Lanka Nippon BizTech Institute

Course Code: ICT2403

Course Name: Object Oriented Programming

Name of the System: Passport Automation System

Name: Shanaya Hassen

Index Number: UOG0322008

Contents

1. Introduction

(a brief introduction about the passport system, unified models and some theories)

The Passport Automation System streamlines passport issuance by digitizing application processes and verifying applicant information against existing records. After applicants submit online registration forms, their details undergo authentication to ensure accuracy. Verified data is then forwarded to regional administrator offices for manual processing, with any discrepancies potentially leading to penalties. The system also facilitates appointment scheduling for document verification at administrator offices, allowing applicants to choose convenient dates. Police conduct additional verification, with findings reported back to administrators. Applicants can track their application status online. Once all requirements are met, the system updates the database and dispatches passports to applicants, enhancing processing efficiency while maintaining rigorous verification standards.

In the design phase of the project, various UML (Unified Modeling Language) diagrams were employed to illustrate different aspects of the system. These diagrams included use case diagrams, which depicted the various interactions between users and the system, sequence diagrams, illustrating the sequence of actions or messages exchanged between objects, class diagrams, detailing the structure of the system by showing classes, attributes, and their relationships, activity diagrams, mapping out the flow of activities or processes within the system, and state chart diagrams, which modeled the different states that objects within the system could transition through.

In the implementation phase, Java and MySQL were chosen as the primary technologies. Java, being an object-oriented programming language, aligned well with the object-oriented principles considered during the design phase. It facilitated the creation of modular, reusable, and easily maintainable code, thus promoting scalability and flexibility in the system's architecture. MySQL, a popular relational database management system, provided a robust and efficient means for storing and retrieving data, ensuring the system's data integrity and reliability.

Central to the system's implementation was the adherence to object-oriented principles. Object orientation allowed for the modeling of real-world entities as objects, encapsulating their data and behavior within classes. This approach promoted code reusability, modularity, and extensibility, enabling the development team to efficiently manage the complexity of the system and adapt it to changing requirements over time.

By leveraging UML diagrams for design and employing Java and MySQL for implementation, the project benefited from a systematic and well-structured approach, facilitating effective communication among stakeholders, fostering collaboration among development team members, and ultimately leading to the successful realization of the system's objectives.

2. Requirements

2.1. Functional Requirements

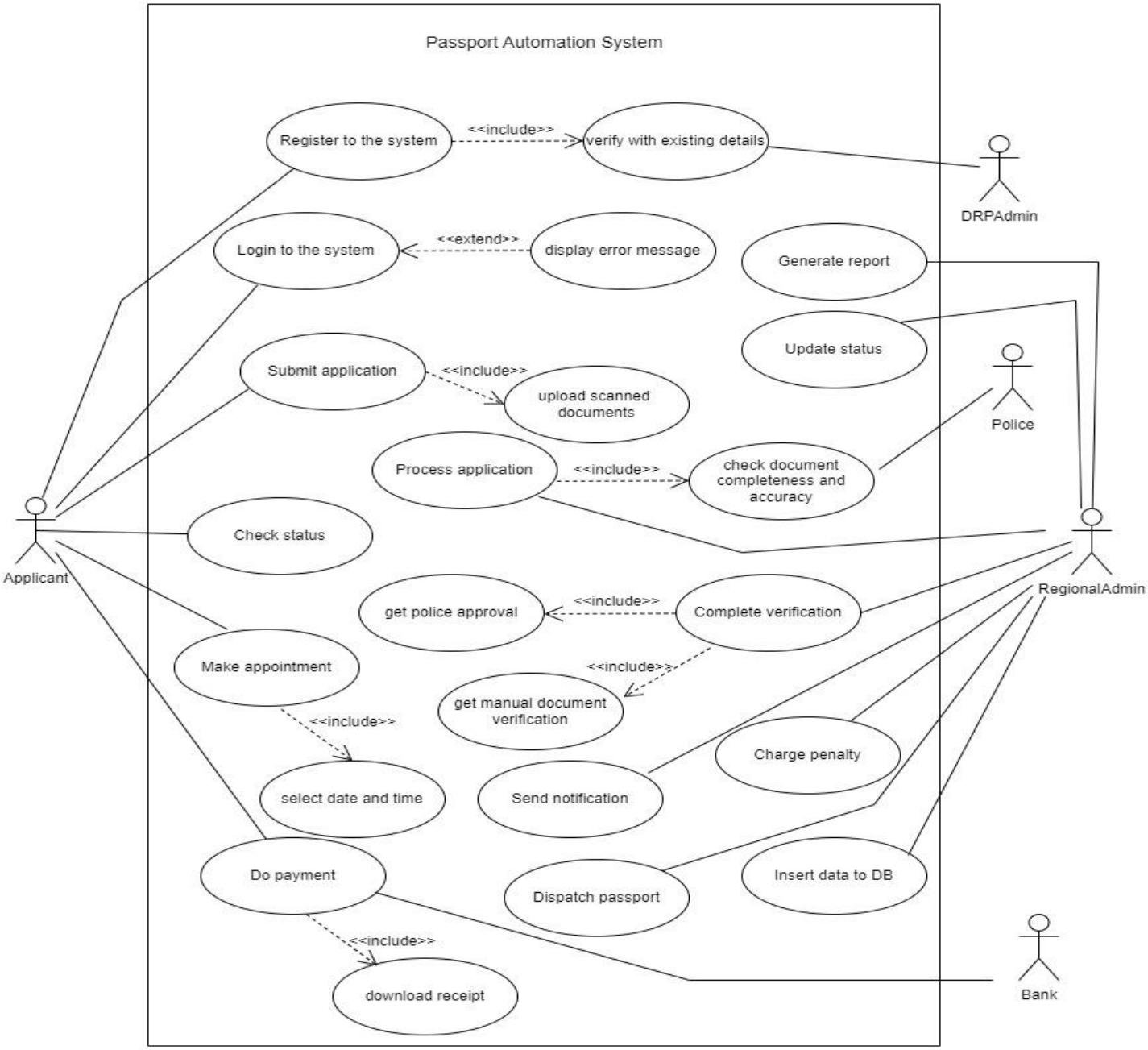
1. User Registration - Allow users to register their personal details, including name, contact information, and biographical data.
2. Application Submission - Enable users to submit passport applications electronically, providing necessary documentation and supporting information.
3. Application Processing - Automate the processing of passport applications, including verification of applicant details, background checks, and approval procedures.
4. Document Management - Manage digital copies of applicant documents, including photographs, identification proofs, and supporting certificates.
5. Appointment Scheduling - Provide functionality for applicants to schedule appointments for document verification and biometric data capture.
6. Status Tracking - Allow applicants to track the status of their passport application at different stages of processing.
7. Reporting - Generate reports for administrative purposes, including application statistics, processing times, and system performance metrics.
8. Integration with External Systems - Integrate with external databases and verification systems, such as law enforcement databases and immigration authorities, for background checks and validation.
9. Payment Processing - Facilitate online payment for passport application fees and associated services.
10. User Authentication and Authorization - Implement secure authentication mechanisms to verify the identity of users and grant appropriate access permissions based on roles.

2.2. Non-Functional Requirements

1. Security - Ensure the confidentiality, integrity, and availability of applicant data throughout the system.
2. Scalability - Design the system to handle varying levels of application volume without compromising performance.
3. Reliability - Minimize system downtime and errors to ensure continuous availability and smooth operation.
4. Usability - Provide an intuitive user interface that is easy to navigate and understand, catering to users with varying levels of technical expertise.
5. Performance - Optimize system performance to ensure timely processing of applications and minimal response times for user interactions.
6. Accessibility - Ensure that the system is accessible to users with disabilities, complying with accessibility standards and guidelines.
7. Compliance - Adhere to regulatory requirements and industry standards related to passport issuance, data protection, and privacy.
8. Interoperability - Support interoperability with other government systems and international standards for passport issuance and verification.
9. Maintainability - Design the system with modular components and clear documentation to facilitate maintenance, updates, and enhancements.

10. Auditability - Enable logging and auditing mechanisms to track system activities, user interactions, and changes to sensitive data for accountability and compliance purposes.

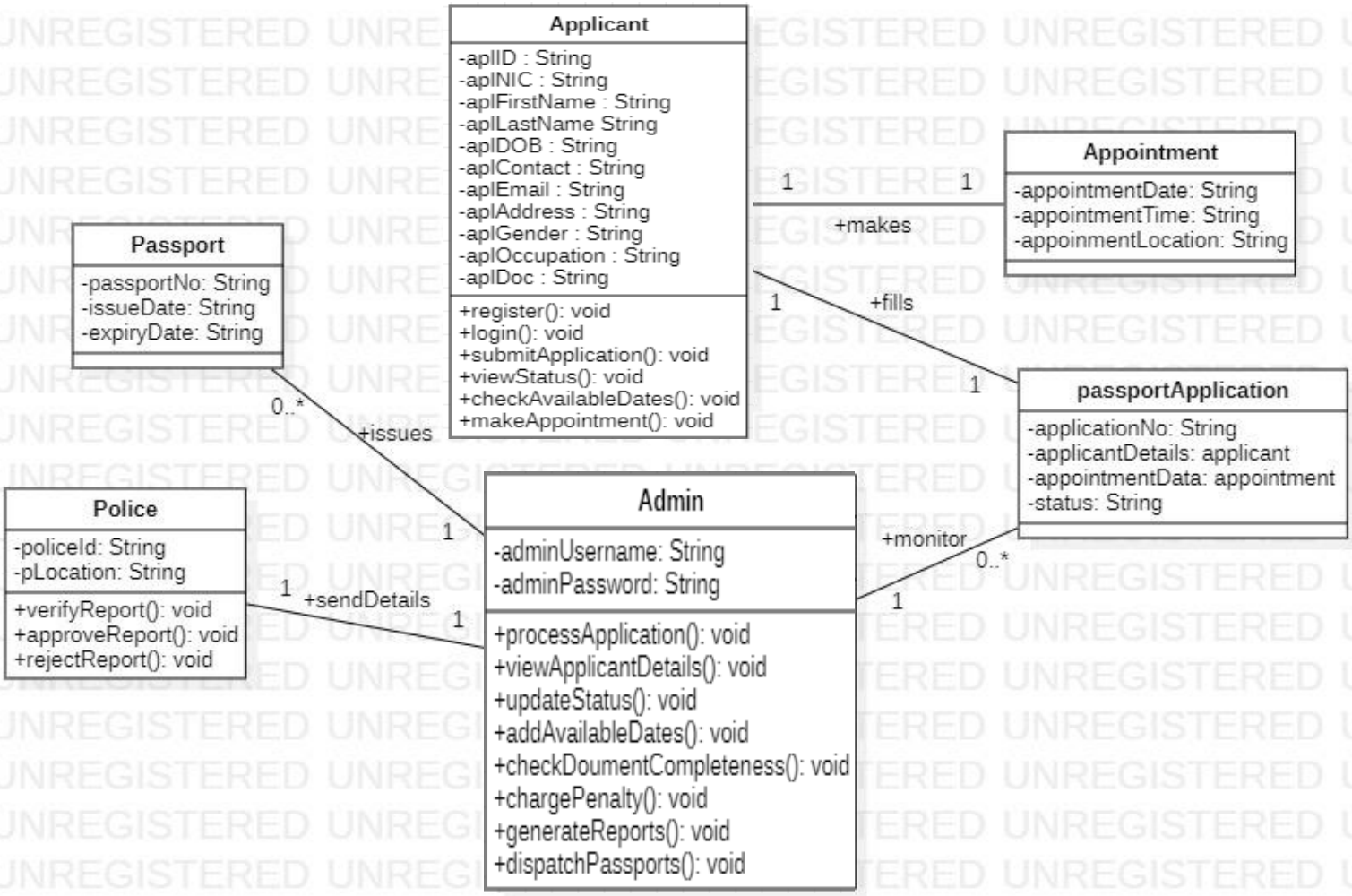
2.3. Use-case Diagram



3. Analysis and Design

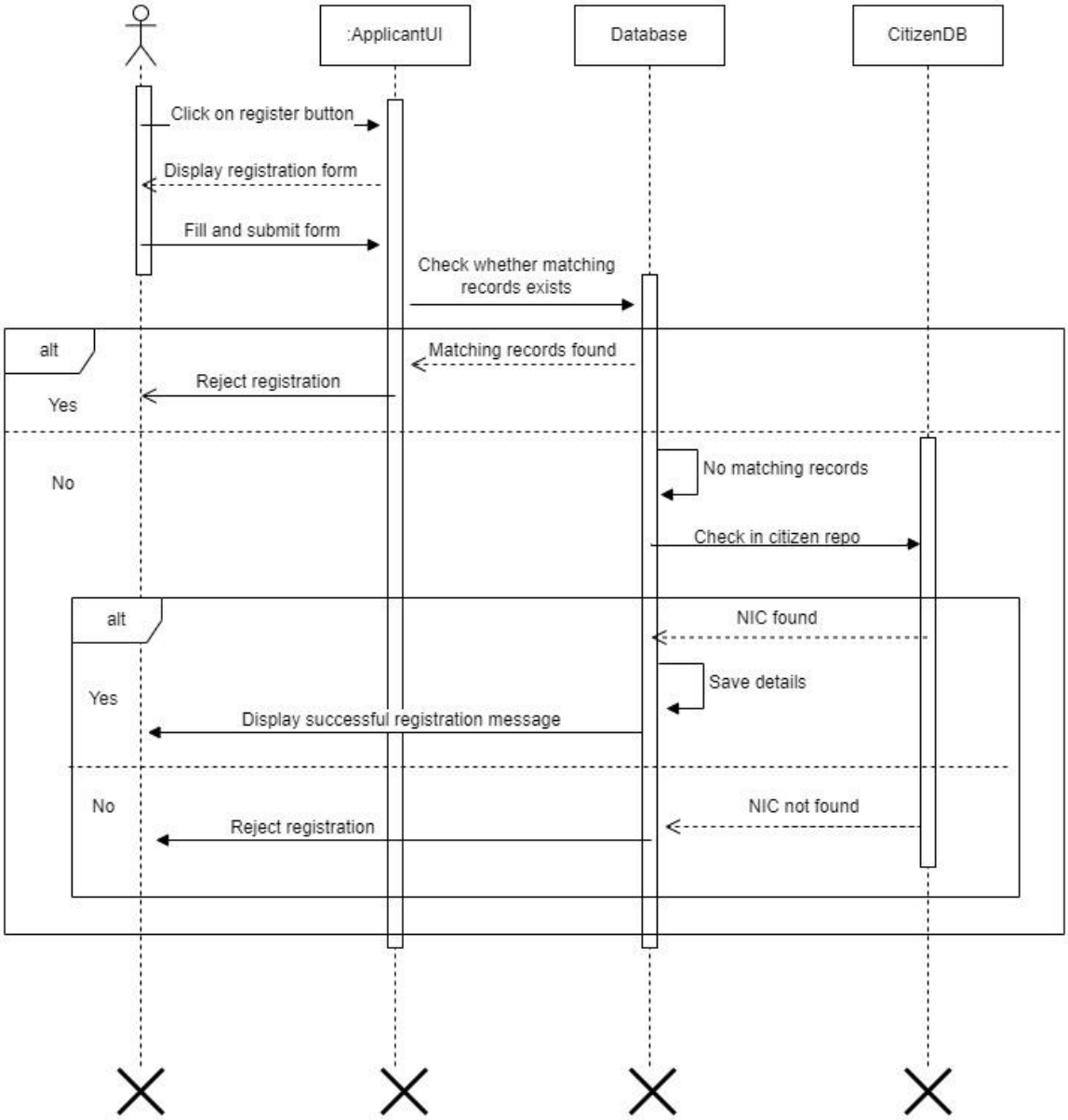
3.1.System Design Diagrams

Class diagram

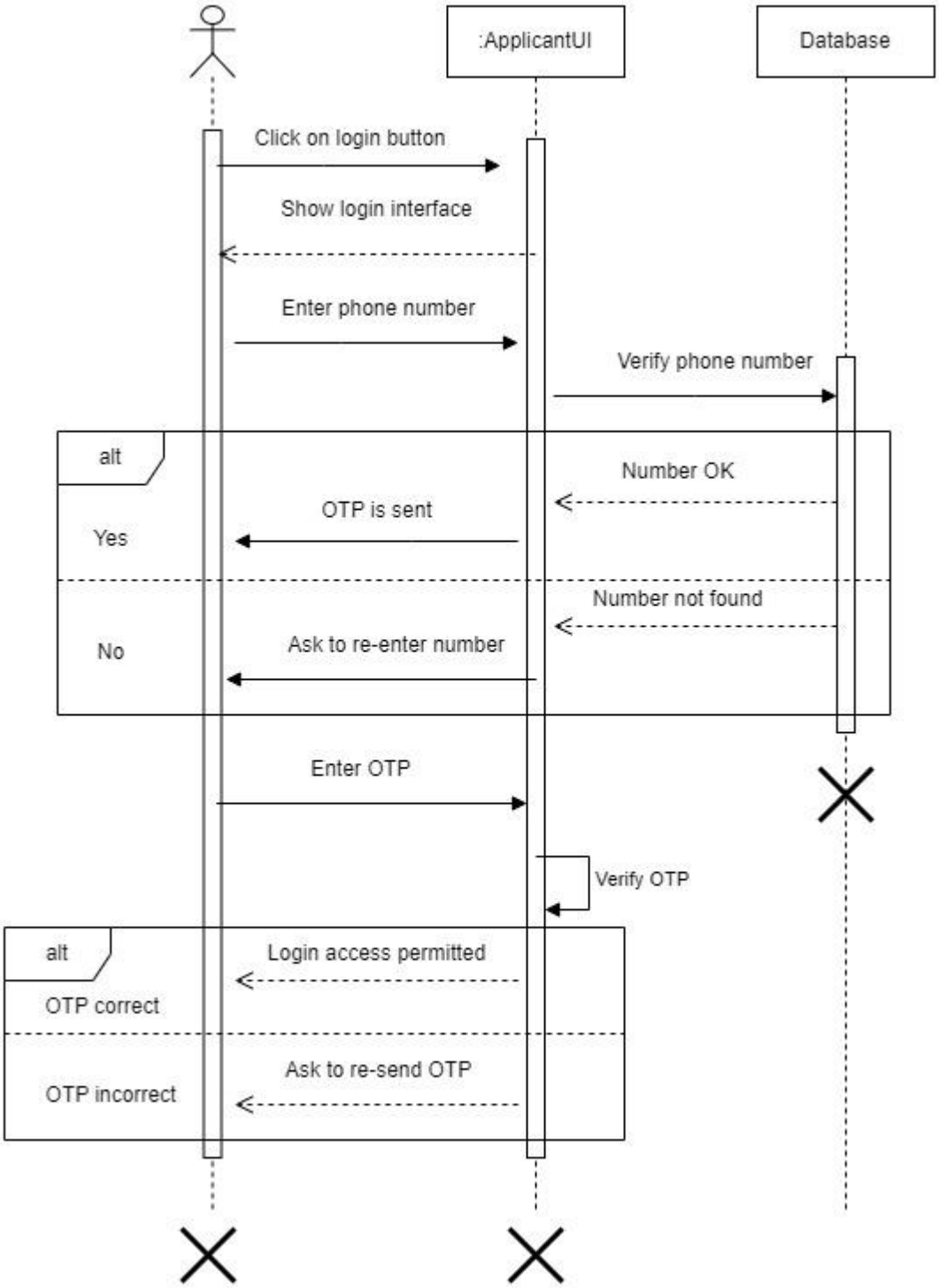


Sequence Diagrams

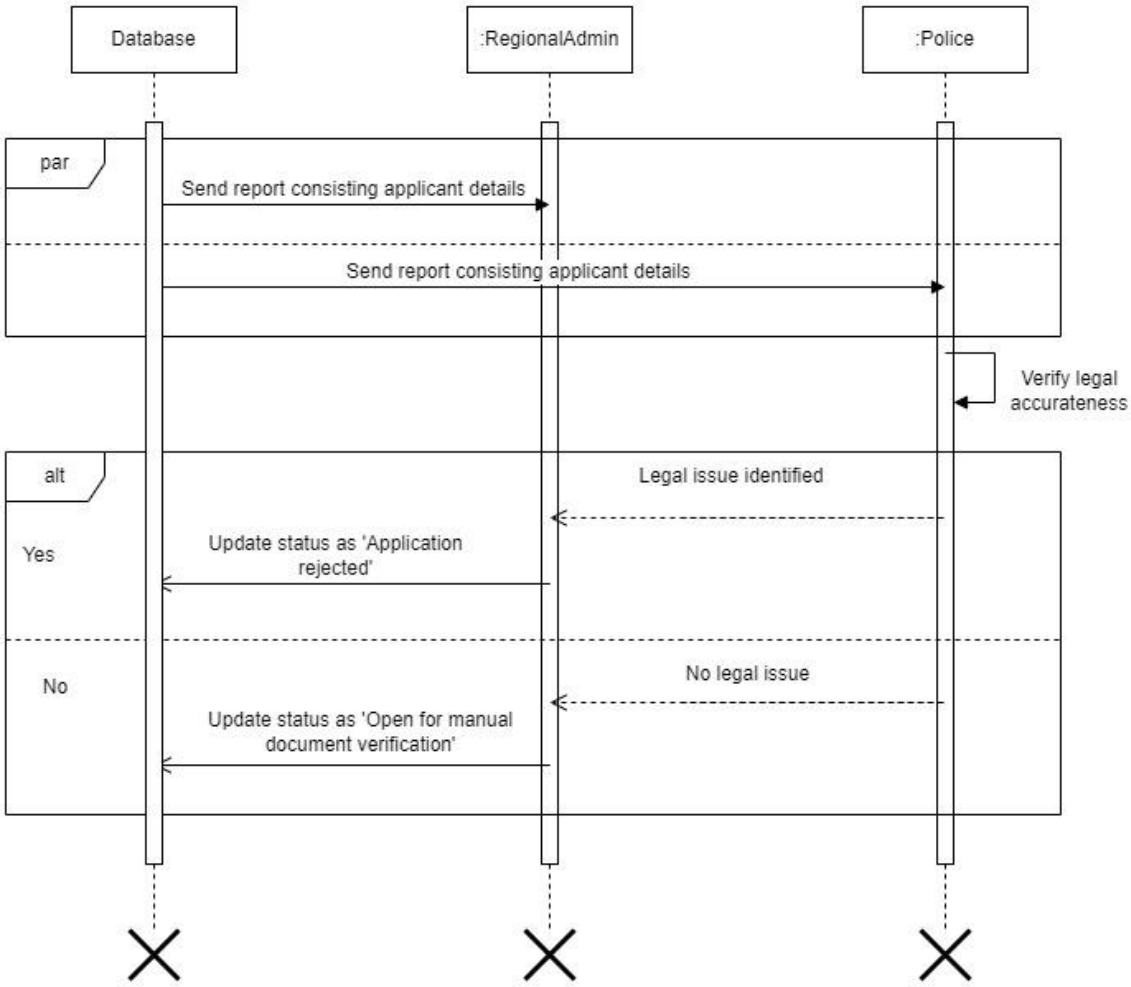
1. Registration



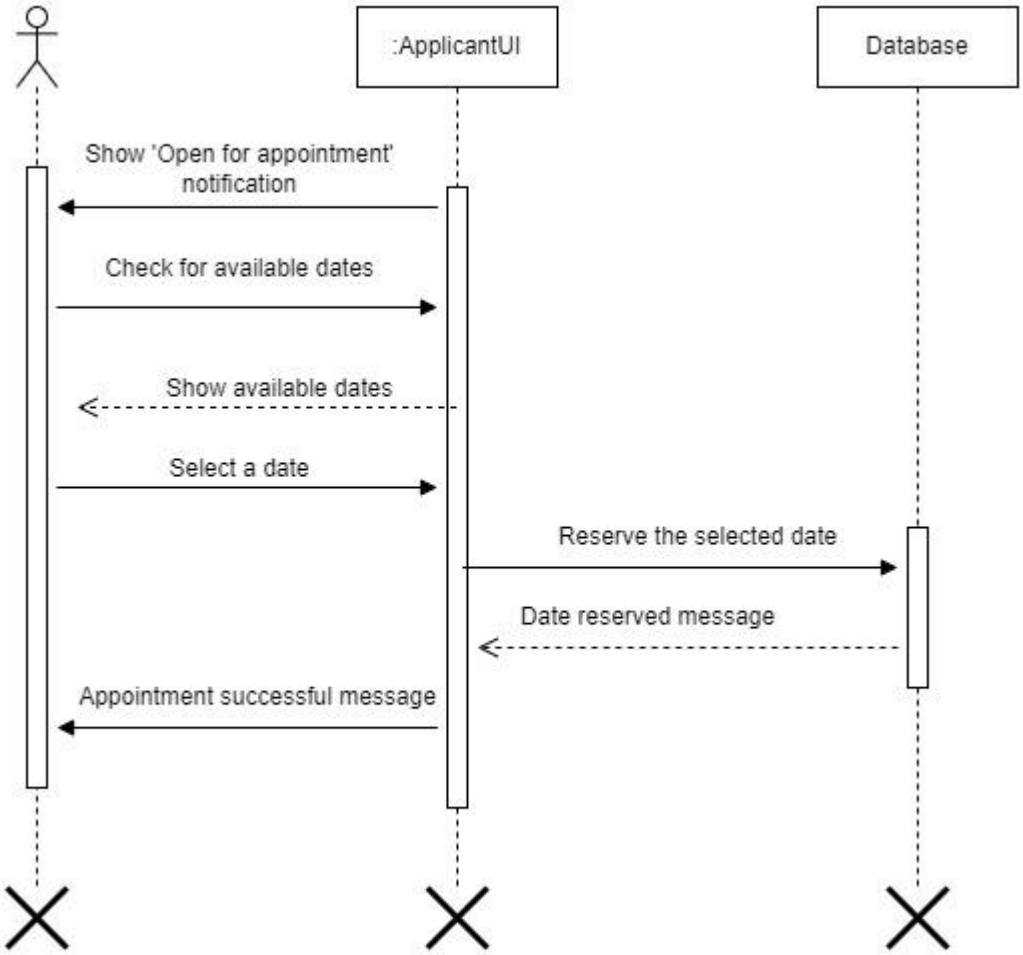
2. Login


:ApplicantUI
Database
Click on login button
Show login interface
Enter phone number
Verify phone number
alt
Number OK
OTP is sent
Yes
Number not found
No
Ask to re-enter number
Enter OTP
Verify OTP
alt
Login access permitted
OTP correct
Ask to re-send OTP
OTP incorrect

3. Verification

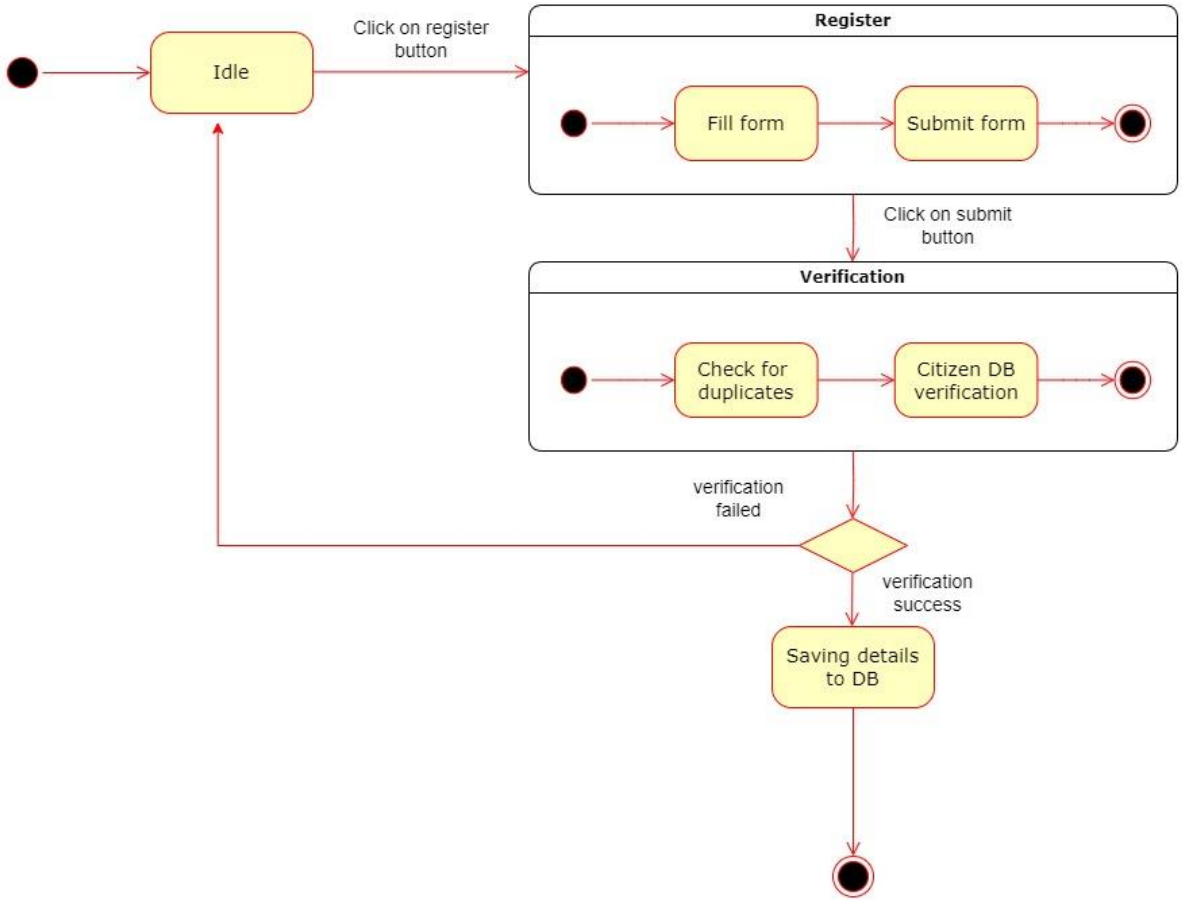


4. Make appointment

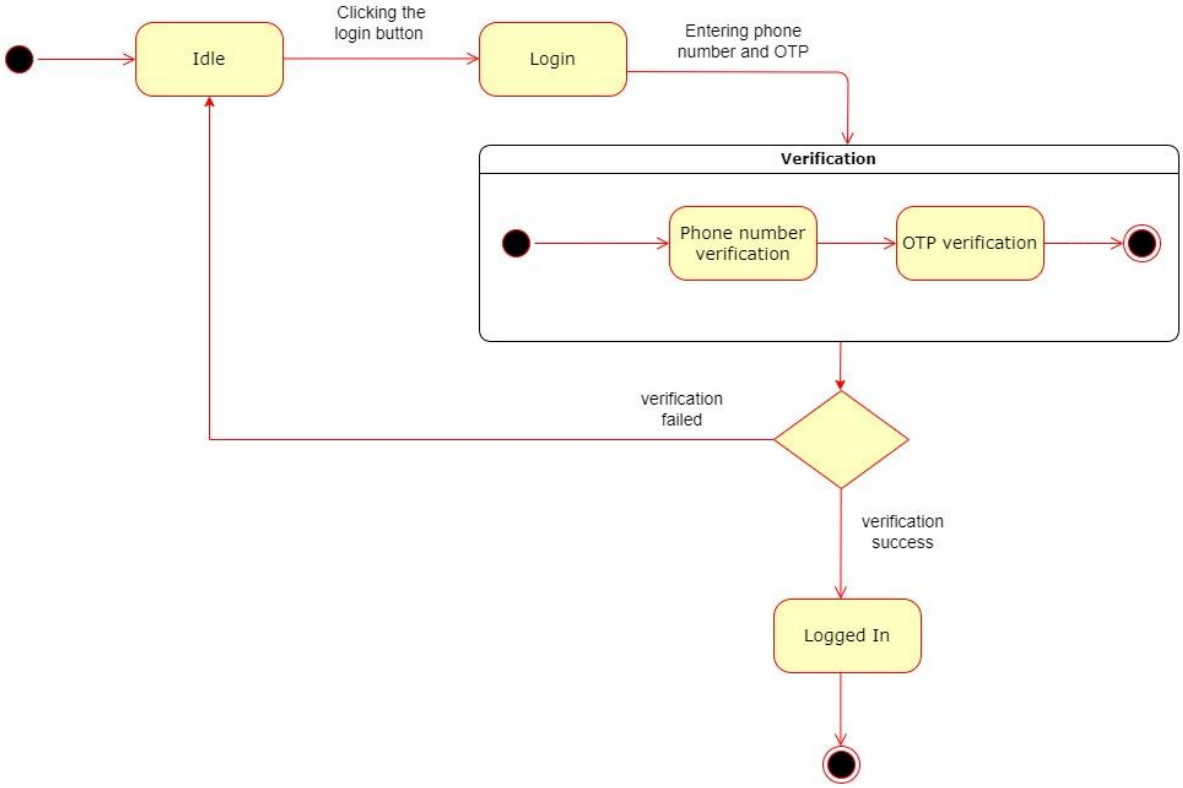


State Chart Diagrams

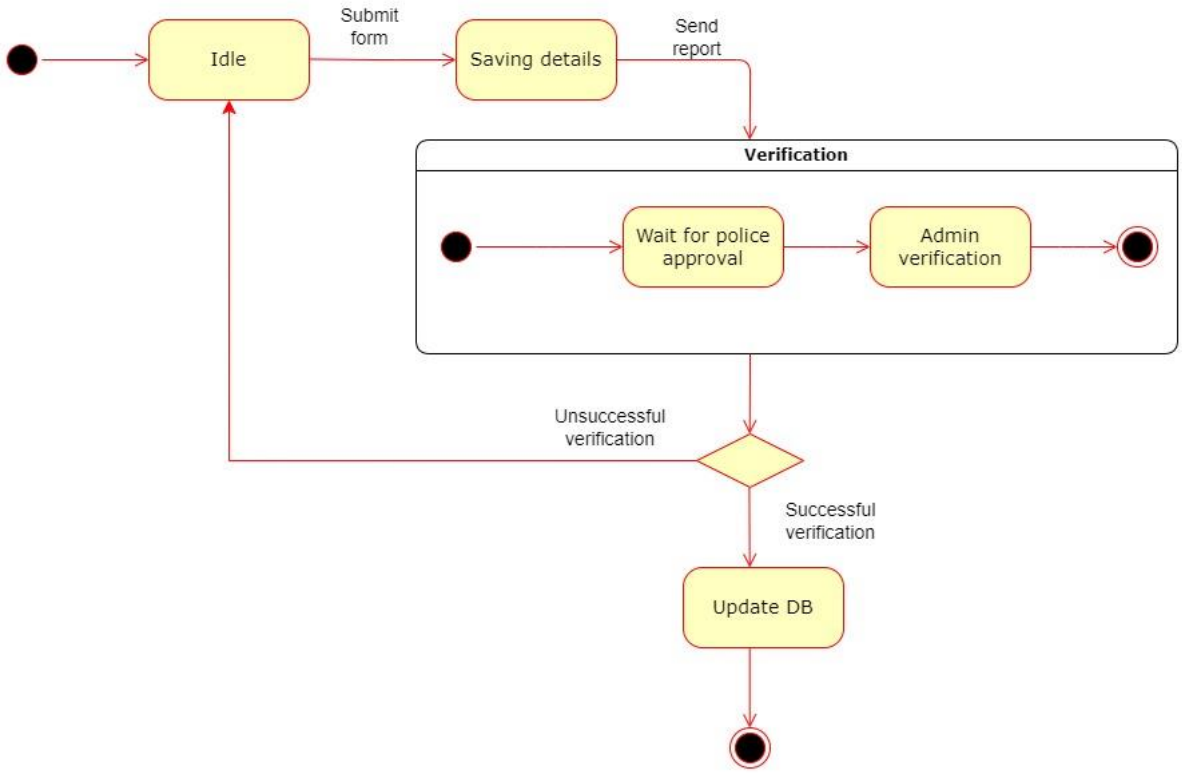
1. Registration



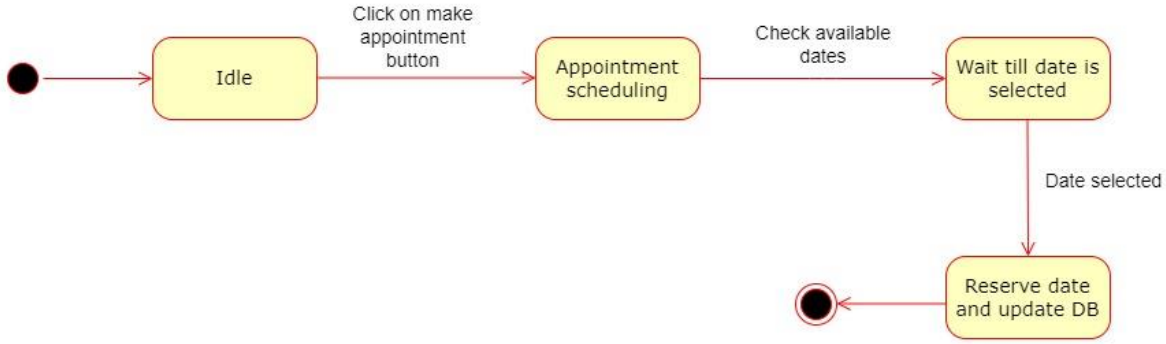
2. Login



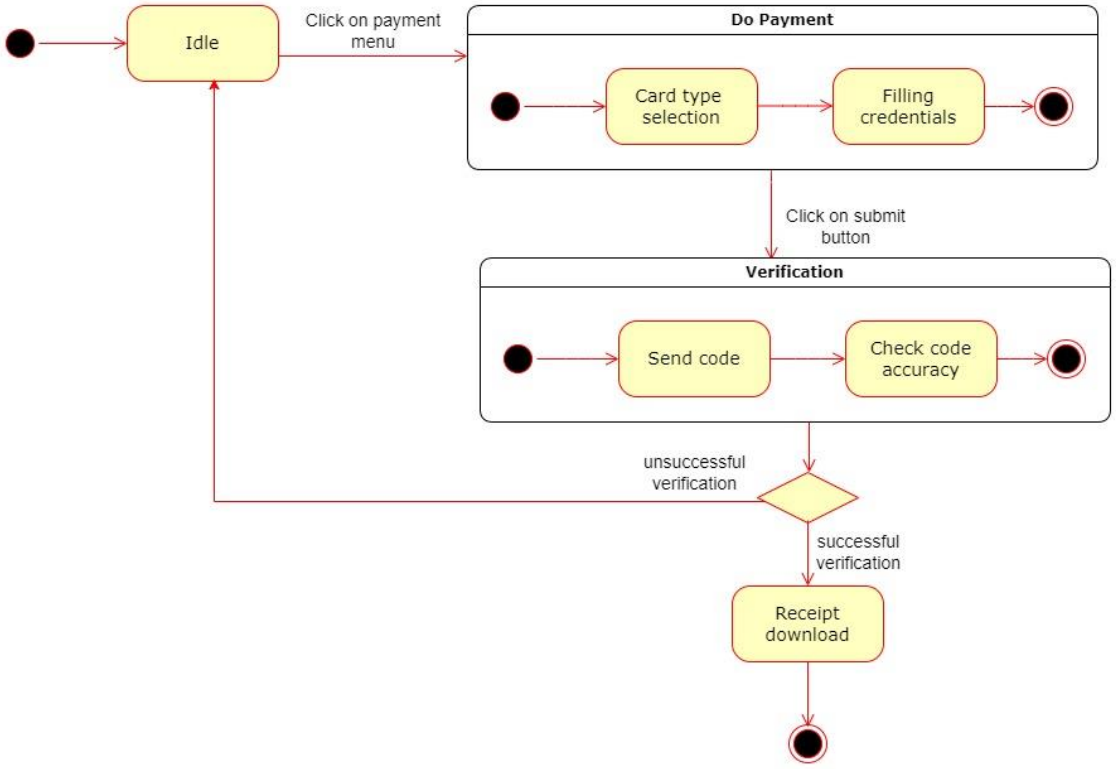
3. Verification



4. Make appointment

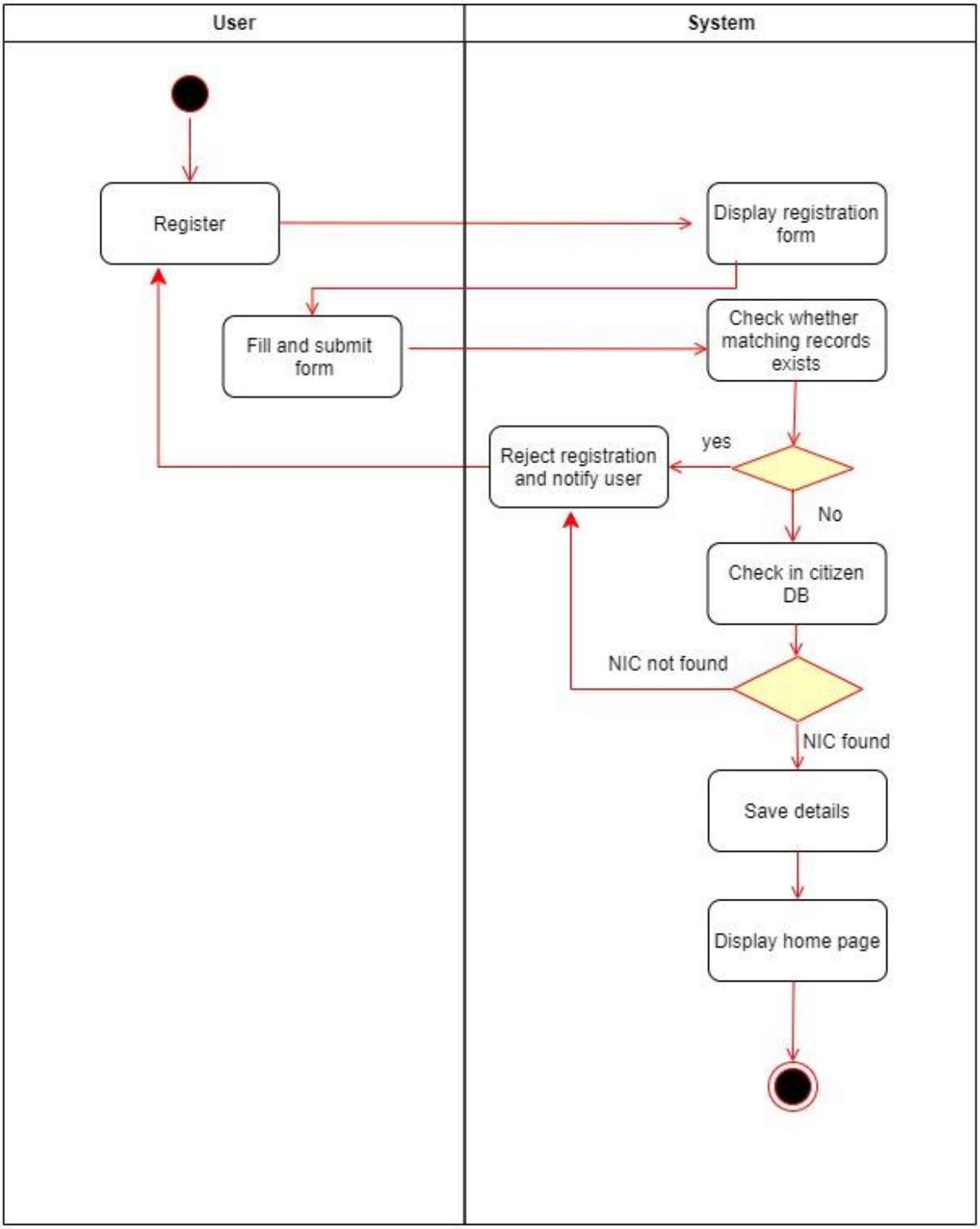


5. Online payment

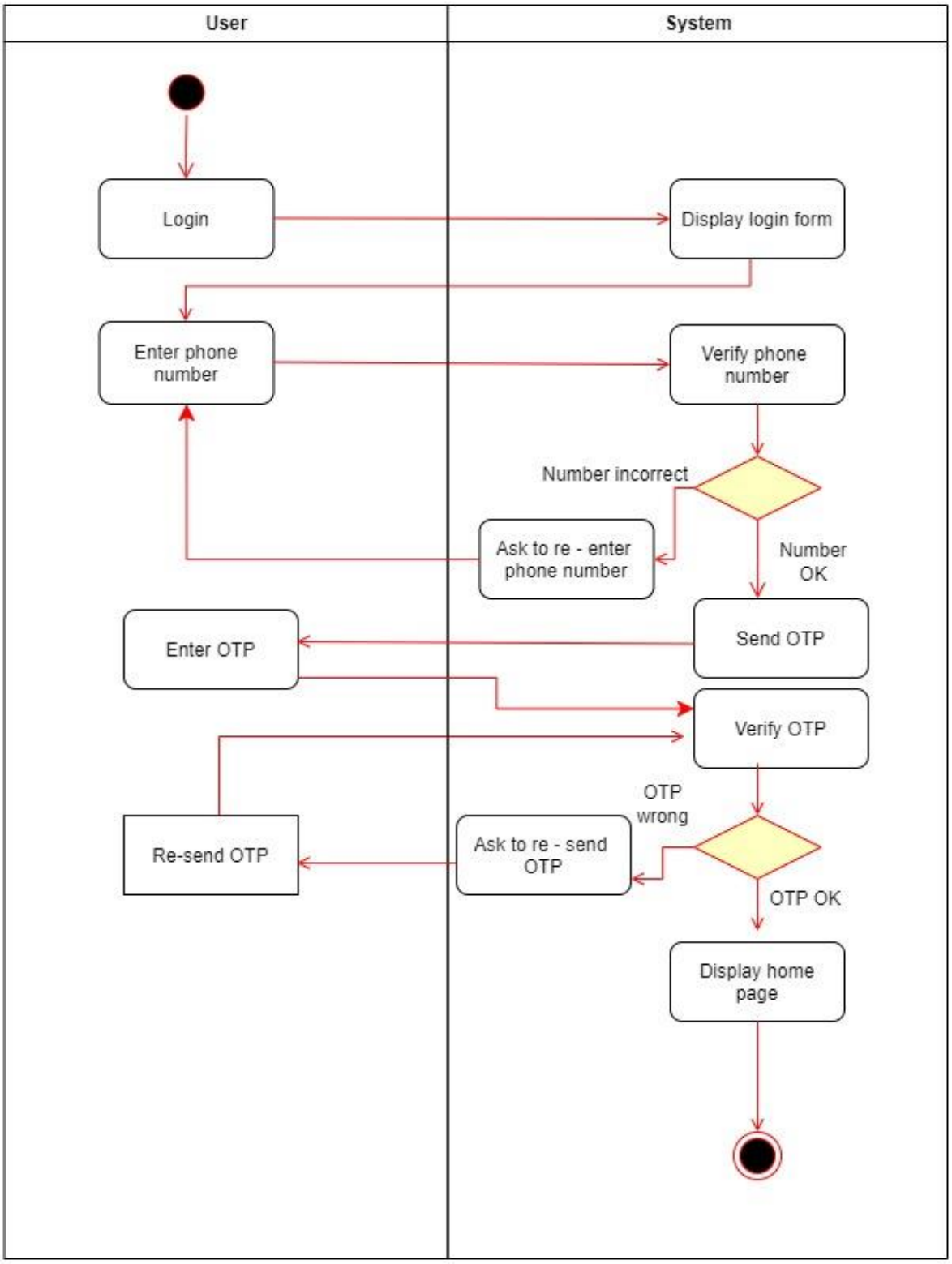


Activity Diagrams

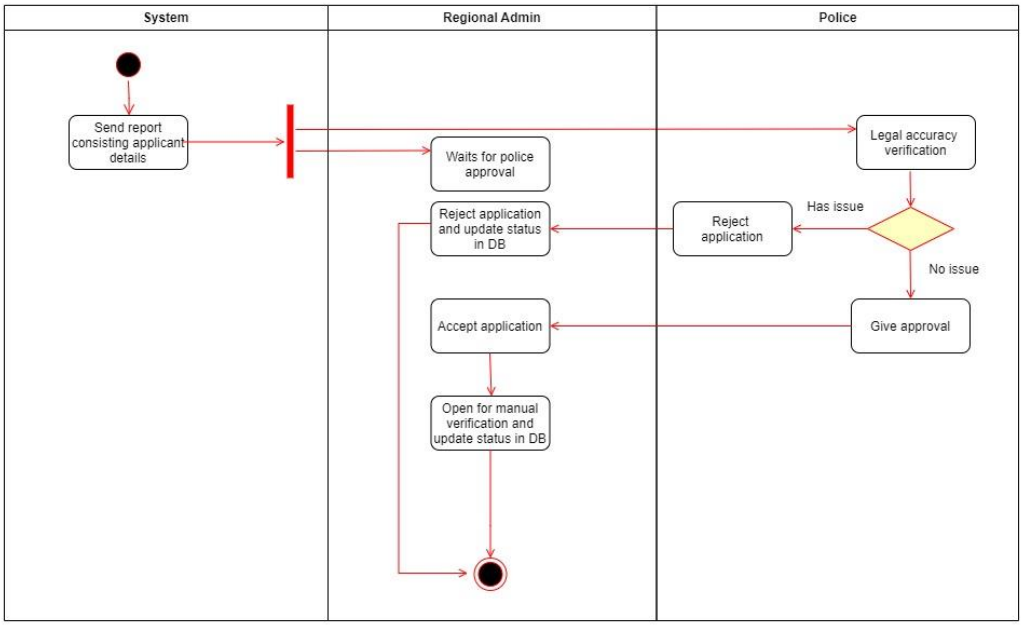
1. Registration



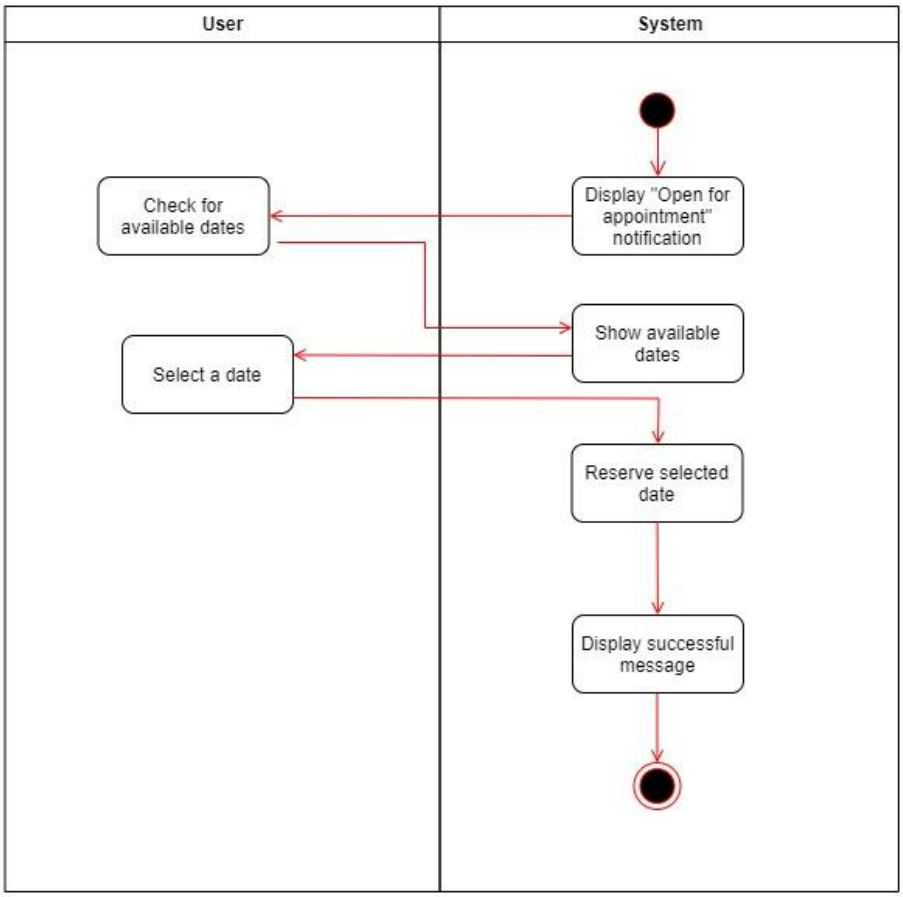
2. Login


User
System
Login
Display login form
Enter phone number
Verify phone number
Number incorrect
Ask to re - enter phone number
Number OK
Enter OTP
Send OTP
Verify OTP
OTP wrong
Re-send OTP
Ask to re - send OTP
OTP OK
Display home page

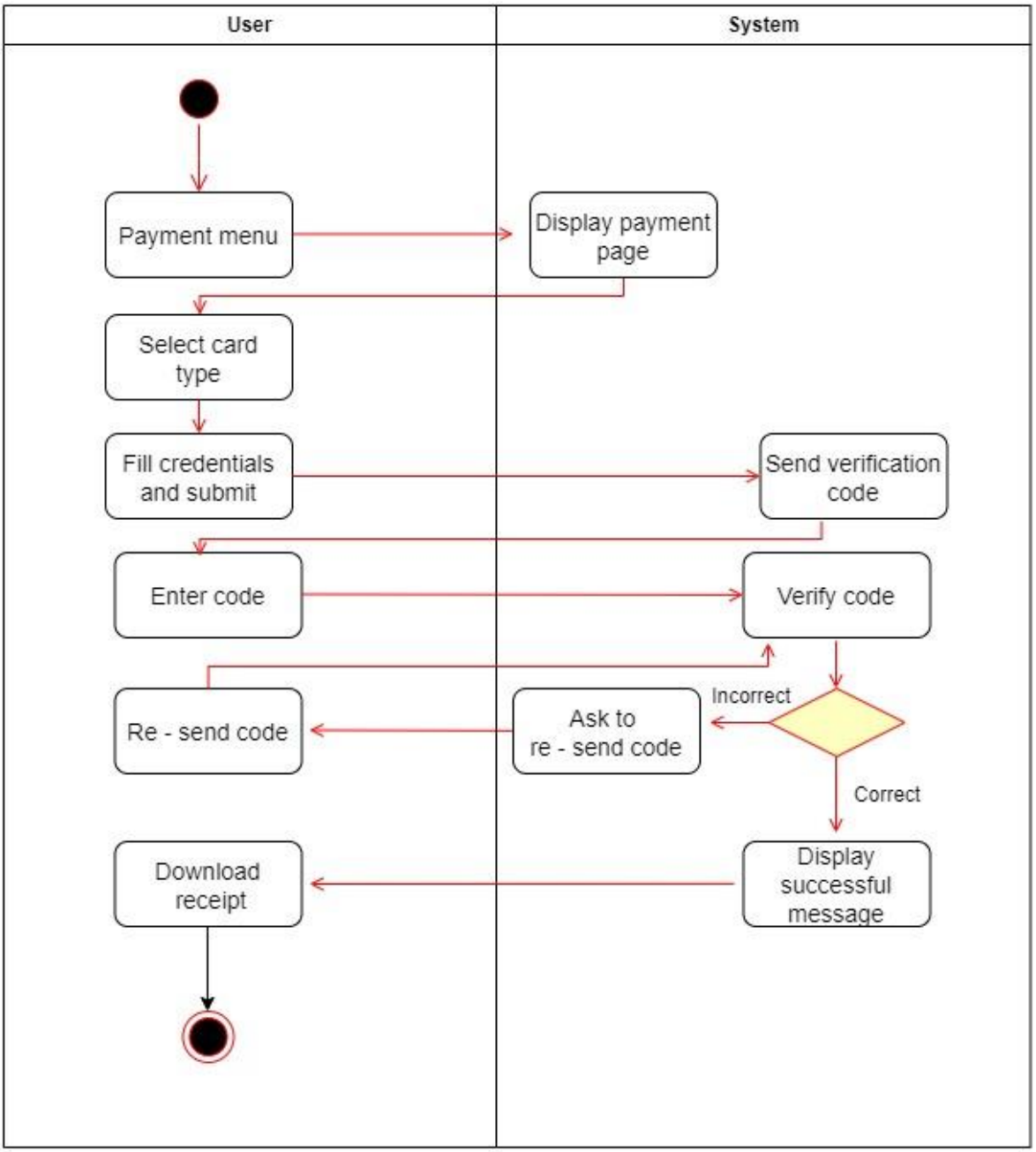
3. Verification



4. Make appointment



5. Online payment



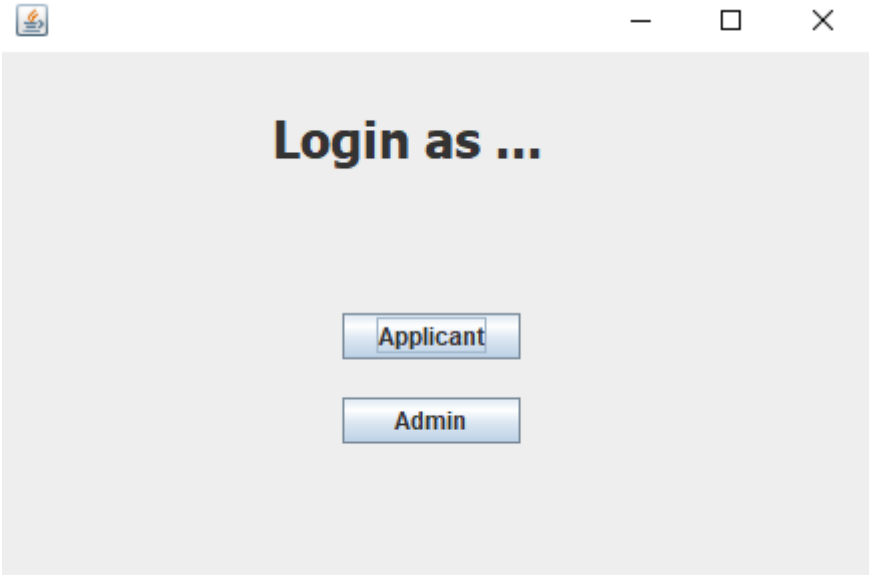
3.2. Database Design

Database	Table	Fields
PAS (system DB)	AdminLogin	adminUsername, adminPassword
	Applicant	aplNo, aplNic, aplFirstName, aplLastName, aplDOB, aplContact, aplEmail, aplAddress, aplGender, aplOccupation, aplPasPhoto, aplNicBc
	ApplicantStatus	aplNic, aplStatus
	AppointmentDetails	aplNic, aplAppoinment
	LoginAndSignUp	aplNic, aplPassword
Citizen	CitizenDetails	nic

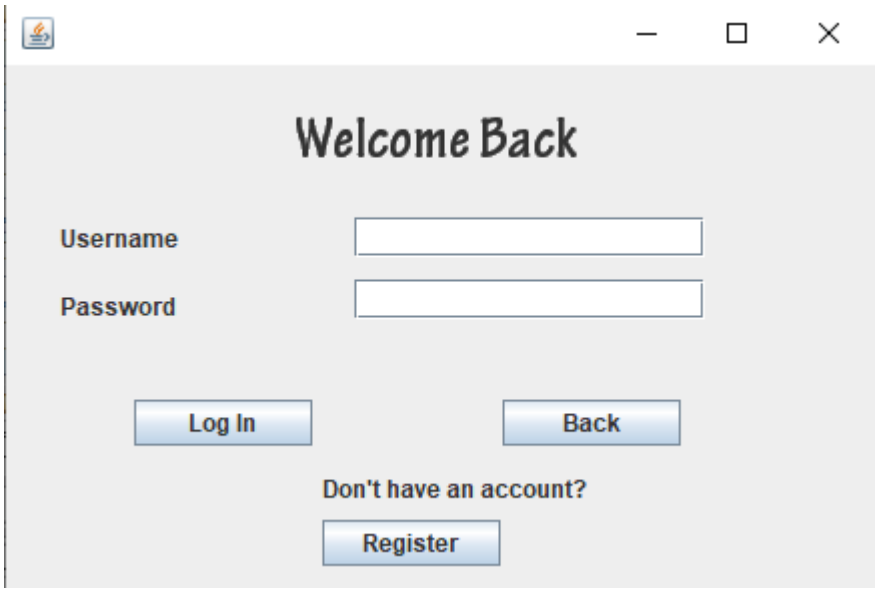
There are 2 databases as mentioned above. The PAS database is the system database where all the applicant details are stored. The system stores the applicant personal details in a separate table, the login details in a separate table, the status of the passport in a separate table and the appointment details in a separate table. The citizen database has the details of the citizens of the country.

3.3.UI Designs

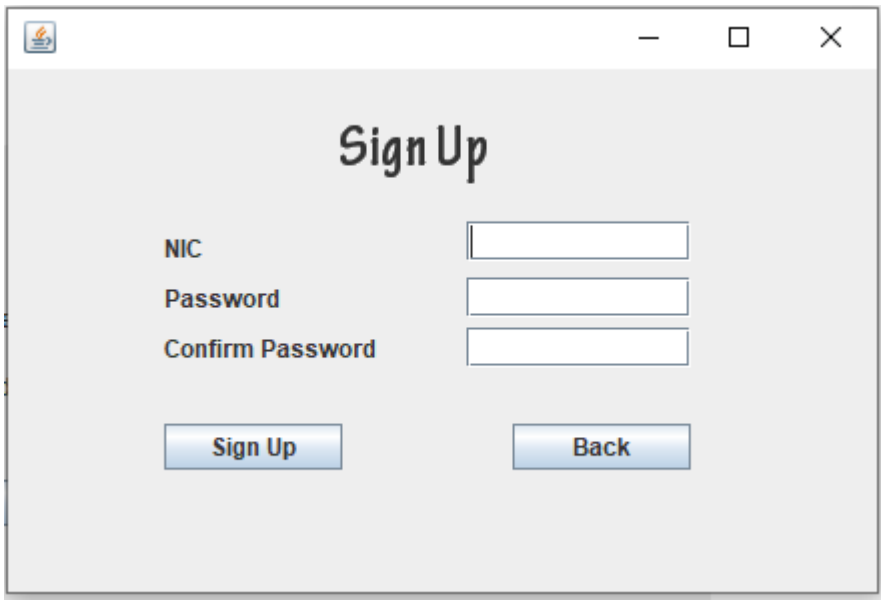
1. Home page



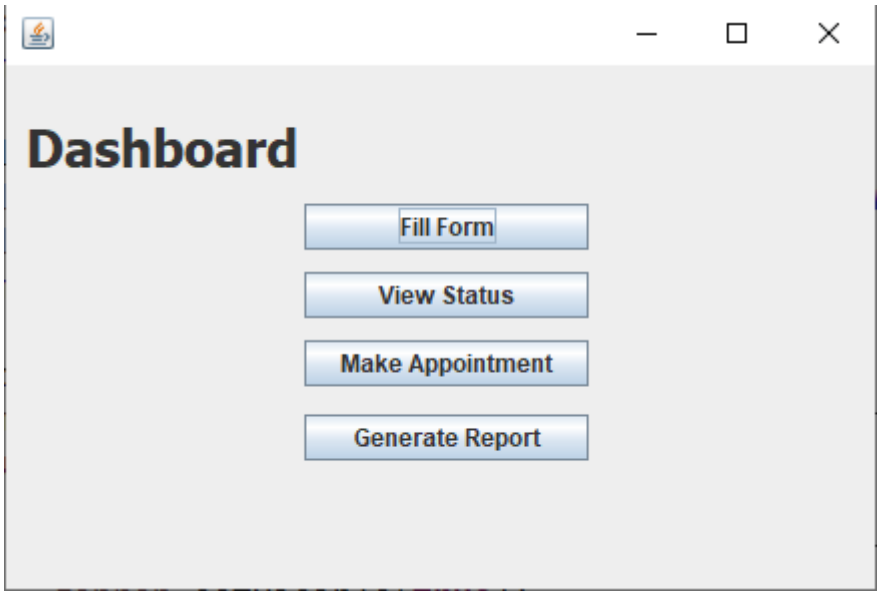
2. Applicant login



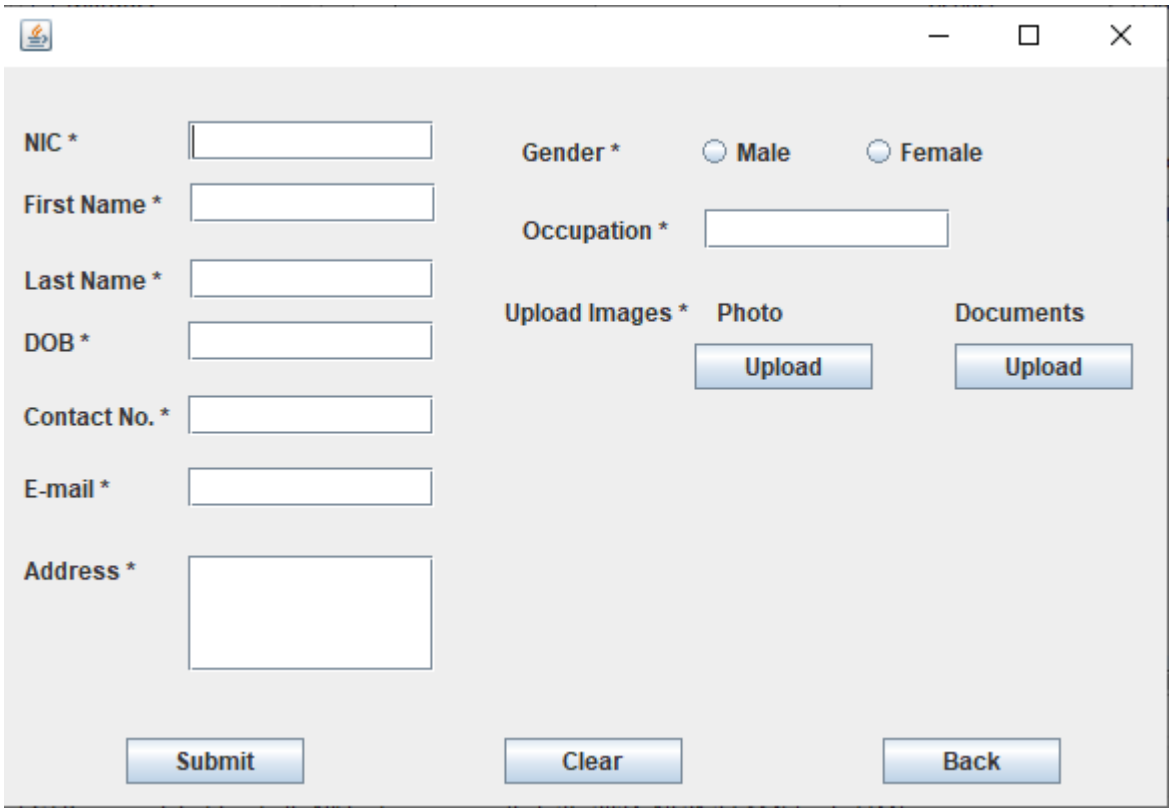
3. Applicant signup



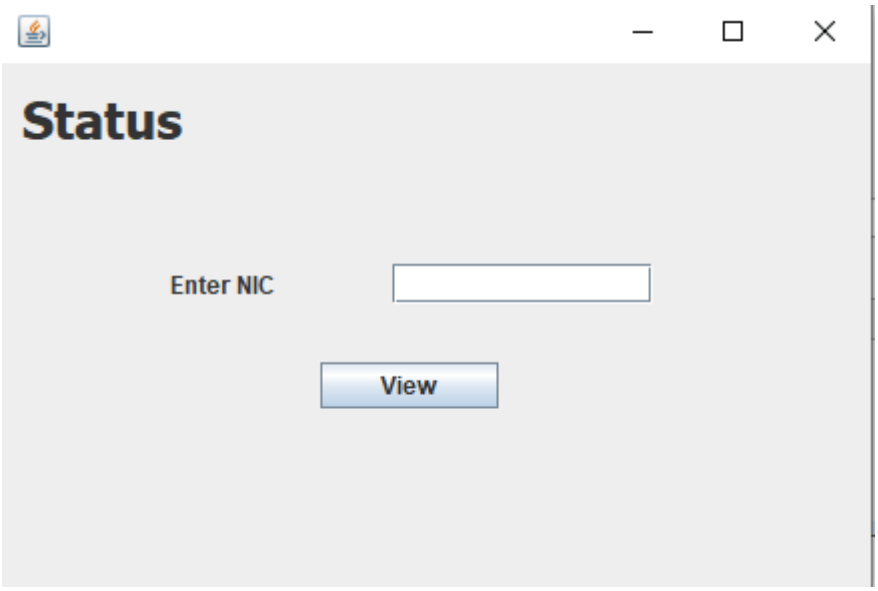
4. Applicant Dashboard



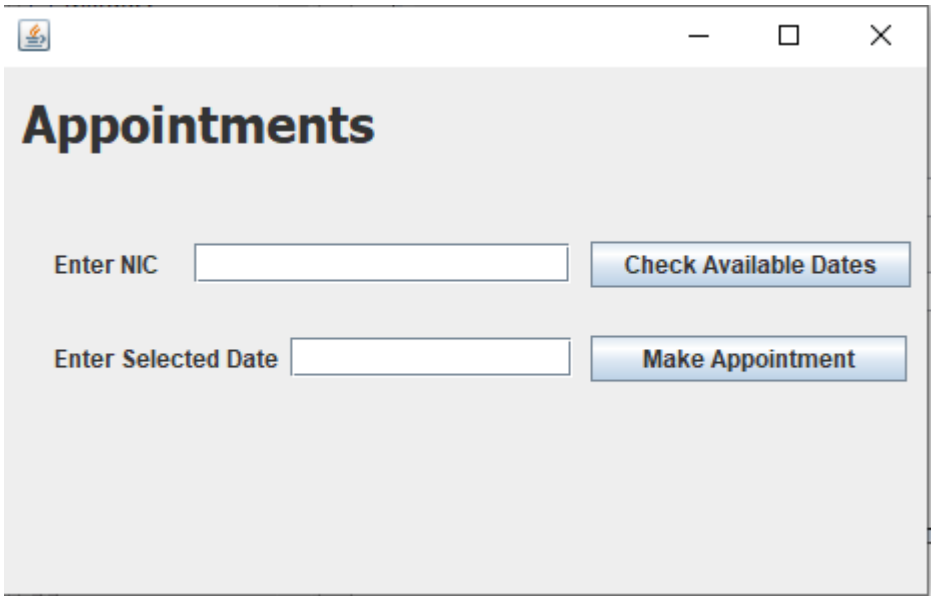
5. Application form



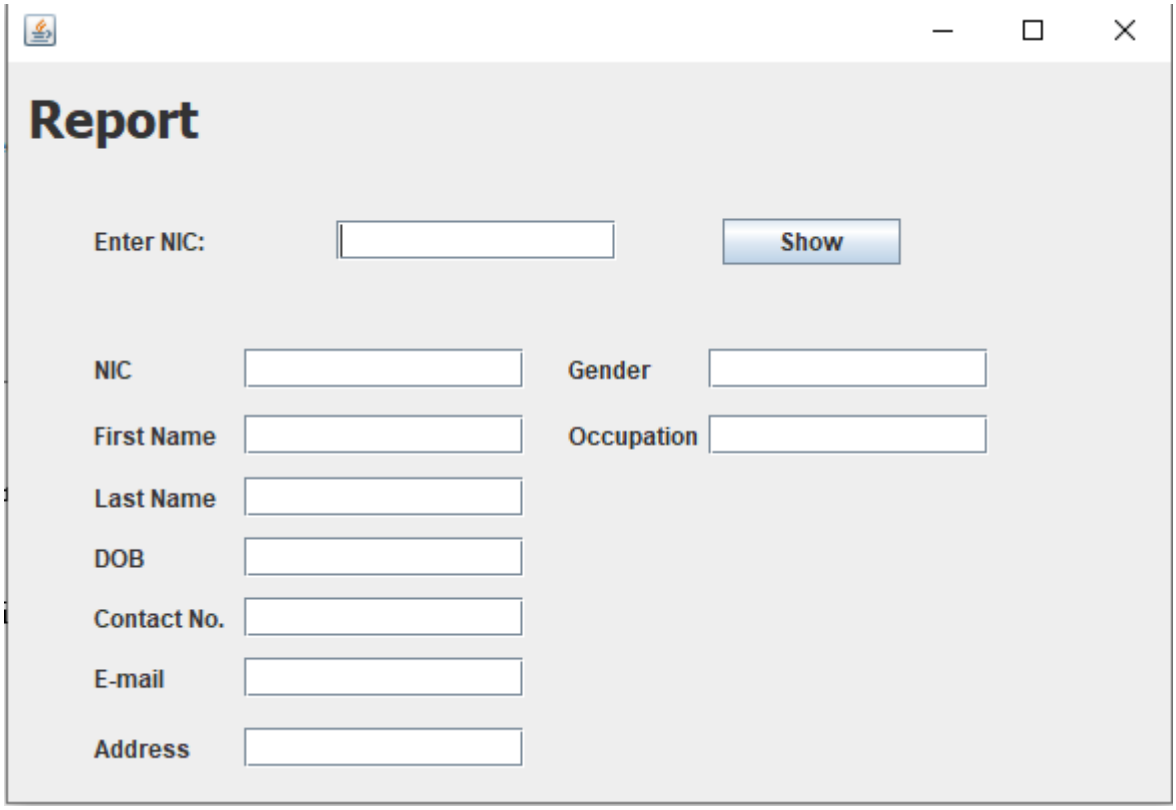
6. View Status



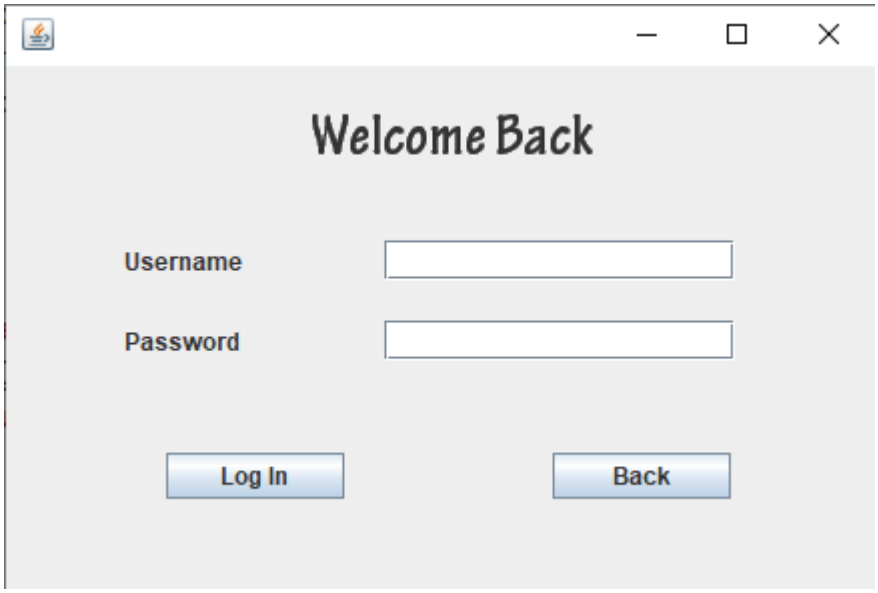
7. Make appointment



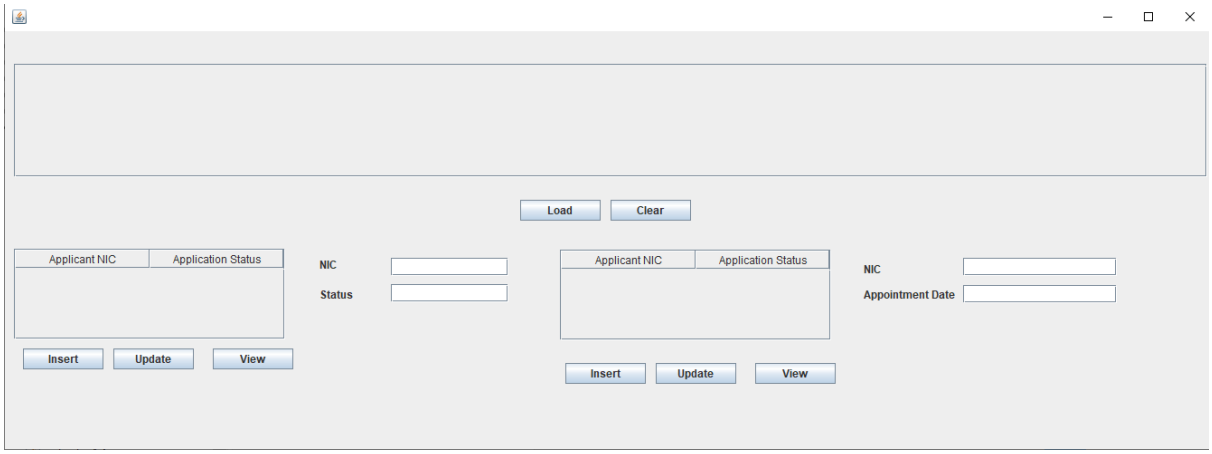
8. Generate Report



9. Admin Login



10. Admin Panel



4. Implementation

4.1.Hardware Requirements

Minimum requirements

- RAM 4GB
- Processor i3
- Storage - 128GB solid-state drive (SSD) or 500GB hard disk drive (HDD)
- Graphics - Integrated graphics or a dedicated graphics card with at least 1GB of VRAM (Video RAM).
- Display - A monitor with a minimum resolution of 1366x768 pixels.

4.2.Software Requirements

Minimum requirements

- Operating system – Windows 7
- JDK 17
- Eclipse IDE for Java Developers – Oxygen
- MySQL workbench 6.2
- MySQL Connector/J 5.7.
- Window builder 1.14
- Any web browser.

5. Testing

5.1.Test Plan

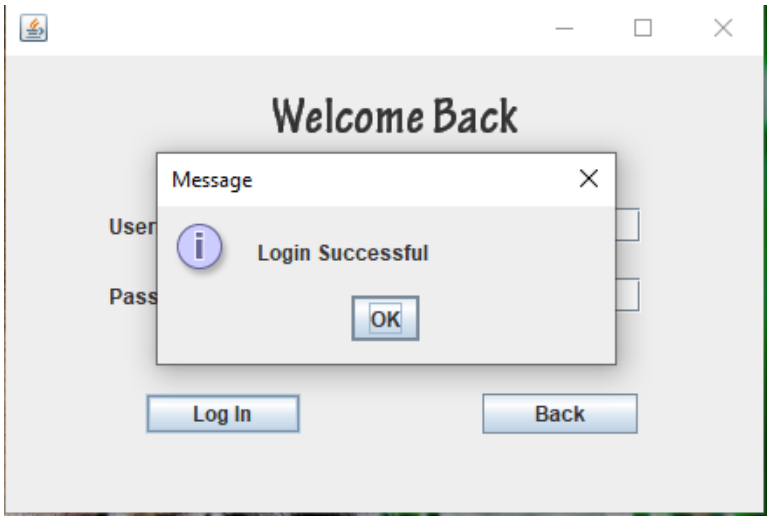
Passport Automation System	
Test plan ID	1
Brief introduction about the system	The Passport Automation System streamlines passport issuance through digitalization, minimizing manual effort and expediting resource allocation. Online forms are verified against existing records and processed manually at administrative offices. Document verification appointments are scheduled for convenience, with separate police verifications reported to administrators. Applicants track applications online, ensuring compliance before dispatching passports. The system modernizes issuance, enhancing user experience, and ensures legal compliance. If the process of issuing passports were entirely manual, it would result in significant delays, taking several months for applicants to receive their passports. With the growing number of passport applicants annually, an automated system becomes imperative to cope with the demand efficiently. This system employs various programming and database techniques to streamline the process. Due to the critical nature of national security, the system undergoes thorough verification and validation to ensure compliance.
Test objectives	1. Inserting data 2. Searching data 3. Updating data
Features to be tested	1. Inserting data • Inserting applicant login data • Inserting application data • Inserting scheduling data 2. Searching data • Searching applicant login data • Searching application data • Searching scheduling data 3. Updating data • Updating scheduling data • Updating application status data
Test environment	• Device – Laptop • Operating System – Microsoft Windows • Eclipse IDE for Java Developers • JDK 17.0.1 • JDBC Driver - mysql-connector-j-8.3.0

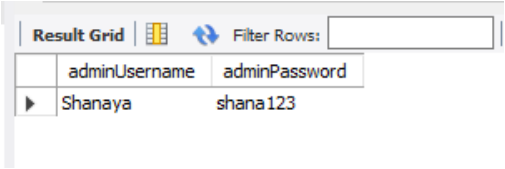
	• MySQL Server • MySQL Workbench 8.0 CE • Window builder 1.14
Test approach	Black Box Testing
Testing tasks	1. Test planning. 2. Test design. 3. Test development. 4. Test execution. 5. Test evaluation.
Test deliverables	1. Test plan. 2. Test environment. 3. Test summary. 4. Test result. 5. Test evaluation report.
Schedule	2024.02.25 8:00 PM

5.2.Test Cases

Test Case	
Test unit: Admin Login	Tester: Shanaya
Test case ID: 01	Test Type: Black Box
Test Description: Logging in by providing credentials	Test Execution Date: 2024. 02. 25
Title: Admin login details	Test Execution Time: 9:34 PM

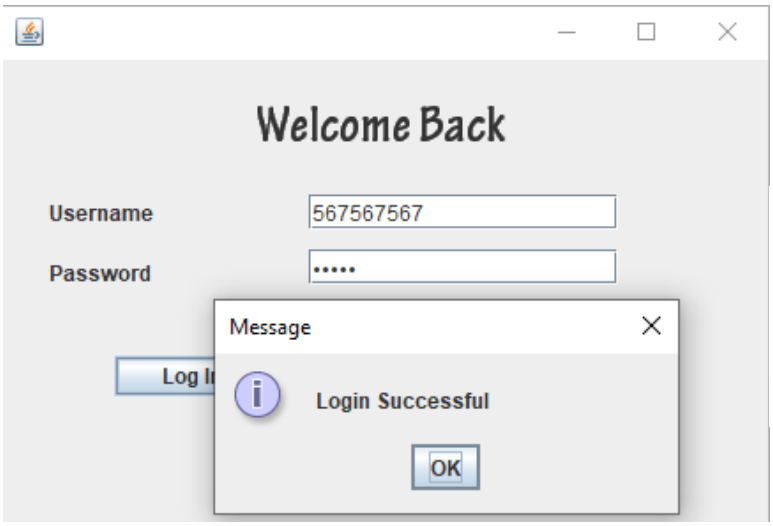
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Add login data of admin.	01	Admin username, Admin password	Show successfully log in pop-up.	Login success pop-up.	Pass

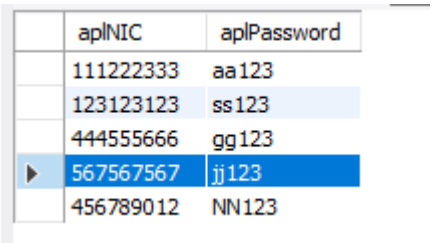




Test Case	
Test unit: Applicant Login	Tester: Shanaya
Test case ID: 02	Test Type: Black Box
Test Description: Logging in by providing credentials	Test Execution Date: 2024. 02. 25
Title: Applicant login details	Test Execution Time: 9:35 PM

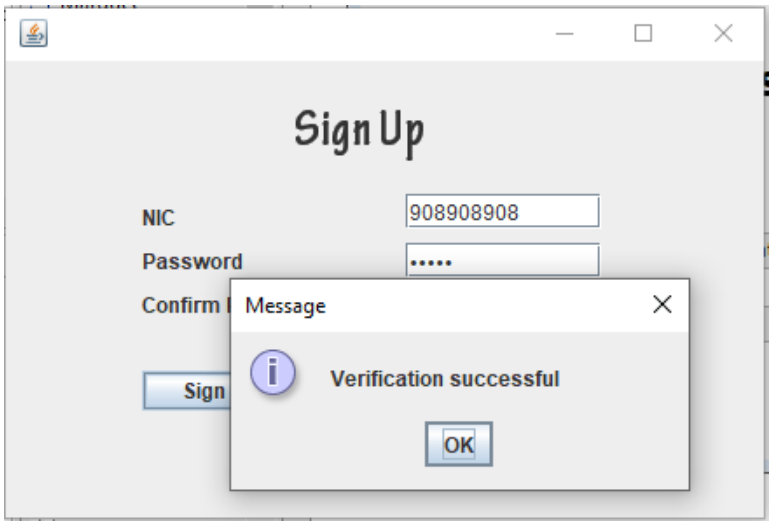
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Add login data of an applicant.	02	applicant username, applicant password	Show successfully log in pop-up.	Login success pop-up.	Pass

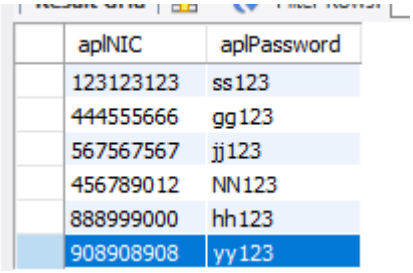




Test Case	
Test unit: Applicant Sign up	Tester: Shanaya
Test case ID: 03	Test Type: Black Box
Test Description: Registering by providing necessary details	Test Execution Date: 2024. 02. 25
Title: Applicant sign up details	Test Execution Time: 9:37 PM

Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Add sign up data of an applicant.	03	applicant username, applicant password, confirm password	Show verification successful in pop-up.	Verification success pop-up.	Pass



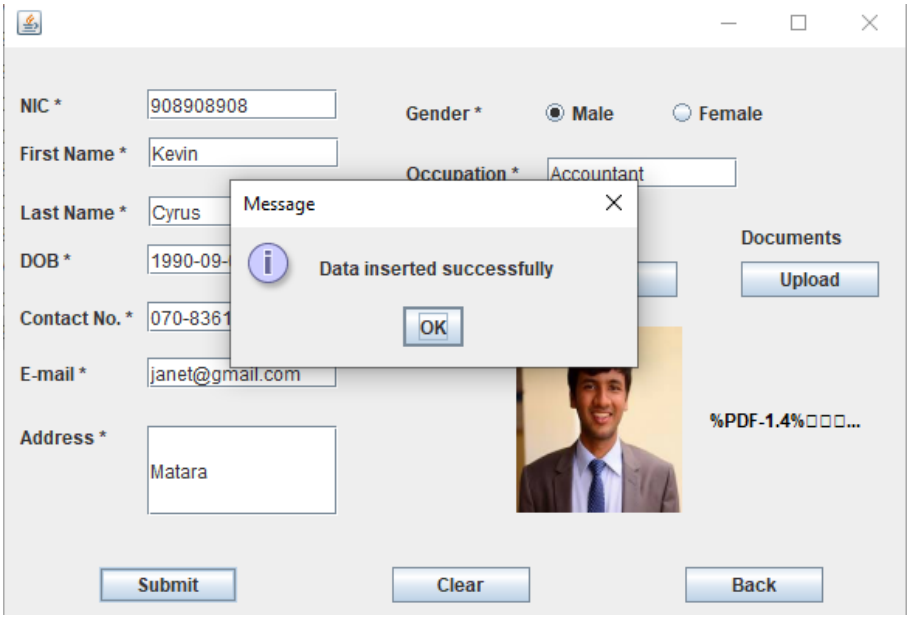


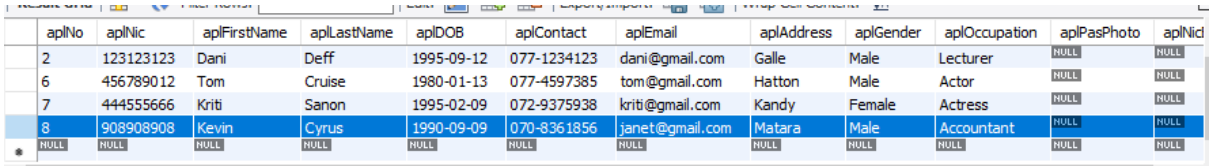
aplNIC	aplPassword
123123123	ss123
444555666	gg123
567567567	jj123
456789012	NN123
888999000	hh123
908908908	yy123

Test Case	
Test unit: Filling application	Tester: Shanaya
Test case ID: 04	Test Type: Black Box
Test Description: Filling the form with required details	Test Execution Date: 2024. 02. 25
Title: Application form	Test Execution Time: 9:40 PM

Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Add required form data of applicant.	04	Nic, first name, last name, dob, email, contact, address, gender, occupation,	Show data inserted successfully in pop-up.	Data insertion success pop-up.	Pass

			passport size image, scanned documents			

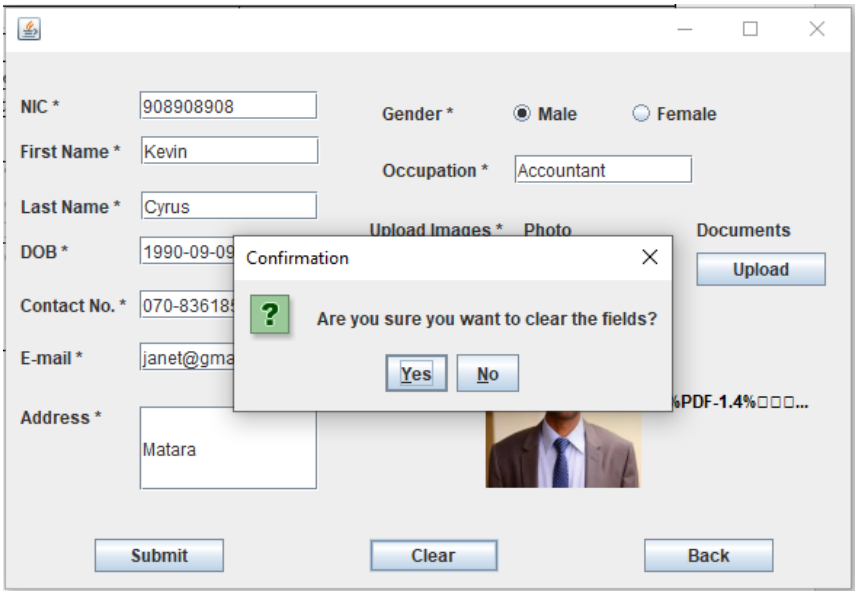




aplNo	aplNic	aplFirstName	aplLastName	aplDOB	aplContact	aplEmail	aplAddress	aplGender	aplOccupation	aplPasPhoto	aplNic
2	123123123	Dani	Deff	1995-09-12	077-1234123	dani@gmail.com	Galle	Male	Lecturer	NULL	NULL
6	456789012	Tom	Cruise	1980-01-13	077-4597385	tom@gmail.com	Hatton	Male	Actor	NULL	NULL
7	444555666	Kriti	Sanon	1995-02-09	072-9375938	kriti@gmail.com	Kandy	Female	Actress	NULL	NULL
8	908908908	Kevin	Cyrus	1990-09-09	070-8361856	janet@gmail.com	Matara	Male	Accountant	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Test Case	
Test unit: Clear form data	Tester: Shanaya
Test case ID: 05	Test Type: Black Box
Test Description: Clearing the form data	Test Execution Date: 2024. 02. 25
Title: Clear Details	Test Execution Time: 9:45 PM

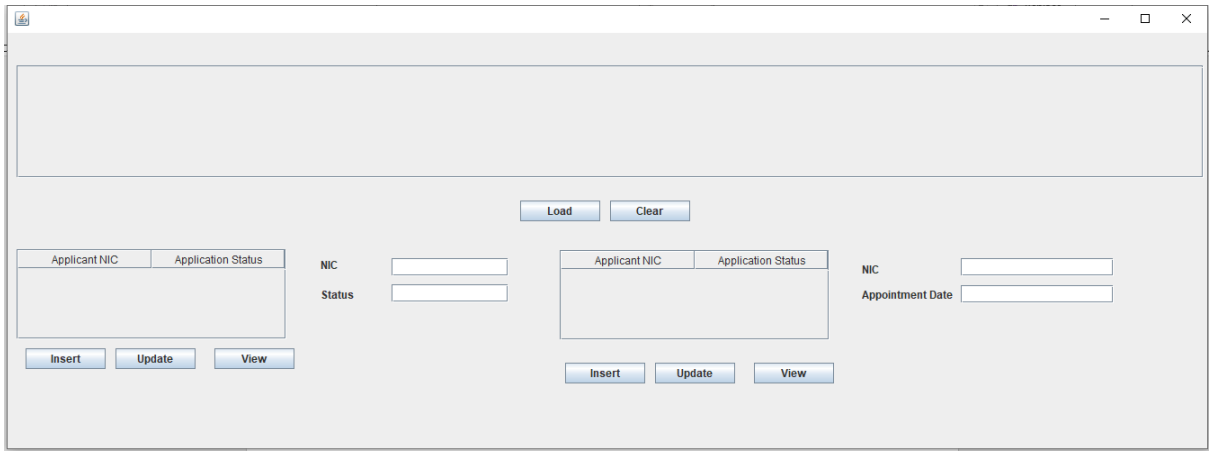
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Click on the “Clear” button	05	Click “Clear” button	Show confirm dialog.	Confirm dialog pop-up.	Pass



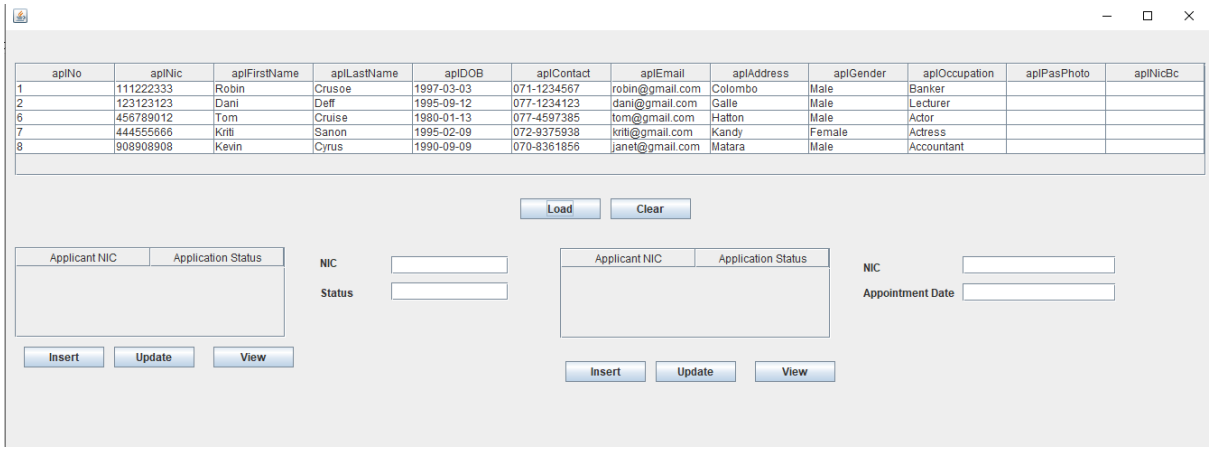
Test Case	
Test unit: Load data	Tester: Shanaya
Test case ID: 06	Test Type: Black Box
Test Description: Loading data to the table	Test Execution Date: 2024. 02. 25
Title: Applicant details	Test Execution Time: 9:46 PM

Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Click on the "Clear" button	06	Click "Load" button	Loading data to table.	Loaded table.	Pass

Before

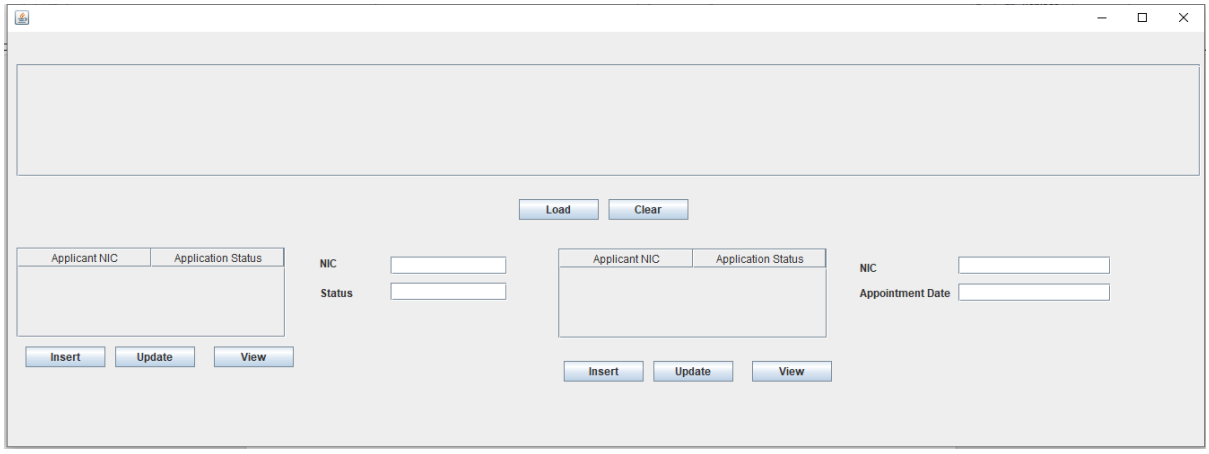


After



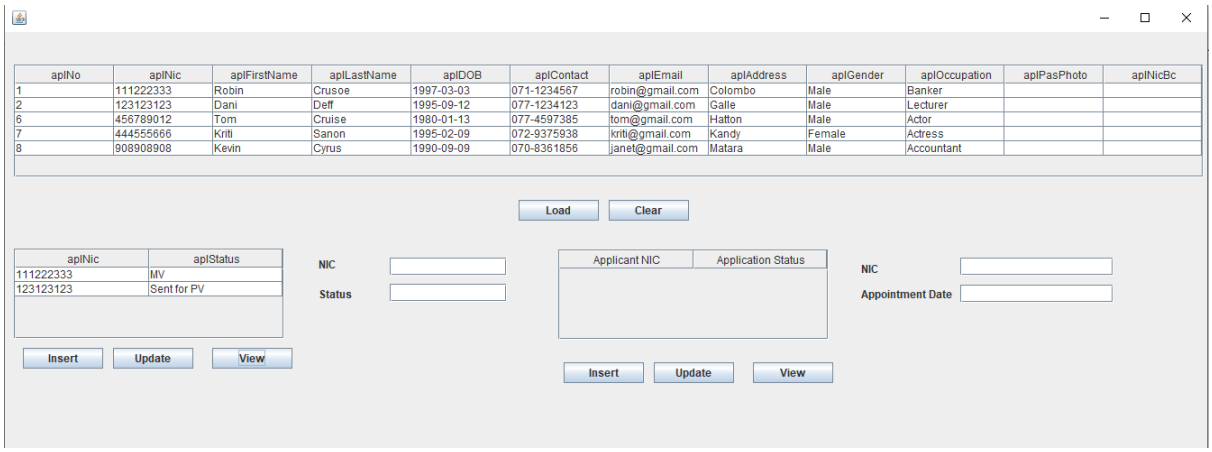
Test Case	
Test unit: Clear data	Tester: Shanaya
Test case ID: 07	Test Type: Black Box
Test Description: Clearing the table data	Test Execution Date: 2024. 02. 25
Title: Applicant table details	Test Execution Time: 9:46 PM

Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Click on the “Clear” button	07	Click “Clear” button	Empty table.	Empty table	Pass



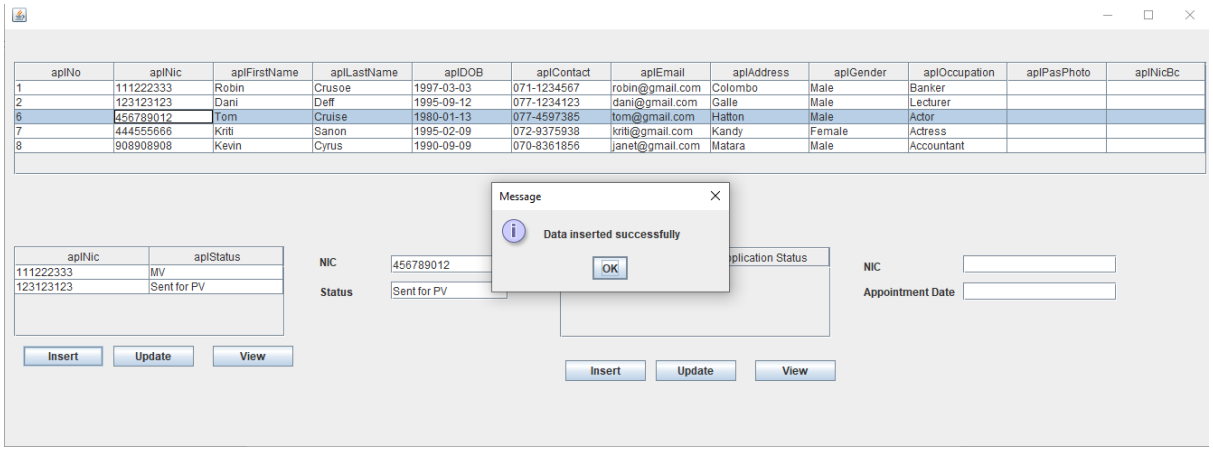
Test Case	
Test unit: View applicant status	Tester: Shanaya
Test case ID: 08	Test Type: Black Box
Test Description: Retrieving applicant status from the status table	Test Execution Date: 2024. 02. 25
Title: Applicant status	Test Execution Time: 9:48 PM

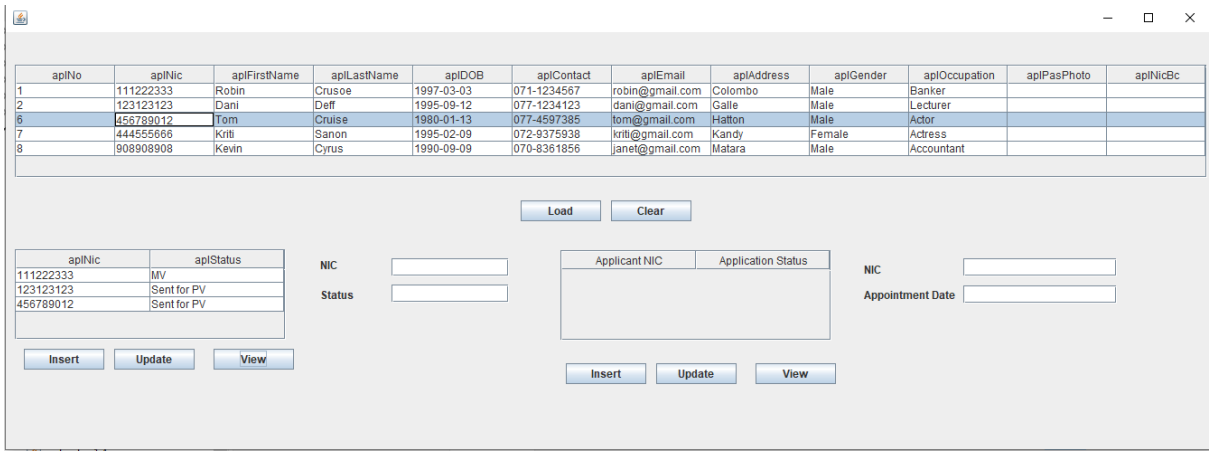
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Click on the "View" button	08	Click "View" button	Loading data to table.	Loaded table.	Pass



Test Case	
Test unit: Insert applicant status	Tester: Shanaya
Test case ID: 09	Test Type: Black Box
Test Description: Inserting applicant status to the status table	Test Execution Date: 2024. 02. 25
Title: Applicant status details	Test Execution Time: 9:47 PM

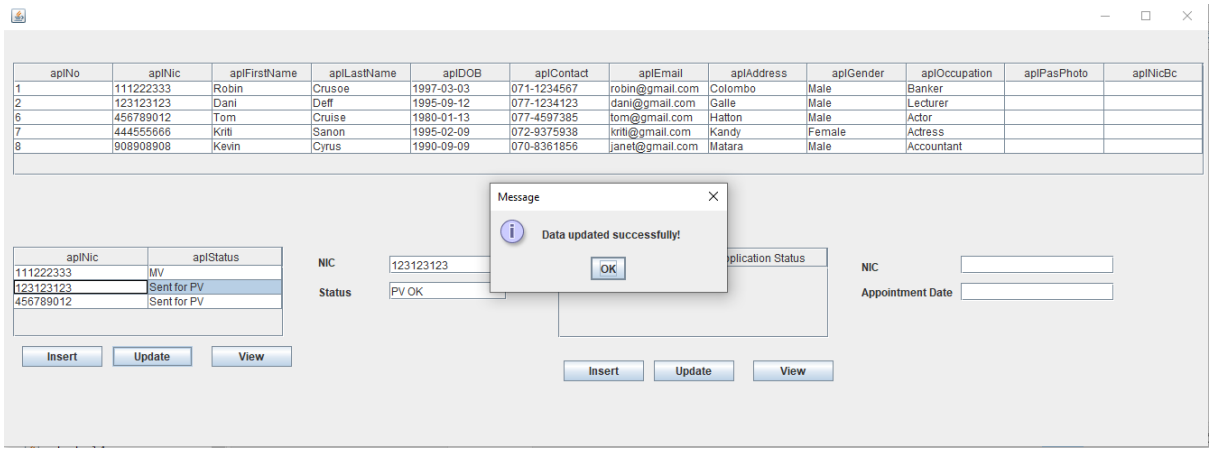
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Add status of an applicant	09	Applicant NIC, Applicant status	Show data inserted successfully pop up and filled table.	Success pop up and filled table	Pass

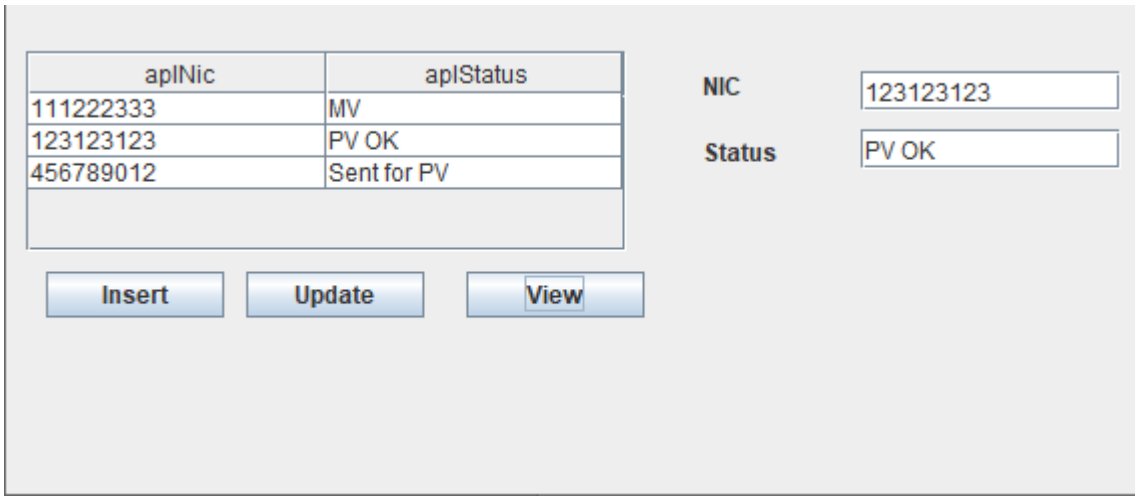




Test Case	
Test unit: Update applicant status	Tester: Shanaya
Test case ID: 10	Test Type: Black Box
Test Description: Updating applicant status to the status table	Test Execution Date: 2024. 02. 25
Title: Applicant status details	Test Execution Time: 9:48 PM

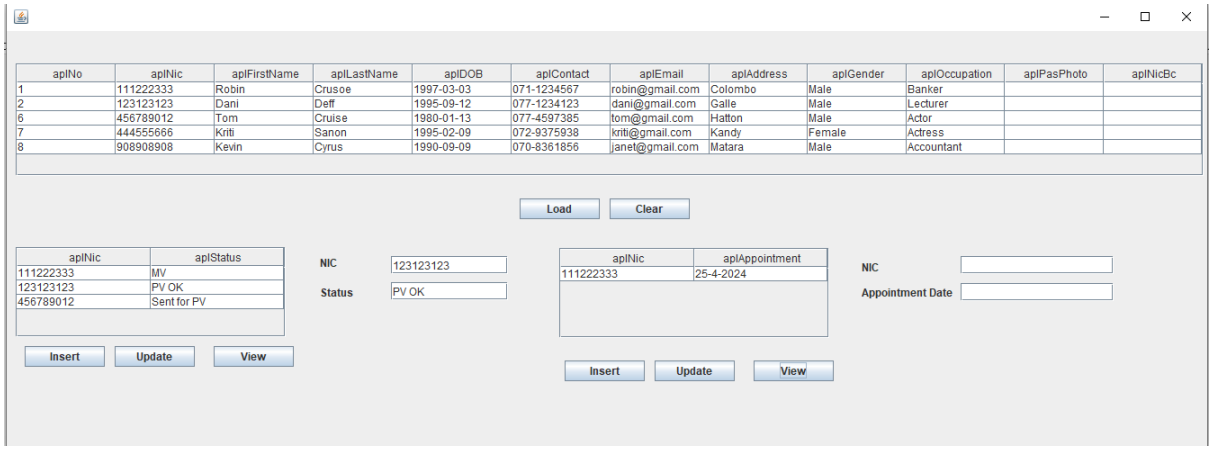
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Add status of an applicant	10	Applicant NIC, Applicant status	Show data updated successfully pop up and updated table.	Success pop up and updated table	Pass





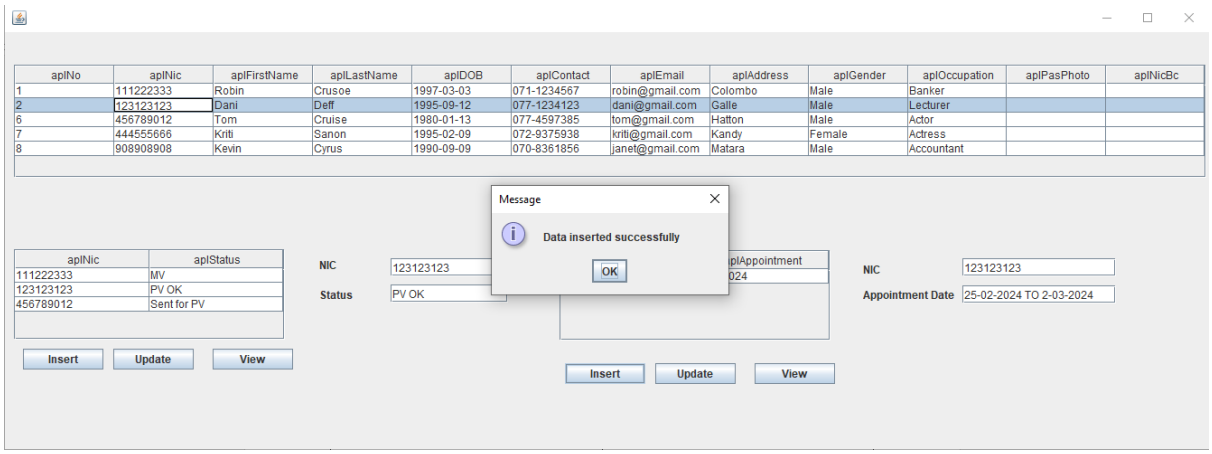
Test Case	
Test unit: View appointment data	Tester: Shanaya
Test case ID: 11	Test Type: Black Box
Test Description: Viewing appointment data	Test Execution Date: 2024. 02. 25
Title: Appointment details	Test Execution Time: 9:50 PM

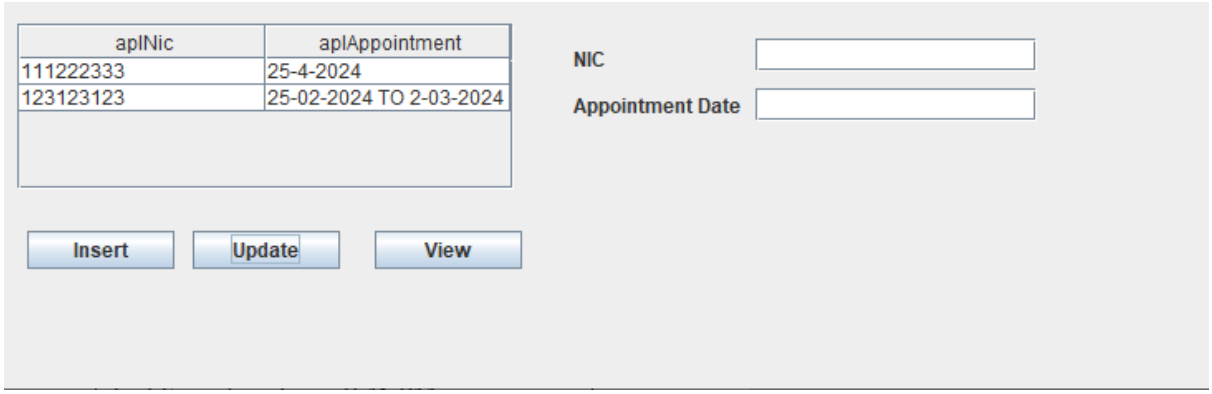
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Click on the “View” button	11	Click “View” button	Loading data to table.	Loaded table.	Pass



Test Case	
Test unit: Insert appointment data	Tester: Shanaya
Test case ID: 12	Test Type: Black Box
Test Description: Inserting date range from where applicant can select one	Test Execution Date: 2024. 02. 25
Title: Appointment details	Test Execution Time: 9:52 PM

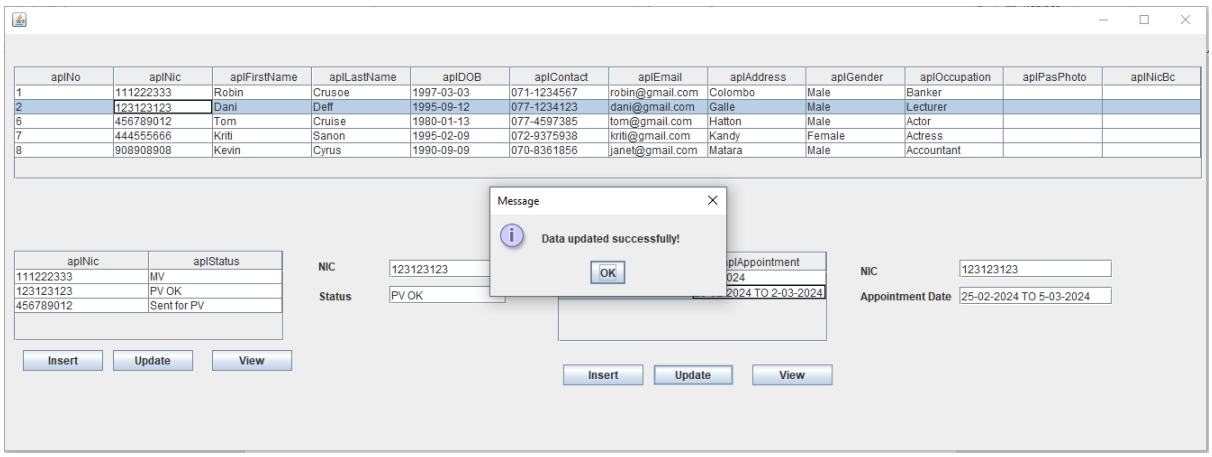
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Add appointment date for an applicant	12	Applicant NIC, Date range	Show data inserted successfully pop up and filled table.	Success pop up and filled table	Pass

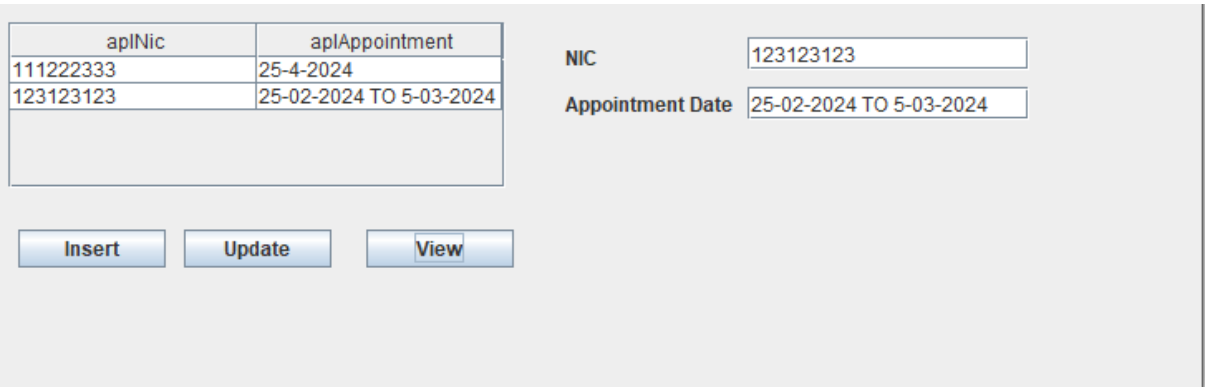




Test Case	
Test unit: Update appointment data	Tester: Shanaya
Test case ID: 13	Test Type: Black Box
Test Description: Updating appointment data	Test Execution Date: 2024. 02. 25
Title: Appointment details	Test Execution Time: 9:55 PM

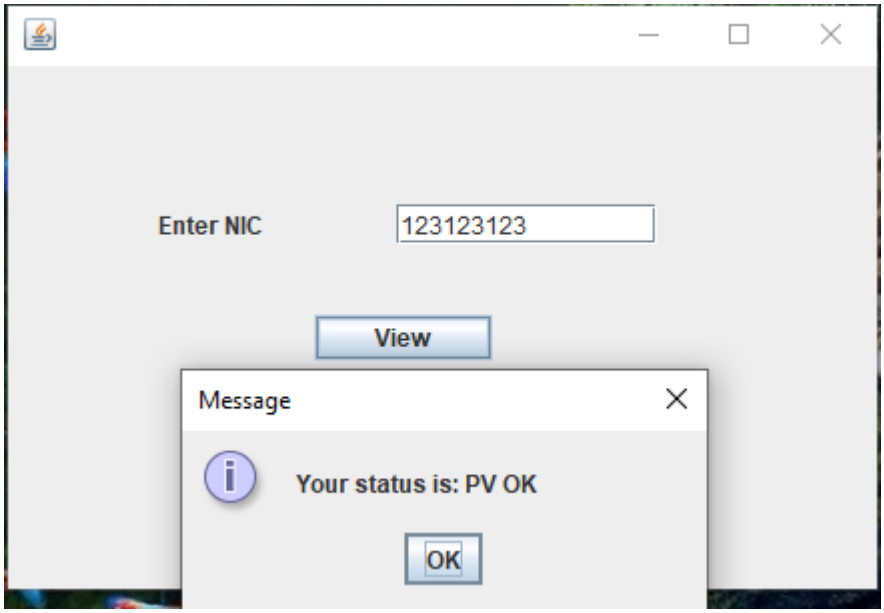
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Add date range of an applicant	13	Applicant NIC, Date range	Show data updated successfully pop up and updated table.	Success pop up and updated table	Pass





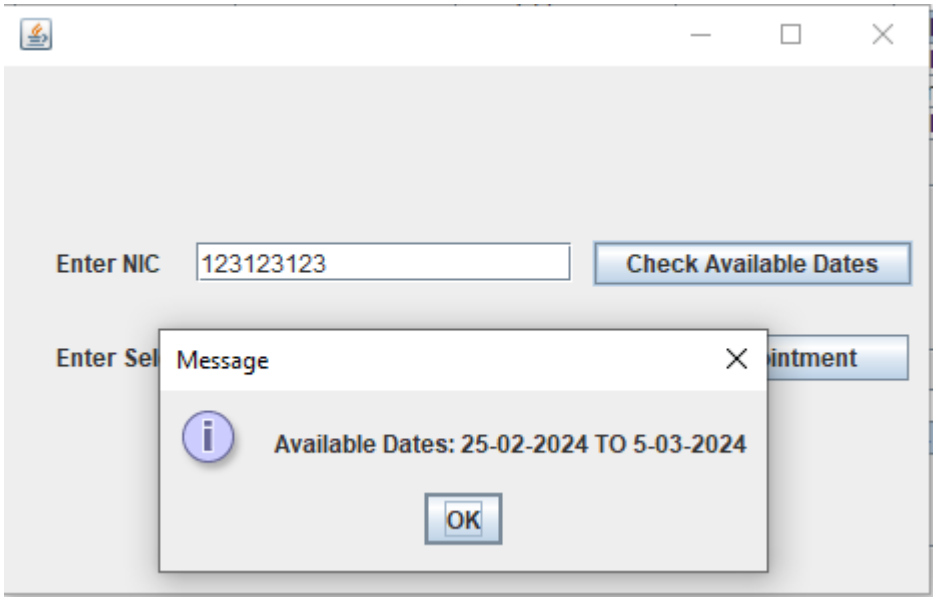
Test Case	
Test unit: View status	Tester: Shanaya
Test case ID: 14	Test Type: Black Box
Test Description: Viewing passport status	Test Execution Date: 2024. 02. 25
Title: Passport processing status	Test Execution Time: 9:59 PM

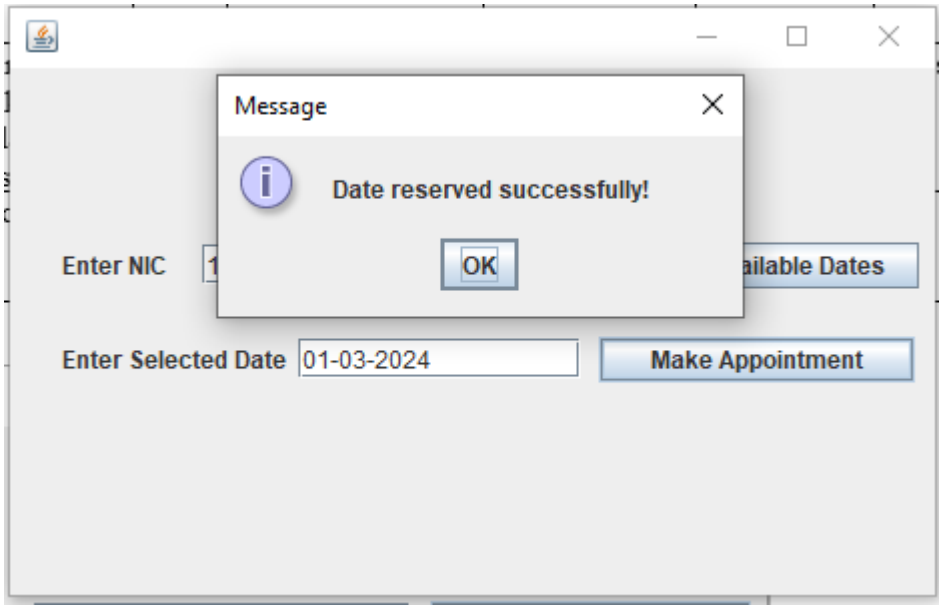
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Click on the "View" button	14	"View" button click	Show status in a pop up.	Status pop-up.	Pass



Test Case	
Test unit: Make appointment	Tester: Shanaya
Test case ID: 15	Test Type: Black Box
Test Description: Selecting a date from the given dates	Test Execution Date: 2024. 02. 25
Title: Appointment scheduling	Test Execution Time: 10:04 PM

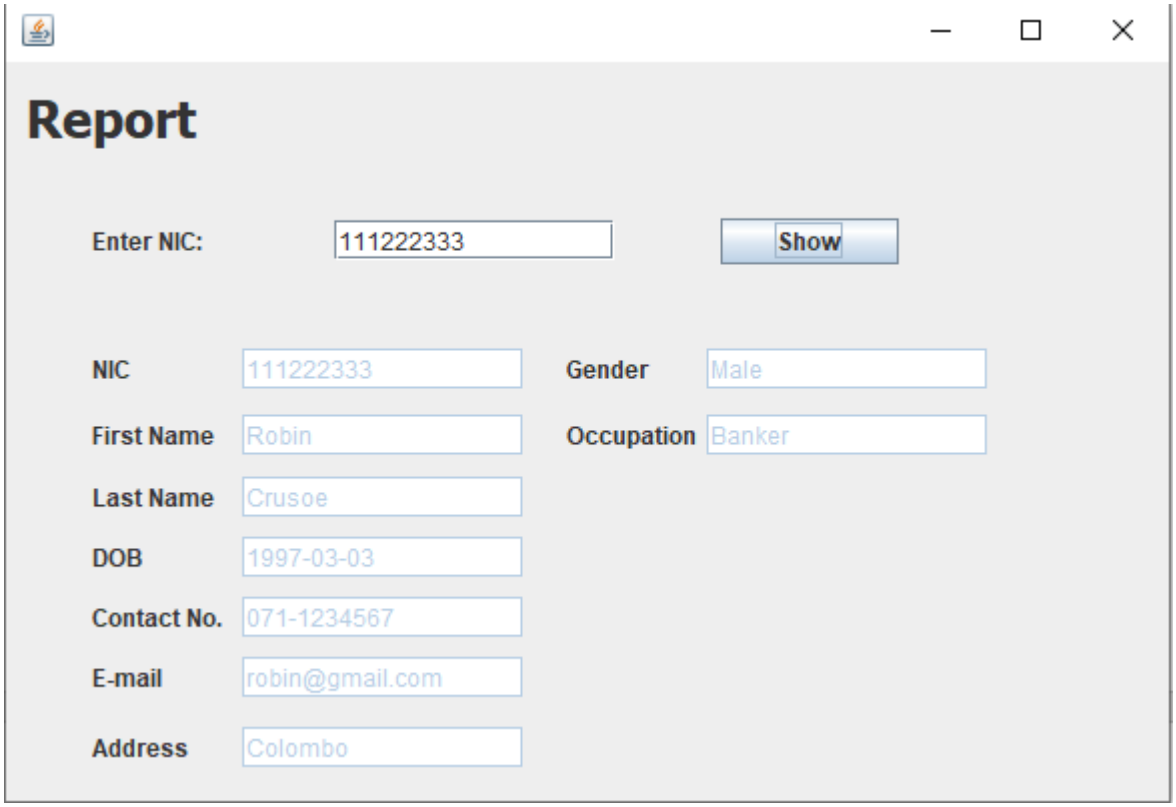
Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Click on the "Make appointment" button	15	"Make appointment" button click	Redirects to "Appointment Scheduling" form	Redirected page.	Pass
02	Enter NIC to check available dates	15	Applicant NIC	Show available dates in a pop up	Available dates showing pop up	Pass
03	Select date	15	One particular date of given range	Show dates reserved successfully pop up	Success pop up	Pass





Test Case	
Test unit: Generate report	Tester: Shanaya
Test case ID: 16	Test Type: Black Box
Test Description: Viewing applicant report	Test Execution Date: 2024. 02. 25
Title: Report details	Test Execution Time: 10:10 PM

Step No.	Test Step	Test Case ID	Test Input	Expected Result	Actual Result	Test Result (Pass/Fail)
01	Click on the “Show” button	16	NIC	Report with applicant details	Generated report	Pass


Report
Enter NIC:
111222333
Show
NIC
111222333
Gender
Male
First Name
Robin
Occupation
Banker
Last Name
Crusoe
DOB
1997-03-03
Contact No.
071-1234567
E-mail
robin@gmail.com
Address
Colombo

6. References

1. https://m.youtube.com/watch?v=frafcK6fhdQ&pp=ygUnSG93IHRvIGdldCBkYXRhIGZyb20gZGF0YWJhc2UgdG8ganRhYmxl#bottom-sheet
2. https://www.tutorialspoint.com/how-to-insert-an-image-in-to-mysql-database-using-java-program
3. https://www.geeksforgeeks.org
4. https://chat.openai.com

7. Annexure

1. Database connection class

```
public class DatabaseConnection {

	Connection conn = null;
	Connection conn2 = null;

	public Connection createConnection() {
		try {
			conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/PAS","root","Gnei
$810#");
			System.out.println("Connection established");
		}
		catch(SQLException e) {
			System.out.println("Connection Failed");
		}
		return conn;
	}

	public Connection createConnection2() {
		try {
			conn2 =
DriverManager.getConnection("jdbc:mysql://localhost:3306/Citizen","root","
Gnei$810#");
			System.out.println("Connection established");
		}
		catch(SQLException e) {
			System.out.println("Connection Failed");
		}
		return conn2;
	}
}
```

2. Applicant class

```
public class Applicant {
	private Connection conn;
	private String aplNic;
	private String aplPassword;
	private String aplFirstName;
	private String aplLastName;
	private String aplEmail;
	private String aplContact;
	private String aplGender;
	private String aplDOB;
	private String aplAddress;
	private String aplOccupation;
	private String aplPasPhoto;
	private Blob aplNicBc;
	private ByteArrayOutputStream baos;;

	public Applicant() {}

	public Applicant(String aplNic, String aplFirstName, String aplLastName,
String aplDOB, String aplContact, String aplEmail, String aplAddress, String
aplGender,String aplOccupation) {
		this.aplNic = aplNic;
		this.aplFirstName = aplFirstName;
		this.aplLastName = aplLastName;
		this.aplDOB = aplDOB;
		this.aplContact = aplContact;
		this.aplEmail = aplEmail;
		this.aplAddress = aplAddress;
		this.aplGender = aplGender;
		this.aplOccupation = aplOccupation;
	}


	public Applicant(String aplNic,String aplPassword) {
		this.aplNic = aplNic;
		this.aplPassword = aplPassword;
	}

	public void signUpApplicant(String aplNic, String aplPassword) {

		DatabaseConnection dbCon = new DatabaseConnection();
		conn = dbCon.createConnection();

		try {
			String sql = "INSERT INTO LoginAndSignup (aplNic,
aplPassword) VALUES (?,?)";
			PreparedStatement pstatement = conn.prepareStatement(sql);

			pstatement.setString(1, aplNic);
			pstatement.setString(2, aplPassword);

			int rowsInserted = pstatement.executeUpdate();

			if (rowsInserted > 0) {
```

```
                        JOptionPane.showMessageDialog(null, "Data inserted
successfully");
                    } else {
                        JOptionPane.showMessageDialog(null, "Data insertion
failed");
                    }
                }
            catch (SQLException e) {
                    System.out.println("Error: " + e.getMessage());
                }
        }


      public void submitApplication() {

            DatabaseConnection dbCon = new DatabaseConnection();
            conn = dbCon.createConnection();

            try {
            String sql = "INSERT INTO Applicant (aplNic, aplFirstName,
aplLastName, aplDOB, aplContact, aplEmail, aplAddress, aplGender, aplOccupation)
VALUES (?,?,?,?,?,?,?,?,?)";
             PreparedStatement pstmt = conn.prepareStatement(sql);

             pstmt.setString(1, aplNic);
             pstmt.setString(2, aplFirstName);
             pstmt.setString(3, aplLastName);
             pstmt.setString(4, aplDOB);
             pstmt.setString(5, aplContact);
             pstmt.setString(6, aplEmail);
             pstmt.setString(7, aplAddress);
             pstmt.setString(8, aplGender);
             pstmt.setString(9, aplOccupation);

             int rowsInserted = pstmt.executeUpdate();

             if (rowsInserted > 0) {
                 JOptionPane.showMessageDialog(null, "Data inserted
successfully");
             } else {
                 JOptionPane.showMessageDialog(null, "Data insertion failed");
             }
          }
          catch(SQLException e) {
            System.out.println("Error: " + e.getMessage());
          }
      }

      public void uploadDocs(byte[] documentBytes) {
          DatabaseConnection dbCon = new DatabaseConnection();
          conn = dbCon.createConnection();

          try {
             String sql = "INSERT INTO Applicant (aplNicBc) VALUES (?)";
             PreparedStatement pstmt = conn.prepareStatement(sql);

             // Set the byte array directly to the Blob column
             pstmt.setBytes(1, documentBytes);
```

```
                int rowsInserted = pstmt.executeUpdate();

                if (rowsInserted > 0) {
                    JOptionPane.showMessageDialog(null, "Uploaded successfully");
                } else {
                    JOptionPane.showMessageDialog(null, "Uploading failed");
                }
            } catch (SQLException e) {
                System.out.println("Error: " + e.getMessage());
            }
        }

        private void insertImage(byte[] imageData) {
          try {
              String sql = "INSERT INTO Images (image_data) VALUES (?)";
              PreparedStatement pstmt = conn.prepareStatement(sql);
              pstmt.setBytes(1, imageData);

              int rowsInserted = pstmt.executeUpdate();

              if (rowsInserted > 0) {
                  JOptionPane.showMessageDialog(null, "Image uploaded
successfully.");
              } else {
                  JOptionPane.showMessageDialog(null, "Failed to upload image.",
"Error", JOptionPane.ERROR_MESSAGE);
              }
          } catch (SQLException e) {
              JOptionPane.showMessageDialog(null, "Database error: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
          }
      }

        public void viewStatus(String aplNic){
              DatabaseConnection dbcon = new DatabaseConnection();
            Connection conn = dbcon.createConnection();

            try {
              String sql = "SELECT aplStatus FROM applicantStatus WHERE aplNic
= ?";
                PreparedStatement pstatement = conn.prepareStatement(sql);
                pstatement.setString(1, aplNic);
                ResultSet resultSet = pstatement.executeQuery();
                while (resultSet.next()) {
                      String status = resultSet.getString("aplStatus");

                      JOptionPane.showMessageDialog(null, "Your status is: "+
status);
                     }
            }
            catch(SQLException e) {
              JOptionPane.showMessageDialog(null, e.getMessage());
            }
        }

        public void checkAvailableDates(String aplNic){
              DatabaseConnection dbcon = new DatabaseConnection();
            Connection conn = dbcon.createConnection();
```

```
            try {
               String sql = "SELECT aplAppointment FROM AppointmentDetails WHERE
aplNic = ?";
                PreparedStatement pstatement = conn.prepareStatement(sql);
                pstatement.setString(1, aplNic);
                ResultSet resultSet = pstatement.executeQuery();
                while (resultSet.next()) {
                      String appointment = resultSet.getString("aplAppointment");

                      JOptionPane.showMessageDialog(null, "Available Dates: "+
appointment);
                     }
            }
            catch(SQLException e) {
               JOptionPane.showMessageDialog(null, e.getMessage());
            }
        }

        public void updateAppointment(String aplNic, String aplAppointment) {

               DatabaseConnection dbcon = new DatabaseConnection();
            Connection conn = dbcon.createConnection();

            try {
               String sql = "UPDATE AppointmentDetails SET aplAppointment = ? WHERE
aplNic = ?";
               PreparedStatement pstatement = conn.prepareStatement(sql);

               pstatement.setString(1, aplAppointment);
               pstatement.setString(2, aplNic);

               int rowsUpdated = pstatement.executeUpdate();
               if (rowsUpdated > 0) {
                      JOptionPane.showMessageDialog(null, "Date reserved
successfully!");
               }
               else {
                      JOptionPane.showMessageDialog(null, "Date reservation
Failed!");
               }
            }
            catch(SQLException e) {
               JOptionPane.showMessageDialog(null, e.getMessage());
            }
        }

}
```

3. Admin class

```
public class Admin {
      Connection conn;
      private String aplNic;
      private String aplStatus;

      public Admin() {}

      public void insertStatus(String aplNic,String aplStatus) {

            DatabaseConnection dbcon = new DatabaseConnection();
          Connection conn = dbcon.createConnection();

          try {
              String sql = "INSERT INTO ApplicantStatus (aplNic,
aplStatus) VALUES (?,?)";
              PreparedStatement pstatement = conn.prepareStatement(sql);

              pstatement.setString(1, aplNic);
              pstatement.setString(2, aplStatus);

              int rowsInserted = pstatement.executeUpdate();

              if (rowsInserted > 0) {
                  JOptionPane.showMessageDialog(null, "Data inserted
successfully");
              } else {
                  JOptionPane.showMessageDialog(null, "Data insertion
failed");
              }
          }
      catch (SQLException e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
          }
      }

      public void updateStatus(String aplNic,String aplStatus) {

            DatabaseConnection dbcon = new DatabaseConnection();
          Connection conn = dbcon.createConnection();

          try {
            String sql = "UPDATE ApplicantStatus SET aplStatus = ? WHERE
aplNic = ?";
            PreparedStatement pstatement = conn.prepareStatement(sql);

            pstatement.setString(1, aplStatus);
            pstatement.setString(2, aplNic);

            int rowsUpdated = pstatement.executeUpdate();
            if (rowsUpdated > 0) {
```

```
                    JOptionPane.showMessageDialog(null, "Data updated
successfully!");
            }
            else {
                    JOptionPane.showMessageDialog(null, "Data update
Failed!");
            }
          }
          catch(SQLException e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
          }
      }

    public void scheduleAppointment(String aplNic, String aplAppointment)
{
      DatabaseConnection dbcon = new DatabaseConnection();
          Connection conn = dbcon.createConnection();

          try {
              String sql = "INSERT INTO AppointmentDetails (aplNic,
aplAppointment) VALUES (?,?)";
              PreparedStatement pstatement = conn.prepareStatement(sql);

              pstatement.setString(1, aplNic);
              pstatement.setString(2, aplAppointment);

              int rowsInserted = pstatement.executeUpdate();

              if (rowsInserted > 0) {
                  JOptionPane.showMessageDialog(null, "Data inserted
successfully");
              } else {
                  JOptionPane.showMessageDialog(null, "Data insertion
failed");
              }
          }
      catch (SQLException e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
          }
    }

    public void updateAppointment(String aplNic, String aplAppointment) {

            DatabaseConnection dbcon = new DatabaseConnection();
          Connection conn = dbcon.createConnection();

          try {
            String sql = "UPDATE AppointmentDetails SET aplAppointment = ?
WHERE aplNic = ?";
            PreparedStatement pstatement = conn.prepareStatement(sql);

            pstatement.setString(1, aplAppointment);
            pstatement.setString(2, aplNic);
```

```
            int rowsUpdated = pstatement.executeUpdate();
            if (rowsUpdated > 0) {
                  JOptionPane.showMessageDialog(null, "Data updated
successfully!");
            }
            else {
                  JOptionPane.showMessageDialog(null, "Data update
Failed!");
            }
          }
          catch(SQLException e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
          }
      }
}
```

4. Verification class

```
public class Verification {

      Connection conn = null;
      Connection conn2 = null;
      String username = null;
      String password = null;


      public boolean isDuplicateNIC(String username, String password) {

            DatabaseConnection dbCon = new DatabaseConnection();
            conn = dbCon.createConnection();

            try {
                  String sql = "SELECT aplNic FROM loginandsignup WHERE
aplNic = ?";
                  PreparedStatement pstmt = conn.prepareStatement(sql);
                  pstmt.setString(1, username);
                  ResultSet result = pstmt.executeQuery();

            if(result.next()) {
                        JOptionPane.showMessageDialog(null, "You're
already registered!");
                        pstmt.close();
                        conn.close();
                  }else {
                        isValidCitizen(username,password);
                  }
                  return true;
            }
        catch (SQLException e){
            JOptionPane.showMessageDialog(null, "Error: " +
e.getMessage());
         }
            return false;
    }


      public boolean isValidCitizen(String username, String password) {
            DatabaseConnection dbCon = new DatabaseConnection();
            Connection conn2 = dbCon.createConnection2();

            Applicant ap = new Applicant(username, password);

            try {
                  String sql = "SELECT nic FROM citizendetails WHERE nic
= ?";
              PreparedStatement stmt = conn2.prepareStatement(sql);
              stmt.setString(1, username);
              ResultSet result = stmt.executeQuery();
```

```
                if (result.next()) {
                    result.getString("nic");

                    JOptionPane.showMessageDialog(null, "Verification
successful");
                    ap.signUpApplicant(username, password);

                return true;

                } else {
                    // Verification failed, user is not a citizen of Sri
Lanka
                    JOptionPane.showMessageDialog(null, "Not a citizen of
Sri Lanka");
                }
            }
        catch (SQLException e){
            JOptionPane.showMessageDialog(null, "Error: " +
e.getMessage());
         }
            return false;
      }


      public boolean validateLogin(String username, String password) {
            DatabaseConnection dbcon = new DatabaseConnection();
            conn = dbcon.createConnection();

            try {
                  String sql = "SELECT * FROM loginandsignup WHERE aplNic
= ? AND aplPassword = ?";

                PreparedStatement pstatement = conn.prepareStatement(sql);
                pstatement.setString(1, username);
                pstatement.setString(2, password);

                ResultSet resultSet = pstatement.executeQuery();

                if (resultSet.next()) {
                    String storedPassword =
resultSet.getString("aplPassword");

                    if (storedPassword.equals(password)) {
                        JOptionPane.showMessageDialog(null, "Login
Successful");
                    }
                    return true;
                }
                else {
                    JOptionPane.showMessageDialog(null, "Incorrect Username
or Password");
                }
            }
```

```
            catch (SQLException e) {
                  JOptionPane.showMessageDialog(null, "Error: " +
e.getMessage());
            }
            return false;
    }

      public boolean validateAdminLogin(String username, String password)
{
            DatabaseConnection dbcon = new DatabaseConnection();
            conn = dbcon.createConnection();

            try {
                  String sql = "SELECT * FROM AdminLogin WHERE
adminUsername = ? AND adminPassword = ?";

              PreparedStatement pstatement = conn.prepareStatement(sql);
              pstatement.setString(1, username);
              pstatement.setString(2, password);

              ResultSet resultSet = pstatement.executeQuery();

              if (resultSet.next()) {
                  String storedPassword =
resultSet.getString("adminPassword");

                  if (storedPassword.equals(password)) {
                      JOptionPane.showMessageDialog(null, "Login
Successful");
                  }
                  return true;
              }
              else {
                  JOptionPane.showMessageDialog(null, "Incorrect Username
or Password");
              }
            }
            catch (SQLException e) {
                  JOptionPane.showMessageDialog(null, "Error: " +
e.getMessage());
            }
            return false;
    }
}
```

5. Admin login frame

- Login button

```
btnlogin.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {

                  String username =
txtusername.getText();
                  String password =
txtpassword.getText();

                  Verification verify = new
Verification();
                  boolean isValidLogin =
verify.validateAdminLogin(username, password);

                  if (isValidLogin) {
                    dispose();
                    AdminPanel adpanel = new
AdminPanel();
                    adpanel.setVisible(true);
                }
              }
          });
```

- Back Button

```
btnback.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {

                  HomePage HomePageFrame = new
HomePage();
                  HomePageFrame.setVisible(true);
                  dispose();
              }
          });
```

6. Applicant login frame

- Login button

```
btnlogin.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {

                        String username = txtusername.getText();
                        String password = txtpassword.getText();

                        Verification verify = new Verification();
                        boolean isValidLogin =
verify.validateLogin(username, password);

                        if (isValidLogin) {
                      dispose();
                      ApplicantDashboard appDashboardFrame = new
ApplicantDashboard();
                          appDashboardFrame.setVisible(true);
                  }
                  }
            });
```

- Back button

```
btnback.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {

                        HomePage HomePageFrame = new HomePage();
                        HomePageFrame.setVisible(true);
                        dispose();
                  }
            });
```

- Register button

```
btnNewButton.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {

                        Register register = new Register();
                        register.setVisible(true);
                        dispose();
                  }
            });
```

7. Applicant sign up

- Signup button

```
btnsignup.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        String nic = txtnic.getText();
                        String password = txtpassword.getText();
                        String confirmPassword =
txtconfirmpassword.getText();

                         if (!password.equals(confirmPassword)) {
                                JOptionPane.showMessageDialog(null,
"Password and confirm password do not match!", "Password Mismatch",
JOptionPane.ERROR_MESSAGE);
                                return; // Exit the method if
passwords do not match
                            }

                        Verification verify = new Verification();
                        verify.isDuplicateNIC(nic, password);

                        boolean isValidCitizen = false;

                        if(isValidCitizen == true) {
                              ApplicantDashboard appDashboardFrame =
new ApplicantDashboard();
                              appDashboardFrame.setVisible(true);
                              dispose();
                        }
                }
            });
```

- Back button

```
btnback.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        ApplicantLogin applicantLoginFrame = new
ApplicantLogin();
                        applicantLoginFrame.setVisible(true);
                        dispose();
                }
            });
```

8. Admin panel

- Load button

```
btnload.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                     DatabaseConnection dbCon = new
DatabaseConnection();
                     Connection conn = dbCon.createConnection();
                     try {
                          String sql = "SELECT * FROM
Applicant";
                          Statement stmt =
conn.createStatement();

                          ResultSet result =
stmt.executeQuery(sql);
                          ResultSetMetaData rsmd =
result.getMetaData();
                          DefaultTableModel model =
(DefaultTableModel) table.getModel();

                          int cols = rsmd.getColumnCount();
                          String[] colName = new String [cols];


                          for(int i=0; i<cols;i++)
                               colName[i] =
rsmd.getColumnName(i+1);

                          model.setColumnIdentifiers(colName);

                          String NIC, FirstName, LastName,
Email, Contact, Gender, aplDOB, Address, Docs, Status;

                          while (result.next()) {
                               NIC = result.getString(1);
                               FirstName = result.getString(2);
                               LastName = result.getString(3);
                               Email  = result.getString(4);
                               Contact = result.getString(5);
                               Gender = result.getString(6);
                               aplDOB = result.getString(7);
                               Address = result.getString(8);
                               Docs = result.getString(9);
```

```
                                Status = result.getString(10);

                                String row[] = {NIC, FirstName,
LastName, Email, Contact, Gender, aplDOB, Address, Docs,
Status};
                                model.addRow(row);
                            }
                            stmt.close();
                            conn.close();
                        }
                        catch(SQLException ex) {
                            System.out.println(ex.getMessage());
                        }
                    }
                });
```

- Clear button

```
btnclear.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {

                        table.setModel(new DefaultTableModel());
                    }
                });
```

- Update button (of status)

```
btnupdate.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {

                        String nic = txtnic.getText();
                        String status = txtstatus.getText();

                        Admin ad = new Admin();
                        ad.updateStatus(nic, status);

                        int i = table_1.getSelectedRow();
                        if (i>=0) {
                            model.setValueAt(txtnic.getText(), i,
0);
                            model.setValueAt(txtstatus.getText(),
i, 1);
```

```
                    }
                }
            });
```

- View button (of status)

```
btnsave.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                    DatabaseConnection dbCon = new
DatabaseConnection();
                    Connection conn = dbCon.createConnection();

                    DefaultTableModel model = new
DefaultTableModel();
                   try {
                       String sql = "SELECT * FROM
ApplicantStatus";
                       PreparedStatement pstmt =
conn.prepareStatement(sql);
                       ResultSet rs = pstmt.executeQuery();

                       // Get metadata about the ResultSet
(columns)
                       ResultSetMetaData metaData =
(ResultSetMetaData) rs.getMetaData();
                       int columnCount =
metaData.getColumnCount();

                       // Add columns to the table model
                       for (int column = 1; column <=
columnCount; column++) {

model.addColumn(metaData.getColumnLabel(column));
                       }

                       // Add rows to the table model
                       while (rs.next()) {
                           Object[] row = new
Object[columnCount];
                           for (int i = 0; i < columnCount;
i++) {
                               row[i] = rs.getObject(i + 1);
                           }
```

```
                    model.addRow(row);
                }

                rs.close();
                pstmt.close();
                conn.close();

                // Set the model to the table
                table_1.setModel(model);

            }
            catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
    });
```

- Insert button (of status)

```
btnAdd.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            String nic = txtnic.getText();
            String status = txtstatus.getText();

            if(txtnic.getText().equals("") ||
txtstatus.getText().equals("")) {
                JOptionPane.showMessageDialog(null,
"Please fill all fields");
            }
            else {
                row[0] = txtnic.getText();
                row[1] = txtstatus.getText();

                model.addRow(row);

                Admin ad = new Admin();
                ad.insertStatus(nic, status);

                txtnic.setText("");
                txtstatus.setText("");
            }
        }
    });
```

- Insert button (of appointment)

```
btnAdd_1.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        String nic = txtnic_1.getText();
                        String app = txtappdate.getText();

                        if(txtnic_1.getText().equals("") ||
txtappdate.getText().equals("")) {
                                JOptionPane.showMessageDialog(null,
"Please fill all fields");
                        }
                        else {
                                row1[0] = txtnic_1.getText();
                                row1[1] = txtappdate.getText();

                                model.addRow(row1);

                                Admin ad = new Admin();
                                ad.scheduleAppointment(nic, app);

                                txtnic_1.setText("");
                                txtappdate.setText("");
                        }
                }
            });
```

- Update button (of appointment)

```
btnupdate_1.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        String nic = txtnic_1.getText();
                        String app = txtappdate.getText();

                        Admin ad = new Admin();
                        ad.updateAppointment(nic, app);

                        int i = table_2.getSelectedRow();
                        if (i>=0) {
                                model.setValueAt(txtnic_1.getText(),
i, 0);
```

```
    model.setValueAt(txtappdate.getText(), i, 1);
                    }
                }
            });
```

- View button (of appointment)

```
btnsave_1.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                    DatabaseConnection dbCon = new
DatabaseConnection();
                    Connection conn = dbCon.createConnection();

                    DefaultTableModel model = new
DefaultTableModel();
                   try {
                        String sql = "SELECT * FROM
AppointmentDetails";
                        PreparedStatement pstmt =
conn.prepareStatement(sql);
                        ResultSet rs = pstmt.executeQuery();

                        // Get metadata about the ResultSet
(columns)
                        ResultSetMetaData metaData =
(ResultSetMetaData) rs.getMetaData();
                        int columnCount =
metaData.getColumnCount();

                        // Add columns to the table model
                        for (int column = 1; column <=
columnCount; column++) {

model.addColumn(metaData.getColumnLabel(column));
                        }

                        // Add rows to the table model
                        while (rs.next()) {
                            Object[] row = new
Object[columnCount];
                            for (int i = 0; i < columnCount;
i++) {
```

```
                    row[i] = rs.getObject(i + 1);
                }
                model.addRow(row);
            }

            rs.close();
            pstmt.close();
            conn.close();

            // Set the model to the table
            table_2.setModel(model);

        }
        catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
});
```

9. Applicant dashboard

- Fill form button

```
btnFillform.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        PassportApplication passportApplFrame = new
PassportApplication();
                        passportApplFrame.setVisible(true);
                        dispose();
                }
            });
```

- View status button

```
btnViewstatus.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        ViewStatus vs = new ViewStatus();
                        vs.setVisible(true);
                        dispose();
                }
            });
```

- Make appointment button

```
btnMakeAppointment.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        MakeAppointment ma = new MakeAppointment();
                        ma.setVisible(true);
                        dispose();
                }
            });
```

- Generate report button

```
btnGenerateReport.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        GenerateReport genrep = new GenerateReport();
                        genrep.setVisible(true);
```

```
                dispose();
            }
        });
```

10. View status frame

- View button

```
btnview.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                String enterednic =
txtenternic.getText();

                Applicant ap = new Applicant();
                ap.viewStatus(enterednic);
            }
        });
```

11. Make appointment frame

- Check available dates button

```
btndates.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        String nic = txtenternic.getText();

                        Applicant ap = new Applicant();
                        ap.checkAvailableDates(nic);
                }
            });
```

- Make appointment button

```
btnmakeapp.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        String nic = txtenternic.getText();
                        String date = txtselecteddate.getText();

                        Applicant ap = new Applicant();
                        ap.updateAppointment(nic,date);
                }
            });
```

12. Passport application frame

- Submit button

```
btnSubmit.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                    if(txtNIC.getText().equals("") ||
txtfirstname.getText().equals("") ||
txtlastname.getText().equals("") || txtdob.getText().equals("") ||
txtphone.getText().equals("") || txtemail.getText().equals("") ||
txtoccupation.getText().equals("") ) {
                        JOptionPane.showMessageDialog(null,
"Please fill all fields");
                    }
                    if(!rdbtnMale.isSelected()
&& !rdbtnFemale.isSelected()) {
                        JOptionPane.showMessageDialog(null,
"Please select gender");
                    }
                    else {
                        String nic = txtNIC.getText();
                        String fname = txtfirstname.getText();
                        String lname = txtlastname.getText();
                        String dob = txtdob.getText();
                        String phone = txtphone.getText();
                        String email = txtemail.getText();
                        String address = txtaddress.getText();

                        String gender = "";
                        if(rdbtnMale.isSelected()) {
                            gender = "Male";
                        }
                        if(rdbtnFemale.isSelected()) {
                            gender = "Female";
                        }
                        String occupation  =
txtoccupation.getText();


                        Applicant ap = new Applicant(nic, fname,
lname, dob, phone, email, address, gender,occupation);
                        ap.submitApplication();
                    }

                }
            });
```

- Clear button

```
btnClear.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                      int confirm =
JOptionPane.showConfirmDialog(null, "Are you sure you want to clear
the fields?", "Confirmation", JOptionPane.YES_NO_OPTION);
                if (confirm == JOptionPane.YES_OPTION) {
                txtNIC.setText("");
                      txtfirstname.setText("");
                      txtlastname.setText("");
                      txtdob.setText("");
                      txtphone.setText("");
                      txtemail.setText("");
                      txtaddress.setText("");
                      txtoccupation.setText("");
                      group.clearSelection();
                }
                }
           });
```

- Back button

```
btnBack.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                      ApplicantDashboard applDashboardFrame = new
ApplicantDashboard();
                      applDashboardFrame.setVisible(true);
                      dispose();
                }
           });
```

- Upload button (of photo)

```
btnUploadPP.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                      JFileChooser chooser = new JFileChooser();
                      chooser.showOpenDialog(null);
                      File file = chooser.getSelectedFile();
                      String path = file.getAbsolutePath();
                              try {
                                  BufferedImage bi =
ImageIO.read(new File(path));
```

```
                                    Image img =
bi.getScaledInstance(107, 130, Image.SCALE_SMOOTH);
                                    ImageIcon icon = new
ImageIcon(img);
                                    lblPhoto_2.setIcon(icon);
                                }
                                catch (IOException e1) {
                                    e1.printStackTrace();
                                }
                            }
                        });
```

- Upload button (of docs)

```
btnUploadNICBC.addActionListener(new ActionListener() {

                public void actionPerformed(ActionEvent e) {

                     JFileChooser chooser = new JFileChooser();
                   chooser.showOpenDialog(null);
                   File file = chooser.getSelectedFile();
                     if (file != null) { // Ensure a file is
selected
                      try {
                          FileInputStream fis = new
FileInputStream(file);
                          ByteArrayOutputStream baos = new
ByteArrayOutputStream();
                          byte[] buffer = new byte[1024];
                          int bytesRead;
                          while ((bytesRead = fis.read(buffer)) !=
-1) {
                              baos.write(buffer, 0, bytesRead);
                          }
                          String pdfContent = baos.toString("UTF-
8"); // Convert byte array to string

                          // Truncate the content to fit within the
JTextField
                          int maxLength = 1000; // Maximum length
of content to display
                          if (pdfContent.length() > maxLength) {
                              pdfContent = pdfContent.substring(0,
maxLength);
                          }

                          // Set the PDF content to the JTextField
```

```
                    lblPhoto_1.setText(pdfContent);

                    // Close the input stream
                    fis.close();
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        }
});
```

13. Generate report panel

- Show button

```
btnShow.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {

                        DatabaseConnection dbcon = new
DatabaseConnection();
                      Connection conn = dbcon.createConnection();

                      try {
                          String sql = "SELECT aplNic, aplFirstName,
aplLastName, aplDOB, aplContact, aplEmail, aplAddress, aplGender,
aplOccupation FROM Applicant WHERE aplNic = ?";

                          String nic = txtenterednic.getText();

                          PreparedStatement pstmt =
conn.prepareStatement(sql);
                          pstmt.setString(1, nic);
                          ResultSet resultSet = pstmt.executeQuery();

                          if (!resultSet.next()) {
                              JOptionPane.showMessageDialog(null, "NIC not
found");
                          }
                          else {
                              String NIC = resultSet.getString(1);
                              String firstName = resultSet.getString(2);
                              String lastName = resultSet.getString(3);
                              String dob = resultSet.getString(4);
                              String contact = resultSet.getString(5);
                              String email = resultSet.getString(6);
                              String address = resultSet.getString(7);
                              String gender = resultSet.getString(8);
                              String occupation = resultSet.getString(9);

                              txtnic.setText(NIC);
                              txtfirstname.setText(firstName);
                              txtlastname.setText(lastName);
                              txtdob.setText(dob);
                              txtcontact.setText(contact);
                              txtemail.setText(email);
                              txtaddress.setText(address);
                              txtgender.setText(gender);
                              txtoccupation.setText(occupation);

                              txtnic.setEnabled(false);
                              txtfirstname.setEnabled(false);
                              txtlastname.setEnabled(false);
                              txtdob.setEnabled(false);
                              txtcontact.setEnabled(false);
```

```
                                txtemail.setEnabled(false);
                                txtaddress.setEnabled(false);
                                txtgender.setEnabled(false);
                                txtoccupation.setEnabled(false);
                            }
                        }
                        catch(SQLException ex) {
                          JOptionPane.showMessageDialog(null,
ex.getMessage());
                        }
                    }
            });
```

Assumptions

1. The system has special permissions to access part of the citizen database for the applicant genuineness verification process.
2. The police verification is done externally to the system. The police report is emailed to a separate mail handling system.