DIGITAL PRESCRIPTION & DRUG INTERACTION CHECKER

A MINI-PROJECT REPORT

Submitted by

KRITHIKA RAGHAVAN	**240701278**
MERLYN SOSA SAJU	**240701309**

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2025

BONAFIDE CERTIFICATE

Certified that this project **"DIGITAL PRESCRIPTION & DRUG INTERACTION CHECKER"** is the bonafide work of **KRITHIKA RAGHAVAN (240701278)** and **MERLYN SOSA SAJU (240701309)** who carried out the project work under my supervision.

SIGNATURE

ANITHA M

ASSISTANT PROFESSOR SG

Dept. of Computer Science and Engineering,
Rajalakshmi Engineering College
Chennai

This mini project report is submitted for the viva voce examination to be held on

____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

Modern healthcare often involves patients taking multiple medications, sometimes prescribed by different specialists. This increases the risk of harmful drug-drug interactions, potentially leading to adverse effects or reduced treatment efficacy. While large healthcare systems may have integrated DDI-checking tools, accessing such safety features conveniently for local clinics or individual patients remains a challenge. To address this, our team developed a Digital Prescription and Drug Interaction Checker using Java and Oracle SQL. This database-driven system provides secure login portals for doctors and patients, enabling real-time detection of potential interactions by cross-referencing new prescriptions against a patient's existing medications and a comprehensive interaction database. Patients can also check interactions among drugs they intend to take. Key features include patient profile management, prescription history storage, instant warning generation, and alternative medication suggestions. This system enhances prescription safety, supports accurate record-keeping, and promotes informed treatment decisions for both doctors and patients.

ACKNOWLEDGEMENT

We express our thanks to our beloved and honorable chairman **Mr. S. MEGANATHAN** and the chairperson **Dr. M. THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honorable principal **Dr. S.N. MURUGESAN** for his able support and guidance.

No words of gratitude will suffice for the unquestioning support extended to us by our Head Of The Department **Dr. E.M. MALATHY** and our Deputy Head Of The Department **Dr. J. MANORANJINI** for being ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **ANITHA M,** for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering.

1. KRITHIKA RAGHAVAN

2. MERLYN SOSA SAJU

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

Prescribing medication ensures patient safety requires careful consideration of all drugs a patient is taking. Adverse Drug Events (ADEs) due to drug-drug interactions are a significant concern, leading to complications, hospitalizations, and increased healthcare costs. Our project, the Digital Prescription & Drug Interaction Checker, provides a software solution to help mitigate these risks. It serves as an informative tool for both prescribing doctors and patients managing their medications.

1.2 SCOPE OF THE WORK

This system provides a platform for doctors to manage patient prescriptions and receive real-time alerts about potential drug interactions. It also empowers patients with a tool to check for interactions among medications they are considering taking.

The scope includes:

- Implement secure authentication for doctors and patients while managing patient data, prescription history, and known drug interactions.
- Enable automatic interaction checks and suggest safe alternative medications based on therapeutic class and interaction data.

1.3 PROBLEM STATEMENT

Patients are often prescribed multiple medications by different healthcare providers, increasing the likelihood of harmful drug interactions going unnoticed until an adverse event occurs. Manual checking is time-consuming and prone to error. Lack of a centralized, easily accessible system for cross-referencing medications poses a significant risk to patient safety. Patients also lack accessible tools to self-check potential interactions between prescribed and over-the-counter medications.

1.4 AIM AND OBJECTIVES OF THE PROJECT

The main aim of this project is to enhance medication safety by providing a system that automatically detects and warns about potential drug interactions.

Develop an integrated Oracle SQL and Java-based system with secure login, real-time interaction checks, and alternative medicine suggestions

CHAPTER 2
SYSTEM SPECIFICATIONS

2.1 HARDWARE SPECIFICATIONS

Component	Specification
Processor	Intel i3 / AMD Ryzen 3 (or equivalent) Minimum
Memory Size	8GB RAM (Minimum)
Hard Disk Drive	500 GB (Minimum)

2.2 SOFTWARE SPECIFICATIONS

Component	Specification
Operating System	WINDOWS 10 / 11
Front – End	JavaFX
Back - End	Java (JDK 17+)
Database	Oracle 21c SQL
Development Tool	VS Code
Build Tool	Apache Maven

CHAPTER 3
MODULE DESCRIPTION

This application consists of distinct modules accessible after an initial role selection:

Doctor Portal Module:

• Authentication: Requires login (username/password). Includes a registration option requiring a specific code ('REC') for security. Authenticates against the Users table.

• Patient Management: Allows doctors to create new patient profiles (storing basic details).

• Prescription Interaction Check: Allows the doctor to select an existing patient ID and choose a new medicine from a dropdown list. The system checks this new medicine against the selected patient's active prescriptions stored in the database.

• Warning & Alternatives: If an interaction is found in the Drug_Interactions table, a warning message detailing the severity and nature of the interaction is displayed. The system also queries the database to suggest alternative drugs from the same therapeutic class that do not interact with the patient's existing medication.

Patient Portal Module:

• Authentication: Requires login (username/password). Includes a registration option allowing new patients to create an account, storing credentials in the Patients table.

• Medication List Management: Patients can select medicines from a dropdown list (populated from the Medicines table) and add them to a temporary list displayed on screen.

• Self-Check Interaction: Patients can click a button to check for interactions only between the medicines currently in their temporary list. The system compares all pairs of drugs in the list against the Drug_Interactions table.

• Warning Display: If any interactions are found within the list, warning messages are displayed.

CHAPTER 4
SAMPLE CODING

Sample 1: Database Connection - DatabaseConnector.java

```
package com.checker;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DatabaseConnector {
  // Replace with your Oracle DB details
  // Example URL for Oracle Express Edition
  private static final String ORACLE_URL = "jdbc:oracle:thin:@//localhost:1521/XEPDB1";
  private static final String USER = "drug_app_user"; // Application-specific user
  private static final String PASSWORD = "YourPassword"; // User's password
  public static Connection getConnection() throws SQLException {
    // Load the Oracle JDBC driver (Optional for modern JDBC)
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
    } catch (ClassNotFoundException e) {
      System.err.println("Oracle JDBC Driver not found!");
      e.printStackTrace();
      return null; // Or throw a runtime exception
    }
    // Establish and return the connection
    return DriverManager.getConnection(ORACLE_URL, USER, PASSWORD);
  }
}
```

Explanation: This code snippet shows the DatabaseConnector class responsible for establishing a connection to the Oracle database using JDBC (Java Database Connectivity). It defines the connection URL, username, and password, and provides a static method getConnection() that other parts of the application use to interact with the database. Proper error handling for database unavailability or incorrect credentials is included.

Sample 2: Interaction Check within a List - InteractionService.java

```
public List checkListInteractions(List medicineIds) {
  List interactions = new ArrayList<>();
  // SQL to find interactions between two specific medicine IDs
  String sql = "SELECT description, severity FROM Drug_Interactions " +
  "WHERE (medicine_id_1 = ? AND medicine_id_2 = ?) OR (medicine_id_1 = ? AND
medicine_id_2 = ?)";
  try (Connection conn = DatabaseConnector.getConnection()) {
    // Nested loops compare every unique pair of medicines in the input list
    for (int i = 0; i < medicineIds.size(); i++) {
      for (int j = i + 1; j < medicineIds.size(); j++) {
        int medId1 = medicineIds.get(i);
        int medId2 = medicineIds.get(j);
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
          // Set parameters for the SQL query
          ps.setInt(1, medId1);
          ps.setInt(2, medId2);
          ps.setInt(3, medId2);
          ps.setInt(4, medId1);
          ResultSet rs = ps.executeQuery();
          // If a result is found, an interaction exists
          if (rs.next()) {
            String severity = rs.getString("severity");
            String description = rs.getString("description");
            // Format a warning message
            interactions.add("WARNING (" + severity + "): Interaction found between
Medicine ID " + medId1 + " and " + medId2 + ". Details: " + description);
          }
        } // PreparedStatement closed automatically
      }
    }
  } catch (SQLException e) {
    e.printStackTrace(); // Log database errors
  }
  return interactions; // Return the list of found interaction warnings
}
```

Explanation: This method accepts a list of medicine IDs and checks all unique pairs for interactions using the **Drug_Interactions** table. If an interaction is found, it adds a formatted warning to a list and returns it.

Sample 3: Suggesting Alternatives - InteractionService.java

```
public List suggestAlternatives(int conflictingDrugId, int otherDrugId) {
  List alternatives = new ArrayList<>();
  String therapeuticClass = null;
  String getTherapeuticClassSQL = "SELECT therapeutic_class FROM Medicines WHERE
medicine_id = ?";
  try (Connection conn = DatabaseConnector.getConnection()) {
    // 1. Find the class of the drug causing the conflict
    try (PreparedStatement ps1 = conn.prepareStatement(getTherapeuticClassSQL)) {
      ps1.setInt(1, conflictingDrugId);
      ResultSet rs1 = ps1.executeQuery();
      if (rs1.next()) {
        therapeuticClass = rs1.getString("therapeutic_class");}}
        if (therapeuticClass != null && !therapeuticClass.isEmpty()) {
          // 2. Find other drugs in the same class that DON'T interact with the 'otherDrugId'
          String findAlternativesSQL = "SELECT m.generic_name " + "FROM Medicines m
" + "WHERE m.therapeutic_class = ? AND m.medicine_id != ? " +  // Same class, not itself
"AND NOT EXISTS ( " + // Crucial: Ensure no interaction exist  " SELECT 1 FROM
Drug_Interactions di " + " WHERE (di.medicine_id_1 = m.medicine_id AND
di.medicine_id_2 = ?) " + " OR (di.medicine_id_1 = ? AND di.medicine_id_2 =
m.medicine_id) " + ")";
          try (PreparedStatement ps2 = conn.prepareStatement(findAlternativesSQL)) {
            ps2.setString(1, therapeuticClass);
            ps2.setInt(2, conflictingDrugId);
            ps2.setInt(3, otherDrugId);
            ps2.setInt(4, otherDrugId);
            ResultSet rs2 = ps2.executeQuery();
            while (rs2.next()) {
              alternatives.add(rs2.getString("generic_name")); // Add safe alternatives
            }
          }
      }
    }
    catch (SQLException e) {
    /* ... error handling ... */
  }
   return alternatives;
}
```

Explanation: This method finds the therapeutic class of a conflicting drug and queries for other medicines in the same class that have no recorded interactions with the patient's current drugs, suggesting safer alternatives.

CHAPTER 5
SCREENSHOTS

Fig 5.1 Main Menu (Doctor/Patient Selection)

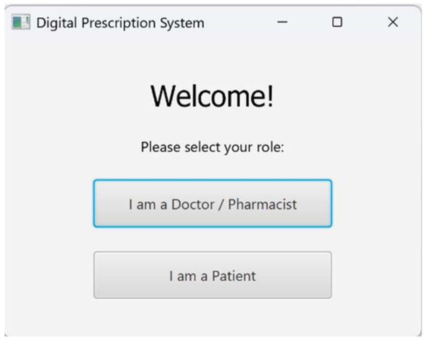
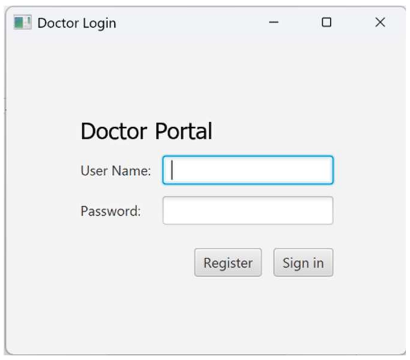
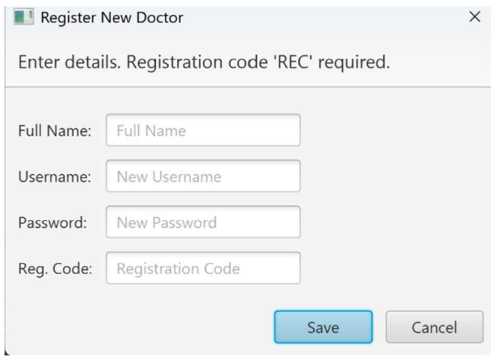


Fig 5.2 Doctor Login / Registration





Enter details. Registration code 'REC' required.

Fig 5.3 Doctor Portal - Interaction Checker

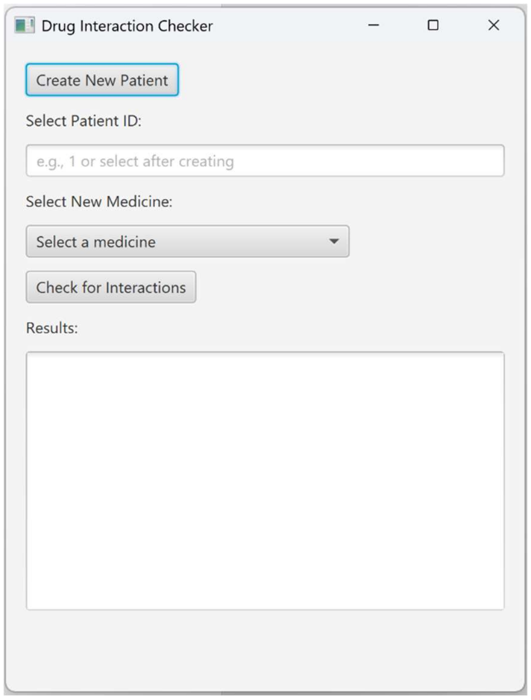
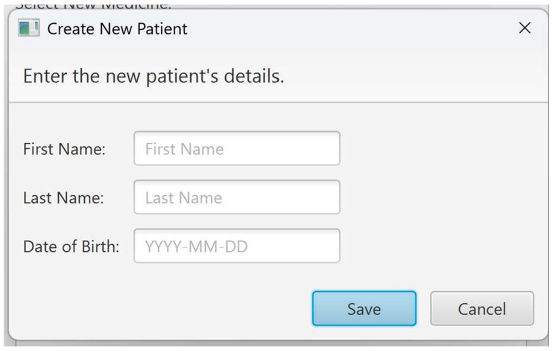


Fig 5.4 Doctor Portal - Create Patient Dialog



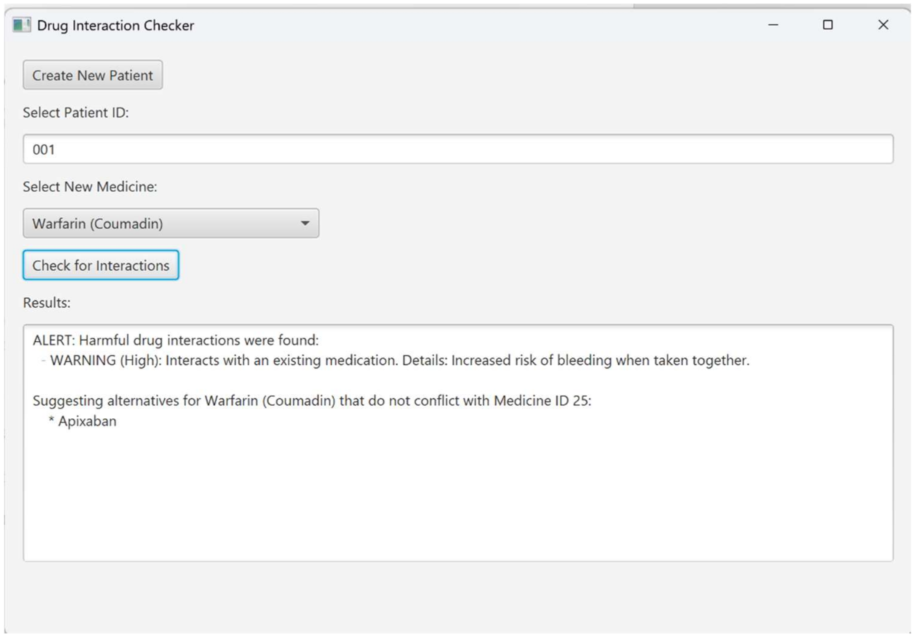
Enter the new patient's details.

First Name: First Name

Last Name: Last Name

Date of Birth: YYYY-MM-DD

Fig 5.5 Doctor Portal - Interaction Warning & Alternatives



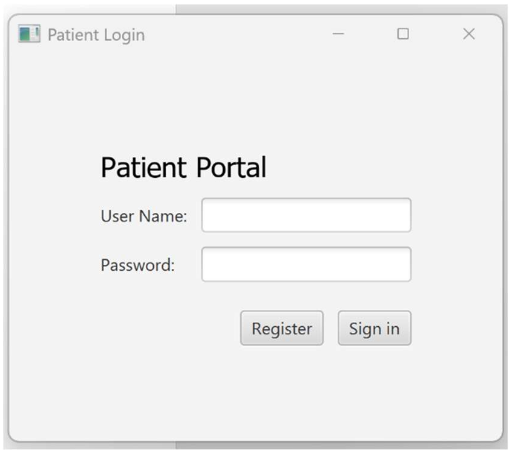
Select Patient ID: 001

Select New Medicine: Warfarin (Coumadin)

Results:

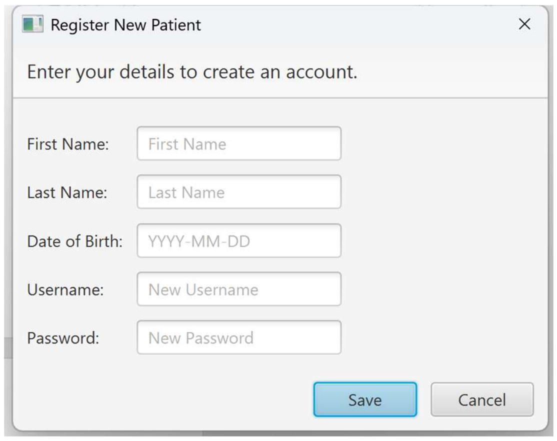
```
ALERT: Harmful drug interactions were found:
 - WARNING (High): Interacts with an existing medication. Details: Increased risk of bleeding when taken together.

Suggesting alternatives for Warfarin (Coumadin) that do not conflict with Medicine ID 25:
  * Apixaban
```

Fig 5.6 Patient Login / Registration



User Name:

Password:



Enter your details to create an account.

First Name: First Name

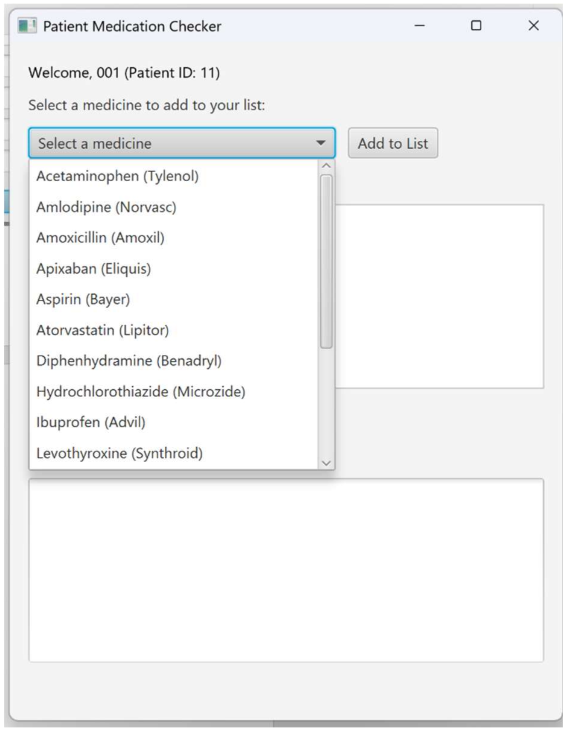
Last Name: Last Name

Date of Birth: YYYY-MM-DD

Username: New Username

Password: New Password

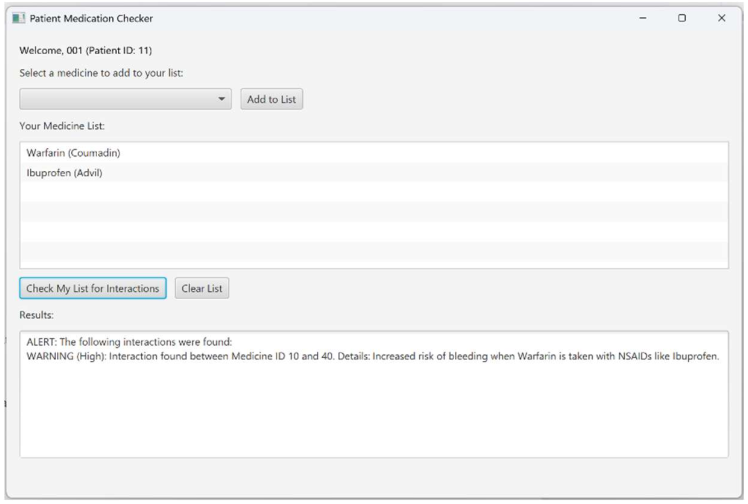
Fig 5.7 Patient Portal - Medication List & Check



Welcome, 001 (Patient ID: 11)

Select a medicine to add to your list:

Fig 5.8 Patient Portal - Interaction Warning



Welcome, 001 (Patient ID: 11)

Select a medicine to add to your list:

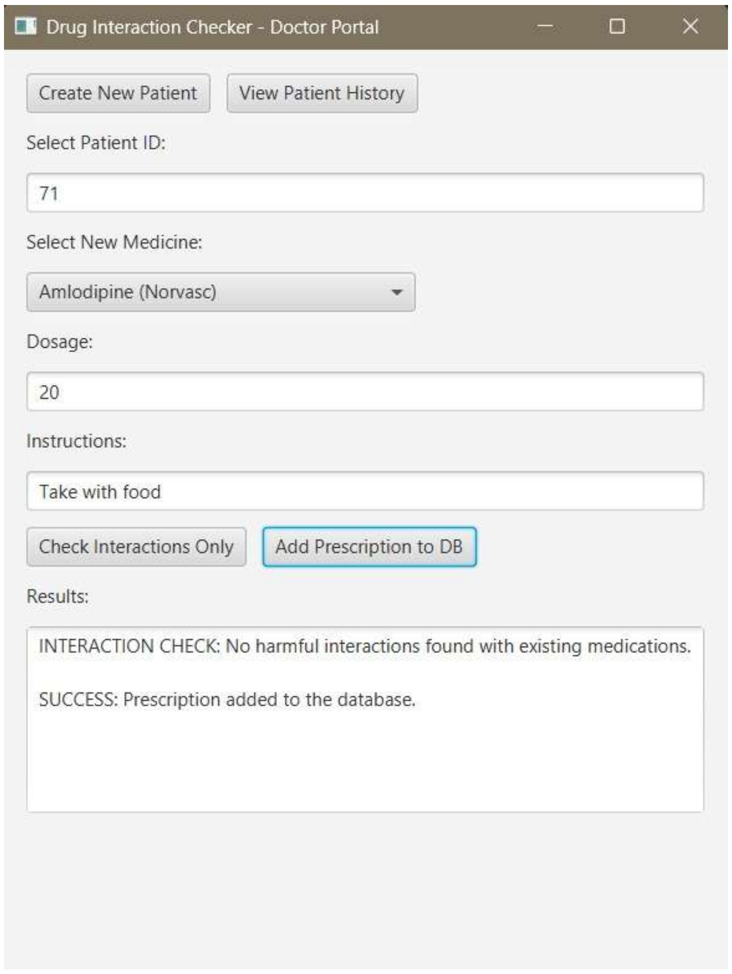
Your Medicine List:

Warfarin (Coumadin)
Ibuprofen (Advil)

Results:

```
ALERT: The following interactions were found:
WARNING (High): Interaction found between Medicine ID 10 and 40. Details: Increased risk of bleeding when Warfarin is taken with NSAIDs like Ibuprofen.
```

Fig 5.9 Doctor Portal – Add Prescription to DB



Select Patient ID: 71

Select New Medicine: Amlodipine (Norvasc)

Dosage: 20

Instructions: Take with food

Results:

```
INTERACTION CHECK: No harmful interactions found with existing medications.

SUCCESS: Prescription added to the database.
```

CHAPTER 6
CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

This project successfully developed a Digital Prescription & Drug Interaction Checker using JavaFX, Java, and Oracle SQL. The system provides separate, functional portals for doctors and patients. Doctors can manage patient records, receive real-time interaction warnings during prescription, and get suggestions for safer alternatives. Patients can register, log in, and perform self-checks for interactions among potential medications. By centralizing medication and interaction data in a database, the application effectively addresses the problem of harmful drug interactions, promoting safer medication practices. The use of clear UI elements like dropdowns enhances usability compared to requiring users to know medicine IDs.

FUTURE ENHANCEMENT

The current system provides core functionality, several enhancements could be implemented in the future:

• Expand Interaction Database: Populate the Drug_Interactions table with a comprehensive dataset from official medical sources.

• Dosage and Allergy Checks: Incorporate checks for incorrect dosages or known patient allergies.

• Severity Levels: Implement more nuanced handling based on interaction severity (e.g., critical vs. minor warnings).

• Patient History Integration: In the Doctor Portal, make the alternative suggestion logic dynamically identify the specific conflicting drug from the patient's history instead of using a hardcoded example.

• Password Hashing: Implement proper password hashing (e.g., using bcrypt) for secure storage instead of plain text.

• EHR Integration: Explore possibilities for integrating with existing Electronic Health Record systems.

• Mobile Application: Develop a mobile version for increased accessibility, especially for patients.

REFERENCES

1. https://docs.oracle.com/en/database/oracle/oracle-database/21/
2. https://docs.oracle.com/en/java/javase/17/
3. https://openjfx.io/openjfx-docs/
4. https://code.visualstudio.com/docs
5. https://www.w3schools.com/sql/
6. https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
7. https://www.tutorialspoint.com/javafx/
8. https://maven.apache.org/
9. https://www.jetbrains.com/guide/java/
10. https://apexapps.oracle.com/pls/apex/f?p=44785:1