TESTING POLICY DOCUMENT

Stacks On Stacks

Abhinav	Thakur	U13286383
Akua	Afrane-Okese	U15019773
Brian	Ndung'u	U15322913
Linda	Zwane	U14199468
Ntiko	Mathaba	U14012503
Tyler	Matthews	U15302424

MAY 1, 2018
UNIVERSITY OF PRETORIA

Contents

Introduction

Purpose of Document

The purpose of this document is to provide a clear statement as to the testing process for software and to outline who is responsible for which parts of that process.

Purpose of Policy

Testing is needed in order to detect potential problems within the software as early as possible, so that they can be corrected at minimum cost.

Mission of Testing

- To strive towards perfection in the delivery of a software product which is as defect-free as possible.
- To constantly seek better methods and processes to help insure the delivery of a quality product.
- To fulfil software requirements as stipulated in the systems requirement documentation

Description of the test process

In our project, we have decided to use Test-Driven Development. Test driven development allows us to continuously test our code throughout the development cycle of our application. The following are the steps that we have used for our testing process:

1. Prepare for test-driven development

The skeleton code for the class is generated. The team member that is in charge of testing the class needs to fully understand the classes implementation. A UML diagram will aid the team member in understanding the class.

We have decided to use Java's JUnit framework for Unit testing and Mockito. Junit allows us to test a single class. Mockito allows us to mock dependencies. We are also going to use Espresso to test the user interface.

Testing Framework	Version
Junit 4	4.12
Mockito	1.10.19
Espresso	

2. Write tests

2.1 Features to be tested

1. Login
2. User interface
3. Performance

These are the following test cases that are going to be conducted to test the login functionality:

Steps	Expected Results
Enter a valid username and password. Click Login button	The application should display the Splash Screen
Logout. Enter a valid username and an invalid password. Click the Login button.	The application should display an error message and reopen the Login page
Enter an invalid username and an invalid password. Click Login button	The application should display an error message and reopen the Login page
Leave the username and password blank. Click Login button.	The application should display an error message and reopen the Login page

3. Implement and test the features

3.1 Item pass / fail criteria

The following are the possible combinations input values the user might enter and click on the login button. A test can only be deemed passed if the expected result matches the actual result. The results are as follows:

Verify username and password	Expected Result	Actual Result	Home Page / Error
Valid username, Valid Password	Success	Success	Home Page
Valid username, Invalid Password	Fail	Fail	Error

Valid username, Blank	Fail	Fail	Error
Invalid username, Valid Password	Fail	Fail	Error
Invalid username, Invalid Password	Fail	Fail	Error
Blank, Invalid Password	Fail	Fail	Error
Blank, Valid Password	Fail	Fail	Error
Blank, Invalid Password	Fail	Fail	Error
Blank, Blank	Fail	Fail	Error

The above table shows us the different combinations that the user might enter when trying to login into the Arivl application and the result of the input combination. It shows us that there is only one case that the user will be able to login into the Arivl application and that is when the user has entered a valid username and valid password.

4. Repeat until all features are correctly implemented

Iterate through step 2 and step 3 until all the features are implemented and all the tests pass.

5. Accomplish test coverage

Android Studios allows us to export the results of our tests into an html file. Below is a screen shot of our test coverage:

LoginPresenterTest: **4 total, 4 passed**   131 ms

                                                                                    Collapse | Expand

Process finished with exit code 0

LoginPresenterTest.showErrorMessageWhenPhonenumberAndOTPAreIncorrect	passed	129 ms
LoginPresenterTest.showErrorWhenPhonenumberIsEmpty	passed	1 ms
LoginPresenterTest.startMainActrivityWhenPhonenumberAndOTPAreCorrect	passed	0 ms
LoginPresenterTest.showErrorWhenOTPIsEmpty	passed	1 ms

Generated by Android Studio on 2018/05/11 12:36 AM

Levels of Testing and approaches used

The different levels of testing promote mitigation and quality risk as early as possible.

Level	Owner	Objective	Key Areas of Testing
Unit	Developer	• Detect defective units of code • Reduce the risk of unit failure in production	• Functionality • Resource Utilization
Integration	Developer	• Detect defects in unit interfaces • Reduce risk of dataflow and workflow failures in production	• Functionality • Unit interoperability and compatibility • Performance
Acceptance	Business owner	• Detect in user workflows • Demonstrate that the product works as expected • Ensure that all the requirements are met	• Functionality • Operational processes

1. Unit Testing
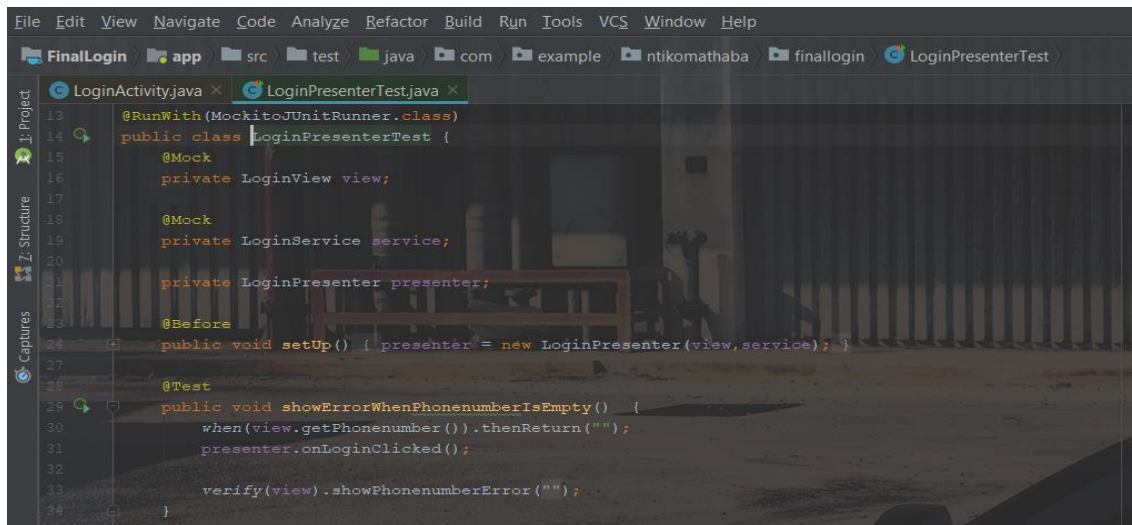
Unit tests seek to identify defects in the code and verify the functioning of testable software modules.

Unit tests are capable of verifying functionalities. Unit tests are used in order to be certain that the individual components are working correctly.

Unit tests mainly make use of white-box testing techniques and are therefore carried out with a knowledge of the code being tested and possibly also with the support of the development environment (the unit test framework, debugging tool, etc.). That is why these tests are often carried out by the developer that wrote the code or by a party of a similar calibre. Any defects identified are resolved as soon as they are detected.

In our project, we used Unit Testing for the Login Module. We used Android Studio's Junit testing framework to test this module. We also used an integrated testing framework called Mockito. Mockito is a framework that allows us to mock certain dependencies in our unit testing.
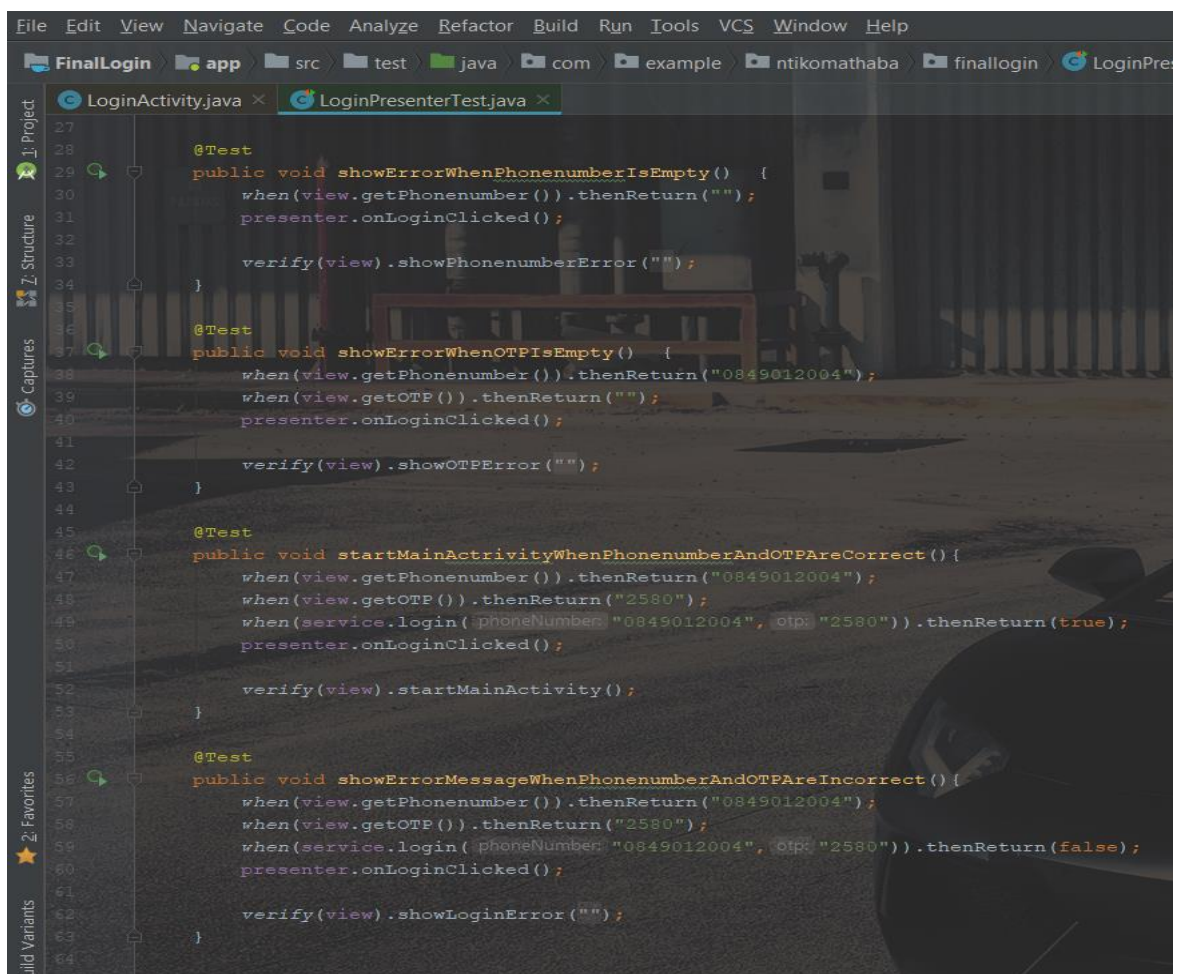
Down below is a screenshots of how we used Android Studios and Mockito's testing framework for testing our Login module.

Down below is a screenshot of the different tests we have conducted using Junits Automated Unit Testing framework

Down below is a screen shot of the tests that we have written and they have passed. They have been sorted according to time duration. It is evident that the test that takes the longest is when the user enters an invalid username and invalid password.