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.

ANNA UNIVERSITY
PROGRESS THROUGH KNOWLEDGE

A Project Report

On

TRAFFIC CONTROL MANAGEMENT SYSTEM

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

Of

CGB1201 – JAVA PROGRAMMING

Under the guidance of

Dr. S.BHARANINAYAGI, M.E.,Ph.D.,

Assistant Professor / AI

Submitted by

SIBIRAJ S (927624BAD094)

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous), KARUR

DECEMBER 2025

M.KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on **"TRAFFIC CONTROL MANAGEMENT SYSTEM"** is the bonafide work of **SIBIRAJ S (927624BAD094)** who carried out the project work during the academic year 2025 - 2026 under my supervision.

Signature

Dr.S.Bharaninayagi, M.E., Ph.D.,

SUPERVISOR,

Department of Artificial Intelligence and Machine Learning,

M.Kumarasamy College of Engineering,

Thalavapalayam, Karur-639113.

Signature

Dr.A.Selvi, M.E., Ph.D.,

HEAD OF THE DEPARTMENT,

Assistant Professor,

Department of Artificial Intelligence

M.Kumarasamy College of Engineering,

Thalavapalayam, Karur-639113.

This Report has been submitted for the Project Work held on _____

INTERNAL EXAMINER

Programme: B.Tech. – Artificial Intelligence and Data Science

Vision of the Department:

To excel in education, innovation, and research in Artificial Intelligence and Data Science to fulfil industrial demands and societal expectations.

Mission of the Department:

M1: To educate future engineers with solid fundamentals, continually improving teaching methods using modern tools.

M2: To collaborate with industry and offer top-notch facilities in a conducive learning environment.

M3: To foster skilled engineers and ethical innovation in AI and Data Science for global recognition and impact research.

M4: To tackle the societal challenge of producing capable professionals by instilling employ ability skills and human values.

Programme Educational Objectives (PEOs):

Graduates will be able to:

PEO 1: Compete on a global scale for a professional career in Artificial Intelligence and Data Science.

PEO 2: Provide industry-specific solutions for the society with effective communication and ethics.

PEO 3: Hone their professional skills through research and lifelong learning initiatives.

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 reports and design documentation, 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: Capable of finding the important factors in large datasets, simplify the data, and improve predictive model accuracy.

PSO2: Capable of analyzing and providing a solution to a given real-world problem by designing an effective program.

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 Traffic Control Management System is a Java-based desktop application developed using AWT and Swing. It simulates real-time traffic light operations and monitors congestion levels at road junctions. The system includes modules for user authentication, traffic signal control, route optimization, and report generation. A dashboard interface allows users to navigate between login, simulation, and reporting features. The simulation engine dynamically switches between red, yellow, and green signals and displays congestion alerts. The report module records statistics such as simulation duration, signal cycles, congestion alerts, and route optimization counts. This project demonstrates how software tools can support smart traffic management and reduce congestion in urban environments.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED
The Traffic Control Management System is a Java-based desktop application developed using AWT and Swing. It simulates real-time traffic light operations and monitors congestion levels at road junctions. The system includes modules for user authentication, traffic signal control, route optimization, and report generation. A dashboard interface allows users to navigate between login, simulation, and reporting features. The simulation engine dynamically switches between red, yellow, and green signals and displays congestion alerts. The report module records statistics such as simulation duration, signal cycles, congestion alerts, and route optimization counts. This project demonstrates how software tools can support smart traffic management and reduce congestion in urban environments..	**PO1(3)** **PO2(3)** **PO3(3)** **PO4(2)** **PO5(3)** **PO6(2)** **PO7(2)** **PO8(1)** **PO9(2)** **PO10(3)** **PO11(2)** **PO12(2)**	**PSO1(1)** **PSO2(2)**

Note: 1- Low, 2-Medium, 3- High

SUPERVISOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Objective

- The objective of this project is to develop a Java-based Traffic Control Management System that simulates real-time traffic signal operations and monitors congestion levels. The system aims to automate signal transitions, provide route optimization suggestions, and generate useful reports based on simulation data. By using AWT and Swing, the project demonstrates how software tools can help improve traffic management and support smart city development.

1.2 Overview

- The Traffic Control Management System is a Java-based application created using AWT and Swing to simulate traffic signal behavior.
- It controls the automatic switching of Red, Yellow, and Green lights using a timer-based simulation engine.
- The system monitors traffic density and displays congestion levels such as low, moderate, or heavy.
- It provides route optimization suggestions to help reduce traffic delays during high congestion.
- A dashboard and report module allow users to view simulation results and save detailed traffic performance reports.

1.3 Java Programming Concepts

1. **AWT and Swing Components:**

   Used to build the GUI using JFrame, JPanel, JLabel, JButton, JTextField, etc.

2. **Event Handling:**

   Implemented using ActionListener to respond to button clicks such as Login, Start, Stop, Optimize Route, and Save Report.

3. **Timer Class:**

   Controls the automatic switching of traffic lights (Red → Yellow → Green) at fixed intervals during the simulation.

4. **Object-Oriented Programming (OOP):**

   Separate classes like LoginFrame, DashboardFrame, SimulationFrame, ReportsFrame, and Stats ensure modular, reusable, and organized code.

5. **File Handling:**

   Used to create and save .txt reports containing simulation statistics.

6. **Random Class:**

   Generates random traffic levels to simulate low, moderate, and heavy congestion.

CHAPTER 2
PROJECT METHODOLOGY

2.1 Proposed Work

1. Traffic Signal Simulation

Develop a module to simulate Red, Yellow, and Green light transitions using Java Timer. This helps demonstrate real-time traffic control behavior at intersections.

2. User Authentication Module

Create a secure login system to verify user credentials before accessing the application. It ensures only authorized users can use the simulation features.

3. Interactive Dashboard

Design a central dashboard for navigating to Simulation and Report modules. It provides users with a clean and easy interface for accessing system functions.

4. Congestion Monitoring

Simulate varying traffic levels to show low, moderate, and heavy congestion. Alerts will be triggered automatically during high traffic situations.

5. Route Optimization

Provide alternate route suggestions during heavy congestion conditions. This feature demonstrates smart routing to reduce delays in traffic flow.

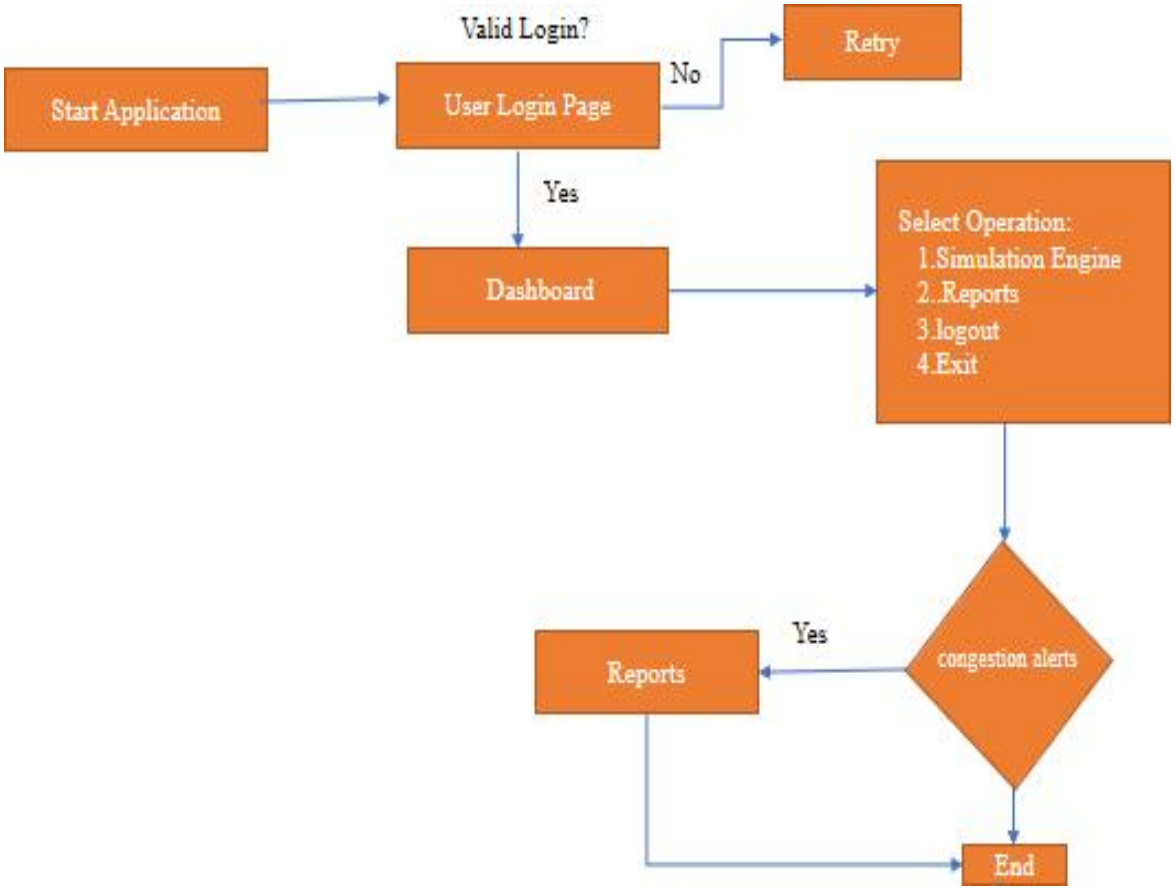
6. Report Generation

Generate reports showing simulation statistics such as alerts, route usage, and runtime. These reports can be saved for future analysis using file handling.

7. System Integration

Integrate all modules into a single Java Swing application for smooth operation. This ensures seamless interaction between login, dashboard, simulation, and reports.

1.1 Block Diagram :



CHAPTER 3

MODULE DESCRIPTION

3.1 Login Module

The Login Module is responsible for managing user authentication. Users must enter a valid username and password to access the system. This ensures that only authorized personnel can use the traffic control application. The module provides error messages in case of invalid credentials and redirects the user to the Dashboard upon successful login. It is implemented using Java Swing components such as JFrame, JTextField, and JPasswordField for a simple and interactive interface.

3.2 Dashboard Module

The Dashboard Module acts as the main control center of the application. After successful login, the dashboard provides navigation buttons to access the Simulation Module, Report Module, or to log out. It displays the current user and session information, including the login time. This module ensures smooth navigation between different functionalities and serves as a central hub for the system's operations. The design uses JPanel, JButton, and layout managers to organize components neatly.

3.3 Traffic Signal Control (Simulation) Module

This module simulates the operation of traffic signals at road intersections in real time. It controls the Red, Yellow, and Green lights using a timer-based logic, with each signal lasting for a predefined duration. The module also monitors traffic congestion by generating random traffic levels to represent low, moderate, and heavy traffic. Alerts are displayed when congestion reaches high levels. Additionally, this module tracks runtime statistics, including the number of simulation starts, congestion alerts, and total simulation time. The interface is created using Swing components such as JLabel and JPanel for signal display.

3.4 Route Optimization Module

The Route Optimization Module suggests alternate routes when high congestion is detected in the simulation. This feature demonstrates how intelligent traffic management can improve vehicle flow and reduce delays. The module selects the best route from predefined options and displays the recommendation on the dashboard or simulation screen. It keeps track of the number of optimizations performed to provide data for reporting. This module uses simple logic combined with Java Swing UI elements for visualization.

3.5 Report Module

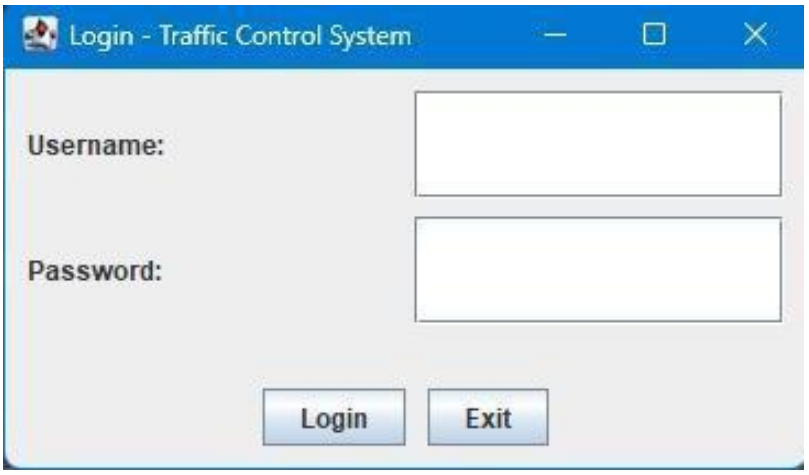
The Report Module generates detailed summaries of simulation performance. It includes data such as the number of simulation starts, congestion alerts, route optimization counts, and total simulation time. Users can view reports in the application and save them as .txt files for future reference. This module ensures that simulation data is preserved and can be analyzed to evaluate the system's performance. It is implemented using JTextArea and file handling classes like FileWriter and PrintWriter.

CHAPTER 4

RESULTS AND DISCUSSION

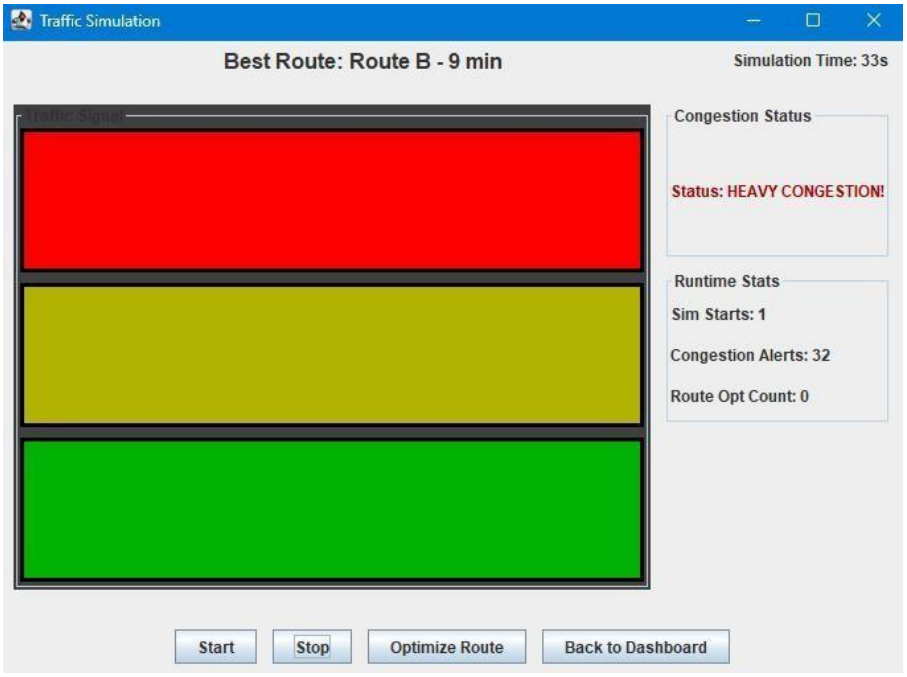
4.1 Results

1. LOGIN PGAE:

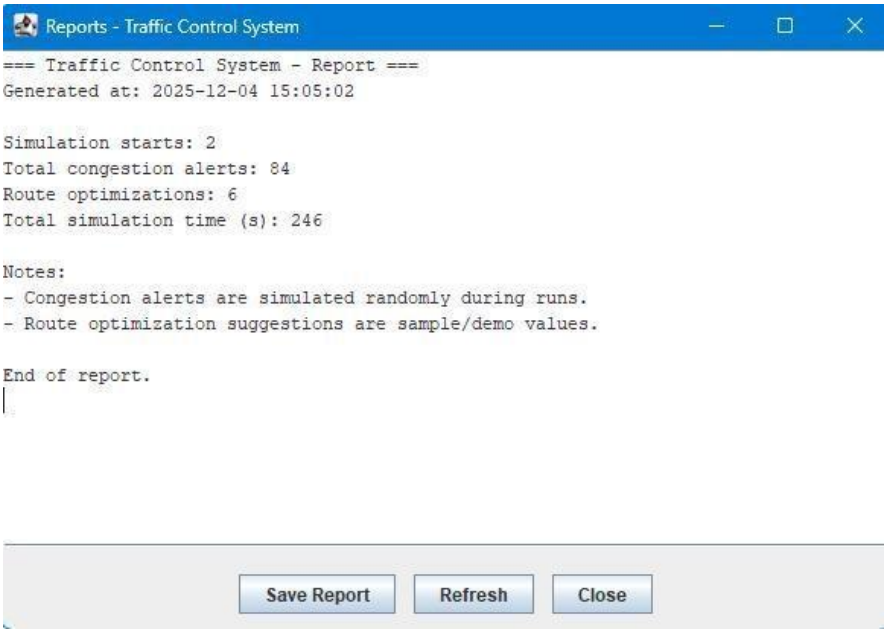


2. DASHBOARD:

3. **SIMULATION:**



4. REPORTS PAGE:



2. Discussion

- The Traffic Control Management System demonstrates how software can manage and control urban traffic effectively. Using Java AWT and Swing, the system simulates traffic signals with Red, Yellow, and Green lights and displays real-time changes on the dashboard. The simulation engine helps understand how congestion forms and how signal timing affects traffic flow.
- The login module ensures secure access, while the dashboard provides organized navigation between Simulation and Reports modules. The simulation module monitors traffic congestion using random traffic levels and triggers alerts for heavy traffic, helping users visualize traffic conditions. The route optimization module suggests alternate paths to minimize delays, showing the concept of intelligent traffic management.
- The report module collects runtime statistics, including the number of simulations, congestion alerts, route optimizations, and total simulation time. Reports can be saved for future analysis, helping users evaluate system performance. Even though the current model is basic, it clearly demonstrates the concept of smart traffic management.
- This project lays a foundation for future enhancements. Integrating real-time sensor data, AI algorithms, and predictive analytic can make the system more intelligent, adaptive, and suitable for real-world Smart City applications. Overall, the system provides practical insights into traffic flow management and decision-making processes in urban traffic systems.

CHAPTER 5
CONCLUSION

- The Traffic Control Management System effectively demonstrates how software-based control and simulation can enhance real-time traffic monitoring and decision-making. The system provides a clear understanding of traffic signal management, congestion detection, and route optimization using Java AWT and Swing.

- Future improvements could include integrating live sensor data, map APIs, and machine learning models to predict congestion trends more accurately. This project serves as a foundation for developing advanced Smart Traffic Management Systems in urban cities, contributing to efficient traffic flow and smarter urban infrastructure.

REFERENCES:

1. Herbert Schildt, *Java: The Complete Reference*, McGraw-Hill Education.
2. Oracle Java Documentation – AWT & Swing:

https://docs.oracle.com/javase/8/docs/

3. Tutorials Point, *Java Swing Tutorial*.
4. Geeks for Geeks, *Java AWT and Swing Programming Concepts*.
5. Official Java™ Platform Standard Edition API Specification.

APPENDIX

Program:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Random;


/**
 * TrafficControl.java
 * 1) Login
 * 2) Dashboard
 * 3) Traffic Simulation
 * 4) Reports
 * Run:
 *   javac TrafficControl.java
 *   java TrafficControl
 * user: admin
 * pass: 1234
 *
 */
public class TrafficControl{

    // Keep simple runtime stats for demo (shared)
    static class Stats {
        int simulationStarts = 0;
        int totalCongestionAlerts = 0;
        int routeOptimizations = 0;
```

```
    long totalSimulationSeconds = 0;
}

public static void main(String[] args)
    { SwingUtilities.invokeLater(() -> new LoginFrame(new Stats()));
}

// ---------------------- Login Frame ----------------------
static class LoginFrame extends JFrame
    { private JTextField userField;
    private JPasswordField passField;
    private Stats stats;

    public LoginFrame(Stats stats)
        { this.stats = stats;
        setTitle("Login - Traffic Control System");
        setSize(380, 220);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout(8, 8));
        add(createForm(), BorderLayout.CENTER);
        setVisible(true);
    }

    private JPanel createForm() {
        JPanel panel = new JPanel(new BorderLayout(8, 8));
        JPanel inputs = new JPanel(new GridLayout(2, 2, 8, 8));
        inputs.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        inputs.add(new JLabel("Username:"));
        userField = new JTextField();
        inputs.add(userField);
```

```
        inputs.add(new JLabel("Password:"));
        passField = new JPasswordField();
        inputs.add(passField);

        panel.add(inputs, BorderLayout.CENTER);

        JPanel buttons = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));
        JButton loginBtn = new JButton("Login");
        JButton exitBtn = new JButton("Exit");

        loginBtn.addActionListener(e -> doLogin());
        exitBtn.addActionListener(e -> System.exit(0));

        buttons.add(loginBtn);
        buttons.add(exitBtn);
        panel.add(buttons, BorderLayout.SOUTH);
        return panel;
    }

    private void doLogin() {
        String user = userField.getText().trim();
        String pass = new String(passField.getPassword()).trim();

        // Demo credentials (change as you like)
        if ("admin".equals(user) && "1234".equals(pass)) {
            // Open Dashboard and close login
            new DashboardFrame(stats);
            dispose();
        } else {
            JOptionPane.showMessageDialog(this,
"Invalid username or password!",
```

```
"Login Failed", JOptionPane.ERROR_MESSAGE);

        }
    }
}

// ---------------------- Dashboard Frame ----------------------
static class DashboardFrame extends JFrame {
    private Stats stats;

    public DashboardFrame(Stats stats)
        { this.stats = stats;
        setTitle("Dashboard - Traffic Control System");
        setSize(500, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout(10, 10));

        JLabel title = new JLabel("Traffic Control Management - Dashboard",
SwingConstants.CENTER);
        title.setFont(new Font("Arial", Font.BOLD, 16));
        add(title, BorderLayout.NORTH);

        add(createCenterPanel(), BorderLayout.CENTER);
        add(createBottomPanel(), BorderLayout.SOUTH);

        setVisible(true);
    }

    private JPanel createCenterPanel() {
        JPanel p = new JPanel(new GridLayout(1, 2, 12, 12));
        p.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
```

```
        JPanel left = new JPanel(new BorderLayout(6,6));
        JPanel right = new JPanel(new BorderLayout(6,6));

        JLabel info = new JLabel("<html><b>System Info</b><br/>"
                + "Active user: admin<br/>"
                + "Start time: " +
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")) +
"</html>");
        info.setBorder(BorderFactory.createTitledBorder("Info"));
        left.add(info, BorderLayout.CENTER);

        JPanel nav = new JPanel(new GridLayout(4,1,8,8));
        JButton simBtn = new JButton("Open Simulation");
        JButton reportsBtn = new JButton("Open Reports");
        JButton logoutBtn = new JButton("Logout");
        JButton exitBtn = new JButton("Exit");

        simBtn.addActionListener(e -> new SimulationFrame(stats));
        reportsBtn.addActionListener(e -> new ReportsFrame(stats));
        logoutBtn.addActionListener(e -> {
           new LoginFrame(stats);
           dispose();
        });
        exitBtn.addActionListener(e -> System.exit(0));

        nav.add(simBtn);
        nav.add(reportsBtn);
        nav.add(logoutBtn);
        nav.add(exitBtn);

        right.setBorder(BorderFactory.createTitledBorder("Actions"));
        right.add(nav, BorderLayout.CENTER);
```

```
        p.add(left);
        p.add(right);
        return p;
    }

    private JPanel createBottomPanel() {
        JPanel bottom = new JPanel(new FlowLayout(FlowLayout.CENTER));
        JLabel hint = new JLabel("Tip: Use Simulation to start/stop traffic simulation &
generate reports.");
        bottom.add(hint);
        return bottom;
    }
}

// ---------- Simulation Frame ----------
static class SimulationFrame extends JFrame implements ActionListener
    { private JPanel signalPanel;
    private JLabel redLight, yellowLight, greenLight;
    private JLabel statusLabel, routeLabel, timerLabel;
    private JButton startBtn, stopBtn, routeBtn, backBtn;
    private Timer timer;
    private int signalState = 0; // 0=Red,1=Yellow,2=Green
    private Random random = new Random();
    private Stats stats;
    private long simulationStartMillis = 0;

    public SimulationFrame(Stats stats) {
        this.stats = stats;
        setTitle("Traffic Simulation");
        setSize(700, 520);
        setLocationRelativeTo(null);
```

```
        setLayout(new BorderLayout(10, 10));
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        add(createTopPanel(), BorderLayout.NORTH);
        add(createCenterPanel(), BorderLayout.CENTER);
        add(createBottomPanel(),  BorderLayout.SOUTH);

        // timer every 1 second: update lights every N seconds (simulate sub-second
control)
        timer = new Timer(1000, e -> {
            // change signal every few seconds depending on state for demo
            changeSignal();
            simulateCongestion();
            updateTimerLabel();
        });

        setVisible(true);
    }

    private JPanel createTopPanel() {
        JPanel p = new JPanel(new BorderLayout());
        routeLabel = new JLabel("Route Suggestion: N/A", SwingConstants.CENTER);
        routeLabel.setFont(new Font("SansSerif", Font.BOLD, 16));
        p.add(routeLabel, BorderLayout.CENTER);

        timerLabel = new JLabel("Simulation Time: 0s", SwingConstants.RIGHT);
        timerLabel.setBorder(BorderFactory.createEmptyBorder(6, 6, 6, 12));
        p.add(timerLabel, BorderLayout.EAST);

        return p;
    }
```

```
    private JPanel createCenterPanel() {
        JPanel center = new JPanel(new BorderLayout(10, 10));
        center.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        // Signal visual
        signalPanel = new JPanel(new GridLayout(3, 1, 8, 8));
        signalPanel.setBackground(Color.DARK_GRAY);
        signalPanel.setBorder(BorderFactory.createTitledBorder("Traffic Signal"));

        redLight = createLight(Color.RED.darker());
        yellowLight = createLight(Color.YELLOW.darker());
        greenLight = createLight(Color.GREEN.darker());

        signalPanel.add(redLight);
        signalPanel.add(yellowLight);
        signalPanel.add(greenLight);

        center.add(signalPanel, BorderLayout.CENTER);

        // Right side: status
        JPanel right = new JPanel(new GridLayout(3, 1, 8, 8));
        statusLabel = new JLabel("Status: Idle", SwingConstants.CENTER);
        statusLabel.setBorder(BorderFactory.createTitledBorder("Congestion Status"));

        JPanel statsPanel = new JPanel(new GridLayout(3, 1));
        statsPanel.setBorder(BorderFactory.createTitledBorder("Runtime Stats"));
        JLabel simStarts = new JLabel("Sim Starts: " + stats.simulationStarts);
        JLabel congestionCount = new JLabel("Congestion Alerts: " +
stats.totalCongestionAlerts);
        JLabel routeCount = new JLabel("Route Opt Count: " + stats.routeOptimizations);
        // We'll refresh these on events
        statsPanel.add(simStarts);
```

```
    statsPanel.add(congestionCount);
    statsPanel.add(routeCount);

    right.add(statusLabel);
    right.add(statsPanel);

    center.add(right, BorderLayout.EAST);

    return center;
}

private JPanel createBottomPanel() {
    JPanel bottom = new JPanel(new FlowLayout(FlowLayout.CENTER, 12, 10));
    startBtn = new JButton("Start");
    stopBtn = new JButton("Stop");
    routeBtn = new JButton("Optimize Route");
    backBtn = new JButton("Back to Dashboard");

    startBtn.addActionListener(this);
    stopBtn.addActionListener(this);
    routeBtn.addActionListener(this);
    backBtn.addActionListener(this);

    bottom.add(startBtn);
    bottom.add(stopBtn);
    bottom.add(routeBtn);
    bottom.add(backBtn);

    return bottom;
}

private JLabel createLight(Color color) {
```

```
    JLabel l = new JLabel();
    l.setOpaque(true);
    l.setBackground(color);
    l.setBorder(BorderFactory.createLineBorder(Color.BLACK,  3));
    l.setPreferredSize(new Dimension(130, 80));
    return l;
}

private int secCounter = 0;
private final int RED_SECONDS = 5;    // demo durations
private final int YELLOW_SECONDS = 2;
private final int GREEN_SECONDS = 4;

private int secondsInState = 0;

private void changeSignal() {
    secondsInState++;
    switch (signalState) {
        case 0: // RED
            if (secondsInState >= RED_SECONDS)
                { toYellow();
            }
            break;
        case 1: // YELLOW
            if (secondsInState >= YELLOW_SECONDS)
                { toGreen();
            }
            break;
        case 2: // GREEN
            if (secondsInState >= GREEN_SECONDS)
                { toRed();
            }
```

```
            break;
        }
    }

    private void setLights(Color red, Color yellow, Color green)
        { redLight.setBackground(red);
        yellowLight.setBackground(yellow);
        greenLight.setBackground(green);
    }

    private void toRed() {
        setLights(Color.RED, Color.YELLOW.darker(), Color.GREEN.darker());
        signalState = 0;
        secondsInState = 0;
    }

    private void toYellow() {
        setLights(Color.RED.darker(), Color.YELLOW, Color.GREEN.darker());
        signalState = 1;
        secondsInState = 0;
    }

    private void toGreen() {
        setLights(Color.RED.darker(), Color.YELLOW.darker(), Color.GREEN);
        signalState = 2;
        secondsInState = 0;
    }

    private void simulateCongestion() {
        // random traffic measure
        int level = random.nextInt(100);
        if (level < 30) {
```

```
            statusLabel.setText("Status: Low Traffic");
            statusLabel.setForeground(Color.GREEN.darker());
        } else if (level < 70) {
            statusLabel.setText("Status: Moderate Traffic");
            statusLabel.setForeground(Color.ORANGE.darker());
        } else {
            statusLabel.setText("Status: HEAVY CONGESTION!");
            statusLabel.setForeground(Color.RED.darker());
            stats.totalCongestionAlerts++;
        }
    }

    private void updateTimerLabel()
        { if (simulationStartMillis != 0)
        {
            long elapsed = (System.currentTimeMillis() - simulationStartMillis) / 1000;
            timerLabel.setText("Simulation Time: " + elapsed + "s");
            stats.totalSimulationSeconds = elapsed;
        }
    }

    private void optimizeRoute() {
        String[] routes = {"Route A - 12 min", "Route B - 9 min", "Route C - 6 min"};
        int best = random.nextInt(routes.length);
        routeLabel.setText("Best Route: " + routes[best]);
        stats.routeOptimizations++;
    }

    @Override
    public void actionPerformed(ActionEvent e)
        { Object src = e.getSource();
        if (src == startBtn) {
            if (!timer.isRunning()) {
```

```
            timer.start();
            stats.simulationStarts++;
            simulationStartMillis = System.currentTimeMillis();
            secondsInState = 0;
            toRed(); // start at red for clarity
          }
        } else if (src == stopBtn) {
          if (timer.isRunning()) timer.stop();
          updateTimerLabel();
        } else if (src == routeBtn)
          { optimizeRoute();
        } else if (src == backBtn) {
          // close simulation window
          dispose();
        }
    }
}

// ..... Reports Frame .....
static class ReportsFrame extends JFrame {
    private Stats stats;
    private JTextArea reportArea;
    private JButton saveBtn, refreshBtn, closeBtn;

    public ReportsFrame(Stats stats)
        { this.stats = stats;
        setTitle("Reports - Traffic Control System");
        setSize(600, 420);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout(8,8));
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
        reportArea = new JTextArea();
        reportArea.setEditable(false);
        reportArea.setFont(new Font("Monospaced", Font.PLAIN, 12));
        add(new JScrollPane(reportArea), BorderLayout.CENTER);

        JPanel bottom = new JPanel(new FlowLayout(FlowLayout.CENTER, 12, 10));
        saveBtn = new JButton("Save Report");
        refreshBtn = new JButton("Refresh");
        closeBtn = new JButton("Close");

        saveBtn.addActionListener(e -> saveReport());
        refreshBtn.addActionListener(e -> refreshReport());
        closeBtn.addActionListener(e -> dispose());

        bottom.add(saveBtn);
        bottom.add(refreshBtn);
        bottom.add(closeBtn);

        add(bottom, BorderLayout.SOUTH);

        refreshReport();
        setVisible(true);
    }

    private String generateReportText()
        { StringBuilder sb = new
        StringBuilder();
        sb.append("=== Traffic Control System - Report ===\n");
        sb.append("Generated at:
").append(LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"))).append("\n\n");
        sb.append("Simulation starts: ").append(stats.simulationStarts).append("\n");
        sb.append("Total congestion alerts:
").append(stats.totalCongestionAlerts).append("\n");
```

```
        sb.append("Route optimizations: ").append(stats.routeOptimizations).append("\n");
        sb.append("Total simulation time (s):
").append(stats.totalSimulationSeconds).append("\n\n");

        sb.append("Notes:\n");
        sb.append("- Congestion alerts are simulated randomly during runs.\n");
        sb.append("- Route optimization suggestions are sample/demo values.\n");
        sb.append("\nEnd of report.\n");
        return sb.toString();
    }

    private void refreshReport()
        { reportArea.setText(generateReportText()
        );
    }

    private void saveReport() {
        String text = generateReportText();
        String filename = "TrafficReport_" +
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMdd_HHmmss")) +
".txt";
        try (PrintWriter pw = new PrintWriter(new FileWriter(filename)))
            { pw.print(text);
            JOptionPane.showMessageDialog(this, "Report saved as: " + filename, "Saved",
JOptionPane.INFORMATION_MESSAGE);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(this, "Failed to save report: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
  }
}
```