Republic of the Philippines
Polytechnic University of the Philippines
Sta. Mesa, Manila
College of Computer and Information Sciences

BarrioSeguro: Barangay Monitoring and Incident Reporting System

A Project Documentation presented to the
Faculty of the College of Computer and Information Sciences,
Polytechnic University of the Philippines

In partial fulfillment for the course
COMP 20133 – Applications Development and Emerging Technologies

Submitted by:
Dela Cruz, Juan Gabriel
Estonilo, Julius Evan
BSCS 3-3

Submitted to:
Prof. Aleta A. Fabregas, DIT

February 4, 2025

I) Introduction

Barangays are the most basic political and administrative divisions in the Philippines, and more than 42,000 barangays are spread across the country [1], [2]. Being the primary access points for local initiatives and community participation, barangays are therefore crucial in addressing public safety concerns, managing local affairs, and maintaining order [3]. However, barangays face significant challenges in information management, monitoring local activities, and disseminating timely alerts [5]. These problems are further aggravated by dependence on manual procedures that bar barangays from performing their administrative functions of providing certification and reporting incidents properly [4]. The absence of streamlined systems typically results in inefficiencies that harm organizational effectiveness as well as public trust [1], [8].

At an even larger scale, a study reveals that although incident report mechanisms have full potential to enrich governance if properly utilized, the presently designed structures are often inadequate and, in some areas, not user-friendly enough. This thereby deters how much users' total potential might be realized [7], [8]. In addition, many of these systems neglect to meet the unique needs of small-scale operations vital to barangay operations. These challenges underscore the need for technology-based interventions designed for barangay contexts. Modernized systems will help overcome bottlenecks, improve response effectiveness, and establish community trust to enable communities to act responsively in addressing local concerns and promoting transparency and responsibility [7], [8].

By using emerging technologies and incorporating findings from improved incident reporting systems, barangays can transition from older models to modern approaches that enhance their capacity for effective public administration and statecraft [4], [7].

II) Problem Identification

In consequence, barangays face great challenges in implementing peace and order in their respective areas because they are dependent on traditional and old manual systems [3], [4]. Such systems are often associated with the use of physical paper documentation and primitive audio devices for alerts, causing delay in response operations and ineffective data handling [4], [7]. The lack of an efficient mechanism in the storage, retrieval, and analysis of vital information also creates inefficiency among incident management systems [3], [7]. These old techniques raise the chances of incorrect records and block trust and responsibility in barangay governance [5], [7].

Another major issue is the lack of barangay-level systems' coordination with other institutional structures, like the Philippine National Police. The lack of coordination in this regard reduces the accuracy and swiftness of responses and, therefore, fails to deliver in dealing with critical community problems [4], [6]. In addition, due to a lack of proper mechanisms for detecting trends and patterns in incidents, barangays fail to exhibit a reduced effectiveness in taking proactive measures to safeguard their communities [4], [7].

These inefficiencies are continued, creating weaknesses that impact the comprehensive safety and operational effectiveness of barangay activities [3], [5]. Research emphasizes the fact that the provision of technology-driven solutions to barangays can enhance data management, responsiveness, and strengthen their partnership with larger institutions [7], [8]. Without updated

systems, barangays stand the risk of failing to fulfill their obligations as the primary units of grassroots governance [7]. Tackling these deficiencies through creative approaches is essential for promoting resilience, enhancing operational efficiency, and cultivating trust within barangay communities [4], [8].

III) Solution

The researchers suggest a proposed solution for the problems of inefficient barangay monitoring and incident reporting system with the development of a **desktop application** called "***BarrioSeguro***".

The key functionalities and features of the application will be as follows:

1	**Incident Reporting Module**	This feature allows barangay officials to digitally log, organize, and store incidents information.
2	**Community Announcement Feature**	The feature that allows officials to disseminate emergency alerts, barangay events, and other information quickly through local channels that can be printed or viewed within the desktop system.
3	**Resident Database Information**	Store contact information like phone numbers and emails for notices sent through text or email.
4	**Data Reports**	Barangay officials can make outlines or summaries of logs, thus, it becomes easier for them to share the reports with the higher authorities.

Table 1: *Functionalities and Features of "BarrioSeguro" Application*

The desktop application will have a **user-friendly interface** that will enable administrators to access features easily and understandable. Administrators will have access to information on residents information, incident reports, and providing announcements. It will also

have the feature of integrating with other **Application Programming Interfaces (APIs)** for text messaging and emailing services to enhance the delivery of information and receive reports and notifications in the alerts and events.

The proposed solution is to improve the governance of the barangays and increase the efficiency of incident reporting through the use of technological facilities. This is to ensure that this system can incorporate future development and enhance the connection between the barangays in the future. With the help of *BarrioSeguro*, the researchers are hoping that the barangays will be able to enhance their governance, gain the trust of the people, and make the communities safer places to live in.

System Architecture

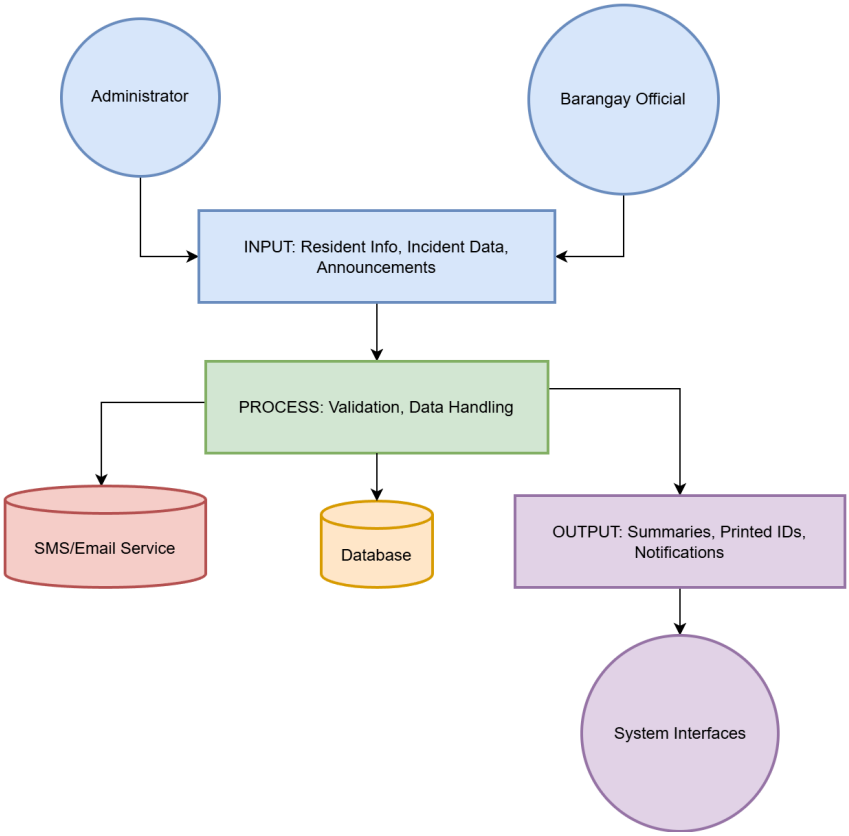


Figure 1: *System Architecture of "BarrioSeguro" Application*

Our system comprises **five (5) main components**:

1	**Users**	Barangay Official and Administrator, which are the primary users who provide key data and also oversee operations
2	**Processing Module**	where the inputs are validated, stored and transformed
3	**Database**	the repository of all the records which is also set as a secure one

4	**Third Party Services**	real time alerts through SMS / Email APIs
5	**System Interfaces**	where interfaces generator get summaries and verify notification interface and generate IDs for printing

Table 2: *Five Key Components with their Description about BarrioSeguro's System Architecture*

This layered approach to integrating barangay management processes ensures data entry provides for organization, efficiency and transparency of overall barangay management processes.

IV) Interface

User Interfaces

A. Login Interface

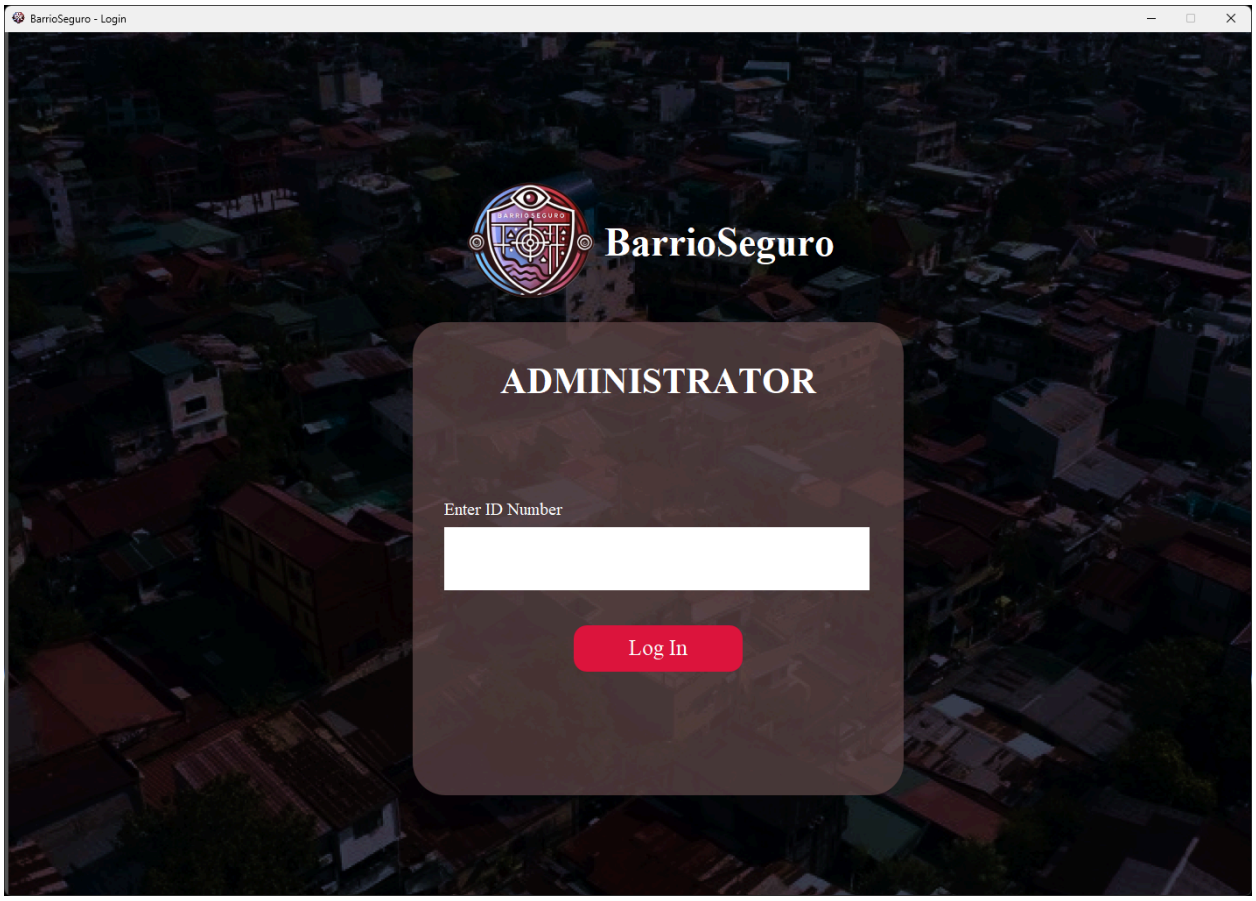


Figure 2: *Administrator Landing Page*

At the center, the interface features a login form where administrators are prompted to enter their **ID Number** to gain access. Below the input field, a **"Log In" button** enables secure entry into the application.

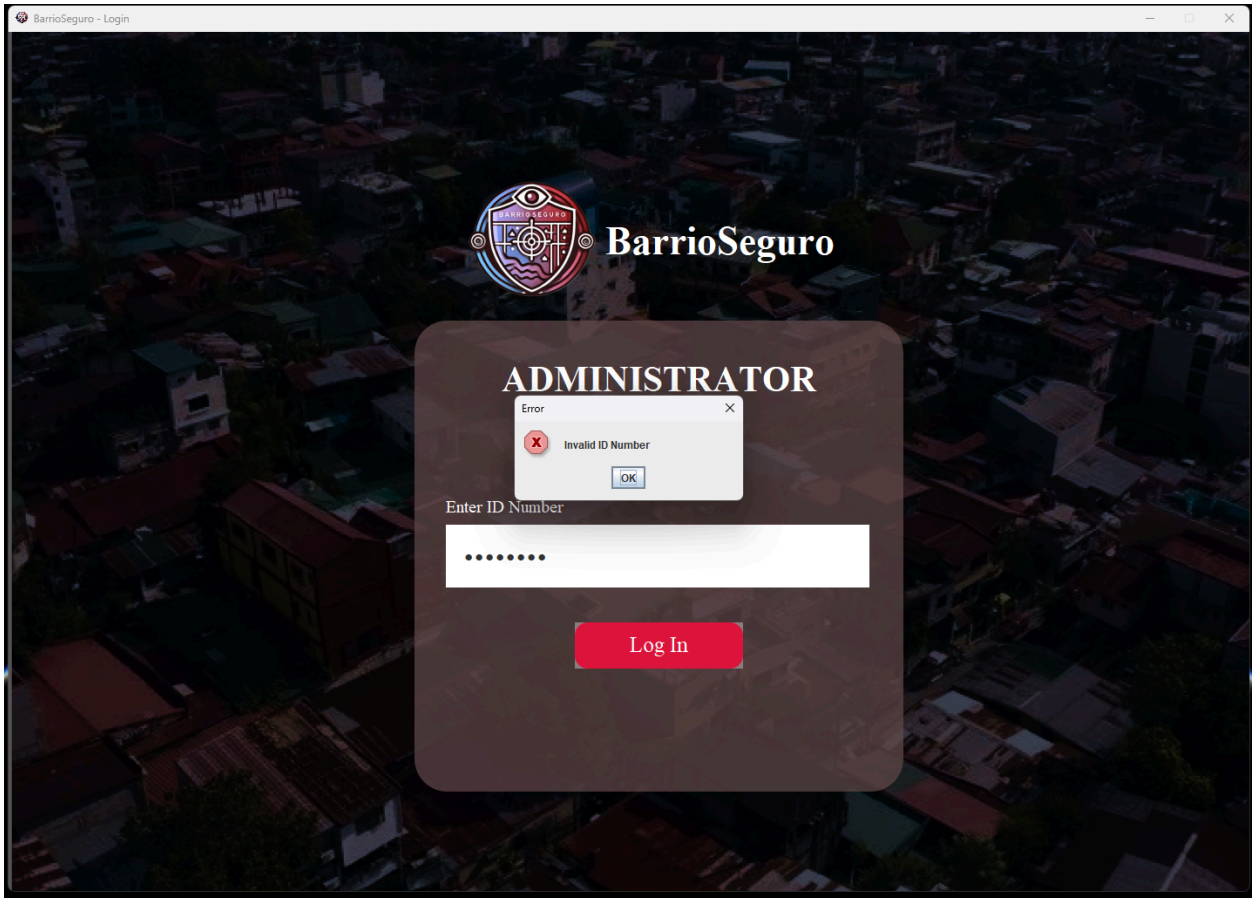


Figure 3: *Login interface when wrong ID number is entered after clicking "Log In."*

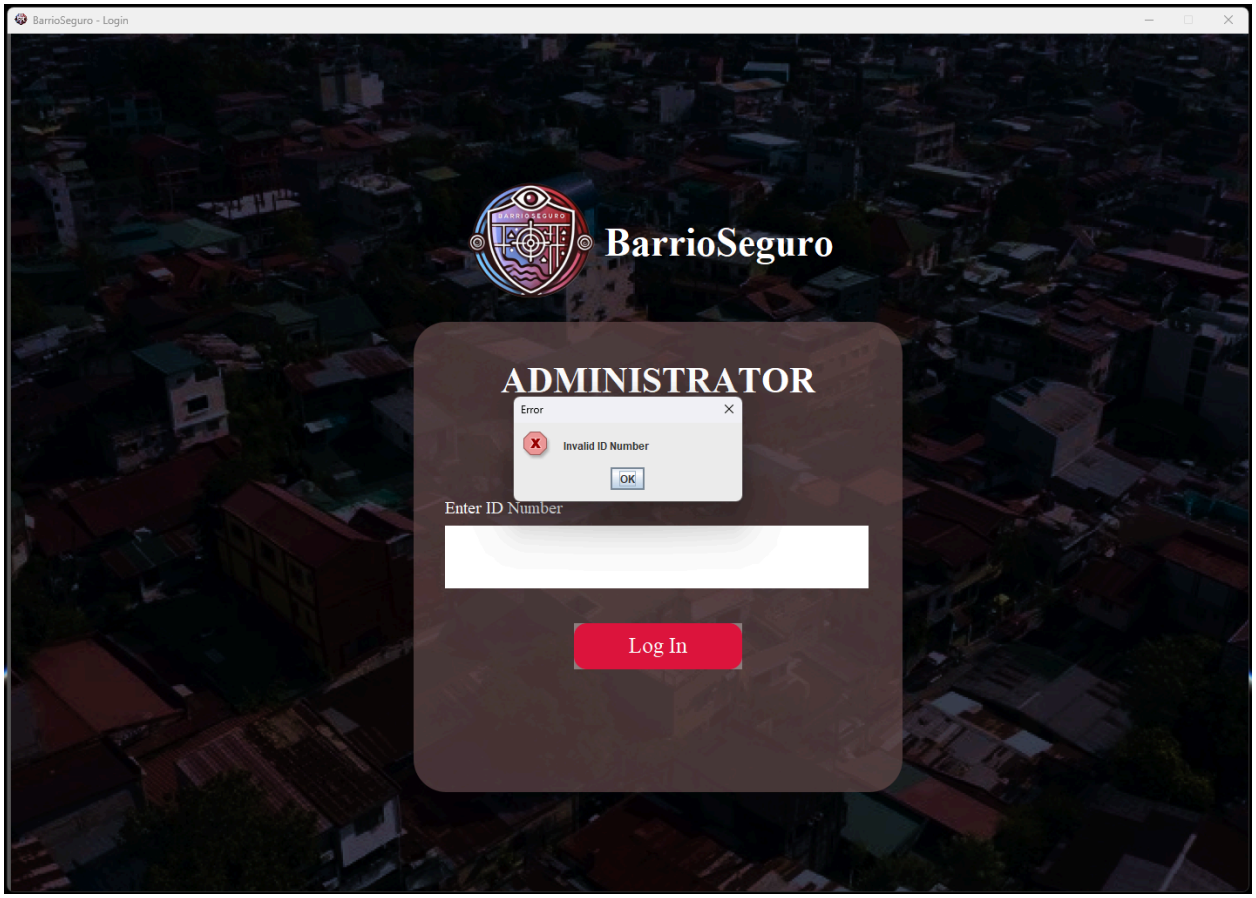


Figure 4: *Login interface when no ID number is entered after clicking "Log In."*

B. Menu Interface

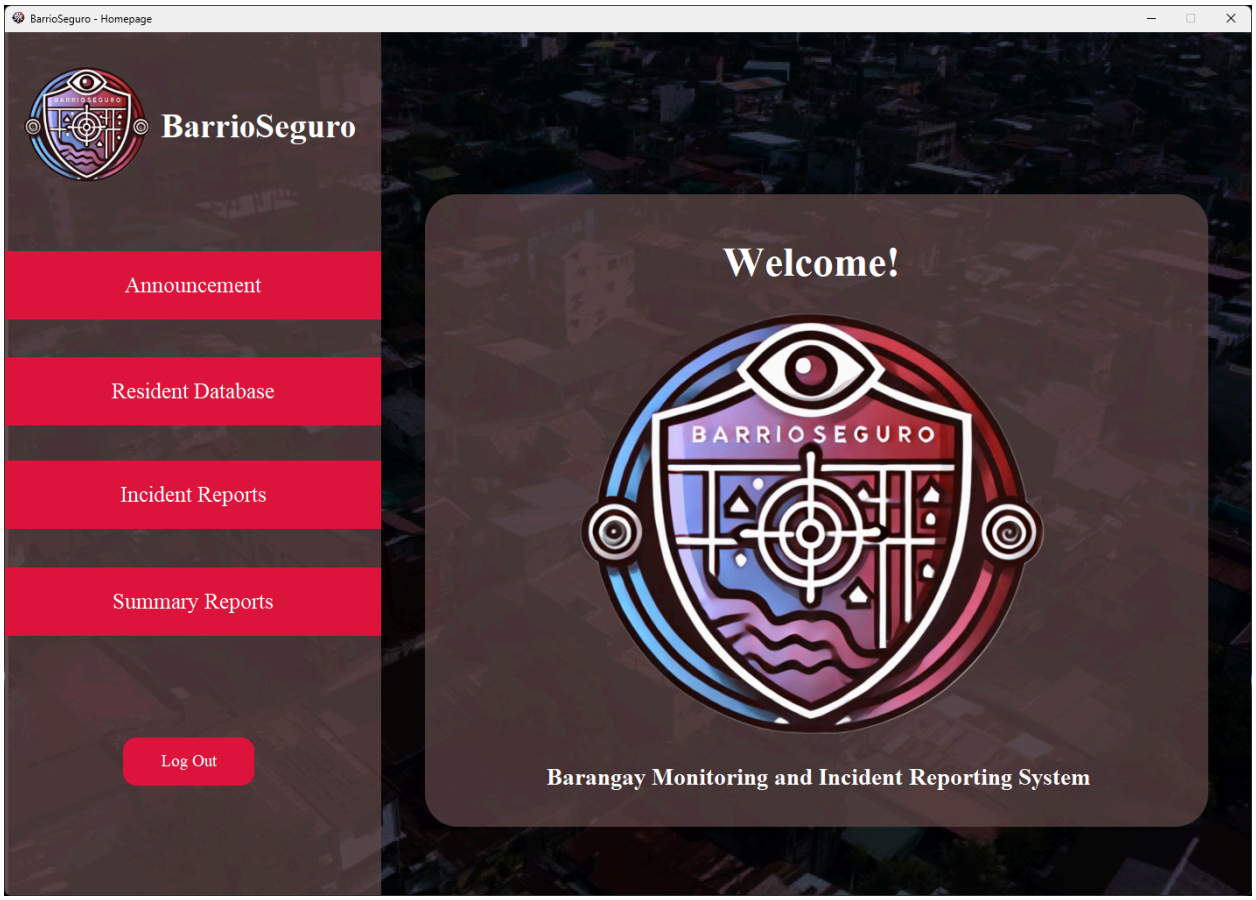


Figure 5: *Welcome Page*

This image depicts the **Welcome Page (Menu Interface)** of the **BarrioSeguro** desktop application. On the left-hand side, a vertical menu lists the core functionalities of the system, including:

1	**Announcements**	Allows barangay officials to disseminate updates or emergency alerts to the community.
2	**Resident Database**	Provides access to residents' information for efficient management and communication purposes.
3	**Incident Reports**	Enables logging and tracking of incidents within the barangay.
4	**Summary Reports**	Generates comprehensive outlines of data summary.
5	**Log Out**	At the bottom of the menu is a button for secure system exit.

Table 3: *Lists of Core Functionalities of the System*

C. Announcement Interface

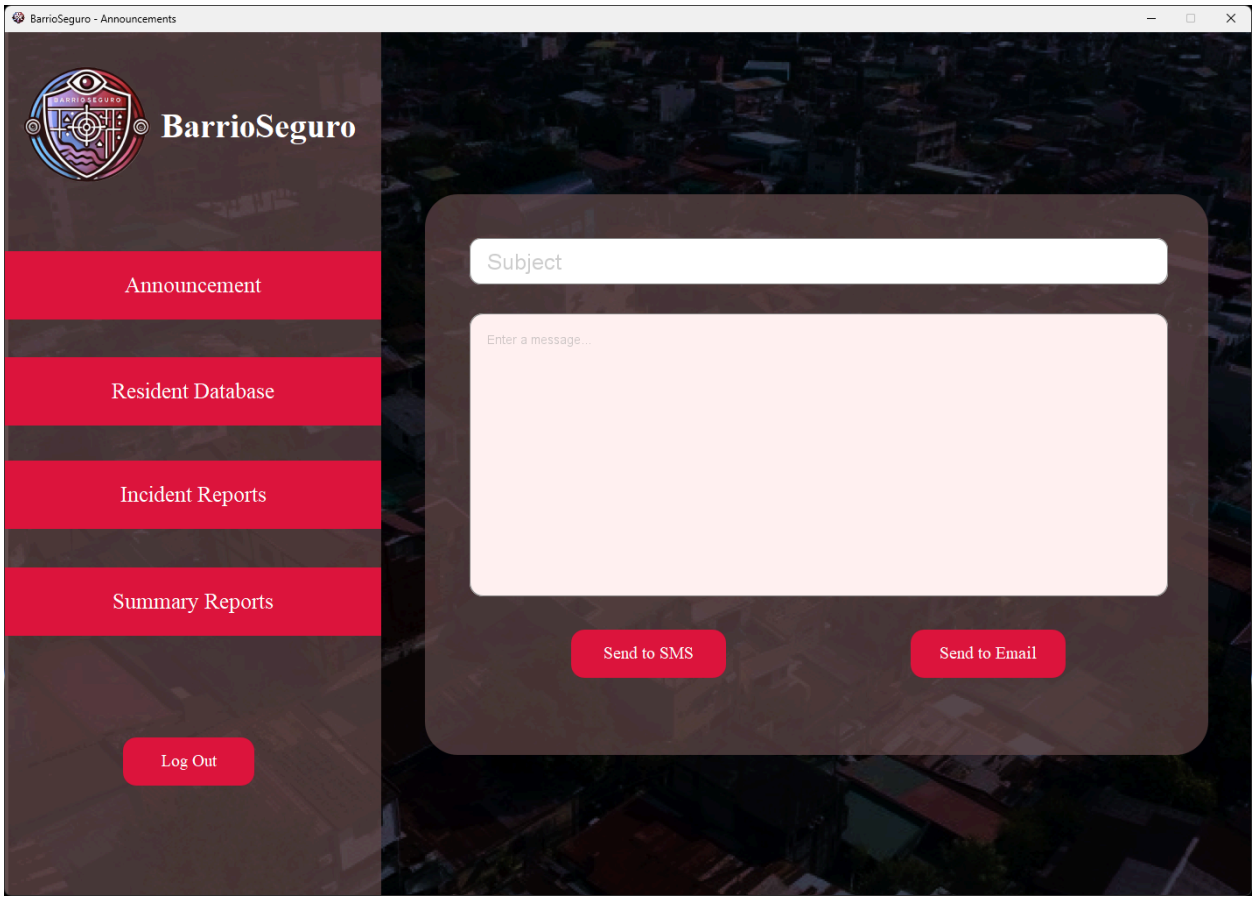


Figure 6: *Community Announcement Page*

This interface is designed for barangay administrators to compose and send announcements to residents quickly and efficiently. Key features include:

1	***Input Fields***	Text box for **subject** and **message** where administrators can type announcements or emergency alerts
2	***Send Options***	Buttons labeled "**Send to SMS**" and "**Send to Email**" allow administrators to disseminate the announcement through text messages or emails.
3	***Efficient Navigation***	The menu on the left aids users in seamlessly accessing other features, such as incident reports or the resident database.

Table 4: *Key Features of Announcement Page*

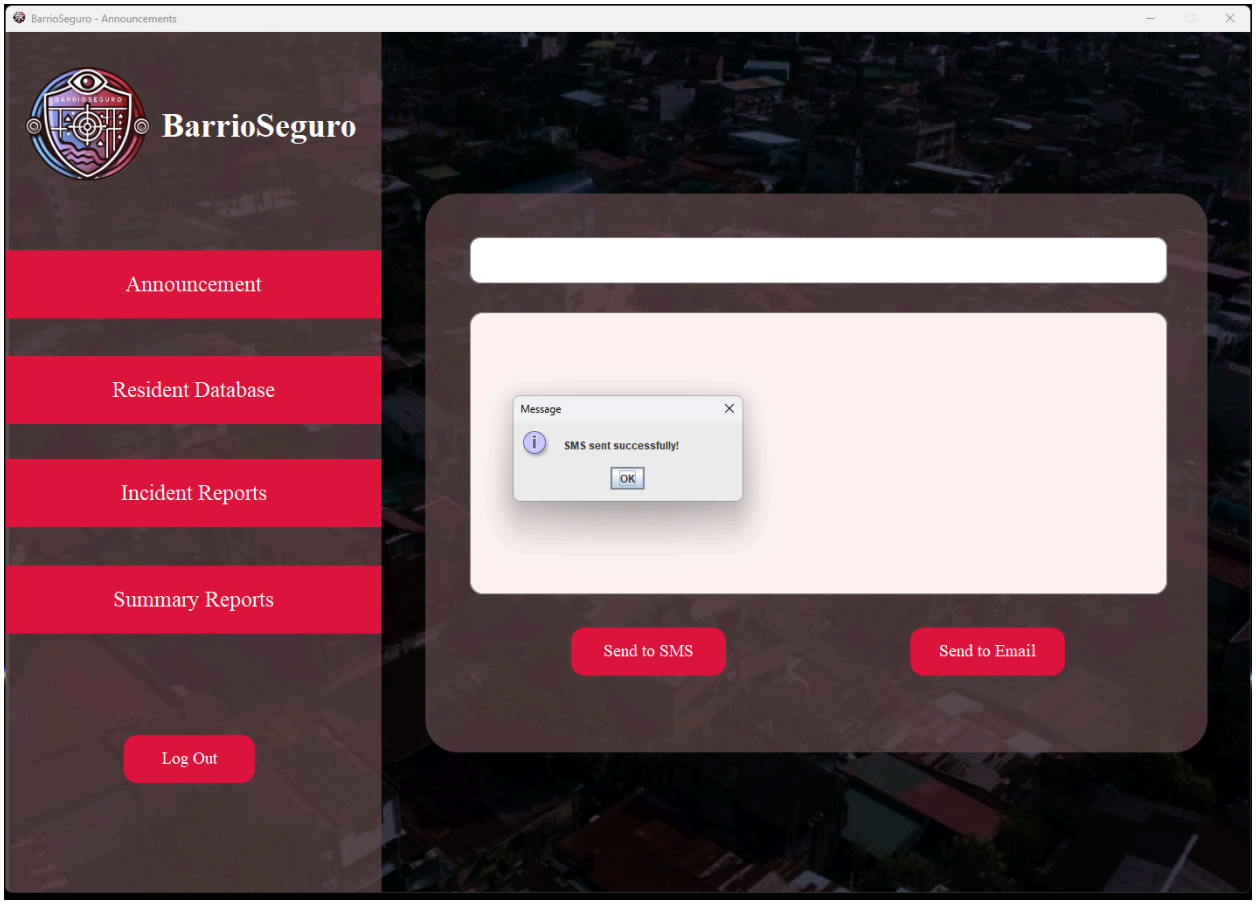


Figure 7: *Announcement interface after successfully sending an SMS.*

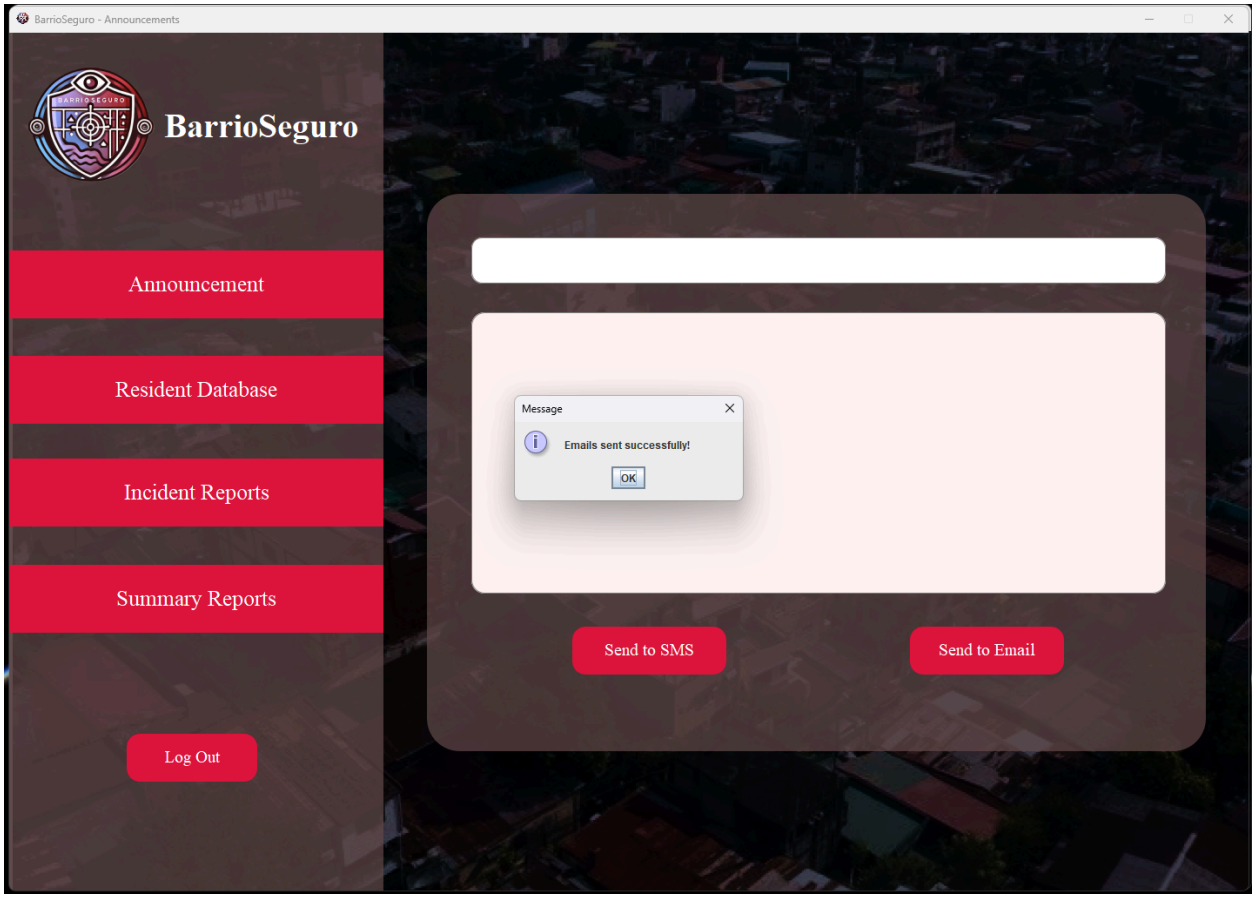


Figure 8: *Announcement interface after successfully sending an email.*

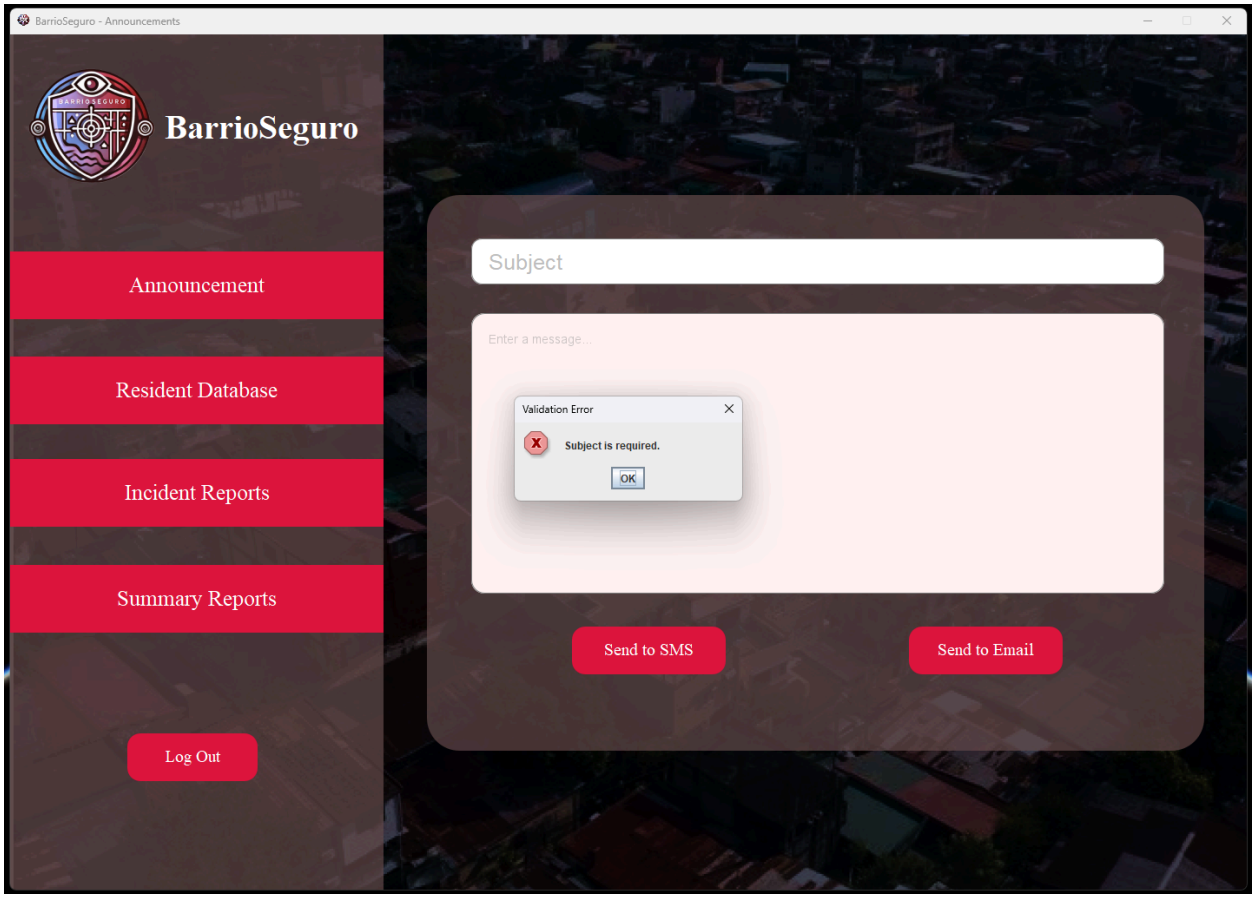


Figure 9: *Announcement interface showing a validation error when no subject is entered after attempting to send SMS or email.*

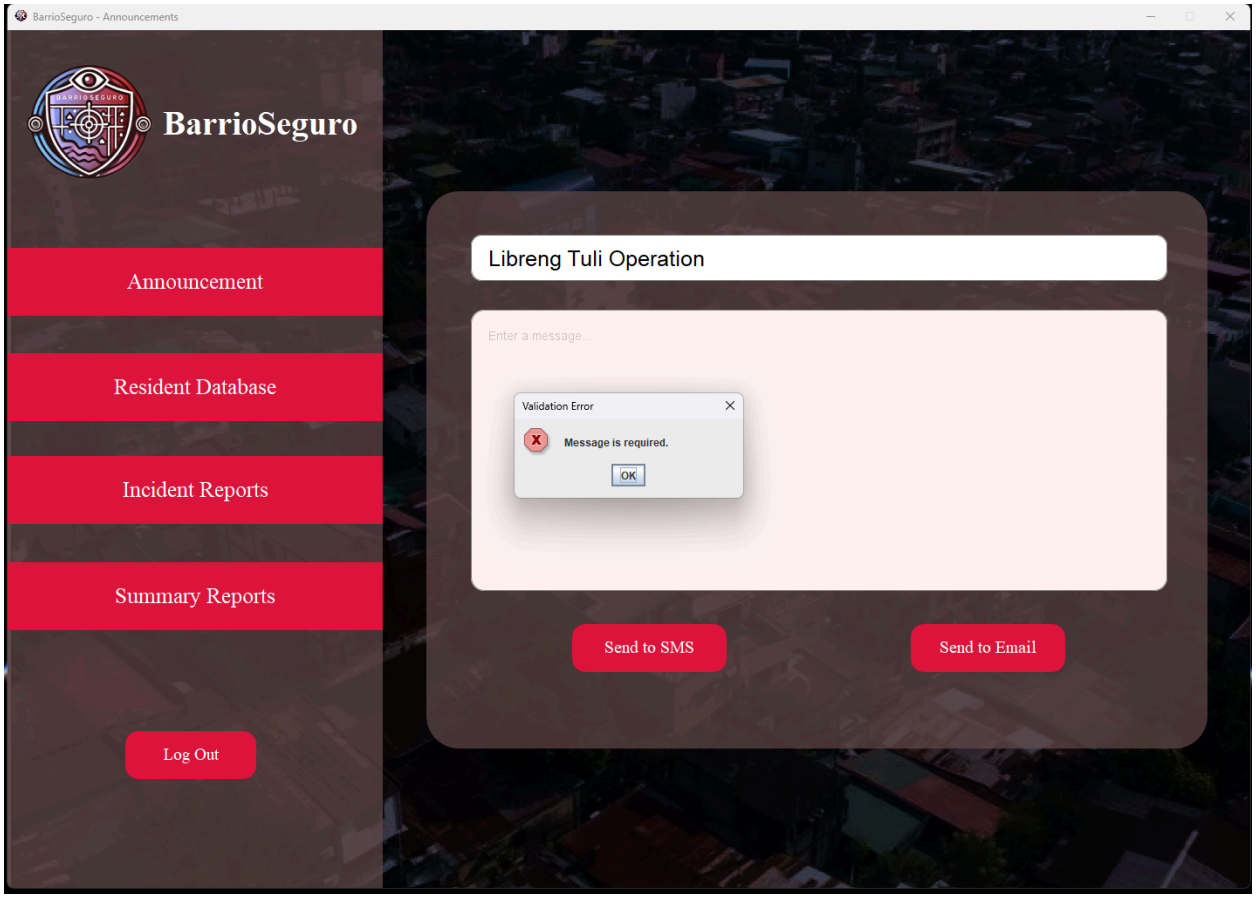


Figure 10: *Announcement interface showing a validation error when no message is entered after attempting to send SMS or email.*

D. Resident Information Interface

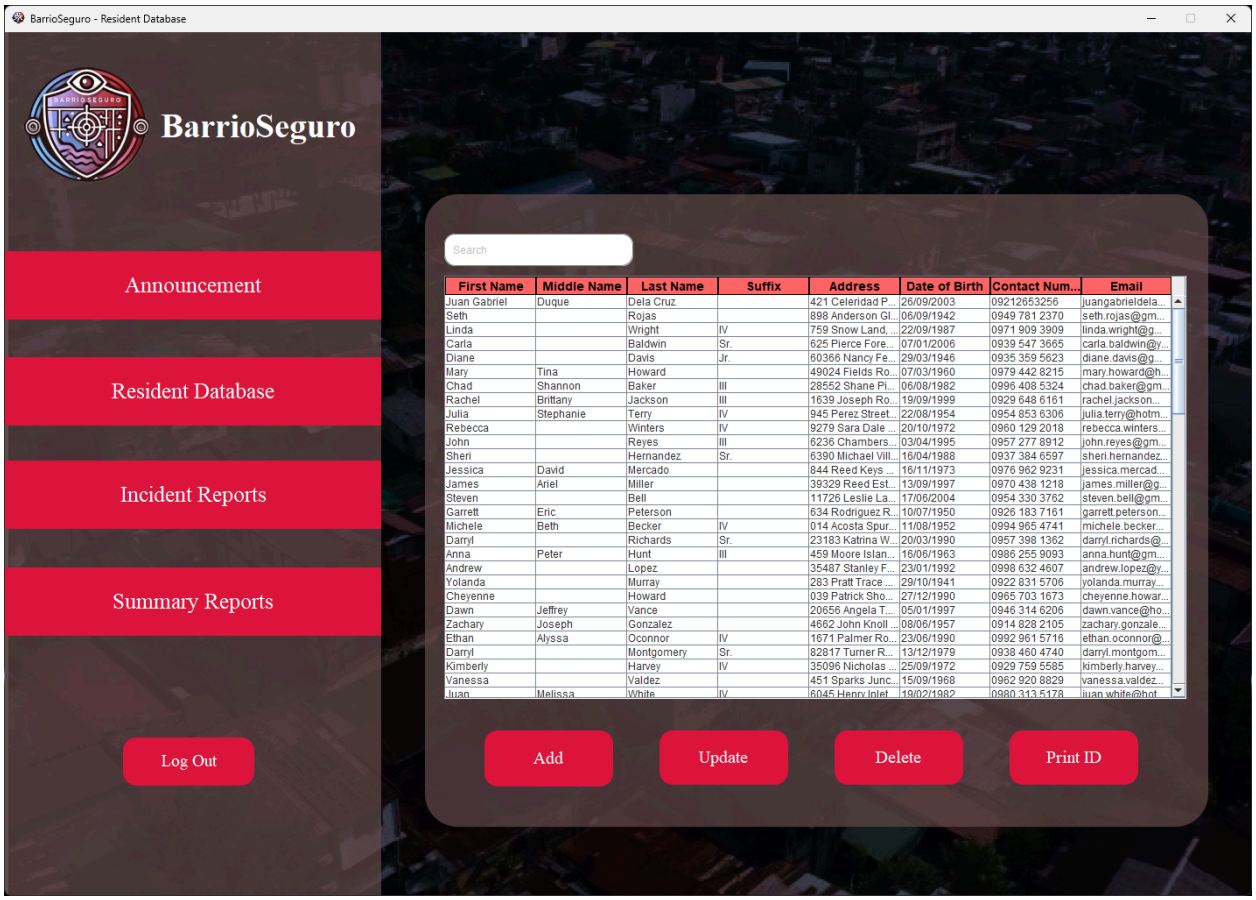


Figure 11: *Resident database showing a list of all registered residents.*

This interface enables barangay administrators to manage and organize resident data effectively. Key features include:

1	**Table Display**	A grid layout to list resident details such as names, addresses, date of birth, contact numbers, and emails.
2	***Action Buttons***	**Add:** Allows the entry of new resident information. **Update:** Enables editing of existing records. **Delete:** Removes outdated or incorrect entries. **Print ID:** Facilitates the issuance of identification cards for residents.
3	***Search and Scroll Functionality***	Enhances ease of locating specific resident data.

4	***Efficient Navigation***	The menu on the left aids users in seamlessly accessing other features, such as incident reports or the resident database.

Table 5: *Key Features of Resident Database Page*

D.1. Adding a Resident

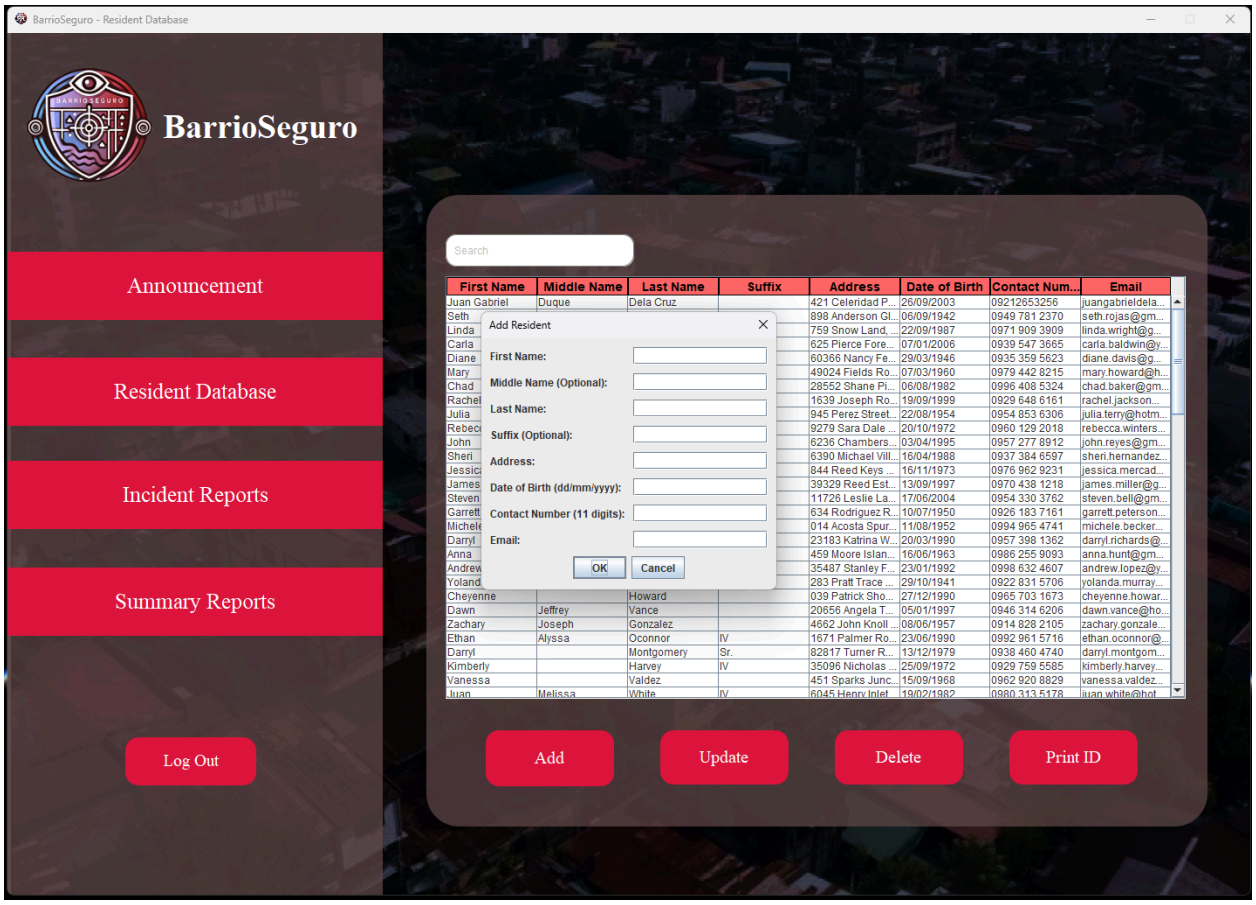


Figure 12: *Resident Addition Page*

This interface allows barangay administrators to **add a new resident's details to the database**. It includes fields for the resident's **full name**, **contact number**, **address**, **email**, and **date of birth**. Upon entering all the required information, pressing the "**OK**" button stores the data securely and pressing the "**Cancel**" button does not add this new data. This feature ensures that accurate records of barangay residents are created for future reference.

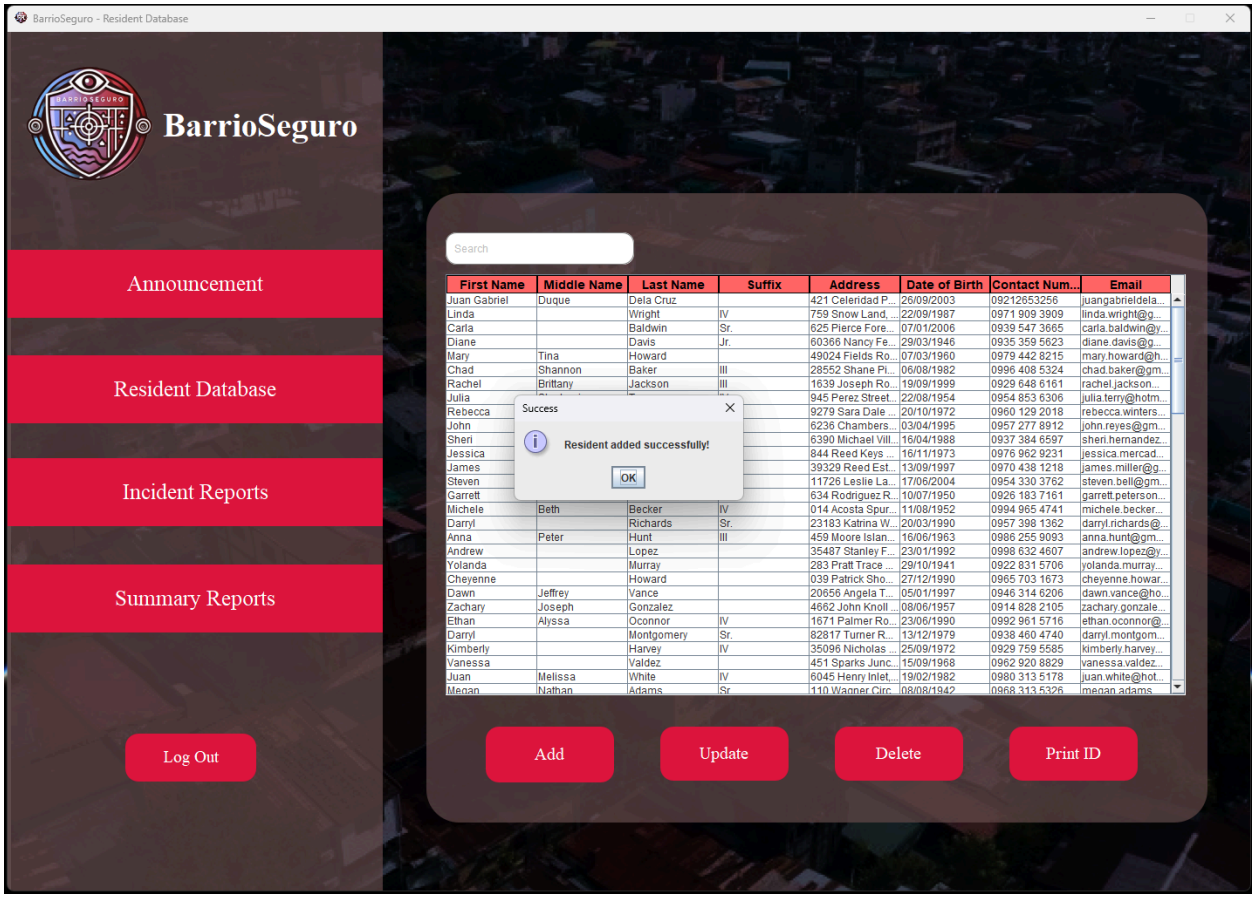


Figure 13: *Resident database updated after successfully adding a new resident.*

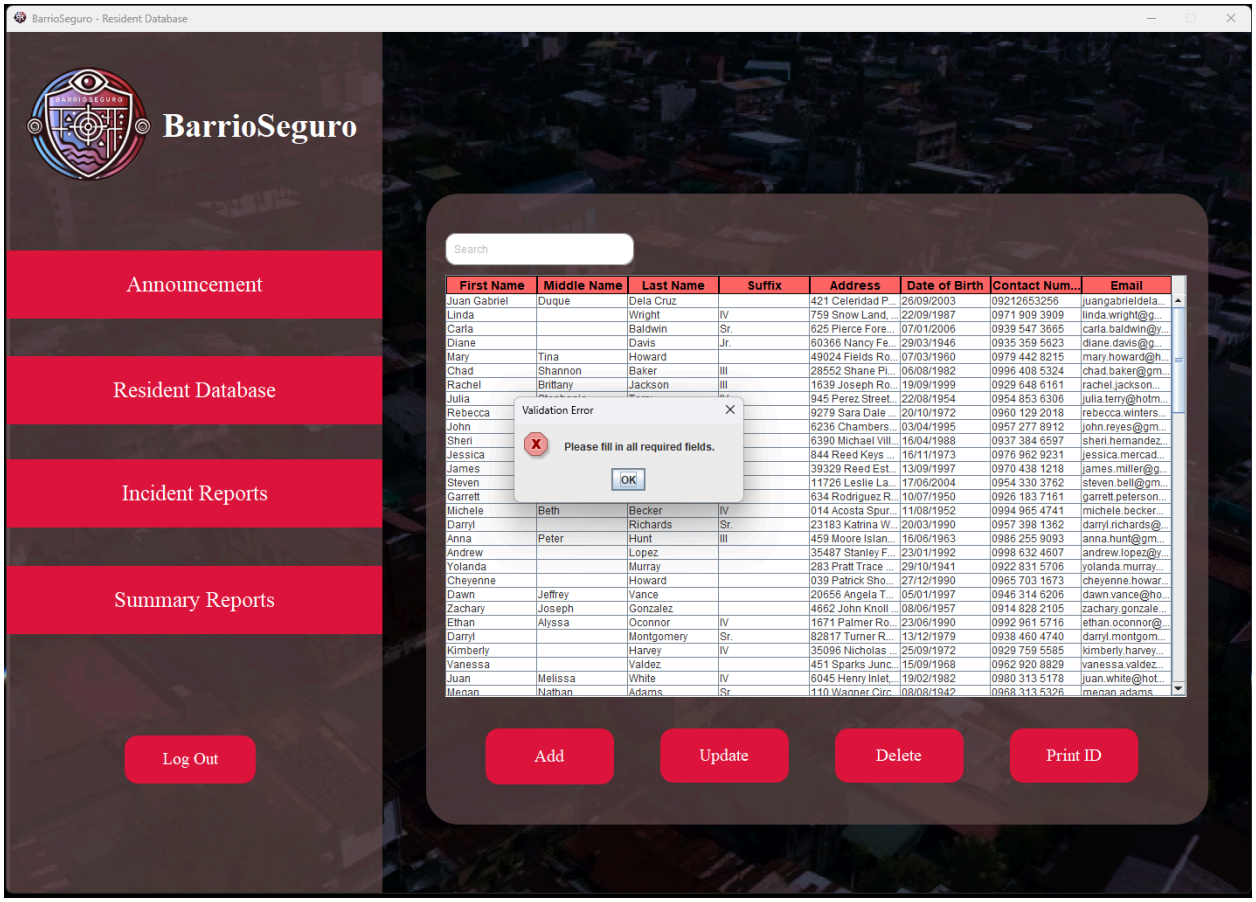


Figure 14: *Add resident interface with incomplete inputs, showing a validation error.*

D.2. Updating a Resident's Information

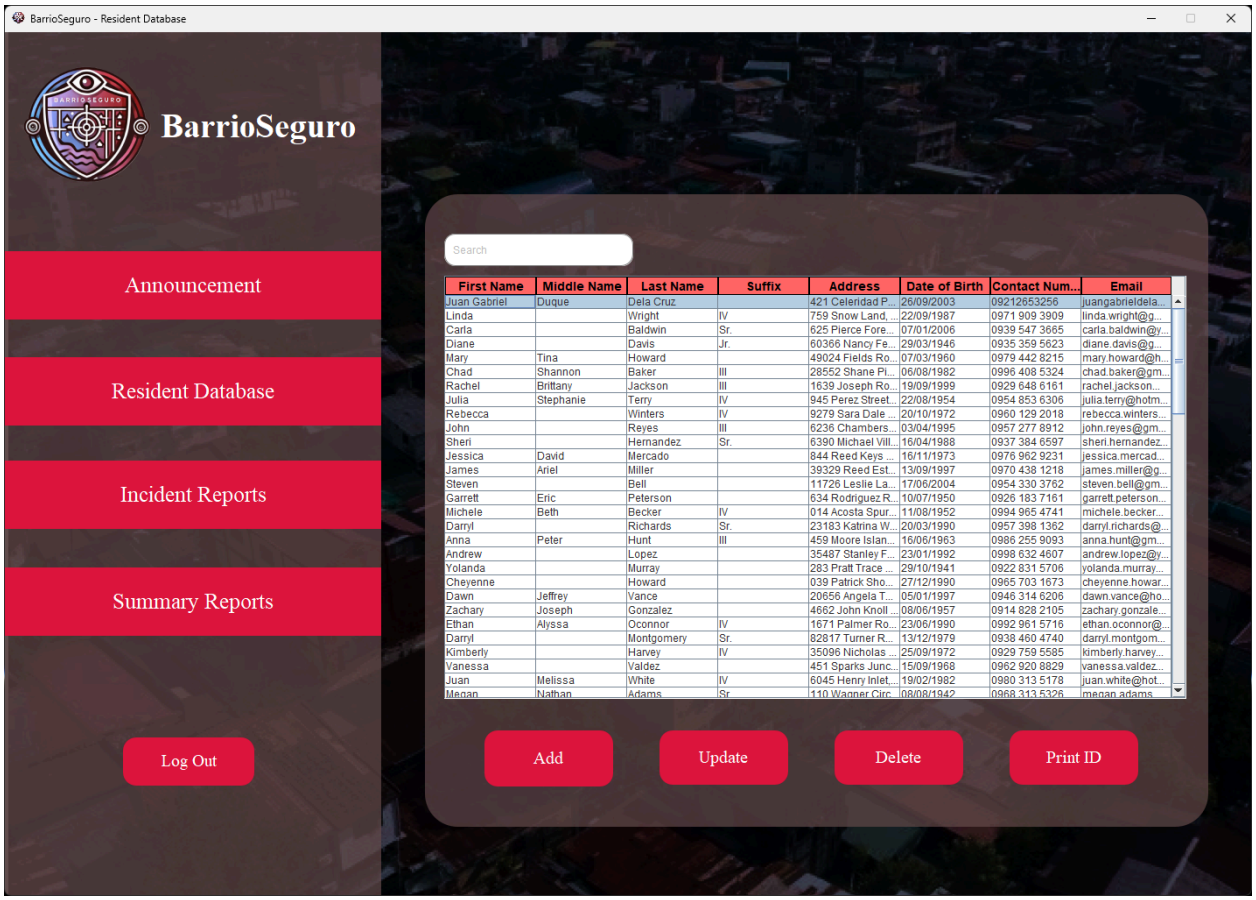


Figure 15: *Resident database interface after clicking a specific row.*

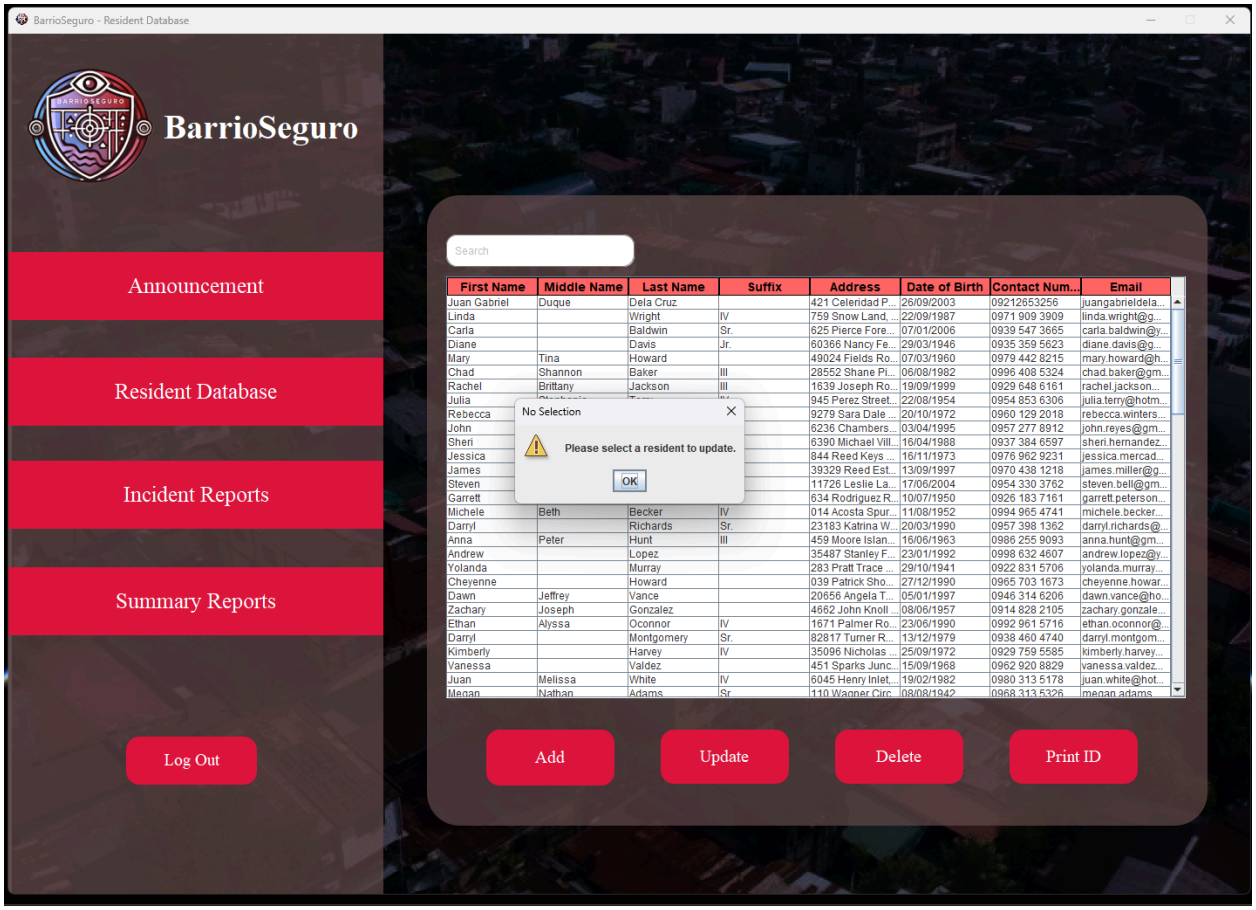


Figure 16: *Attempt to click the "Update" button without selecting a row, showing a warning.*

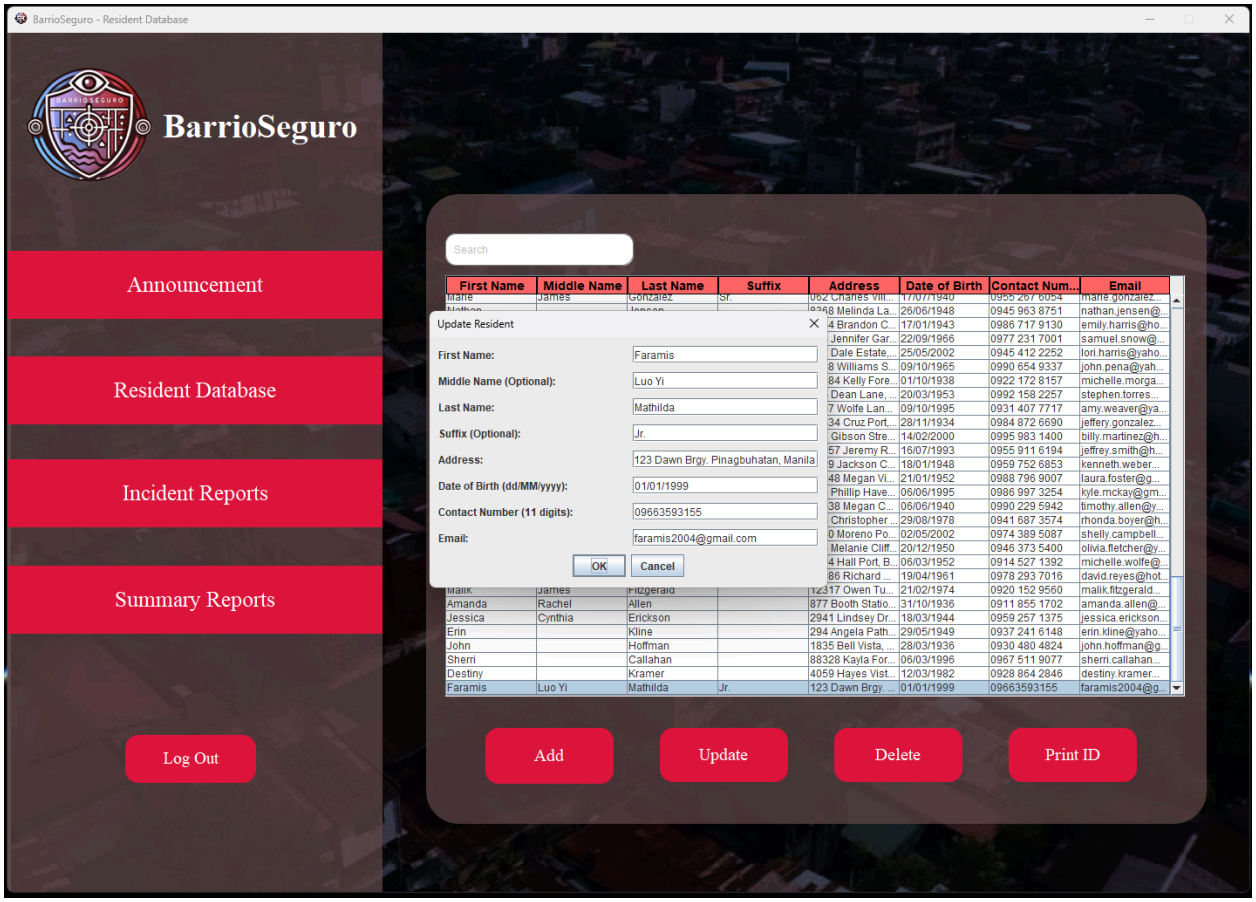


Figure 17: *Resident Update Page*

This interface is used for **updating an existing resident's information**. Administrators can search for a resident, modify their details (e.g., contact number, address), and save the updated data using the "**OK**" button. Clicking the "**Cancel**" button does not save any modified details. This ensures that the barangay database remains **accurate and up-to-date**.

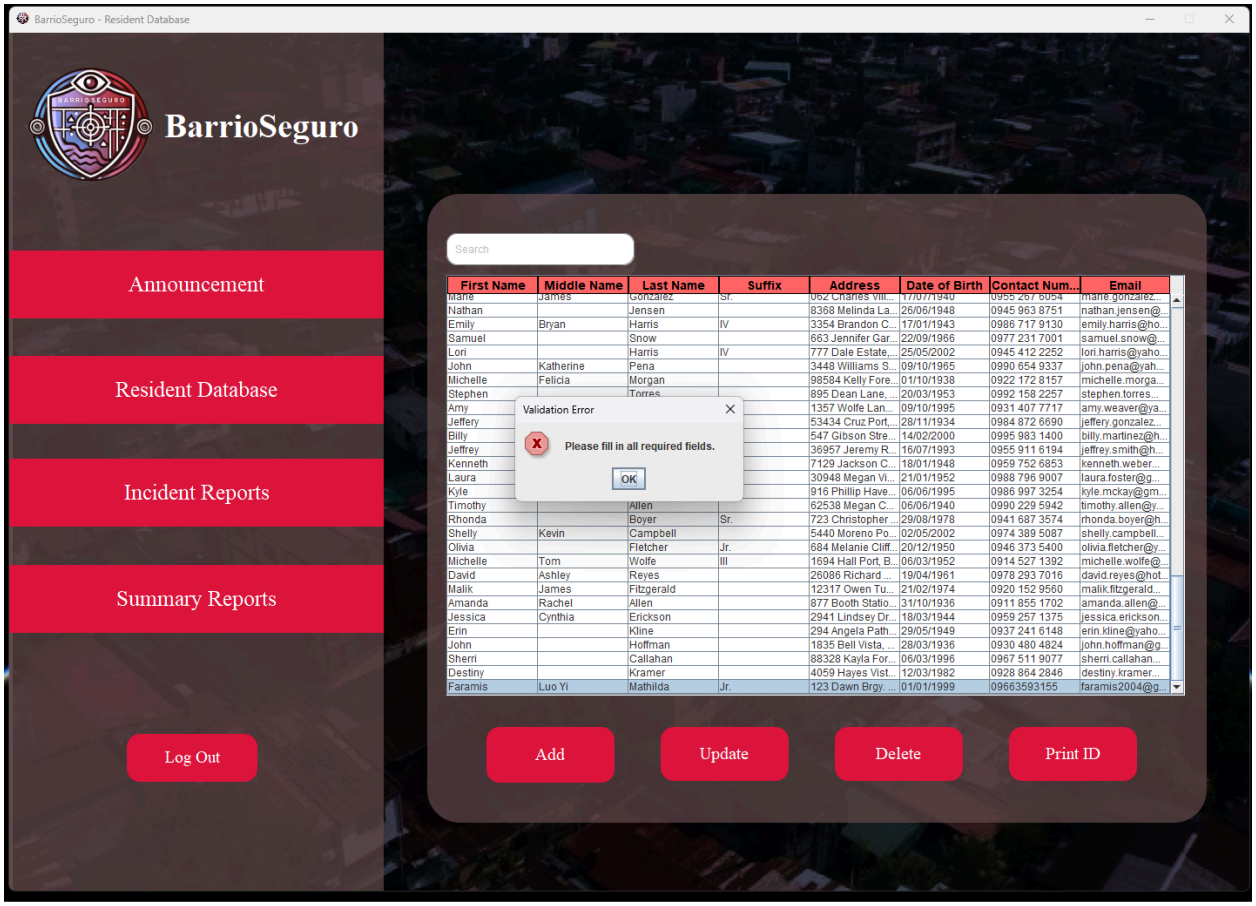


Figure 18: *Update interface showing a validation error due to incomplete input.*

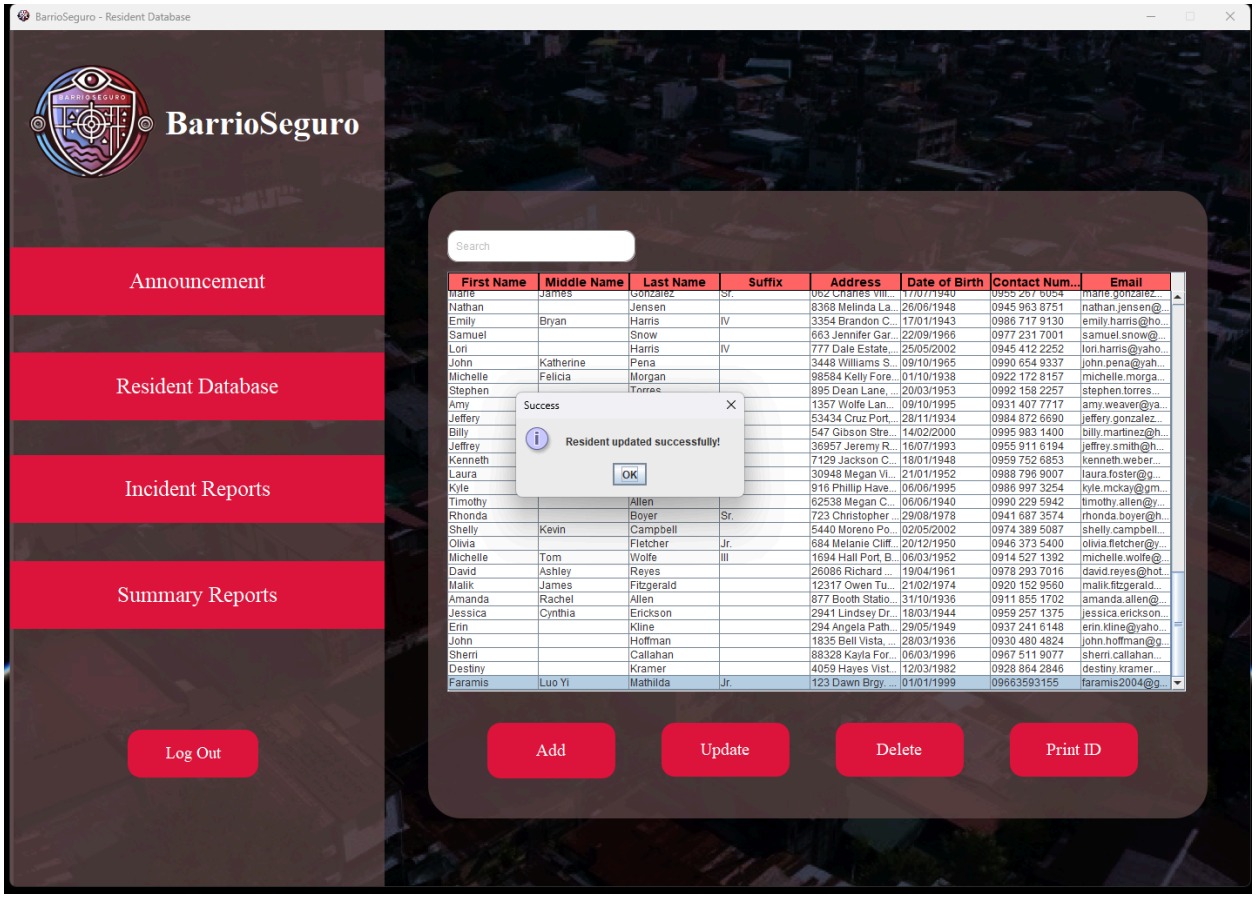


Figure 19: *Resident database updated after modifying a resident's information.*

D.3. Deleting a Resident's Information

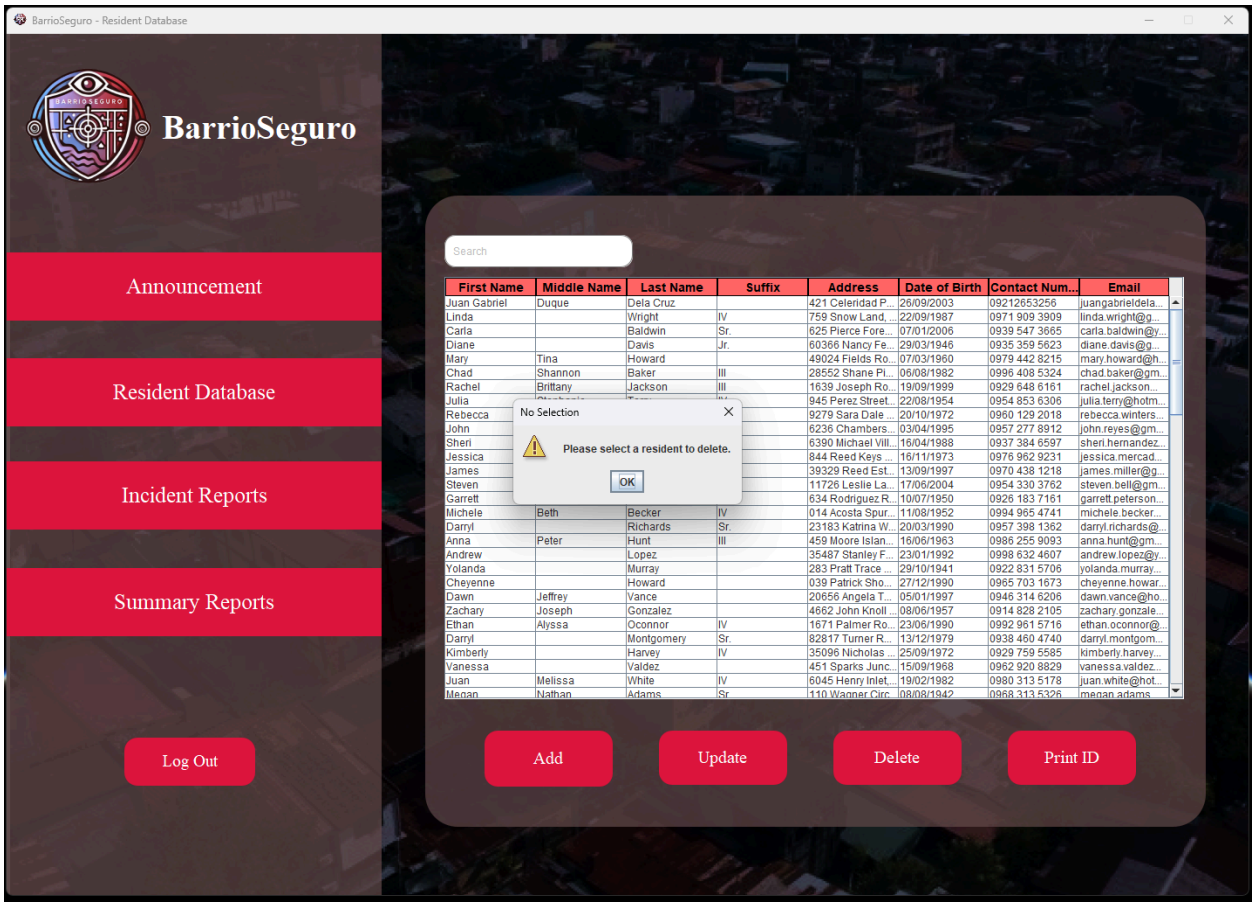


Figure 20: *Attempt to delete a resident without selecting a row, showing a warning.*

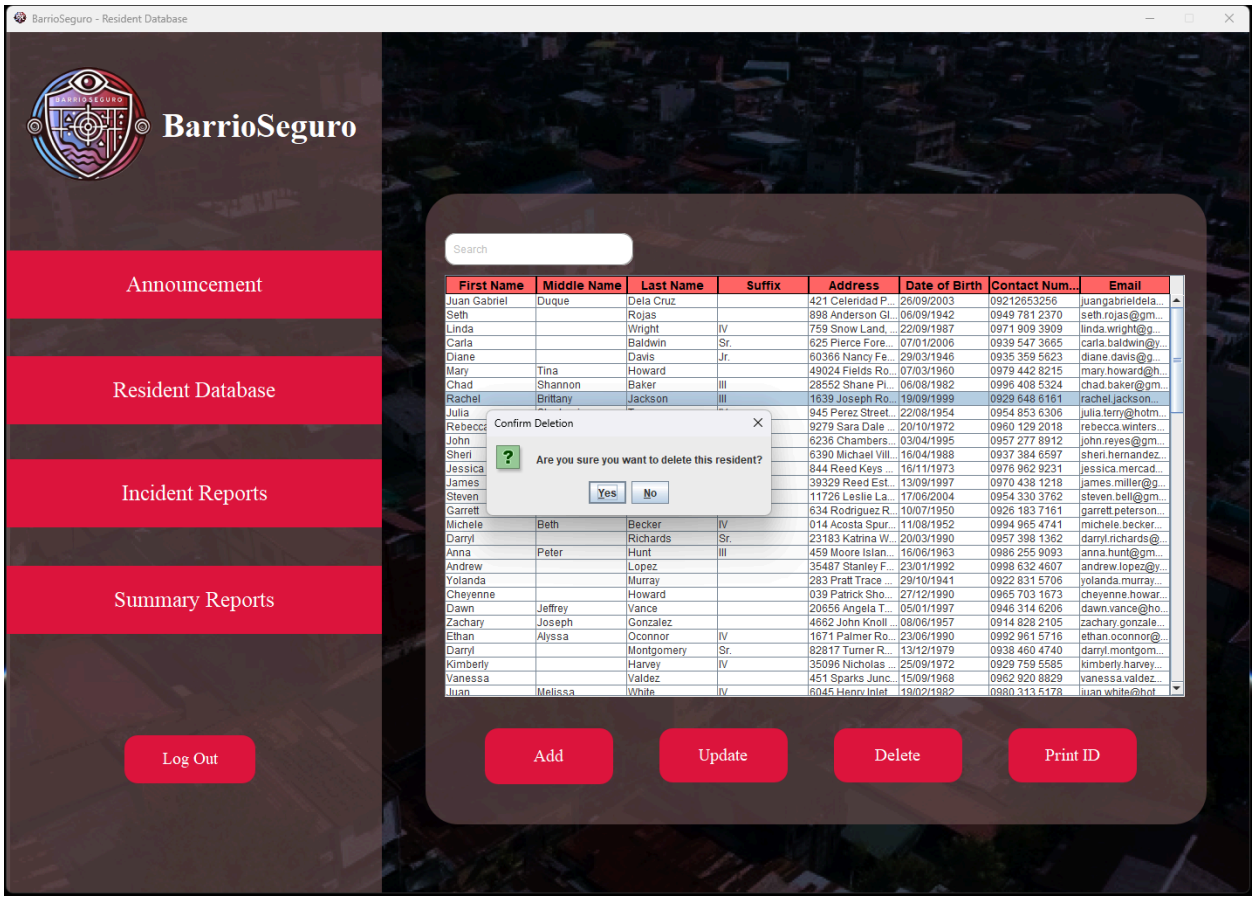


Figure 21: *Resident database interface for deleting a specific resident's information.*

This interface is used for **deleting an existing resident's information**. Administrators can search for a resident, and delete their details (e.g., contact number, address) if the administrator clicks the "**Yes**" button. Clicking the "**No**" button does not delete the specific resident's details.

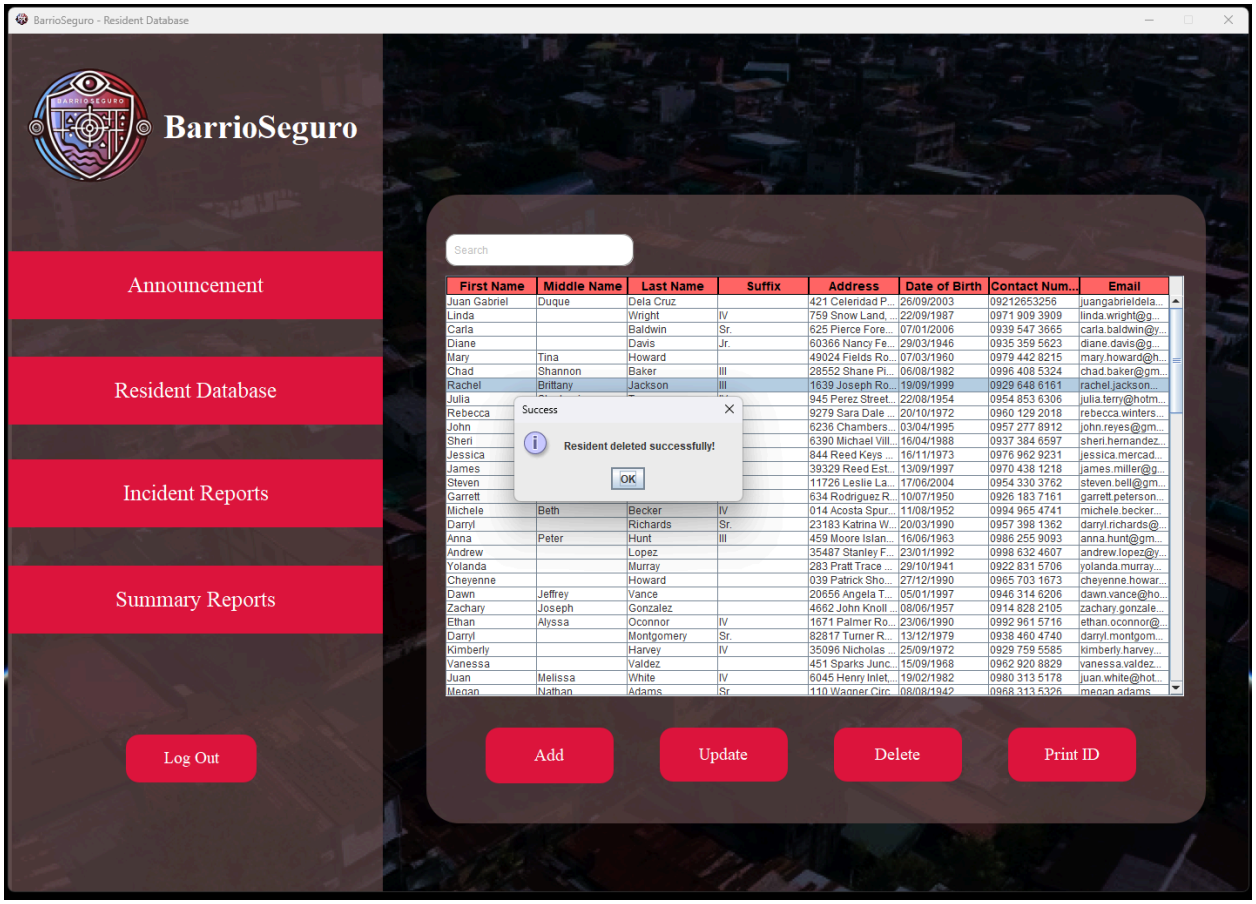


Figure 22: *Resident database updated after successfully deleting a resident's information.*

D.4. Printing ID of a Resident's Information

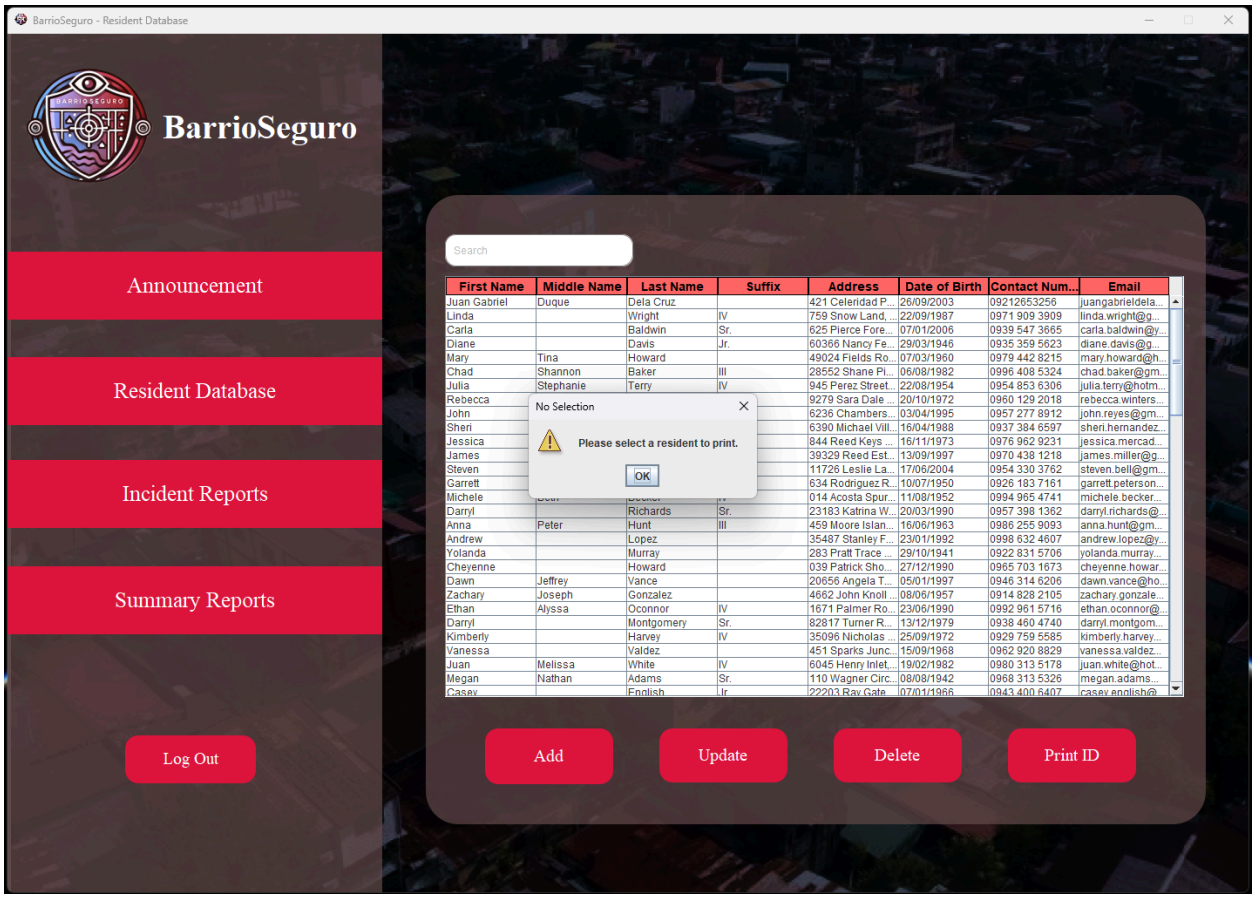


Figure 23: *Attempt to click "Print ID" without selecting a resident, showing a warning.*

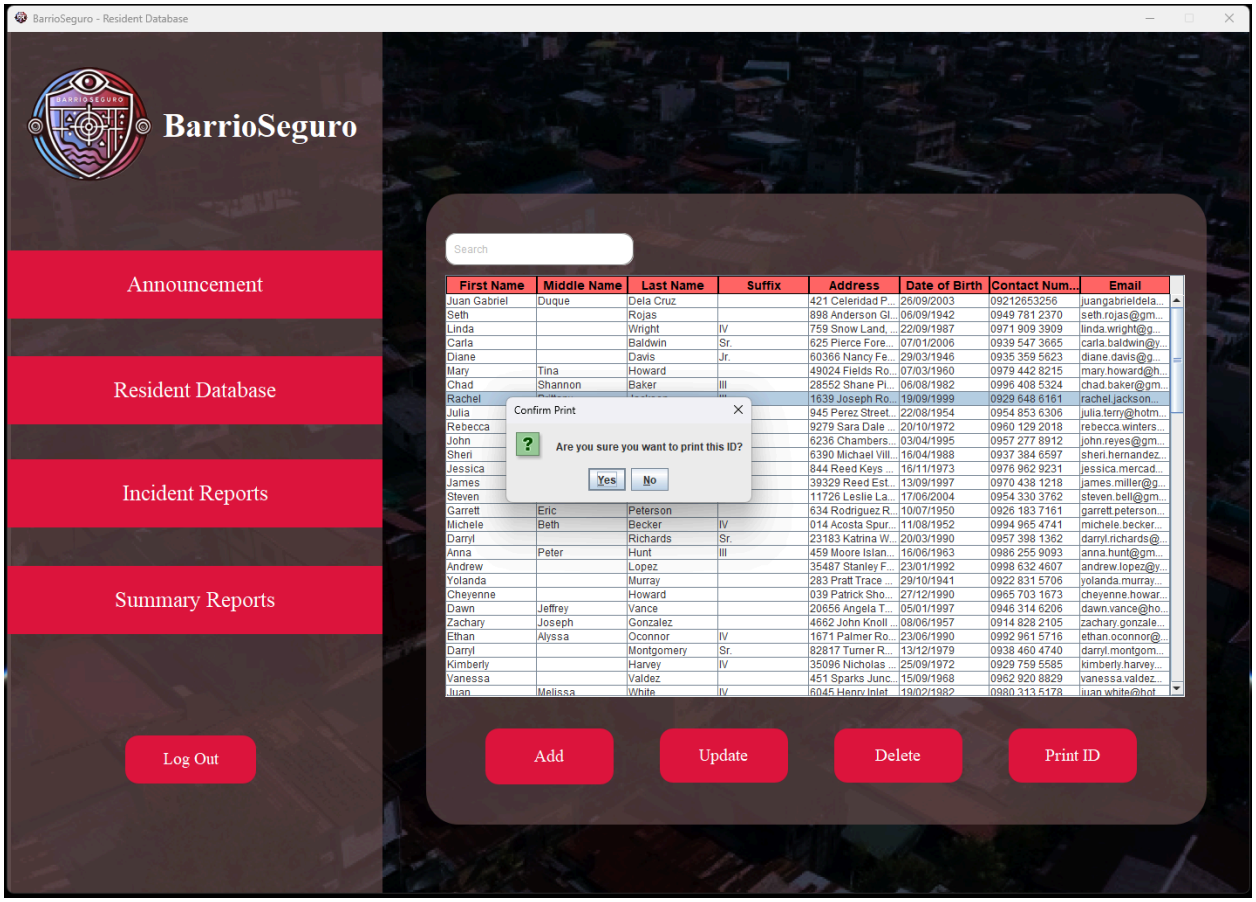


Figure 24: *Resident ID Printing Page*

This interface allows barangay administrators to **generate and print an identification (ID) card** for a resident. The option pane asks the administrator if he or she wants to print the selected resident's information using a physical ID card. Clicking the "**Yes**" button produces a **physical ID card**, which serves as an official document for the resident. Clicking the "**No**" button will not continue the printing process.

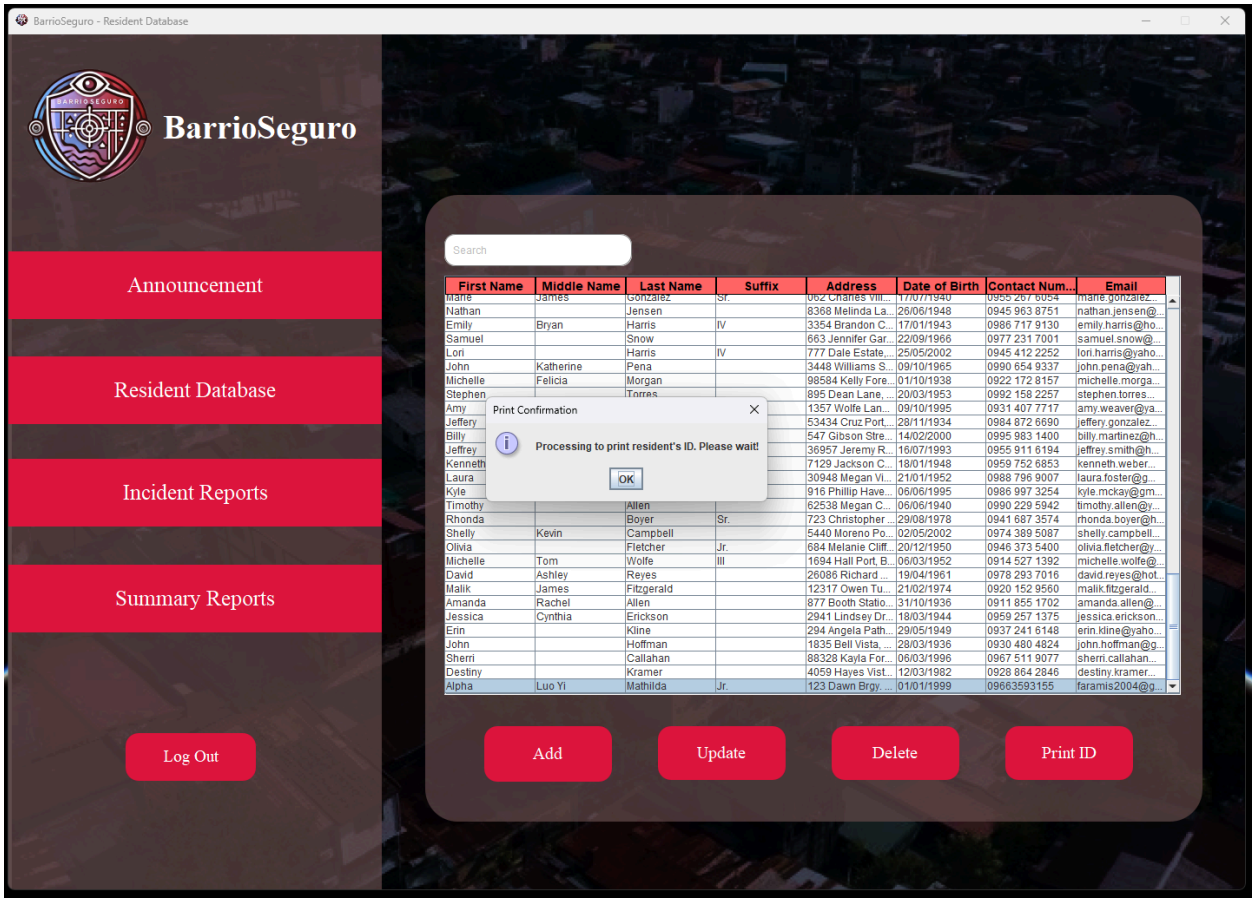


Figure 25: *Resident database interface after clicking the "Yes" button for the Resident's ID.*

E. Incident Log List Interface

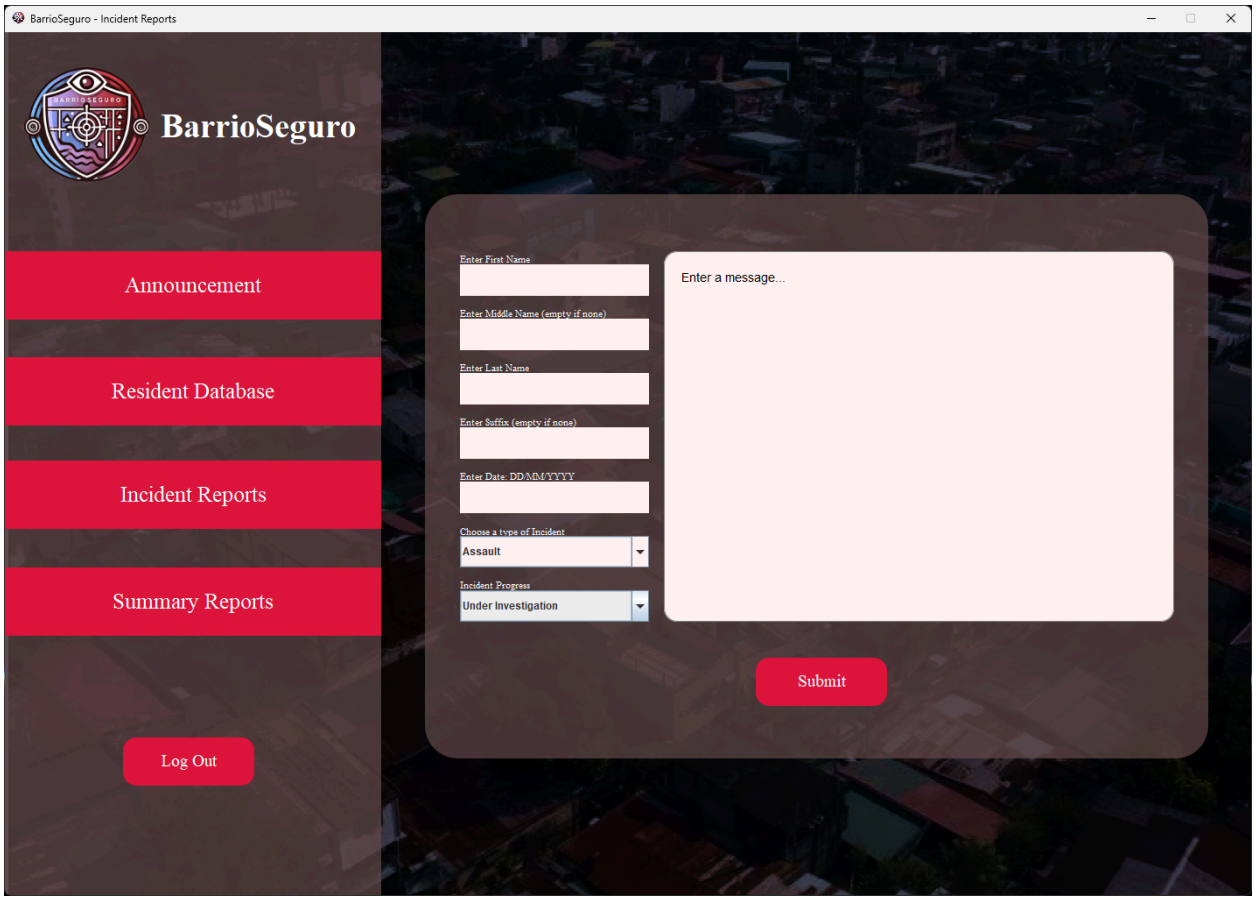


Figure 26: *Interface for creating incident reports.*

The "Incident Reports Page" interface, shown in the image, is designed for barangay officials to record and document incidents digitally:

1	**Name Input Fields**	These four text fields are provided for entering the full name of the person involved or relevant details. It divides the prompts into **first name**, **middle name**, **last name**, and **suffix**.
2	**Date Input Field**	Dedicated fields (DD/MM/YYYY) let users specify the date of the incident for proper documentation.
3	**Selection of Incident Type**	A dropdown menu allows the user to choose a specific type of incident from predefined options for accurate classification.
4	**Selection of Incident Progress**	A dropdown menu allows the user to choose a specific type of progress for this incident report.

4	**Description Area**	A large text box is available for detailed descriptions of the incident, ensuring all essential information is logged comprehensively.
5	**Submission Button ("Submit")**	Once all information is filled in, clicking the "Submit" button saves the report to the system.
6	***Efficient Navigation***	A menu on the left allows easy access to other functionalities within the application.

Table 6: *Key Features of Incident Reports Page*

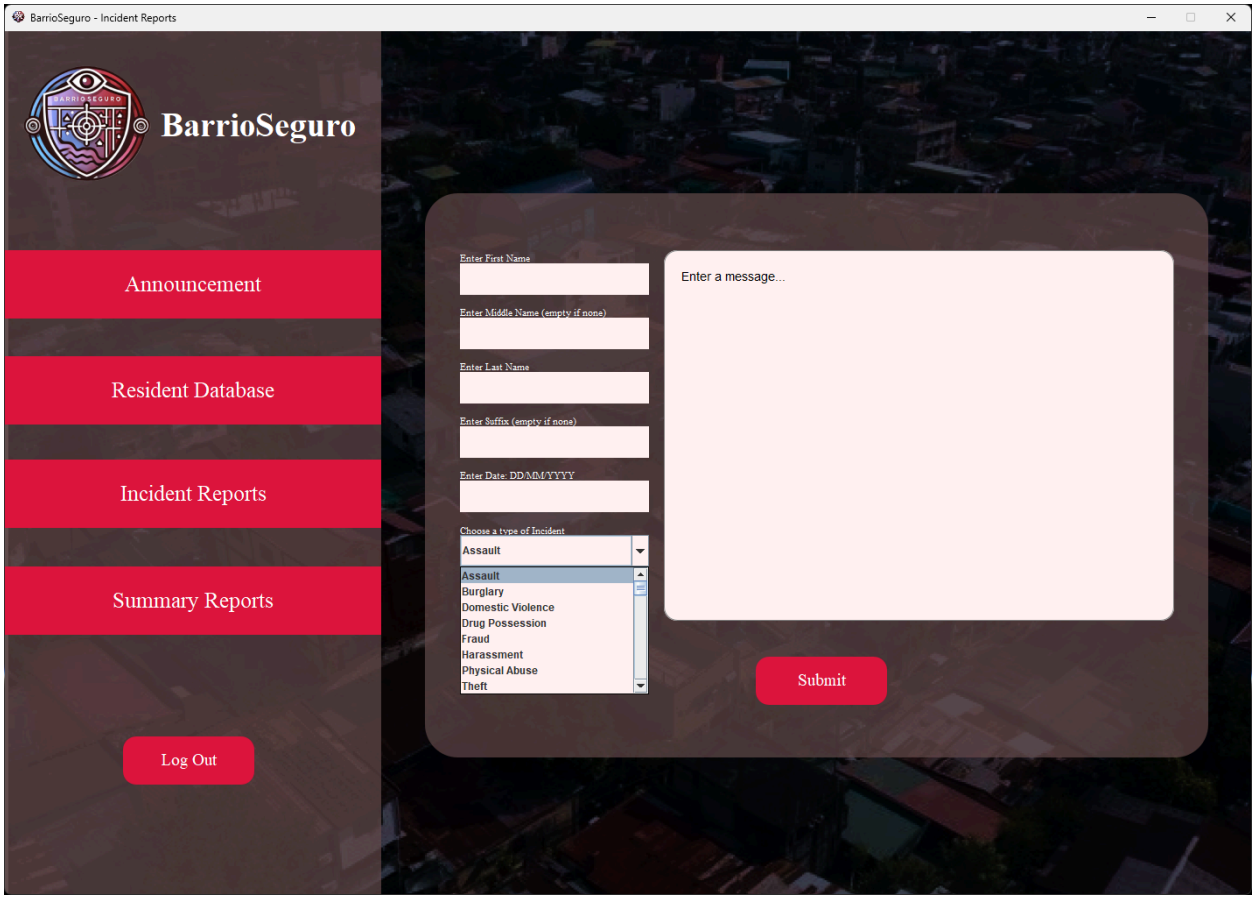


Figure 27: *Incident reports interface when selecting a specific type of incidents.*

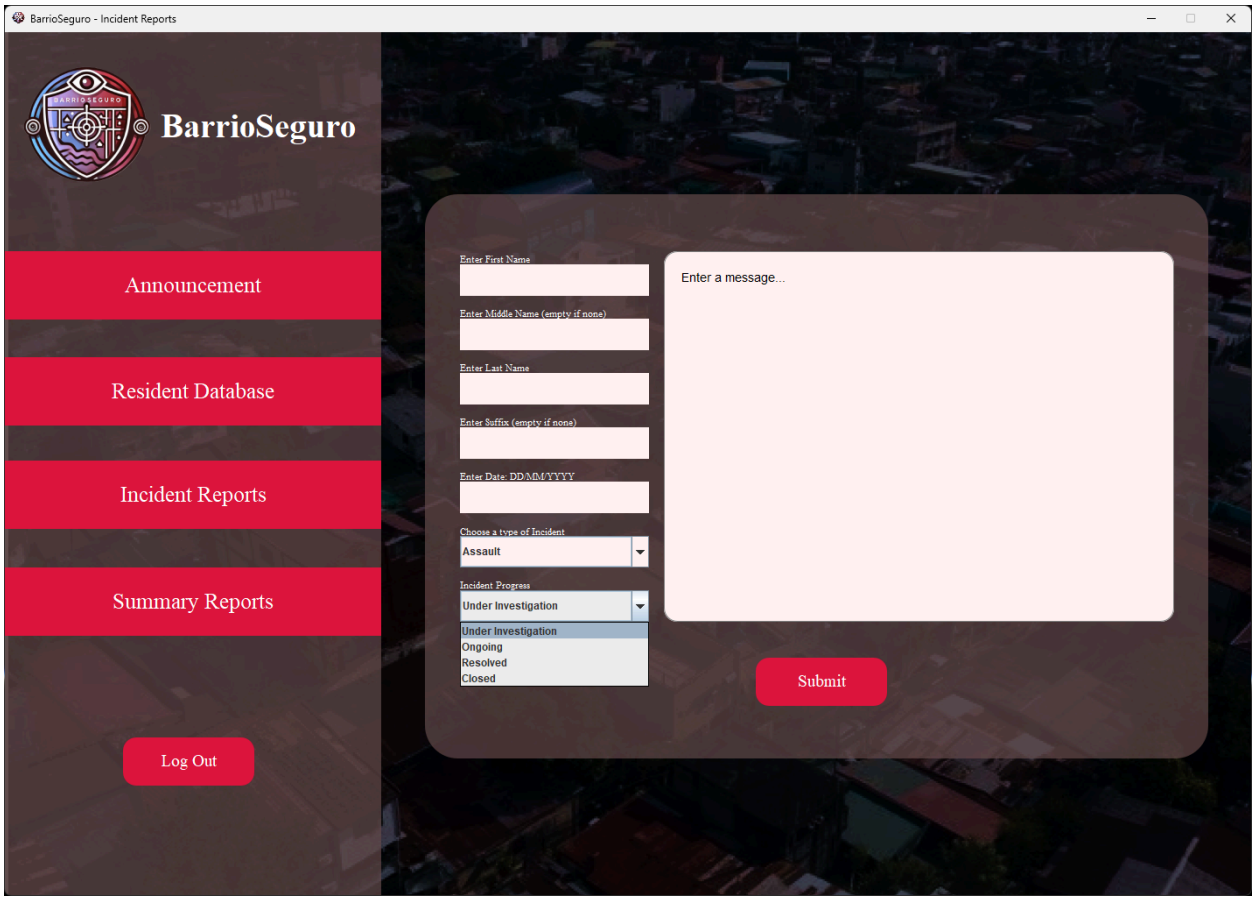


Figure 28: *Incident reports interface when selecting a specific type of incident report progress.*

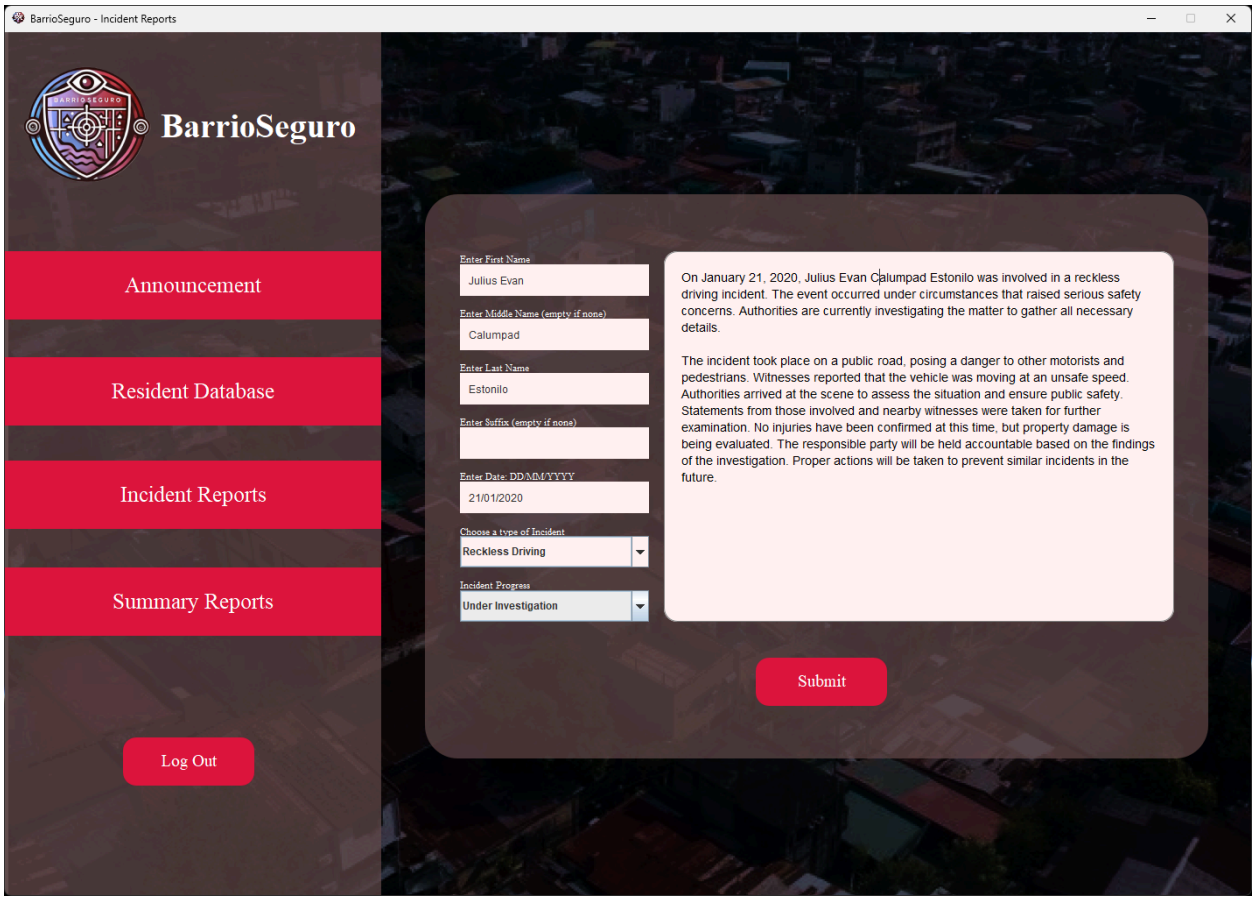


Figure 29: *Interface for entering other details such as name, date, and description of the incidents.*

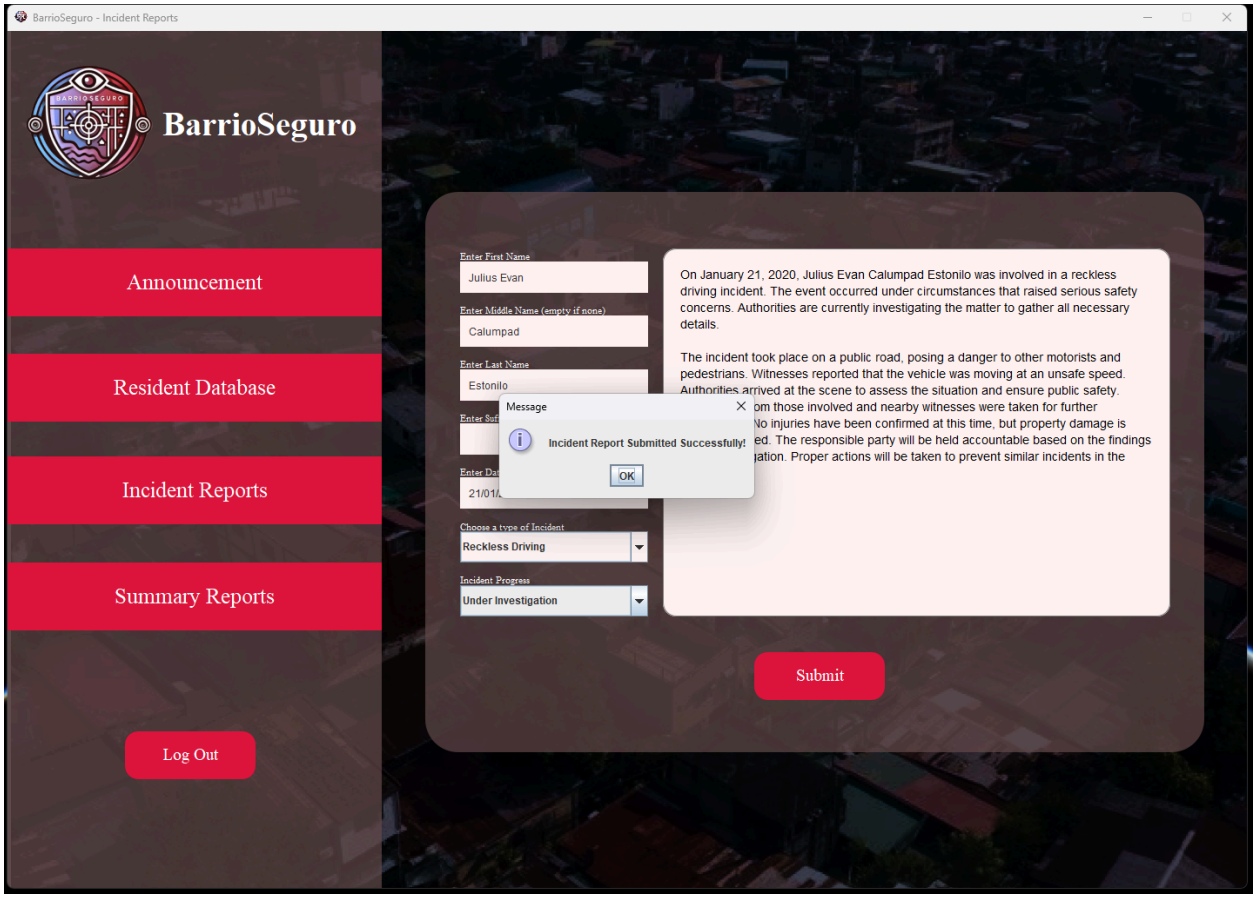


Figure 30: *Confirmation interface after clicking "Submit," indicating the incident report was added to the database.*

F. Data Reports Interface

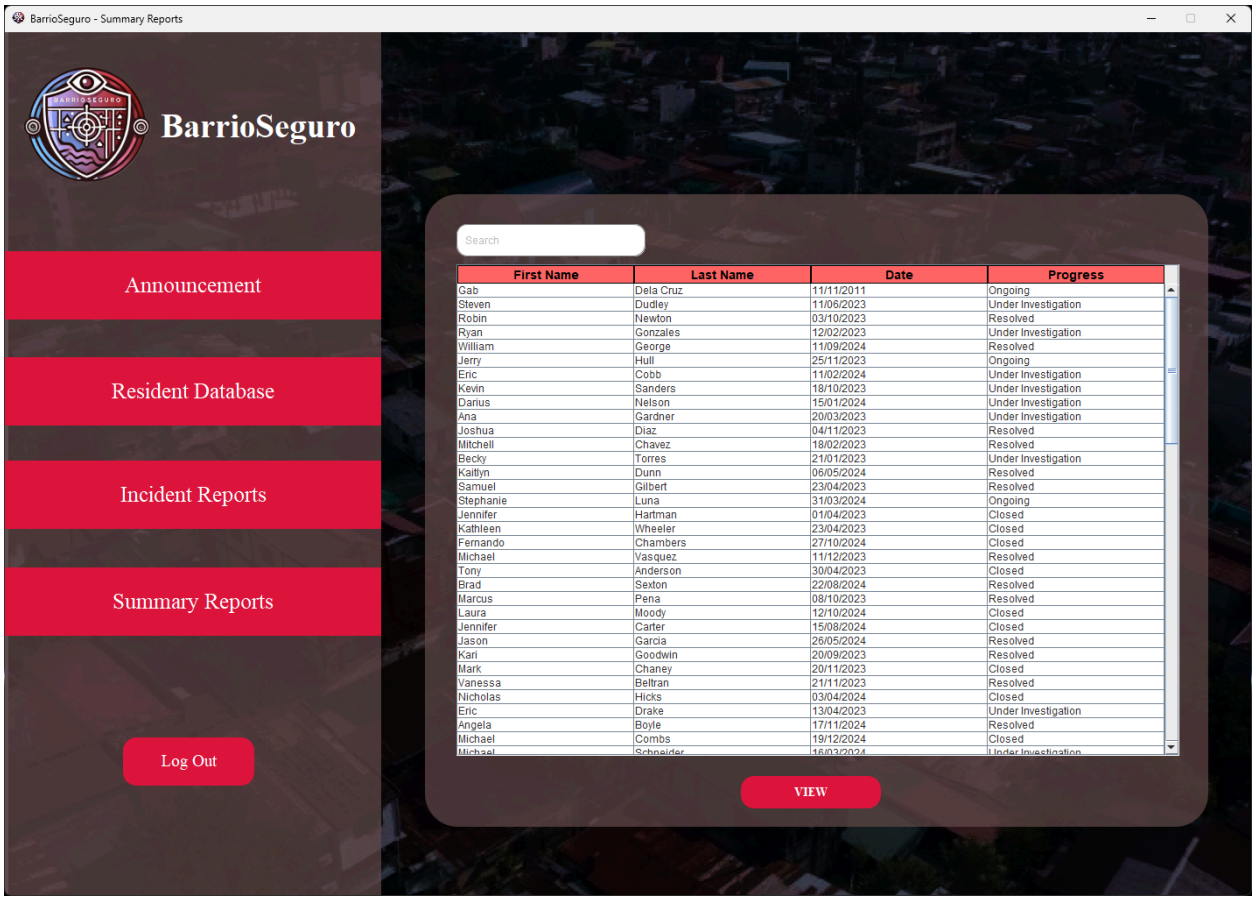


Figure 31: *Interface displaying summary reports.*

This page provides a concise summary of recorded incidents in the barangay. It displays organized data in a table format, including:

1	**The individual involved**	The individual associated with the incident. Display his/her first and last name.
2	**Incident Date**	Date of the incident.
3	**Incident Progress**	Type of Progress for the specific incident report
4	**View Button**	A button for selected row that allows administrators to access detailed information about a specific case.
5	***Search and Scroll Functionality***	Enhances ease of locating specific incident reports.

6	***Efficient Navigation***	A menu on the left allows easy access to other functionalities within the application.

Table 7: *Key Features of Summary Reports Page*

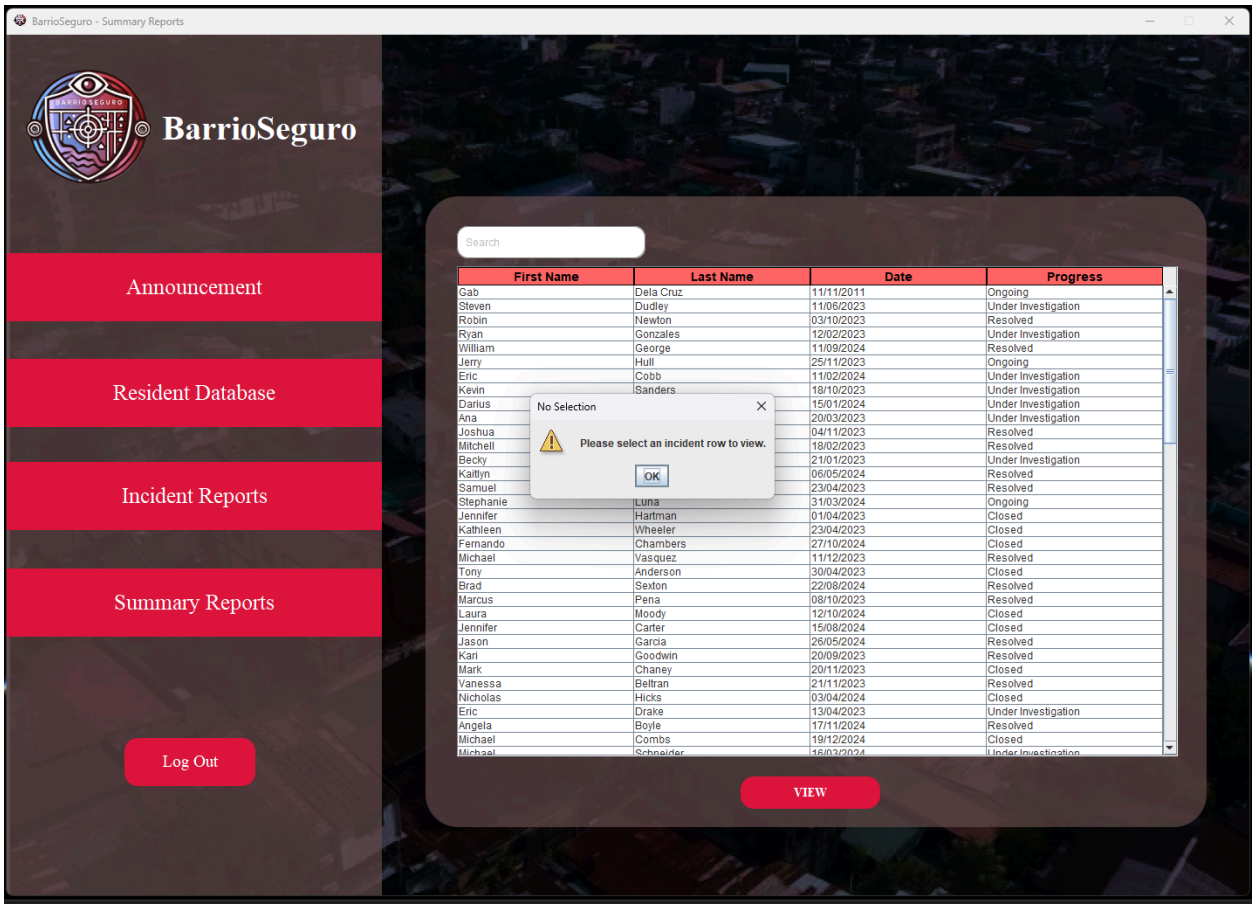


Figure 32: *Attempt to click "View" without selecting a resident report, showing a warning.*

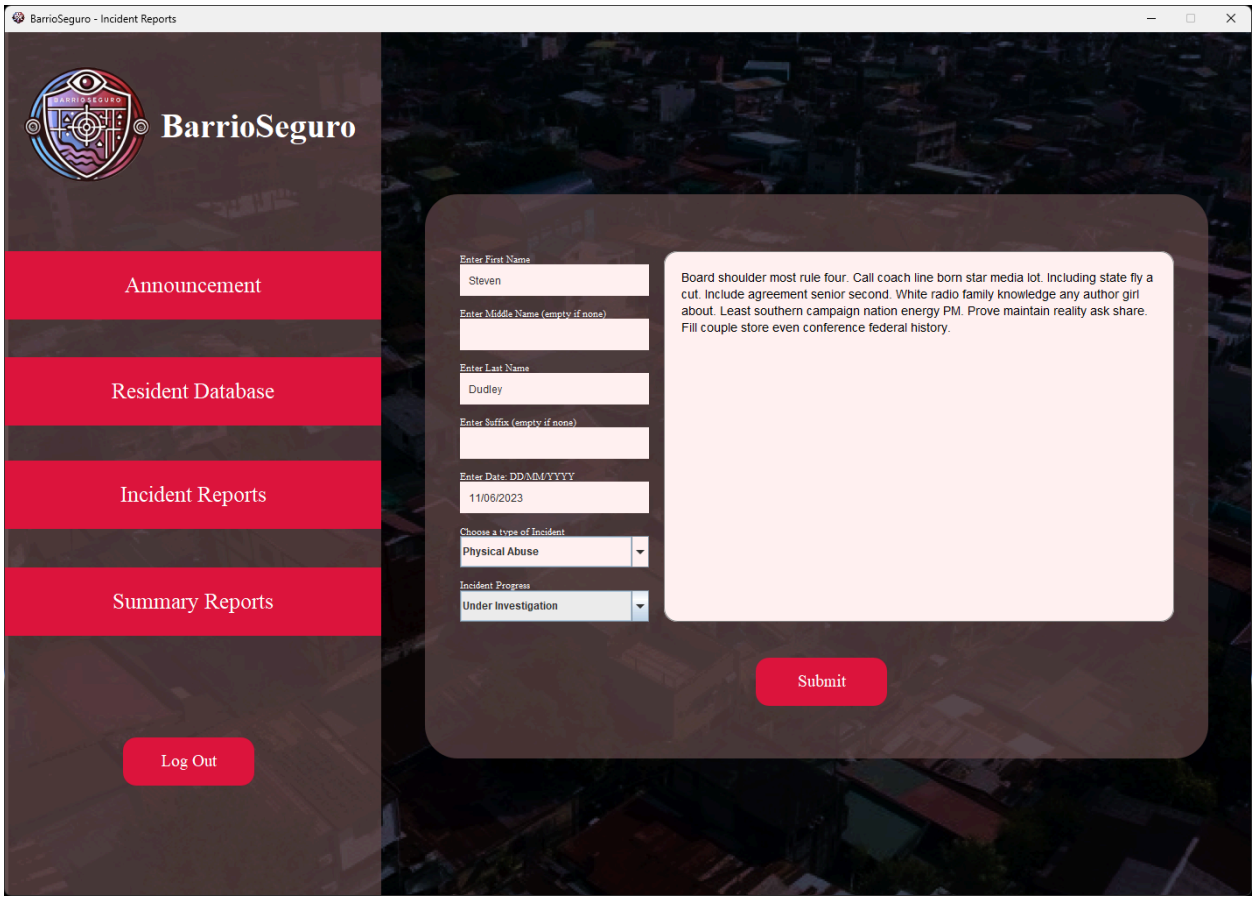


Figure 33: *Goes back to the incident reports interface when "View" is clicked with selected row incident report, showing a full report detail that is modifiable.*

This page is accessed when the “View” button is clicked on the Summary Reports Page with selected row incident report. This detailed view allows barangay officials to review and assess the reported incident for documentation or further action.

V) Program Codes

The program utilizes Java as its main programming language and is supported by a well-defined System Architecture (Figure 1). Microsoft Access serves as the database for vital information about barangay administrators, residents, and incident reports. In addition, an integration with an external API has been implemented to allow for the management of announcements, thus ensuring that all relevant data can be easily updated and shared within the community.

Each class in the program serves a specific purpose. The base form class serves as a template or structure for the following forms. The announcement form class is tasked with posting and retrieving announcements through the API. The resident form class handles resident profiles stored in our MS Access database, while the incident form class records important details related to local issues or emergencies. Lastly, the summary incident form class displays an overview of the compiled data from incident reports, thus improving the ability to analyze and understand community events.

By having a well-organized and efficient codebase, we allow for continuous maintenance and future development of the system. Our simple yet effective design ensures that people can access important resources, from timely updates on important announcements to complete records of incidents and resident information. We believe that this approach will continue to serve local communities in a timely and reliable manner.

Below are the screenshots of all our Java source codes:

1) BarrioSeguroApp.java [MAIN CLASS]

```java
// This statement groups our code in a folder-like structure, called a "package"
// Package means we are organizing files so they are easier to find
package system.BarrioSeguro;

// This line tells our program to use part of Java called "SwingUtilities"
// "SwingUtilities" is a helper that runs visual actions safely inside the program
import javax.swing.SwingUtilities;

// This says we are creating a class named "BarrioSeguroApp"
// A class is like a blueprint; it holds our code
public class BarrioSeguroApp {

    // This function is "main", the first thing the program runs
    // "main" is where the software starts working when we open it
    public static void main(String[] args) {

        // We use "SwingUtilities.invokeLater" to schedule our code for the "Event Dispatch Thread"
        // In simple words, it means: run the next part safely for the visual part of our program
        SwingUtilities.invokeLater(() -> {

            // We create a new object named "appController" from the "BarrioSeguro" class
            // "BarrioSeguro" is another part of our program that does the main tasks
            BarrioSeguro appController = new BarrioSeguro();

            // Here we call the "startApplication" method on our "appController"
            // This means: "Begin the program and show any interface or work we need"
            appController.startApplication();
        });
    }
}
```

2) BarrioSeguro.java

```java
// This line says we put this code in a "package" named "system.BarrioSeguro"
// Think of a package like a folder that keeps our code nice and organized
package system.BarrioSeguro;

// We declare a public class named "BarrioSeguro"
// A class is like a special box where we store related instructions
public class BarrioSeguro {

    // This is the constructor for the "BarrioSeguro" class
    // A constructor is a small intro that sets up our new "BarrioSeguro" object
    public BarrioSeguro() {
```

```
    }

    // This method is named "startApplication" and is marked 'protected'
    // It does the job of launching the program's main screen so we can start using it
    protected void startApplication() {
        // We create a "LoginForm" and pass "this" (the current class) to it
        // "LoginForm" is like the sign-in window that appears when the app starts
        LoginForm loginScreen = new LoginForm(BarrioSeguro.this);

        // We make the login screen visible so people can see and use it
        // 'setVisible(true)' just means: "Show this window on the screen now"
        loginScreen.setVisible(true);
    }

    // This method is named "openHomepageForm"
    // It's responsible for opening the homepage window after we log in
    protected void openHomepageForm() {
        // Creating a "HomepageForm" and passing our current "BarrioSeguro" to it
        // "HomepageForm" is the primary page people see after they sign in
        HomepageForm homepageScreen = new HomepageForm(BarrioSeguro.this);

        // Show the homepage window so people can use it
        homepageScreen.setVisible(true);
    }

    // This method is called "openAnnouncementForm"
    // It displays a place for neighborhood announcements or messages
    protected void openAnnouncementForm() {
        // "AnnouncementForm" is a separate window for posting news or updates
        AnnouncementForm announcementBoardScreen = new AnnouncementForm(BarrioSeguro.this);

        // Show the announcement window to share information
        announcementBoardScreen.setVisible(true);
    }

    // This method is called "openResidentForm"
    // It opens a page for managing people's information, like residents' names
    protected void openResidentForm() {
        // "ResidentForm" is a window where we keep track of all residents
        ResidentForm residentManagementScreen = new ResidentForm(BarrioSeguro.this);

        // Display the resident management window on the screen
        residentManagementScreen.setVisible(true);
    }

    // This method is named "openIncidentForm"
    // It's for reporting events or problems happening in our area
    protected void openIncidentForm() {
        // "IncidentForm" is a window to record details of things that went wrong
        IncidentForm incidentReportScreen = new IncidentForm(BarrioSeguro.this);

        // Show the incident report window so people can file reports
        incidentReportScreen.setVisible(true);
    }

    // This method is called "openSummaryForm"
    // It shows a summary of all incidents or other data
    protected void openSummaryForm() {
        // "SummaryForm" is where we gather and display an overview of information
        SummaryForm incidentSummaryScreen = new SummaryForm(BarrioSeguro.this);

        // Show the summary window so we can view a quick recap
        incidentSummaryScreen.setVisible(true);
    }
}
```

3) BaseForm.java

```java
// This line organizes our file under the "system.BarrioSeguro" package
// Think of a package like a folder name for better file grouping
package system.BarrioSeguro;

// These lines bring in tools from Java that we need
// Like color settings, fonts, pictures, and text fields
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Insets;
import java.awt.RenderingHints;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusAdapter;
import java.awt.event.FocusEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import java.io.File;

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

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

import javax.swing.border.EmptyBorder;

import javax.swing.plaf.basic.BasicButtonUI;

// "BaseForm" is an abstract class, extending "JFrame"
// A class is like a blueprint, and "JFrame" is a window on the screen
public abstract class BaseForm extends JFrame {

    // This is a "static final String" for the database path
    // It's a text label that doesn't change, telling us where our database is
    protected static final String DATABASE_PATH = "jdbc:ucanaccess://Database/BarrioSeguroDB.accdb";

    // This is a variable named "appController" of type "BarrioSeguro"
    // It lets us call functions from the "BarrioSeguro" class
    protected BarrioSeguro appController;

    // These are three 'Color' variables for default, click, and hover states
    // They help us make nice color changes when someone clicks a button
    protected final Color DEFAULT_COLOR = new Color(220, 20, 60);
    protected final Color CLICK_COLOR = new Color(180, 0, 40);
    protected final Color HOVER_COLOR = new Color(200, 0, 50);

    // This is the constructor for "BaseForm"
    // A constructor sets up the window, including its size and behavior
    public BaseForm(BarrioSeguro accessAppControl) {
        // We set our 'appController' to the one given from outside
        // So we can use it to navigate or open different screens
```

```
        appController = accessAppControl;

        // The window closes when you hit the exit button (X)
        // That means the program will stop if you click the close button
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Sets the window position and size: (100, 100) is the top-left corner,
        // 1440 wide x 1024 tall
        setBounds(100, 100, 1440, 1024);

        // We don't want the user to resize this window
        // So the window's size cannot be changed
        setResizable(false);

        // Puts the window in the center of the screen
        // This makes it easier to see
        setLocationRelativeTo(null);

        // Calls a separate function to set up our app's icon
        // The icon is basically the small picture on the top-left of the window
        setApplicationIcon();
    }

    // This method sets the window icon if the file exists
    // If not, we print an error message for debugging
    private void setApplicationIcon() {
        // Creates a File object pointing to "Visuals/logoIcon.png"
        // That's our icon image
        File iconFile = new File("Visuals/logoIcon.png");
        // Checks if the file is really there
        // If yes, we set it as our window icon
        if (iconFile.exists()) {
            ImageIcon appIcon = new ImageIcon(iconFile.getAbsolutePath());
            setIconImage(appIcon.getImage());
        } else {
            System.err.println("Error: Window icon not found at " + iconFile.getAbsolutePath());
        }
    }

    // This function gives us a database connection
    // "throws SQLException" means it might show an error if we can't connect
    protected Connection getConnection() throws SQLException {
        // Uses "DriverManager" to get a connection to our Access database
        // We must have that .accdb file for it to work
        return DriverManager.getConnection(DATABASE_PATH);
    }

    // Adds a background image to a Layered pane
    // A layered pane can stack items (like pictures and panels) on top of each other
    protected void addBackgroundImage(JLayeredPane mainPane) {
        // We load the background picture from "Visuals/backgroundImage.png"
        // "ImageIcon" holds the picture data
        ImageIcon backgroundImage = new ImageIcon("Visuals/backgroundImage.png");
        // We create a "JLabel" for showing that picture
        // Inside "paintComponent", we draw the image to fill the whole panel
        JLabel backgroundLabel = new JLabel() {
            @Override
            protected void paintComponent(Graphics paintGraphics) {
                super.paintComponent(paintGraphics);
                Graphics2D paintGraphicsWith2D = (Graphics2D) paintGraphics;
```

```java
            // Draw the image so it covers the label
            paintGraphicsWith2D.drawImage(backgroundImage.getImage(), 0, 0, getWidth(), getHeight(), null);

            // Draw a dark translucent rectangle on top
            // This gives a shaded effect
            paintGraphicsWith2D.setColor(new Color(0, 0, 0, 160));
            paintGraphicsWith2D.fillRect(0, 0, getWidth(), getHeight());
        }
    };

    // We place the label at (0,0) with size 1440 x 1024
    // That means it covers our entire window background
    backgroundLabel.setBounds(0, 0, 1440, 1024);

    // We add this label to the mainPane, behind everything else
    // "DEFAULT_LAYER" is basically the bottom layer
    mainPane.add(backgroundLabel, JLayeredPane.DEFAULT_LAYER);
}

// Adds a dashboard side panel to our main window
// This panel has buttons and a Logo so users can navigate
protected void addDashboardPanel(JLayeredPane mainPane) {
    // Creates a "JPanel" that's semi-transparent (with RGBA color)
    // The last value (178) is the alpha channel for transparency
    JPanel dashboardPanel = new JPanel();
    dashboardPanel.setBackground(new Color(102, 77, 77, 178));
    // We put this panel on the left side: 0,0 to 430 wide, 1024 tall
    // It spans from top to bottom
    dashboardPanel.setBounds(0, 0, 430, 1024);
    // We place it on a layer above the default background
    mainPane.add(dashboardPanel, JLayeredPane.PALETTE_LAYER);

    // We call separate methods to put items on the dashboard
    addLogoToDashboard(dashboardPanel);
    addTextToDashboard(dashboardPanel);
    addDashboardButtons(dashboardPanel);
}

// This method adds a Logo image onto the dashboard panel
// The Logo is just a small picture at the top
private void addLogoToDashboard(JPanel dashboardPanel) {
    // Load the "logoIcon.png" file for display
    // "ImageIcon" prepares the picture so we can show it
    ImageIcon logoImage = new ImageIcon("Visuals/logoIcon.png");

    // Create a "JLabel" that holds that icon
    JLabel logoLabel = new JLabel(logoImage);
    // Position the icon in the panel at (18, 33), with size 150x150
    logoLabel.setBounds(18, 33, 150, 150);
    // Use no layout manager so we can manually place items
    dashboardPanel.setLayout(null);
    // Add the label to the dashboard panel
    dashboardPanel.add(logoLabel);
}

// This method adds text to the dashboard panel, like the app's name
// "JLabel" shows words on the screen
private void addTextToDashboard(JPanel dashboardPanel) {
    // "BarrioSeguro" is the text we want to display
    // We set the color to white for visibility
```

```java
    JLabel logonamelabel = new JLabel("BarrioSeguro");
    logonamelabel.setForeground(Color.WHITE);
    // We set a big font style so people can notice it
    logonamelabel.setFont(new Font("Times New Roman", Font.BOLD, 39));
    // Position it in the panel
    logonamelabel.setBounds(178, 69, 237, 78);
    // Actually add it to our dashboard
    dashboardPanel.add(logonamelabel);
}

// This sets up all the buttons on the dashboard panel, like "Announcement" or "Log Out"
// We create each button, style it, and tell it what to do when clicked
private void addDashboardButtons(JPanel dashboardPanel) {
    // First button is called "announceBtn" with text "Announcement"
    // We use "styleButton" to make it look nice
    JButton announceBtn = new JButton("Announcement");
    styleButton(announceBtn);
    announceBtn.setBounds(0, 250, 430, 78);

    // This is what happens when we click "Announcement"
    announceBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent eventForAnnounceBtn) {
            // Move to a new screen called the Announcement Form
            // We do it with EventQueue.invokeLater to handle it safely
            EventQueue.invokeLater(new Runnable() {
                public void run() {
                    try {
                        appController.openAnnouncementForm();
                    } catch (Exception handleAnnounceException) {
                        handleAnnounceException.printStackTrace();
                    }
                    // Close this window after opening the next
                    dispose();
                }
            });
        }
    });
    // Finally, add this button to the dashboard panel
    dashboardPanel.add(announceBtn);


    // Next button: "Resident Database"
    JButton residentBtn = new JButton("Resident Database");
    styleButton(residentBtn);
    residentBtn.setBounds(0, 371, 430, 78);

    // When we click "Resident Database," open the Resident Form
    residentBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent eventForResidentBtn) {
            EventQueue.invokeLater(new Runnable() {
                public void run() {
                    try {
                        appController.openResidentForm();
                    } catch (Exception handleResidentException) {
                        handleResidentException.printStackTrace();
                    }
                    dispose();
                }
            });
        }
```

```
});
dashboardPanel.add(residentBtn);


// Another button: "Incident Reports"
JButton incidentBtn = new JButton("Incident Reports");
styleButton(incidentBtn);
incidentBtn.setBounds(0, 489, 430, 78);

// Clicking "Incident Reports" calls the method to open it
incidentBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent eventForIncidentBtn) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    appController.openIncidentForm();
                } catch (Exception handleIncidentException) {
                    handleIncidentException.printStackTrace();
                }
                dispose();
            }
        });
    }
});
dashboardPanel.add(incidentBtn);


// Button for "Summary Reports"
JButton summaryBtn = new JButton("Summary Reports");
styleButton(summaryBtn);

// Click "Summary Reports" to open that form
summaryBtn.setBounds(0, 611, 430, 78);
summaryBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent eventForSummaryBtn) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    appController.openSummaryForm();
                } catch (Exception handleSummaryException) {
                    handleSummaryException.printStackTrace();
                }
                dispose();
            }
        });
    }
});
dashboardPanel.add(summaryBtn);

// "Log Out" button for leaving the program
JButton logoutBtn = new JButton("Log Out");
styleRoundedButton(logoutBtn);
logoutBtn.setBounds(135, 805, 150, 55);

// When clicked, close this form and go to the login screen
logoutBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent eventForSummaryBtn) {
        dispose();

        EventQueue.invokeLater(() -> {
```

```
                logoutAndGoToLogin();
            });
        }
    });
    dashboardPanel.add(logoutBtn);
}

// This method changes how a normal button looks and behaves
// We set colors, fonts, and events for pressing, releasing, and hovering
private void styleButton(JButton button) {
    button.setFont(new Font("Times New Roman", Font.PLAIN, 25));
    button.setBorderPainted(false);
    button.setFocusPainted(false);
    button.setContentAreaFilled(false);
    button.setOpaque(true);
    button.setForeground(Color.WHITE);
    button.setBackground(DEFAULT_COLOR);

    // We use a "MouseAdapter" so we can respond to mouse interactions
    button.addMouseListener(new MouseAdapter() {
        @Override
        public void mousePressed(MouseEvent eventButtonPress) {
            // Change color when you press the button
            button.setBackground(CLICK_COLOR);
        }

        @Override
        public void mouseReleased(MouseEvent eventButtonRelease) {
             // Reset color when you lift the mouse button
            button.setBackground(DEFAULT_COLOR);
        }

        @Override
        public void mouseEntered(MouseEvent eventButtonEnter) {
            // Slightly different color when you hover over it
            button.setBackground(HOVER_COLOR);
        }

        @Override
        public void mouseExited(MouseEvent eventButtonExit) {
             // Go back to the usual color if your mouse leaves
            button.setBackground(DEFAULT_COLOR);
        }
    });
}

// Styles a "rounded button" using the same base style, with curved edges
// Rounded corners are done by painting a round rectangle in the background
protected void styleRoundedButton(JButton button) {
    // First, apply the normal styling
    styleButton(button);
    button.setFont(new Font("Times New Roman", Font.PLAIN, 20));
    // Make the button see-through for the round effect
    button.setOpaque(false);
    button.setContentAreaFilled(false);
    button.setBorderPainted(false);
    button.setFocusPainted(false);

    button.addMouseListener(new MouseAdapter() {
        @Override
```

```
        public void mousePressed(MouseEvent eventButtonPress) {
            button.setBackground(CLICK_COLOR);
        }

        @Override
        public void mouseReleased(MouseEvent eventButtonRelease) {
            button.setBackground(DEFAULT_COLOR);
        }
    });

    // Change how the button is drawn (the UI) to get those rounded edges
    button.setUI(new BasicButtonUI() {
        @Override
        public void paint(Graphics paintGraphics, JComponent paintComponent) {
            Graphics2D paintGraphics2D = (Graphics2D) paintGraphics.create();
            // Turn on smoother edges
            paintGraphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
            // Use the button's current background color
            paintGraphics2D.setColor(button.getBackground());
            // Paint a round rectangle in the entire shape of the component
            paintGraphics2D.fillRoundRect(0, 0, paintComponent.getWidth(), paintComponent.getHeight(), 30, 30);
            paintGraphics2D.dispose();
            // Call the usual painting so the text shows up
            super.paint(paintGraphics, paintComponent);
        }
    });
}

// A small function to log out and show the login screen again
// It closes the current window and creates a new "LoginForm"
private void logoutAndGoToLogin() {
    // Stop showing this window
    dispose();
    // Create a new "LoginForm" using our controller and show it
    LoginForm login = new LoginForm(appController);
    login.setVisible(true);
}

// Creates a text field with rounded corners and optional initial text
// Good for user input boxes like name or address
protected JTextField createRoundedTextField(String text, int cornerRadius) {
    JTextField givenTextField = new JTextField(text) {
        // "Override" the paint methods so we can draw the curved shape
        @Override
        protected void paintComponent(Graphics paintGraphics) {
            Graphics2D paintGraphics2D = (Graphics2D) paintGraphics.create();
            paintGraphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
            // Fill with white color inside the rectangle
            paintGraphics2D.setColor(Color.WHITE);
            paintGraphics2D.fillRoundRect(0, 0, getWidth(), getHeight(), cornerRadius, cornerRadius);
            // Call the normal painting to show text
            super.paintComponent(paintGraphics);
            paintGraphics2D.dispose();
        }

        @Override
        protected void paintBorder(Graphics paintGraphics) {
            // Draw a light-gray border around the curved rectangle
            Graphics2D paintGraphics2D = (Graphics2D) paintGraphics.create();
            paintGraphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
```

```
            paintGraphics2D.setColor(Color.GRAY);
            paintGraphics2D.drawRoundRect(0, 0, getWidth() - 1, getHeight() - 1, cornerRadius, cornerRadius);
            paintGraphics2D.dispose();
        }
    };
    // Make the text field see-through, except the white shape we draw
    givenTextField.setOpaque(false);
    // Add extra space inside the text field
    givenTextField.setBorder(new EmptyBorder(10, 20, 10, 20));
    // Finally, return the completed text field so we can use it
    return givenTextField;
}

// Creates a rounded text area for typing multiple lines of text
// Like writing longer comments or messages
public JTextArea createRoundedTextArea(String placeholder, int width, int height) {
    // We override painting again for the shape
    JTextArea givenTextArea = new JTextArea(placeholder) {
        @Override
        protected void paintComponent(Graphics paintGraphics) {
            Graphics2D paintGraphics2D = (Graphics2D) paintGraphics.create();
            paintGraphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
            // Fill with the background color (light shade)
            paintGraphics2D.setColor(getBackground());
            paintGraphics2D.fillRoundRect(0, 0, getWidth(), getHeight(), 30, 30);
            paintGraphics2D.dispose();
            // Then do the normal text painting
            super.paintComponent(paintGraphics);
        }

        @Override
        protected void paintBorder(Graphics paintGraphics) {
            // Draw a gray border around the curved rectangle
            Graphics2D paintGraphics2D = (Graphics2D) paintGraphics.create();
            paintGraphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
            paintGraphics2D.setColor(Color.GRAY);
            paintGraphics2D.drawRoundRect(0, 0, getWidth() - 1, getHeight() - 1, 30, 30);
            paintGraphics2D.dispose();
        }
    };

    // Make it see-through so the round shape is shown
    givenTextArea.setOpaque(false);
    // Light pinkish background to differentiate
    givenTextArea.setBackground(new Color(255, 244, 244));
    // The text color is soft
    givenTextArea.setForeground(new Color(0,0,0,50));
    // The font is "SansSerif" size 14
    givenTextArea.setFont(new Font("SansSerif", Font.PLAIN, 14));
    // Add a border so text has some padding on all sides
    givenTextArea.setBorder(new EmptyBorder(20, 20, 20, 20));
    // Set the location and size in our Layout
    givenTextArea.setBounds(51, 136, width, height);
    // Wrap words properly to the next line (no horizontal scrolling)
    givenTextArea.setWrapStyleWord(true);
    // Automatically move words to next line
    givenTextArea.setLineWrap(true);
    // Extra margin around the text
    givenTextArea.setMargin(new Insets(20, 20, 20, 20));

```

```
        // Listen for when people click inside or leave the text area
        givenTextArea.addFocusListener(new FocusAdapter() {
            @Override
            public void focusGained(FocusEvent eventFocusTextArea) {
                // If our text area still has the placeholder text, clear it
                if (givenTextArea.getText().equals(placeholder)) {
                    givenTextArea.setText("");
                    givenTextArea.setForeground(Color.BLACK);
                }
            }

            @Override
            public void focusLost(FocusEvent eventFocusTextArea) {
                // If the user left it empty, bring back the placeholder
                if (givenTextArea.getText().isEmpty()) {
                    givenTextArea.setText(placeholder);
                    givenTextArea.setForeground(Color.LIGHT_GRAY);
                }
            }
        });

        // Return the final ready-to-use text area
        return givenTextArea;
    }
}
```

4) LoginForm.java

```
// We're declaring our file under "system.BarrioSeguro"
// A package is basically a folder that groups similar code together
package system.BarrioSeguro;

// These lines give us the tools we need, like colors, fonts, and images
// We also import classes for handling actions, layouts, and database connections
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```java

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

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;

import javax.swing.border.EmptyBorder;

// "LoginForm" is a public class that extends "BaseForm"
// This means "LoginForm" is a special type of window with extra functions built on "BaseForm"
public class LoginForm extends BaseForm {

    // Here is our constructor named "LoginForm" which takes a "BarrioSeguro" object
    // This sets up the basic window settings and calls "initialize"
    public LoginForm(BarrioSeguro appController) {
        // Tells "BaseForm" to set up the window and pass in our app logic
        super(appController);
        // Title text at the top of the window
        setTitle("BarrioSeguro - Login");
        // Calls a separate method to build and add the various elements like panels
        initialize();
    }

    // "initialize" is a private method, meaning it's only used here
    // It sets up the main layered pane and calls code to add the background and login controls
    private void initialize() {
        // "JLayeredPane" lets us stack panels or images
        // We assign it to our form as the main area
        JLayeredPane loginPane = new JLayeredPane();
        setContentPane(loginPane);

        // Use a function from "BaseForm" to add the background image
        // This puts a picture and a shading layer behind everything
        addBackgroundImage(loginPane);

        // We create a "JPanel" that holds the login controls and add it on top
        JPanel loginPanel = createLoginPanel();
        loginPane.add(loginPanel, JLayeredPane.PALETTE_LAYER);
    }

    // We build a "JPanel" called "createLoginPanel"
    // This panel holds the logo, the title, and the admin section
    private JPanel createLoginPanel() {
        // Make a new panel, size it, and give it a transparent color so we can see the background
        JPanel loginPanel = new JPanel();
        loginPanel.setBounds(405, 135, 685, 814);
        loginPanel.setBackground(new Color(0, 0, 0, 0)); // Transparent background
        loginPanel.setLayout(null);

        // Puts the logo image on the panel's top
        addLogoImage(loginPanel);
        // Adds the big "BarrioSeguro" text
```

```java
        addTextLabel(loginPanel);

        // This panel is for admin login controls, like the username or ID text field
        JPanel adminPanel = createAdminPanel();
        loginPanel.add(adminPanel);

        // Return this panel so we can place it in the layered pane
        return loginPanel;
    }

    // Adds a small icon or picture on the loginPanel
    // Typically a brand or app icon
    private void addLogoImage(JPanel loginPanel) {
        // "ImageIcon" loads the picture file called "logoIcon.png"
        ImageIcon logoImage = new ImageIcon("Visuals/logoIcon.png");
        // We create a JLabel that actually shows that icon
        JLabel imageLabel = new JLabel(logoImage);
        // Position the label at (120, 30) with a 151x151 size
        imageLabel.setBounds(120, 30, 151, 151);
        // Finally, add this label onto the panel
        loginPanel.add(imageLabel);
    }

    // Adds the "BarrioSeguro" text to our login panel
    private void addTextLabel(JPanel loginPanel) {
         // A simple label with white text
        JLabel textLabel = new JLabel("BarrioSeguro");
        textLabel.setForeground(Color.WHITE);
        // Pick a large, bold Times New Roman font, size 45
        textLabel.setFont(new Font("Times New Roman", Font.BOLD, 45));
        // Put it at (280, 61) in the panel with 265x88 size
        textLabel.setBounds(280, 61, 265, 88);
        // Place it on the login panel
        loginPanel.add(textLabel);
    }

    // This private method creates a panel for administrator login
    // It has a background color and a text field for entering an ID number
    @SuppressWarnings("unused")
    private JPanel createAdminPanel() {
        // We override "paintComponent" to give this panel rounded corners or special style
        JPanel adminPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics paintGraphics) {
                super.paintComponent(paintGraphics);
                Graphics2D paintGraphicsWith2D = (Graphics2D) paintGraphics;
                // Make edges smooth
                paintGraphicsWith2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
                // Paint the panel's background color in a rectangle with rounded corners
                paintGraphicsWith2D.setColor(getBackground());
                paintGraphicsWith2D.fillRoundRect(0, 0, getWidth(), getHeight(), 75, 75); // Rounded corners
            }
        };

        // Sets size and color for the panel (with a semitransparent dark shade)
        adminPanel.setBounds(61, 196, 561, 540);
        adminPanel.setBackground(new Color(102, 77, 77, 178)); // Semi-transparent dark background
        adminPanel.setLayout(null);
        adminPanel.setOpaque(false);
```

```java
    // A heading inside the panel that says "ADMINISTRATOR"
    JLabel adminTextLabel = new JLabel("ADMINISTRATOR");
    adminTextLabel.setForeground(Color.WHITE);
    adminTextLabel.setFont(new Font("Times New Roman", Font.BOLD, 41));
    adminTextLabel.setBounds(100, 24, 375, 88);
    adminPanel.add(adminTextLabel);

    // Add a small label telling the user to "Enter ID Number"
    JLabel idNumberTextLabel = new JLabel("Enter ID Number");
    idNumberTextLabel.setForeground(Color.WHITE);
    idNumberTextLabel.setFont(new Font("Times New Roman", Font.PLAIN, 20));
    idNumberTextLabel.setBounds(36, 201, 160, 26);
    adminPanel.add(idNumberTextLabel);

    // A "JPasswordField" to keep the ID secret (like a password)
    // This is where the user types their ID
    JPasswordField idNumberTextBox = new JPasswordField();
    idNumberTextBox.setFont(new Font("SansSerif", Font.PLAIN, 30));
    idNumberTextBox.setBounds(36, 234, 486, 72);
    // Add an empty border inside the text box for spacing
    idNumberTextBox.setBorder(new EmptyBorder(10, 20, 10, 20));
    adminPanel.add(idNumberTextBox);

    // We create a "Log In" button and give it rounded style
    JButton loginButton = new JButton("Log In");
    styleRoundedButton(loginButton); // Custom styling for button
    loginButton.setFont(new Font("Times New Roman", Font.PLAIN, 25));
    loginButton.setBounds(184, 346, 193, 53);

    adminPanel.add(loginButton);

    // When the user clicks "Log In," check if the ID is correct
    loginButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent eventForLogIn) {
            // If "isValidID" is true, we close this form and open the homepage
            if (isValidID(new String(idNumberTextBox.getPassword()))) {
                dispose(); // Close login form
                appController.openHomepageForm();
            } else {
                // If it's not a valid ID, show an error box
                JOptionPane.showMessageDialog(LoginForm.this, "Invalid ID Number","Error",JOptionPane.ERROR_MESSAGE);
            }
        }
    });

    // Finally, return the panel so we can add it to the login screen
    return adminPanel;
}

// This method checks if the typed ID exists in the database table named "Barangay_Official"
private boolean isValidID(String idNumber) {
    // A text command that counts how many times this ID number appears
    String query = "SELECT COUNT(*) FROM Barangay_Official WHERE idnumber = ?";

    // "try" ensures we close our connection each time
    // "Connection" is how we talk to the database
    try (Connection connectIDnumber = getConnection()) {
        // Prepare the SQL statement so we can safely set the ID
        try (PreparedStatement statementIDnumber = connectIDnumber.prepareStatement(query)) {
```

```
                // Replace the question mark with the actual ID
                statementIDnumber.setString(1, idNumber);

                // Execute the query and get results
                ResultSet resultIDnumber = statementIDnumber.executeQuery();
                resultIDnumber.next();
                // Get the count from the first column. If more than 0, ID is valid
                int count = resultIDnumber.getInt(1);
                return count > 0; // Returns true if ID exists
            }
        } catch (SQLException handleDatabaseException) {
            // If something is wrong, we print the error and show a message box
            handleDatabaseException.printStackTrace();
            JOptionPane.showMessageDialog(LoginForm.this, "Database error: " + handleDatabaseException.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }

        // If all else fails, we return false (invalid ID)
        return false;
    }
}
```

5) HomepageForm.java

```
// This line says our file is part of a package called "system.BarrioSeguro"
// Think of a package as a folder or grouping for related classes
package system.BarrioSeguro;

// These lines let us use Java's color, font, graphics features, images, labels,
// layered panes, panels, and alignment options
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;

import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

// "HomepageForm" is a class that extends "BaseForm"
// "extends" means "HomepageForm" gets tools and abilities from "BaseForm"
public class HomepageForm extends BaseForm {

    // This is the constructor: it runs when we make a new "HomepageForm" object
    // It calls the parent class constructor and sets the page title, then calls "initialize"
    public HomepageForm(BarrioSeguro appController) {
        super(appController);
        setTitle("BarrioSeguro - Homepage");
        initialize();
    }

    // This private method sets up everything on the homepage
    // It makes a layered area and adds a background, dashboard, and welcome panel
    private void initialize() {
        // Create a new layered pane to hold our components
```

```java
        // Layered panes let us stack panels on top of the background
        JLayeredPane homepagePane = new JLayeredPane();
        // Tells Java that our main window content is this layered pane
        setContentPane(homepagePane);

        // Use a method from "BaseForm" to show the background image
        addBackgroundImage(homepagePane);
        // Also from "BaseForm," adding the left-hand dashboard panel
        addDashboardPanel(homepagePane);
        // A custom panel that says "Welcome!" and shows a big image
        addWelcomePanel(homepagePane);
    }

    // This method creates and adds a panel that says "Welcome!" and shows an app image
    // It also has a label for "Barangay Monitoring and Incident Reporting System"
    private void addWelcomePanel(JLayeredPane homepagePane) {
        // Make a JPanel with a custom shape (rounded corners) using "paintComponent"
        JPanel welcomePanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics paintGraphics) {
                // Calls the usual painting code
                super.paintComponent(paintGraphics);
                Graphics2D paintGraphicsWith2D = (Graphics2D) paintGraphics;
                // Make graphics edges smoother
                paintGraphicsWith2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
                // Fill the panel with its background color
                paintGraphicsWith2D.setColor(getBackground());
                // Draw a round rectangle matching our panel's size
                paintGraphicsWith2D.fillRoundRect(0, 0, getWidth(), getHeight(), 75, 75);
            }
        };
        // Place the panel on the screen at x=480, y=185; width=895, height=722
        welcomePanel.setBounds(480, 185, 895, 722);
        // Give it a semi-transparent dark background color (R=102, G=77, B=77, Alpha=178)
        welcomePanel.setBackground(new Color(102, 77, 77, 178));
        // Use no automatic layout manager, so we can position stuff ourselves
        welcomePanel.setLayout(null);
        // Make sure we can see the panel's transparency
        welcomePanel.setOpaque(false);

        // Add the panel on top of other layers
        // "PALETTE_LAYER" means it's in front of the background
        homepagePane.add(welcomePanel, JLayeredPane.PALETTE_LAYER);

        // This label says "Welcome!" in the center, Large white text
        JLabel lblWelcome = new JLabel("Welcome!");
        lblWelcome.setHorizontalAlignment(SwingConstants.CENTER);
        lblWelcome.setForeground(Color.WHITE);
        lblWelcome.setFont(new Font("Times New Roman", Font.BOLD, 47));
        // Position the label at (305, 25) with width=272 and height=106
        lblWelcome.setBounds(305, 25, 272, 106);
        // Add it onto the welcomePanel
        welcomePanel.add(lblWelcome);

        // A label to hold an image from "databaseLogoIcon.png"
        JLabel logoLabel = new JLabel(new ImageIcon("Visuals/databaseLogoIcon.png"));
        // Position this image in the center area of the panel
        logoLabel.setBounds(166, 123, 551, 529);
        // Add the logo to the welcomePanel
        welcomePanel.add(logoLabel);

        // Another label that spells out the full system name for clarity
        JLabel lblBrgyMonitorIncidentReport = new JLabel("Barangay Monitoring and Incident Reporting System");
        // Align its text in the center
        lblBrgyMonitorIncidentReport.setHorizontalAlignment(SwingConstants.CENTER);
        // Make the text color white
        lblBrgyMonitorIncidentReport.setForeground(Color.WHITE);
        // Use Times New Roman, bold, size 27
        lblBrgyMonitorIncidentReport.setFont(new Font("Times New Roman", Font.BOLD, 27));
        // Position the label near the bottom of the panel
        lblBrgyMonitorIncidentReport.setBounds(73, 619, 752, 92);
        // Add it to the panel
        welcomePanel.add(lblBrgyMonitorIncidentReport);
    }
}
```

6) AnnouncementForm.java

```java
// This line places our file inside the "system.BarrioSeguro" package
// A package is like a folder name to keep related classes together
package system.BarrioSeguro;

// These lines import the tools and libraries we need
// They include colors, fonts, event listeners, database connections, and email utilities
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

import java.util.Properties;

import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;

import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

import javax.swing.JButton;
import javax.swing.JLayeredPane;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

import javax.swing.border.EmptyBorder;

// "AnnouncementForm" is a class that extends "BaseForm"
// "extends" means our class has the functions and properties from "BaseForm"
public class AnnouncementForm extends BaseForm {

    // We declare two input fields: one "JTextField" for the subject, one "JTextArea" for the message
    private JTextField subjectTextField;
    private JTextArea messageTextField;

    // This is the constructor for "AnnouncementForm"
    // It sets the window title and calls the "initialize()" method
    public AnnouncementForm(BarrioSeguro appController) {
        // Gives the parent class (BaseForm) our application controller
        super(appController);
        setTitle("BarrioSeguro - Announcements");
        initialize();
    }

    // "initialize" sets up everything on the form: background, panels, etc.
    private void initialize() {
        // Create a layered pane, which lets us stack panels and background images
        JLayeredPane announcePane = new JLayeredPane();
        // Make this pane the main content area
```

```
        setContentPane(announcePane);

        // Adds a background image behind all other panels
        addBackgroundImage(announcePane);
        // Adds a side panel (dashboard) for navigation
        addDashboardPanel(announcePane);
        // Adds our main announcement panel for subject and message
        addAnnouncePanel(announcePane);
    }

    // "addAnnouncePanel" makes a special panel for writing announcements
    // It uses a rounded rectangle and some transparent color
    private void addAnnouncePanel(JLayeredPane announcePane) {
        // Create a JPanel that draws rounded corners in "paintComponent"
        JPanel announcePanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics paintGraphics) {
                super.paintComponent(paintGraphics);
                Graphics2D paintGraphicsWith2D = (Graphics2D) paintGraphics;
                // Turn on smoother edges
                paintGraphicsWith2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
                // Fill this area with the panel's background color
                paintGraphicsWith2D.setColor(getBackground());
                paintGraphicsWith2D.fillRoundRect(0, 0, getWidth(), getHeight(), 75, 75);
            }
        };
        // Position the panel and give it a semi-transparent brownish background
        announcePanel.setBounds(480, 185, 895, 640);
        announcePanel.setBackground(new Color(102, 77, 77, 178));  // Semi-transparent background
        announcePanel.setLayout(null);
        announcePanel.setOpaque(false);

        // Place this panel in a higher layer than the background image
        announcePane.add(announcePanel, JLayeredPane.PALETTE_LAYER);

        // Now we build our subject text field
        subjectTextField = createRoundedTextField("Subject", 25);
        // "setToolTipText" can show a small note if the user hovers over the field
        subjectTextField.setToolTipText("");
        subjectTextField.setHorizontalAlignment(SwingConstants.LEFT);
        subjectTextField.setForeground(new Color(0,0,0,50));
        subjectTextField.setBackground(new Color(255, 244, 244));
        subjectTextField.setFont(new Font("SansSerif", Font.PLAIN, 25));
        subjectTextField.setBorder(new EmptyBorder(10, 20, 10, 20));
        subjectTextField.setBounds(51, 50, 798, 53);

        // We add "FocusListener" so that if the text says "Subject," it goes away when clicked
        subjectTextField.addFocusListener(new FocusListener() {
            @Override
            public void focusGained(FocusEvent eventForSubjectField) {
                 // If it's still the default text, empty it and switch the color to black
                if (subjectTextField.getText().equals("Subject")) {
                    subjectTextField.setText("");
                    subjectTextField.setForeground(Color.BLACK);
                }
            }

            @Override
            public void focusLost(FocusEvent eventForSubjectField) {
                // If the user left the field empty, return the placeholder text
```

```
            if (subjectTextField.getText().isEmpty()) {
                subjectTextField.setText("Subject");
                subjectTextField.setForeground(Color.LIGHT_GRAY);
            }
        }
    });

    // Add this subject field to our panel
    announcePanel.add(subjectTextField);

    // Build a text area for the message part, using a helper method
    messageTextField = createRoundedTextArea("Enter a message...", 798, 323);
    // Position it below the subject
    announcePanel.add(messageTextField);

     // Make a button for sending emails
    JButton emailBtn = new JButton("Send to Email");
    // Use our style for rounded buttons from "BaseForm"
    styleRoundedButton(emailBtn);
    // When clicked, check the fields, and if valid, send the email
    emailBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent eventForEmailBtn) {
            if (validateFields()) {
                String subject = subjectTextField.getText();
                String messageContent = messageTextField.getText();
                // Actually send the email with the typed subject and message
                sendEmail(subject, messageContent);
            }
        }
    });
    // Position the button on the panel
    emailBtn.setBounds(555, 497, 177, 55);
    announcePanel.add(emailBtn);

    // Make a button that says "Send to SMS" (not fully implemented, just a placeholder)
    JButton smsBtn = new JButton("Send to SMS");
    styleRoundedButton(smsBtn);
    smsBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent eventForSMSbtn) {
            // If our subject and message are okay, attempt an SMS send
            if (validateFields()) {
                String subject = subjectTextField.getText();
                String messageContent = messageTextField.getText();
                sendSMS(subject, messageContent);
            }
        }
    });
    smsBtn.setBounds(167, 497, 177, 55);
    announcePanel.add(smsBtn);
}

// "validateFields" checks if "Subject" and "Message" fields are not empty
// If they are empty, we show an error and return false; otherwise, return true
private boolean validateFields() {
    // Ensures fields are not empty before sending email or SMS
    if (subjectTextField.getText().equals("Subject") || subjectTextField.getText().trim().isEmpty()) {
        JOptionPane.showMessageDialog(AnnouncementForm.this, "Subject is required.", "Validation Error", JOptionPane.ERROR_MESSAGE);
        return false;
    }
    if (messageTextField.getText().equals("Enter a message...") || messageTextField.getText().trim().isEmpty()) {
```

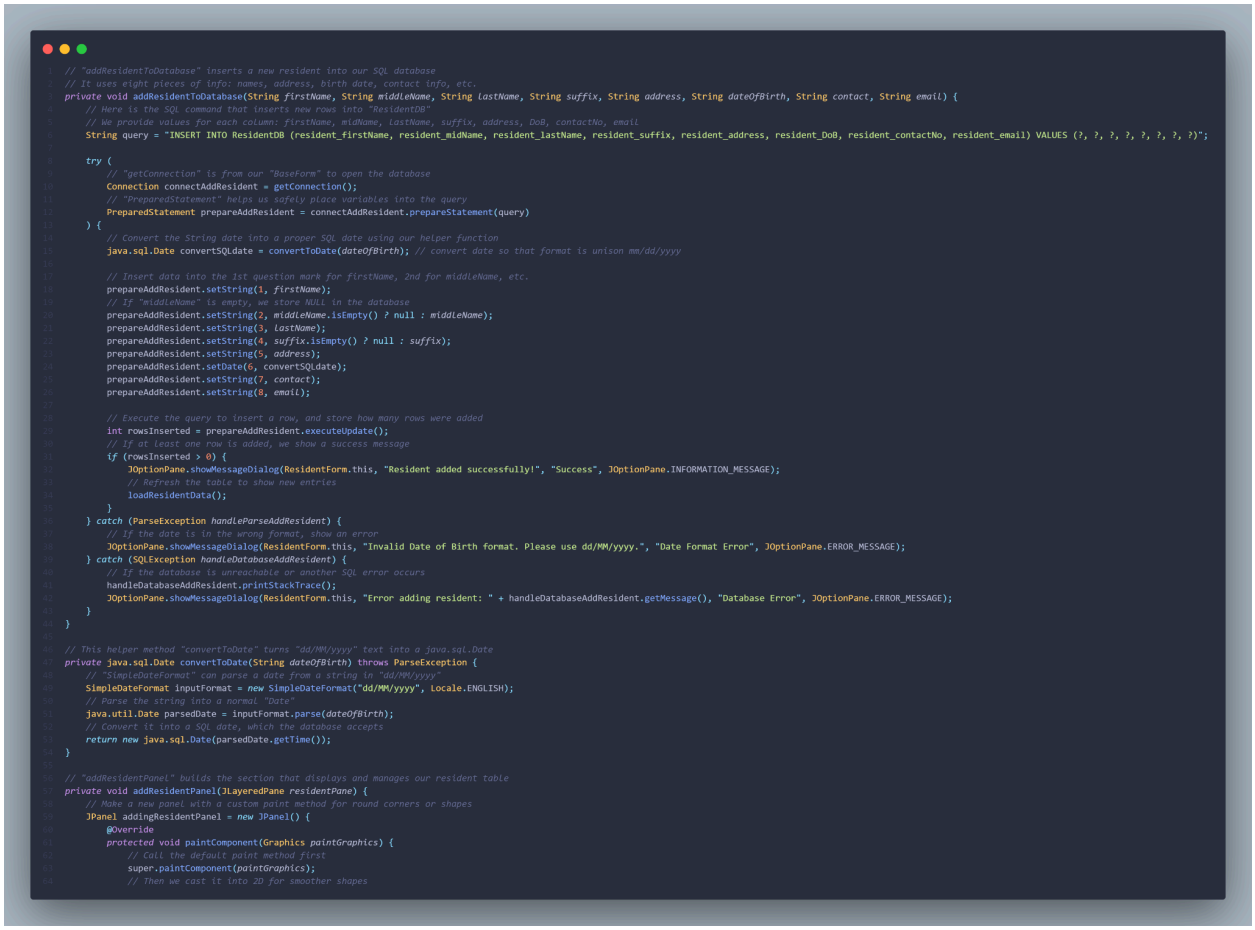
```
        JOptionPane.showMessageDialog(AnnouncementForm.this, "Message is required.", "Validation Error", JOptionPane.ERROR_MESSAGE);
        return false;
    }
    return true;
}

// "sendEmail" sends an email to the addresses in our database of residents
// It uses the JavaMail library for sending email
private void sendEmail(String subject, String messageContent) {
    // Our Gmail address and password for sending emails
    // Must be valid, or the email won't go through
    final String username = "gabdelacruz926@gmail.com";
    final String password = "wnvmmbowuvxtbbvr";

    // A set of properties that tell JavaMail how to talk to Gmail's server
    Properties propsGmailServer = new Properties();
    propsGmailServer.put("mail.smtp.auth", "true");
    propsGmailServer.put("mail.smtp.starttls.enable", "true");
    propsGmailServer.put("mail.smtp.host", "smtp.gmail.com");
    propsGmailServer.put("mail.smtp.port", "587");

    // An "Authenticator" helps verify we have the correct username and password
    Authenticator emailAuthenticator = new Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password);
        }
    };

    // A session is the connection to the mail server
    Session sessionForEmail = Session.getInstance(propsGmailServer, emailAuthenticator);

    try {
        // Open our database and find every resident's email
        Connection connectSendEmail = getConnection();
        String query = "SELECT resident_email FROM ResidentDB";
        Statement statementSendEmail = connectSendEmail.createStatement();
        ResultSet resultSendEmail = statementSendEmail.executeQuery(query);

        // We gather all emails into a string separated by commas
        StringBuilder listEmailAddress = new StringBuilder();
        while (resultSendEmail.next()) {
            String email = resultSendEmail.getString("resident_email");
            listEmailAddress.append(email).append(",");
        }
        // Remove the last comma
        if (listEmailAddress.length() > 0) {
            listEmailAddress.setLength(listEmailAddress.length() - 1);
        }

        // "MimeMessage" is the actual email we're sending
        Message createNewMessage = new MimeMessage(sessionForEmail);
        createNewMessage.setFrom(new InternetAddress(username));
        // "Message.RecipientType.TO" passes our list of emails
        createNewMessage.setRecipients(Message.RecipientType.TO, InternetAddress.parse(listEmailAddress.toString()));
        createNewMessage.setSubject(subject);
        createNewMessage.setText(messageContent);

        // Send the email
        Transport.send(createNewMessage);

```

```
            // Clear out the fields after sending
            subjectTextField.setText("");
            messageTextField.setText("");

            // Close everything: result set, statement, connection
            resultSendEmail.close();
            statementSendEmail.close();
            connectSendEmail.close();

            // Show a success message
            JOptionPane.showMessageDialog(AnnouncementForm.this, "Emails sent successfully!");

        } catch (Exception handleEmailException) {
            // If something goes wrong, show an error
            handleEmailException.printStackTrace();
            JOptionPane.showMessageDialog(AnnouncementForm.this, "Failed to send emails. " + handleEmailException.getMessage());
        }
    }

    // "sendSMS" is a placeholder for sending text messages
    // It just shows a success message for now
    private void sendSMS(String subject, String messageContent) {
        /* Assume that we send the SMS to the residents */
        // Clear the subject and message fields
        subjectTextField.setText("");
        messageTextField.setText("");

        // Show a dialog box that says it's successful
        JOptionPane.showMessageDialog(AnnouncementForm.this, "SMS sent successfully!");
    }
}
```

7) ResidentForm.java

```
// This line says our file belongs to package "system.BarrioSeguro"
// A package is like a folder that keeps similar classes together
package system.BarrioSeguro;

// This line says our file belongs to package "system.BarrioSeguro"
// A package is like a folder that keeps similar classes together
import java.awt.Color;
import java.awt.Component;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.GridLayout;
import java.awt.RenderingHints;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
```

```java
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;

import java.text.ParseException;
import java.text.SimpleDateFormat;

import java.util.Locale;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.RowFilter;
import javax.swing.SwingConstants;

import javax.swing.border.EmptyBorder;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;

// Our class "ResidentForm" extends "BaseForm," so it inherits features like window setup
public class ResidentForm extends BaseForm {

    // "residentTable" will show our rows of resident data
    private JTable residentTable;
    // "searchTextField" will let us type text for searching the table
    private JTextField searchTextField;

    // This constructor runs when we create "ResidentForm"
    // It sets the title and calls methods to build the screen and load data
    public ResidentForm(BarrioSeguro appController) {
        // Tells the parent class (BaseForm) to set basic window info
        super(appController);
        // Sets the window's title bar text
        setTitle("BarrioSeguro - Resident Database");
        // Calls a private method to set up the layout and panels
        initialize();
        // Loads existing resident data from the database into the table
        loadResidentData();
    }

    // Private method "initialize" builds the main layout inside the form
    private void initialize() {
        // "JLayeredPane" allows stacking of components (like background, panels)
        JLayeredPane residentPane = new JLayeredPane();
        // Attach this pane as the main content of our window
        setContentPane(residentPane);

        // Add a background image from the base class
        addBackgroundImage(residentPane);
        // Add a dashboard panel with home, announcements, etc.
        addDashboardPanel(residentPane);
        // Add a panel that displays and manages resident information
        addResidentPanel(residentPane);
    }
```



```java
// "addResidentToDatabase" inserts a new resident into our SQL database
// It uses eight pieces of info: names, address, birth date, contact info, etc.
private void addResidentToDatabase(String firstName, String middleName, String lastName, String suffix, String address, String dateOfBirth, String contact, String email) {
    // Here is the SQL command that inserts new rows into "ResidentDB"
    // We provide values for each column: firstName, midName, LastName, suffix, address, DoB, contactNo, email
    String query = "INSERT INTO ResidentDB (resident_firstName, resident_midName, resident_lastName, resident_suffix, resident_address, resident_DoB, resident_contactNo, resident_email) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

    try (
        // "getConnection" is from our "BaseForm" to open the database
        Connection connectAddResident = getConnection();
        // "PreparedStatement" helps us safely place variables into the query
        PreparedStatement prepareAddResident = connectAddResident.prepareStatement(query)
    ) {
        // Convert the String date into a proper SQL date using our helper function
        java.sql.Date convertSQLdate = convertToDate(dateOfBirth); // convert date so that format is unison mm/dd/yyyy

        // Insert data into the 1st question mark for firstName, 2nd for middleName, etc.
        prepareAddResident.setString(1, firstName);
        // If "middleName" is empty, we store NULL in the database
        prepareAddResident.setString(2, middleName.isEmpty() ? null : middleName);
        prepareAddResident.setString(3, lastName);
        prepareAddResident.setString(4, suffix.isEmpty() ? null : suffix);
        prepareAddResident.setString(5, address);
        prepareAddResident.setDate(6, convertSQLdate);
        prepareAddResident.setString(7, contact);
        prepareAddResident.setString(8, email);

        // Execute the query to insert a row, and store how many rows were added
        int rowsInserted = prepareAddResident.executeUpdate();
        // If at least one row is added, we show a success message
        if (rowsInserted > 0) {
            JOptionPane.showMessageDialog(ResidentForm.this, "Resident added successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            // Refresh the table to show new entries
            loadResidentData();
        }
    } catch (ParseException handleParseAddResident) {
        // If the date is in the wrong format, show an error
        JOptionPane.showMessageDialog(ResidentForm.this, "Invalid Date of Birth format. Please use dd/MM/yyyy.", "Date Format Error", JOptionPane.ERROR_MESSAGE);
    } catch (SQLException handleDatabaseAddResident) {
        // If the database is unreachable or another SQL error occurs
        handleDatabaseAddResident.printStackTrace();
        JOptionPane.showMessageDialog(ResidentForm.this, "Error adding resident: " + handleDatabaseAddResident.getMessage(), "Database Error", JOptionPane.ERROR_MESSAGE);
    }
}

// This helper method "convertToDate" turns "dd/MM/yyyy" text into a java.sql.Date
private java.sql.Date convertToDate(String dateOfBirth) throws ParseException {
    // "SimpleDateFormat" can parse a date from a string in "dd/MM/yyyy"
    SimpleDateFormat inputFormat = new SimpleDateFormat("dd/MM/yyyy", Locale.ENGLISH);
    // Parse the string into a normal "Date"
    java.util.Date parsedDate = inputFormat.parse(dateOfBirth);
    // Convert it into a SQL date, which the database accepts
    return new java.sql.Date(parsedDate.getTime());
}

// "addResidentPanel" builds the section that displays and manages our resident table
private void addResidentPanel(JLayeredPane residentPane) {
    // Make a new panel with a custom paint method for round corners or shapes
    JPanel addingResidentPanel = new JPanel() {
        @Override
        protected void paintComponent(Graphics paintGraphics) {
            // Call the default paint method first
            super.paintComponent(paintGraphics);
            // Then we cast it into 2D for smoother shapes
```

```
        Graphics2D paintGraphicsWith2D = (Graphics2D) paintGraphics;
        paintGraphicsWith2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        // Fill our panel background with the chosen color
        paintGraphicsWith2D.setColor(getBackground());
        paintGraphicsWith2D.fillRoundRect(0, 0, getWidth(), getHeight(), 75, 75);
    }
};
// Place it at x=480, y=185, width=895, height=722
addingResidentPanel.setBounds(480, 185, 895, 722);
// The color is a semi-transparent shade (R=102, G=77, B=77, alpha=178)
addingResidentPanel.setBackground(new Color(102, 77, 77, 178));
// We'll arrange items manually
addingResidentPanel.setLayout(null);
// Make the panel see-through
addingResidentPanel.setOpaque(false);

// Add this panel onto the "residentPane"
residentPane.add(addingResidentPanel, JLayeredPane.PALETTE_LAYER);

// Create our table that shows the resident data
residentTable = new JTable();
// A scrollable container that lets us scroll through the table if it's long
JScrollPane scrollResidentTable = new JScrollPane(residentTable);
// Position the scroll area on the panel at x=22, y=93, width=848, height=483
scrollResidentTable.setBounds(22, 93, 848, 483);
addingResidentPanel.add(scrollResidentTable);;

// Build a text field to handle searching
searchTextField = createRoundedTextField("Search", 25);
// No tooltip text
searchTextField.setToolTipText("");
// The text is aligned to the left
searchTextField.setHorizontalAlignment(SwingConstants.LEFT);
// Light gray text color for placeholder
searchTextField.setForeground(Color.LIGHT_GRAY);
// Background is white
searchTextField.setBackground(new Color(254, 254, 254));
// Font is plain style, size 12
searchTextField.setFont(new Font("SansSerif", Font.PLAIN, 12));
// Empty border to add a little bit of breathing space around text
searchTextField.setBorder(new EmptyBorder(10, 10, 10, 10));
// Position of the search bar on the panel
searchTextField.setBounds(22, 45, 216, 37);

// When the user clicks inside the box, if the placeholder is "Search," we remove it
searchTextField.addFocusListener(new FocusListener() {
    @Override
    public void focusGained(FocusEvent eventForSearchField) {
        if (searchTextField.getText().equals("Search")) {
            searchTextField.setText("");
            searchTextField.setForeground(Color.BLACK);
        }
    }

    @Override
    public void focusLost(FocusEvent eventForSearchField) {
        // If the user leaves it empty, put the placeholder back
        if (searchTextField.getText().isEmpty()) {
            searchTextField.setText("Search");
            searchTextField.setForeground(Color.LIGHT_GRAY);
        }
    }
});
// This "KeyListener" triggers a filter function whenever the user types
```

```java
searchTextField.addKeyListener(new KeyAdapter() { // search function
    @Override
    public void keyReleased(KeyEvent eventKeyForSearchField) {
        // We convert everything to lowercase so searching is easier
        String searchQuery = searchTextField.getText().trim().toLowerCase();
        // Filter the table results based on this text
        filterTable(searchQuery);
    }
});

// Add the search field to the panel
addingResidentPanel.add(searchTextField);

// Make an "Add" button for adding a new resident
JButton btnAdd = new JButton("Add");
styleRoundedButton(btnAdd);
// If they press "Add," we open a small form that collects the new resident's info
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent eventForAddBtn) {
        // "addResidentInfoPanel" is a small grid layout to hold text fields
        JPanel addResidentInfoPanel = new JPanel(new GridLayout(0, 2, 10, 10));
        // We label and then create a text field for each piece of info
        addResidentInfoPanel.add(new JLabel("First Name:"));
        JTextField firstNameField = new JTextField();
        addResidentInfoPanel.add(firstNameField);

        addResidentInfoPanel.add(new JLabel("Middle Name (Optional):"));
        JTextField middleNameField = new JTextField();
        addResidentInfoPanel.add(middleNameField);

        addResidentInfoPanel.add(new JLabel("Last Name:"));
        JTextField lastNameField = new JTextField();
        addResidentInfoPanel.add(lastNameField);

        addResidentInfoPanel.add(new JLabel("Suffix (Optional):"));
        JTextField suffixField = new JTextField();
        addResidentInfoPanel.add(suffixField);

        addResidentInfoPanel.add(new JLabel("Address:"));
        JTextField addressField = new JTextField();
        addResidentInfoPanel.add(addressField);

        addResidentInfoPanel.add(new JLabel("Date of Birth (dd/mm/yyyy):"));
        JTextField dobField = new JTextField();
        addResidentInfoPanel.add(dobField);

        addResidentInfoPanel.add(new JLabel("Contact Number (11 digits):"));
        JTextField contactField = new JTextField();
        addResidentInfoPanel.add(contactField);

        addResidentInfoPanel.add(new JLabel("Email:"));
        JTextField emailField = new JTextField();
        addResidentInfoPanel.add(emailField);

        // We show a simple dialog with "OK" and "Cancel"
        int result = JOptionPane.showConfirmDialog(null, addResidentInfoPanel, "Add Resident",
                JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

        // If the user clicks OK, we collect the text field values
        if (result == JOptionPane.OK_OPTION) {
            String firstName = firstNameField.getText().trim();
            String middleName = middleNameField.getText().trim();
            String lastName = lastNameField.getText().trim();
            String suffix = suffixField.getText().trim();
```

```
            String address = addressField.getText().trim();
            String dateOfBirth = dobField.getText().trim();
            String contact = contactField.getText().trim();
            String email = emailField.getText().trim();

            // Check if some required fields are empty
            if (firstName.isEmpty() || lastName.isEmpty() || address.isEmpty() || dateOfBirth.isEmpty() || contact.isEmpty() || email.isEmpty()) {
                // Show a warning if needed
                JOptionPane.showMessageDialog(null, "Please fill in all required fields.", "Validation Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Contact must be exactly 11 digits, or it's invalid
            if (!contact.matches("\\d{11}")) {
                JOptionPane.showMessageDialog(null, "Contact number must be 11 digits.", "Validation Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Date must follow dd/mm/yyyy
            if (!dateOfBirth.matches("\\d{2}/\\d{2}/\\d{4}")) {
                JOptionPane.showMessageDialog(null, "Date of Birth must be in the format dd/mm/yyyy.", "Validation Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Everything looks okay, so attempt to store it in the database
            addResidentToDatabase(firstName, middleName, lastName, suffix, address, dateOfBirth, contact, email);
        }
    }
});
// Position and add the "Add" button
btnAdd.setBounds(68, 612, 147, 64);
addingResidentPanel.add(btnAdd);

// A button to update an existing record
JButton btnUpdate = new JButton("Update");
styleRoundedButton(btnUpdate);
btnUpdate.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent eventForUpdateBtn) {
        int selectedRow = residentTable.getSelectedRow();

        // We check if a row is selected; if not, alert the user
        if (selectedRow == -1) {
            JOptionPane.showMessageDialog(null, "Please select a resident to update.", "No Selection", JOptionPane.WARNING_MESSAGE);
            return;
        }

        // We gather data from the selected row
        int residentId = (Integer) residentTable.getValueAt(selectedRow, 0);
        String firstName = (String) residentTable.getValueAt(selectedRow, 1);
        String middleName = (String) residentTable.getValueAt(selectedRow, 2);
        String lastName = (String) residentTable.getValueAt(selectedRow, 3);
        String suffix = (String) residentTable.getValueAt(selectedRow, 4);
        String address = (String) residentTable.getValueAt(selectedRow, 5);
        String dateOfBirth = residentTable.getValueAt(selectedRow, 6).toString();
        String contact = (String) residentTable.getValueAt(selectedRow, 7);
        String email = (String) residentTable.getValueAt(selectedRow, 8);

        // We build an "updateResidentInfoPanel" to let us edit existing data
        JPanel updateResidentInfoPanel = new JPanel(new GridLayout(0, 2, 10, 10));
        JTextField firstNameField = new JTextField(firstName);
        JTextField middleNameField = new JTextField(middleName);
        JTextField lastNameField = new JTextField(lastName);
        JTextField suffixField = new JTextField(suffix);
        JTextField addressField = new JTextField(address);
```

```
        JTextField dobField = new JTextField(dateOfBirth);
        JTextField contactField = new JTextField(contact);
        JTextField emailField = new JTextField(email);

        // Add labels and text fields
        updateResidentInfoPanel.add(new JLabel("First Name:"));
        updateResidentInfoPanel.add(firstNameField);
        updateResidentInfoPanel.add(new JLabel("Middle Name (Optional):"));
        updateResidentInfoPanel.add(middleNameField);
        updateResidentInfoPanel.add(new JLabel("Last Name:"));
        updateResidentInfoPanel.add(lastNameField);
        updateResidentInfoPanel.add(new JLabel("Suffix (Optional):"));
        updateResidentInfoPanel.add(suffixField);
        updateResidentInfoPanel.add(new JLabel("Address:"));
        updateResidentInfoPanel.add(addressField);
        updateResidentInfoPanel.add(new JLabel("Date of Birth (dd/MM/yyyy):"));
        updateResidentInfoPanel.add(dobField);
        updateResidentInfoPanel.add(new JLabel("Contact Number (11 digits):"));
        updateResidentInfoPanel.add(contactField);
        updateResidentInfoPanel.add(new JLabel("Email:"));
        updateResidentInfoPanel.add(emailField);

        // Show a dialog with the new panel
        int result = JOptionPane.showConfirmDialog(null, updateResidentInfoPanel, "Update Resident", JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

        // If the user chooses OK, we collect the updated text
        if (result == JOptionPane.OK_OPTION) {
            String updatedFirstName = firstNameField.getText().trim();
            String updatedMiddleName = middleNameField.getText().trim();
            String updatedLastName = lastNameField.getText().trim();
            String updatedSuffix = suffixField.getText().trim();
            String updatedAddress = addressField.getText().trim();
            String updatedDob = dobField.getText().trim();
            String updatedContact = contactField.getText().trim();
            String updatedEmail = emailField.getText().trim();

            // Check if some required fields are empty
            if (updatedFirstName.isEmpty() || updatedLastName.isEmpty() || updatedAddress.isEmpty() || updatedDob.isEmpty() || updatedContact.isEmpty() || updatedEmail.isEmpty()) {
                // Show a warning if needed
                JOptionPane.showMessageDialog(null, "Please fill in all required fields.", "Validation Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Contact must be exactly 11 digits, or it's invalid
            if (!updatedContact.matches("\\d{11}")) {
                JOptionPane.showMessageDialog(null, "Contact number must be 11 digits.", "Validation Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Date must follow dd/mm/yyyy
            if (!updatedDob.matches("\\d{2}/\\d{2}/\\d{4}")) {
                JOptionPane.showMessageDialog(null, "Date of Birth must be in the format dd/mm/yyyy.", "Validation Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
            // Try to update the database, or show an error if something goes wrong
            try {
                updateResidentInDatabase(residentId, updatedFirstName, updatedMiddleName, updatedLastName, updatedSuffix, updatedAddress, updatedDob, updatedContact, updatedEmail);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Error updating resident: " + ex.getMessage(), "Update Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});
// Position and add the "Update" button
```

```
    btnUpdate.setBounds(268, 612, 147, 62);
    addingResidentPanel.add(btnUpdate);

    // A button to remove a resident record
    JButton btnDel = new JButton("Delete");
    styleRoundedButton(btnDel);
    btnDel.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent eventForDeleteBtn) {
            int selectedRow = residentTable.getSelectedRow();

            // Check if one row is selected
            if (selectedRow == -1) {
                JOptionPane.showMessageDialog(null, "Please select a resident to delete.", "No Selection", JOptionPane.WARNING_MESSAGE);
                return;
            }

            // Get the ID for that row
            int residentId = (int) residentTable.getValueAt(selectedRow, 0);

            // Ask the user if they want to delete it for sure
            int confirm = JOptionPane.showConfirmDialog(null, "Are you sure you want to delete this resident?", "Confirm Deletion", JOptionPane.YES_NO_OPTION);

            if (confirm == JOptionPane.YES_OPTION) {
                // Actually delete the selected row from the database
                deleteResidentFromDatabase(residentId);
            }
        }
    });
    // Position and add the "Delete" button
    btnDel.setBounds(468, 612, 147, 62);
    addingResidentPanel.add(btnDel);

    // A button to print ID of a resident record
    JButton btnPrintID = new JButton("Print ID");
    styleRoundedButton(btnPrintID);
    btnPrintID.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent eventForPrintBtn) {
            int selectedRow = residentTable.getSelectedRow();

            // Check if one row is selected
            if (selectedRow == -1) {
                JOptionPane.showMessageDialog(null, "Please select a resident to print.", "No Selection", JOptionPane.WARNING_MESSAGE);
                return;
            }

            // Show the yes/no confirmation dialog for printing
            int printConfirm = JOptionPane.showConfirmDialog(null, "Are you sure you want to print this ID?", "Confirm Print", JOptionPane.YES_NO_OPTION);

            // Proceed based on the user's response
            if (printConfirm == JOptionPane.YES_OPTION) {
                JOptionPane.showMessageDialog(null, "Processing to print resident's ID. Please wait!", "Print Confirmation", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(null, "Print action canceled.", "Print Cancellation", JOptionPane.INFORMATION_MESSAGE);
            }
        }
    });
    // Position and add the "Print ID" button
    btnPrintID.setBounds(668, 612, 147, 62);
    addingResidentPanel.add(btnPrintID);
}

// "filterTable" uses a TableRowSorter to show only rows matching user's search text
private void filterTable(String searchQuery) {
    DefaultTableModel filterModel = (DefaultTableModel) residentTable.getModel();
```

```java
    TableRowSorter<DefaultTableModel> sorter = new TableRowSorter<>(filterModel);
    residentTable.setRowSorter(sorter);

    // If the user typed something, create a filter for that searchQuery
    if (searchQuery.isEmpty() || searchQuery.equalsIgnoreCase("search")) {
        sorter.setRowFilter(null);
    } else {
        sorter.setRowFilter(RowFilter.regexFilter("(?i)" + searchQuery));
    }
}

// "deleteResidentFromDatabase" removes a record by using the ID
private void deleteResidentFromDatabase(int residentId) {
     // Our DELETE query references "ResidentDB" by "resident_id"
    String query = "DELETE FROM ResidentDB WHERE resident_id = ?";

    try (
            Connection connectDeleteResident = getConnection();
            PreparedStatement prepareDeleteResident = connectDeleteResident.prepareStatement(query)
        ) {
        // Put the correct resident id in the DELETE command
        prepareDeleteResident.setInt(1, residentId);

        // Actually run the command
        int rowsDeleted = prepareDeleteResident.executeUpdate();
        // If at least one row is removed, we show success
        if (rowsDeleted > 0) {
            JOptionPane.showMessageDialog(ResidentForm.this, "Resident deleted successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            loadResidentData();
        } else {
            JOptionPane.showMessageDialog(ResidentForm.this, "No resident found to delete.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    } catch (SQLException handleDatabaseDeleteResident) {
        // If something fails, print error details and show a message
        handleDatabaseDeleteResident.printStackTrace();
        JOptionPane.showMessageDialog(ResidentForm.this, "Error deleting resident: " + handleDatabaseDeleteResident.getMessage(), "Database Error", JOptionPane.ERROR_MESSAGE);
    }
}

// This function updates an existing record by changing its data in the database
private void updateResidentInDatabase(int residentId, String firstName, String middleName, String lastName, String suffix, String address, String dateOfBirth, String contact, String email) {
    // A SQL command to modify "ResidentDB" for a matching "resident_id"
    String query = "UPDATE ResidentDB " +
                   "SET resident_firstName = ?, resident_midName = ?, resident_lastName = ?, resident_suffix = ?, resident_address = ?, resident_DoB = ?, resident_contactNo = ?, resident_email = ? " +
                   "WHERE resident_id = ?";

    try (
            Connection connectUpdateResident = getConnection();
            PreparedStatement prepareUpdateResident = connectUpdateResident.prepareStatement(query)
        ) {
        // Convert the text date into a valid SQL date
        java.sql.Date normalizeSQLdate = normalizeDate(dateOfBirth);

        // Fill each question mark with the provided data
        prepareUpdateResident.setString(1, firstName.trim());
        prepareUpdateResident.setString(2, middleName.isEmpty() ? null : middleName.trim());
        prepareUpdateResident.setString(3, lastName.trim());
        prepareUpdateResident.setString(4, suffix.isEmpty() ? null : suffix.trim());
        prepareUpdateResident.setString(5, address.trim());
        prepareUpdateResident.setDate(6, normalizeSQLdate);
        prepareUpdateResident.setString(7, contact.trim());
        prepareUpdateResident.setString(8, email.trim());
        prepareUpdateResident.setInt(9, residentId);
```

```
        // Run the update command
        int rowsUpdated = prepareUpdateResident.executeUpdate();
        // If at least one row changed, success message
        if (rowsUpdated > 0) {
            JOptionPane.showMessageDialog(ResidentForm.this, "Resident updated successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            loadResidentData();
        } else {
            JOptionPane.showMessageDialog(ResidentForm.this, "No resident found to update.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    } catch (SQLException handleDatabaseUpdateResident) {
        handleDatabaseUpdateResident.printStackTrace();
        JOptionPane.showMessageDialog(ResidentForm.this, "Error updating resident: " + handleDatabaseUpdateResident.getMessage(), "Database Error", JOptionPane.ERROR_MESSAGE);
    } catch (ParseException handleParseUpdateResident) {
        handleParseUpdateResident.printStackTrace();
        JOptionPane.showMessageDialog(ResidentForm.this, "Invalid Date format. Please use dd/MM/yyyy.", "Date Format Error", JOptionPane.ERROR_MESSAGE);
    }
}

// "normalizeDate" also turns dd/MM/yyyy into a SQL date. Similar to "convertToDate," but used for updates
private java.sql.Date normalizeDate(String dateOfBirth) throws ParseException {
    SimpleDateFormat createFormatDate = new SimpleDateFormat("dd/MM/yyyy");
    createFormatDate.setLenient(false);
    java.util.Date parsedDate = createFormatDate.parse(dateOfBirth);
    return new java.sql.Date(parsedDate.getTime());
}

// "loadResidentData" pulls all existing records from "ResidentDB" into the table
private void loadResidentData() {
    // SQL command to select every column from the ResidentDB
    String query = "SELECT * FROM ResidentDB";

    try (
            Connection connectLoadResident = getConnection();
            Statement statementLoadResident = connectLoadResident.createStatement();
            ResultSet resultLoadResident = statementLoadResident.executeQuery(query)
        ) {
        // Make a "DefaultTableModel" so we can manage rows
        DefaultTableModel loadModel = new DefaultTableModel() {
            // We override "isCellEditable" so user can't type into table cells directly
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };

        // Add column headings in the same order as the database
        loadModel.addColumn("Resident ID");
        loadModel.addColumn("First Name");
        loadModel.addColumn("Middle Name");
        loadModel.addColumn("Last Name");
        loadModel.addColumn("Suffix");
        loadModel.addColumn("Address");
        loadModel.addColumn("Date of Birth");
        loadModel.addColumn("Contact Number");
        loadModel.addColumn("Email");

        // Loop over the results
        while (resultLoadResident.next()) {
            // Add each row to the table model
            loadModel.addRow(new Object[]{
                resultLoadResident.getInt("resident_id"),
                resultLoadResident.getString("resident_firstName"),
                resultLoadResident.getString("resident_midName"),
                resultLoadResident.getString("resident_lastName"),
```

```
                    resultLoadResident.getString("resident_suffix"),
                    resultLoadResident.getString("resident_address"),
                    new SimpleDateFormat("dd/MM/yyyy").format(resultLoadResident.getDate("resident_DoB")),
                    resultLoadResident.getString("resident_contactNo"),
                    resultLoadResident.getString("resident_email")
                });
            }

            // Style the table header so it has a colorful background
            residentTable.getTableHeader().setDefaultRenderer(new DefaultTableCellRenderer() {
                @Override
                public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
                    JLabel headerLabel = new JLabel(value.toString());
                    headerLabel.setOpaque(true);
                    headerLabel.setFont(new Font("SansSerif", Font.BOLD, 14));
                    // A bright color for the header row
                    headerLabel.setBackground(new Color(255, 104, 101));
                    headerLabel.setForeground(Color.BLACK);
                    headerLabel.setHorizontalAlignment(SwingConstants.CENTER);
                    headerLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 1));
                    return headerLabel;
                }
            });

            // Apply this model to our residentTable
            residentTable.setModel(loadModel);
            // Hide the ID column so it doesn't distract the user
            residentTable.getColumnModel().getColumn(0).setMinWidth(0);
            residentTable.getColumnModel().getColumn(0).setMaxWidth(0);
            residentTable.getColumnModel().getColumn(0).setWidth(0);

        } catch (SQLException handleDatabaseLoad) {
            // If the database load fails, print details and show a popup
            handleDatabaseLoad.printStackTrace();
            JOptionPane.showMessageDialog(ResidentForm.this, "Error loading data: " + handleDatabaseLoad.getMessage(), "Database Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

8) IncidentForm.java

```
package system.BarrioSeguro; // This line says the code belongs to the "BarrioSeguro" group

import java.awt.Color; // This is used to add colors
import java.awt.Font; // This is used to add different text styles
import java.awt.Graphics; // This is used to draw things in the window
import java.awt.Graphics2D; // This is used for more advanced drawing tools
import java.awt.Insets; // This is used to set space around the text
import java.awt.RenderingHints; // This helps make drawn things look better

import java.awt.event.FocusEvent; // This is used to detect when we click or leave a text box
import java.awt.event.FocusListener; // This is used to know when we interact with the text box

import java.sql.Connection; // This is used to connect to the database
import java.sql.PreparedStatement; // This is used to run database commands
import java.sql.ResultSet; // This is used to store data from the database
import java.sql.SQLException; // This is used to handle database errors

import java.text.SimpleDateFormat; // This is used to format dates

import java.util.Date; // This is used to work with dates

import javax.swing.DefaultComboBoxModel; // This helps to manage the dropdown menu items
import javax.swing.JButton; // This is used to add buttons
import javax.swing.JComboBox; // This is used to add dropdown menus
import javax.swing.JLabel; // This is used to add text labels
import javax.swing.JLayeredPane; // This is used to Layer components over each other
import javax.swing.JOptionPane; // This is used to show messages to the user
import javax.swing.JPanel; // This will hold different parts of the Layout
import javax.swing.JTextArea; // This adds area where we can write lots of text
import javax.swing.JTextField; // This adds boxes where we can write text
import javax.swing.SwingConstants; // This is used to align text

import javax.swing.border.EmptyBorder; // This adds space around the text fields

```

```java
public class IncidentForm extends BaseForm { // This line defines our main Incident form

    // Creating text fields to enter incident details
    private JTextField incidentFirstName; // Field for first name
    private JTextField incidentMidName; // Field for middle name
    private JTextField incidentLastName; // Field for last name
    private JTextField incidentSuffixName; // Field for suffix name
    private JTextField incidentDate; // Field for date
    private JComboBox<String> incidentType; // Dropdown for types of incidents
    private JTextArea incidentDescription; // Area for incident description
    private JComboBox<String> incidentProgress; // Dropdown for incident progress

    // Constructor to set up the form
    public IncidentForm(BarrioSeguro appController) {
        super(appController); // Calling the parent class constructor
        setTitle("BarrioSeguro - Incident Reports"); // Setting the window title
        initialize(); // Calling the method to set up everything
    }

    // Method to set up the form layout and components
    private void initialize() {
        JLayeredPane incidentPane = new JLayeredPane(); // Creating a layer to add components
        setContentPane(incidentPane); // Making incidentPane the main content layer

        addBackgroundImage(incidentPane); // Adding background image to the layer
        addDashboardPanel(incidentPane); // Adding dashboard panel to the layer
        addIncidentPanel(incidentPane); // Adding the panel to enter incident details
    }

    // Method to add the panel where user enters incident details
    @SuppressWarnings("unused")
    private void addIncidentPanel(JLayeredPane incidentPane) {
        JPanel incidentPanel = new JPanel() { // Creating a new panel for the form
            @Override
            protected void paintComponent(Graphics paintGraphics) { // Customizing how the panel looks
                super.paintComponent(paintGraphics); // Calling the default painting method
                Graphics2D paintGraphicsWith2D = (Graphics2D) paintGraphics; // Using advanced graphics
                paintGraphicsWith2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON); // Making it look smooth
                paintGraphicsWith2D.setColor(getBackground()); // Setting background color
                paintGraphicsWith2D.fillRoundRect(0, 0, getWidth(), getHeight(), 75, 75); // Drawing a rounded rectangle
            }
        };
        incidentPanel.setBounds(480, 185, 895, 644); // Setting the position and size of the panel
        incidentPanel.setBackground(new Color(102, 77, 77, 178)); // Setting background color with transparency
        incidentPanel.setLayout(null); // Using absolute positioning
        incidentPanel.setOpaque(false); // Making the panel background transparent
        incidentPane.add(incidentPanel, JLayeredPane.PALETTE_LAYER); // Adding panel to the main layer

        // Setting up the text field for first name
        incidentFirstName = new JTextField(""); // Initial empty state
        incidentFirstName.setToolTipText(""); // No tooltip text
        incidentFirstName.setHorizontalAlignment(SwingConstants.LEFT); // Aligning text to the left
        incidentFirstName.setBackground(new Color(255, 244, 244)); // Background color
        incidentFirstName.setFont(new Font("SansSerif", Font.PLAIN, 12)); // Text font and size
        incidentFirstName.setBorder(new EmptyBorder(10, 10, 10, 10)); // Padding around text field
        incidentFirstName.setBounds(40, 80, 216, 36); // Position and size of the text field
        incidentPanel.add(incidentFirstName); // Adding text field to the panel

        // Setting up the text field for middle name (same steps as first name)
        incidentMidName = new JTextField("");
        incidentMidName.setToolTipText("");
        incidentMidName.setHorizontalAlignment(SwingConstants.LEFT);
        incidentMidName.setBackground(new Color(255, 244, 244));
        incidentMidName.setFont(new Font("SansSerif", Font.PLAIN, 12));
```

```java
incidentMidName.setBorder(new EmptyBorder(10, 10, 10, 10));
incidentMidName.setBounds(40, 142, 216, 36);
incidentPanel.add(incidentMidName);

// Setting up the text field for last name (same steps as first name)
incidentLastName = new JTextField("");
incidentLastName.setToolTipText("");
incidentLastName.setHorizontalAlignment(SwingConstants.LEFT);
incidentLastName.setBackground(new Color(255, 244, 244));
incidentLastName.setFont(new Font("SansSerif", Font.PLAIN, 12));
incidentLastName.setBorder(new EmptyBorder(10, 10, 10, 10));
incidentLastName.setBounds(40, 204, 216, 36);
incidentPanel.add(incidentLastName);

// Setting up the text field for suffix name (same steps as first name)
incidentSuffixName = new JTextField("");
incidentSuffixName.setToolTipText("");
incidentSuffixName.setHorizontalAlignment(SwingConstants.LEFT);
incidentSuffixName.setBackground(new Color(255, 244, 244));
incidentSuffixName.setFont(new Font("SansSerif", Font.PLAIN, 12));
incidentSuffixName.setBorder(new EmptyBorder(10, 10, 10, 10));
incidentSuffixName.setBounds(40, 266, 216, 36);
incidentPanel.add(incidentSuffixName);

// Setting up the text field for the date (same steps as first name)
incidentDate = new JTextField("");
incidentDate.setToolTipText("");
incidentDate.setHorizontalAlignment(SwingConstants.LEFT);
incidentDate.setBackground(new Color(255, 244, 244));
incidentDate.setFont(new Font("SansSerif", Font.PLAIN, 12));
incidentDate.setBorder(new EmptyBorder(10, 10, 10, 10));
incidentDate.setBounds(40, 328, 216, 36);
incidentPanel.add(incidentDate);

// Setting up the dropdown for incident types
incidentType = new JComboBox<String>();
incidentType.setBackground(new Color(255, 244, 244));
incidentType.setModel(new DefaultComboBoxModel<String>(new String[] {"Assault", "Burglary", "Domestic Violence", "Drug Possession", "Fr
incidentType.setBounds(40, 390, 216, 36); // Position and size of the dropdown
incidentPanel.add(incidentType);

// Setting up the dropdown for incident progress
incidentProgress = new JComboBox<String>();
incidentProgress.setModel(new DefaultComboBoxModel<String>(new String[] {"Under Investigation", "Ongoing", "Resolved", "Closed"}));
incidentProgress.setBounds(40, 452, 216, 36); // Position and size of the dropdown
incidentPanel.add(incidentProgress);

// Setting up the text area for incident description
incidentDescription = createRoundedTextArea("Enter a message...", 583, 423); // Custom method to create a text area
incidentDescription.setForeground(Color.BLACK); // Set initial color to black

incidentDescription.setWrapStyleWord(true); // Words won't break
incidentDescription.setBounds(273, 65, 583, 423); // Position and size of the text area
incidentDescription.setLineWrap(true); // Text wraps to the next line

incidentDescription.setMargin(new Insets(20, 20, 20, 20)); // Padding around text area
incidentDescription.addFocusListener(new FocusListener() {
    @Override
    public void focusGained(FocusEvent eventForIncidentDesc) { // When the text area is clicked
        if (incidentDescription.getText().equals("Enter a message...")) {
            incidentDescription.setText(""); // Clear the default text
            incidentDescription.setForeground(Color.BLACK); // Set text color to black
        }
    }
```

```
    @Override
    public void focusLost(FocusEvent eventForIncidentDesc) { // When the text area loses focus
        if (incidentDescription.getText().isEmpty()) {
            incidentDescription.setText("Enter a message..."); // Reset to default text
            incidentDescription.setForeground(Color.LIGHT_GRAY); // Set text color to gray
        }
    }
});
incidentPanel.add(incidentDescription); // Add text area to the panel

// Setting up labels for the text fields
JLabel lblNewLabel = new JLabel("Enter First Name");
lblNewLabel.setForeground(new Color(255, 255, 255));
lblNewLabel.setFont(new Font("Times New Roman", Font.PLAIN, 11));
lblNewLabel.setBounds(40, 65, 118, 19);
incidentPanel.add(lblNewLabel);

// Setting up label for middle name (same steps as first name label)
JLabel lblEnterMiddleName = new JLabel("Enter Middle Name (empty if none)");
lblEnterMiddleName.setForeground(new Color(255, 255, 255));
lblEnterMiddleName.setFont(new Font("Times New Roman", Font.PLAIN, 11));
lblEnterMiddleName.setBounds(40, 127, 216, 18);
incidentPanel.add(lblEnterMiddleName);

// Setting up label for last name (same steps as first name label)
JLabel lblEnterLastName = new JLabel("Enter Last Name");
lblEnterLastName.setFont(new Font("Times New Roman", Font.PLAIN, 11));
lblEnterLastName.setForeground(new Color(255, 255, 255));
lblEnterLastName.setBounds(40, 189, 184, 18);
incidentPanel.add(lblEnterLastName);

// Setting up label for suffix name (same steps as first name label)
JLabel lblEnterSuffixempty = new JLabel("Enter Suffix (empty if none)");
lblEnterSuffixempty.setForeground(new Color(255, 255, 255));
lblEnterSuffixempty.setFont(new Font("Times New Roman", Font.PLAIN, 11));
lblEnterSuffixempty.setBounds(40, 251, 184, 18);
incidentPanel.add(lblEnterSuffixempty);

// Setting up label for the date (same steps as first name label)
JLabel lblEnterDate = new JLabel("Enter Date: DD/MM/YYYY");
lblEnterDate.setForeground(new Color(255, 255, 255));
lblEnterDate.setFont(new Font("Times New Roman", Font.PLAIN, 11));
lblEnterDate.setBounds(40, 313, 184, 18);
incidentPanel.add(lblEnterDate);

// Setting up label for incident type (same steps as first name label)
JLabel lblChooseAType = new JLabel("Choose a type of Incident");
lblChooseAType.setForeground(new Color(255, 255, 255));
lblChooseAType.setFont(new Font("Times New Roman", Font.PLAIN, 11));
lblChooseAType.setBounds(40, 375, 184, 21);
incidentPanel.add(lblChooseAType);

// Setting up label for incident progress (same steps as first name label)
JLabel lblIncidentProgress = new JLabel("Incident Progress");
lblIncidentProgress.setForeground(new Color(255, 255, 255));
lblIncidentProgress.setFont(new Font("Times New Roman", Font.PLAIN, 11));
lblIncidentProgress.setBounds(40, 437, 184, 18);
incidentPanel.add(lblIncidentProgress);

// Setting up submit button
JButton submitbtn = new JButton("Submit");
styleRoundedButton(submitbtn); // Custom method to style the button
submitbtn.setBounds(378, 529, 150, 55); // Position and size of the button
```

```java
    submitbtn.addActionListener(eventForSubmitBtn -> { // When the button is clicked
        if (validateIncidentDescription()) { // Check if description is valid
            submitIncident(); // Submit the incident
        }
    });
    incidentPanel.add(submitbtn); // Add the button to the panel
}

// Method to validate that the description is filled in
private boolean validateIncidentDescription() {
    if (incidentDescription.getText().equals("Enter a message...") || incidentDescription.getText().trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "Incident description is required.", "Validation Error", JOptionPane.ERROR_MESSAGE);
        return false;
    }
    return true;
}

// Method to validate important fields are filled in
private boolean validateFormFields() {
    if (incidentFirstName.getText().isEmpty() || incidentLastName.getText().isEmpty() || incidentDate.getText().isEmpty()) {
        JOptionPane.showMessageDialog(this, "First Name, Last Name, and Date are required fields.");
        return false;
    }
    return true;
}

// Method for submitting the incident form to the database
private void submitIncident() {
    if (!validateFormFields()) { // Check if required fields are filled
        return;
    }

    try {
        String dateString = incidentDate.getText(); // Get date from text field
        SimpleDateFormat formatDateString = new SimpleDateFormat("dd/MM/yyyy"); // Date format we want
        formatDateString.setLenient(false); // Exact format is expected

        Date parsedDate = null;
        try {
            parsedDate = formatDateString.parse(dateString); // Try to convert text to date
        } catch (Exception handleParsedDateException) {
            JOptionPane.showMessageDialog(this, "Invalid date format. Please use dd/MM/yyyy.");
            return;
        }

        java.sql.Date convertedSQLdate = new java.sql.Date(parsedDate.getTime()); // Convert to SQL date

        try (Connection connectSubmitIncident = getConnection()) { // Establish database connection

            String checkQuery = "SELECT COUNT(*) FROM IncidentDB WHERE incident_firstName = ? " +
                                "AND incident_lastName = ? AND incident_date = ?"; // SQL query
            try (PreparedStatement checkStmt = connectSubmitIncident.prepareStatement(checkQuery)) {
                // Fill the SQL query with form data
                checkStmt.setString(1, incidentFirstName.getText());
                checkStmt.setString(2, incidentLastName.getText());
                checkStmt.setDate(3, convertedSQLdate);

                // Execute the query
                ResultSet resultSubmitIncident = checkStmt.executeQuery();
                resultSubmitIncident.next();
                int count = resultSubmitIncident.getInt(1);

                if (count > 0) {
                    // Update query if entry exists
```

```
                    String updateQuery = "UPDATE IncidentDB SET incident_midName = ?, incident_suffix = ?, " +
                                         "incident_type = ?, incident_description = ?, incident_progress = ? " +
                                         "WHERE incident_firstName = ? AND incident_lastName = ? AND incident_date = ?";

                    try (PreparedStatement updateStmt = connectSubmitIncident.prepareStatement(updateQuery)) {
                        // Fill the update query with form data
                        updateStmt.setString(1, incidentMidName.getText());
                        updateStmt.setString(2, incidentSuffixName.getText());
                        updateStmt.setString(3, (String) incidentType.getSelectedItem());
                        updateStmt.setString(4, incidentDescription.getText());
                        updateStmt.setString(5, (String) incidentProgress.getSelectedItem());
                        updateStmt.setString(6, incidentFirstName.getText());
                        updateStmt.setString(7, incidentLastName.getText());
                        updateStmt.setDate(8, convertedSQLdate);

                        int rowsUpdated = updateStmt.executeUpdate(); // Execute update
                        if (rowsUpdated > 0) {
                            JOptionPane.showMessageDialog(this, "Incident Report Saved Successfully!");
                        } else {
                            JOptionPane.showMessageDialog(this, "No matching record found to update.");
                        }
                    }
                } else {
                    // Insert query if no matching entry
                    String insertQuery = "INSERT INTO IncidentDB (incident_firstName, incident_midName, " +
                                         "incident_lastName, incident_suffix, incident_date, incident_type, " +
                                         "incident_description, incident_progress) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

                    try (PreparedStatement insertStmt = connectSubmitIncident.prepareStatement(insertQuery)) {
                        // Fill the insert query with form data
                        insertStmt.setString(1, incidentFirstName.getText());
                        insertStmt.setString(2, incidentMidName.getText());
                        insertStmt.setString(3, incidentLastName.getText());
                        insertStmt.setString(4, incidentSuffixName.getText());
                        insertStmt.setDate(5, convertedSQLdate);
                        insertStmt.setString(6, (String) incidentType.getSelectedItem());
                        insertStmt.setString(7, incidentDescription.getText());
                        insertStmt.setString(8, (String) incidentProgress.getSelectedItem());

                        insertStmt.executeUpdate(); // Execute insert
                        JOptionPane.showMessageDialog(this, "Incident Report Submitted Successfully!");
                    }
                }

                clearFormFields(); // Clear form fields after submission

                incidentFirstName.requestFocus(); // Set focus back to the first name field
            }
        } catch (SQLException handleDatabaseException) {
            JOptionPane.showMessageDialog(this, "Error submitting incident report: " + handleDatabaseException.getMessage());
        }
    } catch (Exception handleSubmitException) {
        JOptionPane.showMessageDialog(this, "Unexpected error: " + handleSubmitException.getMessage());
    }
}

// Clear function after successfully inputting incident
private void clearFormFields() {
    incidentFirstName.setText("");
    incidentMidName.setText("");
    incidentLastName.setText("");
    incidentSuffixName.setText("");
    incidentDate.setText("");
    incidentType.setSelectedIndex(0); // Reset incident type
```

```
        incidentProgress.setSelectedIndex(0); // Reset incident progress
        incidentDescription.setText("Enter a message...");
        incidentDescription.setForeground(Color.LIGHT_GRAY); // Reset description color
    }

    // Function for to be used in summaryForm (Fill form with given data)
    public void fillData(String firstName, String middleName, String lastName, String suffix, String date, String progress, String description, String typeOfIncident) {
        incidentFirstName.setText(firstName);
        incidentMidName.setText(middleName);
        incidentLastName.setText(lastName);
        incidentSuffixName.setText(suffix);
        incidentDate.setText(date);
        incidentType.setSelectedItem(typeOfIncident);
        incidentDescription.setText(description);
        incidentProgress.setSelectedItem(progress);
    }
}
```

9) SummaryForm.java

```
package system.BarrioSeguro; // This specifies the code belongs to the "BarrioSeguro" group

import java.awt.Color; // This is used to add colors
import java.awt.Component; // For creating and managing different parts of the interface
import java.awt.Font; // This is used to add different text styles
import java.awt.Graphics; // This is used to draw things in the window
import java.awt.Graphics2D; // This is used for more advanced drawing tools
import java.awt.RenderingHints; // This helps make drawn things look better

import java.awt.event.ActionEvent; // This is used to detect button clicks
import java.awt.event.ActionListener; // This is used to handle button click events
import java.awt.event.FocusEvent; // This is used to detect when we click or leave a text box
import java.awt.event.FocusListener; // This is used to know when we interact with the text box
import java.awt.event.KeyAdapter; // This is used to handle keyboard events
import java.awt.event.KeyEvent; // This is used to detect key presses

import java.sql.Connection; // This is used to connect to the database
import java.sql.PreparedStatement; // This is used to run database commands
import java.sql.ResultSet; // This is used to store data from the database
import java.sql.Statement; // This is used to run simple database commands
import java.sql.SQLException; // This is used to handle database errors

import java.text.SimpleDateFormat; // This is used to format dates

import java.util.Date; // This is used to work with dates

import javax.swing.BorderFactory; // This is used to create borders around components
import javax.swing.JButton; // This is used to add buttons
import javax.swing.JLabel; // This is used to add text labels
import javax.swing.JLayeredPane; // This is used to layer components over each other
import javax.swing.JOptionPane; // This is used to show messages to the user
import javax.swing.JPanel; // This will hold different parts of the layout
import javax.swing.JScrollPane; // This is used to make scrollable components
import javax.swing.JTable; // This is used to display tables
import javax.swing.JTextField; // This is used to add text boxes
import javax.swing.RowFilter; // This is used to filter table rows based on conditions
import javax.swing.SwingConstants; // This is used to align text

import javax.swing.border.EmptyBorder; // This adds space around the text fields
import javax.swing.table.DefaultTableCellRenderer; // This is used to control how table cells look
import javax.swing.table.DefaultTableModel; // This is used to manage table data
import javax.swing.table.TableRowSorter; // This is used to sort and filter table rows

```

```java
public class SummaryForm extends BaseForm { // This line defines our main Summary form

    private JTable incidentTable; // Table to display incident records
    private JTextField searchTextField; // Text field to search in the table

    public SummaryForm(BarrioSeguro appController) { // Constructor to set up the form
        super(appController); // Calling the parent class constructor
        setTitle("BarrioSeguro - Summary Reports"); // Setting the window title
        initialize(); // Calling the method to set up everything
        loadIncidentData(); // Loading incident data from the database
    }

    // Method to set up the form layout and components
    private void initialize() {
        JLayeredPane summaryPane = new JLayeredPane(); // Creating a layer to add components
        setContentPane(summaryPane); // Making summaryPane the main content layer

        addBackgroundImage(summaryPane); // Adding background image to the layer
        addDashboardPanel(summaryPane); // Adding dashboard panel to the layer
        addSummaryPanel(summaryPane); // Adding the panel to display summary details
    }

    // Method to add the panel for summary details
    private void addSummaryPanel(JLayeredPane summaryPane) {
        JPanel summaryPanel = new JPanel() { // Creating a new panel for the form
            @Override
            protected void paintComponent(Graphics paintGraphics) { // Customizing how the panel looks
                super.paintComponent(paintGraphics); // Calling the default painting method
                Graphics2D paintGraphicsWith2D = (Graphics2D) paintGraphics; // Using advanced graphics
                paintGraphicsWith2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON); // Making it look smooth
                paintGraphicsWith2D.setColor(getBackground()); // Setting background color
                paintGraphicsWith2D.fillRoundRect(0, 0, getWidth(), getHeight(), 75, 75); // Drawing a rounded rectangle
            }
        };
        summaryPanel.setBounds(480, 185, 895, 722); // Setting the position and size of the panel
        summaryPanel.setBackground(new Color(102, 77, 77, 178)); // Setting background color with transparency
        summaryPanel.setLayout(null); // Using absolute positioning
        summaryPanel.setOpaque(false); // Making the panel background transparent

        summaryPane.add(summaryPanel, JLayeredPane.PALETTE_LAYER); // Adding panel to the main layer

        incidentTable = new JTable() { // Creating a new table for displaying data
            @Override
            public boolean isCellEditable(int row, int column) { // Making table cells not editable editable
                return false;
            }
        };
        incidentTable.setBounds(36, 38, 826, 581); // Setting the position and size of the table
        summaryPanel.add(incidentTable); // Adding the table to the panel

        String[] columnNames = { // Defining column names for the table
            "First Name", "Last Name", "Date", "Progress"
        };

        DefaultTableModel summaryTableModel = new DefaultTableModel(null, columnNames); // Creating a table model with column names

        incidentTable.getTableHeader().setDefaultRenderer(new DefaultTableCellRenderer() { // Customizing table header
            @Override
            public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
                JLabel headerLabel = new JLabel(value.toString()); // Creating a label for header cell
                headerLabel.setOpaque(true); // Making the label background show
                headerLabel.setFont(new Font("SansSerif", Font.BOLD, 14)); // Setting font for header text
                headerLabel.setBackground(new Color(255, 104, 101)); // Setting background color of header cell
                headerLabel.setForeground(Color.BLACK); // Setting text color of header cell
```

```
        headerLabel.setHorizontalAlignment(SwingConstants.CENTER); // Aligning header text to center
        headerLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 1)); // Adding border around header cell
        return headerLabel;
    }
});

incidentTable.setModel(summaryTableModel); // Assigning the table model to the table

JScrollPane scrollSummaryTable = new JScrollPane(incidentTable); // Making the table scrollable
scrollSummaryTable.setBounds(36, 80, 826, 561); // Setting the position and size of the scroll pane
summaryPanel.add(scrollSummaryTable); // Adding the scroll pane to the panel

// Setting the preferred width for each table column
incidentTable.getColumnModel().getColumn(0).setPreferredWidth(150);
incidentTable.getColumnModel().getColumn(1).setPreferredWidth(150);
incidentTable.getColumnModel().getColumn(2).setPreferredWidth(150);
incidentTable.getColumnModel().getColumn(3).setPreferredWidth(150);

// Setting up the text field for searching in the table
searchTextField = createRoundedTextField("Search", 25); // Custom method to create a text field
searchTextField.setToolTipText(""); // No tooltip text
searchTextField.setHorizontalAlignment(SwingConstants.LEFT); // Aligning text to the left
searchTextField.setForeground(Color.LIGHT_GRAY); // Initial text color
searchTextField.setFont(new Font("SansSerif", Font.PLAIN, 12)); // Text font and size
searchTextField.setBorder(new EmptyBorder(10, 10, 10, 10)); // Padding around text field
searchTextField.setBounds(36, 34, 216, 37); // Position and size of the text field

// Adding a focus listener to handle focus events on the text field
searchTextField.addFocusListener(new FocusListener() {
    @Override
    public void focusGained(FocusEvent eventForSearchField) { // When the text field is clicked
        if (searchTextField.getText().equals("Search")) {
            searchTextField.setText(""); // Clear the default text
            searchTextField.setForeground(Color.BLACK); // Set text color to black
        }
    }

    @Override
    public void focusLost(FocusEvent eventForSearchField) { // When the text field loses focus
        if (searchTextField.getText().isEmpty()) {
            searchTextField.setText("Search"); // Reset to default text
            searchTextField.setForeground(Color.LIGHT_GRAY); // Set text color to gray
        }
    }
});

// Adding a key listener to handle key events on the text field
searchTextField.addKeyListener(new KeyAdapter() {
    @Override
    public void keyReleased(KeyEvent eventKeyForSearchField) { // When a key is released in the text field
        String searchQuery = searchTextField.getText().trim().toLowerCase(); // Get the text from the field and convert to lowercase
        filterTable(searchQuery); // Filter the table rows based on the text
    }
});
summaryPanel.add(searchTextField); // Add the text field to the panel

// Setting up the button to view selected row details
JButton viewBtn = new JButton("VIEW");
styleRoundedButton(viewBtn); // Custom method to style the button
viewBtn.addActionListener(new ActionListener() { // Adding an action listener for the button
    public void actionPerformed(ActionEvent eventForViewBtn) { // When the button is clicked
        int selectedRow = incidentTable.getSelectedRow(); // Get the selected row in the table

        if (selectedRow == -1) { // If no row is selected
```

```
                JOptionPane.showMessageDialog(null, "Please select an incident row to view.", "No Selection", JOptionPane.WARNING_MESSAGE);
                return; // Show a message and return
            }

            if (selectedRow != -1) { // If a row is selected
                // Get details from the selected row
                String firstName = (String) incidentTable.getValueAt(selectedRow, 0);
                String lastName = (String) incidentTable.getValueAt(selectedRow, 1);
                String dateString = (String) incidentTable.getValueAt(selectedRow, 2);
                String progress = (String) incidentTable.getValueAt(selectedRow, 3);

                SimpleDateFormat formatDateConverter = new SimpleDateFormat("dd/MM/yyyy"); // Date format for conversion
                java.sql.Date giveDate = null;
                try {
                    java.util.Date convertedDate = formatDateConverter.parse(dateString);
                    giveDate = new java.sql.Date(convertedDate.getTime()); // Convert text to date
                } catch (Exception handleDateException) {
                    JOptionPane.showMessageDialog(null, "Invalid date format."); // Show error if date is wrong
                    return;
                }

                String description = ""; // Initialize description variable
                String typeOfIncident = ""; // Initialize incident type variable

                try (Connection connectSummary = getConnection()) { // Establish database connection
                    // SQL query to get the incident description and type
                    String query = "SELECT incident_description, incident_type FROM IncidentDB " +
                                   "WHERE incident_firstName = ? AND incident_lastName = ? AND incident_date = ?";

                    // Fill the SQL query with row details
                    try (PreparedStatement statementSummary = connectSummary.prepareStatement(query)) {
                        statementSummary.setString(1, firstName);
                        statementSummary.setString(2, lastName);
                        statementSummary.setDate(3, giveDate);
                        ResultSet resultSummary = statementSummary.executeQuery(); // Execute the query

                        if (resultSummary.next()) { // If result found
                            description = resultSummary.getString("incident_description"); // Get the description
                            typeOfIncident = resultSummary.getString("incident_type"); // Get the type of incident
                        } else {
                            JOptionPane.showMessageDialog(null, "No matching record found."); // Show error if no result
                        }
                    }
                } catch (SQLException handleViewDatabaseException) {
                    handleViewDatabaseException.printStackTrace(); // Print any database errors
                }

                IncidentForm createIncidentForm = new IncidentForm(appController); // Create the incident form

                // Fill the incident form with data from the selected row
                createIncidentForm.fillData(firstName, "", lastName, "", dateString, progress, description, typeOfIncident);

                createIncidentForm.setVisible(true); // Show the incident form

                dispose(); // Close the summary form
            }
        }
    });
    viewBtn.setFont(new Font("Times New Roman", Font.BOLD, 14)); // Setting the font for the button
    viewBtn.setBounds(361, 664, 160, 37); // Setting the position and size of the button
    summaryPanel.add(viewBtn); // Adding the button to the panel
}

// Method to filter the table rows based on the search query
```

```java
    private void filterTable(String searchQuery) { // function for search
        DefaultTableModel filterTableModel = (DefaultTableModel) incidentTable.getModel(); // Get the table model
        TableRowSorter<DefaultTableModel> sorter = new TableRowSorter<>(filterTableModel); // Create a row sorter
        incidentTable.setRowSorter(sorter); // Set the sorter for the table

        if (searchQuery.isEmpty() || searchQuery.equalsIgnoreCase("search")) {
            sorter.setRowFilter(null); // If query is empty, show all rows
        } else {
            sorter.setRowFilter(RowFilter.regexFilter("(?i)" + searchQuery)); // Filter rows based on the query
        }
    }

    // Method to load incident data from the database and display it in the table
    private void loadIncidentData() {
        String query = "SELECT incident_firstName, incident_lastName, incident_date, incident_progress " +
                       "FROM IncidentDB"; // SQL query to get incident data

        try (Connection connectLoadIncident = getConnection()) { // Establish database connection
            Statement statementLoadIncident = connectLoadIncident.createStatement(); // Create a statement
            ResultSet resultLoacIncident = statementLoadIncident.executeQuery(query); // Execute the query

            DefaultTableModel loadTableModel = (DefaultTableModel) incidentTable.getModel(); // Get the table model
            SimpleDateFormat formatDateConverter = new SimpleDateFormat("dd/MM/yyyy"); // Date format for conversion

            while (resultLoacIncident.next()) { // Iterate over the results
                // Get data from the result set
                String firstName = resultLoacIncident.getString("incident_firstName");
                String lastName = resultLoacIncident.getString("incident_lastName");
                Date incidentDate = resultLoacIncident.getDate("incident_date");
                String progress = resultLoacIncident.getString("incident_progress");

                String formattedDate = formatDateConverter.format(incidentDate); // Format the date

                // Add a row to the table with the data
                loadTableModel.addRow(new Object[]{firstName, lastName, formattedDate, progress});
            }
        } catch (SQLException handleDatabaseException) {
            handleDatabaseException.printStackTrace(); // Print any database errors
        }
    }
}
```

REFERENCES:

[1] C. O. Carpio, "Barangay management system," *Int. J. Multidisciplinary Res. Publ. (IJMRAP)*, vol. 3, no. 2, pp. 26–32, 2020. [Online]. Available: http://ijmrap.com/wp-content/uploads/2020/07/IJMRAP-V3N1P78Y20.pdf

[2] Department of the Interior and Local Government, "Engaging the Barangays: Guide for Mayors," DILG Regional Office No. 5, 2022. [Online]. Available: https://region5.dilg.gov.ph/lgrrc/wp-content/uploads/2022/07/Engaging-the-Barangays-Guide-for-Mayors-rev2.pdf

[3] G. A. Empinado *et al.*, "Strengthening community safety awareness of barangay public safety officers on their duties and responsibilities," *Int. J. Recent Adv. Multidisciplinary Res.*, vol. 11, no. 1, pp. 9436–9442, 2024. [Online]. Available: https://ijramr.com/sites/default/files/issues-pdf/5015.pdf

[4] L. Antonio, "Enhancing barangay justice system through the development of a web-based crime monitoring module," *SSRN*, 2020. [Online]. Available: https://doi.org/10.2139/ssrn.3642023

[5] E. Bangad *et al.*, "Environmental factors affecting incidence of crime in a high-risk barangay of a commercial town," *EasyChair Preprint*, Oct. 2024. [Online]. Available: https://login.easychair.org/publications/preprint/ddbH

[6] J. Sigales, "Index crime rate in PH down by 61.87% from 2022 to 2024 – PNP," *Inquirer.net*, Oct. 29, 2024. [Online]. Available: https://newsinfo.inquirer.net/1998778/

[7] L. H. Lubomski, P. J. Pronovost, D. A. Thompson, C. G. Holzmueller, T. Dorman, L. L. Morlock, F. Dickman, M. Fahey, and A. W. Wu, "Building a better incident reporting

system: Perspectives from a multisite project," *Journal of Science Communication*, vol. 11, no. 5, May 2004. Available: https://www.researchgate.net/publication/265103821.

[8] C. Macrae, "The problem with incident reporting," *BMJ Quality & Safety*, vol. 25, no. 2, pp. 71–75, 2016, doi: 10.1136/bmjqs-2015-004732. Available: http://qualitysafety.bmj.com/content/25/2/71.