CSCD01 Deliverable 2

Team Number: 11
Team Name: Phantom Developers
Date: March 11th, 2020

Repository: GitHub

Table of Contents

Issue List

Feature - Automatically Merge Duplicate Patients Identifier Types

Feature:
Depending on the system, a patient identifier can be a Social Security Number, Driver's License Number, OpenMRS ID, Medical Record Number, etc. They want to find a way to merge duplicate patient identifier types if they exist. The feature is difficult to implement because we need to make queries to figure out if there are any duplicates.

Problems with implementing:
Suppose there are patient identifier types named social security number and SSN. They represent the same thing but it's difficult to create an algorithm that would be able to distinguish this. Choosing which identifier type to merge would have to be done manually. Then we need to somehow merge every instance of SSN and social security number together. One could use xxx-xxx-xxx formatting while the other is xxxxxxxxx. If there are conflicts, then we don't know which one to keep or discard. If there are conflicts when merging, we need to decide what to do such as throwing an exception.

Complexity: Medium, Priority: Could

Possible Fix in Code:
Need to add to the PatientDAO a method to merge identifier types. And in the Patient service add a method to call the DAO.

Estimated time to complete:
We estimate that it will take approximately 9-11 hours (4-5 days) to complete the feature. Two hours to find out where and how the patient identifier types are stored in the database. An hour to write the queries in MySQL workbench, and then another two hours to figure out how to write similar queries in the OpenMRS app. Lastly, 4-6 hours to complete the API and testing.

Pros
- Removes unnecessary duplicates
- Useful feature for OpenMRS users.
- Implementation open to interpretation

Cons
- Not enough information is given in the issue report. Very vague
- Created in 2013. No one has ever worked on it. No updates.
- Cannot automatically decide what information to keep when there are merge conflicts

Bug - PatientService.savePatient isn't threadsafe and can allow duplicate patient identifiers

PatientServiceImpl.savePatient(...) ends with these two lines:

```
if (!patient.isVoided()) {
     checkPatientIdentifiers(patient);
}
Return dao.savePatient(patient);
```

These should be synchronized so that if two threads are trying to save patient objects with duplicate identifiers, then they cannot both complete checkPatientIdentifiers before saving the patient.

Complexity: Medium, Priority: Should

Possible Fix in Code:
Add locks in the PatientService.savePatient() method

Estimated time to complete:
We estimate that it will take approximately 1 hours to complete the testing (JUnit testing) and approximately 3 hours to fix the given bug, 2 hours on POC of Java Multi-thread

Pros:
- Task description is very clear
- Medium difficulty (Complex, but doable)

Cons:
- Bug that deals with multi-thread, hard to test manually
- Old ticket, created from 2013

Bug - Under Manage Reports, reports that are marked as red should disappear if either SQL or Reporting Mapping is done.

This bug is that when the user goes to manage reports, the reports that are marked as red should disappear if there is SQL or a report mapping done for it. However, currently, it only checks for the SQL and not the report mapping. Unfortunately, the ticket doesn't explain what it means by SQL being done. E.g. if the report is in the database, or something else. Also, we assume that report mapping means clicking the "Map Report" button. This bug is for version 2.0.

Complexity: Low, Priority: Should

Possible Fix in Code:
To fix this issue there would likely be work needed on the SQL queries in the DAO to properly update if the report should be marked red or not when the user does the report mapping

Estimated time to complete:
12 hours. It is estimated that to understand the problem it would take 3 hours. For implementing the changes it would take 4 hours. To write the tests and debug it would take 5 hours.

Pros:
- Priority is “should”, so it is something that the community wants to have
- Known estimates and affected versions

Cons:
- Unclear on the issue description and follow-ups on what “checks for SQL” means or “report mapping” means
- Some people were not able to reproduce the issue, not sure how to reproduce

Feature - Support formulary status for drugs

This feature adds a “formulary” status to drugs, which are used to put drugs into a list. Each drug can be assigned to more than one formulary. Queries and management of formulary should also be supported.

Complexity: Medium, Priority: Should

Possible Fix in Code:
Add a new class “formulary” which will store a list of drugs. To support queries for the user, adding methods to ConceptService and ConceptDao for formulary would be needed.

Estimated time to complete: 5-7 days
It is estimated that this would take around 18-24 hours. 12-18 hours to implementation and understanding of the code. 4-6 hours to create test cases.

Pros:

- Assigning formularies to drugs is a practice used by hospitals, clinics, etc. So having this feature would be helpful
- Priority on the issues is should, so it would be a nice addition to the program
- Has a list of acceptance requirements

Cons:

- Relatively big compared to other issues
- No estimation time of task on issues board

Feature - Map the username to the email of the user

This issue is a sub-task of a feature to allow the use of a user’s email address as their username. This feature has already been implemented but there are still some inconveniences where the user has to enter the email multiple times. Specifically, the user should not have to put in their email twice, once as their username and the other in the Notifications section of the My Profile.

Complexity: Medium, Priority: Non-essential

Possible Fix in Code:
The user notification address should be set as a username if the username is a valid email.

Estimated time to complete:
Four hours. Three hours on implementing it so that the email used for a user is also saved for the notifications section and one hour for tests.

Pros:

- Saves time when creating a user
- Actual implementation of email as a username is already in place
- Locations in code to modify already listed

Cons:

- Ticket created in 2012 and not updated since
- Requires modifying the frontend to set the email notification upon user creation

Features Implemented

Feature 1 - Map the username to the email of the user

Why we chose this feature

We chose this feature because the requirement of this feature is very clear. In addition, Since the other feature that we chose to implement is relatively large and complex, we want to choose a feature that is not very complex. Since this feature is a subtask of an existing feature, adding the setting to allow emails to be used as a user's username is simple, as most of the work has already been completed. The only thing we had to do was to set the additional property when a new user is created.

Time estimate

3 hours for investigation and implementing the feature and 1 hour for writing the unit test

Anticipated risks

This was a subtask for an issue that was reported in 2012, so the design of the system may have changed between then and now, so it may require investigating how the system may have changed. The issue also seems very simple and has been left in the "Ready for Work" state with no assignees, so there may be some unforeseen risks that may not be apparent on the surface.

Design changes

No change to the structure of the file was made. Only added a couple of lines.

Code changes

In UserServiceImpl.java under the createUser method. The user notification address should be set as a username if the username is a valid email.

Customer acceptance tests

1. Build and run project
2. Login in as admin (username: admin password: Admin123)
3. Go to "Advanced Settings" under "Administration" tab
4. Set "user.requireEmailasUsername" to "true"
5. Go back to "Administration" and go to "Manage user" section
6. Create a new user using a valid email as username
7. Logout current admin account
8. Login as newly created user
9. Go to "My Profile"
10. Go to "Notification" section
11. The email for notifications should be automatically set to the username (email)

Tests Implemented

All tests were implemented in UserServiceTest.java

1. Implemented a test that creates a user with a valid email as a username. The notification address should be automatically set as a username when “user.requireEmailasUsername” is set to “true”
2. Implemented a test that refuses to create the user if the username is an invalid email when “user.requireEmailasUsername” is set to “true”

Feature 2 - Automatically Merge Duplicate Patients Identifier Types

Why we chose this feature

This feature would bring value to the project because it allows users to automatically merge patient identifier types instead of having to do them manually. In a system with a large amount of people, this could save lots of time doing a merge.

Time estimate

We estimate that it will take approximately 9-11 hours (4-5 days) to complete the feature. Two hours to find out where and how the patient identifier types are stored in the database. An hour to write the queries in MySQL workbench, and then another two hours to figure out how to write similar queries in the OpenMRS app. Lastly, 4-6 hours to complete the API and testing.

Anticipated risks

Because we need to change the patient DAO, there is a risk of unsafe SQL queries that have unintended consequences on the database. Also, the SQL queries could be insecure from SQL injection. We could miss the numerous edge cases that have to do with merging different patient identifiers and potential duplicates.
The only requirement was vague which meant that we had to make assumptions about what feature was supposed to be.

Design changes

No changes to the architecture. Only added code to the existing DAO and service class.
Nothing else was changed/deleted.

Code changes

- Added another method to interface PatientDAO.java called mergePatientIdentifier. It is called to merge two patient identifier types.
- Implemented the actual query to merge identifier types in HibernatePatientDAO.java called mergePatientIdentifier.
- Added tests to test the new feature in HibernatePatientDAOTest.java called *getPatientIdentifer_shouldGiveErrorWhenThereAreDuplicateIdentifiers* and *getPatientIdentifer_shouldMergeThePatientIdentiferTypes.*
- Added method in PatientService.java called *mergePatientIdentifierType* to merge the identifier types
- Added tests in PatientServiceTest.java to test that the new method in the service can run the DAO, called *mergePatientIdentifierType_shouldDeleteOtherIdentifierType* and *mergePatientIdentifierType_shouldChangePatientIdentifierTypesOfPatients*

Customer acceptance tests

Unfortunately, the feature cannot be tested from the UI and can only be tested through JUnit. The UI is not a feature of OpenMRS-Core so we decided not to modify it for this deliverable.

Tests Implemented

2 tests were implemented in HibernatePatientDAOTest.java

1. Implemented a test that merges two patient identifier types. After merging, it checks whether all PatientIdentifierTypes have been merged to the same one (chosen by the user). Lastly, it checks whether the PatientIdentifierType id has been changed for the Patients.
2. Implemented a test that tests whether duplicate PatientIdentifiers give an exception. If there are two duplicate patient identifier types when merging, then our function should stop and throw an exception. If there is an exception, then the test passes, otherwise, it fails. This test tests whether the code can handle unexpected inputs, and throw a correct exception when expected.

2 tests were implemented in PatientServiceTest.java

3. Implemented a test that tests that if one identifier type gets merged into another, the merged identifier type should be deleted
4. Implemented a test that tests if a patient's identifier type gets merged, it should change the patient's identifier type correctly

Software Development Process

Kanban Board

Our team used the Kanban Board to have a visualization of the tasks needed to be accomplished. We never had to increase our In Progress limit as we did not have that many cards and multiple people worked on the same card.

Initially, we only had one card for each issue we were working on. However, we soon realized that having a single card for the entire issue made it hard to track who was doing which task in the checklist. Therefore, we split the tasks up into multiple cards, each with their own checklist, label, due date, and members. This made sure that the members know what they should work on and not worry about overlapping work.

In addition to modifying the number of cards, we added an extra column to our Kanban board, Under Review. We found that it was difficult to indicate the state of a card by the checklist within the card, so we added the extra column to make it easier to visualize. This made sure that the code was verified and looked over by other members

Team Meetings

Online Daily Stand-Up Meetings

The team originally planned on having daily meetings as Discord calls, but it turned out to be difficult to keep it up for the team. The main reason being that there are times where some or all members may not have made progress due to external factors such as other schoolwork, which made the meeting unnecessary.

We learned that the better way of doing these is to have these kinds of meetings to be structured around due dates that are set by the cards on the kanban board. By having a meeting one or two days before the due date, members that are assigned to those tasks are more likely to have progress made, and let other team members know what they have done so far and if they are stuck on a problem. Thus making the team more productive by making sure that there is something to gain from the meetings.

In-Person Meetings

The team had a mandatory meeting on Monday and an optional one on Friday, which so far was treated as a mandatory meeting as well. In these meetings, we discussed broader topics about a project, such as deciding as a team on which issues to work on for deliverable 2 and what tasks or cards will be put on the Kanban board. While these meetings are usually used for planning out a deliverable, they were also useful in providing in-person help to other members who may be running into problems or have questions about the project.

Code Repository

The way we structured our branches made sense initially as we named branches corresponding to the issue. However, we soon diverged from this structure and multiple people created multiple branches for the same issue, making it hard to maintain and resulted in difficulties when merging. Also, we realized in the middle that the OpenMRS repository style required feature and bug fix branches to be named by the issue id on their Jira issue tracking website.

Installing and Running Tests

1. Clone our project repository.
2. Follow the OpenMRS installation guide for developers.
3. Open the JUnit Test file for the specific feature (HibernatePatientDAOTest).
4. Run as JUnit through an IDE. It can also run using “mvn -Dtest=HibernatePatientDAOTest test” for a specific test or can do “mvn clean package” to build and run all tests.