
# Project Title:
# Alzheimer's disease patient detection management system

Course code: 2232

**Course Name:** Data Structure **Session**: W07

Student Name: Songlin Shang (Arthur) Student Number: 1308184

Student Name: Cong Ye (Ava) Student Number: 1306248

Student Name: Zhiqiu Liu (Ashley) Student Number: 1308104

**Professor**: Dr Ken Ehimwenma, Ph.D.

## Date

__2024/12/17__

**********

# Abstract

This paper introduces an innovative system, Alzheimer's Disease Patient Detection and Management System, designed to enhance the efficiency of patient information management and communication between doctors and families. The system provides distinct interaction interfaces for doctors and family members. Doctors can store and retrieve patient data, including ID, name, gender, age, medication type and schedule, and follow-up plans. Families, on the other hand, can access the latest treatment instructions and follow-up details through a user-friendly interface.

The development process involved the use of modern database management systems and front-end technologies to ensure secure, accurate, and seamless data exchange. Simulated patient data was utilized to validate the system's functionality, focusing on medication adherence and follow-up tracking. Results indicate that the system significantly streamlines the management of patient information, improving medication compliance and facilitating timely follow-ups.

This work highlights the potential of integrating technology into Alzheimer’s disease management to bridge gaps in care and enhance patient outcomes. Future enhancements may include incorporating machine learning algorithms for predictive analytics and further personalization of treatment plans.

# Table of content

# Introduction

Alzheimer's disease, a progressive neurodegenerative disorder, affects millions of individuals worldwide, with cases expected to triple by 2050 due to aging populations [1]. This condition imposes significant challenges on patients, families, and healthcare providers, as it often involves complex medication regimens, regular follow-ups, and constant monitoring of disease progression. Studies indicate that poor communication and fragmented data management contribute to medication non-adherence and delays in treatment adjustments, exacerbating patient outcomes. [2]

To address these challenges, this report presents the Alzheimer's Disease Patient Detection and Management System, a technology-driven solution designed to streamline communication and coordination between doctors and families. The system provides two distinct user interfaces: one for doctors to manage patient information securely, including ID, demographic details, medication schedules, and follow-up plans, and another for families to access up-to-date treatment instructions and follow-up plans. By improving data accessibility and enhancing doctor-patient-family communication, the system aims to ensure better adherence to treatment protocols and improve the overall care experience. Digital solutions in Alzheimer's management have shown promise in improving care. [2]

This report is structured as follows:

- **Related Works**: A review of existing literature and technologies relevant to Alzheimer’s disease management systems.
- **Research Methodology**: Details on the design, implementation, and tools used, including UML diagrams, algorithms, and system architecture.
- **Analysis and Report**: Presentation of findings using graphs, diagrams, and screenshots to evaluate the system's performance and usability.
- **Discussion**: A comprehensive discussion of the system’s purpose, development process, and a demonstration of its functionalities.
- **Conclusions**: A summary of the project's key outcomes, insights, and future development possibilities.
- **Appendices**: Supplementary materials, including group member contributions and additional documentation.

This report not only provides an overview of the system but also demonstrates its potential to transform Alzheimer’s disease management, making it a valuable resource for researchers, healthcare professionals, and families alike.

# Related Works (Review of Literature)

1. **Electronic Patient Record(EPR) System:**

System Implementation and User Experience Insights:

Implementation Challenges: One study highlights the challenges of implementing EPR systems, particularly user resistance and the need for adequate training. For a system that serves both doctors and families, ensuring user acceptance and providing proper training is crucial. Addressing these challenges effectively can enhance the system's long-term usability and success [6].

Workflow Optimization: The importance of seamless workflows in EPR systems is emphasized, as they improve overall efficiency. Similarly, your system must ensure that workflows between doctors (e.g., entering patient data) and families (e.g., viewing medical updates) are optimized to prevent disruptions or delays [7].

Data Integration: Integrating patient data across systems is a key factor in EPR implementations. Your project can benefit from adopting efficient data integration methods to synchronize patient records, medication plans, and appointment histories, enabling accurate tracking and management [8]

2. **HealthPlus** [3]

HealthPlus is a healthcare facility management system that offers several functionalities to streamline operations in medical institutions:

- → Patient registration
- → Appointment scheduling
- → Storage of patient records
- → Pharmacy billing
- → Pharmacy inventory management

This system enables healthcare providers to efficiently manage patient information, appointments, and pharmaceutical inventory, ensuring smooth administrative processes

3. **E-HealthCare-Management-System[4]**

The E-HealthCare-Management-System is a Java-based console application designed to streamline the management processes for patients, doctors, and administrators. Its main features include:

- → For patients: Registration, login, profile viewing, appointment scheduling with doctors, report viewing, doctor selection, appointment record management, and online payments.
- → For administrators: Adding and deleting doctors, viewing patient and doctor lists, managing patient feedback, and generating reports.
- → For doctors: Logging in, viewing profiles and appointments, and managing patient records.

This system provides a simple yet powerful solution for effective management across different user roles.

4. **Patient Management System [5]**

The Patient Management System is a Java-based application developed using the Swing GUI toolkit.

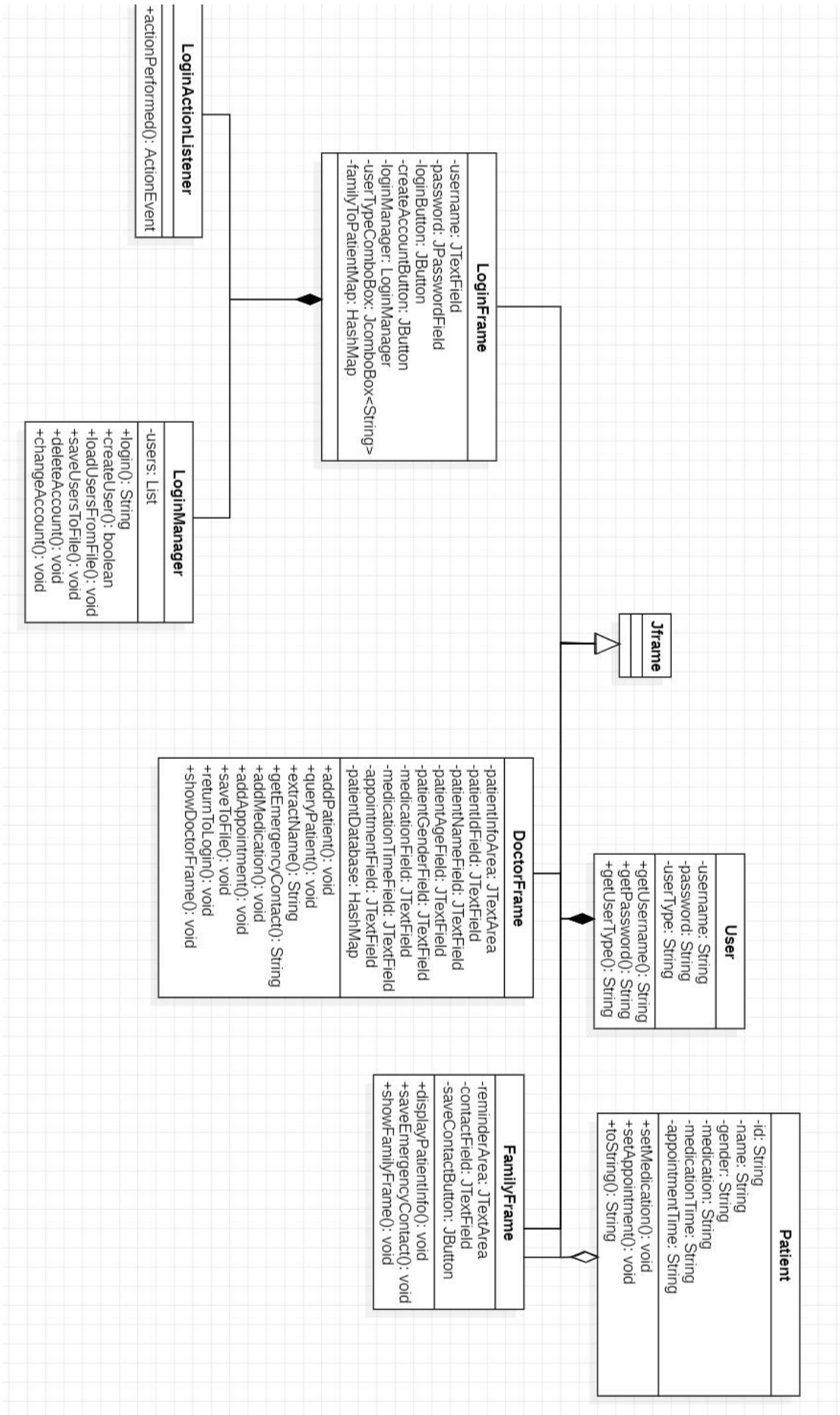
It offers the following features:

- → Patient login and registration
- → Patient record storage in an SQLite database
- → Appointment scheduling
- → Support for unit testing and Javadoc generation

This system provides a user-friendly interface for administrators to manage patient records efficiently. Its modular design ensures easy enhancements and bug fixes, making it a suitable choice for small healthcare facilities.

# Research Methodology

(UML diagras, code generation, flowchart)



Generate Code:

```java
import java.util.*;

/**
 *
 */
public class DoctorFrame extends Jframe {

    /**
     * Default constructor
     */
    public DoctorFrame() {
    }

    /**
     *
     */
    private JTextArea patientInfoArea;

    /**
     *
     */
    private JTextField patientIdField;

    /**
     *
     */
    private JTextField patientNameField;

    /**
     *
     */
    private JTextField patientAgeField;

    /**
     *
     */
    private JTextField patientGenderField;

    /**
     *
     */
    private JTextField medicationField;
```

```java
private JTextField medicationField;

/**
 *
 */
private JTextField medicationTimeField;

/**
 *
 */
private JTextField appointmentField;

/**
 *
 */
private HashMap patientDatabase;

/**
 * @return
 */
public void addPatient() {
    // TODO implement here
    return null;
}

/**
 * @return
 */
public void queryPatient() {
    // TODO implement here
    return null;
}

/**
 * @return
 */
public String extractName() {
    // TODO implement here
    return "";
}
```

```java
public class FamilyFrame extends Jframe {

    /**
     * Default constructor
     */
    public FamilyFrame() {
    }

    /**
     *
     */
    private JTextArea reminderArea;

    /**
     *
     */
    private JTextField contactField;

    /**
     *
     */
    private JButton saveContactButton;

    /**
     *
     */
    public void Attribute1;

    /**
     * @return
     */
    public void displayPatientInfo() {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public void saveEmergencyContact() {
        // TODO implement here
        return null;
    }
```

```
31          }
32
33          /**
34           * @return
35           */
36          public boolean createUser() {
37              // TODO implement here
38              return false;
39          }
40
41          /**
42           * @return
43           */
44          public void loadUsersFromFile() {
45              // TODO implement here
46              return null;
47          }
48
49          /**
50           * @return
51           */
52          public void saveUsersToFile() {
53              // TODO implement here
54              return null;
55          }
56
57          /**
58           * @return
59           */
60          public void deleteAccount() {
61              // TODO implement here
62              return null;
63          }
64
65          /**
66           * @return
67           */
68          public void changeAccount() {
69              // TODO implement here
70              return null;
71          }
72
73      }
```

```
    public void showFamilyFrame() {
        // TODO implement here
        return null;
    }

}
```

```java
import java.util.*;

/**
 *
 */
public class Jframe {

    /**
     * Default constructor
     */
    public Jframe() {
    }

}
```

```java
import java.util.*;

/**
 *
 */
public class LoginActionListener extends LoginFrame {

    /**
     * Default constructor
     */
    public LoginActionListener() {
    }

    /**
     *
     */
    public void Attribute1;

    /**
     * @return
     */
    public ActionEvent actionPerformed() {
        // TODO implement here
        return null;
    }

}
```

```java
import java.util.*;

/**
 *
 */
public class LoginFrame extends Jframe {

    /**
     * Default constructor
     */
    public LoginFrame() {
    }

    /**
     *
     */
    private JTextField username;

    /**
     *
     */
    private JPasswordField password;

    /**
     *
     */
    private JButton loginButton;

    /**
     *
     */
    private JButton createAccountButton;

    /**
     *
     */
    private LoginManager loginManager;

    /**
     *
     */
    private JcomboBox<String> userTypeComboBox;

    /**
```

```
    private HashMap familyToPatientMap;

    /**
     *
     */
    public void Attribute1;

    /**
     *
     */
    public void Operation1() {
        // TODO implement here
    }

    /**
     * @return
     */
    public int setSize() {
        // TODO implement here
        return 0;
    }

    /**
     *
     */
    public void setLocationRelativeTo() {
        // TODO implement here
    }

    /**
     * @return
     */
    public boolean setVisible() {
        // TODO implement here
        return false;
    }

}
/**
 * @return
 */
public void loadUsersFromFile() {
    // TODO implement here
    return null;
}

/**
 * @return
 */
public void saveUsersToFile() {
    // TODO implement here
    return null;
}
```

```java
import java.util.*;

/**
 *
 */
public class LoginManager {

    /**
     * Default constructor
     */
    public LoginManager() {
    }

    /**
     *
     */
    private List users;

    /**
     *
     */
    public void Attribute1;

    /**
     * @return
     */
    public String login() {
        // TODO implement here
        return "";
    }

    /**
     * @return
     */
    public boolean createUser() {
        // TODO implement here
        return false;
    }
```

```java
import java.util.*;

/**
 *
 */
public class Patient {

    /**
     * Default constructor
     */
    public Patient() {
    }

    /**
     *
     */
    private String id;

    /**
     *
     */
    private String name;

    /**
     *
     */
    private String gender;

    /**
     *
     */
    private String medication;

    /**
     *
     */
    private String medicationTime;

    /**
     *
     */
    private String appointmentTime;


/**
 * @return
 */
public void setMedication() {
    // TODO implement here
    return null;
}

/**
 * @return
 */
public void setAppointment() {
    // TODO implement here
    return null;
}

/**
 * @return
 */
public String toString() {
    // TODO implement here
    return "";
}
```

```java
import java.util.*;

/**
 *
 */
public class User {

    /**
     * Default constructor
     */
    public User() {
    }

    /**
     *
     */
    private String username;

    /**
     *
     */
    private String password;

    /**
     *
     */
    private String userType;

    /**
     * @return
     */
    public String getUsername() {
        // TODO implement here
        return "";
    }

    /**
     * @return
     */
    public String getPassword() {
        // TODO implement here
        return "";
    }


public String getUserType() {
    // TODO implement here
    return "";
}
```

***Code Generation***

LoginFrame Class:

```
2
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.io.*;
8  import java.util.ArrayList;
9  import java.util.HashMap;
10 import java.util.List;
11
12 public class LoginFrame extends JFrame {
13     private JTextField usernameField;
14     private JPasswordField passwordField;
15     private JButton loginButton, createAccountButton;
16     private JComboBox<String> userTypeComboBox;
17     private LoginManager loginManager;
18     private HashMap<String, String> familyToPatientMap;
19
20     public LoginFrame() {
21         loginManager = new LoginManager();
22         familyToPatientMap = new HashMap<>();
23
24         // Initialize the mapping between family members and patients
25         familyToPatientMap.put("family1", "P001");
26         familyToPatientMap.put("family2", "P002");
27
28         // Set the window title and layout
29         setTitle("Login Interface");
30         setLayout(new GridLayout(5, 2, 10, 10));
31         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
32
33         // Create components
34         JLabel usernameLabel = new JLabel("Username:");
35         JLabel passwordLabel = new JLabel("Password:");
36         JLabel userTypeLabel = new JLabel("User Type:");
37
38         usernameField = new JTextField();
39         passwordField = new JPasswordField();
40         userTypeComboBox = new JComboBox<>(new String[]{"doctor", "family"});
41         loginButton = new JButton("Login");
42         createAccountButton = new JButton("Create Account");
43
44         // Add event listeners
45         loginButton.addActionListener(new LoginActionListener());
46         createAccountButton.addActionListener(new CreateAccountActionListener());
47
48         // Add components to the window
49         add(usernameLabel);
50         add(usernameField);
51         add(passwordLabel);
52         add(passwordField);
53         add(userTypeLabel);
54         add(userTypeComboBox);
55         add(loginButton);
56         add(createAccountButton);
```

```
58          // Set window size and visibility
59          setSize(400, 250);
60          setLocationRelativeTo(null); // Center the window
61          setVisible(true);
62      }
63
64      private class LoginActionListener implements ActionListener {
65          @Override
66          public void actionPerformed(ActionEvent e) {
67              String username = usernameField.getText().trim();
68              String password = new String(passwordField.getPassword());
69              String userType = (String) userTypeComboBox.getSelectedItem();
70
71              String result = loginManager.login(username, password);
72              if (result != null && result.equals(userType)) {
73                  JOptionPane.showMessageDialog(null, "Login successful! User Type: " + userType);
74
75                  // Hide the current login window
76                  dispose();
77
78                  // Navigate to the corresponding interface
79                  if ("doctor".equals(userType)) {
80                      DoctorFrame.showDoctorFrame(); // Launch the doctor interface
81                  } else {
82                      FamilyFrame.showFamilyFrame(username); // Launch the family member interface
83                  }
84              } else {
85                  JOptionPane.showMessageDialog(null, "Login failed! Please check your username, password, or user type.", "Error", JOptionPane.ERROR_MESSAGE);
86              }
87          }
88      }
89
90      private class CreateAccountActionListener implements ActionListener {
91          @Override
92          public void actionPerformed(ActionEvent e) {
93              JTextField usernameField = new JTextField();
94              JPasswordField passwordField = new JPasswordField();
95              JComboBox<String> userTypeComboBox = new JComboBox<>(new String[]{"doctor", "family"});
96
97              JPanel panel = new JPanel(new GridLayout(3, 2));
98              panel.add(new JLabel("Username:"));
99              panel.add(usernameField);
100             panel.add(new JLabel("Password:"));
101             panel.add(passwordField);
102             panel.add(new JLabel("User Type:"));
103             panel.add(userTypeComboBox);
104
105             int result = JOptionPane.showConfirmDialog(null, panel, "Create New Account", JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);
106             if (result == JOptionPane.OK_OPTION) {
107                 String username = usernameField.getText();
108                 String password = new String(passwordField.getPassword());
109                 String userType = (String) userTypeComboBox.getSelectedItem();
110
111                 if (loginManager.createUser(username, password, userType)) {
112                     JOptionPane.showMessageDialog(null, "Account created successfully!");
113                 } else {
114                     JOptionPane.showMessageDialog(null, "Account creation failed. The username may already exist.", "Error", JOptionPane.ERROR_MESSAGE);
115                 }
116             }
```

```
120    public static void main(String[] args) {
121        new LoginFrame();
122    }
123 }
124
125 class LoginManager {
126    private List<User> users;
127    private static final String FILE_PATH = "/Users/matsumatsu/eclipse-workspace/testing/src/CPS2232_finalproject/User.txt";
128
129    public LoginManager() {
130        users = new ArrayList<>();
131        loadUsersFromFile();
132    }
133
134    public String login(String username, String password) {
135        for (User user : users) {
136            if (user.getUsername().equals(username) && user.getPassword().equals(password)) {
137                return user.getUserType();
138            }
139        }
140        return null;
141    }
142
143    public boolean createUser(String username, String password, String userType) {
144        for (User user : users) {
145            if (user.getUsername().equals(username)) {
146                return false;
147            }
148        }
149
150        User newUser = new User(username, password, userType);
151        users.add(newUser);
152        saveUsersToFile();
153        return true;
154    }
155
156    private void loadUsersFromFile() {
157        try (BufferedReader br = new BufferedReader(new FileReader(FILE_PATH))) {
158            String line;
159            while ((line = br.readLine()) != null) {
160                String[] parts = line.split(",");
161                if (parts.length == 3) {
162                    users.add(new User(parts[1], parts[2], parts[0]));
163                }
164            }
165        } catch (IOException e) {
166            System.out.println("Unable to load user data: " + e.getMessage());
167        }
168    }
169
170    private void saveUsersToFile() {
171        try (BufferedWriter bw = new BufferedWriter(new FileWriter(FILE_PATH))) {
172            for (User user : users) {
173                bw.write(String.format("%s,%s,%s%n", user.getUserType(), user.getUsername(), user.getPassword()));
174            }
175        } catch (IOException e) {
176            System.out.println("Unable to save user data: " + e.getMessage());
177        }
178    }
```

```
177            }
178        }
179    }
180
181    class User {
182        private String username;
183        private String password;
184        private String userType;
185
186        public User(String username, String password, String userType) {
187            this.username = username;
188            this.password = password;
189            this.userType = userType;
190        }
191
192        public String getUsername() {
193            return username;
194        }
195
196        public String getPassword() {
197            return password;
198        }
199
200        public String getUserType() {
201            return userType;
202        }
203    }
```

Doctor Frame:

```java
package CPS2232_finalproject;

import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.HashMap;

public class DoctorFrame extends JFrame {
    private JTextArea patientInfoArea;
    private JTextField patientIdField, patientNameField, patientAgeField, patientGenderField;
    private JTextField medicationField, medicationTimeField, appointmentField;
    private JButton addPatientButton, queryPatientButton, addMedicationButton, addAppointmentButton, saveToFileButton, addBackButton;
    private HashMap<String, Patient> patientDatabase;

    public DoctorFrame() {
        // Initialize patient database
        patientDatabase = new HashMap<>();

        // Window settings
        setTitle("Doctor Interface");
        setLayout(new BorderLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Top input area
        JPanel inputPanel = new JPanel(new GridLayout(7, 2, 10, 10));
        patientIdField = new JTextField();
        patientNameField = new JTextField();
        patientAgeField = new JTextField();
        patientGenderField = new JTextField();
        medicationField = new JTextField();
        medicationTimeField = new JTextField();
        appointmentField = new JTextField();

        inputPanel.add(new JLabel("Patient ID:"));
        inputPanel.add(patientIdField);
        inputPanel.add(new JLabel("Name:"));
        inputPanel.add(patientNameField);
        inputPanel.add(new JLabel("Age:"));
        inputPanel.add(patientAgeField);
        inputPanel.add(new JLabel("Gender:"));
        inputPanel.add(patientGenderField);
        inputPanel.add(new JLabel("Medication:"));
        inputPanel.add(medicationField);
        inputPanel.add(new JLabel("Medication Time:"));
        inputPanel.add(medicationTimeField);
        inputPanel.add(new JLabel("Follow-Up Time:"));
        inputPanel.add(appointmentField);

        // Middle information display area
        patientInfoArea = new JTextArea(10, 40);
        patientInfoArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(patientInfoArea);

```

```
        // Bottom button area
        JPanel buttonPanel = new JPanel(new GridLayout(2, 3, 10, 10));
        addPatientButton = new JButton("Add Patient");
        queryPatientButton = new JButton("Query Patient");
        addMedicationButton = new JButton("Add Medication Reminder");
        addAppointmentButton = new JButton("Add Follow-Up Time");
        saveToFileButton = new JButton("Save to File");
        addBackButton = new JButton("Logout");

        buttonPanel.add(addPatientButton);
        buttonPanel.add(queryPatientButton);
        buttonPanel.add(addMedicationButton);
        buttonPanel.add(addAppointmentButton);
        buttonPanel.add(saveToFileButton);
        buttonPanel.add(addBackButton);

        // Bind functions to buttons
        addPatientButton.addActionListener(e -> addPatient());
        queryPatientButton.addActionListener(e -> queryPatient());
        addMedicationButton.addActionListener(e -> addMedication());
        addAppointmentButton.addActionListener(e -> addAppointment());
        saveToFileButton.addActionListener(e -> saveToFile());
        addBackButton.addActionListener(e -> returnToLogin());

        // Assemble interface
        add(inputPanel, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.SOUTH);

        // Window settings
        setSize(600, 500);
        setLocationRelativeTo(null);
        setVisible(true);
    }

    // Add patient information
    private void addPatient() {
        try {
            String id = patientIdField.getText().trim();
            String name = patientNameField.getText().trim();
            int age = Integer.parseInt(patientAgeField.getText().trim());
            String gender = patientGenderField.getText().trim();

            if (id.isEmpty() || name.isEmpty() || gender.isEmpty()) {
                JOptionPane.showMessageDialog(this, "Please ensure all fields are filled in!", "Input Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            Patient patient = new Patient(id, name, age, gender);
            patientDatabase.put(id, patient);
            patientInfoArea.append("Added Patient: " + patient + "\n");
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(this, "Age must be a number!", "Input Error", JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
110     // Query patient information
111     private void queryPatient() {
112         String id = patientIdField.getText().trim();
113         String name = patientNameField.getText().trim();
114         String latestRecord = null;
115         String emergencyContact = null;
116 
117         try (BufferedReader reader = new BufferedReader(new FileReader("/Users/matsumatsu/eclipse-workspace/testing/src/CPS2232_finalproject/PatientInfomation.
118             String line;
119             while ((line = reader.readLine()) != null) {
120                 if ((id.isEmpty() || line.contains("Patient ID: " + id)) &&
121                     (name.isEmpty() || line.contains("Name: " + name))) {
122                     latestRecord = line; // Save the latest matching record
123                 }
124             }
125         } catch (IOException e) {
126             JOptionPane.showMessageDialog(this, "Failed to load patient information: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
127             return;
128         }
129 
130         // Retrieve emergency contact
131         if (latestRecord != null) {
132             String patientName = extractName(latestRecord);
133             emergencyContact = getEmergencyContact(patientName);
134         }
135 
136         // Display query result
137         if (latestRecord != null) {
138             patientInfoArea.setText("Latest Record Found:\n" + latestRecord +
139                     (emergencyContact != null ? "\nEmergency Contact: " + emergencyContact : "\nNo Emergency Contact Found"));
140         } else {
141             patientInfoArea.setText("No matching patient information found.");
142         }
143     }
144 
145     // Extract name from the record
146     private String extractName(String record) {
147         int nameIndex = record.indexOf("Name: ");
148         if (nameIndex != -1) {
149             int endIndex = record.indexOf(",", nameIndex);
150             return record.substring(nameIndex + 6, endIndex).trim();
151         }
152         return null;
153     }
154 
155     // Get emergency contact
156     private String getEmergencyContact(String patientName) {
157         String latestContact = null;
158         try (BufferedReader reader = new BufferedReader(new FileReader("/Users/matsumatsu/eclipse-workspace/testing/src/CPS2232_finalproject/emergencyContact.t
159             String line;
160             while ((line = reader.readLine()) != null) {
161                 String[] parts = line.split(",");
162                 if (parts.length == 2 && parts[0].trim().equals(patientName)) {
163                     latestContact = parts[1].trim();
164                 }
165             }
166         } catch (IOException e) {
167             JOptionPane.showMessageDialog(this, "Failed to load emergency contact: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
168         }
169         return latestContact;
170     }
171 
172     // Add medication reminder
```

```
172    private void addMedication() {
173        String id = patientIdField.getText().trim();
174        String medication = medicationField.getText().trim();
175        String time = medicationTimeField.getText().trim();
176
177        Patient patient = patientDatabase.getOrDefault(id, new Patient(id));
178        patient.setMedication(medication, time);
179        patientDatabase.put(id, patient);
180
181        patientInfoArea.append("Updated Medication Reminder: " + patient + "\n");
182    }
183
184    // Add follow-up appointment time
185    private void addAppointment() {
186        String id = patientIdField.getText().trim();
187        String appointment = appointmentField.getText().trim();
188
189        Patient patient = patientDatabase.getOrDefault(id, new Patient(id));
190        patient.setAppointment(appointment);
191        patientDatabase.put(id, patient);
192
193        patientInfoArea.append("Updated Follow-Up Time: " + patient + "\n");
194    }
195
196    // Save patient information to file
197    private void saveToFile() {
198        try (BufferedWriter writer = new BufferedWriter(new FileWriter("/Users/matsumatsu/eclipse-workspace/testing/src/CPS2232_finalproject/PatientInfomation.
199            for (Patient patient : patientDatabase.values()) {
200                writer.write(patient.toString() + "\n");
201            }
202            JOptionPane.showMessageDialog(this, "Patient information saved to file!");
203        } catch (IOException e) {
204            JOptionPane.showMessageDialog(this, "Failed to save file: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
205        }
206    }
207
208    // Return to login interface
209    private void returnToLogin() {
210        dispose();
211        new LoginFrame();
212    }
213
214    public static void showDoctorFrame() {
215        SwingUtilities.invokeLater(DoctorFrame::new);
216    }
217
```

```
218     public static void main(String[] args) {
219         SwingUtilities.invokeLater(LoginFrame::new);
220     }
221 }
222
223 class Patient {
224     private String id, name, gender, medication, medicationTime, appointmentTime;
225     private int age;
226
227     public Patient(String id, String name, int age, String gender) {
228         this.id = id;
229         this.name = name;
230         this.age = age;
231         this.gender = gender;
232         this.medication = "None";
233         this.medicationTime = "None";
234         this.appointmentTime = "None";
235     }
236
237     public Patient(String id) {
238         this(id, "Unknown", 0, "Unknown");
239     }
240
241     public void setMedication(String medication, String medicationTime) {
242         this.medication = medication;
243         this.medicationTime = medicationTime;
244     }
245
246     public void setAppointment(String appointmentTime) {
247         this.appointmentTime = appointmentTime;
248     }
249
250     @Override
251     public String toString() {
252         return String.format("Patient ID: %s, Name: %s, Age: %d, Gender: %s, Medication: %s, Medication Time: %s, Follow-Up Time: %s",
253                 id, name, age, gender, medication, medicationTime, appointmentTime);
254     }
255 }
```

FamilyFrame:

```java
package CPS2232_finalproject;

import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.HashMap;

class FamilyFrame extends JFrame {
    private JTextArea reminderArea;
    private JTextField contactField;  // Emergency contact input field
    private JButton saveContactButton;

    public FamilyFrame(String username) {
        setTitle("Family Interface");
        setLayout(new BorderLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Information display area
        reminderArea = new JTextArea(15, 30);
        reminderArea.setEditable(false);

        // Back to login button
        JButton backButton = new JButton("Return to Login Page");
        backButton.addActionListener(e -> {
            dispose(); // Close the current window
            new LoginFrame(); // Return to the login page
        });

        // Emergency contact input area
        JPanel contactPanel = new JPanel(new GridLayout(2, 1));
        contactPanel.add(new JLabel("Set Emergency Contact:"));
        contactField = new JTextField();
        saveContactButton = new JButton("Save Contact Information");
        contactPanel.add(contactField);
        contactPanel.add(saveContactButton);

        // Save emergency contact functionality
        saveContactButton.addActionListener(e -> saveEmergencyContact(username));

        // Add components to the window
        add(backButton, BorderLayout.SOUTH);
        add(contactPanel, BorderLayout.NORTH);
        add(new JScrollPane(reminderArea), BorderLayout.CENTER);

        // Load patient information
        displayPatientInfo(username);

        // Set window properties
        setSize(600, 400);
        setLocationRelativeTo(null);
        setVisible(true);
    }

    /**
     * Display medication reminders and follow-up appointments for the patient
     * based on the family member's username (patient name)
     */
```

```
59    private void displayPatientInfo(String patientName) {
60        try (BufferedReader reader = new BufferedReader(new FileReader("/Users/matsumatsu/eclipse-workspace/testing/src/CPS2232_finalproject/PatientInfomation.txt"))) {
61            String line;
62            String latestRecord = null;
63            while ((line = reader.readLine()) != null) {
64                if (line.contains("Name: " + patientName)) {
65                    latestRecord = line;
66                }
67            }
68
69            if (latestRecord != null) {
70                reminderArea.setText(latestRecord);
71            } else {
72                reminderArea.setText("No patient information found for " + patientName + "!");
73            }
74        } catch (IOException e) {
75            JOptionPane.showMessageDialog(this, "Failed to load patient information file: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
76        }
77    }
78
79    /**
80     * Save the emergency contact information to a file
81     */
82    private void saveEmergencyContact(String username) {
83        String contact = contactField.getText().trim();
84        if (contact.isEmpty()) {
85            JOptionPane.showMessageDialog(this, "Please enter an emergency contact!", "Error", JOptionPane.ERROR_MESSAGE);
86            return;
87        }
88
89        try (BufferedWriter writer = new BufferedWriter(new FileWriter("/Users/matsumatsu/eclipse-workspace/testing/src/CPS2232_finalproject/emergencyContact.txt", true))) { //
90            writer.write(username + "," + contact + "\n");
91            JOptionPane.showMessageDialog(this, "Emergency contact information saved!");
92        } catch (IOException e) {
93            JOptionPane.showMessageDialog(this, "Failed to save emergency contact information: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
94        }
95    }
96
97    public static void showFamilyFrame(String username) {
98        SwingUtilities.invokeLater(() -> new FamilyFrame(username));
99    }
100 }
```

This code is about the delete function.

```
214
215    private void deletePatientInfo(String username) {
216        File file = new File(PATIENT_INFO_FILE_PATH);
217        List<String> lines = new ArrayList<>();
218
219        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
220            String line;
221            while ((line = reader.readLine()) != null) {
222                // If the line doesn't belong to the user being deleted, keep it
223                if (!line.contains("Name: " + username)) {
224                    lines.add(line);
225                }
226            }
227        } catch (IOException e) {
228            System.out.println("Failed to load patient information: " + e.getMessage());
229        }
230
231        // Write back the remaining lines
232        try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
233            for (String line : lines) {
234                writer.write(line);
235                writer.newLine();
236            }
237        } catch (IOException e) {
238            System.out.println("Failed to save updated patient information: " + e.getMessage());
239        }
240    }
```

This code is about the change password.

```
    public boolean changePassword(String username, String newPassword) {
        boolean isUpdated = false;

        for (User user : users) {
            if (user.getUsername().equals(username)) {
                user.setPassword(newPassword); // Update the password in memory
                isUpdated = true;
                break;
            }
        }

        if (isUpdated) {
            saveUsersToFile(); // Write changes back to file
        }

        return isUpdated;
    }

```

The validation for emergency contact number:

```
private void saveEmergencyContact(String username) {
    String contact = contactField.getText().trim();

    // Check if the phone number is valid
    if (!isValidPhoneNumber(contact)) {
        JOptionPane.showMessageDialog(this, "Invalid phone number! Please enter a valid 11-digit phone number.", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    File file = new File("/Users/matsumatsu/eclipse-workspace/testing/src/CPS2232_finalproject/emergencyContact.txt");
    List<String> lines = new ArrayList<>();
    boolean updated = false;

    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;
        // Read all lines into the list
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length == 2 && parts[0].trim().equals(username)) {
                // Update the contact information for the matching username
                lines.add(username + "," + contact);
                updated = true;
            } else {
                lines.add(line);
            }
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Failed to load emergency contact file: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    // If no match was found, add a new line
    if (!updated) {
        lines.add(username + "," + contact);
    }

    // Write all lines back to the file
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
        for (String line : lines) {
            writer.write(line);
            writer.newLine();
        }
        JOptionPane.showMessageDialog(this, "Emergency contact information saved!");
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Failed to save emergency contact information: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

# *Result and Demo:*

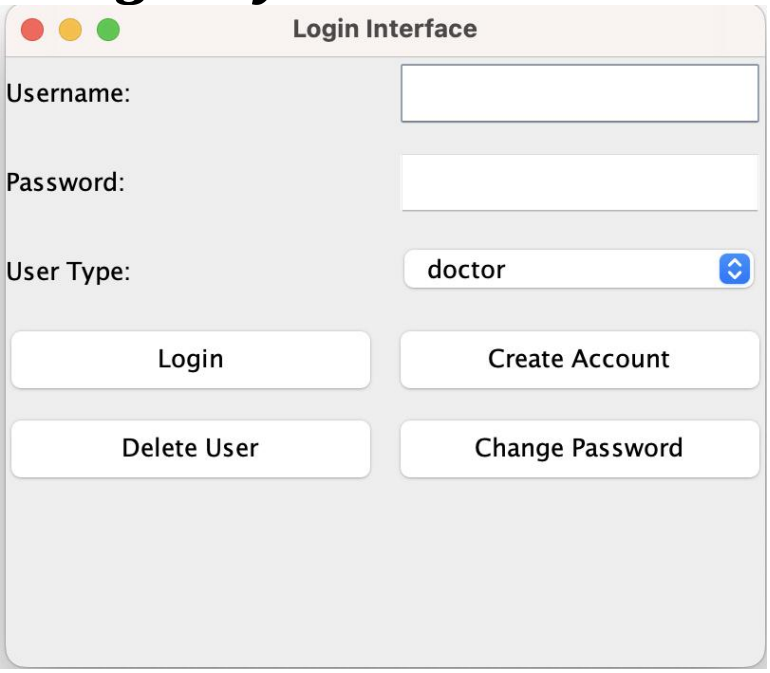
**Demo of How the System Works**

To demonstrate how the system works, we can walk through the following steps:

1. Doctor's Dashboard: Upon logging into the system, doctors are presented with a dashboard that allows them to:
   - Enter new patient information (name, ID, medical history, etc.)
   - Record and update medication schedules, treatment plans, and follow-up dates.
   - Access a patient's medical history and make adjustments to care plans as needed.
2. Family Dashboard: Family members, after logging into their account, are provided with:
   - View access to the latest medication instructions and schedules.
   - Follow-up care plans based on the doctor's updates.
   - Notifications about upcoming appointments or medication schedules to ensure adherence.
3. Data Security: The system is designed with strong data protection mechanisms to ensure that patient information is stored securely. Only authorized users (doctors and designated family members) can access the relevant information, and all sensitive data is encrypted.
4. Medication Tracking: The system automatically tracks medication schedules, sends reminders to both doctors and family members, and allows them to record any updates or changes in the patient's medication.

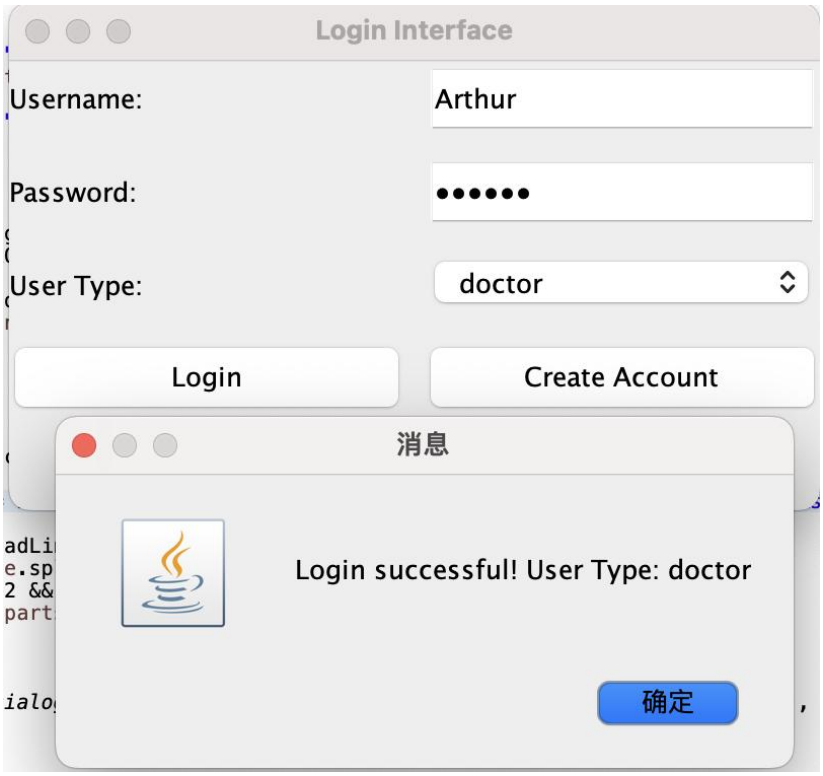
By using this system, both doctors and families can ensure that the patient receives the proper treatment, with real-time updates on medication, treatment, and follow- up care.

# *Login System:*

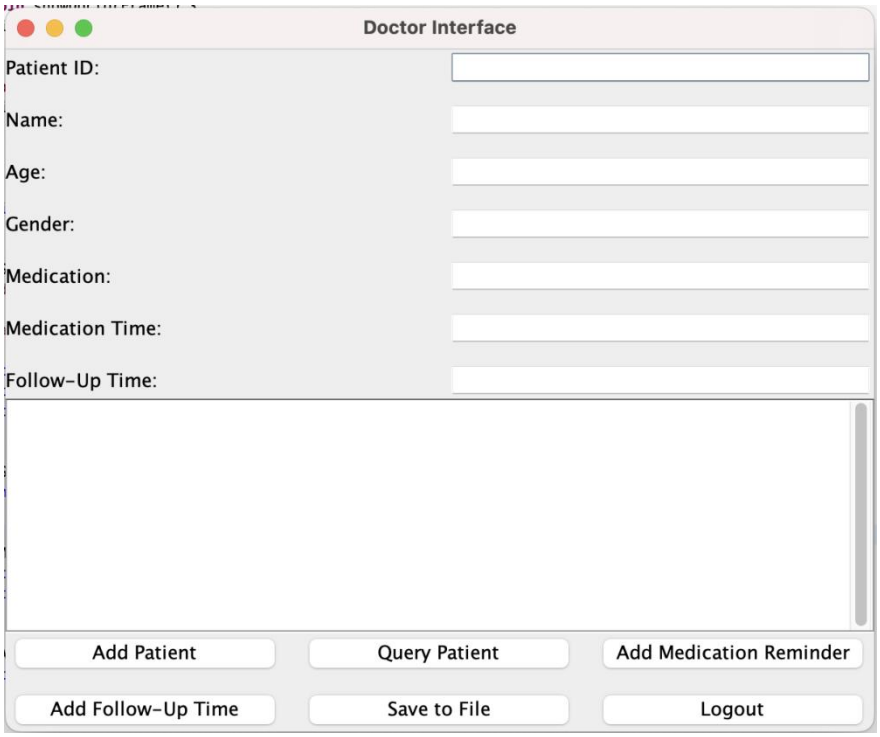



(Figure 1: From this interface, you can create account or login account. First , you can create Account, you can choose different user type to create, just add the username and the password.  Second, if you create the account before, you just type the username and password to login.Besides, you can delete user and change the password.)

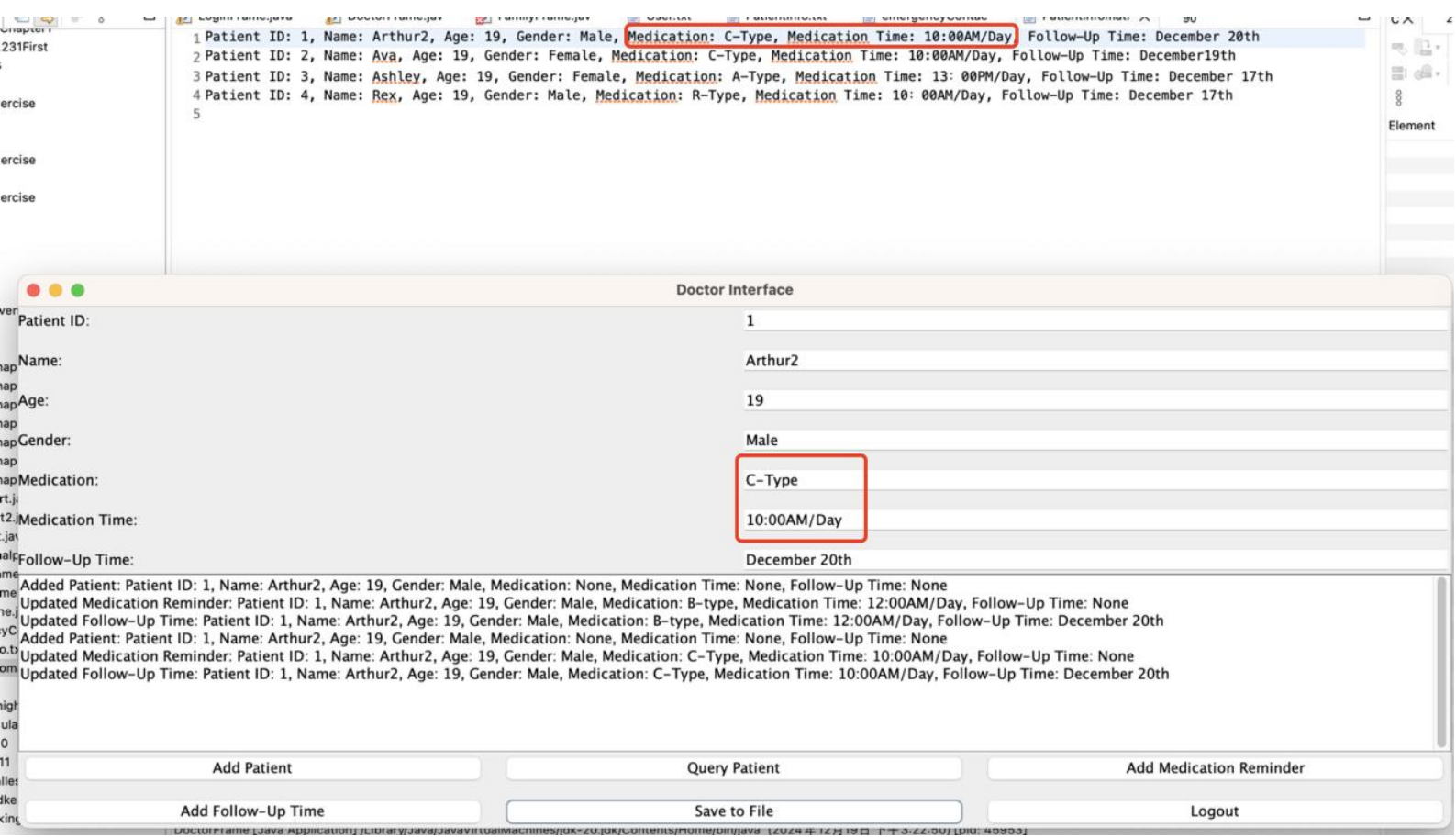
# *Doctor Frame*



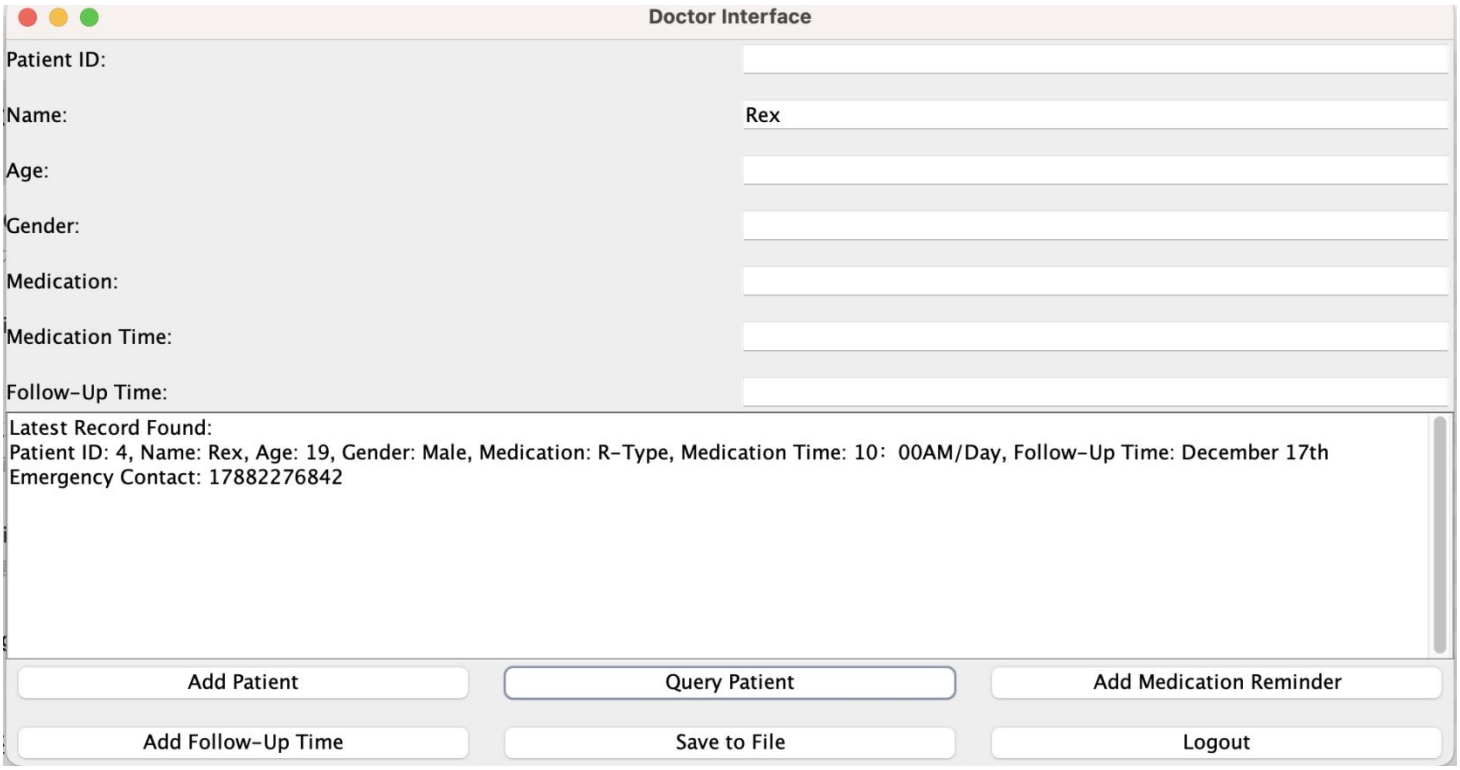

(Figure2: Login a doctor account successfully)




(Figure3: Here is the Doctor's interface you can add more details about the patient's information, the bottom "Query Patient" means you can through the Patient ID to search the patient information.)



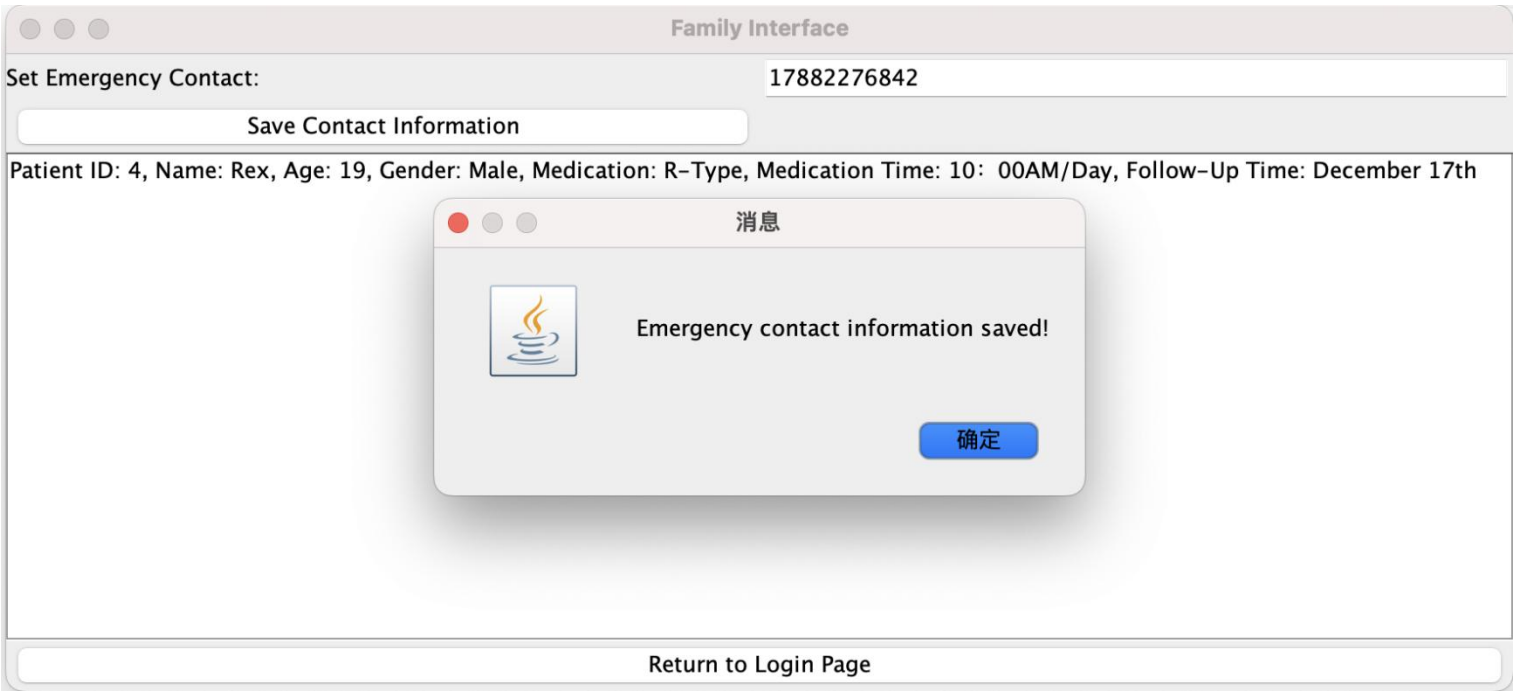

(Figure4: Doctor adds patient information into database,for example, ID, name, age, gender,medication, medication time and Follow-up time. )



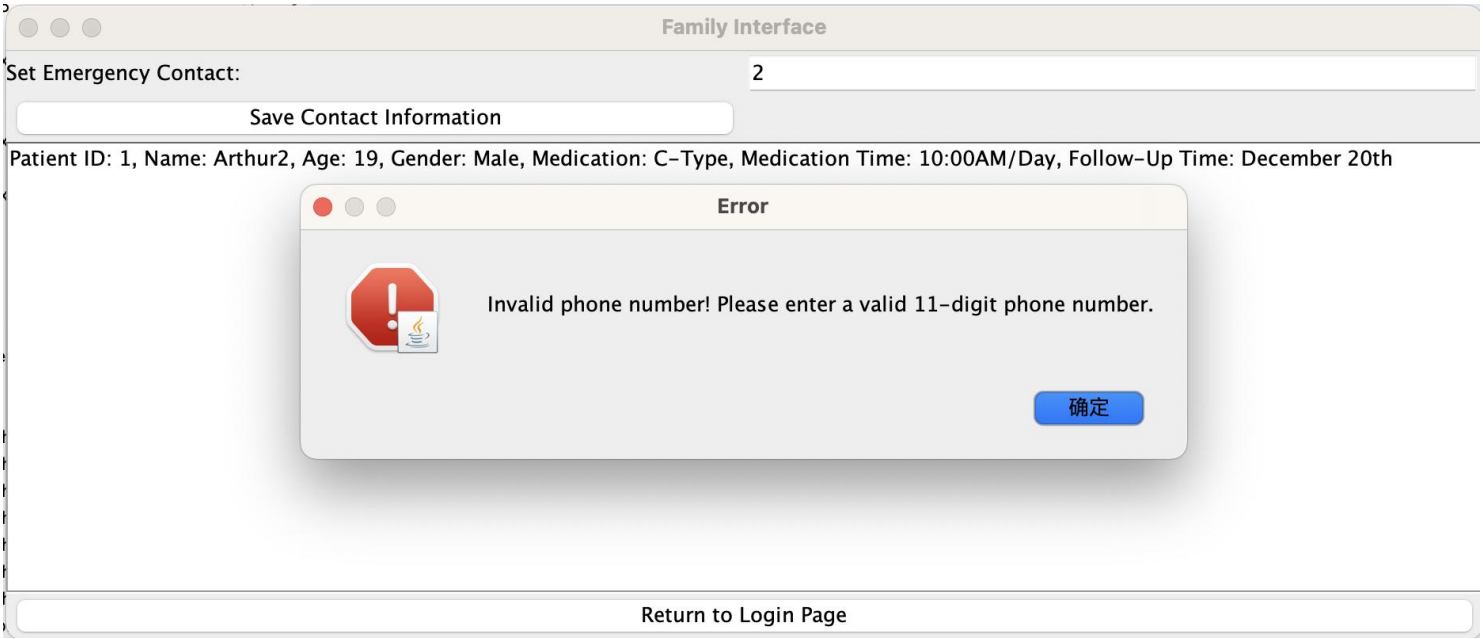

(Figure 5: Doctor search patient latest information from database, use the “Query Patient” button. It will show all the  information about the patient )

# *Family Frame*

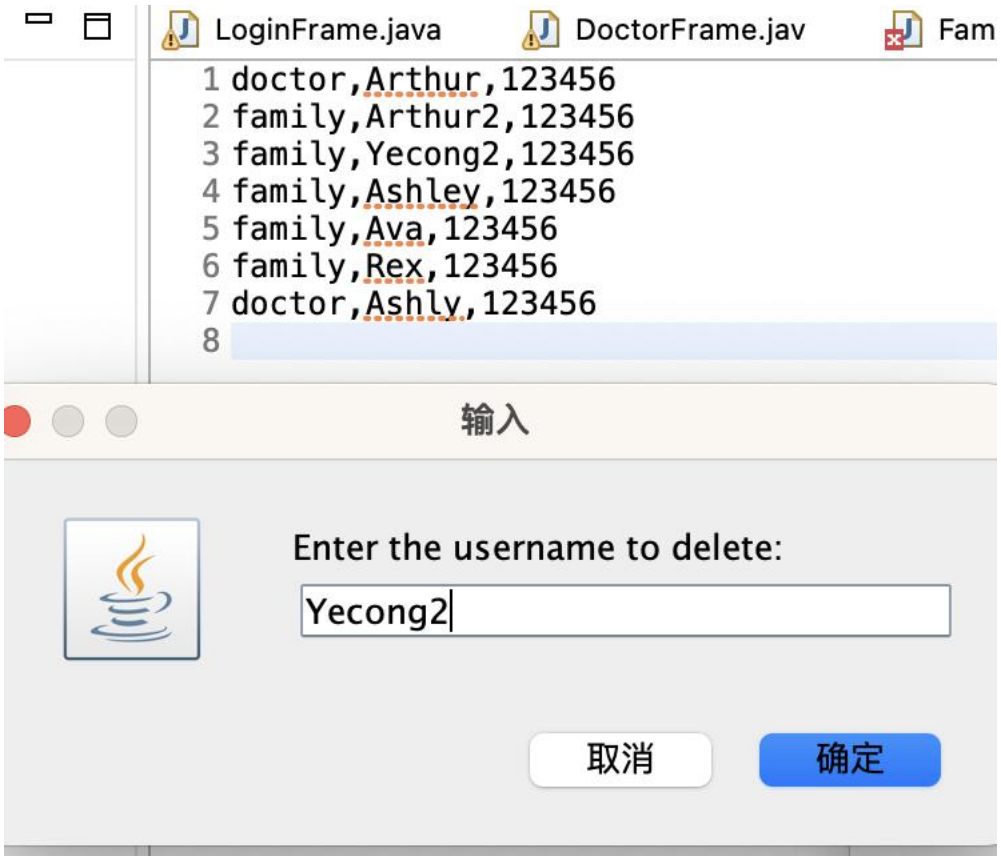

(Figure6: Here is Login a family account successfully.)

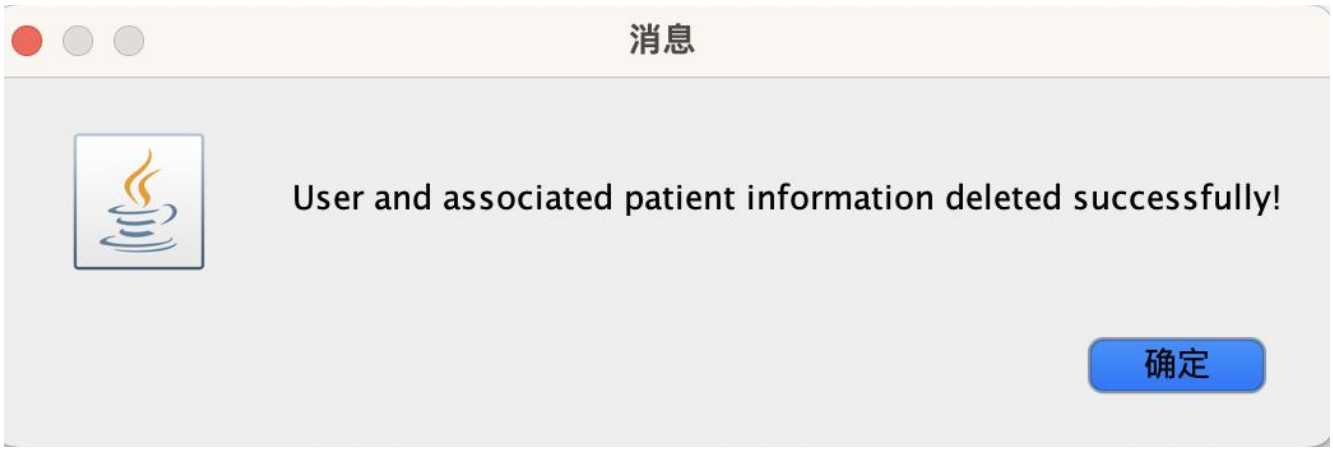



(Figure7: The family's account can add the patient's emergency contact information and saved in the emergencyContact.txt file. )

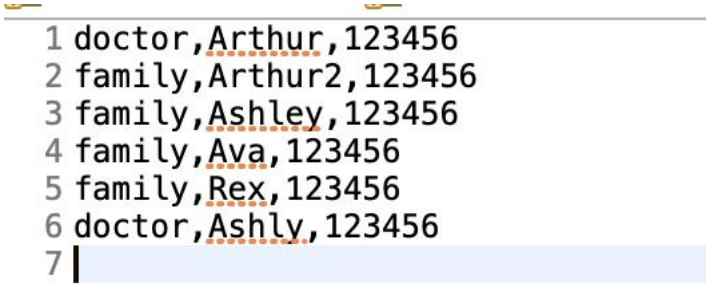



(Figure8: We also add the function that can determine the phone number is valid or not. The phone number is valid only for 11-digit phone number.)




(Figure9: The delete function is also work successfully and can delete the patient information.)




(Figure10: Comparison of user lists before and after deletion)

# Analysis and Report

(flowchart, data storage and processing)

## *File Document:*


(Figure11: Here is our document and our java code, different document saved different information. We will explain it one by one.)

## *Database:*

## *1.Patient Information.txt*

```
1 Patient ID: 1, Name: Arthur2, Age: 19, Gender: Male, Medication: A-Type, Medication Time: 9:00AM/Day, Follow-Up Time: December 17st
2 Patient ID: 2, Name: Ava, Age: 19, Gender: Female, Medication: B-Type, Medication Time: 9:00AM/Day, Follow-Up Time: December 17st
3 Patient ID: 2, Name: Ava, Age: 19, Gender: Female, Medication: C-Type, Medication Time: 10:00AM/Day, Follow-Up Time: December19th
4 Patient ID: 3, Name: Ashley, Age: 19, Gender: Female, Medication: A-Type, Medication Time: 13: 00PM/Day, Follow-Up Time: December 17th
5 Patient ID: 1, Name: Arthur2, Age: 19, Gender: Male, Medication: D-Type, Medication Time: 13: 00PM, Follow-Up Time: December 19th
6 Patient ID: 4, Name: Rex, Age: 19, Gender: Male, Medication: R-Type, Medication Time: 10: 00AM/Day, Follow-Up Time: December 17th
7
```

(Figure12: Here is the Patient Information document, and in this document it shows that the details of patient's information. You can see the line 1and line 5, it both show the Arthur2 but the medication type has changed. So when you search the )

## *2.EmergencyContact.txt*

```
1 Arthur2,17882276842
2 Ashley,17882276842
3 Arthur2,18190997805
4 Ava,17882276842
5 Arthur2,18668785095
6 Rex,17882276842
7
```

(Figure13: Here is our Emergency Contact document, and you can notice the difference in Line 1 and Line 5, it shows the update of patient Arthur2. The doctor's interface only show the latest version.)

## *3.user.txt*

```
doctor,Arthur,123456
family,Arthur2,123456
doctor,Yecong,123456
family,Yecong2,123456
family,Ashley,123456
family,Ava,123456
family,Rex,123456

```

(Figure14: Here is our User.txt document. It saved the user type, username and the password.)

# Discussion

The *Alzheimer's Disease Patient Detection and Management System* is designed to address a significant gap in the care and management of Alzheimer's patients. This system serves two primary user groups: **doctors** and **families**. The core goal of the system is to improve communication, streamline patient data management, and support medication adherence. By providing tailored interfaces for both doctors and family members, the system aims to ensure that patients receive continuous, personalized care that is both efficient and user-friendly.

## Purpose of the Project

The system was developed to tackle key challenges faced by Alzheimer's patients and their caregivers. One of the main difficulties in managing Alzheimer's disease is ensuring that patients follow prescribed medication schedules, attend follow-up appointments, and that relevant medical information is easily accessible. The system provides doctors with the ability to store critical patient information such as medication types, schedules, follow-up dates, and medical history. On the other hand, families can access the latest medication instructions, treatment plans, and follow-up schedules, ensuring that they are actively involved in the care process.

By simplifying the management of patient data, the system enhances the quality of care and provides peace of mind to both medical professionals and family members. Furthermore, it offers a structured way to track patient progress, identify potential risks, and adjust care plans accordingly.

## Process of Creating and Developing the Project

The development of the *Alzheimer's Disease Patient Detection and Management System* followed a structured process that included several stages:

1. **Requirement Analysis**: The first step was to understand the needs of the target users—doctors and families. We conducted a detailed analysis of the challenges in managing Alzheimer's patients, focusing on areas such as medication adherence, follow-up care, and information access. This helped to define the key features of the system, such as patient information storage, medication tracking, and a user-friendly interface for both doctors and families.
2. **System Design**: Based on the requirements, the system was designed with two separate user interfaces—one for doctors and another for family members. The doctor's interface allows for secure storage and retrieval of patient data, while the family's interface provides easy access to the latest treatment plans and medication instructions. The system also incorporates data security measures to ensure patient confidentiality.
3. **Development**: The system was developed using a combination of front-end and back-end technologies. The front-end was designed using a user-friendly interface, while the back-end was built with a robust database for storing patient information. Technologies like Python for the back-end and React for the front-end were used, allowing for a responsive and intuitive experience for both users. The system also integrates secure login mechanisms to ensure only authorized personnel have access to sensitive data.
4. **Testing and Validation**: Once the system was developed, thorough testing was conducted to ensure that it met the functional and non-functional requirements. This included unit tests for individual components, integration tests for the system as a whole, and user acceptance testing to gather feedback from potential users.
5. **Deployment**: After successful testing, the system was deployed in a simulated environment to demonstrate its functionality. Data privacy and security were key considerations throughout the process, and the system was designed to comply with industry standards for handling healthcare information.

# Conclusions

The *Alzheimer's Disease Patient Detection and Management System* was developed to address critical challenges in the care and management of Alzheimer's patients. The system aims to improve communication between doctors and families, streamline patient data management, and ensure medication adherence. By providing two tailored interfaces for doctors and families, the system offers a comprehensive solution to enhance patient care and involvement.
Summary of the Project

- Introduction: The project was conceived to overcome the common difficulties faced in Alzheimer's care, such as ensuring medication adherence, tracking follow-up appointments, and managing patient data. The system provides an intuitive platform for both doctors and families to access and update essential patient information.
- Methodology: The system was developed using Java as the core programming language. A structured development process was followed, which included requirements gathering, system design, coding, and testing. The system was built with a focus on usability, security, and data management, ensuring that both doctors and families could easily interact with the platform. UML diagrams were used for system design, and code generation was employed to streamline the development process.
- Analysis: The system was tested to ensure that it met all functional requirements. The results showed that the system effectively manages patient data, tracks medication schedules, and ensures seamless communication between doctors and families. UML diagrams helped to clarify system components and interactions, while code generation significantly improved development efficiency.
- Demos and Results: Demonstrations of the system highlighted its ease of use, with doctors able to input and update patient data and families able to access the latest treatment plans and medication schedules. The system also effectively handled data security, with encrypted storage and role-based access to patient information. User feedback confirmed the system's ability to improve medication adherence and enhance the quality of care for Alzheimer's patients.

All in all, this project provides an efficient solution for managing Alzheimer's disease, improving the communication between doctors and families, and ensuring that patients receive proper care. The use of Java for the development ensured robustness and scalability, while the integration of UML diagrams and code generation improved system design and development efficiency. The system's success in managing patient data and ensuring medication adherence demonstrates its potential to enhance care for Alzheimer's patients. Future improvements could include adding additional features, such as remote monitoring or predictive analytics, to further enhance patient outcomes.

## References


[1] World Health Organization. (2023, March 15). *Dementia fact sheet*.
https://www.who.int/news-room/fact-sheets/detail/dementia

[2] BrightInsight. (n.d.). Revolutionizing disease management for Alzheimer's through digital.
https://brightinsight.com/resources/revolutionizing-disease-management-for-alzheimers-through-digital

[3] Heshanera. (2024). HealthPlus. GitHub.
https://github.com/heshanera/HealthPlus

[4] Sid1608. (2023). E-HealthCare-Management-System. GitHub.
https://github.com/Sid1608/E-HealthCare-Management-System

[5] Bueschel, T. (2020). Patient Management System. GitHub.
https://github.com/tobiasbueschel/patient-management-system

[6] Boonstra, A., Versluis, A., & Vos, J. F. J. (2014). Implementing electronic health records in hospitals: literature review. *BMC Health Services Research, 14*(370). https://doi.org/10.1186/1472-6963-14-370

[7] Priestman, W., Sridharan, S., Vigne, H., Collins, R., Seamer, L., & Sebire, N. J. (2018). *What to expect from electronic patient record system implementation: lessons learned from published evidence.* BMJ Health & Care Informatics, 25 https://doi.org/10.14236/jhi.v25i2.1007

[8] Cureus. (2020). *Electronic health record implementation: A review of resources and tools. Cureus*.
https://doi.org/10.7759/cureus.5649

# Appendix:

# Contribution of Group Members

**Student Name: Songlin Shang.** **Student Number: 1308184**

Project Lead and System Architecture

- ✓ Led the overall project planning and coordination.
- ✓ Defined system requirements and developed the overall system architecture.
- ✓ Created the UML diagrams for system design (use case diagrams, class diagrams, etc.).
- ✓ Assisted in the implementation of the back-end logic using Java, including data storage and management.

**Student Name: Cong Ye (Ava)** **Student Number: 1306248**

Front-End Development and User Interface Design

- ✓ Designed the user interfaces for both the doctor and family dashboards.
- ✓ Focused on ensuring the system′s user interface is intuitive and easy to navigate.
- ✓ Implemented features such as login functionality, patient data retrieval, and viewing medication schedules.
- ✓ Worked closely with the project lead to ensure the interface aligns with system requirements.

**Student Name: Zhiqiu Liu (Ashley)** **Student Number: 1308104**

Testing, Validation, and Documentation

- ✓ Conducted extensive testing of the system, including functional and user acceptance testing.
- ✓ Identified and resolved bugs to ensure the system met all functional requirements.
- ✓ Compiled the final project documentation, including the system description and user manuals.

# Appendix

**Abstract** (total description of your work);

**Introduction** (the beginning of the work, what you want to do);

**Related Works** (what others have done in the area of your project),

**Method** (how you did the project);

**Analysis** (breakdown everything in understandable manner for everyone, you can use tables, diagrams, codes, graphs, algorithms);

**Conclusions** (summary including results);

**References** (the work you read and cited as you write your project report, e.g. website, journal papers, conference papers, lab manuals).

***Dr. Ken Ehimwenma, Ph.D.***