automatically reflect in challan generation

Output

- A complete rule configuration file or object
- Clear fine structure ready for challan generation
- Flexibility to modify and adapt to changing traffic policies

2.1.4. Algorithm Development for Challan Generation

Objective

To design the logic that automatically generates accurate e-challans based on the recorded traffic violations and predefined fine rules.

Tasks

- Filter violation records based on vehicle number, driver details, violation type, and severity
- Apply generation logic using Java Collections to fetch corresponding fine amounts and rule details
- Prevent duplication of challans for the same violation instance
- Ensure the challan contains all mandatory fields such as date, time, location, officer ID, and penalty amount
- Validate that the generated challan reflects the correct fine structure according to configured rules

Output

- Automatically generated e-challan
- Consistent, rule-based challan creation with accurate fine mapping
- Fast and error-free alternative to manual challan issuing

2.1.5 Validation Process

Objective

To ensure that every generated e-challan is accurate, complete, and fully aligned with the predefined traffic rules and system requirements.

Tasks

- Verify that the violation details (type, severity, date, location) are correctly entered
- Check that the calculated fine amount matches the configured rule set
- Confirm that the vehicle and driver records exist and are valid
- Prevent duplicate challans for the same violation instance

Output

- A validated and approved e-challan ready for issue
- Clear error messages for incorrect, missing, or conflicting violation data
- Higher accuracy, transparency, and system reliability during challan generation

2.1.6 Output Formatting and Generation

Objective

To generate a clear, professional, and readable e-challan document that can be issued to violators and stored for official records.

Tasks

- Arrange challan details in a structured format with proper headings and numbering
- Format fields such as violation type, fine amount, vehicle details, date, time, and officer ID
- Ensure alignment, spacing, and layout follow official challan standards
- Export the challan as a printable document (PDF, text file, or on-screen receipt)
- Generate a digital copy for database storage and future reference

Output

- A well-formatted and professional e-challan.
- Printable and digitally stored challan document.
- Clear presentation suitable for official traffic enforcement use

2.1.7 Testing

Objective

To ensure the e-challan system works correctly, remains stable under different conditions, and provides a smooth user experience.

Tasks

- Perform functional testing for all modules such as violation entry, challan generation, login, and payment tracking
- Test challan generation with different violation types, fine amounts, and driver/vehicle scenarios
- Conduct validation testing to ensure prevention of duplicate challans and incorrect data
- Fix bugs, resolve inconsistencies, and optimize system performance

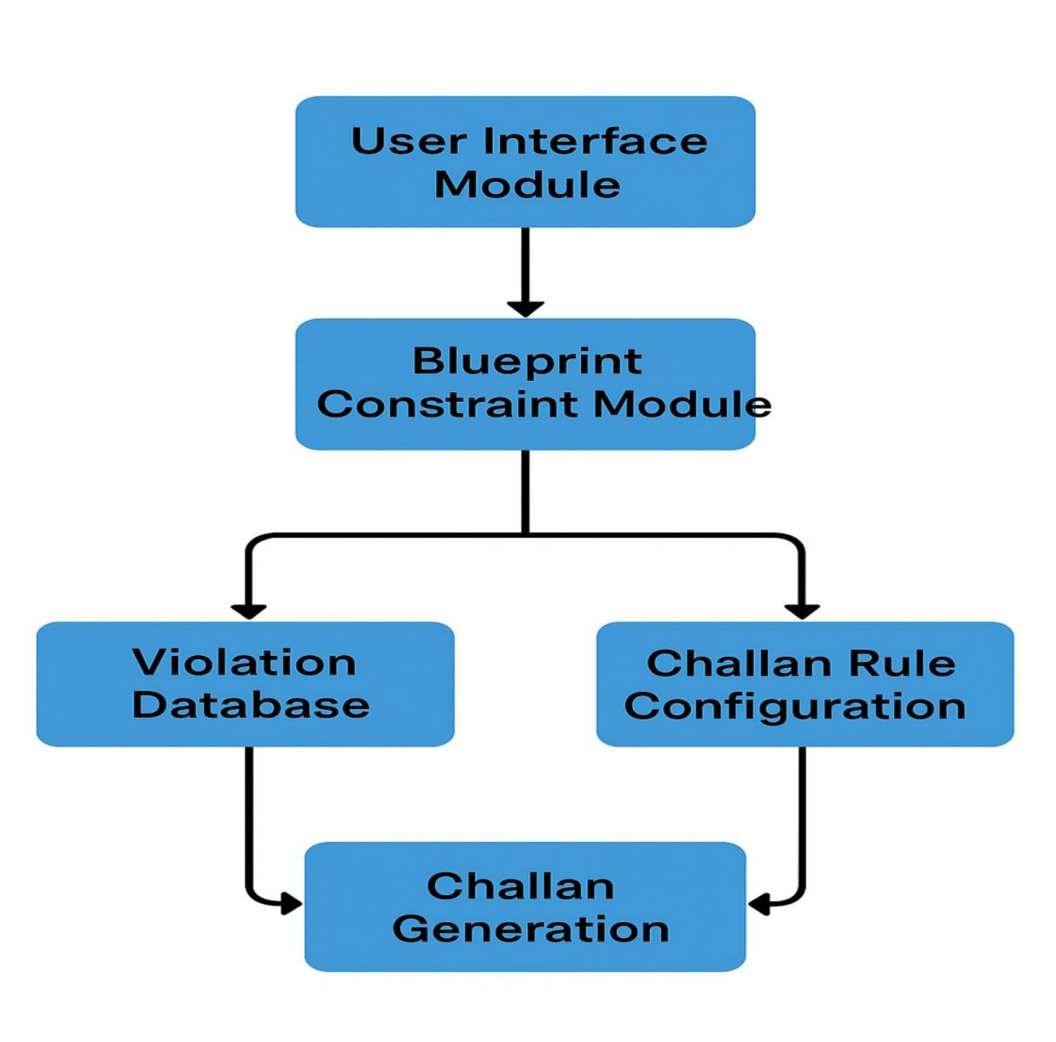
- Carry out usability testing to ensure the interface is user-friendly for traffic officers

Output

- A stable, bug-free, and reliable e-challan management system
- Verified accuracy of violation-based fine calculation and challan generation
- Enhanced performance and improved user experience for real-time usage

2.2Block Diagram

8



CHAPTER 3
MODULES

3.1 User module:

Purpose

- To manage all user interactions and provide secure access to the system’s functionality for traffic officers and administrators.
- To allow officers to input violation details, access violation rules, initiate challan generation, and view or update records.

Features

- Officer login and role-based access — secure authentication to restrict features by role (officer, admin).
- Violation entry form — easy input for vehicle number, violation type, date/time, location, and officer ID.
- Navigation to modules — quick links to Violation Database, Rule Configuration, Challan Generation, Payment Tracking, and Reports.
- Search and retrieval — search by vehicle number, challan ID, or date range.
- View and update records — view challan history, mark payments, and edit incorrect entries (with audit log).
- Simple, guided workflow — minimal screens and clear labels so non-technical users can operate quickly.

Technical Requirements

- GUI framework: Java Swing (or JavaFX) for desktop GUI; alternatively console I/O for a simple prototype.
- Input validation: client-side and server-side checks for required fields, valid vehicle number formats, and numeric fine amounts.
- Backend integration: interfaces/classes to call Violation Database, Rule Engine, Validation Layer, and Challan Generator modules.
- Security: password hashing, session management, and role-based checks.
- Logging & audit: record user actions for accountability (who issued/edited a challan and when).
- Error handling: graceful messages for missing data, DB errors, or duplicate challans.

3.2 Blueprint Module

Purpose

- To define the structure and rules used for generating accurate e-challans.
- To store essential constraints such as violation categories, fine amounts, severity levels, and penalty rules.

Features

- Create, edit, and save rule configurations for different types of violations.
- Validation of rule inputs to ensure correct fine mapping and proper data consistency.
- Ability to support multiple rule sets or fine structures (optional, based on requirements or policy updates).

Technical Requirements

- Java classes representing violation rules, fine properties, and severity categories.
- Use of Collections (Map, List) to store, organize, and manage all rule configurations.
- File or database storage to save, load, and update rule configuration templates.

3.3 Violation Database Module

Purpose

- To store and organize all traffic violation types and their associated rules.
- To provide fast and accurate retrieval of violation details based on category, severity, fine amount, or description.

Features

- Add, update, or delete violation entries.
- Categorization of violations by type (helmet violation, speeding, signal jump, parking offense, etc.), severity, and fine amount.
- Search and filtering options to quickly locate violations.
- Backend storage using file or database for long-term record management.
- Mapping between violation type and its penalty structure.

Technical Requirements

- Java Collections Framework (List, HashMap, ArrayList) to store and manage violations efficiently.
- JDBC (if using database) or File I/O (if using text/JSON/XML storage).
- Classes to represent violation objects, including attributes like violation ID, title, description, fine, and severity.

3.4 Challan Generation Module:

Purpose

- To automatically generate a complete e-challan based on recorded violation details and predefined rule configurations.
- To select the appropriate fine, format all violation data, and assemble them into a final challan document.

Features

- Logic to map violation details (vehicle number, violation type, severity) with the correct fine rules.
- Ensures accurate and consistent challan creation without duplication.
- Automatically fills challan fields such as challan ID, officer ID, date, time, location, and penalty amount.
- Generates a clean, structured challan ready for printing or storing in the system.
- Maintains record of all generated challans for future reference.

Technical Requirements

- Core Java logic to implement challan creation algorithms.
- Use of Collections, Java Streams, and filtering to fetch required violation and rule data efficiently.
- Strong integration with the Rule Configuration and Violation Database modules.
- Object-oriented programming principles for modular, scalable design.

3.5 Validation Module:

Purpose

- To ensure that every generated e-challan is accurate, complete, and follows all rule configurations.
- To alert officers if any violation details, fine rules, or driver/vehicle information are missing or incorrect.

Features

- Verify violation type, severity, fine amount, and required fields before generating challan.Check for duplicate challans for the same incident.
- Error reporting with clear messages when fields are incomplete or inconsistent.
- Mandatory verification before final challan generation.

Technical Requirements

- Conditional validation checks and exception handling
- Rule-verification logic using Java classes and Collections.
- Integration with Challan Generation and Rule Configuration modules.
- Input validation (format checking, empty field detection, and data type validation).

3.6 Output module

Purpose

- To generate the final e-challan in a clean, professional, and readable format.
- To provide export options for printing, saving, or displaying the challan.

Features

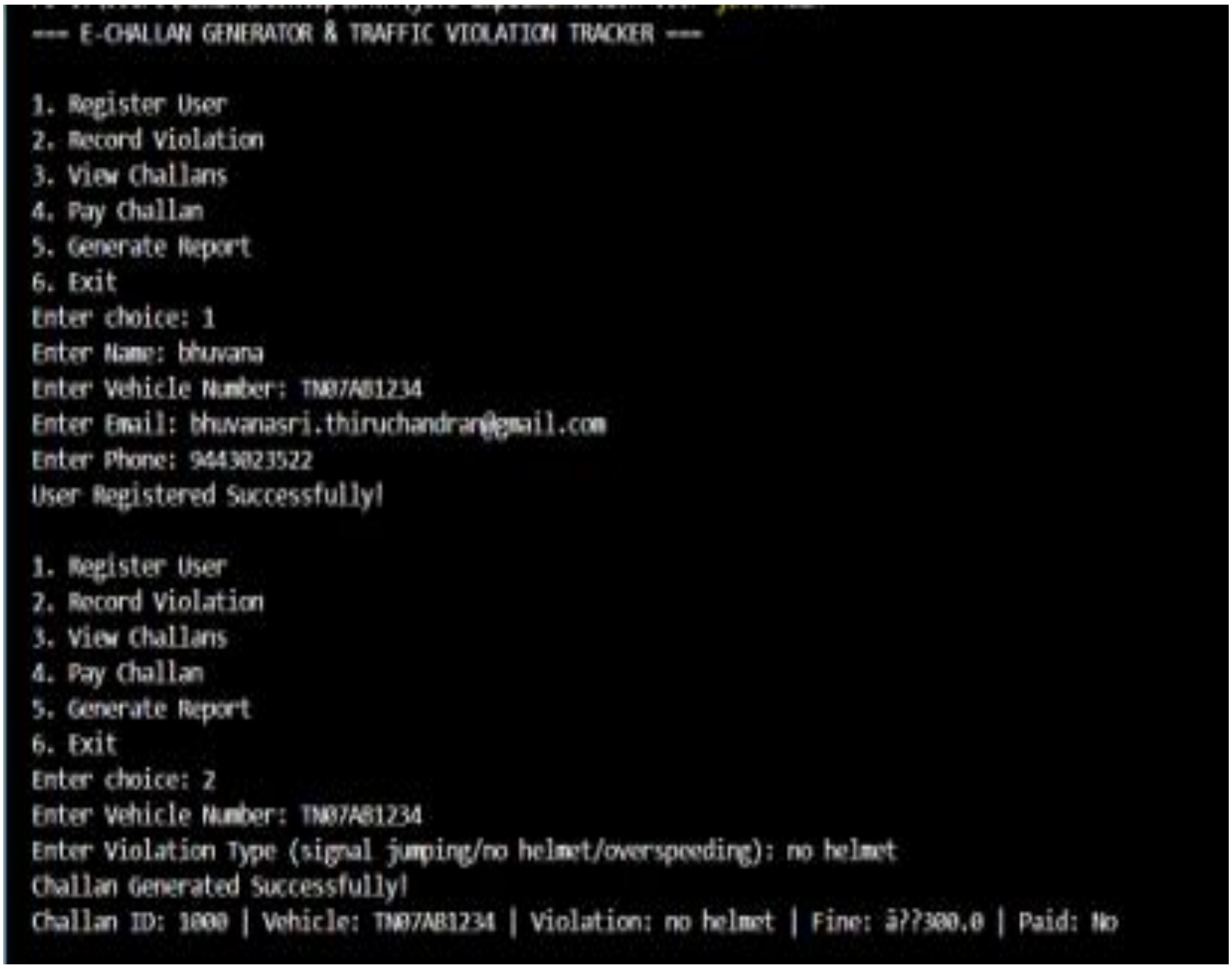
- Format challan details (challan ID, violation type, fine amount, vehicle number, date, time, location, and officer ID).
- Structured layout with proper spacing, alignment, and headings.
- Export as text/PDF (optional).
- Preview display for officers before finalizing or printing.

Technical Requirements

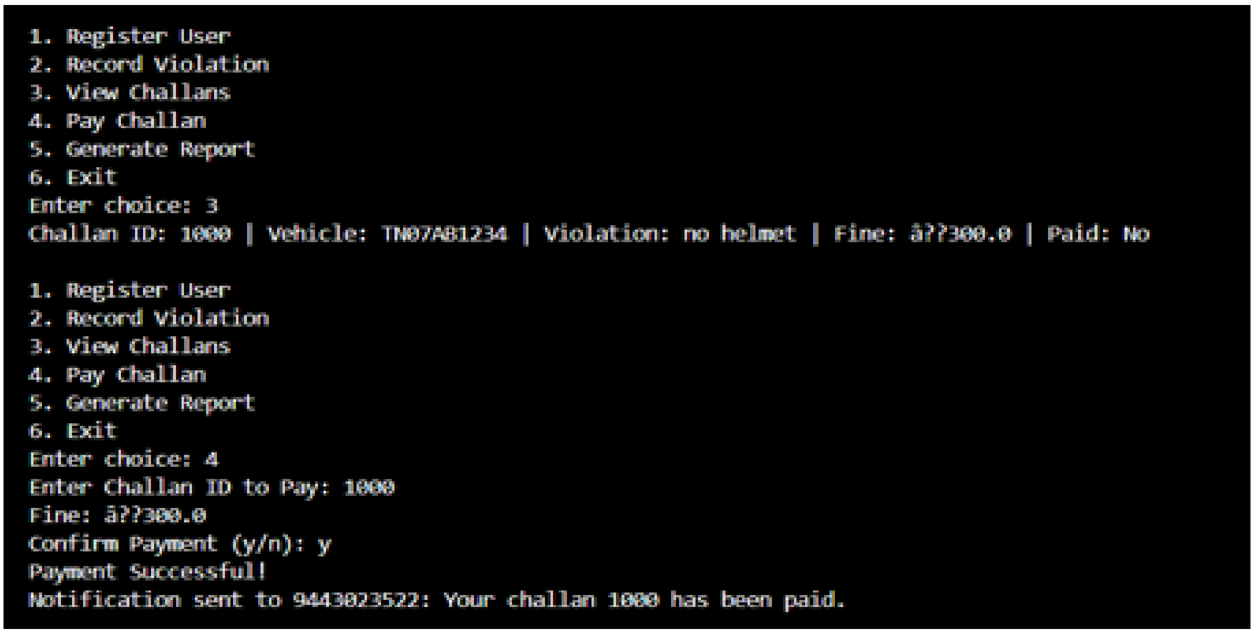
- File I/O for saving the challan in text or PDF format.
- Optional PDF library support (iText or Apache PDFBox).
- String formatting and template-based output generation.
- GUI or console display support for challan preview.

CHAPTER 4

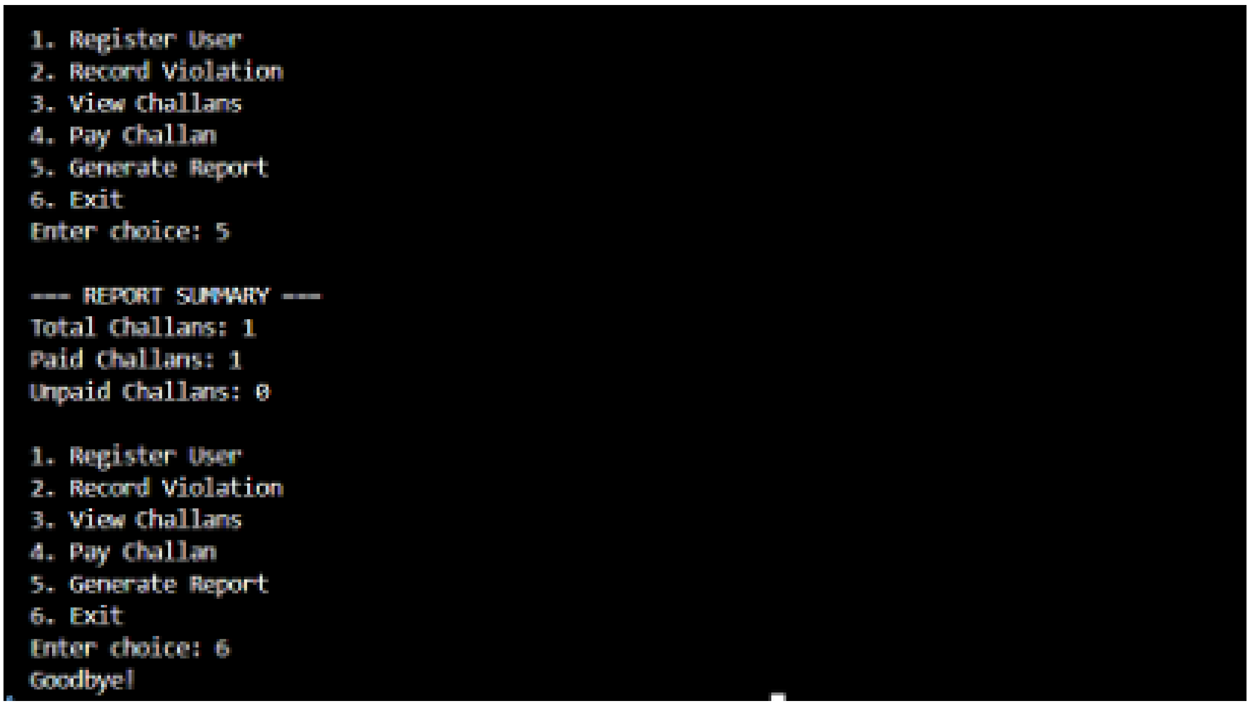
RESULTS AND DISCUSSION



```
--- E-CHALLAN GENERATOR & TRAFFIC VIOLATION TRACKER ---

1. Register User
2. Record Violation
3. View Challans
4. Pay Challan
5. Generate Report
6. Exit
Enter choice: 1
Enter Name: bhuvana
Enter Vehicle Number: TN07AB1234
Enter Email: bhuvanasri.thiruchandran@gmail.com
Enter Phone: 9443023522
User Registered Successfully!

1. Register User
2. Record Violation
3. View Challans
4. Pay Challan
5. Generate Report
6. Exit
Enter choice: 2
Enter Vehicle Number: TN07AB1234
Enter Violation Type (signal jumping/no helmet/overspeeding): no helmet
Challan Generated Successfully!
Challan ID: 1000 | Vehicle: TN07AB1234 | Violation: no helmet | Fine: â??300.0 | Paid: No
```



```
1. Register User
2. Record Violation
3. View Challans
4. Pay Challan
5. Generate Report
6. Exit
Enter choice: 3
Challan ID: 1000 | Vehicle: TN07AB1234 | Violation: no helmet | Fine: â??300.0 | Paid: No

1. Register User
2. Record Violation
3. View Challans
4. Pay Challan
5. Generate Report
6. Exit
Enter choice: 4
Enter Challan ID to Pay: 1000
Fine: â??300.0
Confirm Payment (y/n): y
Payment Successful!
Notification sent to 9443023522: Your challan 1000 has been paid.
```



```
1. Register User
2. Record Violation
3. View Challans
4. Pay Challan
5. Generate Report
6. Exit
Enter choice: 5

--- REPORT SUMMARY ---
Total Challans: 1
Paid Challans: 1
Unpaid Challans: 0

1. Register User
2. Record Violation
3. View Challans
4. Pay Challan
5. Generate Report
6. Exit
Enter choice: 6
Goodbye!
```

4.1 Result

The E-Challan Generator and Traffic Violation Tracker system was tested under multiple real-world traffic scenarios, and the program consistently demonstrated accurate and reliable functionality. When violations such as no helmet, signal jump, and speeding were entered with corresponding vehicle numbers, the system correctly matched each violation with its predefined fine amount and generated a complete e-challan containing all required details. In cases where duplicate violation entries were attempted for the same vehicle at the same time, the system successfully detected the duplication and displayed an appropriate warning, preventing redundant challans from being issued. During testing, if a violation was entered without required fields such as vehicle number or location, the validation module immediately flagged the missing information and prevented the challan from being generated, ensuring data integrity.

4.2 Discussion

The results demonstrate that the E-Challan Generator and Traffic Violation Tracker performs its core functions reliably across various test scenarios. The system accurately maps each violation to its corresponding fine rules and ensures that all necessary information—such as vehicle number, location, date, and violation type—is properly validated before generating a challan. The program's ability to detect duplicate entries and missing data highlights its strong validation mechanism, preventing errors that commonly occur in manual challan issuing processes. The system also handled situations involving updated fine rules effectively, immediately reflecting changes in newly generated challans, which shows the robustness of the rule configuration module. Additionally, the successful creation and storage of challans in text/PDF format confirms that the output module and file handling operations work as expected.

CHAPTER 5
CONCLUSION

The E-Challan Generator and Traffic Violation Tracker successfully automates the process of issuing challans and managing traffic violations, providing a faster, more accurate, and transparent alternative to manual methods. The system reliably calculates fines based on predefined rules, validates all required data, prevents duplicate entries, and generates clean, professional challans ready for printing or digital storage. Testing confirms that the application performs consistently across different violation scenarios and adapts effectively to changes in rule configurations. Overall, the project fulfills its objectives by improving efficiency, reducing human error, and offering a practical solution for modern traffic enforcement.

REFERENCES

- Herbert Schildt, *Java: The Complete Reference*, McGraw-Hill Education.
- Kathy Sierra & Bert Bates, *Head First Java*, O’Reilly Media.
- Official Java Documentation – *docs.oracle.com*
- Oracle, Java Swing Tutorials -*oracle.com/technical-resources/articles/javase/swing.html*

APPENDIX

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.FileWriter;
import java.util.*;

// ===========================
// MAIN CLASS
// ===========================
public class Main {
    public static void main(String[] args) {
        new LoginPage();
    }
}

// ===========================
// LOGIN MODULE
// ===========================
class LoginPage extends JFrame {

    HashMap<String, String> users = new HashMap<>();
    JTextField userField = new JTextField();
    JPasswordField passField = new JPasswordField();

    LoginPage() {

        users.put("admin", "1234");

        setTitle("E-Challan Login");
        setSize(350, 200);
        setLayout(new GridLayout(3, 2));
        add(new JLabel("Username: "));
        add(userField);

        add(new JLabel("Password: "));
        add(passField);

        JButton loginBtn = new JButton("Login");
        add(loginBtn);

        loginBtn.addActionListener(e -> {
            String user = userField.getText();
            String pass = passField.getText();

            if (users.containsKey(user) && users.get(user).equals(pass)) {
                new HomePage();
                dispose();
            } else {
                JOptionPane.showMessageDialog(null, "Invalid Login!");
            }
        });

        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}

// ===========================
// HOME PAGE / USER MODULE
// ===========================
class HomePage extends JFrame {

    HomePage() {
        setTitle("E-Challan System");
        setSize(400, 200);
        setLayout(new FlowLayout());

        JButton generateBtn = new JButton("Generate Challan");
        add(generateBtn);

        generateBtn.addActionListener(e -> new ChallanPage());

        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}

// ==========================
// VIOLATION CLASS + DATABASE
// ==========================
class Violation {
    String type;
    int fine;
    String severity;

    Violation(String type, int fine, String severity) {
        this.type = type;
        this.fine = fine;
        this.severity = severity;
    }
}

class ViolationDatabase {
    ArrayList<Violation> list = new ArrayList<>();

    public ViolationDatabase() {
        list.add(new Violation("No Helmet", 500, "Medium"));
        list.add(new Violation("Signal Jump", 1000, "High"));
        list.add(new Violation("Overspeeding", 1500, "High"));
    }

    public Violation getViolation(String type) {
        for (Violation v : list) {
            if (v.type.equalsIgnoreCase(type)) return v;
        }
        return null;
    }
}

// ===========================
// CHALLAN CLASS
// ===========================
class Challan {

    String challanID;
    String vehicleNo;
    Violation violation;
    String date;
    String officer;

    Challan(String vehicleNo, Violation violation, String officer) {
        this.vehicleNo = vehicleNo;
        this.violation = violation;
        this.officer = officer;

        this.date = java.time.LocalDate.now().toString();
        this.challanID = "CH" + System.currentTimeMillis();
    }
    public String formatChallan() {
        return  "===== E-CHALLAN =====\n" +
                "Challan ID: " + challanID + "\n" +
                "Vehicle No: " + vehicleNo + "\n" +
                "Violation: " + violation.type + "\n" +
                "Severity: " + violation.severity + "\n" +
                "Fine: Rs. " + violation.fine + "\n" +
                "Officer: " + officer + "\n" +
                "Date: " + date + "\n";
    }
}

// ==========================
// VALIDATION MODULE
// ==========================
class Validator {

    public static boolean validate(String vehicle, String violation) {

        if (vehicle.isEmpty() || violation.isEmpty()) {
            JOptionPane.showMessageDialog(null, "All fields are required!");
            return false;
        }

        if (!vehicle.matches("^[A-Z]{2}[0-9]{2}[A-Z]{2}[0-9]{4}$")) {
            JOptionPane.showMessageDialog(null, "Invalid Vehicle Number Format!");
            return false;
        }

        return true;
    }
}
```

```
// ===========================
// OUTPUT MODULE (FILE SAVING)
// ===========================
class FileManager {

    public static void saveChallan(Challan c) {
        try {
            FileWriter fw = new FileWriter("challan_" + c.challanID + ".txt");
            fw.write(c.formatChallan());
            fw.close();
            JOptionPane.showMessageDialog(null, "Challan Saved Successfully!");
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error Saving File!");
        }
    }
}

// ===========================
// CHALLAN GENERATION SCREEN
// ===========================
class ChallanPage extends JFrame {

    JTextField vehicleField = new JTextField();
    JTextField officerField = new JTextField();
    JTextField violationField = new JTextField();

    ViolationDatabase db = new ViolationDatabase();

    ChallanPage() {

        setTitle("Generate Challan");
        setSize(400, 300);
        setLayout(new GridLayout(5, 2));
        add(new JLabel("Vehicle Number:"));
        add(vehicleField);

        add(new JLabel("Violation:"));
        add(violationField);

        add(new JLabel("Officer ID:"));
        add(officerField);

        JButton generateBtn = new JButton("Generate");
        add(generateBtn);

        generateBtn.addActionListener(e -> handleChallan());

        setVisible(true);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }

    void handleChallan() {

        String vehicle = vehicleField.getText();
        String off = officerField.getText();
        String vio = violationField.getText();

        if (!Validator.validate(vehicle, vio)) return;

        Violation v = db.getViolation(vio);
        if (v == null) {
            JOptionPane.showMessageDialog(null, "Violation Not Found!");
            return;
        }
        Challan c = new Challan(vehicle, v, off);
        FileManager.saveChallan(c);

        JOptionPane.showMessageDialog(null, c.formatChallan());
    }
}
```