**ROADIES TEST PLAN**

The purpose of this testing plan is to document on both functional testing and user testing; furthermore, functional testing intended to determine the strategy that has been used to test on each function that run on the application and user testing will be able to gather feedback on the usability of the application interface.

**Functional Testing Plan**

Unit test will be conducted to ensure that each individual function in the application is running stable and predictable; the following are the functions that has been tested on:

**Mobile Application (Function-based test descriptions)**

**Function:**

Login function

**Description:**

Test on the input value of username and password to see is the value are valid.

**Strategy:**

An assumption had been made that the input value of username will be set “name” and password “pwd” and the emulator will run through the process to check does the input value equals to the expect value.

**Input:**

sendKeys(KeyEvent.*KEYCODE\_N*, KeyEvent.*KEYCODE\_A*, KeyEvent.*KEYCODE\_M*,

KeyEvent.*KEYCODE\_E*);

sendKeys(KeyEvent.*KEYCODE\_P*, KeyEvent.*KEYCODE\_W*, KeyEvent.*KEYCODE\_D*);

**Output:**

*assertEquals*("name", user.getUserName())

*assertEquals*("pwd", user.getPassword())

**Function:**

Dashboard

**Description:**

Test on the Repair button to ensure it can direct user to the other activity.

**Strategy:**

Setup the “repairBtn” as a test image button and the condition will return a true value if the click action performs and be able to lead to the other activity.

**Input:**

On click event

**Output:**

True

**Function:**

User Test

**Description:**

To ensure that the get username(), set username(), get password() and set password() method are working properly.

**Strategy:**

Two instant of user had been set with different username and password; test will be conduct to ensure that the username get the expect password.

**Input:**

Two set of mock value for username and password.

**Output:**

True

**Function**

Problem1

**Description:**

To ensure that the item on list views match the store string value.

**Strategy:**

getChildCount() was one of the main component in the Item click method and was use to return the number of values in the group.

**Input:**

*assertEquals*(4, mActivity.getListView().getChildCount());

*assertEquals*("Brakeww", mActivity.getListView().getItemAtPosition(0));

*assertEquals*("Engineee", mActivity.getListView().getItemAtPosition(1));

*assertEquals*("Tyre", mActivity.getListView().getItemAtPosition(2));

*assertEquals*("Battery", mActivity.getListView().getItemAtPosition(3));

**Output:**

True.

**Function**

Problem2

**Description:**

Ensure that Problem2 Activity be able to retrieve the string value from Problem1 Activity.

**Strategy:**

Conduct a mock string array and try to store string array pass from Problem1 Activity and display on the problem2 Activity list view with the expecting value.

**Input:**

Mock string value

**Output:**

True

**Function**

RequestActivity Layout

**Description:**

To ensure that all the text view and button are valid.

**Strategy:**

Precondition will be setup to ensure that text view and button return the expect value.

**Input:**

assertNotNull(sendB);  
assertNotNull(msgV);

**Output:**

True

**Function**

RequestActivity Submit button

**Description:**

To ensure that the username, password and GPS value are the same as the other Activity Class

**Strategy:**

Mock values will be conduct for “Login Activity” “Problem2 Activity” “GPS Activity” and in the test class, the method will check to see if the mock values go through the expecting class and return with the expecting value, too.

**Input:**

Mock Values

**Output:**

True

**Function**

RequestActivity GPS

**Description:**

To ensure that the longitude, latitude value of the GPS is accurate for getlocation() method to use.

**Strategy:**

Mock values will be conduct for GPS’s longitude, latitude and in the test class, the method will check to see if the mock values are the same as the GPS Activity class expecting value.

**Input:**

Mock Values

**Output:**

True

**Mobile Application (Method-based test descriptions)**

**Class: LoginActivityTest**

**Methods:**

/\*\*

\* Setting up the elements for the LoginActivity.

\* Creating an instrumentation instance and a LoginActivity to perform the

\* test.

\*/

@Override

**protected** **void** setUp() **throws** Exception;

/\*\*

\* Make sure there are not any null value are created

\* Name input != null.

\* Password input != null.

\* Login button !=null.

\*/

**public** **void** testAPreconditions();

/\*\*

\* Test whether the input name on editview is expected input and

\* also the password editview.

\*/

**public** **void** testInput();

/\*\*

\* Test whether the name input is passed to the userName of User type.

\*/

**public** **void** testSetUserName();

/\*\*

\* Test whether the password input is passed to the password of User type.

\*/

**public** **void** testSetPassword();

/\*\*

\* Test whether the username and password are

\* saved into the savedPreferences

\*/

**public** **void** testWriteSharedPreferences();

/\*\*

\* Clean up after test.

\*/

@Override

**public** **void** tearDown();

/\*\*

\* This method mock the human input behaviours for the name and password

\* field by sending keyEvents.

\*

\*/

**public** **void** setUpInput().

**Class: DashBoardActivityTest**

**Methods:**

/\*\*

\* Seting up the test properties to run the test.

\*/

@Override

**public** **void** setUp() **throws** Exception;

/\*\*

\* Test preconditions.

\*/

**public** **void** testPreConditions();

/\*\*

\* Test the button "repair" to function properly.

\***@UithreadTest** make sure the test run on the UI thread.

\*/

@UiThreadTest

**public** **void** testToClick();

/\*\*

\* Clean up after the test

\*/

@Override

**public** **void** tearDown() **throws** Exception.

**Class: Problem1ActivityTest**

**Methods:**

/\*\*

\* set up the test activity

\*/

@Override

**public** **void** setUp();

/\*\*

\* Test the mActivity to be not null

\*/

**public** **void** testPrecondition();

/\*\*

\* return the numbers of value in the group

\*/

**public** **void** testItemNumber();

/\*\*

\* ensure the list view is equal equal to the expecting value

\*/

**public** **void** testGetItem();

/\*\*

\* Clean up after test.

\*/

@Override

**public** **void** tearDown() **throws** Exception.

**Class: Problem2ActivityTest**

**Methods:**

/\*\*

\* Setting up the basic properties of testing Problem2Acticity

\* Mock the string passed from Problem1Activity by Intent, it is saved in

\* bundle by putString() method.

\*/

@Override

**public** **void** setUp();

/\*\*

\* Test if the Problem2Activity is null activity

\*/

**public** **void** testPrecondition();

/\*\*

\* Test the number of items passed from the Problem1Activity match the

\* expected number.

\*/

**public** **void** testItemNumber();

/\*\*

\* Test whether each element of the value match the expected ones.

\*/

**public** **void** testGetItem();

/\*\*

\* Clear up after test, inherit from the super class

\*/

@Override

**public** **void** tearDown() **throws** Exception;

**Class: UserTest**

**Methods:**

/\*\*

\* Set up the case which need to be tested.

\* **@throws** Exception

\*/

@Before

**public** **void** setUp() **throws** Exception;

\* A test for testing getUserName() method and setUserName() method.

\* Test whether the getter method can get the values by a passing new

\* username.

\*/

@Test

**public** **void** testGetSetName();

/\*\*

\* A test for testing getPassword() method and setPassword() method.

\* Test whether the getter method can get the values by a passing new

\* password.

\*/

@Test

**public** **void** testGetSetPassword().

**Class: Request ActivityTestButton**

**Methods:**

@Override

/\*\*

\* Initialize activity

\*/

**protected** **void** setUp() **throws** Exception;

@Override

/\*\*

\* Clearing up actitivy using

\*/

**protected** **void** tearDown() **throws** Exception;

/\*\*

\* close activity

\* **@param** activity the target activity

\*/

**private** **void** closeActivity(Activity activity);

/\*\*

\* Start this activity to test

\*/

**public** **void** openActivity() **throws** Exception;

/\*\*

\* set problems

\*/

**public** **void** setTestProblems(Bundle bundle, String problem, String item);

/\*\*

\* set loctaion

\*/

**public** **void** setTestLocation();

/\*\*

\* Helper method to add a test provider with given name.

\*/

**private** **void** addTestProvider(**final** String providerName);

/\*\*

\* store the user information in the setSharedPrefereced

\*/

**public** **void** setSharedPrefereced() ;

**Class: Request ActivityTestGPS**

**Methods:**

@Override

/\*\*

\* Initialize activity

\*/

**protected** **void** setUp() **throws** Exception;

/\*\*

\* Clearing up actitivy using

\*/

**protected** **void** tearDown() **throws** Exception;

/\*\*

\* close activity

\* **@param** activity the target activity

\*/

**private** **void** closeActivity(Activity activity);

/\*\*

\* Start this activity to test

\*/

**public** **void** openActivity() **throws** Exception;

/\*\*

\* Perform Click

\*/

**public** **void** clickButton();

/\*\*

\* set problems

\*/

**public** **void** setTestProblems(Bundle bundle, String problem, String item);

/\*\*

\* set loctaion

\*/

**public** **void** setTestLocation();

/\*\*

\* Helper method to add a test provider with given name.

\*/

**private** **void** addTestProvider(**final** String providerName);

/\*\*

\* store the user information in the setSharedPrefereced

\*/

**public** **void** setSharedPrefereced

**Class: Request ActivityTestPrecondition**

**Methods:**

@Override

/\*\*

\* Initialize activity

\*/

**protected** **void** setUp() **throws** Exception;

@Override

/\*\*

\* Clearing up actitivy using

\*/

**protected** **void** tearDown() **throws** Exception ;

/\*\*

\* close activity

\* **@param** activity the target activity

\*/

**private** **void** closeActivity(Activity activity);

/\*\*

\* Start this activity to test

\*/

**public** **void** openActivity() **throws** Exception;

/\*\*

\* Test whether it is initialized correctly

\*/

**public** **void** testPreCondition();

/\*\*

\* set problems

\*/

**public** **void** setTestProblems(Bundle bundle, String problem, String item);

/\*\*

\* set loctaion

\*/

**public** **void** setTestLocation();

/\*\*

\* Helper method to add a test provider with given name.

\*/

**private** **void** addTestProvider(**final** String providerName);

**Class: Request ActivityTestPro**

**Methods:**

@Override

/\*\*

\* Initialize activity

\*/

**protected** **void** setUp() **throws** Exception;

@Override

/\*\*

\* Clearing up actitivy using

\*/

**protected** **void** tearDown() **throws** Exception;

/\*\*

\* close activity

\* **@param** activity the target activity

\*/

**private** **void** closeActivity(Activity activity);

/\*\*

\* Start this activity to test

\*/

**public** **void** openActivity() **throws** Exception;

/\*\*

\* Perform Click

\*/

**public** **void** clickButton();

/\*\*

\* set problems

\*/

**public** **void** setTestProblems(Bundle bundle, String problem, String item)

/\*\*

\* set loctaion

\*/

**public** **void** setTestLocation();

/\*\*

\* Helper method to add a test provider with given name.

\*/

**private** **void** addTestProvider(**final** String providerName);

/\*\*

\* store the user information in the setSharedPrefereced

\*/

**public** **void** setSharedPrefereced() ;

**Class: Request ActivityTestTextview**

**Methods:**

@Override

/\*\*

\* Initialize activity

\*/

**protected** **void** setUp() **throws** Exception;

@Override

/\*\*

\* Clearing up actitivy using

\*/

**protected** **void** tearDown() **throws** Exception;

/\*\*

\* close activity

\* **@param** activity the target activity

\*/

**private** **void** closeActivity(Activity activity);

/\*\*

\* Start this activity to test

\*/

**public** **void** openActivity() **throws** Exception;

/\*\*

\* set problems

\*/

**public** **void** setTestProblems(Bundle bundle, String problem, String item);

/\*\*

\* set loctaion

\*/

**public** **void** setTestLocation();

/\*\*

\* Helper method to add a test provider with given name.

\*/

**private** **void** addTestProvider(**final** String providerName);

/\*\*

\* store the user information in the setSharedPrefereced

\*/

**public** **void** setSharedPrefereced() ;

**Class: Request ActivityTestTextviewClick**

**Methods:**

@Override

/\*\*

\* Initialize activity

\*/

**protected** **void** setUp() **throws** Exception;

@Override

/\*\*

\* Clearing up actitivy using

\*/

**protected** **void** tearDown() **throws** Exception;

/\*\*

\* close activity

\* **@param** activity the target activity

\*/

**private** **void** closeActivity(Activity activity);

/\*\*

\* Start this activity to test

\*/

**public** **void** openActivity() **throws** Exception;

/\*\*

\* set problems

\*/

**public** **void** setTestProblems(Bundle bundle, String problem, String item)**;**

/\*\*

\* set loctaion

\*/

**public** **void** setTestLocation();

/\*\*

\* Helper method to add a test provider with given name.

\*/

**private** **void** addTestProvider(**final** String providerName);

/\*\*

\* Test if after click text in textview is right.

\*/

@UiThreadTest

**public** **void** testMsgAfterClick()

**Class: Request ActivityTestUser**

**Methods:**

@Override

/\*\*

\* Initialize activity

\*/

**protected** **void** setUp() **throws** Exception;

@Override

/\*\*

\* Clearing up actitivy using

\*/

**protected** **void** tearDown() **throws** Exception;

/\*\*

\* close activity

\* **@param** activity the target activity

\*/

**private** **void** closeActivity(Activity activity);

/\*\*

\* Start this activity to test

\*/

**public** **void** openActivity() **throws** Exception;

/\*\*

\* set problems

\*/

**public** **void** setTestProblems(Bundle bundle, String problem, String item);

/\*\*

\* set loctaion

\*/

**public** **void** setTestLocation();

/\*\*

\* Helper method to add a test provider with given name.

\*/

**private** **void** addTestProvider(**final** String providerName);

/\*\*

\* Test if after click text in textview is right.

\*/

@UiThreadTest

**public** **void** testMsgAfterClick();

/\*\*

\* ensure that the expecting data is equal

\*/

**public** **void** testUser.

**Web Application Test(Function-based test descriptions)**

Main Sectioned Tested:

**Function**

JdbcCaseDAO

**Description:**

This section has been tested using JdbcCaseDAOTest and involves various methods in obtaining and inserting Cases into the database in the class JdbcCaseDAO.

**Strategy:**

In JdbcCaseDAO the datasource is injected through the context but to test it I have added a setter for it. The various data access methods have been tested by using the connection from the datasource set through the setter.

**Function**

JdbcUserDAO

**Description:**

This section has been tested using JdbcUserDAOTest and involves methods for obtaining and inserting Cases into the database in the class JdbcUserDAO.

**Strategy:**

In JdbcUserDAO, the datasource is injected through the context but to test it I have added a setter for it. The various data access methods have been tested by using the connection from the datasource set through the setter.

**Function**

CaseModel

**Description:**

A case that contains all the information for user to report to the server.

**Strategy:**

Use JUnit to test the get and set method to ensure the expecting value is been retrieved.

**Function**

UserModel

**Description:**

To ensure that the getUsername(), setUsername(), getPassword() and set password() method are working properly.

**Strategy:**

Create two instances of users to retrieve the expecting value.

**Function**

CaseServicelmpl

**Description:**

The CaseService Implementation has been tested using CaseServiceImplTest, which is the service layer handling cases data.

**Strategy:**

The DAO objects has been mocked for ease. Injection might require integration so Mockito has been used to mock DAO objects

**Function**

UserServiceImplTest.java

**Description:**

This class tests the UserService Implementation, which is the service layer handling users data. Mockito has been used to mock DAO object.

**Strategy:**

The DAO objects has been mocked for ease. Injection might require integration so Mockito has been used to mock DAO objects.

**Web Application Test (Method-based test descriptions)**

Class: LoginControllerTest

Methods:

/\*\*

\* Test that checks if the login method in the controller is returning the

\* correct view

\*/

@Test

public void testControllerLogin();

/\*\*

\* Test that checks if the logout method in the controller is returning the

\* correct view

\*/

@Test

public void testControllerLogout();

/\*\*

\* Test that checks if the login method in the controller is returning the

\* correct view during invalid username/password

\*/

@Test

public void testControllerLoginerror();

Class:JdbcCaseDAOtest

Methods:

/\*\*

\* Setting up common objects required during the test

\* JdbcUserDAO, JdbcCaseDAO

\*/

@Before

public void setUp();

/\*\*

\* Method that tests the insertion of Case

\*/

@Test

public void testInsertCase() ;

/\*\*

\* Method that tests finding the case by username

\*/

@Test

public void testFindByUsername();

/\*\*

\* Method to test finding the case by uid

\*/

@Test

public void testFindByUid();

Class:JdbcUserDAOTest

Methods:

/\*\*

\* Test setup that sets up the UserDAO and the DataSource

\*/

@Before

public void setUp();

/\*\*

\* Method that tests the insertion of user

\*/

@Test

public void testInsertUser();

/\*\*

\* Method for testing finding user by username

\*/

@Test

public void testFindByUsername();

/\*\*

\* Method to test finding user by user id

\*/

@Test

public void testFindByUid();

Class:CaseModelTest

Methods:

/\*\*

\* Sets up the test class with a sample case

\* @throws Exception

\*/

@Before

public void setUp() throws Exception;

/\*\*

\* Method that tests the getter and setters of Case Model

\*/

@Test

public void getterSetterTest();

Class:UserModelTest

Methods:

/\*\*

\* Sets up the test class with two sample users

\* @throws Exception

\*/

@Before

public void setUp() throws Exception;

/\*\*

\* Tests the getter and setters for the User model class

\*/

@Test

public void getterSetterTest();

Class:CaseServiceImplTest

Methods:

/\*\*

\* Sets up the test classes with mock userDAO and caseDAO

\* using Mockito library and also create cases and user objects

\*/

@Before

public void setUp();

/\*\*

\* Method that tests the creation a case

\*/

@Test

public void testCreateCase();

/\*\*

\* Method that test the getting case by id

\*/

@Test

public void testGetCaseByUid();

/\*\*

\* Method that tests the getting case by username

\*/

@Test

public void testGetCaseByUsername();

Class:UserServiceImplTest

Methods:

/\*\*

\* Setup method for the test class which creates a mock object

\* for userDAO using Mockito library

\*/

@Before

public void setUp();

/\*\*

\* Method that tests the user creation by the service class

\*/

@Test

public void testCreateUser();

/\*\*

\* Method that tests the getting of user using the username

\*/

@Test

public void testGetUserByUsername();

/\*\*

\* Method that tests the getting of user by user id

\*/

@Test

public void testGetUserByUserId().