COMSATS University Islamabad,
Islamabad Campus

Object Oriented Programming
Course Code: CSC241
Final Project Report

Submitted to

Ms. Sajida Kalsoom

Submitted by

Ahmad Mustafa Iqbal (SP24-BCS-008)

Hafiz Muhammad Noor (SP24-BCS-031)

Date

15th June 2025

Table of Contents

Smart City Resource Management System

Introduction

The Smart City Resource Management System is a Java-based desktop application designed to manage key urban services like transportation, power supply, and emergency response. It uses Java Swing for the graphical user interface and applies core object-oriented programming concepts such as inheritance, abstraction, interfaces, and composition. The system allows admins to perform CRUD operations, generate reports, and simulate real-time scenarios like traffic and outages, offering an efficient way to manage city resources in a smart and structured manner.

UML Representation of the System

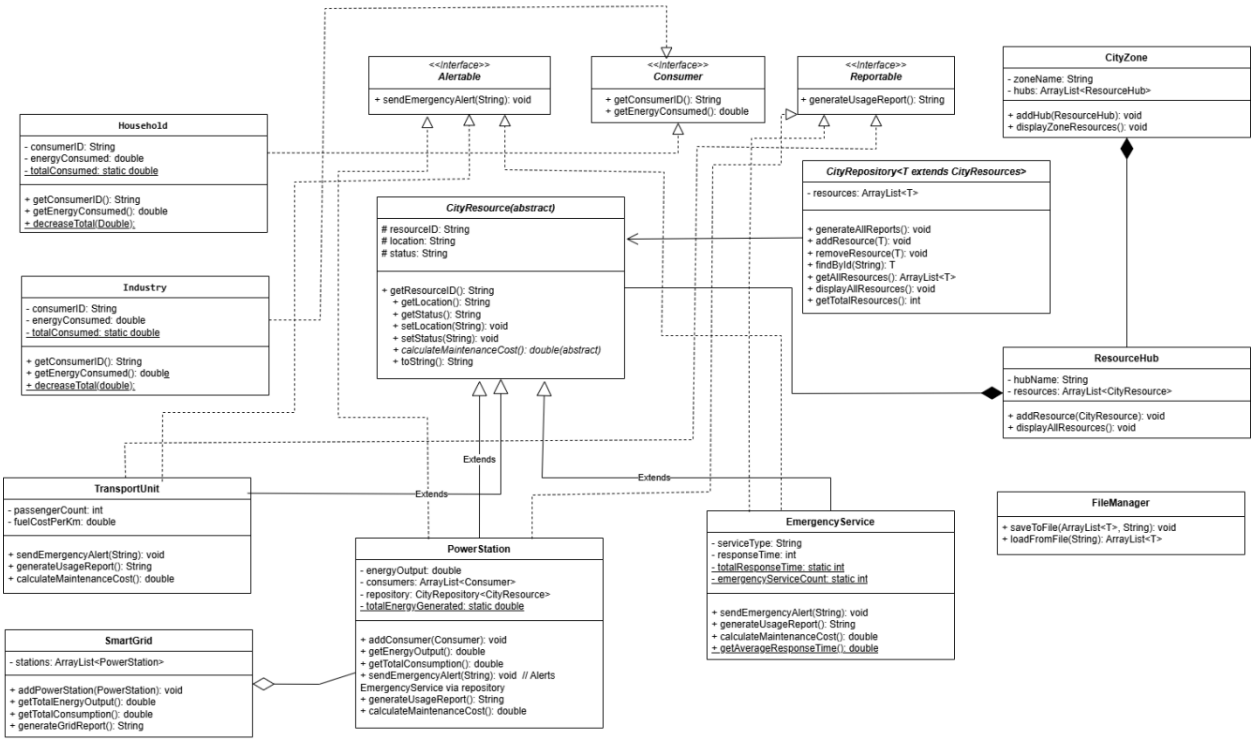

Figure 1 UML Diaghram of Smart City Management System

The above UML diagram was created using the Draw.io UML builder tool, which helped visualize the complete structure of the Smart City Resource Management System. It clearly shows the use of inheritance (e.g., TransportUnit, PowerStation, and EmergencyService extending CityResource), interfaces like Alertable, Reportable, and Consumer, and their implementations. Composition and aggregation are shown between classes such as CityZone → ResourceHub and SmartGrid → PowerStation. The diagram also highlights the use of generics in the CityRepository<T> class and static members for tracking city-wide metrics. All important relationships—associations, inheritances, and compositions—are represented through appropriate UML notations, giving a complete architectural overview of the system.

Implementation Details: Backend Classes

Interfaces

Alertable

```
package interfaces;

public interface Alertable {
    void sendEmergencyAlert(String message);
}
```

This interface defines a contract for any class that should be able to send emergency alerts. It includes one method sendEmergencyAlert(String message), which can be used by classes like EmergencyService and PowerStation to notify about critical situations (e.g., outages, accidents).

Concept used: Interface (for behavior sharing)

Consumer

```
package interfaces;

public interface Consumer {
    String getConsumerID();
```

```
    double getEnergyConsumed();
}
```

The Consumer interface is used for classes that consume energy, like Household and Industry. It ensures that all consumer classes have a way to get their ID and the amount of energy they've used.

Concept used: Interface (for common structure)

Reportable

```
package interfaces;

public interface Reportable {
    String generateUsageReport();
}
```

This interface is used to provide a reporting feature to different classes. Any class that implements Reportable must define how it generates a usage report — like number of passengers (for TransportUnit) or energy output (for PowerStation).

Concept used: Interface.

Model Classes:

CityResource Class

```
package model;

import java.io.Serializable;

public abstract class CityResource implements Serializable {
    private static final long serialVersionUID = 1L;
    protected String resourceID;
    protected String location;
    protected String status;
```

```
    public CityResource(String resourceID, String location, String status) {
        this.resourceID = resourceID;
        this.location = location;
        this.status = status;
    }

    public String getResourceID() {
        return resourceID;
    }

    public String getLocation() {
        return location;
    }

    public String getStatus() {
        return status;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public abstract double calculateMaintenanceCost();

    @Override
    public String toString() {
        return "ID: " + resourceID + ", Location: " + location + ", Status: " + status;
    }
}
```

This is an abstract base class that defines the common structure for all city resources (like transport, power, emergency).

Key Features:

- Fields: resourceID, location, status (shared by all resources)
- Implements Serializable for file saving
- Abstract method calculateMaintenanceCost() — must be implemented by subclasses
- Overridden toString() for easy display in GUI/report
- Uses OOP concepts: Abstraction, Inheritance, Polymorphism, and Serialization.

CityRepository<T extends CityResource> Class

```
package model;

import java.io.Serializable;
import java.util.ArrayList;

import interfaces.Reportable;

public class CityRepository<T extends CityResource> implements Serializable{
    private ArrayList<T> resources;

    public CityRepository() {
        resources = new ArrayList<>();
    }

    public void generateAllReports() {
        for (T resource : resources) {
            if (resource instanceof Reportable) {
                System.out.println(((Reportable) resource).generateUsageReport());
            }
        }
    }

    public void addResource(T resource) {
        if (findById(resource.getResourceID()) == null) {
            resources.add(resource);
        } else {
            System.out.println("Duplicate Resource ID: " + resource.getResourceID());
        }
    }

    public void removeResource(T resource) {
        resources.remove(resource);
    }

    public T findById(String resourceID) {
        for (T resource : resources) {
            if (resource.getResourceID().equals(resourceID)) {
                return resource;
            }
        }
```

```
        return null;
    }

    public ArrayList<T> getAllResources() {
        return resources;
    }

    public void displayAllResources() {
        for (T resource : resources) {
            System.out.println(resource);
        }
    }

    public int getTotalResources() {
        return resources.size();
    }

    public void updateResource(T updatedResource) {
        for (int i = 0; i < resources.size(); i++) {
            if (resources.get(i).getResourceID().equals(updatedResource.getResourceID())) {
                resources.set(i, updatedResource);
                return;
            }
        }
    }

    public void removeResource(String resourceID) {
        resources.removeIf(r -> r.getResourceID().equals(resourceID));
    }

    public void clear() {
        resources.clear();
    }

}
```

This is a generic repository class that manages different types of city resources — like TransportUnit, PowerStation, and EmergencyService.

Key Features:

- Uses Generics (<T extends CityResource>) to support all resource types
- Stores resources in an ArrayList<T>
- Provides full CRUD operations: add, remove, update, findById, clear

- generateAllReports() method calls the generateUsageReport() from any class that implements the Reportable interface
- displayAllResources() uses overridden toString() to show summaries
- Avoids duplicates using ID check
- Implements OOP concepts: Generics, Polymorphism (instanceof + interfaces), and Encapsulation

CityZone Class

```
package model;

import java.io.Serializable;
import java.util.ArrayList;

public class CityZone implements Serializable {
    private String zoneName;
    private ArrayList<ResourceHub> hubs;

    public CityZone(String zoneName) {
        this.zoneName = zoneName;
        this.hubs = new ArrayList<>();
    }

    public String getZoneName() {
        return zoneName;
    }

    public ArrayList<ResourceHub> getHubs() {
        return hubs;
    }

    public void addHub(ResourceHub hub) {
        hubs.add(hub);
    }

    public void removeHub(ResourceHub hub) {
        hubs.remove(hub);
    }

    public void displayZoneResources() {
        System.out.println("City Zone: " + zoneName);
        for (ResourceHub hub : hubs) {
```

```
            hub.displayAllResources();
        }
    }
}
```

CityZone represents a specific area of the city (like North Zone, South Zone) that contains multiple ResourceHubs (e.g., transport, power hubs).

Key Features:

- Field zoneName stores the name of the zone
- Uses an ArrayList<ResourceHub> to store all hubs in the zone
- Methods to add/remove hubs and display all resources in the zone
- Represents composition — a zone is made up of multiple hubs
- Implements OOP concepts: Composition, Encapsulation, and Modular Design
- Serializable for file saving/loading

EmergencyService Class

```
package model;

import java.io.Serializable;
import interfaces.Alertable;
import interfaces.Reportable;

public class EmergencyService extends CityResource implements Serializable,Alertable, Reportable {
    private String serviceType;
    private int responseTime;
    private static int totalResponseTime = 0;
    private static int emergencyServiceCount = 0;


    public EmergencyService(String resourceID, String location, String status, String serviceType, int
responseTime) {
        super(resourceID, location, status);
        this.serviceType = serviceType;
        this.responseTime = responseTime;
        totalResponseTime += responseTime;
        emergencyServiceCount++;
    }

    public EmergencyService(String resourceID, String location, String status, String serviceType) {
```

```
        this(resourceID, location, status, serviceType, 5);

    }


    public String getServiceType() {
        return serviceType;
    }

    public int getResponseTime() {
        return responseTime;
    }

    public void setServiceType(String serviceType) {
        this.serviceType = serviceType;
    }

    public void setResponseTime(int responseTime) {
        this.responseTime = responseTime;
    }

    @Override
    public double calculateMaintenanceCost() {
        return 500 + (10 * responseTime);
    }

    @Override
    public void sendEmergencyAlert(String message) {
        System.out.println("[" + serviceType + " Unit Alert] " + message);
    }

    @Override
    public String generateUsageReport() {
        return "Emergency Service (" + serviceType + ") - Response Time: " + responseTime + " min";
    }

    @Override
    public String toString() {
        return super.toString() + ", Type: Emergency (" + serviceType + "), Response Time: " + responseTime +
" min";
    }

    public static double getAverageResponseTime() {
        if (emergencyServiceCount == 0) return 0;
        return (double) totalResponseTime / emergencyServiceCount;
```

```
    }

}
```

This class represents emergency response units such as police, ambulance, or fire services. It extends CityResource and implements both Alertable and Reportable interfaces.

Key Features:

- Inherits basic info (resourceID, location, status) from CityResource
- Alertable: sends alerts with sendEmergencyAlert()
- Reportable: generates usage reports
- serviceType (e.g., Police, Fire)
- responseTime (how quickly unit responds)
- Calculates maintenance cost using responseTime
- Uses static fields to track city-wide average response time
- Uses constructor chaining for default values
- Overridden toString() provides readable output for GUI/report

OOP Concepts Used:

This class uses inheritance (extends CityResource), interfaces (Alertable, Reportable), and polymorphism (method overriding like toString() and calculateMaintenanceCost()). It also includes constructor overloading, and uses static members to track average response time across all emergency units.

PowerStation Class

```
package model;
import interfaces.*;

import java.io.Serializable;
import java.util.ArrayList;

public class PowerStation extends CityResource implements Reportable, Alertable, Serializable {
    private double energyOutput;
    private ArrayList<Consumer> consumers;
    private CityRepository<CityResource> repository;
    private static double totalEnergyGenerated = 0;
```

```
    public PowerStation(String id, String loc, String status, double
energyOutput,CityRepository<CityResource> repository) {
        super(id, loc, status);
        this.energyOutput = energyOutput;
        this.consumers = new ArrayList<>();
        this.repository = repository;
        totalEnergyGenerated += energyOutput;

    }

    public static double getTotalEnergyGenerated() {
        return totalEnergyGenerated;
    }


    public void addConsumer(Consumer c) {
        consumers.add(c);
    }

    public double getEnergyOutput() {
        return energyOutput;
    }

    public double getTotalConsumption() {
        double total = 0;
        for (Consumer c : consumers) {
            total += c.getEnergyConsumed();
        }
        return total;
    }

    @Override
    public double calculateMaintenanceCost() {
        return energyOutput * 0.5;
    }

    @Override
    public String generateUsageReport() {
        return "PowerStation " + getResourceID() + ": Output=" + energyOutput +
                " kWh, Consumption=" + getTotalConsumption() + " kWh";
    }

    @Override
    public void sendEmergencyAlert(String msg) {
        System.out.println("ALERT: PowerStation " + getResourceID() + ": " + msg);
```

```
        System.out.println("PowerStation " + resourceID + ": Output=" + energyOutput + " kWh, Consumption="
+ getTotalConsumption() + " kWh");

        if (repository != null) {
            for (CityResource res : repository.getAllResources()) {
                if (res instanceof EmergencyService) {
                    EmergencyService es = (EmergencyService) res;
                    if (es.getLocation().equalsIgnoreCase(this.location) || es.getStatus().equalsIgnoreCase("Ready")) {
                        System.out.println("[PowerStation -> Emergency Alert] Notifying Emergency Service: " +
es.getResourceID());
                        es.sendEmergencyAlert("Overload at PowerStation " + this.resourceID + " in zone " +
this.location);
                    }
                }
            }
        } else {
            System.out.println(" ⚠ No repository linked to PowerStation. Cannot notify Emergency Services.");
        }
    }



    @Override
    public String toString() {
        return "PowerStation[" + getResourceID() + ", " + getLocation() + ", " + getStatus() +
                ", Output=" + energyOutput + " kWh, Consumers=" + consumers.size() + "]";
    }

    public ArrayList<Consumer> getConsumers() {
        return consumers;
    }

        public static void decreaseTotalEnergyGenerated(double amount) {
        totalEnergyGenerated -= amount;
        if (totalEnergyGenerated < 0) totalEnergyGenerated = 0;
    }
}
```

Key Features

- Represents an electricity-producing unit in the city.
- Stores energy output and a list of consumers.
- Sends alerts to emergency services during overload.
- Generates reports with usage and consumption data.
- Maintains total energy output using static variables.

OOP Concepts Used:

This class uses inheritance from CityResource, and implements the Reportable and Alertable interfaces. It overrides key methods for polymorphic behavior. Composition is used through a list of Consumer objects, and aggregation via a reference to CityRepository. Static members track total energy generated across all instances.

TransportUnit Class

```
package model;

import java.io.Serializable;
import interfaces.Alertable;
import interfaces.Reportable;

public class TransportUnit extends CityResource implements Reportable,Serializable,Alertable {
    private int passengerCount;
    private double fuelCostPerKm;

    public TransportUnit(String resourceID, String location, String status, int passengerCount, double
fuelCostPerKm) {
        super(resourceID, location, status);
        this.passengerCount = passengerCount;
        this.fuelCostPerKm = fuelCostPerKm;
    }

    public int getPassengerCount() {
        return passengerCount;
    }

    public double getFuelCostPerKm() {
        return fuelCostPerKm;
    }

    public void setPassengerCount(int passengerCount) {
        this.passengerCount = passengerCount;
    }

    public void setFuelCostPerKm(double fuelCostPerKm) {
        this.fuelCostPerKm = fuelCostPerKm;
    }

    public void sendEmergencyAlert(String message) {
```

```
        System.out.println("[TransportUnit Alert] " + message);
    }

    @Override
    public double calculateMaintenanceCost() {
        return fuelCostPerKm * 100;
    }

    @Override
    public String generateUsageReport() {
        return "Transport Unit - Passengers: " + passengerCount + ", Fuel/km: " + fuelCostPerKm;
    }

    @Override
    public String toString() {
        return super.toString() + ", Type: Transport, Passengers: " + passengerCount + ", Fuel Cost/km: " +
fuelCostPerKm;
    }
}
```

Key Features:

- Represents transport services like buses or trains.
- Stores passenger count and fuel cost per kilometer.
- Sends alerts in case of emergencies.
- Calculates maintenance based on fuel cost.
- Provides usage report with passenger and fuel data.

OOP Concepts Used:

This class uses inheritance (extends CityResource) and implements Reportable and Alertable interfaces. It overrides key methods (toString(), calculateMaintenanceCost(), generateUsageReport()), showing runtime polymorphism. It also demonstrates encapsulation through getters and setters for fuel and passenger data.

SmartGrid Class

```
package model;

import java.io.Serializable;
import java.util.ArrayList;

public class SmartGrid implements Serializable {
```

```
    private ArrayList<PowerStation> stations;

    public SmartGrid() {
        stations = new ArrayList<>();
    }

    public void addPowerStation(PowerStation ps) {
        stations.add(ps);
    }

    public ArrayList<PowerStation> getAllStations() {
        return stations;
    }

    public double getTotalEnergyOutput() {
        double total = 0;
        for (PowerStation ps : stations) {
            total += ps.getEnergyOutput();
        }
        return total;
    }

    public double getTotalConsumption() {
        double total = 0;
        for (PowerStation ps : stations) {
            total += ps.getTotalConsumption();
        }
        return total;
    }

    public String generateGridReport() {
        StringBuilder sb = new StringBuilder("=== Smart Grid Report ===\n");
        for (PowerStation ps : stations) {
            sb.append(ps.generateUsageReport()).append("\n");
        }
        sb.append("Total Energy Output: ").append(getTotalEnergyOutput()).append(" kWh\n");
        sb.append("Total Energy Consumption: ").append(getTotalConsumption()).append(" kWh\n");
        return sb.toString();
    }
}
```

Key Features:

- Manages a collection of PowerStation objects.

- Calculates total energy output and consumption of the grid.
- Generates detailed reports for all stations.
- Supports adding multiple power stations dynamically.

OOP Concepts Used:

This class uses composition, as it holds a list of PowerStation objects, meaning the grid is made up of stations. It also demonstrates encapsulation by keeping the station list private and accessing it via public methods. The generateGridReport() method shows how object collaboration is used to gather reports from all composed objects.

ResourceHub Class

```
package model;

import java.io.Serializable;
import java.util.ArrayList;

public class ResourceHub implements Serializable{
    private String hubName;
    private ArrayList<CityResource> resources;

    public ResourceHub(String hubName) {
        this.hubName = hubName;
        this.resources = new ArrayList<>();
    }

    public String getHubName() {
        return hubName;
    }

    public ArrayList<CityResource> getResources() {
        return resources;
    }

    public void addResource(CityResource resource) {
        resources.add(resource);
    }

    public void removeResource(CityResource resource) {
        resources.remove(resource);
    }
```

```
    public void displayAllResources() {
        System.out.println("Resources in Hub: " + hubName);
        for (CityResource resource : resources) {
            System.out.println(resource);
        }
    }
}
```

Key Features

- Represents a physical hub that holds multiple CityResource objects.
- Stores and manages transport units, power stations, or emergency services.
- Allows adding
- removing, and displaying all connected resources.
- Connected to CityZone through composition.

OOP Concepts Used:

ResourceHub demonstrates composition by containing a list of CityResource objects. It uses encapsulation by providing getter methods to access private data. The class is part of the system's modular design, allowing flexible grouping of different city resources.

Household & Industry Classes

```
package model;
import interfaces.*;

import java.io.Serializable;

public class Household implements Consumer, Serializable {
    private String consumerID;
    private double energyConsumed;
    private static double totalConsumed = 0;

    public Household(String id, double consumed) {
        this.consumerID = id;
        this.energyConsumed = consumed;
        totalConsumed += consumed;
    }

    public String getConsumerID() {
        return consumerID;
    }
```

```
    public double getEnergyConsumed() {
        return energyConsumed;
    }

    public static double getTotalConsumed() {
        return totalConsumed;
    }

    public static void decreaseTotal(double amount) {
        totalConsumed -= amount;
        if (totalConsumed < 0) totalConsumed = 0;
    }

    @Override
    public String toString() {
        return "Household[" + consumerID + " - Consumed: " + energyConsumed + " kWh]";
    }
}

    package model;
    import interfaces.*;
    import java.io.Serializable;

    public class Industry implements Consumer, Serializable {
    private String consumerID;
    private double energyConsumed;
    private static double totalConsumed = 0;

    public Industry(String id, double consumed) {
        this.consumerID = id;
        this.energyConsumed = consumed;
        totalConsumed += consumed;
    }

    public String getConsumerID() {
        return consumerID;
    }

    public double getEnergyConsumed() {
        return energyConsumed;
    }

    public static double getTotalConsumed() {
        return totalConsumed;
```

```
    }

    public static void decreaseTotal(double amount) {
        totalConsumed -= amount;
        if (totalConsumed < 0) totalConsumed = 0;
    }

    @Override
    public String toString() {
        return "Industry[" + consumerID + " - Consumed: " + energyConsumed + " kWh]";
    }
}
```

Key Features:

- Both classes implement the Consumer interface.
- Represent energy consumers in the city (Households and Industries).
- Each object has a unique ID and a specific energy consumption value.
- totalConsumed is a static field used to track total energy usage.
- The decreaseTotal() method updates the shared total when needed.

OOP Concepts Used:

The Household and Industry classes use interface implementation through Consumer, enabling polymorphism. Encapsulation is maintained via private fields and public getters. Static members are used for shared tracking across all instances, representing class-level data.

FileManager Class

```
package model;
import java.io.*;
import java.util.ArrayList;

public class FileManager {

    public static <T> void saveToFile(ArrayList<T> data, String fileName) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileName))) {
            oos.writeObject(data);
            System.out.println("Saved to " + fileName);
        } catch (IOException e) {
            System.out.println("Error saving to file: " + fileName);
            e.printStackTrace();
        }
```

```
    }

    @SuppressWarnings("unchecked")
    public static <T> ArrayList<T> loadFromFile(String fileName) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {
            return (ArrayList<T>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error loading from file: " + fileName);
            return new ArrayList<>();
        }
    }
}
```

Key Features:

- Generic utility class for saving and loading objects to and from files.
- Uses Java Serialization (ObjectOutputStream / ObjectInputStream).
- All methods are static, so there is no need to create an object of the class.
- Exception handling is implemented to catch and display file operation errors.

OOP Concepts Used:

The FileManager class uses generic programming, allowing it to handle different types of objects flexibly. It applies static methods, enabling access without creating class instances. Exception handling ensures that the application remains stable during file operations. This class also demonstrates abstraction by hiding complex file I/O details and providing a simple, reusable interface.

Implementation Details: GUI Classes

GUI

LoginFrame Class

```
package src;

import javax.swing.*;

import src.gui.SmartCityDashboard;
```

```
import java.awt.*;
import java.awt.event.*;

public class LoginFrame extends JFrame {
  private JComboBox<String> roleCombo;
  private JButton loginButton;

  public LoginFrame() {
    setTitle("Smart City Login");
    setSize(350, 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setLayout(new BorderLayout());

    JPanel centerPanel = new JPanel();
    centerPanel.setOpaque(false);
    centerPanel.setLayout(new BoxLayout(centerPanel, BoxLayout.Y_AXIS));
    centerPanel.setBorder(BorderFactory.createEmptyBorder(20, 40, 20, 40));


    JLabel roleLabel = new JLabel("Select Role:");
    roleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
    roleLabel.setHorizontalAlignment(JLabel.CENTER);

    roleCombo = new JComboBox<>(new String[]{"Admin", "Public"});
    roleCombo.setFont(new Font("Arial", Font.PLAIN, 14));
    roleCombo.setMaximumSize(new Dimension(150, 25));
    roleCombo.setAlignmentX(Component.CENTER_ALIGNMENT);

    centerPanel.add(Box.createVerticalStrut(10));
    centerPanel.add(roleLabel);
    centerPanel.add(Box.createVerticalStrut(7));
    centerPanel.add(roleCombo);
    centerPanel.add(Box.createVerticalStrut(10));

    add(centerPanel, BorderLayout.CENTER);

    loginButton = new JButton("Login");
    loginButton.setBackground(new Color(0, 123, 255));
    loginButton.setForeground(Color.WHITE);
    loginButton.setFont(new Font("Arial", Font.BOLD, 14));
    loginButton.setFocusPainted(false);
    loginButton.setCursor(new Cursor(Cursor.HAND_CURSOR));
    loginButton.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
```

```
loginButton.addMouseListener(new MouseAdapter() {
  public void mouseEntered(MouseEvent e) {
    loginButton.setBackground(new Color(30, 144, 255));
  }

  public void mouseExited(MouseEvent e) {
    loginButton.setBackground(new Color(0, 123, 255));
  }
});

JButton logout = new JButton("Exit");
logout.setBackground(new Color(255, 51, 51));
logout.setForeground(Color.WHITE);
logout.setFont(new Font("Arial", Font.BOLD, 14));
logout.setFocusPainted(false);
logout.setCursor(new Cursor(Cursor.HAND_CURSOR));
logout.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

logout.addMouseListener(new MouseAdapter() {
  public void mouseEntered(MouseEvent e) {
    logout.setBackground(new Color(204, 0, 0));
  }

  public void mouseExited(MouseEvent e) {
    logout.setBackground(new Color(255, 52, 52));
  }
});

logout.addActionListener(e -> System.exit(0));


JPanel bottomPanel = new JPanel();
bottomPanel.setAlignmentX(Component.CENTER_ALIGNMENT);
bottomPanel.setBorder(BorderFactory.createEmptyBorder(10, 40, 20, 40));
bottomPanel.add(loginButton);
bottomPanel.add(logout);
add(bottomPanel, BorderLayout.SOUTH);

loginButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    String role = (String) roleCombo.getSelectedItem();
    boolean isAdmin = role.equalsIgnoreCase("Admin");
    new SmartCityDashboard(isAdmin);
    dispose();
```

```
    }
  });

  setVisible(true);
 }

 public static void main(String[] args) {
  new LoginFrame();
 }
}
```

Key Features:

- GUI window for selecting user role (Admin or Public) and logging in.
- Uses JFrame, JPanel, JComboBox, and JButton for layout and interaction.
- MouseAdapter used to enhance button hover effects.
- Action listeners handle login logic and exit functionality.
- Launches the main dashboard (SmartCityDashboard) after login.

OOP Concepts Used:

The LoginFrame class demonstrates **encapsulation** by bundling GUI components and login logic inside a dedicated class. It applies **event-driven programming** through ActionListener and MouseAdapter. Inheritance is used via JFrame, enabling LoginFrame to function as a window. The use of composition is evident through the inclusion of Swing components like JButton, JComboBox, and JPanel.

SmartCityDashboard Class

```
package src.gui;
import javax.swing.*;

import src.model.CityRepository;
import src.model.CityResource;
import src.model.CityZone;
import src.model.Household;
import src.model.Industry;
import src.model.SmartGrid;

import java.awt.*;
import java.util.ArrayList;

public class SmartCityDashboard extends JFrame {
```

```
  private boolean isAdmin;
  private CityRepository<CityResource> repository;
  private SmartGrid smartGrid;
  private ArrayList<CityZone> zones;
  private ArrayList<Household> households;
  private ArrayList<Industry> industries;

  public SmartCityDashboard(boolean isAdmin) {
    this.isAdmin = isAdmin;
    this.repository = new CityRepository<>();
    this.households = new ArrayList<>();
    this.industries = new ArrayList<>();

    setTitle("Smart City Dashboard - " + (isAdmin ? "Admin" : "Public"));
    setSize(1000, 700);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);

    JTabbedPane tabbedPane = new JTabbedPane();
    JPanel wrapper = new JPanel(new BorderLayout());
    wrapper.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
    getContentPane().setBackground(new Color(204, 255, 255));
    tabbedPane.setOpaque(false);

    FileManagerFront f = new FileManagerFront(repository, zones, smartGrid, households, industries,
isAdmin);
    JPanel emergencyPanel = new EmergencyServicePanel(repository, isAdmin);
    JPanel transportPanel = new TransportUnitPanel(repository, isAdmin);
    JPanel consumerPanel = new ConsumerPanel(households, industries, isAdmin);
    JPanel powerPanel = new PowerStationPanel(repository, isAdmin, smartGrid);

    JPanel backPanel = new BackPanel(this, f);

    tabbedPane.addTab("Power Stations", powerPanel);
    tabbedPane.addTab("Emergency Services", emergencyPanel);
    tabbedPane.addTab("Transport Units", transportPanel);
    tabbedPane.addTab("Consumers", consumerPanel);
    tabbedPane.addTab("Back", backPanel);

    tabbedPane.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
    tabbedPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

    add(tabbedPane);
    setVisible(true);
```

```
    addWindowListener(new java.awt.event.WindowAdapter() {
      public void windowClosing(java.awt.event.WindowEvent e) {
        f.saveData();
      }
    });
  }
}
```

Key Features:

- Main dashboard window for the Smart City System.
- Uses JTabbedPane to organize different functional panels (Power, Emergency, Transport, Consumers, Back).
- Initializes all necessary lists and repositories for resource management.
- Loads GUI panels: PowerStationPanel, EmergencyServicePanel, TransportUnitPanel, ConsumerPanel, and BackPanel.
- Automatically saves data on window close via WindowListener.

OOP Concepts Used:

SmartCityDashboard uses composition by including multiple panels and data components as part of its layout. Inheritance is used by extending JFrame, allowing it to serve as a GUI window. The class encapsulates data and behavior related to dashboard display and functionality. Event handling is implemented using a listener for managing save-on-close logic. Constructor overloading and object composition also demonstrate modular and scalable OOP design.

PowerStationPanel Class

```
package src.gui;


import javax.swing.*;

import src.model.CityRepository;
import src.model.CityResource;
import src.model.PowerStation;
import src.model.SmartGrid;

import java.awt.*;
import java.util.ArrayList;

public class PowerStationPanel extends JPanel {
```

```
private CityRepository<CityResource> repository;
private DefaultListModel<PowerStation> listModel;
private JList<PowerStation> stationList;

private JTextField idField, locationField, statusField, outputField;
private JButton addBtn, updateBtn, deleteBtn, alertBtn, reportBtn;
private boolean isAdmin;
private SmartGrid grid;

public PowerStationPanel(CityRepository<CityResource> repository, boolean isAdmin, SmartGrid grid) {
    this.repository = repository;
    this.grid = grid;
    this.isAdmin = isAdmin;

    setLayout(new BorderLayout(20, 20));
    setBackground(Color.WHITE);

    JPanel content = new JPanel(new BorderLayout(20, 20));
    content.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
    content.setBackground(Color.WHITE);

    JPanel formPanel = new JPanel(new GridLayout(4, 2, 10, 10));
    formPanel.setBackground(Color.WHITE);
    Font font = new Font("Segoe UI", Font.PLAIN, 16);

    idField = new JTextField();
    locationField = new JTextField();
    statusField = new JTextField();
    outputField = new JTextField();

    idField.setFont(font);
    locationField.setFont(font);
    statusField.setFont(font);
    outputField.setFont(font);

    formPanel.add(styledLabel("ID:"));
    formPanel.add(idField);
    formPanel.add(styledLabel("Location:"));
    formPanel.add(locationField);
    formPanel.add(styledLabel("Status:"));
    formPanel.add(statusField);
    formPanel.add(styledLabel("Energy Output (kWh):"));
    formPanel.add(outputField);

    JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15, 10));
```

```
buttonPanel.setBackground(Color.WHITE);

addBtn = styledButton("Add");
updateBtn = styledButton("Update");
deleteBtn = styledButton("Delete");
alertBtn = styledButton("Send Alert");
reportBtn = styledButton("Report");

if (isAdmin) {
    buttonPanel.add(addBtn);
    buttonPanel.add(updateBtn);
    buttonPanel.add(deleteBtn);
}

buttonPanel.add(alertBtn);
buttonPanel.add(reportBtn);

listModel = new DefaultListModel<>();
stationList = new JList<>(listModel);
stationList.setFont(new Font("Monospaced", Font.PLAIN, 14));
stationList.setSelectionBackground(new Color(0, 123, 255));
stationList.setSelectionForeground(Color.WHITE);
JScrollPane scrollPane = new JScrollPane(stationList);
scrollPane.setPreferredSize(new Dimension(400, 200));

loadPowerStations();

content.add(formPanel, BorderLayout.NORTH);
content.add(buttonPanel, BorderLayout.CENTER);
content.add(scrollPane, BorderLayout.SOUTH);

add(content, BorderLayout.CENTER);

stationList.addListSelectionListener(e -> {
    PowerStation ps = stationList.getSelectedValue();
    if (ps != null) {
        idField.setText(ps.getResourceID());
        locationField.setText(ps.getLocation());
        statusField.setText(ps.getStatus());
        outputField.setText(String.valueOf(ps.getEnergyOutput()));
    }
});

addBtn.addActionListener(e -> {
    String id = idField.getText();
```

```
        String loc = locationField.getText();
        String status = statusField.getText();
        double output = Double.parseDouble(outputField.getText());

        PowerStation ps = new PowerStation(id, loc, status, output, repository);
        repository.addResource(ps);
        listModel.addElement(ps);
        clearForm();
    });

    updateBtn.addActionListener(e -> {
        PowerStation selected = stationList.getSelectedValue();
        if (selected != null) {
            selected.setLocation(locationField.getText());
            selected.setStatus(statusField.getText());
            stationList.repaint();
        }
    });

    deleteBtn.addActionListener(e -> {
        PowerStation selected = stationList.getSelectedValue();
        if (selected != null) {
            repository.removeResource(selected);
            listModel.removeElement(selected);
            clearForm();
        }
    });

    alertBtn.addActionListener(e -> {
        PowerStation selected = stationList.getSelectedValue();
        if (selected != null) {
            selected.sendEmergencyAlert("Manual alert triggered from GUI");
            JOptionPane.showMessageDialog(this,
                "Emergency alert sent from PowerStation " + selected.getResourceID(),
                "Alert Sent", JOptionPane.WARNING_MESSAGE);
        }
    });


    reportBtn.addActionListener(e -> {
        PowerStation selected = stationList.getSelectedValue();
        if (selected != null) {
            JOptionPane.showMessageDialog(this, selected.generateUsageReport(), "Usage Report",
JOptionPane.INFORMATION_MESSAGE);
        }
```

```
        });

        if (!isAdmin) {
            disableAdminControls();
        }
    }

    public void loadPowerStations() {
        for (CityResource res : repository.getAllResources()) {
            if (res instanceof PowerStation) {
                listModel.addElement((PowerStation) res);
            }
        }
    }

    private void clearForm() {
        idField.setText("");
        locationField.setText("");
        statusField.setText("");
        outputField.setText("");
    }

    private void disableAdminControls() {
        idField.setEditable(false);
        locationField.setEditable(false);
        statusField.setEditable(false);
        outputField.setEditable(false);
        addBtn.setEnabled(false);
        updateBtn.setEnabled(false);
        deleteBtn.setEnabled(false);
    }

    private JButton styledButton(String text) {
        JButton btn = new JButton(text);
        btn.setBackground(new Color(0, 123, 255));
        btn.setForeground(Color.WHITE);
        btn.setFont(new Font("Segoe UI", Font.BOLD, 14));
        btn.setFocusPainted(false);
        btn.setCursor(new Cursor(Cursor.HAND_CURSOR));
        btn.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
        return btn;
    }

    private JLabel styledLabel(String text) {
        JLabel label = new JLabel(text);
```

```
        label.setFont(new Font("Segoe UI", Font.BOLD, 14));
        label.setForeground(new Color(50, 50, 50));
        return label;
    }
}
```

Key Features:

- Manages power station resources in the smart city system.
- Displays a list of existing power stations using JList.
- Admins can add, update, and delete power stations through form inputs.
- Provides buttons for sending emergency alerts and generating usage reports.
- GUI adapts based on user role (isAdmin flag); disables modification features for public users.
- Uses DefaultListModel for dynamic list updates and form binding for station editing.

OOP Concepts Used:

PowerStationPanel applies encapsulation by managing its internal state (UI components and data logic) privately. Composition is used to integrate objects like CityRepository, SmartGrid, and PowerStation into the panel. Polymorphism is utilized by handling CityResource as a parent type. Inheritance is applied by extending JPanel, allowing UI reuse and event-driven interaction. GUI responsiveness and behavior change dynamically based on the role, demonstrating clean modular design.

EmergencyServicePanel Class

```
package src.gui;

import javax.swing.*;

import src.model.CityRepository;
import src.model.CityResource;
import src.model.EmergencyService;

import java.awt.*;

public class EmergencyServicePanel extends JPanel {
    private CityRepository<CityResource> repository;
    private DefaultListModel<EmergencyService> listModel;
```

```
private JList<EmergencyService> serviceList;

private JTextField idField, locationField, statusField, typeField, timeField;
private JButton addBtn, updateBtn, deleteBtn, alertBtn, reportBtn;
private boolean isAdmin;

public EmergencyServicePanel(CityRepository<CityResource> repository, boolean isAdmin) {
    this.repository = repository;
    this.isAdmin = isAdmin;

    setLayout(new BorderLayout(20, 20));
    setBackground(Color.WHITE);

    JPanel content = new JPanel(new BorderLayout(20, 20));
    content.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
    content.setBackground(Color.WHITE);

    JPanel formPanel = new JPanel(new GridLayout(5, 2, 10, 10));
    formPanel.setBackground(Color.WHITE);
    Font font = new Font("Segoe UI", Font.PLAIN, 16);

    idField = new JTextField();
    locationField = new JTextField();
    statusField = new JTextField();
    typeField = new JTextField();
    timeField = new JTextField();

    idField.setFont(font);
    locationField.setFont(font);
    statusField.setFont(font);
    typeField.setFont(font);
    timeField.setFont(font);

    formPanel.add(styledLabel("ID:"));
    formPanel.add(idField);
    formPanel.add(styledLabel("Location:"));
    formPanel.add(locationField);
    formPanel.add(styledLabel("Status:"));
    formPanel.add(statusField);
    formPanel.add(styledLabel("Service Type:"));
    formPanel.add(typeField);
    formPanel.add(styledLabel("Response Time:"));
    formPanel.add(timeField);

    JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15, 10));
```

```
buttonPanel.setBackground(Color.WHITE);

addBtn = styledButton("Add");
updateBtn = styledButton("Update");
deleteBtn = styledButton("Delete");
alertBtn = styledButton("Send Alert");
reportBtn = styledButton("Report");

if (isAdmin) {
    buttonPanel.add(addBtn);
    buttonPanel.add(updateBtn);
    buttonPanel.add(deleteBtn);
}

buttonPanel.add(alertBtn);
buttonPanel.add(reportBtn);

listModel = new DefaultListModel<>();
serviceList = new JList<>(listModel);
serviceList.setFont(new Font("Monospaced", Font.PLAIN, 14));
serviceList.setSelectionBackground(new Color(0, 123, 255));
serviceList.setSelectionForeground(Color.WHITE);
JScrollPane scrollPane = new JScrollPane(serviceList);
scrollPane.setPreferredSize(new Dimension(400, 200));

listModel.clear();
loadEmergencyServices();

content.add(formPanel, BorderLayout.NORTH);
content.add(buttonPanel, BorderLayout.CENTER);
content.add(scrollPane, BorderLayout.SOUTH);

add(content, BorderLayout.CENTER);

serviceList.addListSelectionListener(e -> {
    EmergencyService es = serviceList.getSelectedValue();
    if (es != null) {
        idField.setText(es.getResourceID());
        locationField.setText(es.getLocation());
        statusField.setText(es.getStatus());
        typeField.setText(es.getServiceType());
        timeField.setText(String.valueOf(es.getResponseTime()));
    }
});
```

```
        addBtn.addActionListener(e -> {
            try {
                String id = idField.getText();
                String location = locationField.getText();
                String status = statusField.getText();
                String type = typeField.getText();
                int time = Integer.parseInt(timeField.getText());

                EmergencyService es = new EmergencyService(id, location, status, type, time);
                repository.addResource(es);
                listModel.addElement(es);
                clearForm();
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(this, "Response time must be a number", "Input Error",
JOptionPane.ERROR_MESSAGE);
            }
        });

        updateBtn.addActionListener(e -> {
            EmergencyService selected = serviceList.getSelectedValue();
            if (selected != null) {
                try {
                    selected.setLocation(locationField.getText());
                    selected.setStatus(statusField.getText());
                    selected.setServiceType(typeField.getText());
                    selected.setResponseTime(Integer.parseInt(timeField.getText()));
                    serviceList.repaint();
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(this, "Response time must be a number", "Input Error",
JOptionPane.ERROR_MESSAGE);
                }
            }
        });

        deleteBtn.addActionListener(e -> {
            EmergencyService selected = serviceList.getSelectedValue();
            if (selected != null) {
                repository.removeResource(selected);
                listModel.removeElement(selected);
                clearForm();
            }
        });

        alertBtn.addActionListener(e -> {
            EmergencyService selected = serviceList.getSelectedValue();
```

```
        if (selected != null) {
            selected.sendEmergencyAlert("Manual alert from GUI");
            JOptionPane.showMessageDialog(this,
                    "Alert Sent!\n[" + selected.getServiceType() + "] - Manual alert from GUI",
                    "Alert Notification",
                    JOptionPane.WARNING_MESSAGE);
        }
    });

    reportBtn.addActionListener(e -> {
        EmergencyService selected = serviceList.getSelectedValue();
        if (selected != null) {
            String report = selected.generateUsageReport();
            double cost = selected.calculateMaintenanceCost();
            JOptionPane.showMessageDialog(this,
                    report + "\nMaintenance Cost: Rs. " + String.format("%.2f", cost),
                    "Usage Report",
                    JOptionPane.INFORMATION_MESSAGE);
        }
    });


    if (!isAdmin) {
        disableAdminControls();
    }
}

private void loadEmergencyServices() {
    for (CityResource res : repository.getAllResources()) {
      System.out.println("Type of resource: " + res.getClass().getName());
        if (res instanceof EmergencyService) {
            listModel.addElement((EmergencyService) res);
        }
    }
}

private void clearForm() {
    idField.setText("");
    locationField.setText("");
    statusField.setText("");
    typeField.setText("");
    timeField.setText("");
}

private void disableAdminControls() {
```

```
        idField.setEditable(false);
        locationField.setEditable(false);
        statusField.setEditable(false);
        typeField.setEditable(false);
        timeField.setEditable(false);
        addBtn.setEnabled(false);
        updateBtn.setEnabled(false);
        deleteBtn.setEnabled(false);
    }

    private JButton styledButton(String text) {
        JButton btn = new JButton(text);
        btn.setBackground(new Color(0, 123, 255));
        btn.setForeground(Color.WHITE);
        btn.setFont(new Font("Segoe UI", Font.BOLD, 14));
        btn.setFocusPainted(false);
        btn.setCursor(new Cursor(Cursor.HAND_CURSOR));
        btn.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
        return btn;
    }

    private JLabel styledLabel(String text) {
        JLabel label = new JLabel(text);
        label.setFont(new Font("Segoe UI", Font.BOLD, 14));
        label.setForeground(new Color(50, 50, 50));
        return label;
    }
}
```

Key Features:

- Manages emergency service units in the smart city system.
- GUI includes input fields for ID, location, status, service type, and response time.
- Admins can add, update, and delete emergency services.
- All users can trigger alerts and view usage reports with maintenance cost.
- Dynamically populates and updates a JList using DefaultListModel.
- Automatically adapts form accessibility and buttons based on isAdmin role.
- Ensures data validation, e.g., response time must be numeric.

OOP Concepts Used:

EmergencyServicePanel demonstrates encapsulation by organizing GUI logic and behavior inside a cohesive panel class. Polymorphism is applied through the use of CityResource references. Inheritance is used as the panel extends JPanel, integrating into the GUI layout seamlessly. Composition is seen in the inclusion of CityRepository and EmergencyService objects for

functionality. The GUI's dynamic behavior and input validation show a clean separation of concerns and strong modular object-oriented design.

TransportUnitPanel Class

```
package src.gui;


import javax.swing.*;

import src.model.CityRepository;
import src.model.CityResource;
import src.model.TransportUnit;

import java.awt.*;

public class TransportUnitPanel extends JPanel {
    private CityRepository<CityResource> repository;
    private DefaultListModel<TransportUnit> listModel;
    private JList<TransportUnit> transportList;

    private JTextField idField, routeField, capacityField, fuelCostField;
    private JComboBox<String> statusCombo;
    private JButton addBtn, updateBtn, deleteBtn, alertBtn, reportBtn;
    private boolean isAdmin;

    public TransportUnitPanel(CityRepository<CityResource> repository, boolean isAdmin) {
        this.repository = repository;
        this.isAdmin = isAdmin;

        setLayout(new BorderLayout(20, 20));
        setBackground(Color.WHITE);

        JPanel content = new JPanel(new BorderLayout(20, 20));
        content.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
        content.setBackground(Color.WHITE);

        JPanel formPanel = new JPanel(new GridLayout(5, 2, 10, 10));
        formPanel.setBackground(Color.WHITE);
        Font font = new Font("Segoe UI", Font.PLAIN, 16);

        idField = new JTextField();
        routeField = new JTextField();
```

```
capacityField = new JTextField();
fuelCostField = new JTextField();
statusCombo = new JComboBox<>(new String[]{
    "Active", "Inactive", "Under Maintenance", "Out of Service"
});

idField.setFont(font);
routeField.setFont(font);
capacityField.setFont(font);
fuelCostField.setFont(font);
statusCombo.setFont(font);

formPanel.add(styledLabel("ID:"));
formPanel.add(idField);
formPanel.add(styledLabel("Route:"));
formPanel.add(routeField);
formPanel.add(styledLabel("Capacity:"));
formPanel.add(capacityField);
formPanel.add(styledLabel("Fuel Cost/KM:"));
formPanel.add(fuelCostField);
formPanel.add(styledLabel("Status:"));
formPanel.add(statusCombo);

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15, 10));
buttonPanel.setBackground(Color.WHITE);

addBtn = styledButton("Add");
updateBtn = styledButton("Update");
deleteBtn = styledButton("Delete");
alertBtn = styledButton("Send Alert");
reportBtn = styledButton("Report");

if (isAdmin) {
    buttonPanel.add(addBtn);
    buttonPanel.add(updateBtn);
    buttonPanel.add(deleteBtn);
}

buttonPanel.add(alertBtn);
buttonPanel.add(reportBtn);

listModel = new DefaultListModel<>();
transportList = new JList<>(listModel);
transportList.setFont(new Font("Monospaced", Font.PLAIN, 14));
transportList.setSelectionBackground(new Color(0, 123, 255));
```

```
transportList.setSelectionForeground(Color.WHITE);
JScrollPane scrollPane = new JScrollPane(transportList);
scrollPane.setPreferredSize(new Dimension(400, 200));

loadTransportUnits();

content.add(formPanel, BorderLayout.NORTH);
content.add(buttonPanel, BorderLayout.CENTER);
content.add(scrollPane, BorderLayout.SOUTH);

add(content, BorderLayout.CENTER);

transportList.addListSelectionListener(e -> {
    TransportUnit tu = transportList.getSelectedValue();
    if (tu != null) {
        idField.setText(tu.getResourceID());
        routeField.setText(tu.getLocation());
        capacityField.setText(String.valueOf(tu.getPassengerCount()));
        fuelCostField.setText(String.valueOf(tu.getFuelCostPerKm()));
        statusCombo.setSelectedItem(tu.getStatus());
    }
});

addBtn.addActionListener(e -> {
    String id = idField.getText();
    String route = routeField.getText();
    int capacity = Integer.parseInt(capacityField.getText());
    double fuelCost = Double.parseDouble(fuelCostField.getText());
    String status = (String) statusCombo.getSelectedItem();

    TransportUnit tu = new TransportUnit(id, route, status, capacity, fuelCost);
    repository.addResource(tu);
    listModel.addElement(tu);
    clearForm();
});

updateBtn.addActionListener(e -> {
    TransportUnit selected = transportList.getSelectedValue();
    if (selected != null) {
        selected.setLocation(routeField.getText());
        selected.setPassengerCount(Integer.parseInt(capacityField.getText()));
        selected.setFuelCostPerKm(Double.parseDouble(fuelCostField.getText()));
        selected.setStatus((String) statusCombo.getSelectedItem());
        transportList.repaint();
    }
```

```
        });

        deleteBtn.addActionListener(e -> {
            TransportUnit selected = transportList.getSelectedValue();
            if (selected != null) {
                repository.removeResource(selected);
                listModel.removeElement(selected);
                clearForm();
            }
        });

        alertBtn.addActionListener(e -> {
            TransportUnit selected = transportList.getSelectedValue();
            if (selected != null) {
                selected.sendEmergencyAlert("Manual alert triggered from GUI");
                JOptionPane.showMessageDialog(this, "Emergency alert sent!\nMessage: Manual alert triggered
from GUI", "Alert", JOptionPane.WARNING_MESSAGE);
            }
        });


        reportBtn.addActionListener(e -> {
            TransportUnit selected = transportList.getSelectedValue();
            if (selected != null) {
                String report = selected.generateUsageReport() + "\nMaintenance Cost: Rs. " +
selected.calculateMaintenanceCost();
                JOptionPane.showMessageDialog(this, report, "Transport Report",
JOptionPane.INFORMATION_MESSAGE);
            }
        });

        if (!isAdmin) {
            disableAdminControls();
        }
    }

    private void loadTransportUnits() {
        for (CityResource res : repository.getAllResources()) {
            if (res instanceof TransportUnit) {
                listModel.addElement((TransportUnit) res);
            }
        }
    }

    private void clearForm() {
```

```
        idField.setText("");
        routeField.setText("");
        capacityField.setText("");
        fuelCostField.setText("");
        statusCombo.setSelectedIndex(0);
    }

    private void disableAdminControls() {
        idField.setEditable(false);
        routeField.setEditable(false);
        capacityField.setEditable(false);
        fuelCostField.setEditable(false);
        statusCombo.setEnabled(false);
        addBtn.setEnabled(false);
        updateBtn.setEnabled(false);
        deleteBtn.setEnabled(false);
    }

    private JButton styledButton(String text) {
        JButton btn = new JButton(text);
        btn.setBackground(new Color(0, 123, 255));
        btn.setForeground(Color.WHITE);
        btn.setFont(new Font("Segoe UI", Font.BOLD, 14));
        btn.setFocusPainted(false);
        btn.setCursor(new Cursor(Cursor.HAND_CURSOR));
        btn.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
        return btn;
    }

    private JLabel styledLabel(String text) {
        JLabel label = new JLabel(text);
        label.setFont(new Font("Segoe UI", Font.BOLD, 14));
        label.setForeground(new Color(50, 50, 50));
        return label;
    }
}
```

Key Features:

- Manages public or private transport units in the smart city system.
- GUI has fields for ID, route, passenger capacity, fuel cost, and status selection.
- Admin functionalities include adding, updating, and deleting transport units.
- All users can send alerts and generate reports showing usage and maintenance costs.

- Data is displayed in a styled JList with dynamic updates using DefaultListModel.
- Validates user input (e.g., parsing capacity and fuel cost from JTextField).
- UI dynamically adapts based on the user's role (isAdmin flag).

OOP Concepts Used:

TransportUnitPanel leverages encapsulation by packaging GUI and logic into a well-structured component. It uses composition with CityRepository and TransportUnit objects to manage data. The system exhibits polymorphism through the use of CityResource as a generic parent type. Extending JPanel showcases inheritance. Input validation, role-based control toggling, and modular structure represent strong object-oriented design principles in action.

ConsumerPanel Class

```
package src.gui;

import javax.swing.*;

import src.model.Household;
import src.model.Industry;

import java.awt.*;
import java.util.ArrayList;

public class ConsumerPanel extends JPanel {
    private ArrayList<Household> households;
    private ArrayList<Industry> industries;
    private DefaultListModel<String> householdModel = new DefaultListModel<>();
    private DefaultListModel<String> industryModel = new DefaultListModel<>();
    private boolean isAdmin;
    private JLabel totalLabel;

    public ConsumerPanel(ArrayList<Household> households, ArrayList<Industry> industries, boolean
isAdmin) {
        this.households = households;
        this.industries = industries;
        this.isAdmin = isAdmin;

        for (Household h : households) {
            householdModel.addElement(h.toString());
        }

        for (Industry i : industries) {
```

```
            industryModel.addElement(i.toString());
        }

        JPanel householdPanel = buildConsumerPanel("Households", householdModel, households, true);
        JPanel industryPanel = buildConsumerPanel("Industries", industryModel, industries, false);

        totalLabel = styledLabel(" 🔋 Total Energy Consumed: " + getOverallConsumption() + " kWh");
        totalLabel.setHorizontalAlignment(SwingConstants.CENTER);
        totalLabel.setFont(new Font("Segoe UI", Font.BOLD, 16));
        totalLabel.setBorder(BorderFactory.createEmptyBorder(10, 0, 0, 0));

        setLayout(new BorderLayout());
        setBackground(Color.WHITE);
        setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

        JPanel listsPanel = new JPanel(new GridLayout(1, 2, 20, 0));
        listsPanel.setBackground(Color.WHITE);
        listsPanel.add(householdPanel);
        listsPanel.add(industryPanel);

        add(listsPanel, BorderLayout.CENTER);
        add(totalLabel, BorderLayout.SOUTH);
    }

    private JPanel buildConsumerPanel(String title, DefaultListModel<String> model, ArrayList<?> dataList,
boolean isHousehold) {
        JPanel panel = new JPanel(new BorderLayout(10, 10));
        panel.setBackground(Color.WHITE);
        panel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorder(Color.GRAY),
title));

        JList<String> list = new JList<>(model);
        list.setFont(new Font("Monospaced", Font.PLAIN, 14));
        list.setSelectionBackground(new Color(0, 123, 255));
        list.setSelectionForeground(Color.WHITE);

        JScrollPane scrollPane = new JScrollPane(list);
        scrollPane.setPreferredSize(new Dimension(400, 200));
        panel.add(scrollPane, BorderLayout.CENTER);

        if (isAdmin) {
            JPanel form = new JPanel(new GridLayout(3, 2, 10, 10));
            form.setBackground(Color.WHITE);
            JTextField idField = new JTextField();
            JTextField consumedField = new JTextField();
```

```
        JButton addBtn = styledButton("Add");
        JButton delBtn = styledButton("Delete");

        form.add(styledLabel("ID:"));
        form.add(idField);
        form.add(styledLabel("Energy Consumed:"));
        form.add(consumedField);
        form.add(addBtn);
        form.add(delBtn);
        panel.add(form, BorderLayout.SOUTH);

        addBtn.addActionListener(e -> {
            try {
                String id = idField.getText();
                double consumed = Double.parseDouble(consumedField.getText());

                if (isHousehold) {
                    Household h = new Household(id, consumed);
                    households.add(h);
                    model.addElement(h.toString());
                } else {
                    Industry i = new Industry(id, consumed);
                    industries.add(i);
                    model.addElement(i.toString());
                }

                updateTotalLabel();
                idField.setText("");
                consumedField.setText("");
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(this, "Enter valid numeric value for energy!", "Invalid Input",
JOptionPane.ERROR_MESSAGE);
            }
        });

            delBtn.addActionListener(e -> {
                int index = list.getSelectedIndex();
                if (index >= 0) {
                    if (isHousehold) {
                        Household removed = households.remove(index);
                        Household.decreaseTotal(removed.getEnergyConsumed());
                    } else {
                        Industry removed = industries.remove(index);
                        Industry.decreaseTotal(removed.getEnergyConsumed());
```

```
                }
                model.remove(index);
                updateTotalLabel();
            }
        });

    }

    return panel;
}

private void updateTotalLabel() {
    if (totalLabel != null) {
        totalLabel.setText(" 🔋 Total Energy Consumed: " + getOverallConsumption() + " kWh");
    }
}

private JLabel styledLabel(String text) {
    JLabel label = new JLabel(text);
    label.setFont(new Font("Segoe UI", Font.BOLD, 14));
    label.setForeground(new Color(60, 60, 60));
    return label;
}

private JButton styledButton(String text) {
    JButton btn = new JButton(text);
    btn.setFont(new Font("Segoe UI", Font.BOLD, 14));
    btn.setBackground(new Color(0, 123, 255));
    btn.setForeground(Color.WHITE);
    btn.setFocusPainted(false);
    btn.setCursor(new Cursor(Cursor.HAND_CURSOR));
    btn.setBorder(BorderFactory.createEmptyBorder(8, 16, 8, 16));
    return btn;
}

public static double getOverallConsumption() {
    return Household.getTotalConsumed() + Industry.getTotalConsumed();
}
}
```

Key Features:

- Displays and manages energy consumption data for both Households and Industries.
- Uses two JList components for listing households and industries separately.

- Admin users can add and delete consumers (households or industries) using a form below each list.
- Input validation ensures correct numeric entry for energy consumption.
- Shows live total energy consumption from both sources combined at the bottom.
- UI updates automatically when data is added or removed.
- Encapsulates all logic for UI display, consumer addition/removal, and total consumption update.

OOP Concepts Used:

ConsumerPanel demonstrates composition by managing Household and Industry objects and their interactions. It uses encapsulation to manage form logic and UI control privately. It supports polymorphism by working generically with both consumer types in similar ways, and applies static members (getTotalConsumed, decreaseTotal) to track shared energy usage efficiently. The panel also reflects role-based access control (admin vs. non-admin) through conditional GUI rendering.

BackPanel Class

```
package src.gui;
import javax.swing.*;

import src.LoginFrame;

import java.awt.*;

public class BackPanel extends JPanel {

  FileManagerFront f;

  public BackPanel(JFrame currentFrame, FileManagerFront ff) {
    this.f = ff;
    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
    setBorder(BorderFactory.createEmptyBorder(50, 50, 50, 50));

    JLabel msg = new JLabel("Save & Go Back to Login?");
    msg.setAlignmentX(Component.CENTER_ALIGNMENT);
    msg.setFont(new Font("Arial", Font.BOLD, 16));

    JButton backBtn = new JButton("Logout");
    backBtn.setAlignmentX(Component.CENTER_ALIGNMENT);
    backBtn.setBackground(new Color(0, 123, 255));
    backBtn.setForeground(Color.WHITE);
    backBtn.setFont(new Font("Arial", Font.BOLD, 14));
```

```
    backBtn.setCursor(new Cursor(Cursor.HAND_CURSOR));
    backBtn.setFocusPainted(false);
    backBtn.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

    backBtn.addActionListener(e -> {
      saveAll();
      currentFrame.dispose();
      new LoginFrame();
    });

    add(msg);
    add(Box.createVerticalStrut(20));
    add(backBtn);
  }

  private void saveAll() {
    System.out.println("Saving all data before exiting.....");
    f.saveData();
  }
}
```

Key Features:

- Simple logout panel prompting the user to save data and return to login.
- Presents a confirmation message and a "Logout" button styled with custom colors and fonts.
- On logout:
- Triggers a call to saveAll() → invokes FileManagerFront.saveData() to persist all data.
- Closes the current window.
- Launches the LoginFrame again.
- Uses BoxLayout to center-align elements vertically with spacing.

OOP Concepts Used:

This class demonstrates composition, as it relies on another object (FileManagerFront) to perform save operations before logout. It uses encapsulation by keeping the save logic private and isolated in saveAll(). The constructor showcases dependency injection, taking both a JFrame and FileManagerFront as arguments for modular, testable design.

FileManagerFront Class

```
package src.gui;
```

```
import javax.swing.*;

import src.model.CityRepository;
import src.model.CityResource;
import src.model.CityZone;
import src.model.FileManager;
import src.model.Household;
import src.model.Industry;
import src.model.PowerStation;
import src.model.SmartGrid;

import java.util.ArrayList;

public class FileManagerFront extends JPanel {
    private CityRepository<CityResource> repository;
    private ArrayList<Household> households;
    private ArrayList<Industry> industries;
    private boolean isAdmin;

    private final String RESOURCE_FILE = "resources.dat";
    private final String ZONE_FILE = "zones.dat";
    private final String GRID_FILE = "grid.dat";
    private final String HOUSEHOLD_FILE = "households.dat";
    private final String INDUSTRY_FILE = "industries.dat";

    public FileManagerFront(CityRepository<CityResource> repository, ArrayList<CityZone> zones,
SmartGrid smartGrid,
                            ArrayList<Household> households, ArrayList<Industry> industries, boolean isAdmin) {
        this.repository = repository;
        this.households = households;
        this.industries = industries;
        this.isAdmin = isAdmin;
        loadData();
    }

    public void saveData() {
        try {
            if (!isAdmin) return;
            FileManager.saveToFile(new ArrayList<>(repository.getAllResources()), RESOURCE_FILE);
            FileManager.saveToFile(households, HOUSEHOLD_FILE);
            FileManager.saveToFile(industries, INDUSTRY_FILE);
            System.out.println("Saved Yippe.....");
        } catch (Exception ex) {
            System.out.println("Error saving data: " + ex.getMessage());
```

```
    }
  }

  public void loadData() {
    try {
      ArrayList<CityResource> loadedResources = FileManager.loadFromFile(RESOURCE_FILE);
      ArrayList<CityZone> loadedZones = FileManager.loadFromFile(ZONE_FILE);
      ArrayList<PowerStation> loadedGrid = FileManager.loadFromFile(GRID_FILE);
      ArrayList<Household> loadedHouseholds = FileManager.loadFromFile(HOUSEHOLD_FILE);
      ArrayList<Industry> loadedIndustries = FileManager.loadFromFile(INDUSTRY_FILE);

      repository.clear();
      for (CityResource res : loadedResources) repository.addResource(res);

      households.clear();
      households.addAll(loadedHouseholds);

      industries.clear();
      industries.addAll(loadedIndustries);

      System.out.println("Data loaded successfully");
    } catch (Exception ex) {
      System.out.println("Error loading data: " + ex.getMessage());
    }
  }
}
```

Key Features:

- Acts as a bridge between frontend and backend file handling.
- Handles persistent storage for:
- CityResource objects (e.g. transport, emergency units)
- Household and Industry consumer records
- Uses Java's serialization through the FileManager utility class to save and load .dat files.
- Automatically loads data when the object is created.
- saveData() only executes if the user is an admin.
- Gracefully handles exceptions and logs status messages on save/load.

OOP Concepts Used:

This class demonstrates encapsulation by managing all file interaction logic internally. It applies composition, containing and operating on multiple model objects (CityRepository, Household, Industry, etc.). Polymorphism is used during the loadFromFile process (handling different object

types). It follows separation of concerns—delegating raw file I/O to FileManager and focusing only on coordination logic.

Complexity Additions

Resource Dependencies

A PowerStation outage triggers alerts to nearby Emergency Service units.

```
    @Override
    public void sendEmergencyAlert(String msg) {
        System.out.println("ALERT: PowerStation " + getResourceID() + ": " +
msg);
        System.out.println("PowerStation " + resourceID + ": Output=" +
energyOutput + " kWh, Consumption=" + getTotalConsumption() + " kWh");

        if (repository != null) {
            for (CityResource res : repository.getAllResources()) {
                if (res instanceof EmergencyService) {
                    EmergencyService es = (EmergencyService) res;
                    if (es.getLocation().equalsIgnoreCase(this.location) ||
es.getStatus().equalsIgnoreCase("Ready")) {
                        System.out.println("[PowerStation -> Emergency Alert]
Notifying Emergency Service: " + es.getResourceID());
                        es.sendEmergencyAlert("Overload at PowerStation " +
this.resourceID + " in zone " + this.location);
                    }
                }
            }
        } else {
            System.out.println("⚠ No repository linked to PowerStation. Cannot
notify Emergency Services.");
        }
    }
```

This method is designed to handle emergency situations in a PowerStation. When triggered, it:Logs a local alert with power station ID, current energy output, and consumption.Checks for a linked CityRepository — if connected, it loops through all city resources.Finds and notifies Emergency Services whose location matches the station's or whose status is "Ready".Sends them a detailed alert message about the overload or issue.If no repository is linked, it prints a warning

saying it can't notify emergency services. This ensures that any overload or malfunction can be communicated automatically to relevant responders in the system.

Role-Based Access

GUI panels vary for admin (full control) vs. public (view-only).

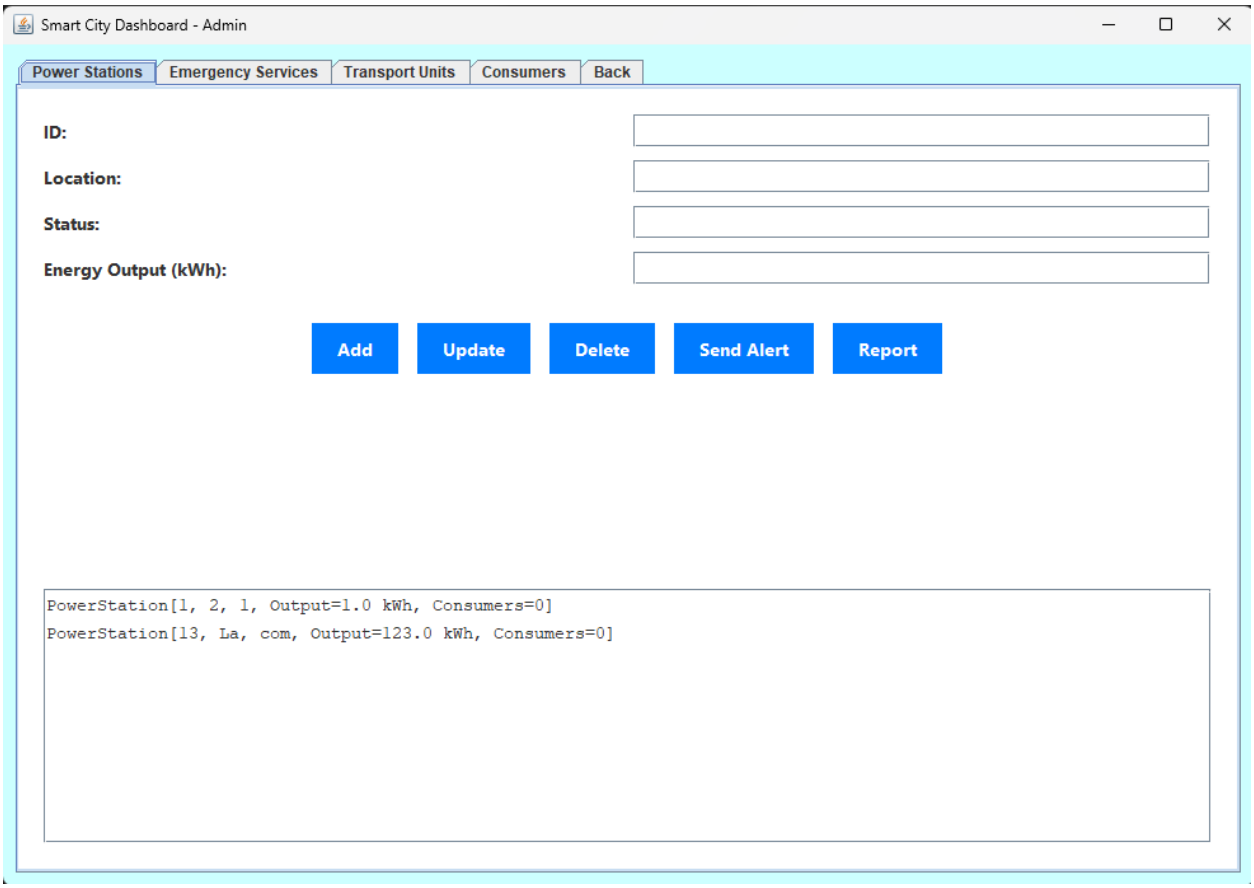

Figure 2 Admin View

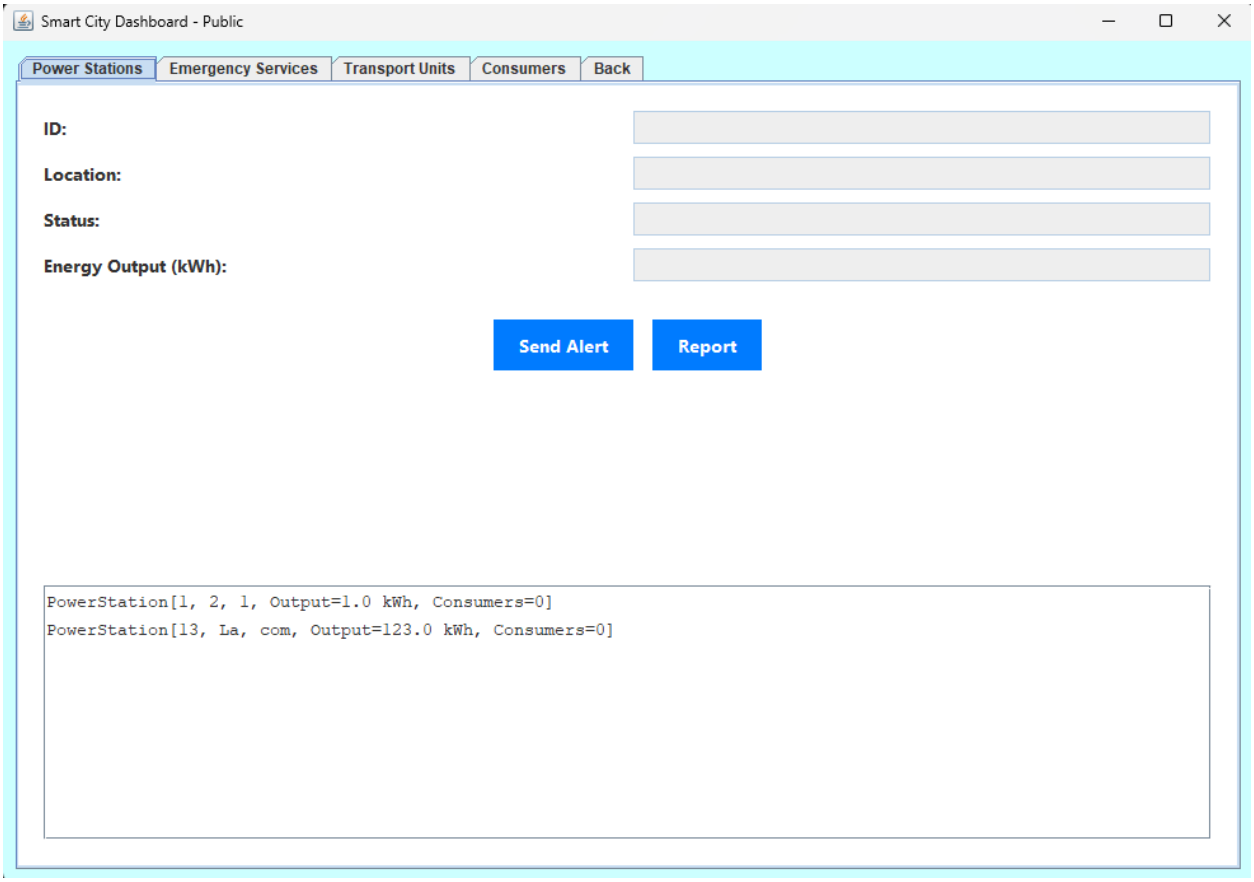

Figure 3 Public View

Test Cases Verified

During the development and testing phase of the Smart City Resource Management System, the following core functionalities were manually verified to ensure correct behavior:

PowerStation

- Verified emergency alert pop up.
- Tested maintenance cost calculation logic.

TransportUnit

- Successfully added, updated, and deleted transport units from the repository.
- Validated capacity and fuel cost fields.
- Verified alert and report generation for selected transport units.

EmergencyService

- Alert messages logged appropriately.
- Usage Reports

FileManager

- Save and load functions tested with .dat files for CityResources, Households, and Industries.
- Confirmed admin-only save permissions.
- Verified successful data loading with appropriate console messages.

GUI Operations

- Admin users had access to add, update, and delete operations.
- Non-admin users had limited access as expected.
- Report buttons and alert functionality worked correctly in both modes.

Summary

Smart City Resource Management System is a Java Swing-based desktop application designed to efficiently manage various city resources such as transport units, emergency services, power stations, households, and industries. It follows Object-Oriented Programming principles including inheritance, composition, interfaces, and generics. The system supports role-based access: admin users can add, update, delete, generate reports, send alerts, and save/load data, while consumers can only view information. All data is saved using file serialization to .dat files through a custom FileManager class. The user interface is clean and interactive, offering real-time alerts and detailed usage reports. Overall, the system is modular, scalable, and ready for future extensions.

Project Enhancements (Future Scope)

Currently, all core backend logic is fully implemented, including alerts, reports, file handling, and resource management. However, due to certain project restrictions, advanced UI elements such as real-time maps and JFreeChart-based graphs were not integrated. In future versions, interactive maps can be used to display City Zones or Smart Grids, and charts can visually represent energy usage, emergency response statistics, and transport metrics. Many other features like live dashboards, database integration, and real-time analytics can also be added as future improvements to make the system more powerful and user-friendly.