****

Project Report

# Project: Software Testing on “Pill Reminder”, an Android Application

**Course**: CSE427

**Course Title**: Software Testing and Quality Assurance

**Section**: 01

**Semester**: Spring 2019

**Submitted to**: Shaikh Shawon Arefin Shimon

**Submitted by**: Group 06

1. Noshin Islam 1521733642

2. Asik Azad 1510612042

**Github Link**: https://github.com/nsuspring2019cse427/Group06

**1. Introduction**

In our developing and technology dependent life we totally rely on gadgets especially smart phones. Today everyone has a smart phone. With this we get an opportunity to use technology in a better way so that it can be made useful to us. And it plays an important part in our daily life and helps us staying fit in many ways. The remarkable problem is that patients forget to take the proper medicines in proper proportion and in proper time. Medication adherence, which refers to the degree or extent to which a patient takes the right medication at the right time according to a doctor’s prescription, has recently emerged as a serious issue because many studies have reported that non-adherence may critically affect the patient, thereby raising medical costs

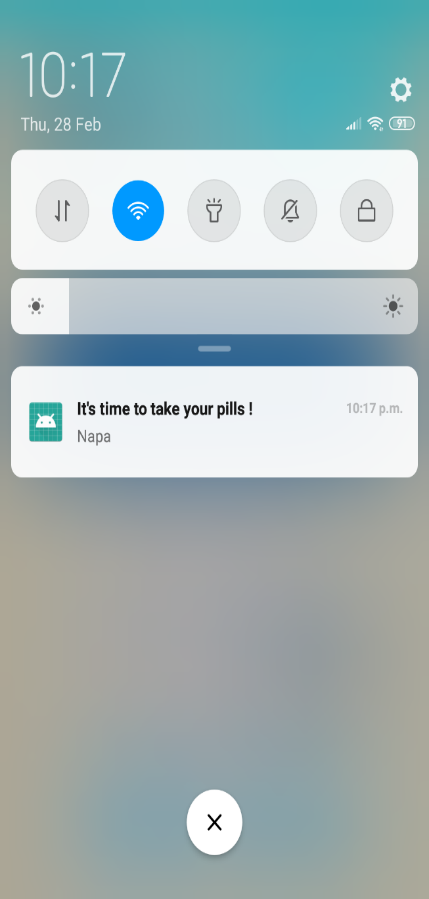
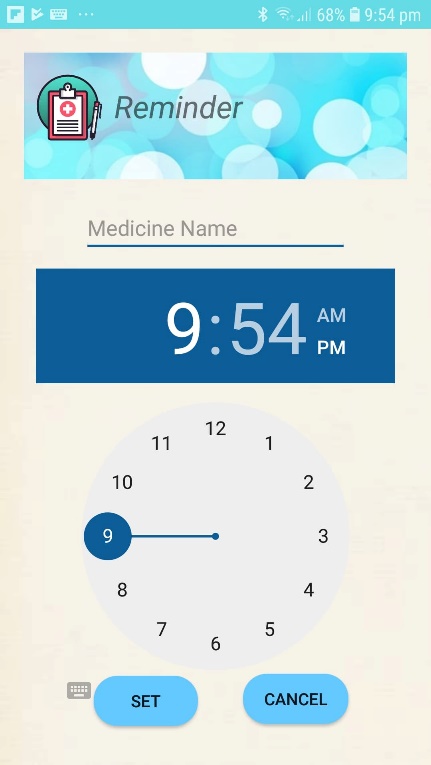
Taking medication is occasionally frustrating, but ultimately necessary. Thankfully, smartphones can make the experience a little easier. The mobile application that we have decided to test the functionality off is named “Pill Reminder, an app that can be used to decrease the stress of remembering to take the pills on a predefined scheduled time. Staying healthy and getting your medications exactly when your body needs them is vital, but sometimes you just forget.

**2. App Functionality:**

1. Easy and understandable User Interface

2. No authentication required

3. Add Medicine names and set a specified time to get the alert

4. A pop-up message is shown when it’s reminder time

*Fig: Medicine reminder time set up (left) Pop up message (right)*

**3. Software Testing & Tools:**

As our application is moslty UI based and doesn’t require any internal calculation for functionality execution. We tested upon different UI componenets and UI flexibility. We checked if all the UI componenets are working together perfectly when using the app. We tested the componenets one by one.

The functionality we tested are: input a medicine name, specify the exact time by using the Timepicker, set the time, set the medicine name, check for invalid medicine name, check for invalid timeChecks.

We used an android app which was obviously developed in Java and thus during testing we used:

1. Language: JAVA

2. App design and Implementation IDE: Android Studio

3. Testing: Junit 5

4. Framework: Espresso

Espresso tests run optimally fast! It lets you leave your waits, syncs, sleeps, and polls behind while it manipulates and asserts on the application UI when it is at rest

**Test cases:**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Status | Step | Expected Result |
| @Before Test  ( Reminder Class ) | Successful | 1. First we create a Test rule of “mainactivity” type. We make an instance of the rule called “mActivityTestRule”  2. By using the @Before annotation, we can test if the required activity can be invoked by the created rule, which is preliminary. In the @Before method we try to invoke the Reminder class activity which was a success. | The activity rule instance should be able to invoke the main activity fille which is “ReminderActivity”. Runs once before any of the test methods in the class |
| @After Test | Successful | It runs after all the test cases of the class. As we made an activity instance with the rule instance we have to make that class instance null after we are done with it.  We use the **TearDown** method to do this | If we allocate external resources in a Before method you need to release them after the test runs. |
| Test Case 1 | Succesful | First test is to test I if the Text view of where patients will enter their message to be invoked is successfully launched or not. The name of the method is **testLaunchActivity\_TextView** ,  First **View** class instance is created to find the necessary id name of the field. | Text View should be successfully shown on the screen |
| Test Case 2 | Successful | Second test is to see if the Cancel button that cancels the written text is working or being viewed or not. The method being tested here is **testLaunchActivity\_cancelbuttonView** | Cancel Button should be sucessfully shown on the app |
| Test Case 3 | Successful | Third test is to see if the Setbutton view is working or not. **testLaunchActivity\_setbuttonView** | Set button should be shown and work |
| Test Case 4 | Successful | There is a timepicker in the app, by which patients can select time precisely. To see if the timepicker is shown: **testLaunchActivity\_timepickerView** | Timepicker should be shown on the main activity view and should be able to navigate properly |
| Test Case 5 | Successful | To test if the medicine name view where the medicine nme will be put is invoked or not we tested **testLaunchActivity\_medicinenameView** | Medicine name text field should be shown |
| Test Case 6 | Successful | To test if the iconreminder view is shown or not this is tested testLaunchActivity\_iconreminderView | Icon reminder should be successfully shown |
| Test Case 7 | Successful | To test if the layoutreminder view is shown or not this is used: testLaunchActivity\_layoutreminderView | Reminder layout should be shown |
| Test Case 8 | Successful | We test the Launch of set button. The activities that needs to be checked are: setbutton view is not null, setbutton should be able to perfom if a click happens, Intent class instances can be made to connect the reminder class and alarm class. All these are tested by the method: **testLaunchOfSetButtonActivity** | Set button should meet all the requirements to have a successful click operation. |
| Test Case 9 | Successful | The test method: **testonClick** It tests the View class instance and its workability, It tests the Timepicker actitivity. It also tests the getTimeInMillis method | Timepicker should be able to pick the time ( hour, minute) in miliseconds. |
| Test Case 10 | Successful | We test the **getTimeInMillis()** separately because it is crucial that we get the hour and minute by using the Calendar class instance | Time should be get is Hour, Minute format and converted to miliseconds. |
| Test Case 11 (Alarm Class) | Successful | 1. @Before We make an ActivityRule for Alarm class and make an instance of to try to test if the Alarm class activity can be invoked or not. We make an instance called alarmActivity of Reminder class.  2. @After reallocates the used resources after their work is finished | Activity Test rule for Alarm class is needed to work if the Intent is created. |
| Test Case 12 (Espresso Input Space Partioning ) | Successful | **testMedicineNameInput(),** **testMedicineNameInput2(), testMedicineNameInput3()** uses Espresso framework to test input type of medicinename field to be only alphabets | To allow users to only input valid medicine names |
| Test Case 13 | Successful | **testMedicineNameInput4InvalidInput()** tests the medicinename field for invalid input types. i.e. 123napa | Patients need to be check before entering the medicine name |
| Test Case 14 | Successful | **improperTimeFalseCase()** invokestocheck the reminder time if it is the right time to set the alarm. | Proper time should be between 5 am to 11 pm. |
| Test Case 15 | Successful | **improperTimeTrueCase()** invokestocheck the reminder time if it is the wrong time to set the alarm. | Improper time should be after 11pm to 5 am. |

**5. Input Space Partitioning:**

Input space partitioning for medicine name input:

We used Espresso framework to set different input texts for testing valid/invalid input. We automated our UI testing as well as did our integration testing by it.

We first changed our layout design for medicine name field. We specified the inputType as textPersonName(only letters), and also specified the digits(a-z,A-Z) to make sure no whitespace character or number can be entered here.

We exampled four inpur types in four different test cases which resides in ReminderTest class, named: testMedicineNameInput(),: testMedicineNameInput2(), testMedicineNameInput3(), testMedicineNameInput4InvalidInput(), where we use the onView method to select medicine name field, perform typeText funtion on it to set the text input and use checkes method to check if what is being put in the field actually matched the input being saved.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Partition** | **B1** | **B2** | **B3** | **B4** |
| q1 = Medicine Name Input text type | All capital letters i.e. NAPA | ALL small letters  i.e. napa | Small and Captail mixed letters i.e. NaPa | Invalid Input i.e. 12napa, 12\_napa, #napa |

Input Space partitioning for TimeCheck:

|  |  |  |
| --- | --- | --- |
| **Function** | **True case** | **False case** |
| improperTime() | hour > 23 or hour < 5 | 5 <= hour <= 23 |

**Graph Partioning:**

Medicine-Name Input type testing

Medicine Name

testMedicineNameInput4InvalidInput() //123napa

testMedicineNameInput() //Napa

testMedicineNameInput3() //NAPA

testMedicineNameInput2() //napa

testCase failed.

testCase Passed.

Fig: Graph for Medicine Name text testing

Alarm setting ImproperTimeCheck:

**improperTime():**

**! improperTime()**

int hour = 20;

assertEquals(false,result)

**improperTime()**

int hour = 23;

assertEquals(true,result)

**Return false**

**Return true**

Fig: Graph for Improper Time-Check