Paramount Khalkhali Sharifi (219125046)

List of user stories that are fully-implemented:
User story 1: hire physicians Harrish
User story 2: hire nurses Harrish
User story 3: resign physicians Harrish
User story 4: resign nurses Harrish
User story 5: store and retrieve vital signs of the patient Amira
User story 6: find a patient and display their information Amira, Mehregan
User story 7: Family doctor to get the patient's information and be searched Parmoun
User story 8: Setting the patient's consent form status Parmoun
User story 13: email family doctor with patient information after discharge Mehregan
User story 15: registration, login and validation module: Gaurav

Harrish - Testing user stories 5,6
Amira - Testing user stories 1, 3
Parmoun - Testing user stories 2,4
Mehregan - Testing user stories 7,15
Gaurav - Testing user stories 8, 13

PART 1:

User story 2:

- **Complete Junit test case for backend:**

```
package Hospital;
import static org.junit.jupiter.api.Assertions.*;
import java.sql.SQLException;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class User2test {

    private DatabaseOps dbOps;

    @BeforeEach
    public void setUp() {
        dbOps = new DatabaseOps();
    }

    @AfterEach
    public void tearDown() {
        // Cleaning DB
    }
```

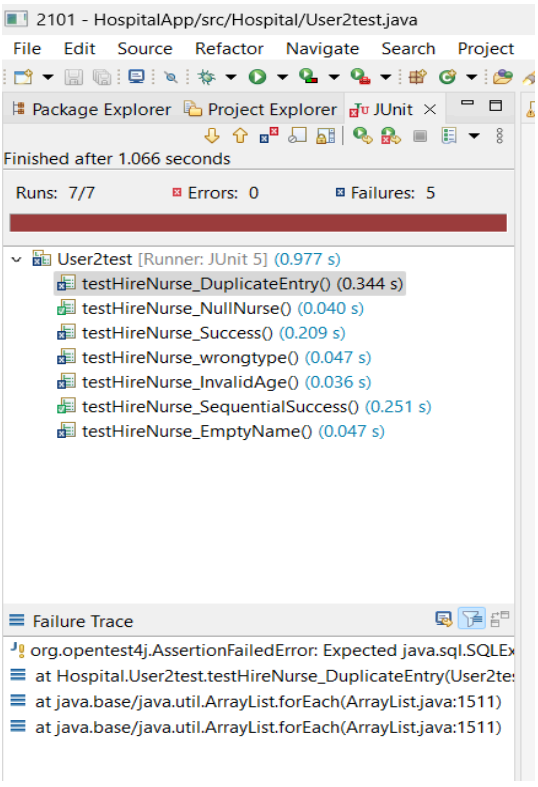
```
    /**
     * This test must fail
     */
    @Test
    public void testHireNurse_DuplicateEntry() {
        Nurse newNurse = new Nurse("Anna", "Small", 28, "Female", "123 Helendale Ave");
        dbOps.addNurse(newNurse);
        Exception exception = assertThrows(SQLException.class, () ->
{dbOps.addNurse(newNurse);});
        assertNotNull(exception);
    }

    /**
     * This test must fail
     */
    @Test
    public void testHireNurse_InvalidAge() {
        Nurse invalidNurse = new Nurse("Invalid", "User", -10, "Unknown", "44 New st");
        Exception exception = assertThrows(SQLException.class, () -> {
            dbOps.addNurse(invalidNurse);
        });
        assertNotNull(exception);
    }
/**
 * This test must Fail
 */
    @Test
    public void testHireNurse_EmptyName() {
        Nurse incompleteNurse = new Nurse("", "Nil", 25, "Female", "123 Valley St");
        Exception exception = assertThrows(SQLException.class, () -> {
            dbOps.addNurse(incompleteNurse);
        });
        assertNotNull(exception);
    }

    /**
     * This test must pass
     */
    @Test
    public void testHireNurse_NullNurse() {
        Exception exception = assertThrows(NullPointerException.class, () -> {
            dbOps.addNurse(null);
        });
        assertNotNull(exception);
    }
    /**
     * This test must fail
     */
    @Test
```

```
    public void testHireNurse_wrongtype() {
        Nurse longNameNurse = new Nurse("Thenameisverylongandweexpecttofail",
"Franklin", 25, "Female", "123 twoway St");
        Exception exception = assertThrows(SQLException.class, () -> {
            dbOps.addNurse(longNameNurse);
        });
        assertNotNull(exception);
    }
    /**
     * This test must pass
     */
    @Test
    public void testHireNurse_SequentialSuccess() {
        Nurse nurse1 = new Nurse("Alice", "Smith", 28, "Female", "33 Amazing St");
        Nurse nurse2 = new Nurse("Bob", "Jones", 32, "Male", "7 Maplefield St");
        dbOps.addNurse(nurse1);
        dbOps.addNurse(nurse2);
        assertTrue(dbOps.getAllNurses().contains(nurse1));
        assertTrue(dbOps.getAllNurses().contains(nurse2));
}
    /**
     * This test must pass
     */
        @Test
        public void testHireNurse_Success() {
            Nurse newNurse = new Nurse("Eli", "Kevin", 30, "Male", "12 Newdale St");
            dbOps.addNurse(newNurse);
            assertTrue(dbOps.getAllNurses().contains(newNurse));
        }
}
```

- **We see that according to the comments, the specified tests are failing and the true implementations are passing the test cases.**

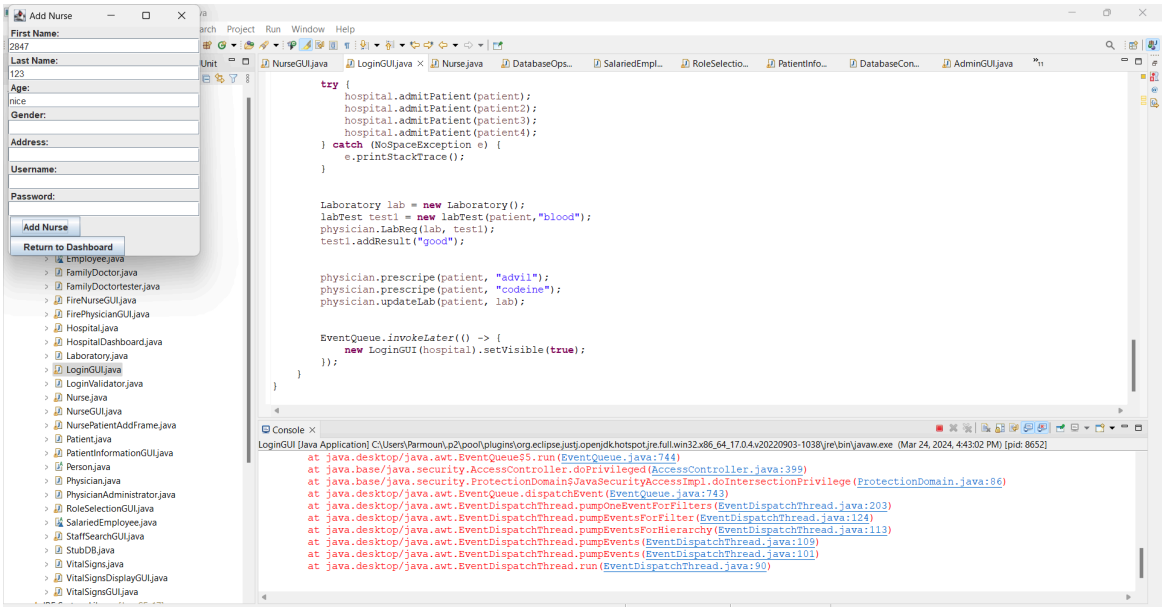


- **General steps for testing DB and GUI:**

1. Choose to log in as admin by pressing “Admin login”
2. Enter username “admin” and password “pass”
3. Then click on “Hire Nurse”
4. Enter the details for the nurse (The main testing step of this user story)
5. Click the “Add Nurse” button to add a nurse to the hospital system and database. Now, Check to see if the nurse was created by checking the “Resign Nurse” part.
6. Return to the dashboard or enter the credentials of the next nurse you want to add.
7. A Nurse should be added to the hospital system and database. We can now log in as the Nurse you created using the credentials from when creating a nurse.

Passed:

- String expected, when added a number, didn’t add and gave error



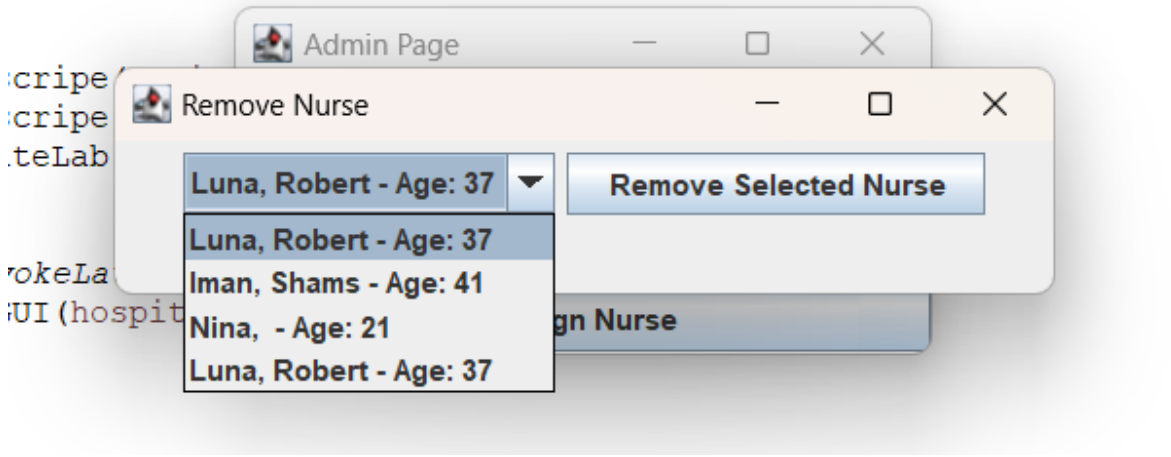
- Multiple Nurses can be Added Sequentially: showcases system's ability to handle multiple entries without failure.

```
postgres=# SELECT * FROM nurses;
 id  | firstname | lastname | age | gender |     address     | username | password
-----+-----------+----------+-----+--------+-----------------+----------+----------
 105 | Luna      | Robert   |  37 | Female | 39 Harrison Ave | luna     | 1234
 106 | Iman      | Shams    |  41 | Male   | 22 Honeyroad    | iman     | 1234
 107 | Nina      |          |  21 |        |                 |          |
 108 | Luna      | Robert   |  37 | Female | 39 Harrison Ave | luna     | 1234
(4 rows)


postgres=#
```

```java
    /**
     * This test must pass
     */
    @Test
    public void testHireNurse_SequentialSuccess() {
        Nurse nurse1 = new Nurse("Alice", "Smith", 28, "Female", "33 Amazing St");
        Nurse nurse2 = new Nurse("Bob", "Jones", 32, "Male", "7 Maplefield St");
        dbOps.addNurse(nurse1);
        dbOps.addNurse(nurse2);
        assertTrue(dbOps.getAllNurses().contains(nurse1));
        assertTrue(dbOps.getAllNurses().contains(nurse2));
}
```

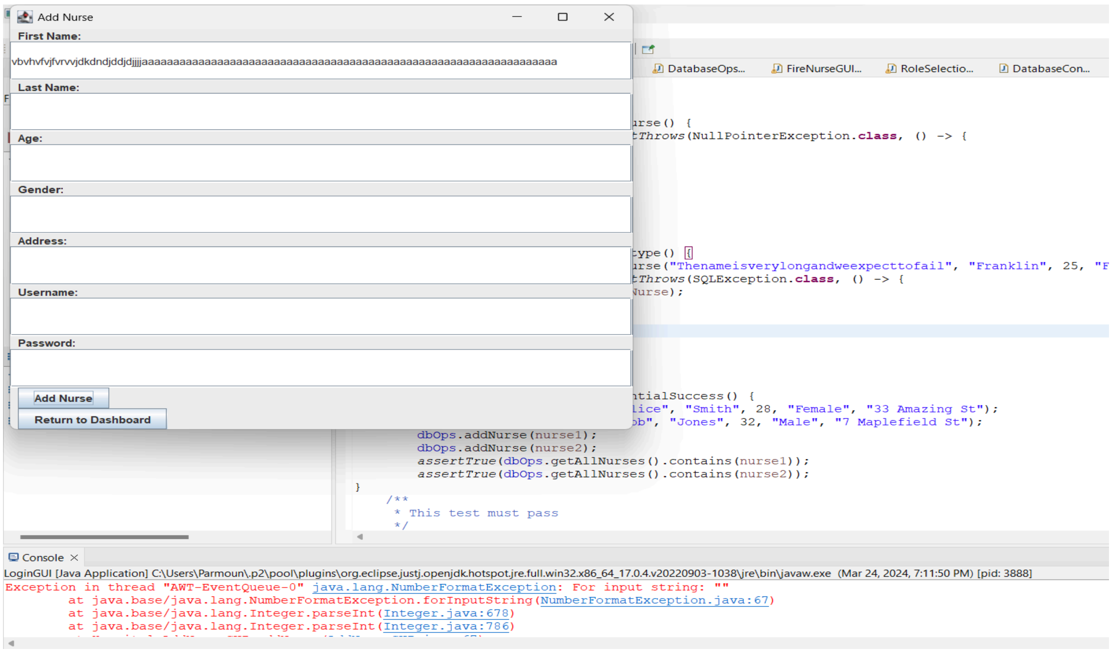
- The different classes in the system are correctly connected. The hire Nurse, passes a list to the method resign Nurse, and the list is fully added and shown for resign nurse.



- The nurses that were added successfully, are able to login to the system using their specific password and username.

- The system doesn't add the nurse with a very long invalid name, which is the expected behavior.

```java
/**
 * This test must fail
 */
@Test
public void testHireNurse_wrongtype() {
    Nurse longNameNurse = new Nurse("Thenameisverylongandweexpecttofail", "Franklin", 25, "Female", "123 twoway St");
    Exception exception = assertThrows(SQLException.class, () -> {
        dbOps.addNurse(longNameNurse);
    });
    assertNotNull(exception);
}
```



User story 2 Bug Report:

Bug 1:

Problem Report Form:

Problem Report Number: 1
Reported by: Paramount Khalkhali Sharifi
Date Reported: 24/03/2024
Program Name: Hire Nurse module
Release Number: 1
Version Identifier: Build 1
Configuration: Windows 10, Java 17, Hospital Management System
Report Type: Coding / Validation Issue
Can Reproduce: Yes
Severity: High
Priority: High
Problem Summary: Incomplete nurse details accepted by the system.
Key Words: Nurse registration, validation, incomplete fields.
Problem Description and How to Reproduce It:

- Navigate to the "Hire Nurse" section.
- Enter the following details: Name - "Nina", Age - "21". Leave other fields blank.
- Submit the form.

- The system adds the nurse without requiring all fields to be filled out.
- Expected behavior: The system should validate that all fields are completed and not allow submission until they are.

Suggested Fix: Implementing field validation to check for empty inputs before submitting the form.
Status: Open
Resolution: Pending for Iteration 3
Resolved by: Will be resolved by Harrish Elango by the Iteration 3

```
/**
 * This test must Fail
 */
    @Test
    public void testHireNurse_EmptyName() {
        Nurse incompleteNurse = new Nurse("", "Nil", 25, "Female", "123 Valley St");
        Exception exception = assertThrows(SQLException.class, () -> {
            dbOps.addNurse(incompleteNurse);
        });
        assertNotNull(exception);
    }
```

Bug 2:

Problem Report Form:

Problem Report Number: 2
Reported by: Paramount Khalkhali Sharifi
Date Reported: 24/03/2024
Program Name: Nurse Hiring Module
Release Number: 1
Version Identifier: Build 1
Configuration(s): Windows 10, Java 17, Hospital Management System
Report Type: Design Issue
Can Reproduce: Yes
Severity: Medium
Priority: Medium
Problem Summary: Duplicate nurse entries are not prevented by the system.
Key Words: Nurse registration, duplicate entry, data integrity.
Problem Description and How to Reproduce It:

- Navigate to the "Hire Nurse" section.
- Enter details for a new nurse and submit the form.
- Repeat the process with the same nurse details.
- The system accepts the duplicate nurse entry.
- Expected behavior: The system should check for existing entries with the same details and prevent or flag duplicates.

Suggested Fix: Implement a uniqueness check for the database for critical fields such as Name and ID before adding a new entry.
Status: Open
Resolution: Pending for Iteration 3
Resolved by: Will be resolved by Harrish Elango by the Iteration 3

```java
/**
 * This test must fail
 */
@Test
public void testHireNurse_DuplicateEntry() {
    Nurse newNurse = new Nurse("Anna", "Small", 28, "Female", "123 Helendale Ave");
    dbOps.addNurse(newNurse);
    Exception exception = assertThrows(SQLException.class, () -> {dbOps.addNurse(newNurse);});
    assertNotNull(exception);
}
```

Bug 3:

Problem Report Form:

Problem Report Number: 3
Reported by: Paramount Khalkhali Sharifi
Date Reported: 24/03/2024
Program Name: Nurse Hiring Module
Release Number: 1
Version Identifier: Build 1
Configuration(s): Windows 10, Java 17, Hospital Management System
Report Type: Validation Issue
Can Reproduce: Yes
Severity: Low
Priority: High
Problem Summary: System allows invalid age input for nurses.
Key Words: Nurse registration, age validation.
Problem Description and How to Reproduce It:

- Navigate to the "Hire Nurse" section.
- Enter an invalid age for a nurse, e.g., -12.
- Submit the form.
- The system incorrectly accepts the invalid age and adds the nurse to the system.
- Expected behavior: The system should validate that the age is within a reasonable range and reject negative values.

Suggested Fix: Include a range check for age (e.g., must be between 20 and 80).
Status: Open
Resolution: Pending for Iteration 3
Resolved by: Will be resolved by Harrish Elango by the Iteration 3

```java
/**
 * This test must fail
 */
@Test
public void testHireNurse_InvalidAge() {
    Nurse invalidNurse = new Nurse("Invalid", "User", -10, "Unknown", "44 New st");
    Exception exception = assertThrows(SQLException.class, () -> {
        dbOps.addNurse(invalidNurse);
    });
    assertNotNull(exception);
}
```

User story 4:

- **Complete Junit test case for backend:**

```
package Hospital;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.sql.SQLException;
import static org.junit.jupiter.api.Assertions.*;

public class User4test {

    private DatabaseOps dbOps;

    @BeforeEach
    public void setUp() {
        dbOps = new DatabaseOps();
       // sample Nurse
        Nurse testNurse = new Nurse("TestFirstName", "TestLastName", 30, "Female",
"Test Address");
        dbOps.addNurse(testNurse);
    }

    @AfterEach
    public void tearDown() {
        dbOps.deleteNurse("TestFirstName");
    }

    @Test
    public void testResignNurse_Successful() {
        // Before removing, ensure the nurse exists
        assertTrue(nurseExists("TestFirstName", "TestLastName"));
        dbOps.deleteNurse("TestFirstName");
        // After removal, the nurse should no longer exist
        assertFalse(nurseExists("TestFirstName", "TestLastName"));
    }

    @Test
    public void testResignNurse_NonExisting() {
        // Attempt to remove a non-existing nurse should not throw an error
```

```
        assertDoesNotThrow(() -> dbOps.deleteNurse("NonExistingFirstName"));
    }

    @Test
    public void testResignNurse_NullName() {
        // Attempt to remove a nurse with a null name should handle the error
        Exception exception = assertThrows(SQLException.class, () ->
dbOps.deleteNurse(null));
        assertNotNull(exception);
    }
    @Test
    public void testResignNurse_Success() {
        Nurse existingNurse = new Nurse("Existing", "Nurse", 30, "F", "123 Test St");
        dbOps.deleteNurse(existingNurse.getFirstName());
        assertFalse(dbOps.getAllNurses().contains(existingNurse));
    }

    @Test
    public void testResignNurse_NotInList() {
        Nurse nonExistentNurse = new Nurse("Nonexistent", "Nurse", 30, "F", "321 Test
St");
        dbOps.deleteNurse(nonExistentNurse.getFirstName());
        assertFalse(dbOps.getAllNurses().contains(nonExistentNurse));
    }



    @Test
    public void testResignNurse_AttemptToRemoveTwice() {
        Nurse existingNurse = new Nurse("Existing", "Nurse", 30, "F", "123 Test St");
        dbOps.deleteNurse(existingNurse.getFirstName());
        assertThrows(SQLException.class, () ->
dbOps.deleteNurse(existingNurse.getFirstName()));
    }

    @Test
    public void testResignNurse_InvalidNurseName() {
        assertThrows(SQLException.class, () -> dbOps.deleteNurse(""));
    }

    @Test
    public void testResignNurse_RemoveUsingPartialName() {
        Nurse existingNurse = new Nurse("Existing", "Nurse", 30, "F", "123 Test St");
        assertThrows(SQLException.class, () -> dbOps.deleteNurse("Exist"));
    }



    private boolean nurseExists(String firstName, String lastName) {
        return dbOps.getAllNurses().stream()
```

```
                .anyMatch(nurse -> nurse.getFirstName().equals(firstName) &&
    nurse.getLastName().equals(lastName));
        }
    }
```

Passed:

Test Case1:

```
@Test
public void testResignNurse_Successful() {
    // Before removing, ensure the nurse exists
    assertTrue(nurseExists("TestFirstName", "TestLastName"));
    dbOps.deleteNurse("TestFirstName");
    // After removal, the nurse should no longer exist
    assertFalse(nurseExists("TestFirstName", "TestLastName"));
}
```

- Explanation: The test confirmed that a nurse could be successfully removed from the database, as evidenced by the nurse no longer existing after the operation.

Test Case 2:

```
@Test
public void testResignNurse_NonExisting() {
    // Attempt to remove a non-existing nurse should not throw an error
    assertDoesNotThrow(() -> dbOps.deleteNurse("NonExistingFirstName"));
}
```

- Explanation: The system correctly handled an attempt to remove a nurse that does not exist, without throwing an error.

Test Case 3:

```
@Test
public void testResignNurse_Success() {
    Nurse existingNurse = new Nurse("Existing", "Nurse", 30, "F", "123 Test St");
    dbOps.deleteNurse(existingNurse.getFirstName());
    assertFalse(dbOps.getAllNurses().contains(existingNurse));
}
```

- Explanation: This test confirmed the successful removal of an existing nurse.

- **General steps for testing DB and GUI:**

1. Choose to log in as admin by pressing “Admin login”
2. Enter username “admin” and password “pass”
3. Click on “Resign Nurse”
4. Choose the nurse that you wish to resign from the list.
5. Click the “Remove selected nurse” button to remove the nurse from the hospital system and database.
6. The nurse should now be removed from the hospital system and database. You can make sure by looking into the “Resign nurse” list or by trying to login to the system with the old nurse credentials which should now be invalid.

Bugs report:

Bug Report 1:

Problem Report Number: 1
Reported By: Paramount Khalkhali Sharifi
Date Reported: 2024-03-24
Program (Component) Name: Nurse Resignation
Release Number:1
Version Identifier:1

Report Type: Coding issue
Can Reproduce: Yes
Severity: Medium
Priority: low

Problem Summary: Attempt to remove a nurse using a null name throws SQLException.
Key Words: null, deleteNurse, SQLException
Problem Description and How to Reproduce It:
When attempting to remove a nurse from the database by passing a null name to deleteNurse, the system throws an SQLException instead of handling the null value.
Suggested Fix: Implement null checks before attempting to execute database operations to prevent SQLException.
Status: Open
Resolution: Pending
Junit test case:

```
@Test
public void testResignNurse_NullName() {
    // Attempt to remove a nurse with a null name should handle the error
    Exception exception = assertThrows(SQLException.class, () -> dbOps.deleteNurse(null));
    assertNotNull(exception);
}
```

Bug Report 2:

Problem Report Number: 2
Reported By: Paramount Khalkhali Sharifi
Date Reported: 2024-03-24
Program (Component) Name: Nurse Resignation
Release Number:1
Version Identifier:1

Report Type: Coding issue
Can Reproduce: Yes
Severity: High
Priority: High

Problem Summary: Removal of a nurse not in the list does not throw an expected exception.
Key Words: non-existent, deleteNurse, exception

Problem Description and How to Reproduce It:
Attempting to remove a non-existent nurse should throw an exception indicating the nurse does not exist, but it does not, potentially indicating an error in exception handling or list management.

Suggested Fix: Ensure that the deleteNurse method throws an exception when attempting to delete a nurse that is not in the list of nurses.

Status: Open

Resolution: Pending

Junit test case:

```
@Test
public void testResignNurse_NotInList() {
    Nurse nonExistentNurse = new Nurse("Nonexistent", "Nurse", 30, "F", "321 Test St");
    dbOps.deleteNurse(nonExistentNurse.getFirstName());
    assertFalse(dbOps.getAllNurses().contains(nonExistentNurse));
}
```

Bug Report 3

Problem Report Number: 3
Reported By: Paramount Khalkhali Sharifi

Date Reported: 2024-03-24
Program (Component) Name: Nurse Resignation
Release Number: 1
Version Identifier:1

Report Type: Design issue
Can Reproduce: Yes
Severity: Low
Priority: Low

Problem Summary: Attempt to remove a nurse using a partial name throws SQLException.
Key Words: partial name, deleteNurse, SQLException

Problem Description and How to Reproduce It:
When attempting to remove a nurse by providing only a partial first name to deleteNurse, an SQLException is thrown instead of the system handling the partial name correctly.

Suggested Fix: Adjust the deleteNurse method to handle partial names appropriately, possibly by using a search before attempting to delete.

Status: Open

Resolution: Pending

Junit test case:

```
@Test
public void testResignNurse_RemoveUsingPartialName() {
    Nurse existingNurse = new Nurse("Existing", "Nurse", 30, "F", "123 Test St");
    assertThrows(SQLException.class, () -> dbOps.deleteNurse("Exist"));
}
```

Bug Report 4:

Problem Report Number: 4
Reported By: Paramount Khalkhali Sharifi
Date Reported: 2024-03-24
Program (Component) Name: Nurse Resignation
Release Number:1
Version Identifier:1

Report Type: Coding issue
Can Reproduce: Yes

Severity: Medium
Priority: Medium

Problem Summary: Nurse can be attempted to be removed twice, throwing SQLException on second attempt.
Key Words: double removal, deleteNurse, SQLException

Problem Description and How to Reproduce It:
When trying to remove a nurse that has already been deleted, the system throws a SQLException on the second deletion attempt. This could indicate that the state of nurse existence is not being properly tracked after the first deletion.

Suggested Fix: Implement a check within the deleteNurse method to determine if a nurse has already been removed before proceeding with the operation, thus avoiding the SQLException.

Status: Open

Resolution: Pending

Junit test case:

```
@Test
public void testResignNurse_AttemptToRemoveTwice() {
    Nurse existingNurse = new Nurse("Existing", "Nurse", 30, "F", "123 Test St");
    dbOps.deleteNurse(existingNurse.getFirstName());
    assertThrows(SQLException.class, () -> dbOps.deleteNurse(existingNurse.getFirstName()));
}
```

All possible user stories Bugs in the Nurses Database:

```
postgres=# SELECT * FROM nurses;
 id  | firstname | lastname | age | gender  |     address     | username | password
-----+-----------+----------+-----+---------+-----------------+----------+----------
 139 | Nina      | Smith    | -12 | Female  | 17 newparkway   | nina     | 12345
 107 | Nina      |          |  21 |         |                 |          |
 100 | Invalid   | User     |  -1 | Unknown | 123 Test St     |          |
 114 | John      | Doe      |  30 | M       | 123 Test St     |          |
 101 | Jane      | Doe      |  30 | F       | 123 Test St     |          |
 102 | John      | Doe      |  30 | M       | 123 Test St     |          |
 109 | Invalid   | User     |  -1 | Unknown | 123 Test St     |          |
 110 |           | Doe      |  25 | F       | 123 Test St     |          |
 111 | Invalid   | User     |  -1 | Unknown | 44 New st       |          |
 112 |           | Nil      |  25 | Female  | 123 Valley St   |          |
 113 | Invalid   | User     |  -1 | Unknown | 44 New st       |          |
 115 |           | Nil      |  25 | Female  | 123 Valley St   |          |
 116 |           | Nil      |  25 | Female  | 123 Valley St   |          |
 117 |           | Nil      |  25 | Female  | 123 Valley St   |          |
 118 | Alice     | Smith    |  28 | Female  | 33 Amazing St   |          |
 119 | Bob       | Jones    |  32 | Male    | 7 Maplefield St |          |
 152 |           | Nil      |  25 | Female  | 123 Valley St   |          |
 120 | Invalid   | User     | -10 | Unknown | 44 New st       |          |
 121 | Alice     | Smith    |  28 | Female  | 33 Amazing St   |          |
 122 | Bob       | Jones    |  32 | Male    | 7 Maplefield St |          |
 163 |           | Nil      |  25 | Female  | 123 Valley St   |          |
 124 | Invalid   | User     | -10 | Unknown | 44 New st       |          |
 125 | Alice     | Smith    |  28 | Female  | 33 Amazing St   |          |
 126 | Bob       | Jones    |  32 | Male    | 7 Maplefield St |          |
 177 |           | Nil      |  25 | Female  | 123 Valley St   |          |
 103 | Alice     | Smith    |  28 | Female  | 33 Amazing St   |          |
 104 | Bob       | Jones    |  32 | Male    | 7 Maplefield St |          |
 161 |           | Nil      |  25 | Female  | 123 Valley St   |          |
 132 | Invalid   | User     | -10 | Unknown | 44 New st       |          |
 133 | Alice     | Smith    |  28 | Female  | 33 Amazing St   |          |
 134 | Bob       | Jones    |  32 | Male    | 7 Maplefield St |          |
 201 |           | Nil      |  25 | Female  | 123 Valley St   |          |
(32 rows)


postgres=#
```

PART 2:

Quick introduction: In this part, we review the user stories to find out if the code smells or not. We will investigate each code based on the Material provided in Tutorials. As I am assigned to test the user stories 2 and 4, I will review all directly related classes and GUIs to these user stories. Here is the list of the classes I review:

- **AdminGUI**
- **FireNurseGUI**
- **DatabaseOps**
- **Nurse**
- **LoginGUI**

AdminGUI:

- Duplicate Static Instances: static DatabaseOps dbOps = new DatabaseOps(); should be a singleton, so it restricts the instantiation of the class to a singular instance.
- Inconsistent Naming Conventions: The naming for buttons like addPhysician and firePhysician is not consistent with Java naming conventions.
- God Object: This class is managing the GUI, as well as direct interactions with the database operations through dbOps, so it's interacting with many methods at the same time.
- Large Class: AdminGUI is performing multiple roles (UI and business logic), which could be separated.

FireNurseGUI:

- Feature Envy: The methods in this class seem more concerned with Nurse and DatabaseOps than with FireNurseGUI's responsibilities.
- Magic Strings: Hardcoded strings for UI labels like "Remove Selected Nurse" should be constants or externalized.
- Inconsistent Naming Conventions: nurseComboBox vs removeButton (field names should consistently use or avoid prefixes).

DatabaseOps:

- God Object: Handles too many different database operations, which could be split into different classes.
- Exception Handling: Exceptions are printed but not re-thrown or handled meaningfully.
- Hardcoded Strings: SQL queries are hardcoded into the methods, which could be externalized for better management.
- Long Methods: Methods like addPatient and getAllPhysicians are quite long and could be broken down.
- Dead Code: Commented-out methods like deletePatient2 should be removed if they are not used. (might be implemented later)

Nurse:

- Large Class: This class has many responsibilities, including managing patients and interacting with StubDB.
- Inconsistent Naming Conventions: Mixed use of camelCase and snake_case in methods and variables.
- Poor Encapsulation: Directly modifying fields of other classes, such as patient.setNurse(this);.
- Magic Numbers: The number 15 in addPatient could be defined as a constant representing the maximum number of patients a nurse can handle.It needs more clarification.
- Duplicate Code: The compareTo method seems to duplicate some logic found in equals.

LoginGUI:

- Magic Strings: Uses hardcoded strings for the admin user and password.
- Potential Security Flaws: Passwords appear to be managed within the code in plain text.
- Duplicate Static Instances: static DatabaseOps dbOps = new DatabaseOps(); is repeated here as well.
- Long Method: The actionPerformed method is doing too much and could be broken down into helper methods.