CSCD01 Team 21

Deliverable #4

Team 21: *The BugWriters*

Chengrong Zhang, Zhongyang Xia, Tan Jiaxin, Zongye Yang,
Xingyuan Zhu

March - April 2020

Contents

0.1 Feature Design

In this section, we will show you how our team designed the feature and how we followed the implementation plan we have created back in deliverable 3.

0.1.1 Code Design

All following modified java files are under the implemented_feature folder:
https://github.com/CSCD01/team_21-project/tree/master/Deliverable4/implemented_feature/api/src/main/java/org/openmrs

Implementation in PatientIdentifierType.java:

The PatientIdentifierType class is implemented inside PatientIdentifierType.java file, thus for adding the new auto-merge feature, the first thing our team needs to do is make the PatientIdentifierType object comparable. Hence, we need to implement a function called isDuplicate() which takes another PatientIdentifierType object as a parameter to compare the self-object with the given one. If the two PatientIdentifierType objects are considered as duplicates, then the function will return true, otherwise, return false. The next thing we need to decide is to declare in what situations the two PatientIdentifierType objects are considered equal. As we can see in PatientIdentifierType.java, the PatientIdentifierType object has two main fields, which are id (integer) and format (string), thus if anyone of those two fields of two PatientIdentifierType objects are the same, then the two objects are considered as the duplicates. But the class also contains other fields for example formatDescription (string) and name (string), as our team did a further discovery, we finally decided three different cases of comparison for two PatientIdentifierType objects:

1. If two PatientIdentifierType objects have the same id or same format, then they are considered duplicates.

2. If two PatientIdentifierType objects have the same format description, then they are considered duplicates.

3. Otherwise, the two PatientIdentifierType objects are not equal.

Here is the code of isDuplicate() function:

```
217        /**
218         * the function checks the similarity of the current IdentifierType and the
219         * given IdentifierType if two consider the same then the function return true,
220         * otherwise return false
221         *
222         * @param patientIdentifier
223         * @return the IdentifierType is equal to the given one
224         */
225        public boolean isDuplicate(PatientIdentifierType patientIdentifier) {
226            // if the id or the format is same, then two PatientIdentifierTypes are same
227            if ((this.patientIdentifierTypeId == patientIdentifier.getPatientIdentifierTypeId())
228                        || (this.format == patientIdentifier.getFormat())) {
229                return true;
230            } else if (this.formatDescription != null
231                        && (this.formatDescription == patientIdentifier.getFormatDescription())) {
232                // if the formatDescription is same, then two PatientIdentifierTypes are same
233                return true;
234            } else {
235                // else the two given PatientIdentifierTypes are not same then return false
236                return false;
237            }
238        }
```

Figure 1.1: isDuplicate() function

The next function we implemented is called merge(), the function meant to merge all fields of two duplicates into one PatientIdentifierType object, for example: assume the new created PatientIdentifierType object named "type1" is considered as a duplicate with the "type0" that already exists in the database, then the merge() function will check if "type0" has null format, null formatDescription, null validator, null locationBehavior, or null locationBehavior, if so then the function will assign any null fields of "type0" with the value of fields in "type1".

Here is the code of merge() function:

```java
240        public void merge(PatientIdentifierType newType) {
241                // peasant short circut
242                if (format == null)
243                        format = newType.getFormat();
244                required = required || newType.getRequired();
245                if (formatDescription == null)
246                        formatDescription = newType.getFormatDescription();
247                if (validator == null)
248                        validator = newType.getValidator();
249                if (locationBehavior == null)
250                        locationBehavior = newType.getLocationBehavior();
251                if (uniquenessBehavior == null)
252                        uniquenessBehavior = newType.getUniquenessBehavior();
253        }
```

Figure 1.2: merge() function

Implementation in HibernatePatientDAO.java:

As we discussed in deliverable 3, the HibernatePatientDAO.java is the file that controls the update/delete of patient-related objects in the database. Therefore, to implement the auto-merge function we need to implement the function called hasDuplicate() inside HibernatePatientDAO.java. The hasDuplicate() function will take a PatientIdentifierType object as a parameter, the function is designed to communicate with the database and get all the existing PatientIdentifierType objects first, then the function will compare all PatientIdentifierType objects with the given parameter PatientIdentifierType, by using the above-implemented comparison function isDuplicate(), if the function returns true, that means the two PatientIdentifierType are equal, and the hasDuplicate() will merge them.

Here is the code of hasDuplicate() function:

```
1041        /**
1042         * Check if there exists a duplicate patientIdentifierType
1043         */
1044        public Optional<PatientIdentifierType> hasDuplicate(PatientIdentifierType patientIdentifierType) {
1045            // Get all patientIdentifierType
1046            Criteria criteria = sessionFactory.getCurrentSession().createCriteria(PatientIdentifierType.class);
1047
1048            criteria.addOrder(Order.desc("required"));
1049            criteria.addOrder(Order.asc("name"));
1050            criteria.addOrder(Order.asc("patientIdentifierTypeId"));
1051
1052            @SuppressWarnings("unchecked")
1053            List<PatientIdentifierType> allList = criteria.list();
1054
1055            for (int i = 0; i < allList.size(); i++) {
1056                PatientIdentifierType type = allList.get(i);
1057                if (type.isDuplicate(patientIdentifierType) == true) {
1058                    // deletePatientIdentifierType(allList.get(i));
1059                    return Optional.of(type);
1060                }
1061            }
1062
1063            return Optional.empty();
1064        }
```

Figure 1.3: hasDuplicate() function

Code from line 1046 to line 1053 is meant to send the required data to the database and get all PatientIdentifierType objects from the database and store them as a list. Code from line 1055 to line 1061 is meant to check every PatientIdentifierType objects with the given one by calling the isDuplicate() function we just implemented in PatientIdentifierType.java, if the two are equal, then the function will the duplicate, otherwise, the function will return an empty Optional.

The above implementation only adds check and merge functionality, by implementing the auto-merge feature we also need to call check_Table() function whenever the database changes. Therefore, we need to rewrite the savePatientIdentifierType() function.

```
310        /**
311         * @see org.openmrs.api.db.PatientDAO#savePatientIdentifierType(org.openmrs.PatientIdentifierType)
312         */
313        @Override
314        public PatientIdentifierType savePatientIdentifierType(PatientIdentifierType patientIdentifierType)
315                        throws DAOException {
316        PatientIdentifierType ret;
317            Optional<PatientIdentifierType> optionalType = hasDuplicate(patientIdentifierType);
318            if (optionalType.isPresent()) {
319                    ret = optionalType.get();
320                    ret.merge(patientIdentifierType);
321            } else {
322                    ret = patientIdentifierType;
323            }
324            sessionFactory.getCurrentSession().saveOrUpdate(ret);
325            return ret;
326        }
```

Figure 1.4: New savePatientIdentifierType() function

Now, whenever a new PatientIdentifierType object is inserted to the database, the savePatientIdentifierType() function will automatically check if there exists a duplicate and merge them if so.
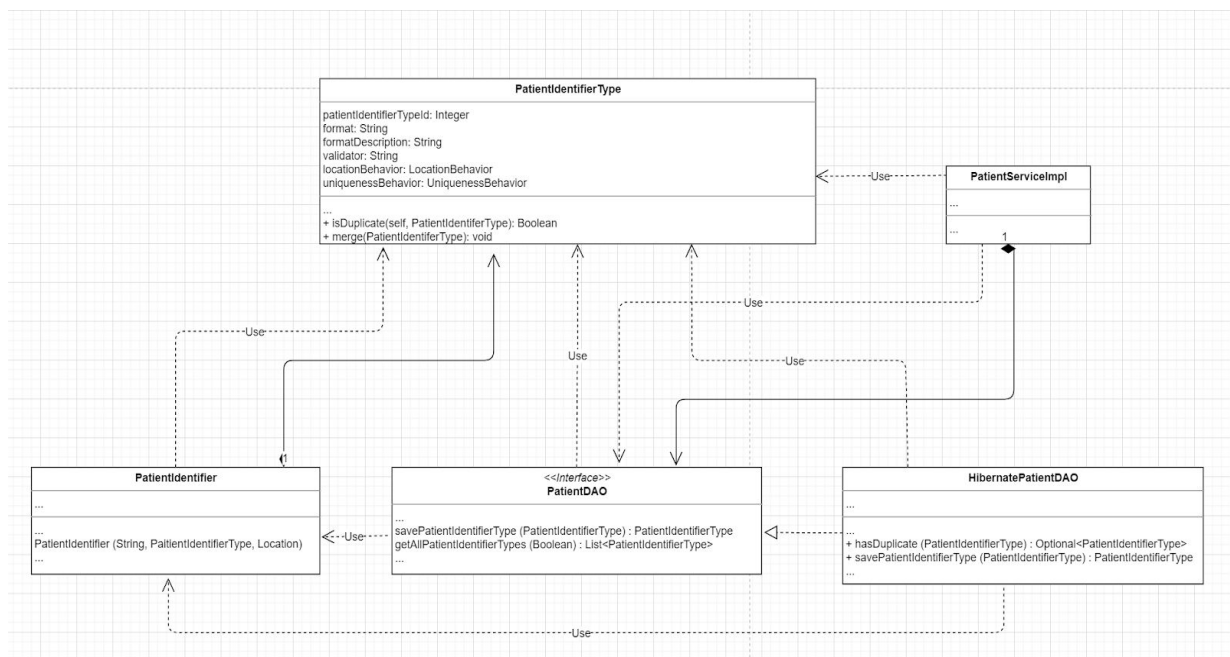
0.1.2 UML

Figure 1.5: UML

0.1.3 Changes with Implementation Plan

As we carried out the implementation plan from deliverable 3, the only big change we made is we added another function called merge(), as the original plan we are thinking about to use deletePatientIdentifierType() function to delete one of duplicates if they exist. But in our final work we reconsider the meaning of "merge" and think there might be a better way to merge duplicates not simply delete it. Therefore we implemented the merge() function to merge all fields of two duplicates into one complete PatientIdentifierType object. And also we did change the structure of deletePatientIdentifierType() function as we wrote in the original plan,since there is no need to track the database to find duplicates when we delete a PatientIdentifierType object. Besides we changed some names of functions, we meant to make the function's purpose more clear to other developers. And anything else is according to the original plan. So here is our final implementation plan:

1. Modify the structure of PatientIdentifierType class.
   Add a compare function to the PatientIdentifierType class thus PatientIdentifierType subject can be compared with another.

2. Implement isDuplicate() function in PatientIdentifierType.java.
   isDuplicate() function will return True if two PatientIdentifierType have the same id or same format or same formatDescription, and return False otherwise.

3. Implement merge() function in PatientIdentifierType.java.
   merge() function will merge all fields of two PatientIdentifierType into one.

4. Implement hasDuplicate() function inside HibernatePatientDAO.java.
   hasDuplicate() function will return PatientIdentifierType if there exist duplicate elements inside the database and merge them, otherwise the method will return empty Optional and do nothing.

5. Modify savePatientIdentifierType() function in HibernatePatientDAO.java.
   Call hasDuplicate() function at savePatientIdentifierType() thus whenever there is a new PatientIdentifierType added into the database, the system will merge duplicates.

0.2 User Guide

New Feature User Guide:

The user should make sure their openmrs-core version is up-to-date. Then, put PatientIdentifierType.java under the path:

openmrs-core/api/src/main/java/org/openmrs/

And put HibernatePatientDAO.java under the path:

openmrs-core/api/src/main/java/org/openmrs/api/db/hibernate/

The feature implemented by our team will give the OpenMRS software the functionality to auto-merge duplicate patient identifier types whenever a new patient identifier type is added to the database. In particular, the user needs to call the savePatientIdentifierType() function from HibernatePatientDAO.java to save a new PatientIdentifierType object into the database.

```
310        /**
311         * @see org.openmrs.api.db.PatientDAO#savePatientIdentifierType(org.openmrs.PatientIdentifierType)
312         */
313        @Override
314        public PatientIdentifierType savePatientIdentifierType(PatientIdentifierType patientIdentifierType)
315                    throws DAOException {
316        PatientIdentifierType ret;
317            Optional<PatientIdentifierType> optionalType = hasDuplicate(patientIdentifierType);
318            if (optionalType.isPresent()) {
319                    ret = optionalType.get();
320                    ret.merge(patientIdentifierType);
321            } else {
322                    ret = patientIdentifierType;
323            }
324            sessionFactory.getCurrentSession().saveOrUpdate(ret);
325            return ret;
326        }
```

Figure 2.1: Modified savePatientIdentifierType() function

And whenever the above function is called, the new feature will auto compare the given PatientIdentifierType object with every PatientIdentifierType object from the database, to check if there exists a duplicate. And for the definition of duplicate, our team believes if any two of PatientIdentifierType objects share the same id or have the same format or have the same formatDescription, then they will be considered as

duplicate. Then, if the duplicate exists, the new feature will auto-merge them by merging all fields of two objects into one.

Finally, if the user wishes to check the new feature to see if it really works, the user can use the following code after saving a new PatientIdentifierType object. The following code will print the id and the format of all PatientIdentifierType objects from the database. Then the user will be able to check if there exists a duplicate.

```java
Criteria criteria = sessionFactory.getCurrentSession().createCriteria(PatientIdentifierType.class);

criteria.addOrder(Order.desc("required"));
criteria.addOrder(Order.asc("name"));
criteria.addOrder(Order.asc("patientIdentifierTypeId"));

for (PatientIdentifierType p : criteria.list()) {
    System.out.println("Id =" + t.getPatientIdentifierTypeId());
    System.out.println("Format =" + t.getFormat());

}
```

Figure 2.2: Code that prints id and format of all PatientIdentifierType objects

Test Files User Guide:

There are two test files folder in git, one folder contains all acceptance test files named acceptance_tests, the other one contains all unit test files named unit_tests.

To run the tests, first the user should make sure the openmrs-core is at the newest version, then the user needs to set up the environment and run maven build success. After that the user need swap HibernatePatientDAOTest.java with the old one under the path:

openmrs-core/api/src/test/java/org/openmrs/api/db/hibernate/

And put PatientIdentifierType.java under the path:

openmrs-core/api/src/test/java/org/openmrs/

And put all acceptance test files under the path:

openmrs-core/api/src/test/java/org/openmrs/api/db/hibernate/

Now the users are able to run the test files.

0.3 Acceptance Test

In deliverable 3 our team designed a series of acceptance tests, they are all able to run and are valid to test for different cases, but once we actually started our implementation work we found that there are some more cases we can test. Therefore, we redesigned the acceptance tests by implementing a number of test files, each of them representing a series of user's actions. The following are the final version of our acceptance test files, with a description of each file's functionality. All following acceptance test files are under the acceptance_tests folder:

https://github.com/CSCD01/team_21-project/tree/master/Deliverable4/acceptance_tests/api/src/test/java/org/openmrs/api/db/hibernate

- test_User1.java
  When the user creates and saves a new PatientIdentifierType object into the database, with the id different with every other object in the database. Then if the user wants to get a list of all PatientIdentifierType objects in the database. As a result, the list should now contain the new saved PatientIdentifierType object the user created.

```java
13    public class test_User1 extends BaseContextSensitiveTest{
14           private HibernatePatientDAO hibernatePatientDao;
15
16           @Before
17           public void beforeEach() {
18                   updateSearchIndex();
19                   hibernatePatientDao = (HibernatePatientDAO) applicationContext.getBean("patientDAO");
20           }
21
22           @Test
23       public void test() {
24                   // Get the id of all PatientIdentifierType objects
25                   List<PatientIdentifierType> list = hibernatePatientDao.getAllPatientIdentifierTypes(true);
26                   for (int i = 0; i < list.size(); i ++) {
27                           System.out.println(list.get(i).getId());
28                   }
29
30                   // Save a new PatientIdentifierType with id be unique
31                   PatientIdentifierType type1 = new PatientIdentifierType();
32                   type1.setName("type1");
33                   hibernatePatientDao.savePatientIdentifierType(type1);
34
35           // The output should be 5 since there are 4 objects already exists in the database
36                   assertEquals(5, hibernatePatientDao.getAllPatientIdentifierTypes(true).size());
37           }
38
39    }
```

Figure 3.1: test_User1.java

- test_User2.java

  When the user creates and saves a new PatientIdentifierType object into the database, with the id = 1. Suppose there already exists a PatientIdentifierType object with id = 1 inside the database, then if the user wants to get a list of all PatientIdentifierType objects in the database. As a result, the list should not contain the new PatientIdentifierType object the user created.

```java
1   package org.openmrs.api.db.hibernate;
2
3   import static org.junit.Assert.assertEquals;
4
5   import org.junit.Before;
6   import org.junit.Test;
7   import org.openmrs.PatientIdentifierType;
8
9   import org.openmrs.test.BaseContextSensitiveTest;
10
11  import java.util.List;
12
13  public class test_User2 extends BaseContextSensitiveTest{
14          private HibernatePatientDAO hibernatePatientDao;
15
16          @Before
17          public void beforeEach() {
18                  updateSearchIndex();
19                  hibernatePatientDao = (HibernatePatientDAO) applicationContext.getBean("patientDAO");
20          }
21
22          @Test
23      public void test() {
24                  // Get the id of all PatientIdentifierType objects
25                  List<PatientIdentifierType> list = hibernatePatientDao.getAllPatientIdentifierTypes(true);
26                  for (int i = 0; i < list.size(); i ++) {
27                          System.out.println(list.get(i).getId());
28                  }
29
30                  // Save a new PatientIdentifierType with id be 1
31                  PatientIdentifierType type1 = new PatientIdentifierType();
32                  type1.setName("type1");
33                  type1.setId(1);
34                  hibernatePatientDao.savePatientIdentifierType(type1);
35
36      // The output should be 4 since there are 4 objects already exists in the database and the new one will not save into the database
37                  assertEquals(4, hibernatePatientDao.getAllPatientIdentifierTypes(true).size());
38          }
39
40  }
```

Figure 3.2: test_User2.java

- test_User3.java

When the user creates and saves two new PatientIdentifierType objects into the database, both with the unique id and unique format. Then if the user wants to get a list of all PatientIdentifierType objects in the database. As a result, the list should now contain two PatientIdentifierType objects the user created.

```java
3    import static org.junit.Assert.assertEquals;
4
5    import org.junit.Before;
6    import org.junit.Test;
7    import org.openmrs.PatientIdentifierType;
8
9    import org.openmrs.test.BaseContextSensitiveTest;
10
11   import java.util.List;
12
13   public class test_User3 extends BaseContextSensitiveTest{
14           private HibernatePatientDAO hibernatePatientDao;
15
16           @Before
17           public void beforeEach() {
18                   updateSearchIndex();
19                   hibernatePatientDao = (HibernatePatientDAO) applicationContext.getBean("patientDAO");
20           }
21
22           @Test
23       public void test() {
24                   // Get the id and the format of all PatientIdentifierType objects
25                   List<PatientIdentifierType> list = hibernatePatientDao.getAllPatientIdentifierTypes(true);
26                   for (int i = 0; i < list.size(); i ++) {
27                           System.out.println(list.get(i).getId());
28                           System.out.println(list.get(i).getFormat());
29                   }
30
31                   // Save two new PatientIdentifierType objects with different format
32                   PatientIdentifierType type1 = new PatientIdentifierType();
33                   type1.setName("type1");
34                   type1.setFormat("format 1");
35                   PatientIdentifierType type2 = new PatientIdentifierType();
36                   type2.setName("type2");
37                   type2.setFormat("format 2");
38                   hibernatePatientDao.savePatientIdentifierType(type1);
39                   hibernatePatientDao.savePatientIdentifierType(type2);
40
41           // The output should be 6 since there are 4 objects already exists in the database
42                   assertEquals(6, hibernatePatientDao.getAllPatientIdentifierTypes(true).size());
43           }
44
45   }
```

Figure 3.3: test_User3.java

- test_User4.java

  When the user creates and saves two PatientIdentifierType objects into the database, both with the unique id but same format = "format 1". Then if the user wants to get a list of all PatientIdentifierType objects in the database. As a result, the list should now contain only one PatientIdentifierType object he saved.

```java
3    import static org.junit.Assert.assertEquals;
4
5    import org.junit.Before;
6    import org.junit.Test;
7    import org.openmrs.PatientIdentifierType;
8
9    import org.openmrs.test.BaseContextSensitiveTest;
10
11   import java.util.List;
12
13   public class test_User4 extends BaseContextSensitiveTest{
14           private HibernatePatientDAO hibernatePatientDao;
15
16           @Before
17           public void beforeEach() {
18                   updateSearchIndex();
19                   hibernatePatientDao = (HibernatePatientDAO) applicationContext.getBean("patientDAO");
20           }
21
22           @Test
23       public void test() {
24                   // Get the id and the format of all PatientIdentifierType objects
25                   List<PatientIdentifierType> list = hibernatePatientDao.getAllPatientIdentifierTypes(true);
26                   for (int i = 0; i < list.size(); i ++) {
27                           System.out.println(list.get(i).getId());
28                           System.out.println(list.get(i).getFormat());
29                   }
30
31                   // Save two new PatientIdentifierType objects with same format
32                   PatientIdentifierType type1 = new PatientIdentifierType();
33                   type1.setName("type1");
34                   type1.setFormat("format 1");
35                   PatientIdentifierType type2 = new PatientIdentifierType();
36                   type2.setName("type2");
37                   type2.setFormat("format 1");
38                   hibernatePatientDao.savePatientIdentifierType(type1);
39                   hibernatePatientDao.savePatientIdentifierType(type2);
40
41          // The output should be 5 since there are 4 objects already exists in the database, and only one new object saved into database
42                   assertEquals(5, hibernatePatientDao.getAllPatientIdentifierTypes(true).size());
43           }
44
45   }
```

Figure 3.4: test_User4.java

0.4 Unit Test

As we implement the feature by adding and modifying functions inside two java files, we also need two test files to do the unit tests. All following unit test files are under the unit_tests folder:

https://github.com/CSCD01/team_21-project/tree/master/Deliverable4/unit_test/api/src/test/java/org/openmrs

- PaitentIdentifierTypeTest.java:
  This file is used to test methods in PaitentIdentifierType.java

```java
1   package org.openmrs;
2
3   import static org.junit.Assert.assertEquals;
4   import static org.junit.Assert.assertFalse;
5   import static org.junit.Assert.assertTrue;
6
7   import org.junit.Test;
8
9   public class PatientIdentifierTypeTest {
10
11      @Test
12      public void testEmpty() {
13          PatientIdentifierType patientIdentifierTypeOne = new PatientIdentifierType();
14          PatientIdentifierType patientIdentifierTypeTwo = new PatientIdentifierType();
15
16          assertTrue(patientIdentifierTypeOne.isDuplicate(patientIdentifierTypeTwo));
17      }
18
19      @Test
20      public void testNoDuplication() {
21          PatientIdentifierType patientIdentifierType = new PatientIdentifierType(1);
22          patientIdentifierType.setFormat("format1");
23          patientIdentifierType.setFormatDescription("formatDescription1");
24          PatientIdentifierType patientIdentifierType2 = new PatientIdentifierType(2);
25          patientIdentifierType2.setFormat("format2");
26          patientIdentifierType2.setFormatDescription("formatDescription2");
27
28          assertFalse(patientIdentifierType.isDuplicate(patientIdentifierType2));
29      }
```

Figure 4.1: PaitentIdentifierTypeTest.java (part 1)

```java
31      @Test
32      public void testIdDuplication() {
33          PatientIdentifierType patientIdentifierType = new PatientIdentifierType(1);
34          patientIdentifierType.setFormat("format1");
35          patientIdentifierType.setFormatDescription("formatDescription1");
36          PatientIdentifierType patientIdentifierType2 = new PatientIdentifierType(1);
37          patientIdentifierType2.setFormat("format2");
38          patientIdentifierType2.setFormatDescription("formatDescription2");
39
40          assertTrue(patientIdentifierType.isDuplicate(patientIdentifierType2));
41
42      }
43
44      @Test
45      public void testFormatDuplication() {
46          PatientIdentifierType patientIdentifierType = new PatientIdentifierType(1);
47          patientIdentifierType.setFormat("format");
48          patientIdentifierType.setFormatDescription("formatDescription1");
49          PatientIdentifierType patientIdentifierType2 = new PatientIdentifierType(2);
50          patientIdentifierType2.setFormat("format");
51          patientIdentifierType2.setFormatDescription("formatDescription2");
52
53          assertTrue(patientIdentifierType.isDuplicate(patientIdentifierType2));
54
55      }
56
57      @Test
58      public void testFormatDescDuplication() {
59          PatientIdentifierType patientIdentifierType = new PatientIdentifierType(1);
60          patientIdentifierType.setFormat("format1");
61          patientIdentifierType.setFormatDescription("formatDescription");
62          PatientIdentifierType patientIdentifierType2 = new PatientIdentifierType(2);
63          patientIdentifierType2.setFormat("format2");
64          patientIdentifierType2.setFormatDescription("formatDescription");
65
66          assertTrue(patientIdentifierType.isDuplicate(patientIdentifierType2));
67
68      }
```

Figure 4.2: PaitentIdentifierTypeTest.java (part 2)

```java
70      @Test
71      public void testMergeFormat() {
72          PatientIdentifierType patientIdentifierType = new PatientIdentifierType(1);
73          PatientIdentifierType patientIdentifierType2 = new PatientIdentifierType(1);
74          patientIdentifierType2.setFormat("format2");
75
76          assertTrue(patientIdentifierType.isDuplicate(patientIdentifierType2));
77
78          patientIdentifierType.merge(patientIdentifierType2);
79          assertEquals("format2", patientIdentifierType.getFormat());
80
81      }
82
83      @Test
84      public void testMergeFormatDesc() {
85          PatientIdentifierType patientIdentifierType = new PatientIdentifierType();
86          PatientIdentifierType patientIdentifierType2 = new PatientIdentifierType();
87          patientIdentifierType2.setFormatDescription("formatDescription2");
88
89          assertTrue(patientIdentifierType.isDuplicate(patientIdentifierType2));
90          patientIdentifierType.merge(patientIdentifierType2);
91
92          assertEquals("formatDescription2", patientIdentifierType.getFormatDescription());
93      }
94
95      @Test
96      public void testShortCircut() {
97          PatientIdentifierType patientIdentifierType = new PatientIdentifierType();
98          patientIdentifierType.setFormat("format1");
99
00          PatientIdentifierType patientIdentifierType2 = new PatientIdentifierType();
01          patientIdentifierType2.setFormat("format2");
02
03          assertEquals("format1", patientIdentifierType.getFormat());
04      }
```

Figure 4.3: PaitentIdentifierTypeTest.java (part 3)

- HibernatePatientDAOTest.java:

  This file is used to test methods in HibernatePatientDAO.java.

```java
12  import static java.util.Collections.emptyList;
13  import static java.util.Collections.singletonList;
14  import static org.hamcrest.Matchers.hasItems;
15  import static org.junit.Assert.assertFalse;
16  import static org.junit.Assert.assertNotNull;
17  import static org.junit.Assert.assertNull;
18  import static org.junit.Assert.assertTrue;
19
20  import java.util.Arrays;
21  import java.util.List;
22  import java.util.stream.Collectors;
23
24  import org.junit.Assert;
25  import org.junit.Before;
26  import org.junit.Test;
27  import org.openmrs.Patient;
28  import org.openmrs.PatientIdentifier;
29  import org.openmrs.PatientIdentifierType;
30  import org.openmrs.test.BaseContextSensitiveTest;
31
32  public class HibernatePatientDAOTest extends BaseContextSensitiveTest {
33
34          private HibernatePatientDAO hibernatePatientDao;
35
36          @Before
37          public void beforeEach() {
38                  updateSearchIndex();
39                  hibernatePatientDao = (HibernatePatientDAO) applicationContext.getBean("patientDAO");
40          }
41
42          @Test
43          public void getPatientIdentifiers_shouldGetByIdentifierType() {
44                  List<PatientIdentifierType> identifierTypes = singletonList(new PatientIdentifierType(2));
45                  List<PatientIdentifier> identifiers = hibernatePatientDao.getPatientIdentifiers(null, identifierTypes,
46                                  emptyList(), emptyList(), null);
47                  List<Integer> identifierIds = identifiers.stream().map(PatientIdentifier::getId).collect(Collectors.toList());
48
49                  Assert.assertEquals(2, identifiers.size());
50                  Assert.assertThat(identifierIds, hasItems(1, 3));
51          }
52
```

Figure 4.4: HibernatePatientDAOTest.java (part 1)

```java
53          @Test
54          public void getPatientIdentifiers_shouldGetByPatients() {
55                  List<Patient> patients = Arrays.asList(hibernatePatientDao.getPatient(6), hibernatePatientDao.getPatient(7));
56                  List<PatientIdentifier> identifiers = hibernatePatientDao.getPatientIdentifiers(null, emptyList(), emptyList(),
57                              patients, null);
58                  List<Integer> identifierIds = identifiers.stream().map(PatientIdentifier::getId).collect(Collectors.toList());
59
60                  Assert.assertEquals(2, identifiers.size());
61                  Assert.assertThat(identifierIds, hasItems(3, 4));
62          }
63
64          @Test
65          public void checkExistedType() {
66                  PatientIdentifierType patientIdentifierType = hibernatePatientDao.getPatientIdentifierType(1);
67                  assertTrue(hibernatePatientDao.hasDuplicate(patientIdentifierType).isPresent());
68          }
69
70          @Test
71          public void checkAnotherExistedType() {
72                  PatientIdentifierType patientIdentifierType = hibernatePatientDao.getPatientIdentifierType(2);
73                  assertTrue(hibernatePatientDao.hasDuplicate(patientIdentifierType).isPresent());
74          }
75
76          @Test
77          public void checkNonExistentType() {
78                  assertFalse(hibernatePatientDao.hasDuplicate(new PatientIdentifierType(114514)).isPresent());
79          }
80
81          @Test
82          public void addAndCheckType() {
83                  PatientIdentifierType patientIdentifierType = new PatientIdentifierType();
84                  patientIdentifierType.setName("test");
85                  patientIdentifierType.setDescription("description");
86                  patientIdentifierType.setRequired(false);
87
88                  assertNull(patientIdentifierType.getPatientIdentifierTypeId());
89
90                  hibernatePatientDao.savePatientIdentifierType(patientIdentifierType);
91                  assertNotNull(hibernatePatientDao.getPatientIdentifierType(patientIdentifierType.getPatientIdentifierTypeId()));
92
93                  assertTrue(hibernatePatientDao.hasDuplicate(patientIdentifierType).isPresent());
94          }
```

Figure 4.5: HibernatePatientDAOTest.java (part 2)

```
96          @Test
97          public void addAndCheckTypeOfSameFormat() {
98                  PatientIdentifierType patientIdentifierTypeOne = new PatientIdentifierType();
99                  patientIdentifierTypeOne.setName("name1");
100                 patientIdentifierTypeOne.setDescription("description1");
101                 patientIdentifierTypeOne.setRequired(false);
102                 patientIdentifierTypeOne.setFormat("format");
103
104                 PatientIdentifierType patientIdentifierTypeTwo = new PatientIdentifierType();
105                 patientIdentifierTypeTwo.setName("name2");
106                 patientIdentifierTypeTwo.setDescription("description2");
107                 patientIdentifierTypeTwo.setRequired(false);
108                 patientIdentifierTypeTwo.setFormat("format");
109
110                 hibernatePatientDao.savePatientIdentifierType(patientIdentifierTypeOne);
111
112                 assertTrue(hibernatePatientDao.hasDuplicate(patientIdentifierTypeTwo).isPresent());
113         }
114 }
```

Figure 4.6: HibernatePatientDAOTest.java (part 3)

The final version unit tests have some changes with the unit test we designed back in deliverable 3, we add more cases and write better comments in the test files. We mean to make each case easy to understand for any user.

0.5 Project Management

In this section, we will record how Team 21 manages the project in deliverable 4.

0.5.1 Kanban

Team 21 is continuing to use Kanban as our development process. And we use Trello to support our project management, our Trello board address is here: https://trello.com/b/pwP6Xoiq/cscd01-project

In this final deliverable, we are going to implement the feature we choose from deliverable 3. As we follow the implementation plan, two of our members are going to implement and change the structure of the PaientIdenifierType.java (also writing a complete unit test suite), the other two will implement the changes inside HibernatePaientDAO.java (also write a complete unit test suite), the last member will create the user guide and an acceptance test suite. Finally, the whole team will finish the deliverable 4 reports together.
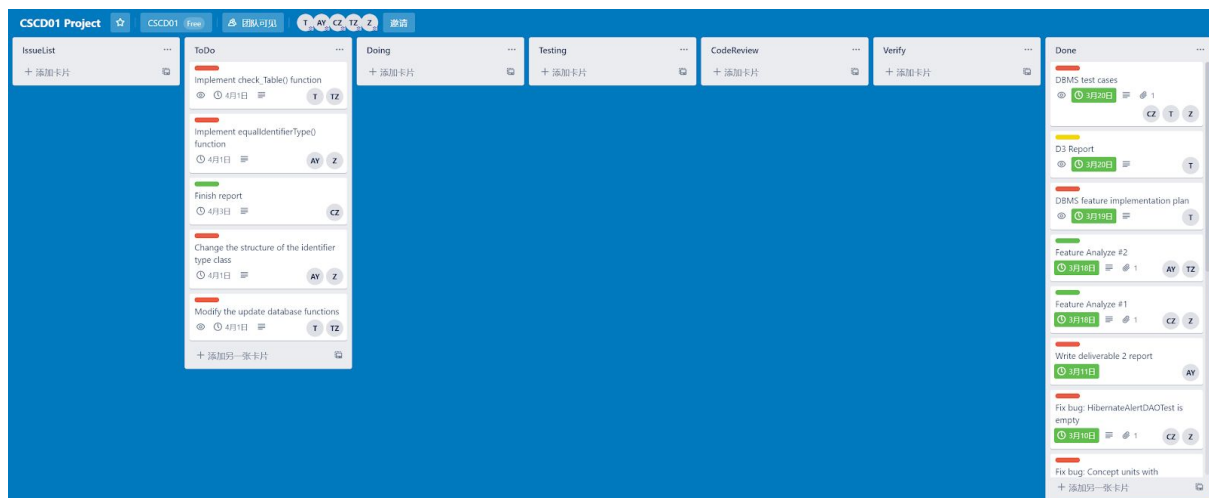

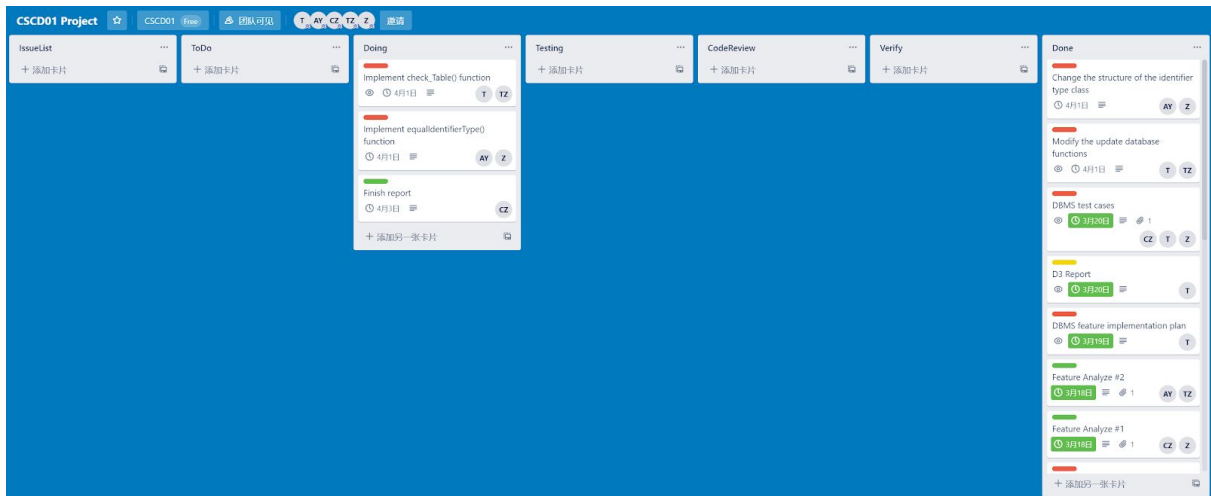*Figure 5.1: Kanban board (at the start of deliverable 4)*

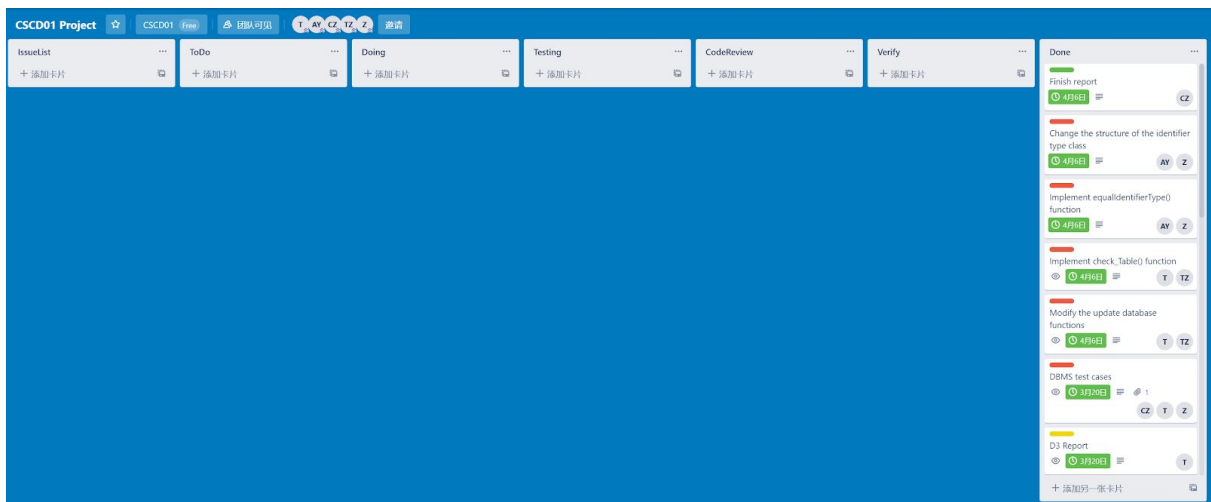*Figure 5.2: Kanban board (after the second meeting)*

Figure 5.3: Kanban board (at the end of the deliverable 4)

0.5.2 Standups

As part of the Kanban process, we are required to do daily standups. However, since we are students and our schedules are not very flexible, we settled on conducting 4 meetings during the work period of deliverable 4.

Mar. 28th: We got everyone up to speed and reviewed the implementation plan from the deliverable 3. Then we decided to split our team into two groups of two working on two main parts of implementation while the leftover one writes the user guide and an acceptance test suite.

Mar. 30th: We had a meeting to catch up on how everyone was doing, and we talked about changes/differences from the original implementation plan.

April. 1st: We test the feature and make sure everything is fine. After that, we talked about adding more test cases.

(Add one, after Dr.Anya extended the due date)
April. 5th: We reviewed all of our work to make sure everything works fine. Then we check and fix the typo and grammar mistakes for the deliverable report before we make the final submission.

0.5.3 Evidence Of Kanban

As we described in earlier deliverables, Kanban is meant to maximize the speed and quality of output, which is suitable for large size program development. And from the two months of development experience with Kanban, our team very much benefited from it.

One thing that makes Kanban so good at large scale program development is that compared to other software development processes, the Kanban process is more flexible since it does not require software engineers to have exactly the same skill level, it allows the developer team to add tasks anytime during the work period. Since every member of our team has no experience in development in OpenMRS, we all practically learn new stuff (ex. OpenMRS structure, API structure) during development, thus it is very common that some tasks need to be changed or added during the process. Kanban's flexibility is helping us with such problems, every time our team decides to change something we just change it without a thought, we don't need to follow an unchangeable task till the end of a sprint like other development processes do, the flexibility saves us a bunch of time.

Another feature that made Kanban unique is limit work in progress (WIP), by limiting the number of tasks it will improve our focus on a few tasks and to identify bottlenecks. Our team has set the number of WIP to be five during the whole development period. It ensures that each member of the team has only one task to

do, such that the quality of our work will remain high and the workload of each person is very likely equal.

Overall, Kanban helps our team a lot during the work term, and we are glad to choose Kanban as our development process so that we can finish the development together.