



MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Object Oriented Programming
Mini Project Report on
AGRICULTURAL MANAGEMENT
SYSTEM

SUBMITTED
BY

Vinay Chilakamarri	45	230905398
Md Amaan	40	230905364
Spandan Chatterjee	56	230905488
Leah Phillip	11	230905079
Harsha N K	58	230905498

Section D

Under the Guidance of:

Dr. Rajashree Krishna

Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal, Karnataka – 576104

October 2024

INTRODUCTION

In today's agricultural landscape, effective information management and resource accessibility are crucial for both farmers and government agencies. Our Agriculture Management System (AMS) was developed as a centralized platform aimed at improving efficiency and access to resources for both farmers and government employees. With a user-friendly interface and role-based access, the AMS facilitates the management of subsidies, crop-specific schemes, market prices, and farmer registrations.

The system's role-based dashboard ensures that each type of user has access to features tailored to their specific needs and responsibilities. **Farmers** can view crop-related schemes, register with essential details like crop type and land size, access market prices, and check available subsidies, helping them make informed decisions about crop production and financial support. **Government employees**, on the other hand, are provided with additional administrative capabilities, including adding new schemes, updating subsidies, viewing registered farmers, and managing overall resource distribution. Access to these extended features is restricted to government personnel through a secure login system.

Key features of the Agriculture Management System include:

- **Role-Based Dashboards:** Separate dashboards for farmers and government employees allow each user type to access only relevant features. Farmers can view schemes and subsidies based on their registered crop type, while government employees have administrative controls to manage schemes, subsidies, and view farmer data.
- **Farmer Registration and Management:** Farmers can register themselves in the system, entering details such as their name, Aadhar number, crop type, and land size. This data is stored securely, providing a reliable record that government employees can access as needed.
- **Scheme Management:** Government employees can add new schemes to the system, specifying crop types and subsidy details to support targeted agricultural development. Farmers only see schemes that match their crop type, ensuring they access relevant information to aid their specific agricultural needs.
- **Subsidy Management:** A dedicated subsidy management feature allows government employees to add and update subsidies linked to specific crops or farming practices. Farmers can view available subsidies on their dashboard, helping them access financial support that aligns with their farming activities and crop types.
- **Market Price Updates:** The system provides access to real-time market prices, helping farmers stay informed about crop values and make strategic decisions regarding selling their produce.

- **Authentication and Security:** Role-based access ensures that only government employees have administrative privileges, while farmers securely access their dashboard with unique credentials. This setup minimizes unauthorized access and protects sensitive data.
- **JavaFX-Based User Interface:** The AMS employs JavaFX, a modern UI framework for Java, to offer a clean and interactive user experience. FXML files define each view's layout, while controller classes manage the logic behind the interface, providing a seamless flow of information.

By integrating these features, the Agriculture Management System aims to bridge the gap between government resource management and farmer needs, providing a streamlined, organized platform for information exchange, subsidy distribution, and resource allocation. This modular, role-based design approach supports future scalability, allowing for the addition of more services or user roles as needed.

PROBLEM STATEMENT

The agricultural sector faces numerous challenges that impede efficiency, transparency, and communication between farmers and government bodies. Traditional systems often lack the integration necessary to provide real-time data and support, leaving farmers without the resources they need to make informed decisions. To address these issues, we are developing an Agriculture Management System (AMS) using Java, which will incorporate key Object-Oriented Programming (OOP) concepts such as exception handling, JavaFX for the user interface, and generics for data management.

- **Farmer Registration:** Farmers need an accessible way to register and manage their profiles. Utilizing **exception handling**, our system will ensure that any errors during the registration process (such as invalid data inputs) are gracefully managed, providing clear feedback to the user.
- **Subsidy and Resource Management:** The system will allow government bodies to efficiently manage subsidies and resources. By employing **generics**, we can create a flexible structure for handling various resource types—whether they be seeds, fertilizers, or equipment—allowing for easy scalability and maintenance of the code.
- **Real-Time Data:** To provide farmers with up-to-date market prices and weather forecasts, we will leverage **JavaFX** for a dynamic and interactive user interface. This will

enhance user experience by allowing farmers to access crucial information seamlessly, with real-time updates displayed in a user-friendly format.

- **Communication and Support:** Establishing a two-way communication channel requires robust error handling to ensure that messages and requests between farmers and government agencies are processed smoothly. Our use of **exception handling** will ensure that any communication failures are managed effectively, allowing users to retry or receive notifications about the status of their requests.
- **Monitoring and Reporting:** For government bodies to monitor crop production and resource distribution effectively, our system will incorporate reporting features that utilize **generics**. This will enable the creation of versatile report templates that can be adapted to different data types, ensuring that the government can generate comprehensive insights for data-driven decision-making.

IMPLEMENTATION DETAILS

In our group project, we developed an Agriculture Management System (AMS) designed to provide a centralized platform for farmer registration and resource management, along with real-time access to market prices. We implemented this system using Object-Oriented Programming (OOP) principles, which helped us structure our code for maintainability and scalability.

Core Classes

1. **User Class:** We created an abstract User class to manage user details and differentiate roles between farmers and government handlers. This class encapsulates common attributes such as username and password while providing methods for login and registration.
2. **Farmer Class:** The Farmer class extends the User class, encapsulating attributes specific to farmers, including name, email, Aadhar number, land size, and crop type. This design allows us to tailor functionalities related to farmer management.
3. **Scheme Class:** We implemented a Scheme class to represent various agricultural schemes. This class contains attributes related to scheme details and associated crop types, allowing us to manage and display schemes effectively.
4. **Market Price Class:** A MarketPrice class was introduced to track and manage the current market prices for different crops. This class provides methods to update and retrieve market prices, ensuring that farmers have access to vital economic information.
5. **Service Classes:** We structured our business logic using service classes:
 - **AuthService:** This class handles user authentication, including methods for user registration and login.

- **FarmerService:** It manages farmer data, providing functionalities to add, retrieve, and update farmer information.
 - **SchemeService:** This service deals with operations related to agricultural schemes, allowing users to add new schemes and fetch them based on crop type.
 - **MarketPriceService:** This service is responsible for managing market prices, including retrieving and updating the latest prices for various crops.
6. **Database Integration:** To manage data persistence, we developed the SQLiteSetup class to handle database connections and implement CRUD (Create, Read, Update, Delete) operations for farmers, schemes, and market prices.
 7. **User Interface:** We utilized JavaFX to create a responsive user interface, with FXML files for different views, including FarmerRegistrationView, MainView, and SchemeManagementView. Each view is linked to its respective controller (e.g., MainController, FarmerRegistrationController) that handles user interactions.
 8. **Exception Handling:** Exception handling was essential in ensuring the system's stability and user experience. In various parts of our application, we implemented exception handling mechanisms to catch and manage potential errors gracefully:
 - **Input Validation:** In the FarmerRegistrationController, we added checks to prevent errors caused by incorrect data entries, such as non-numeric input for fields that require numbers (e.g., land size). For instance, invalid entries trigger a NumberFormatException, which we catch and handle by displaying an error message to guide the user.
 - **Database Connection Errors:** We implemented try-catch blocks in the SQLiteSetup class to handle database connection issues. If a connection fails, the application logs the error and informs the user of connectivity issues without crashing.
 - **File Handling:** For file-based operations, such as loading FXML files with FXMLLoader, we catch IOException to ensure that any missing or unreadable files do not interrupt the user experience.

By incorporating exception handling, we were able to enhance the application's reliability, reduce the likelihood of crashes, and provide meaningful feedback to users when issues arise.

OOP Concepts Applied

Throughout the project, we leveraged several OOP concepts to enhance our system's structure:

- **Encapsulation:** We made attributes of classes like Farmer, Scheme, and MarketPrice private, using public getter and setter methods to control access and modification, ensuring data integrity.
- **Inheritance:** By having the Farmer and GovernmentHandler classes inherit from the User base class, we were able to reuse code and streamline user management.

- **Polymorphism:** We implemented polymorphism by defining methods in the User class and overriding them in subclasses, allowing different user roles to have tailored functionalities, such as the viewSchemes() method.
- **Composition:** Instead of relying solely on inheritance, we utilized composition by having classes like Farmer contain instances of Scheme and MarketPrice to model relationships effectively.

PROJECT CODES

AddSchemeController.java:

```
package com.agriculture.app;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.TextField;
import javafx.scene.control.Alert.AlertType;

public class AddSchemeController {

    @FXML private TextField schemeNameField;
    @FXML private TextField descriptionField;
    @FXML private TextField eligibilityCriteriaField;
    @FXML private TextField regionField;
    @FXML private TextField cropTypeField;

    private SchemeService schemeService = new SchemeService();

    @FXML
    private void handleAddScheme() {
        String schemeName = schemeNameField.getText();
        String description = descriptionField.getText();
        String eligibilityCriteria = eligibilityCriteriaField.getText();
        String region = regionField.getText();
        String cropType = cropTypeField.getText();

        Scheme newScheme = new Scheme(schemeName, description, eligibilityCriteria, region, cropType);
        schemeService.addScheme(newScheme);

        // Clear fields after submission
        schemeNameField.clear();
        descriptionField.clear();
        eligibilityCriteriaField.clear();
        regionField.clear();
        cropTypeField.clear();
    }
}
```

```

        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("Success");
        alert.setHeaderText("Scheme Added");
        alert.setContentText("The scheme was added successfully!");
        alert.showAndWait();
    }
}

```

AddSubsidyController.java:

```

package com.agriculture.app;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class AddSubsidyController {

    @FXML
    private TextField subsidyNameField;

    @FXML
    private TextField descriptionField;

    @FXML
    private TextField eligibilityCriteriaField;

    @FXML
    private TextField regionField;

    @FXML
    private Button addSubsidyButton;

    private final SubsidyService subsidyService = new SubsidyService();

    @FXML
    public void initialize() {
        // Optional: Add initialization code here if needed
    }

    @FXML
    public void handleAddSubsidy() {
        String subsidyName = subsidyNameField.getText().trim();
        String description = descriptionField.getText().trim();
        String eligibilityCriteria = eligibilityCriteriaField.getText().trim();
    }
}

```



```
String region = regionField.getText().trim();
```

```
// Validate input
```

```
if (subsidyName.isEmpty() || description.isEmpty() || eligibilityCriteria.isEmpty() || region.isEmpty()) {  
    showAlert("Input Error", "Please fill in all fields.");  
    return;  
}
```

```
// Create a new Subsidy object
```

```
Subsidy newSubsidy = new Subsidy(subsidyName, description, eligibilityCriteria, region);
```

```
// Add the subsidy to the database
```

```
subsidyService.addSubsidy(newSubsidy);
```

```
// Notify the user of success and clear the input fields
```

```
clearFields();
```

```
showAlert("Success", "Subsidy added successfully!");
```

```
}
```

```
private void clearFields() {
```

```
    subsidyNameField.clear();
```

```
    descriptionField.clear();
```

```
    eligibilityCriteriaField.clear();
```

```
    regionField.clear();
```

```
}
```

```
private void showAlert(String title, String message) {
```

```
    Alert alert = new Alert(AlertType.INFORMATION);
```

```
    alert.setTitle(title);
```

```
    alert.setHeaderText(null);
```

```
    alert.setContentText(message);
```

```
    alert.showAndWait();
```

```
}
```

```
}
```

AuthService.java:

```
package com.agriculture.app;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
```

```
public class AuthService {
```

```
    private String currentFarmer;
```

```

private Connection connect() {
    String url = "jdbc:sqlite:farmers.db";
    Connection conn = null;
    try {
        conn = DriverManager.getConnection(url);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    return conn;
}

```

```

public boolean login(String username, String password) {
    String sql = "SELECT * FROM users WHERE username = ? AND password = ?";

    try (Connection conn = this.connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, username);
        pstmt.setString(2, password);
        ResultSet rs = pstmt.executeQuery();
        if(rs.next()){
            this.currentFarmer = username;
            return true;
        } // Return true if user exists
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    return false;
}

```

```

public boolean register(String username, String password, String role) {
    String sql = "INSERT INTO users(username, password, role) VALUES(?, ?, ?)";

    try (Connection conn = this.connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, username);
        pstmt.setString(2, password);
        pstmt.setString(3, role);
        pstmt.executeUpdate();
        System.out.println("User registered successfully!");
        return true; // Return true after successful registration
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return false; // Return false on error
    }
}

```

```
    }  
}  
  
}
```

Farmer.java:

```
package com.agriculture.app;
```

```
public class Farmer {  
    private String name;  
    private String email;  
    private long aadhar;  
    private double landSize;  
    private String cropType;  
  
    public Farmer(String name, String email, long aadhar, double landSize, String cropType) {  
        this.name = name;  
        this.email = email;  
        this.aadhar = aadhar;  
        this.landSize = landSize;  
        this.cropType = cropType;  
    }  
  
    // Getters and Setters  
    public String getName() {  
        return name;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
  
    public long getAadhar() {  
        return aadhar;  
    }  
  
    public double getLandSize() {  
        return landSize;  
    }  
  
    public String getCropType() {  
        return cropType;  
    }  
  
    // Setters  
    public void setName(String name) {
```

```

        this.name = name;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void setAadhar(long aadhar) {
        this.aadhar = aadhar;
    }

    public void setLandSize(double landSize) {
        this.landSize = landSize;
    }

    public void setCropType(String cropType) {
        this.cropType = cropType;
    }
}

```

FarmerDashboardController.java:

```

package com.agriculture.app;

import java.io.IOException;
import java.util.List;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.fxml.FXML;
import javafx.scene.control.TextArea;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextInputDialog;
import javafx.stage.Stage;

public class FarmerDashboardController {
    @FXML
    private Button logoutButton;

    private Farmer currentFarmer; // Define a Farmer instance

    // Method to set the current farmer, e.g., from the login process
    public void setCurrentFarmer(Farmer farmer) {
        this.currentFarmer = farmer;
    }
}

```

```

}

//private MarketPriceService marketPriceService;

// Setter method to inject the dependency
public void setMarketPriceService(MarketPriceService service) {
    this.marketPriceService = service;
}

private SchemeService schemeService = new SchemeService();

AuthService authService = new AuthService();

@FXML
void handleViewSchemes() {
    // Retrieve all schemes
    List<Scheme> schemes = schemeService.getAllSchemes(); // Assuming you have a method for this

    if (schemes.isEmpty()) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Schemes Information");
        alert.setHeaderText("No Available Schemes");
        alert.setContentText("There are no available schemes at this time.");
        alert.showAndWait();
    } else {
        // Build a string with the details of all available schemes
        StringBuilder schemeDetails = new StringBuilder("Available Schemes:\n");
        for (Scheme scheme : schemes) {
            schemeDetails.append("Scheme Name: ").append(scheme.getSchemeName()).append("\n");
            schemeDetails.append("Description: ").append(scheme.getDescription()).append("\n");
            schemeDetails.append("Eligibility: ").append(scheme.getEligibilityCriteria()).append("\n");
            schemeDetails.append("Region: ").append(scheme.getRegion()).append("\n\n");
        }

        // Show the information in an alert dialog
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Schemes Information");
        alert.setHeaderText("Available Schemes");
        alert.setContentText(schemeDetails.toString());
        alert.showAndWait();
    }
}

private SubsidyService subsidyService = new SubsidyService();

@FXML
void handleViewSubsidies() {
    // Retrieve all subsidies

```

```
List<Subsidy> subsidies = subsidyService.getAllSubsidies();
```

```
if (subsidies.isEmpty()) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Subsidies Information");
    alert.setHeaderText("No Available Subsidies");
    alert.setContentText("There are no available subsidies at this time.");
    alert.showAndWait();
} else {
    // Build a string with the details of all available subsidies
    StringBuilder subsidyDetails = new StringBuilder("Available Subsidies:\n");
    for (Subsidy subsidy : subsidies) {
        subsidyDetails.append("Subsidy Name: ").append(subsidy.getSubsidyName()).append("\n");
        subsidyDetails.append("Description: ").append(subsidy.getDescription()).append("\n");
        subsidyDetails.append("Eligibility: ").append(subsidy.getEligibilityCriteria()).append("\n");
        subsidyDetails.append("Region: ").append(subsidy.getRegion()).append("\n\n");
    }

    // Show the information in an alert dialog
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Subsidies Information");
    alert.setHeaderText("Available Subsidies");
    alert.setContentText(subsidyDetails.toString());
    alert.showAndWait();
}
}
```

```
private TextArea marketPricesArea; // Add a TextArea to display market prices
```

```
private MarketPriceService marketPriceService = new MarketPriceService(); // Declare the service variable
```

```
@FXML
```

```
private void loadMarketPrices() {
    Double wheatPrice = marketPriceService.getMarketPrice("Wheat");
    Double ricePrice = marketPriceService.getMarketPrice("Rice");
    Double milletPrice = marketPriceService.getMarketPrice("Millets");

    StringBuilder prices = new StringBuilder();
    if (wheatPrice != null) {
        prices.append("Wheat: ").append(wheatPrice).append("\n");
    } else {
        prices.append("Wheat price not available.\n");
    }
    if (ricePrice != null) {
        prices.append("Rice: ").append(ricePrice).append("\n");
    } else {
        prices.append("Rice price not available.\n");
    }
}
```

```

    }
    if (milletPrice != null) {
        prices.append("Millets: ").append(milletPrice).append("\n");
    } else {
        prices.append("Millet price not available.\n");
    }

    marketPricesArea.setText(prices.toString());
}

private void showAlert(String title, String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

@FXML
private void handleViewMarketPrices() {
    loadMarketPrices();
}

@FXML
private void handleMarketPrices() {
    Double wheatPrice = marketPriceService.getMarketPrice("Wheat");
    Double ricePrice = marketPriceService.getMarketPrice("Rice");
    Double milletPrice = marketPriceService.getMarketPrice("Millets");

    String pricesMessage = String.format("Wheat: ₹%.2f\nRice: ₹%.2f\nMillets: ₹%.2f",
        wheatPrice, ricePrice, milletPrice);

    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Market Prices");
    alert.setHeaderText("Current Market Prices");
    alert.setContentText(pricesMessage);
    alert.showAndWait();
}

@FXML
public void handleLogout() {
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/main.fxml"));
        Parent root = loader.load();
    }
}

```

```

        // Get the current stage and set the new scene
        Stage stage = (Stage) logoutButton.getScene().getWindow();
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

FarmerRegistrationController.java:

```

package com.agriculture.app;

import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.control.PasswordField;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;

public class FarmerRegistrationController {

    @FXML
    private TextField nameField;
    @FXML
    private TextField emailField;
    @FXML
    private TextField aadharField;
    @FXML
    private TextField landSizeField;
    @FXML
    private TextField cropTypeField;
    @FXML
    private TextField usernameField;
    @FXML
    private PasswordField passwordField;

    private FarmerService farmerService = new FarmerService();
    private AuthService authService = new AuthService();

    @FXML
    public void handleRegister() {
        String name = nameField.getText();
        String email = emailField.getText();
        String aadhar = aadharField.getText();
    }
}

```



```

String landSize = landSizeField.getText();
String cropType = cropTypeField.getText();
String username = usernameField.getText();
String password = passwordField.getText();

// Register the farmer
boolean isRegistered = authService.register(username, password, "farmer");
if (isRegistered) {
    Farmer farmer = new Farmer(name, email, Long.parseLong(aadhar), Integer.parseInt(landSize),
cropType);
    farmerService.addFarmer(farmer);

    Alert alert = new Alert(AlertType.INFORMATION);
    alert.setTitle("Registration Successful");
    alert.setHeaderText(null);
    alert.setContentText("Farmer registered successfully!");
    alert.showAndWait();
} else {
    Alert alert = new Alert(AlertType.ERROR);
    alert.setTitle("Registration Failed");
    alert.setHeaderText(null);
    alert.setContentText("Username already exists. Please choose a different one.");
    alert.showAndWait();
}
}
}

```

FarmerService.java:

```

package com.agriculture.app;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.sql.Statement;

public class FarmerService {

    private Connection connect() {
        String url = "jdbc:sqlite:farmers.db";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);

```

```

    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    return conn;
}

public void addFarmer(Farmer farmer) {
    String sql = "INSERT INTO farmers(name, email, aadhar, landSize, cropType) VALUES(?, ?, ?, ?, ?)";

    try (Connection conn = this.connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, farmer.getName());
        pstmt.setString(2, farmer.getEmail());
        pstmt.setLong(3, farmer.getAadhar());
        pstmt.setDouble(4, farmer.getLandSize());
        pstmt.setString(5, farmer.getCropType());

        pstmt.executeUpdate();
        System.out.println("Farmer added successfully!");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

public List<Farmer> getAllFarmers() {
    List<Farmer> farmers = new ArrayList<>();
    Connection connection = null;
    try {
        connection = DriverManager.getConnection("jdbc:sqlite:farmers.db");
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM farmers");

        while (rs.next()) {
            //int id = rs.getInt("id"); // Get the farmer's ID
            String name = rs.getString("name");
            String email = rs.getString("email");
            long aadhar = rs.getLong("aadhar");
            double landSize = rs.getDouble("landSize"); // Use the correct casing
            String cropType = rs.getString("cropType"); // Use the correct casing

            Farmer farmer = new Farmer(name, email, aadhar, landSize, cropType);
            //farmer.setId(id); // Ensure the setId method exists in Farmer class
            farmers.add(farmer);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {

```

```

        // Close connection
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return farmers;
}
}

```

GovernmentDashboardController.java:

```

package com.agriculture.app;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
//import javafx.fxml.FXML;
//import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.control.TextInputDialog;
import com.agriculture.app.Scheme;
import com.agriculture.app.SchemeService;
//import javafx.fxml.FXML;
//import javafx.scene.control.Alert;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.List;
import java.util.Optional;

public class GovernmentDashboardController {

    private FarmerService farmerService;

```

```

@FXML
private Button addSchemeButton;

@FXML
private Button logoutButton;

public GovernmentDashboardController(){
    this.farmerService = new FarmerService();
}

@FXML private TextField schemeNameField;
@FXML private TextField descriptionField;
@FXML private TextField eligibilityCriteriaField;
@FXML private TextField regionField;
@FXML private TextField cropTypeField;

private SchemeService schemeService = new SchemeService();

@FXML
private void handleAddScheme() {
    try {
        // Load the add_scheme.fxml file
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/add_scheme.fxml"));
        Parent addSchemeRoot = loader.load();

        // Create a new stage for the add scheme form
        Stage addSchemeStage = new Stage();
        addSchemeStage.setTitle("Add Scheme");
        //addSchemeStage.initModality(Modality.APPLICATION_MODAL); // Optional: makes the new
window modal
        addSchemeStage.setScene(new Scene(addSchemeRoot));
        addSchemeStage.showAndWait(); // Show the new stage and wait for it to close
    } catch (IOException e) {
        e.printStackTrace(); // Handle the IOException (e.g., log it or show an error message)
    }
}

@FXML
private void openSchemeForm() {
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/add_scheme.fxml"));
        Parent root = loader.load();

        // Obtain the controller to ensure it has the FXML fields injected
        AddSchemeController controller = loader.getController();

        Stage stage = new Stage();

```

```

        stage.setTitle("Add New Scheme");
        stage.setScene(new Scene(root));
        stage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

@FXML

```

public void openSubsidyForm() {
    try {
        // Load the FXML file for the subsidy form
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/add_subsidy.fxml"));
        Parent subsidyForm = loader.load();

        // Create a new stage for the subsidy form
        Stage stage = new Stage();
        stage.setTitle("Add Subsidy");
        stage.setScene(new Scene(subsidyForm));
        stage.show();

    } catch (IOException e) {
        e.printStackTrace();
        showAlert("Error", "Could not open subsidy form.");
    }
}

```

```

private void showAlert(String title, String message) {
    Alert alert = new Alert(AlertType.ERROR);
    alert.setTitle(title);
    alert.setHeaderText(null); // No header text
    alert.setContentText(message);
    alert.showAndWait();
}

```

@FXML

```

private void handleViewFarmers() {
    // Logic to fetch and display a list of farmers
    System.out.println("Fetching farmers list...");
    List<Farmer> farmers = farmerService.getAllFarmers(); // Fetch the list of farmers
    StringBuilder farmerList = new StringBuilder();

    // Check if the list is empty
    if (farmers.isEmpty()) {
        farmerList.append("No registered farmers found.");
    } else {
        for (Farmer farmer : farmers) {
            farmerList.append("Name: ").append(farmer.getName()).append("\n")

```

```

        .append("Email: ").append(farmer.getEmail()).append("\n")
        .append("Aadhar: ").append(farmer.getAadhar()).append("\n")
        .append("Land Size: ").append(farmer.getLandSize()).append("\n")
        .append("Crop Type: ").append(farmer.getCropType()).append("\n\n");
    }
}

// Display the farmers list in an alert
Alert alert = new Alert(AlertType.INFORMATION);
alert.setTitle("Farmers List");
alert.setHeaderText("Registered Farmers");
alert.setContentText(farmerList.toString());
alert.showAndWait();
}

@FXML
private void handleAddSubsidy() {
    try {
        // Load the add_subsidy.fxml file
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/add_subsidy.fxml"));
        Parent addSubsidyRoot = loader.load();

        // Create a new stage for the add subsidy form
        Stage addSubsidyStage = new Stage();
        addSubsidyStage.setTitle("Add Subsidy");
        // Optional: Uncomment the next line to make the new window modal
        // addSubsidyStage.initModality(Modality.APPLICATION_MODAL);
        addSubsidyStage.setScene(new Scene(addSubsidyRoot));
        addSubsidyStage.showAndWait(); // Show the new stage and wait for it to close
    } catch (IOException e) {
        e.printStackTrace(); // Handle the IOException (e.g., log it or show an error message)
    }
}

@FXML
private void handleUpdateMarketPrices() {
    // Prompt user for input
    TextInputDialog dialog = new TextInputDialog();
    dialog.setTitle("Update Market Prices");
    dialog.setHeaderText("Enter the updated market prices");
    dialog.setContentText("Format: wheatPrice, ricePrice, milletPrice");

    Optional<String> result = dialog.showAndWait();
    result.ifPresent(prices -> {
        String[] priceArray = prices.split(",");
        if (priceArray.length == 3) {
            // Update prices in the database
            updateMarketPrices(priceArray[0].trim(), priceArray[1].trim(), priceArray[2].trim());
        }
    });
}

```

```

    } else {
        showAlert("Input Error", "Please enter prices in the format: wheatPrice,ricePrice,milletPrice");
    }
});
}

```

```

private void updateMarketPrices(String wheatPrice, String ricePrice, String milletsPrice) {
    String sql = "UPDATE market_prices SET price = ? WHERE crop_type = ?";

    try (Connection conn = DriverManager.getConnection("jdbc:sqlite:farmers.db")) {
        // Update wheat price
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, wheatPrice);
            pstmt.setString(2, "wheat");
            pstmt.executeUpdate();
        }
        // Update rice price
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, ricePrice);
            pstmt.setString(2, "rice");
            pstmt.executeUpdate();
        }
        // Update millets price
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, milletsPrice);
            pstmt.setString(2, "millets");
            pstmt.executeUpdate();
        }
        showAlert("Success", "Market prices updated successfully!");
    } catch (SQLException e) {
        showAlert("Error", "Failed to update market prices: " + e.getMessage());
    }
}

```

@FXML

```

public void handleLogout() {
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/main.fxml"));
        Parent root = loader.load();

        // Get the current stage and set the new scene
        Stage stage = (Stage) logoutButton.getScene().getWindow();
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }
}

```

```

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Main.java:

```

package com.agriculture.app;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            // Initialize the SQLite database
            SQLiteSetup dbSetup = new SQLiteSetup();
            dbSetup.createNewDatabase(); // Call this method to create tables

            // Load the Main.fxml file from the resources folder
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/main.fxml")); // Ensure
this path is correct
            Parent root = loader.load();

            primaryStage.setTitle("Agriculture Management System");
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        MarketPriceService.initializeMarketPrices();
        launch(args);
    }
}

```

MainController.java:

```

package com.agriculture.app;

```



```

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.control.Button;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;

public class MainController {

    @FXML
    private TextField usernameField;
    @FXML
    private TextField passwordField;
    @FXML
    private Button loginButton;
    @FXML
    private Button registerButton;

    private AuthService authService = new AuthService();

    @FXML
    public void handleLogin() {
        String username = usernameField.getText();
        String password = passwordField.getText();

        // Check for government employee credentials
        if ("gov".equals(username) && "1234".equals(password)) {
            // Logic to redirect to Government Employee Dashboard
            System.out.println("Government Employee login successful!");
            MarketPriceService.initializeMarketPrices();
            navigateTo("fxml/government_dashboard.fxml"); // Update with the actual path
        } else if (authService.login(username, password)) {
            // Logic to redirect to Farmer Dashboard
            System.out.println("Farmer login successful!");
            navigateTo("fxml/farmer_dashboard.fxml"); // Update with the actual path
        } else {
            System.out.println("Login failed!");
        }
    }
}

```

```

        showAlert("Login Failed", "Invalid username or password.");
    }
}

@FXML
public void handleRegister() {
    try {
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/fxml/farmer_registration.fxml"));
        Parent root = loader.load();
        Stage stage = new Stage();
        stage.setTitle("Farmer Registration");
        stage.setScene(new Scene(root));
        stage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private void showAlert(String title, String message) {
    Alert alert = new Alert(AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

private void navigateTo(String fxmlPath) {
    try {
        // Load the appropriate FXML file based on user role
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/" + fxmlPath));
        Parent root = loader.load();

        // Set the new scene
        Stage stage = (Stage) loginButton.getScene().getWindow(); // Get current stage
        stage.setScene(new Scene(root));
        stage.setTitle("Dashboard"); // You can set the title accordingly
        stage.show();
    } catch (Exception e) {
        e.printStackTrace();
        showAlert("Navigation Error", "Could not load the requested page.");
    }
}
}

```

MarketPriceService.java:

```
package com.agriculture.app;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class MarketPriceService {

    public Connection connect() {
        String url = "jdbc:sqlite:farmers.db";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return conn;
    }

    public static void initializeMarketPrices() {
        // Check if prices already exist
        String checkSql = "SELECT COUNT(*) FROM market_prices";
        try (Connection conn = new MarketPriceService().connect();
            PreparedStatement pstmt = conn.prepareStatement(checkSql)) {

            ResultSet rs = pstmt.executeQuery();
            if (rs.next() && rs.getInt(1) == 0) {
                // If no prices are found, insert initial market prices
                insertInitialMarketPrices();
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }

    public Double getMarketPrice(String crop) {
        String sql = "SELECT price FROM market_prices WHERE cropName = ?";
        try (Connection conn = this.connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, crop);
            ResultSet rs = pstmt.executeQuery();
        }
    }
}
```

```

        if (rs.next()) {
            return rs.getDouble("price");
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    return null; // Return null if no price found
}

public static void insertInitialMarketPrices() {
    String sql = "INSERT INTO market_prices (cropName, crop_type, price) VALUES (?, ?, ?)";
    try (Connection conn = new MarketPriceService().connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, "Wheat");
        pstmt.setString(2, "Cereal");
        pstmt.setDouble(3, 200.0);
        pstmt.executeUpdate();

        pstmt.setString(1, "Rice");
        pstmt.setString(2, "Cereal");
        pstmt.setDouble(3, 150.0);
        pstmt.executeUpdate();

        pstmt.setString(1, "Millets");
        pstmt.setString(2, "Cereal");
        pstmt.setDouble(3, 100.0);
        pstmt.executeUpdate();

        System.out.println("Initial market prices inserted successfully.");
    } catch (SQLException e) {
        System.out.println("SQL Exception: " + e.getMessage());
    }
}

```

```

public static void updateMarketPrices(double wheatPrice, double ricePrice, double milletPrice) {
    String sql = "UPDATE market_prices SET price = ? WHERE cropName = ?";
    try (Connection conn = new MarketPriceService().connect();
        PreparedStatement pstmtWheat = conn.prepareStatement(sql);
        PreparedStatement pstmtRice = conn.prepareStatement(sql);
        PreparedStatement pstmtMillet = conn.prepareStatement(sql)) {

        pstmtWheat.setDouble(1, wheatPrice);
        pstmtWheat.setString(2, "Wheat");
        pstmtWheat.executeUpdate();

        pstmtRice.setDouble(1, ricePrice);

```

```

        pstmtRice.setString(2, "Rice");
        pstmtRice.executeUpdate();

        pstmtMillet.setDouble(1, milletPrice);
        pstmtMillet.setString(2, "Millets");
        pstmtMillet.executeUpdate();

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
}

```

MarketPricesController.java:

```

package com.agriculture.app;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.TextField;

public class MarketPricesController {

    @FXML
    private TextField wheatPriceField;
    @FXML
    private TextField ricePriceField;
    @FXML
    private TextField milletPriceField;

    private final MarketPriceService marketPriceService = new MarketPriceService();

    @FXML
    private void handleUpdateMarketPrices() {
        try {
            double wheatPrice = Double.parseDouble(wheatPriceField.getText().trim());
            double ricePrice = Double.parseDouble(ricePriceField.getText().trim());
            double milletPrice = Double.parseDouble(milletPriceField.getText().trim());
            double milletPrice = 200.0;
            double ricePrice = 1.0;
            double wheatPrice = 1.0;
            // Call updateMarketPrices with all three prices at once
            marketPriceService.updateMarketPrices(wheatPrice, ricePrice, milletPrice);

            showAlert("Success", "Market prices updated successfully!");

        } catch (NumberFormatException e) {

```

```

        showAlert("Input Error", "Please enter valid numbers for all prices.");
    }
}

```

```

private void showAlert(String title, String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}
}

```

SQLiteSetup.java:

```

package com.agriculture.app;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class SQLiteSetup {

    public Connection connect() {
        String url = "jdbc:sqlite:farmers.db";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return conn;
    }

    public void createNewDatabase() {
        String sql = "CREATE TABLE IF NOT EXISTS users ("
            + " id INTEGER PRIMARY KEY AUTOINCREMENT,"
            + " username TEXT NOT NULL,"
            + " password TEXT NOT NULL,"
            + " role TEXT NOT NULL);";

        String farmersSql = "CREATE TABLE IF NOT EXISTS farmers ("
            + " id INTEGER PRIMARY KEY AUTOINCREMENT,"
            + " name TEXT NOT NULL,"
            + " email TEXT NOT NULL,"

```

```

+ " aadhar INTEGER NOT NULL,"
+ " landSize INTEGER NOT NULL,"
+ " cropType TEXT NOT NULL);";

```

```

String schemesSql = "CREATE TABLE IF NOT EXISTS schemes ("
+ " id INTEGER PRIMARY KEY AUTOINCREMENT,"
+ " schemeName TEXT NOT NULL,"
+ " description TEXT NOT NULL,"
+ " eligibilityCriteria TEXT NOT NULL,"
+ " region TEXT NOT NULL,"
+ " cropType TEXT NOT NULL);";

```

// New Subsidies Table

```

String subsidiesSql = "CREATE TABLE IF NOT EXISTS subsidies ("
+ " id INTEGER PRIMARY KEY AUTOINCREMENT,"
+ " subsidyName TEXT NOT NULL,"
+ " description TEXT NOT NULL,"
+ " eligibilityCriteria TEXT NOT NULL,"
+ " region TEXT NOT NULL);";

```

```

String marketPricesSql = "CREATE TABLE IF NOT EXISTS market_prices ("
+ " id INTEGER PRIMARY KEY AUTOINCREMENT,"
+ " cropName TEXT NOT NULL,"
+ "crop_type TEXT NOT NULL,"
+ " price REAL NOT NULL);";

```

```

try (Connection conn = this.connect();
    Statement stmt = conn.createStatement()) {

    stmt.execute(sql);
    stmt.execute(farmersSql);
    stmt.execute(schemesSql);
    stmt.execute(subsidiesSql);
    stmt.execute(marketPricesSql);
    System.out.println("Tables created successfully!");

} catch (Exception e) {
    System.out.println(e.getMessage());
}
}
}

```

Scheme.java:

```
package com.agriculture.app;
```

```

public class Scheme {
    private String schemeName;
    private String description;
    private String eligibilityCriteria;
    private String region;
    private String cropType;

    public Scheme(String schemeName, String description, String eligibilityCriteria, String region, String
cropType) {
        this.schemeName = schemeName;
        this.description = description;
        this.eligibilityCriteria = eligibilityCriteria;
        this.region = region;
        this.cropType = cropType;
    }

    // Getters and Setters
    public String getSchemeName() { return schemeName; }
    public String getDescription() { return description; }
    public String getEligibilityCriteria() { return eligibilityCriteria; }
    public String getRegion() { return region; }
    public String getCropType() { return cropType; }
}

```

SchemeService.java:

```

package com.agriculture.app;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class SchemeService {

    private Connection connect() {
        String url = "jdbc:sqlite:farmers.db";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (Exception e) {

```



```

        System.out.println(e.getMessage());
    }
    return conn;
}

public void addScheme(Scheme scheme) {
    String sql = "INSERT INTO schemes(schemeName, description, eligibilityCriteria, region, cropType)
VALUES(?, ?, ?, ?, ?)";

    try (Connection conn = this.connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, scheme.getSchemeName());
        pstmt.setString(2, scheme.getDescription());
        pstmt.setString(3, scheme.getEligibilityCriteria());
        pstmt.setString(4, scheme.getRegion());
        pstmt.setString(5, scheme.getCropType());
        pstmt.executeUpdate();
        System.out.println("Scheme added successfully!");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

public List<Scheme> getSchemesByCropType(String cropType) {
    List<Scheme> schemes = new ArrayList<>();
    String sql = "SELECT * FROM schemes WHERE cropType = ?";

    try (Connection conn = new SQLiteSetup().connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, cropType);
        ResultSet rs = pstmt.executeQuery();

        while (rs.next()) {
            Scheme scheme = new Scheme(
                //rs.getInt("id"),
                rs.getString("schemeName"),
                rs.getString("description"),
                rs.getString("eligibilityCriteria"),
                rs.getString("region"),
                rs.getString("cropType")
            );
            schemes.add(scheme);
        }

    } catch (Exception e) {
        System.out.println("Error fetching schemes: " + e.getMessage());
    }
}

```

```

    }

    return schemes;
}

public List<Scheme> getAllSchemes() {
    List<Scheme> schemes = new ArrayList<>();
    String sql = "SELECT * FROM schemes";

    try (Connection conn = new SQLiteSetup().connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        ResultSet rs = pstmt.executeQuery();

        while (rs.next()) {
            Scheme scheme = new Scheme(
                rs.getString("schemeName"),
                rs.getString("description"),
                rs.getString("eligibilityCriteria"),
                rs.getString("region"),
                rs.getString("cropType") // You might want to include this too
            );
            schemes.add(scheme);
        }

    } catch (Exception e) {
        System.out.println("Error fetching schemes: " + e.getMessage());
    }

    return schemes;
}
}

```

Subsidy.java:

```

package com.agriculture.app;

public class Subsidy {
    private String subsidyName;
    private String description;
    private String eligibilityCriteria;
    private String region;

    public Subsidy(String subsidyName, String description, String eligibilityCriteria, String region) {

```

```

        this.subsidyName = subsidyName;
        this.description = description;
        this.eligibilityCriteria = eligibilityCriteria;
        this.region = region;
    }

    public String getSubsidyName() {
        return subsidyName;
    }

    public String getDescription() {
        return description;
    }

    public String getEligibilityCriteria() {
        return eligibilityCriteria;
    }

    public String getRegion() {
        return region;
    }
}

```

SubsidyService.java:

```

package com.agriculture.app;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class SubsidyService {

    private Connection connect() {
        String url = "jdbc:sqlite:farmers.db";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return conn;
    }
}

```

```

}

//private static final String url = "jdbc:sqlite:farmers.db";

public void addSubsidy(Subsidy subsidy) {
    String sql = "INSERT INTO subsidies(subsidyName, description, eligibilityCriteria, region)
VALUES(?, ?, ?, ?)";

    try (Connection conn = this.connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, subsidy.getSubsidyName());
        pstmt.setString(2, subsidy.getDescription());
        pstmt.setString(3, subsidy.getEligibilityCriteria());
        pstmt.setString(4, subsidy.getRegion());
        pstmt.executeUpdate();
        System.out.println("Subsidy added successfully!");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

public List<Subsidy> getAllSubsidies() {
    List<Subsidy> subsidies = new ArrayList<>();
    String sql = "SELECT * FROM subsidies";

    try (Connection conn = new SQLiteSetup().connect();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        ResultSet rs = pstmt.executeQuery();

        while (rs.next()) {
            Subsidy subsidy = new Subsidy(
                rs.getString("subsidyName"),
                rs.getString("description"),
                rs.getString("eligibilityCriteria"),
                rs.getString("region")
            );
            subsidies.add(subsidy);
        }

    } catch (Exception e) {
        System.out.println("Error fetching schemes: " + e.getMessage());
    }

    return subsidies;
}

```

```
}
```

FXML Files:

Add_farmer.fxml:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.VBox?>

<VBox xmlns:fx="http://javafx.com/fxml">
    <TextField fx:id="nameField" promptText="Name" />
    <TextField fx:id="emailField" promptText="Email" />
    <TextField fx:id="aadharField" promptText="Aadhar" />
    <TextField fx:id="landSizeField" promptText="Land Size" />
    <TextField fx:id="cropTypeField" promptText="Crop Type" />
    <Button text="Register" onAction="#addFarmer" />
</VBox>
```

Add_scheme.fxml:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.VBox?>

<VBox xmlns:fx="http://javafx.com/fxml" fx:controller="com.agriculture.app.AddSchemeController">
    <Label text="Add New Scheme" style="-fx-font-size: 16px; -fx-font-weight: bold;" />
    <TextField fx:id="schemeNameField" promptText="Scheme Name" />
    <TextField fx:id="descriptionField" promptText="Description" />
    <TextField fx:id="eligibilityCriteriaField" promptText="Eligibility Criteria" />
    <TextField fx:id="regionField" promptText="Region" />
    <TextField fx:id="cropTypeField" promptText="Crop Type" />
    <Button text="Submit" onAction="#handleAddScheme" />
</VBox>
```

Add_subsidy.fxml:

```

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane xmlns:fx="http://javafx.com/fxml"
fx:controller="com.agriculture.app.AddSubsidyController">
    <children>
        <Label text="Subsidy Name:" layoutX="14.0" layoutY="14.0" />
        <TextField fx:id="subsidyNameField" layoutX="150.0" layoutY="10.0" />

        <Label text="Description:" layoutX="14.0" layoutY="50.0" />
        <TextField fx:id="descriptionField" layoutX="150.0" layoutY="46.0" />

        <Label text="Eligibility Criteria:" layoutX="14.0" layoutY="86.0" />
        <TextField fx:id="eligibilityCriteriaField" layoutX="150.0" layoutY="82.0" />

        <Label text="Region:" layoutX="14.0" layoutY="122.0" />
        <TextField fx:id="regionField" layoutX="150.0" layoutY="118.0" />

        <Button fx:id="addSubsidyButton" text="Add Subsidy" layoutX="150.0" layoutY="160.0"
onAction="#handleAddSubsidy"/>
    </children>
</AnchorPane>

```

Farmer_dashboard.fxml:

```

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.VBox?>

<VBox xmlns:fx="http://javafx.com/fxml" fx:controller="com.agriculture.app.FarmerDashboardController"
spacing="15" alignment="CENTER" style="-fx-padding: 20;">
    <Label text="Farmer Dashboard" style="-fx-font-size: 24px; -fx-font-weight: bold;" />

    <Button fx:id="viewSchemesButton" text="View Schemes" onAction="#handleViewSchemes"/>

    <Button fx:id="viewSubsidiesButton" text="View Subsidies" onAction="#handleViewSubsidies"/>

    <Button text="Market Prices" onAction="#handleMarketPrices" prefWidth="200"/>
    <Button fx:id="logoutButton" onAction="#handleLogout" text="Logout" />

</VBox>

```

Farmer_registration.fxml:

```
<VBox xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.agriculture.app.FarmerRegistrationController" alignment="CENTER" spacing="10">
    <Label text="Farmer Registration" style="-fx-font-size: 18px;" />
    <TextField fx:id="nameField" promptText="Name" />
    <TextField fx:id="emailField" promptText="Email" />
    <TextField fx:id="aadharField" promptText="Aadhar Number" />
    <TextField fx:id="landSizeField" promptText="Land Size (in acres)" />
    <TextField fx:id="cropTypeField" promptText="Crop Type" />
    <TextField fx:id="usernameField" promptText="Username" />
    <PasswordField fx:id="passwordField" promptText="Password" />
    <Button text="Register" onAction="#handleRegister" />
</VBox>
```

GovernmentDashboard.fxml:

```
<?xml version="1.0" encoding="UTF-8"?>

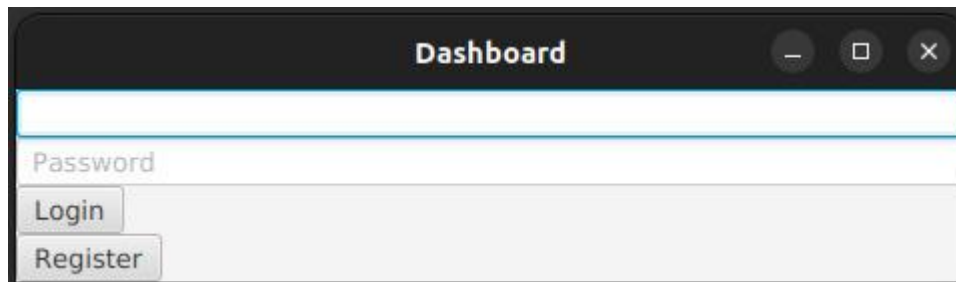
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.VBox?>

<VBox xmlns:fx="http://javafx.com/fxml"
fx:controller="com.agriculture.app.GovernmentDashboardController" spacing="15" alignment="CENTER"
style="-fx-padding: 20;">
    <Label text="Government Dashboard" style="-fx-font-size: 24px; -fx-font-weight: bold;" />

    <Button fx:id="addSchemeButton" onAction="#handleAddScheme" text="Add Scheme" />
    <Button text="View Farmers" onAction="#handleViewFarmers" prefWidth="200" />
    <Button fx:id="addSubsidyButton" text="Add Subsidy" onAction="#handleAddSubsidy" />
    <Button fx:id="updateMarketPricesButton" onAction="#handleUpdateMarketPrices" text="Update
Market Prices" prefWidth="200" />

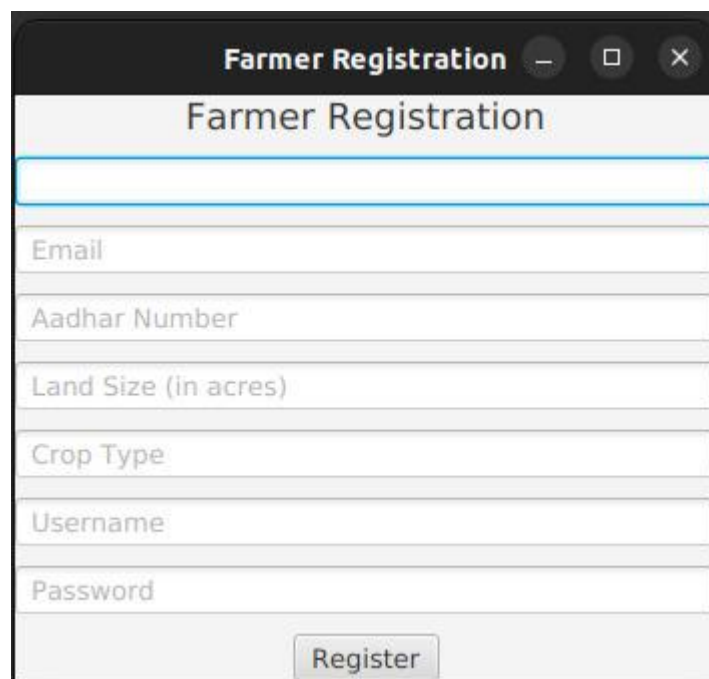
    <Button fx:id="logoutButton" onAction="#handleLogout" text="Logout" />
</VBox>
```

RESULTS



A screenshot of a web application window titled "Dashboard". The window has a dark header bar with the title and standard window control buttons (minimize, maximize, close). Below the header, there is a light gray input field. Underneath the input field, there are two buttons: "Login" and "Register". The "Register" button is highlighted with a gray border.

Fig.1. Initial login page



A screenshot of a web application window titled "Farmer Registration". The window has a dark header bar with the title and standard window control buttons. Below the header, there is a light gray input field. Underneath the input field, there are several text input fields: "Email", "Aadhar Number", "Land Size (in acres)", "Crop Type", "Username", and "Password". At the bottom right of the form, there is a "Register" button.

Fig.2. Farmer Registration Page (when clicked on register)

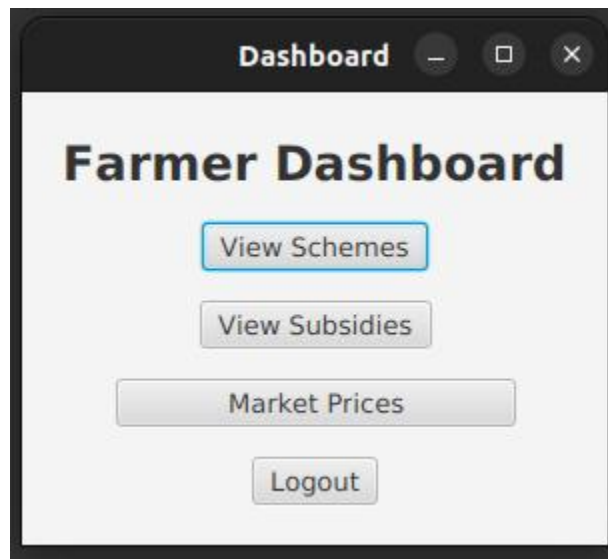


Fig.3. Farmer Dashboard

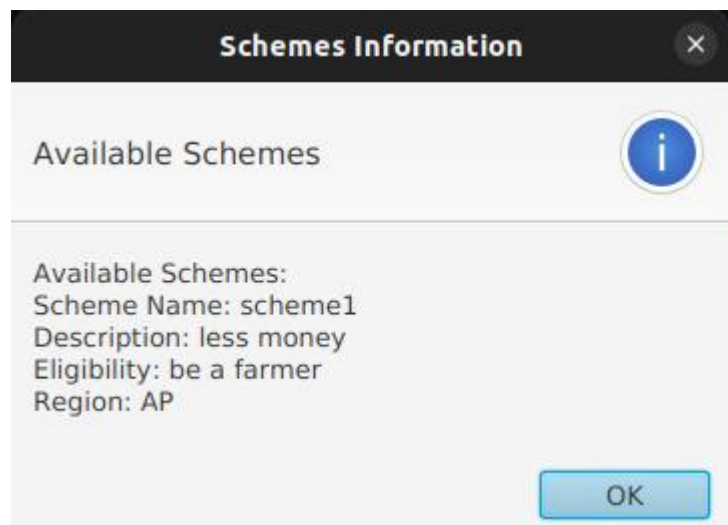


Fig.4. Viewing Schemes

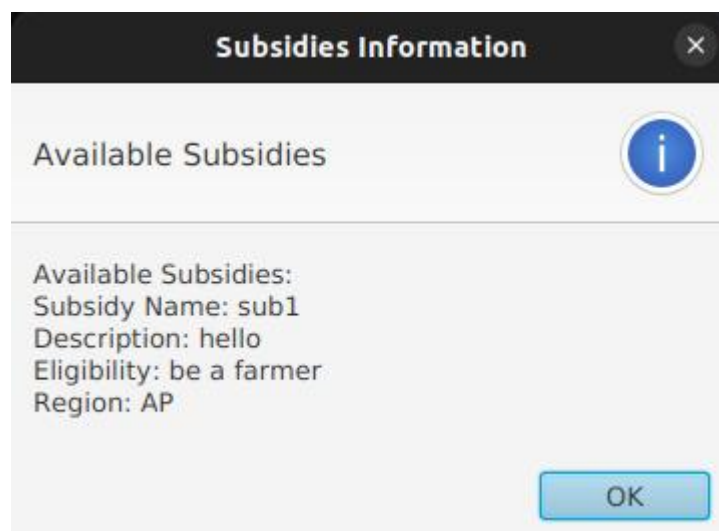


Fig.5. Viewing Subsidies

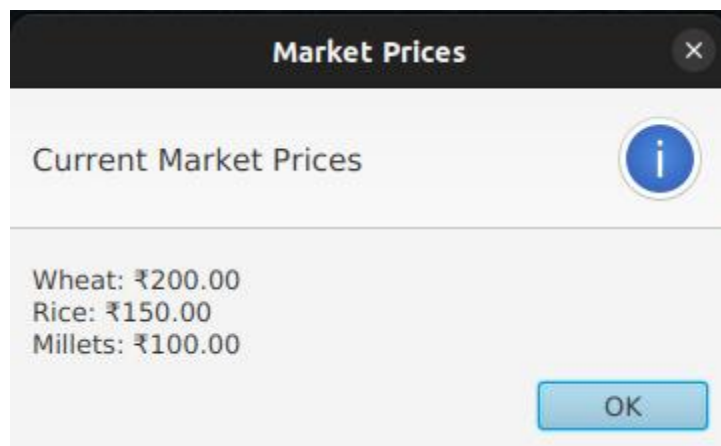


Fig.6. Viewing Market Prices



Fig.7. Government Dashboard

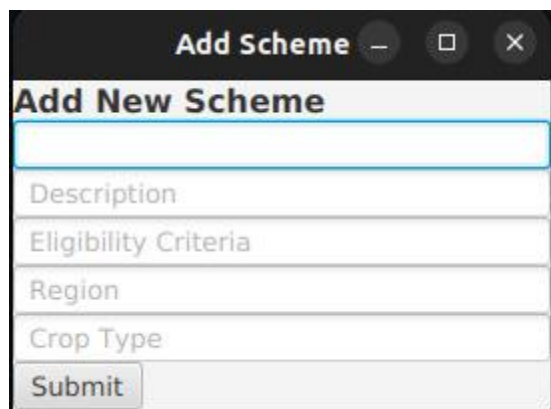


Fig.8. Adding a Scheme

Farmers List [Close]

Registered Farmers [Info]

Name: vinay
 Email: 1
 Aadhar: 1
 Land Size: 1.0
 Crop Type: 1

[OK]

Fig.9. Farmers List

Add Subsidy [Minimize] [Maximize] [Close]

Subsidy Name:

Description:

Eligibility Criteria:

Region:

[Add Subsidy]

Fig.10. Adding Subsidy

Update Market Prices [Close]

Enter the updated market prices [Help]

Format: wheatPrice,ricePrice,milletPrice

[Cancel] [OK]

Fig.11. Updating Market Price(shown only for a few crops)

CONCLUSION

The Agriculture Management System (AMS) successfully provides a centralized and accessible platform for farmers and government employees, bridging information gaps and

streamlining resource management. By offering differentiated access based on user roles, the system ensures that farmers can view relevant crop-specific schemes, subsidies, and market prices, while government employees have the tools to manage resources, schemes, and farmer registrations. Through thoughtful implementation of JavaFX for a user-friendly interface, integrated SQLite database for reliable data management, and structured exception handling for system robustness, AMS effectively meets its objectives.

This project demonstrates how digital tools can empower stakeholders in agriculture by facilitating transparent, organized data management and targeted support. With a flexible design that can accommodate future features and improvements, AMS is positioned as a valuable resource for enhancing decision-making and supporting sustainable agricultural practices. Moving forward, this project could expand to include additional functionalities, such as advanced data analytics, mobile integration, or further real-time market data, enhancing its utility and impact within the agricultural sector.

CONTRIBUTIONS

- **Vinay**



Vinay led the development of the **backend architecture** and **database integration** for the Agriculture Management System, setting up SQLite to handle data storage, retrieval, and management of user information, schemes, subsidies, and market prices. He implemented **user authentication and role-based access control**, ensuring secure and reliable login functionalities for both farmers and government employees. Vinay was also responsible for **exception handling** throughout the system, ensuring data integrity and minimizing application crashes. Additionally, he integrated the JavaFX framework with the SQLite database to allow seamless data flow between the interface and backend.

- **Leah**



Leah managed the **JavaFX-based user interface design and layout** of the system, creating a clean, intuitive experience for users. She used FXML to define view structures for each screen and worked on the design of separate dashboards for farmers and government employees. Leah also implemented **scheme management** in the user interface, which allows government employees to add new schemes and associate them with specific crop types. This design ensured that the platform was user-friendly and accessible to both farmers and government officials.

- **Amaan**



Amaan focused on the **subsidy management module**, allowing government employees to add, update, and manage subsidy details. He ensured that farmers could access relevant subsidy information based on their crop type. Amaan also collaborated with Vinay to connect the subsidy management UI with the SQLite database, allowing for real-time updates and smooth retrieval of subsidy data. Additionally, he helped in testing and debugging the module, ensuring that it worked accurately across user roles.

- **Spandan**



Spandan worked on the **market price updates feature**, integrating it with the farmer dashboard to allow real-time viewing of crop prices. He designed and implemented methods to pull data and update market prices, giving farmers crucial insights into crop values. Spandan also collaborated on the **farmer registration feature**, ensuring new registrations were processed and stored correctly in the database. His efforts focused on keeping the information accessible and accurate for farmer users.

- **Harsha**



Harsha took charge of **user registration and role differentiation**, implementing the system's registration module, which collects farmer information such as Aadhar number, crop type, and land size. Harsha designed the **login flow** for both farmers and government employees, ensuring that each user type only accessed relevant sections. He also helped manage **scheme visibility**, setting up criteria so that farmers only view schemes and subsidies applicable to their crop type.

REFERENCES

Object-Oriented Programming (OOP) Concepts:

- Bloch, J. (2008). *Effective Java*. Addison-Wesley.
- Oracle. (2023). *Lesson: Object-Oriented Programming Concepts*. [Java Tutorials](#).

JavaFX and FXML:

- Oracle (2023). *JavaFX Overview*. Available from [OpenJFX Documentation](#).
- Dea, B. (2014). *JavaFX and FXML: A Tutorial for Building a JavaFX Application with FXML*. Oracle Corporation

Exception Handling in Java:

- Gosling, J., Joy, B., Steele, G., & Bracha, G. (2014). *The Java Language Specification*. Oracle.
- Oracle (2023). *Exceptions*. Available from [Oracle's Java Tutorials](#).

Database Integration and SQLite:

- Owens, M., & Allen, G. (2010). *The Definitive Guide to SQLite*. Apress.
- [SQLite Documentation](#).