Report Title : Test Driven Development (TDD) for The Smart Living Community Project

Course Code: CSE 404

Course Title: Software Engineering and ISD Laboratory

Submitted by

SHANJIDA ALAM(ID: 353)

Submitted to

Dr. Md. MUSHFIQUE ANWAR, Professor

Dr. Md. HUMAYUN KABIR, Professor

Computer Science and Engineering

Jahangirnagar University

Dhaka, Bangladesh

January 07, 2025

Contents

Chapter 1

Introduction

Test-Driven Development (TDD) is an essential agile practice that helps development teams clarify and understand project requirements, especially in the early stages. By combining TDD with the right tools and processes, teams can create comprehensive test suites, enhance software quality, and support **Continuous Integration (CI)** workflows. This report highlights my personal contribution to implementing TDD in our project starting from Sprint 2 of the development phase. It also covers the tools we used, the challenges we encountered, and the lessons we gained along the way.

This report focuses on my individual role in incorporating TDD into our project during the development phase, starting with Sprint 2. It details my efforts in writing test cases before developing features, refining requirements through iterative feedback, and collaborating with the team to integrate TDD workflows effectively. Additionally, it highlights the tools we employed to enhance the process, the problems we encountered while implementing TDD, and the insights we gained, which have shaped our approach to development in subsequent sprints.

Chapter 2

JUnit for Test Driven Development(TDD)

2.1 What is Junit?

JUnit is a widely-used framework for unit testing Java applications. It provides annotations, assertions, and methods to test individual components of the code, ensuring they work as expected. With JUnit, developers can isolate and test each module independently, which helps catch bugs early and improve code quality.

2.2 Why Choose JUnit?

- **Reliability and Popularity:** JUnit is a stable and mature testing framework supported by a vast community. Its popularity in the Java ecosystem makes it a reliable choice with extensive documentation and resources available for troubleshooting.
- **Integration with Android Studio:** For our LivSmart project, which uses Android Studio and Java, JUnit integrates seamlessly, allowing us to write and run tests directly in the IDE.
- **Support for Test-Driven Development (TDD):** JUnit supports TDD principles, making it easier to write tests before or alongside the development of features.
- **Easy Assertion Library:** JUnit provides a simple assertion library that makes it easy to verify expected results. It simplifies test writing and helps maintain clear, readable test code.

- **Compatibility with Mockito:** JUnit pairs well with Mockito, a framework for mocking dependencies. Since we are also exploring Mockito, this makes JUnit an even better choice for unit testing.

Chapter 3

My Contribution to The TDD

3.1 Test Case Created

- Ensure `ComplaintModel` fields are validated.
- Verify that the complaint submission API returns a success response.
- Validate error handling for incomplete data.

3.2 Writing Tests Before Code

At first I started by writing failing tests for each feature. Then, incrementally wrote implementation code to make tests pass. And finally refactored the code while ensuring that tests remained green. Here, attach the test code:

```
1  @Test
2      public void submitComplaint_withValidData_shouldSucceed() {
3          /**
4           * Create a valid complaint test case
5           */
6          ComplaintModel complaint = new ComplaintModel(
7                   "A101", "John Doe",
8                  "Resident", "12345678905", "john@gmail.com", "Test
       complaint"
9          );
10
11 @Test
12     public void submitComplaint_withEmptyUnitCode_shouldFail() {
13         /**
14          * Create an invalid complaint test case
15          */
```

```
16          ComplaintModel complaint = new ComplaintModel();
17          /**
18           * Set the unit code to an empty string
19           */
20          complaint.setUnitCode("");
21          /**
22           * Set the other fields to valid values
23           */
24          complaint.setUserName("John Doe");
25          complaint.setUserRole("Resident");
26          complaint.setPhoneNumber("1234567890");
27          complaint.setEmailAddress("john@gmail.com");
28          complaint.setComplaintDescription("Test complaint");
29
30          // Execute
31          /**
32           * Call the submitComplaint method
33           */
34          LiveData<Boolean> result = viewModel.submitComplaint(
    complaint);
35
36          // Verify
37          /**
38           * Assert that the result is false
39           */
40          assertEquals(false, result.getValue());
41          verify(repository, never()).submitComplaint(any(
    ComplaintModel.class));
42      }
43
44      @Test
45      public void submitComplaint_withEmptyDescription_shouldFail() {
46          /**
47           * Create an invalid complaint test case
48           */
49          ComplaintModel complaint = new ComplaintModel(
50                  "A101", "John Doe",
51                  "Resident", "1234567890", "john@gmail.com", ""
52          );
53
54          /**
55           * Call the submitComplaint method
56           */
57          LiveData<Boolean> result = viewModel.submitComplaint(
    complaint);
58          /**
```

```
59          * Assert that the result is false
60          */
61         assertEquals(false, result.getValue());
62         /**
63          * Verify that the repository was not called
64          */
65         verify(repository, never()).submitComplaint(any(
    ComplaintModel.class));
66     }
67
68     @Test
69     public void submitComplaint_withInvalidPhoneNumber_shouldFail() {
70         /**
71          * Create an invalid complaint test case
72          */
73         ComplaintModel complaint = new ComplaintModel(
74                 "A101", "John Doe",
75                 "Resident", "123", "john@gmail.com", "Test complaint"
76         );
77
78         /**
79          * Call the submitComplaint method
80          */
81         LiveData<Boolean> result = viewModel.submitComplaint(
    complaint);
82
83         /**
84          * Assert that the result is false
85          */
86         assertEquals(false, result.getValue());
87         /**
88          * Verify that the repository was not called
89          */
90         verify(repository, never()).submitComplaint(any(
    ComplaintModel.class));
91     }
92
93     @Test
94     public void submitComplaint_withEmptyUserName_shouldFail() {
95         /**
96          * Create an invalid complaint test case
97          */
98         ComplaintModel complaint = new ComplaintModel(
99                  "A101", "",
100                 "Resident", "1234567890","john@gmail.com", "Test
    complaint"
```

```
101         );
102
103         /**
104          * Call the submitComplaint method
105          */
106         LiveData<Boolean> result = viewModel.submitComplaint(
    complaint);
107
108         /**
109          * Assert that the result is false
110          */
111         assertEquals(false, result.getValue());
112         /**
113          * Verify that the repository was not called
114          */
115         verify(repository, never()).submitComplaint(any(
    ComplaintModel.class));
116     }
117
118     @Test
119     public void submitComplaint_withEmptyUserRole_shouldFail() {
120         /**
121          * Create an invalid complaint test case
122          */
123         ComplaintModel complaint = new ComplaintModel(
124                 "A101", "John Doe",
125                 "", "1234567890", "john@gmail.com","Test complaint"
126         );
127
128         /**
129          * Call the submitComplaint method
130          */
131         LiveData<Boolean> result = viewModel.submitComplaint(
    complaint);
132
133         /**
134          * Assert that the result is false
135          */
136         assertEquals(false, result.getValue());
137         /**
138          * Verify that the repository was not called
139          */
140         verify(repository, never()).submitComplaint(any(
    ComplaintModel.class));
141     }
142
```

```
143 @Test
144     public void submitComplaint_withNullComplaint_shouldFail() {
145         /**
146          * Call the submitComplaint method
147          */
148         LiveData<Boolean> result = viewModel.submitComplaint(null);
149         /**
150          * Assert that the result is false
151          */
152         assertEquals(false, result.getValue());
153         /**
154          * Verify that the repository was not called
155          */
156         verify(repository, never()).submitComplaint(any());
157     }
158 
159      @Test
160     public void submitComplaint_withEmptyEmailAddress_shouldFail() {
161         /**
162          * Create an invalid complaint test case
163          */
164         ComplaintModel complaint = new ComplaintModel(
165                 "A101", "John Doe",
166                 "Resident", "1234567890","", "Test complaint"
167         );
168         complaint.setEmailAddress("");
169 
170         /**
171          * Call the submitComplaint method
172          */
173         LiveData<Boolean> result = viewModel.submitComplaint(
    complaint);
174         /**
175          * Assert that the result is false
176          */
177         assertEquals(false, result.getValue());
178     }
179 
180      @Test
181     public void submitComplaint_withInvalidEmailFormat_shouldFail() {
182         /**
183          * Create an invalid complaint test case
184          */
185         ComplaintModel complaint = new ComplaintModel(
186                 "A101", "John Doe",
```

```
187                 "Resident", "1234567890", "johnmail.com", "Test
    complaint"
188         );
189         complaint.setEmailAddress("invalid.email");
190
191         /**
192          * Call the submitComplaint method
193          */
194         LiveData<Boolean> result = viewModel.submitComplaint(
    complaint);
195         /**
196          * Assert that the result is false
197          */
198         as
```

Listing 3.1: ComplaintViewModelTest Class in Java

3.3 Visual Aspect of Passing Test

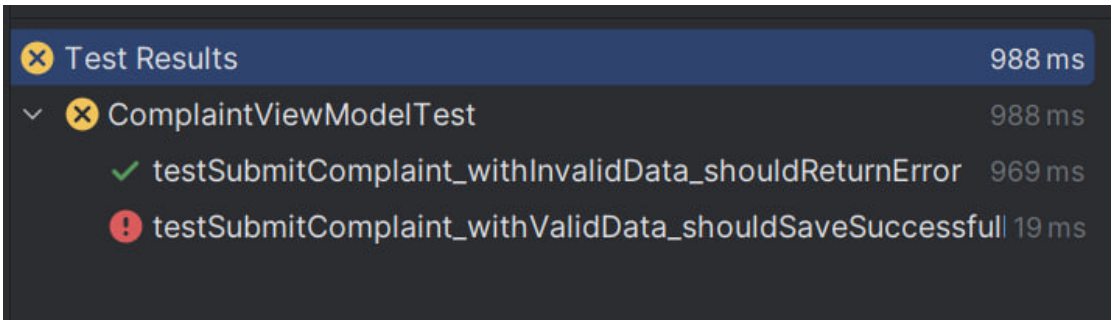

Figure 3.1: This image shows that initially InvalidData pass the test case and ValidData does not pass the test case. And write the code for passing this testcase.

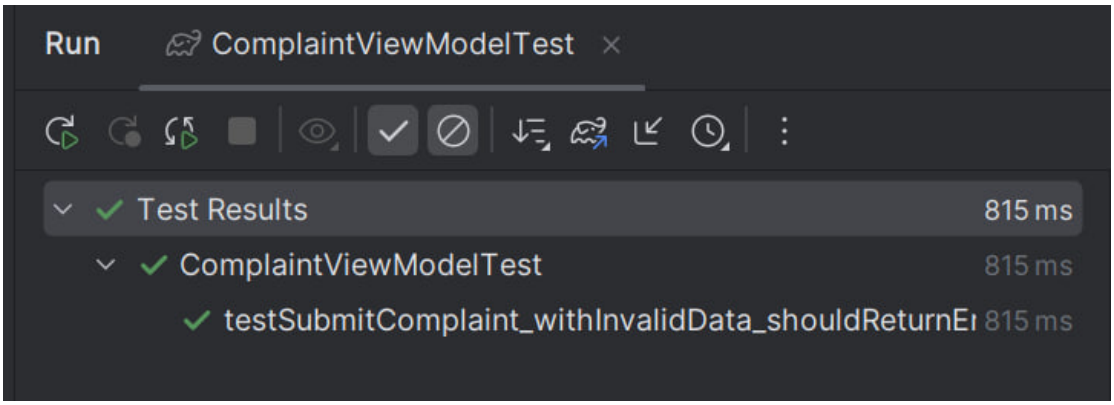

Figure 3.2: This image shows that pass the test case with the Valid Data input.

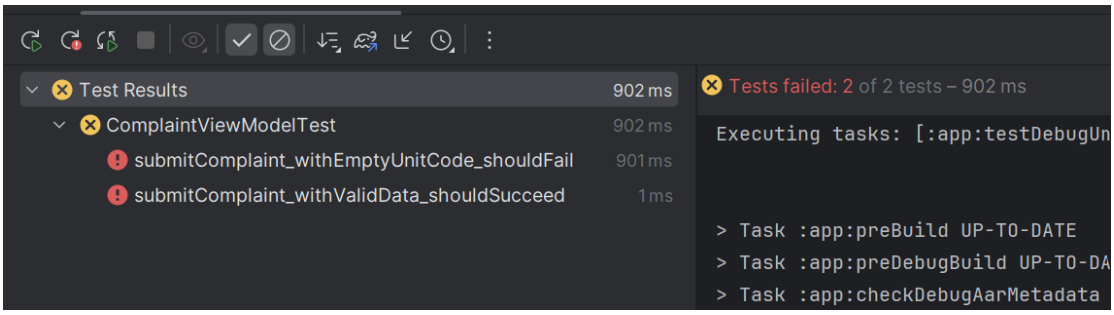

Figure 3.3: This image shows that the empty input test case does not pass in this part.

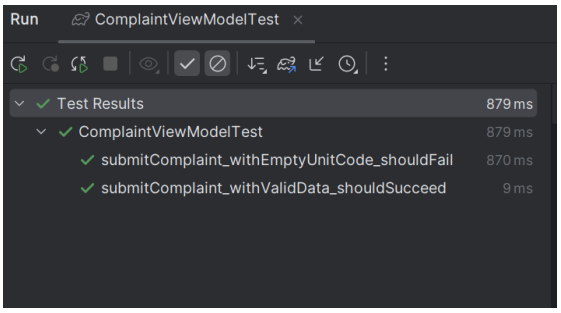

Figure 3.4: This image shows that the empty input test case pass in this part.

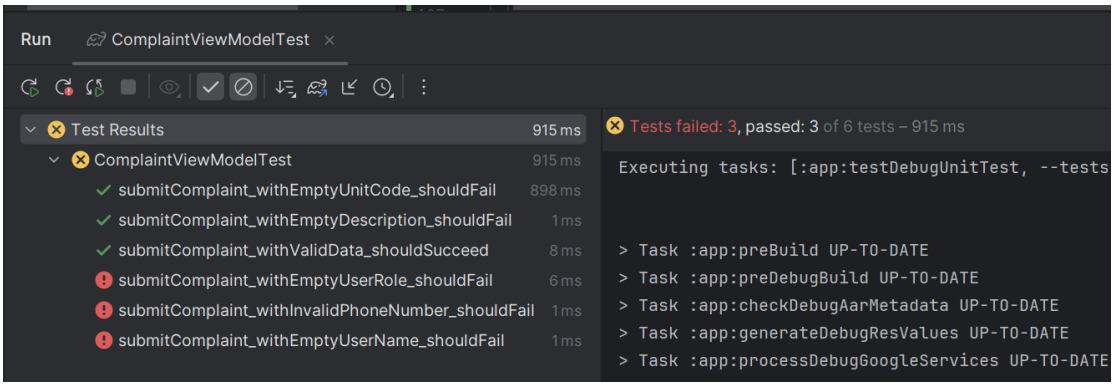

Figure 3.5: This image shows that the empty unitCode pass but emptyUserRole test case does not pass.

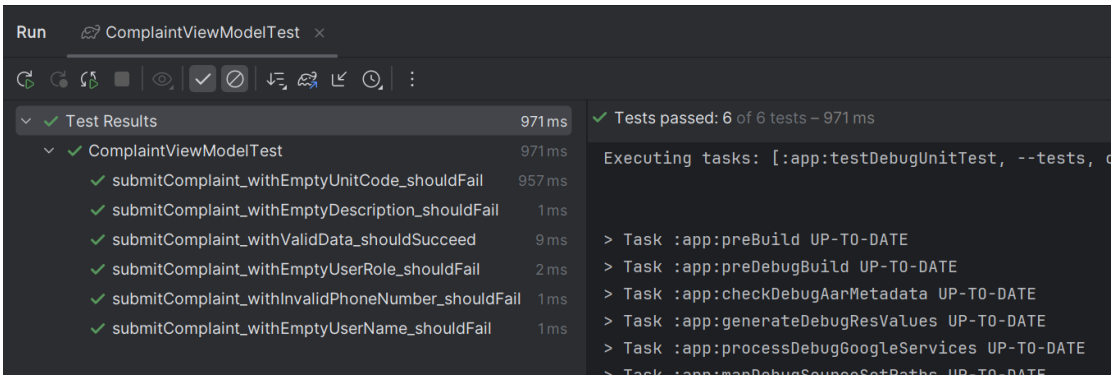

Figure 3.6: All possible test case should be passed in this part.

Chapter 4

Conclusion

Adopting TDD was a best experience for both me and the team. It not only improved our understanding of requirements but also enhanced the quality of our code and processes. By integrating TDD with CI, we ensured that our project maintained high standards of quality and reliability. I look forward to further refining this practice in future projects.