Software Requirement and Specifications(SRS)

Purpose:

The Employee Management System (EMS) is designed to efficiently manage employee records, including adding, updating, removing, and viewing employee details. It provides an intuitive user interface with secure login functionality.

Scope:

The system will allow HR personnel or administrators to manage employees through a graphical user interface (GUI) with authentication features. It will store and retrieve employee information from a database.

Overall Description

Operating Environment:

- Java Runtime Environment (JRE)
- Windows
- MySQL

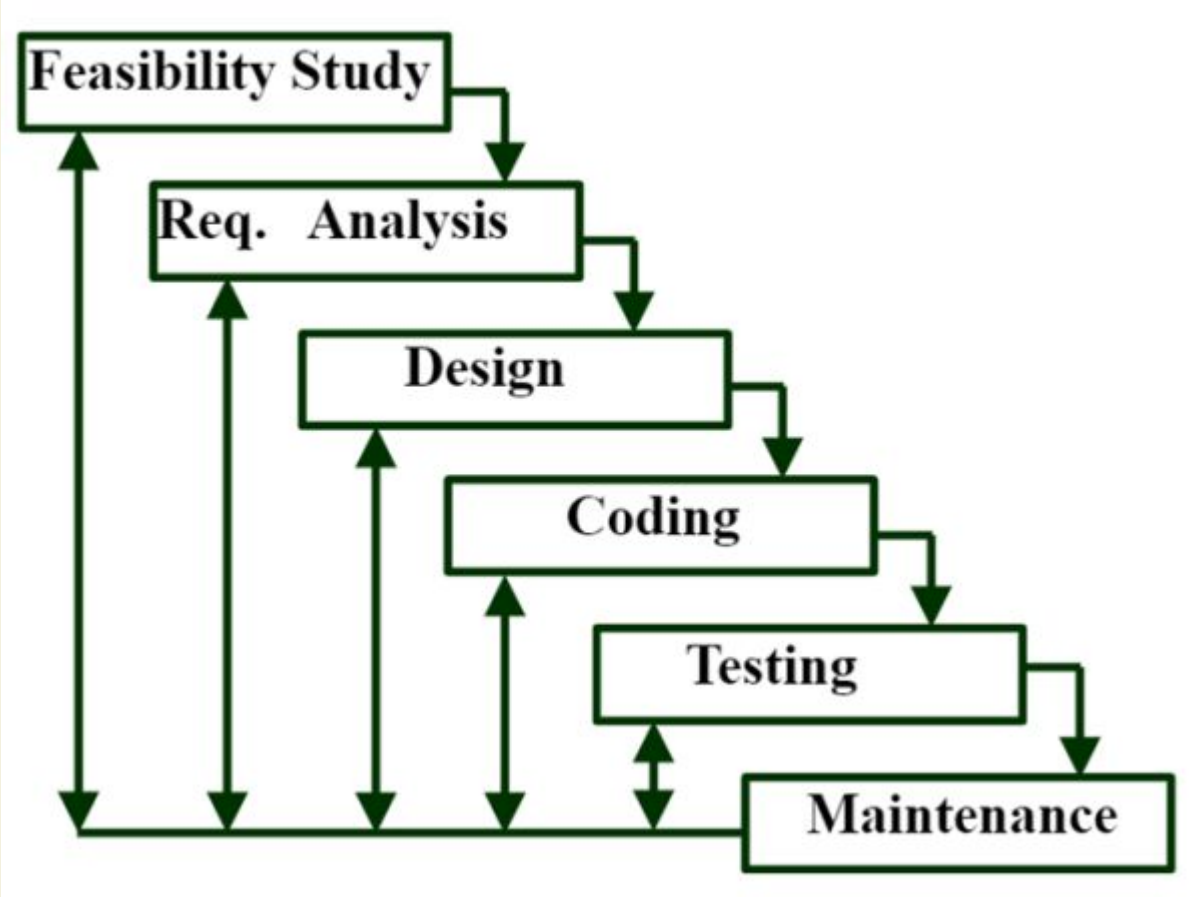
Design and Implementation Constraints:

- System must be developed using Java
- Must use a database for persistent data storage
- User authentication should be secure

Assumptions and Dependencies:

- Users will have valid credentials to access the system
- Database server is properly configured

Iterative Waterfall Model



Functional Requirements

Welcome Page:

- Displays a greeting message and navigation options.

Login Page:

- Allows users to log in using valid credentials.
- Verifies credentials from the database.

Add Employee:

- Provides a form to enter employee details (Name, ID, Department, Salary, etc.).
- Stores employee data in the database.

Update Employee:

- Allows modification of existing employee details.
- Updates the database records accordingly.

Functional Requirements

Remove Employee:

- Allows removal of an employee record
- Deletes the record from the database..

View Employee:

- Allows searching/filtering of records.
- Displays a list of employees with details.

Non Functional Requirements

Performance Requirements:

- System should load employee data within 2 seconds.
- Login authentication should not exceed 3 seconds.

Security Requirements:

- Only authorized users can access modification features.

Usability Requirements:

- User-friendly interface with clear navigation.

Availability and Reliability:

- System should be available 99% of the time.
- Data integrity must be maintained in case of a crash.

Tech Stack Used:

Programming Language:

- Java

Frontend (User Interface):

- Java Swing is used for building the graphical user interface (GUI) for different pages.
- Swing components like JFrame, JButton, JLabel, JTextField, JOptionPane are used for UI elements.

Backend (Database & Business Logic):

- MySQL → The database used for storing employee details
- JDBC (Java Database Connectivity) → Used to interact with MySQL

Tech Stack Used:

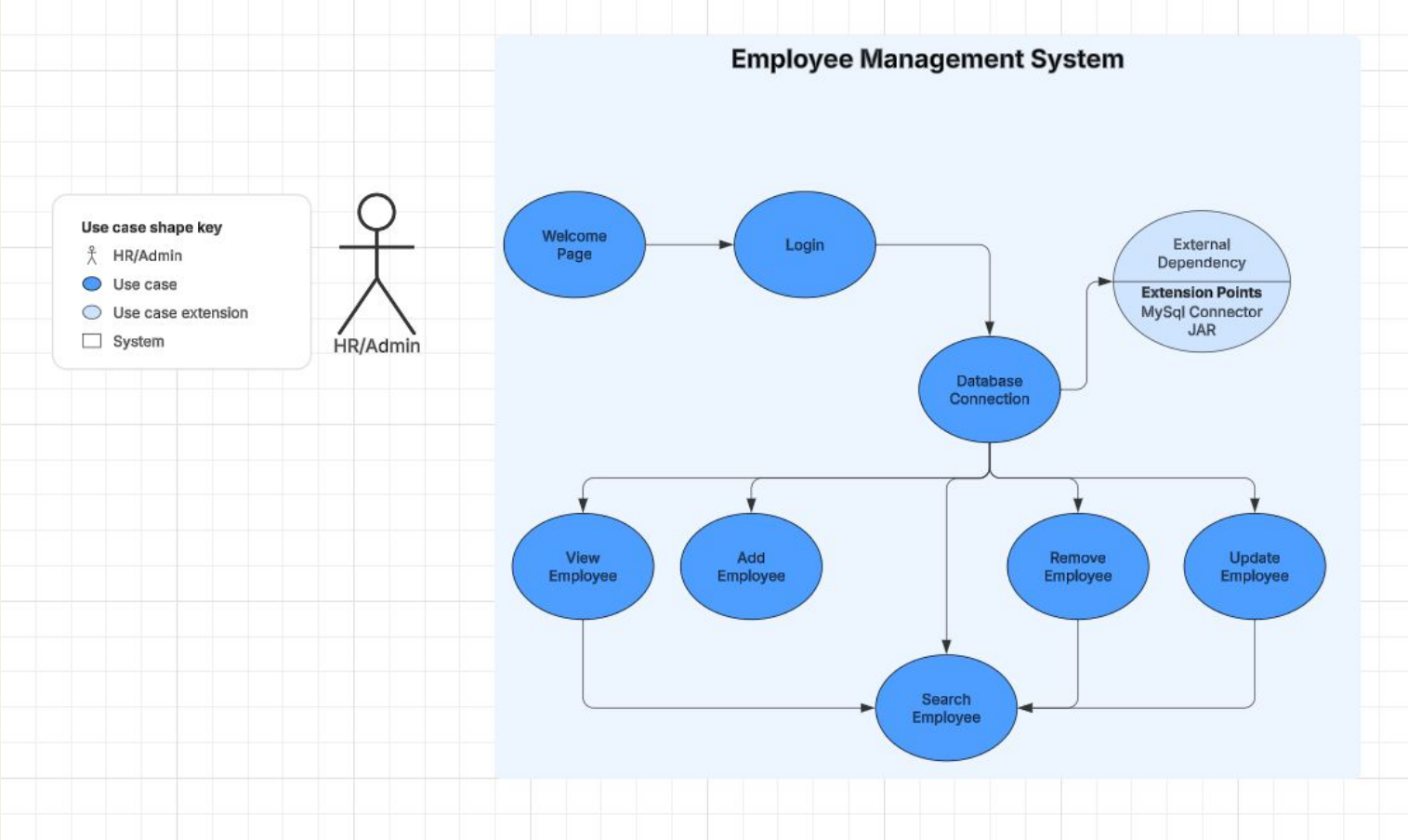
Testing:

- JUnit 5 → Used for writing unit and integration tests.
- Mockito → Used for mocking database interactions (JDBC Connection, Statement, ResultSet).
- Hamcrest → Used for better assertion handling in test cases.

Build & Execution:

- JDK 14 → System runs on Java 14.
- Manual JAR Management → JUnit 5, Mockito, Byte Buddy, Hamcrest,mysql-connector

Use – Case Diagram

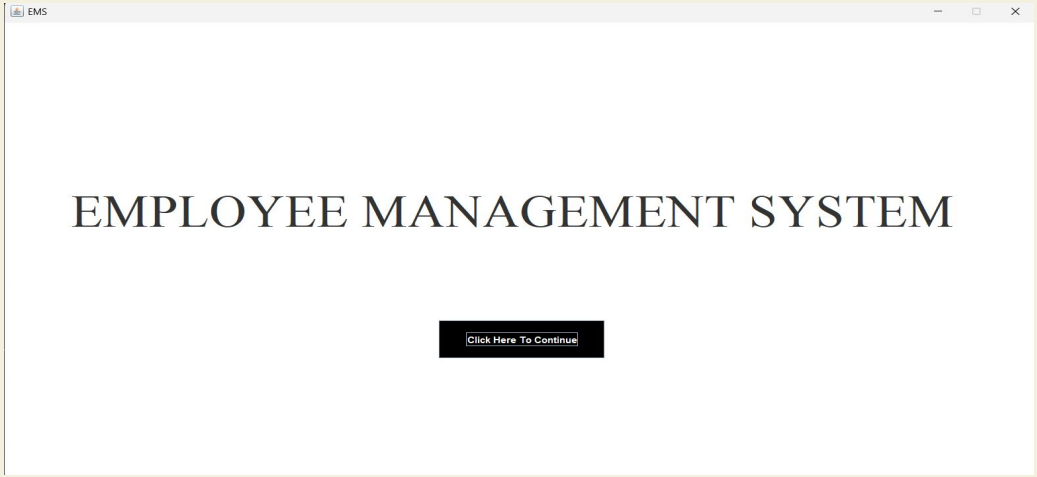


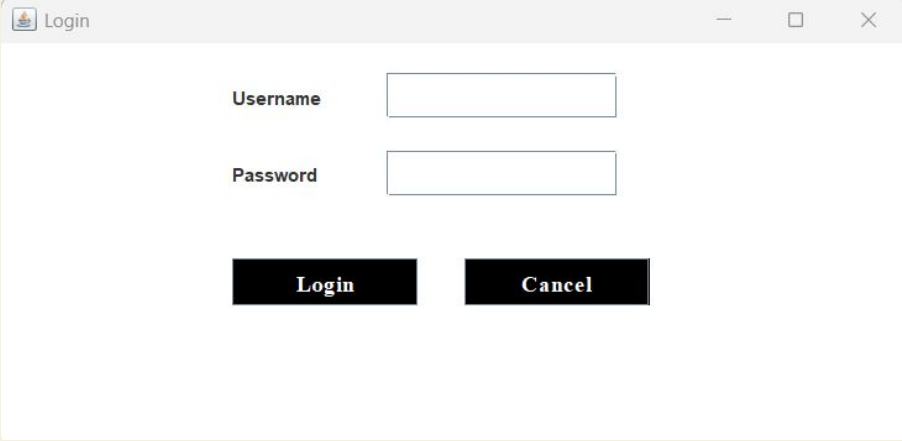
CASE STUDY:

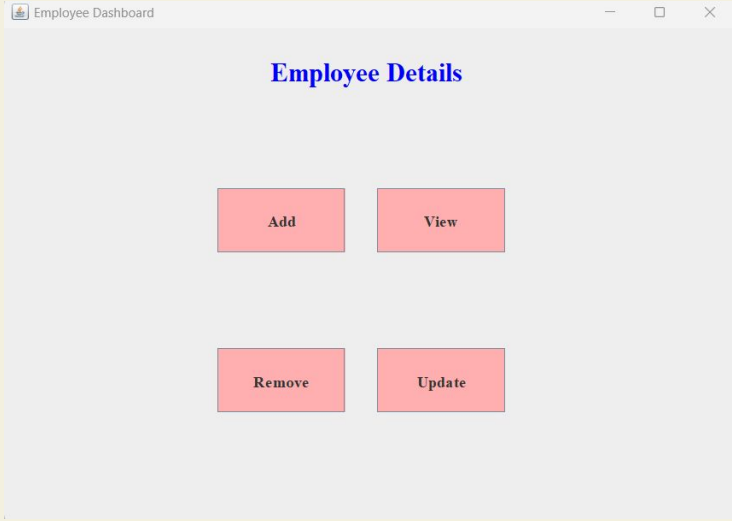
Employee Management System

Implementation and Testing

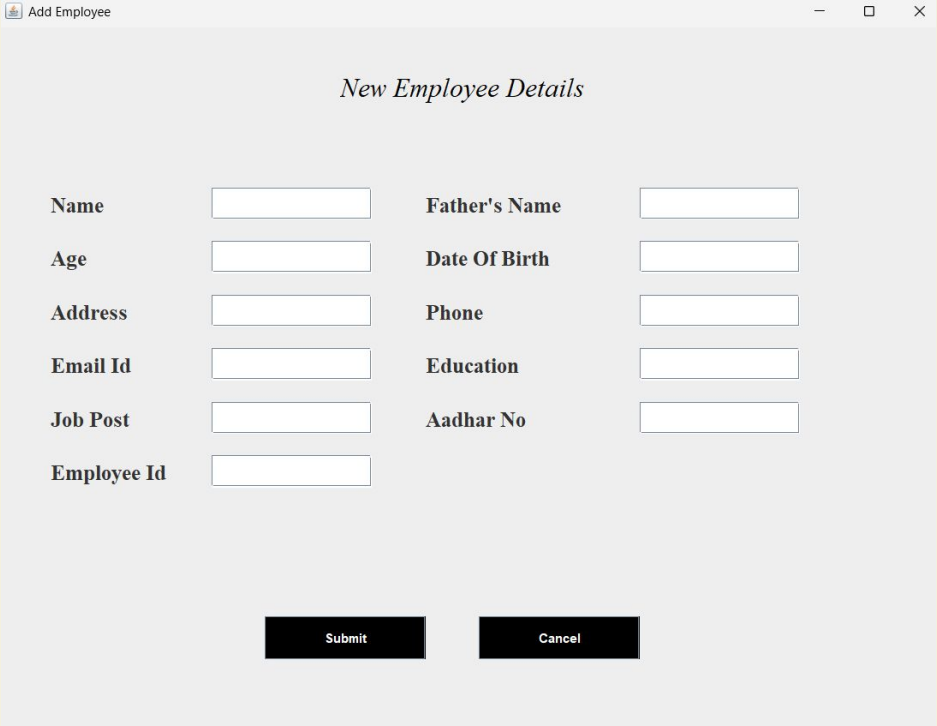
Implementation



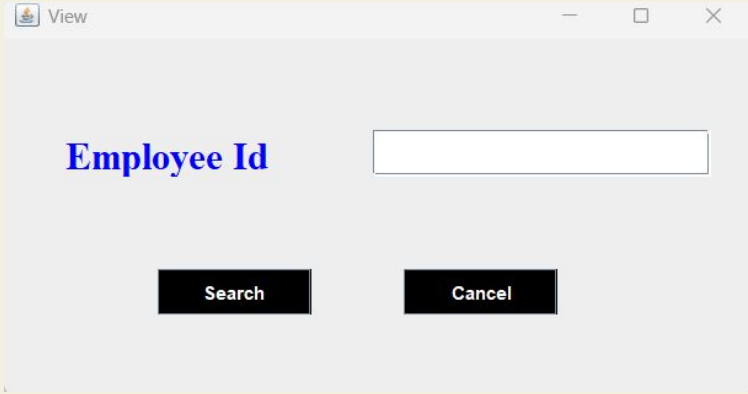


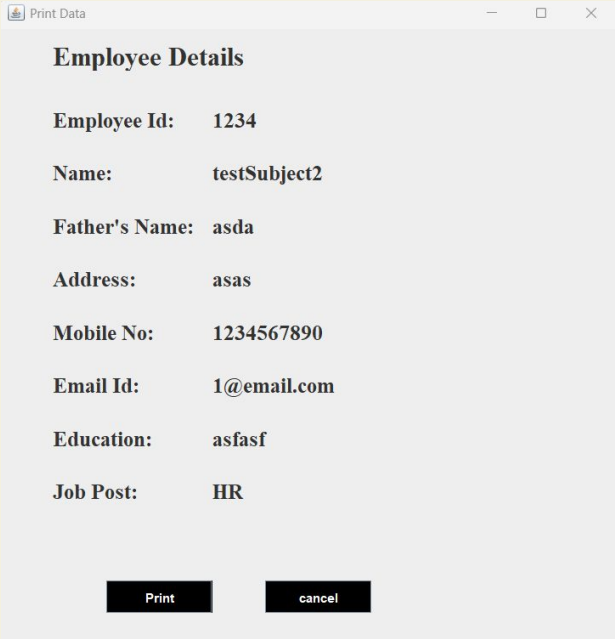


Implementation

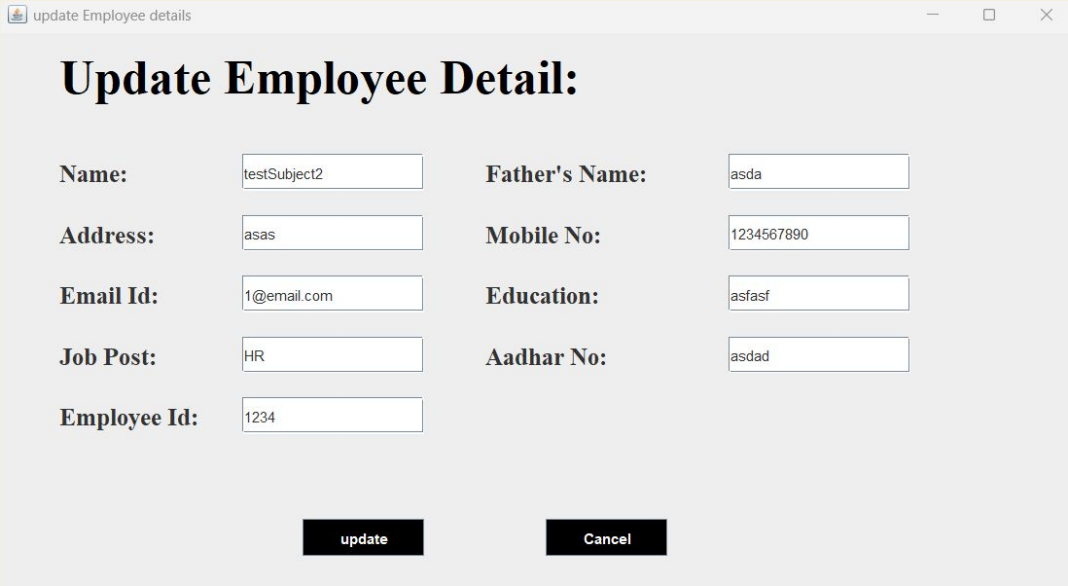


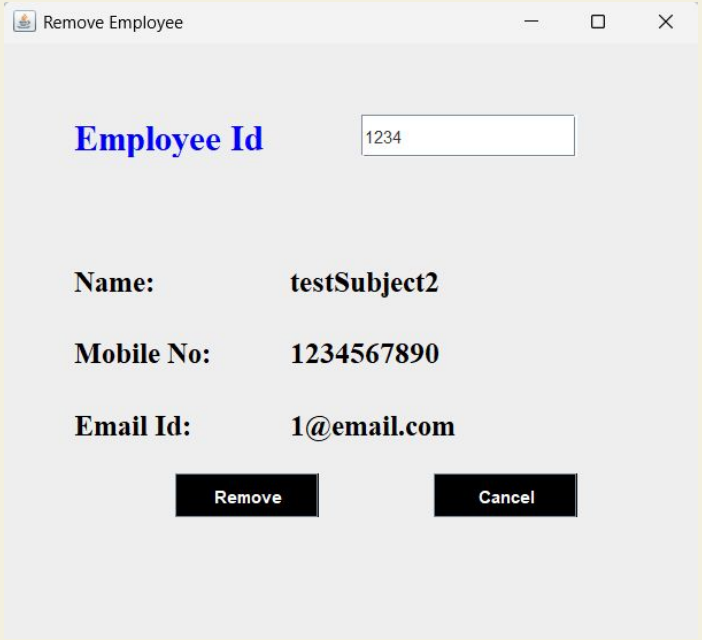
Implementation





Implementation





Unit Testing

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class TestaddEmployee {
    private add_employee addEmp;

    @BeforeEach
    void setUp() {
        addEmp = new add_employee();
        System.out.println(x:"✅ Setup completed.");
    }

    @Test
    void testValidPhoneNumber1() {
        addEmp.t6.setText(t:"9876543210");
        assertTrue(addEmp.t6.getText().matches(regex:"\\d{10}"), "Phone number validation failed");
        System.out.println(x:"✅ Valid phone number test passed.");
    }

    @Test
    void testInvalidPhoneNumber() {
        addEmp.t6.setText(t:"98765");
        assertFalse(addEmp.t6.getText().matches(regex:"\\d{10}"), "Invalid phone should not pass validation");
        System.out.println(x:"✅ Invalid phone number test passed.");
    }
```

PROBLEMS 14 | OUTPUT | DEBUG CONSOLE | TEST RESULTS

```
✅ Setup completed.
✅ Valid email number test passed.
✅ Setup completed.
✅ Valid DOB number test passed.
✅ Setup completed.
✅ Invalid phone number test passed.
✅ Setup completed.
✅ Valid phone number test passed.
✅ Setup completed.
✅ Invalid email test passed.
✅ Setup completed.
✅ Invalid DOB test passed.
```

Test Runner for Java

- testInvalidDOB()
- testInvalidEmail()
- testInvalidPhoneNumber()
- testValidDOB()
- testValidEmail()
- testValidPhoneNumber1()

Unit Testing

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.awt.event.ActionEvent;

public class TestLoginPage {
    private login_page loginPage;

    @BeforeEach
    void setUp() {
        loginPage = new login_page();
        System.out.println(x:"✅ Setup completed for LoginPageTest.");
    }

    @Test
    void testValidLogin() {
        loginPage.t1.setText(t:"admin");
        loginPage.t2.setText(t:"admin");

        ActionEvent e = new ActionEvent(loginPage.b1, ActionEvent.ACTION_PERFORMED, command:null);
        loginPage.actionPerformed(e);

        boolean frameClosed = !loginPage.frame.isVisible();
        System.out.println("✅ Test Valid Login: Frame closed? " + frameClosed);
        assertTrue(frameClosed, "Frame should be closed after successful login");
    }

    @Test
    void testInvalidLogin() {
        loginPage.t1.setText(t:"wrongUser");
        loginPage.t2.setText(t:"wrongPass");
```

```
PROBLEMS 14   OUTPUT   DEBUG CONSOLE   TEST RESULTS
✅ Setup completed for LoginPageTest.
✅ Test Valid Login: Frame closed? true
✅ Setup completed for LoginPageTest.
✅ Test Invalid Login: Frame open? false
```

Test Runner for Java

- testInvalidLogin()
- testValidLogin()

Unit Testing

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.awt.event.ActionEvent;

public class TestUpdateEmployee {
    private update_employee updateEmp;

    @BeforeEach
    void setUp() {
        updateEmp = new update_employee(idaa:"123");
        System.out.println(x:"✅ Setup completed for UpdateEmployeeTest.");
    }

    @Test
    void testShowEmployeeData() {
        boolean dataLoaded = !updateEmp.t1.getText().isEmpty();
        System.out.println("✅ Test Show Employee Data: Data loaded? " + dataLoaded);
        assertTrue(dataLoaded, "Employee data should be loaded into fields");
    }

    @Test
    void testUpdateEmployee() {
        updateEmp.t1.setText(t:"NameUpdated");
        ActionEvent e = new ActionEvent(updateEmp.b, ActionEvent.ACTION_PERFORMED, command:null);
        updateEmp.actionPerformed(e);

        boolean frameClosed = !updateEmp.f.isVisible();
        System.out.println("✅ Test Update Employee: Frame closed? " + frameClosed);
        assertTrue(frameClosed, "Frame should close after updating an employee");
    }
}
```

```
PROBLEMS 14   OUTPUT   DEBUG CONSOLE   TEST RESULTS
✅ Setup completed for UpdateEmployeeTest.
✅ Test Show Employee Data: Data loaded? true
✅ Setup completed for UpdateEmployeeTest.
✅ Test Update Employee: Frame closed? true
```

```
Test Runner for Java
✓ testShowEmployeeData()
✓ testUpdateEmployee()
```

Unit Testing

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.awt.event.ActionEvent;

public class TestRemoveEmployee {
    private remove_employee removeEmp;

    @BeforeEach
    void setUp() {
        removeEmp = new remove_employee();
        System.out.println(x:"✅ Setup completed for RemoveEmployeeTest.");
    }

    @Test
    void testSearchEmployee() {
        removeEmp.t.setText(t:"E123");
        ActionEvent e = new ActionEvent(removeEmp.b, ActionEvent.ACTION_PERFORMED, command:null);
        removeEmp.actionPerformed(e);

        boolean isVisible = removeEmp.l2.isVisible();
        System.out.println("✅ Test Search Employee: Employee details visible? " + isVisible);
        assertFalse(isVisible, "Details should be visible after searching");
    }

    @Test
    void testRemoveEmployee() {
        removeEmp.t.setText(t:"E123");
        ActionEvent e = new ActionEvent(removeEmp.b1, ActionEvent.ACTION_PERFORMED, command:null);
        removeEmp.actionPerformed(e);

        boolean isHidden = !removeEmp.l2.isVisible();
        System.out.println("✅ Test Remove Employee: Employee details hidden? " + isHidden);
        assertTrue(isHidden, "Details should be hidden after removing employee");
    }
```

```
✅ Setup completed for RemoveEmployeeTest.
✅ Test Remove Employee: Employee details hidden? true
✅ Setup completed for RemoveEmployeeTest.
✅ Test Search Employee: Employee details visible? false
```

Test Runner for Java

- testRemoveEmployee()
- testSearchEmployee()

Integration Testing

```java
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.sql.*;

class IntegrationTesting {
    private conn mockConn;
    private Connection mockConnection;
    private Statement mockStatement;
    private ResultSet mockResultSet;

    @BeforeEach
    void setUp() throws Exception {
        mockConn = mock(conn.class);
        mockConnection = mock(Connection.class);
        mockStatement = mock(Statement.class);
        mockResultSet = mock(ResultSet.class);

        when(mockConn.getConnection()).thenReturn(mockConnection);
        when(mockConnection.createStatement()).thenReturn(mockStatement);
        when(mockStatement.executeQuery(anyString())).thenReturn(mockResultSet);
    }
```

```
PROBLEMS 14   OUTPUT   DEBUG CONSOLE   TEST RESULTS   TERMINAL   PO
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only
Test Passed: Login successful!
Test Passed: Login failed as expected!
Test Passed: Employee added successfully!
Test Passed: Employee updated successfully!
Test Passed: Employee removed successfully!
```

```java
// Test 2: Remove Employee
@Test
void testEmployeeRemovedSuccessfully() {
    try {
        String sql = "DELETE FROM employee WHERE emp_id = '1002'";
        mockStatement.executeUpdate(sql);

        verify(mockStatement, times(1)).executeUpdate(sql);
        System.out.println(x:"Test Passed: Employee removed successfully!");
    } catch (Exception e) {
        fail("Exception occurred: " + e.getMessage());
    }
}

// Test 3: Update Employee
@Test
void testEmployeeUpdatedSuccessfully() {
    try {
        String sql = "UPDATE employee SET name='John Updated' WHERE emp_id = '1002'";
        mockStatement.executeUpdate(sql);

        verify(mockStatement, times(1)).executeUpdate(sql);
        System.out.println(x:"Test Passed: Employee updated successfully!");
    } catch (Exception e) {
        fail("Exception occurred: " + e.getMessage());
    }
}
```

```
%TSTTREE7,testEmployeeRemovedSuccessfully(IntegrationTesting),false,1,false,2,testEmployeeRemovedSuccessfully(),,[engine:junit-jupiter]/[class:IntegrationTesting]/[method:testEmployeeRemovedSuccessfully()]
%TESTS  3,testLoginSuccess(IntegrationTesting)

%TESTE  3,testLoginSuccess(IntegrationTesting)

%TESTS  4,testLoginFailure(IntegrationTesting)

%TESTE  4,testLoginFailure(IntegrationTesting)

%TESTS  5,testEmployeeAddedSuccessfully(IntegrationTesting)

%TESTE  5,testEmployeeAddedSuccessfully(IntegrationTesting)

%TESTS  6,testEmployeeUpdatedSuccessfully(IntegrationTesting)

%TESTE  6,testEmployeeUpdatedSuccessfully(IntegrationTesting)

%TESTS  7,testEmployeeRemovedSuccessfully(IntegrationTesting)

%TESTE  7,testEmployeeRemovedSuccessfully(IntegrationTesting)
```

Test Runner for Java

- testEmployeeAddedSuccessfully()
- testEmployeeRemovedSuccessfully()
- testEmployeeUpdatedSuccessfully()
- testLoginFailure()
- testLoginSuccess()

Thank You !!