CPS3222 Assignment

FUNDAMENTALS OF SOFTWARE TESTING

Ryan Falzon (141497M) | Kristi Muscat (17897G)
BSC. IT (HONS) IN SOFTWARE DEVELOPMENT | THIRD YEAR

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

~~I~~/ We*, the undersigned, declare that the [assignment / Assigned ~~Practical Task~~ report / ~~Final Year~~ Project ~~report~~] submitted is my / our* work, except where acknowledged and referenced.

~~I~~ / We* understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Ryan Falzon	
Student Name	Signature
Kristi Muscat	Kristi Muscat
Student Name	Signature
Student Name	Signature
Student Name	Signature

CPS3222 — Course Code

Software Testing Assignment — Title of work submitted

03/01/18 — Date

Contents

Website Overview

The first page the user will be provided is the Home Page, which has one button to be redirected to the Contact Supervisor Page. In our case, the user will be an agent. Whenever the agent wants to use the system, he/she first needs to contact the supervisor to be given a login key. The details that are required to be entered for validation is the agent's ID and name. If the agent's ID starts with 'spy', he would not be granted a login key. On the other hand, if successful validation occurs, the agent would be redirected to the Login Key Page, were the agent's details together with the login key are displayed. The agent has 60 seconds to login to the system, otherwise it would become expired and would need to re-contact the supervisor. Once the agent is logged in to the system, the agent can send and receive messages.

The mailbox allows the user to send 25 messages and subsequently receive another 25 messages. If this limit is exceeded, the system will, automatically logout the agent. A message has a timestamp allocated with it so that each message will stay in the mailbox for 30 minutes, after which they will be inaccessible. A message is considered to be invalid under the following validations:

- The Source Agent whom is trying to send the message would not be the one logged in;
- Even though a message containing blocked words such as 'ginger' and 'recipe' is not considered an invalid message, these blocked words would be removed from the message before sending;
- Messages exceeding 140 characters are considered to be invalid;

After 10 minutes logged in to the system, the user will be automatically logged out and has to proceed in contacting the supervisor again to get a valid login key.

Testing Techniques

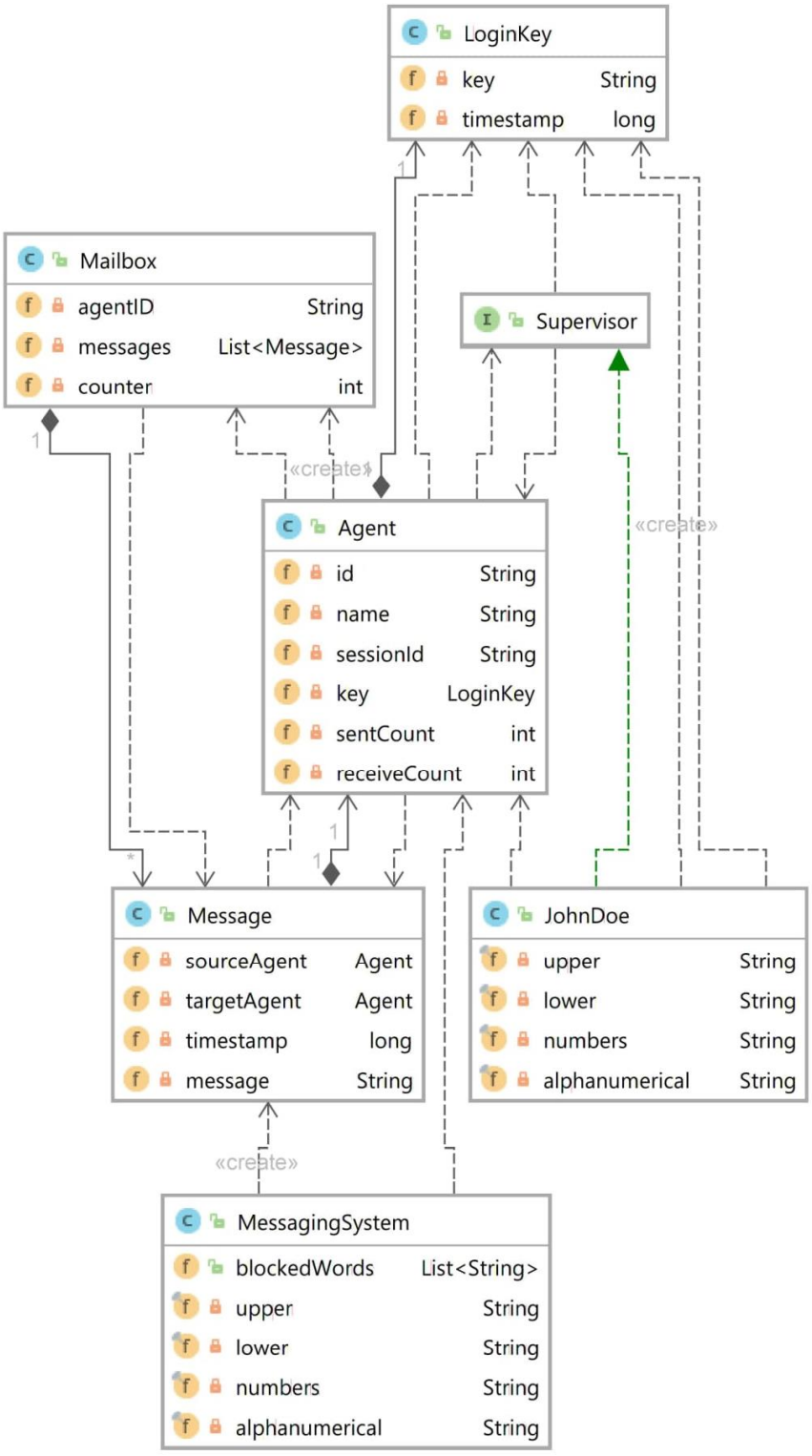
To test the above described system, the following level of testing strategies were implemented:

- Unit Testing – Through Junit;
- User Acceptance Testing – Through Selenium and Cucumber;
- Model Based Testing – Through Junit, Selenium and an FSM;
- Mobile Browser – Through Appium and AndroidDriver;

Unit Testing and Test Driven Development

While writing both the classes and tests we made sure that each test was testing for one thing only and that they always returned the same result no matter how many times they were ran. Moreover, tests were not to have any conditional logic or depend on some other tests.

Class Diagram



Class	Method	Return Type	Description
Agent	contactSupervisor	Boolean	Initiate contact with a supervisor to get a login key and subsequently logs into the system.
	sendMessage	Boolean	Sends a message to the destination agent.
	getMailbox	Integer	Get target agent's mailbox.
Mailbox	consumeNextMessage	String	Returns the next message in the box on a FIFO bases.
	hasMessages	Boolean	Check if there are any messages in the mailbox.
	checkTimestamp	Boolean	Check timestamp of all messages in a mailbox.
Messaging System	login	String	Logs in a user given an agent ID and key.
	registerLoginKey	Boolean	Takes a login key and agent ID such that when an agent with that id tries to login she will only be allowed access if the key also matches.
	getSessionKey	String	Method to generate a session key.
	sendMessage	String	Sends a message from the source agent to the target agent. Creates a message object and stores it in the target agent's mailbox.
	checkMessage	String	Check if passed string contains an element from the blocked words.
	logout	Void	A method to log out the passed user.

Test Coverage

As one can see below, through the series of tests that were written, we got 100% class, method and line coverage from all classes. The only exception was the class 'JohnDoe' whose unit tests were written in task 2 of this assignment and will therefore be seen later on in the report.

Element	Class, %	Method, %	Line, %
Agent	100% (1/1)	100% (16/16)	100% (47/47)
JohnDoe	0% (0/1)	0% (0/2)	0% (0/10)
LoginKey	100% (1/1)	100% (5/5)	100% (10/10)
Mailbox	100% (1/1)	100% (10/10)	100% (27/27)
Message	100% (1/1)	100% (9/9)	100% (18/18)
MessagingSystem	100% (1/1)	100% (7/7)	100% (46/46)

Figure 1: Coverage Analysis

Test Patterns

The type of test pattern used in the system is Dependency Injection so that dependent components were injected into test objects. Parameter injection, constructor injection and setter injection were implemented and later on tested through unit testing. An example of each injection can be seen below:

Constructor Injection

Variables were passed to the test objects via the constructor when the object is instantiated for the first time. The example provided is from the Message class.

```
1.  // Constructor
2.  public Message(Agent sourceAgent, Agent targetAgent, long timestamp, String message
    ) {
3.      this.sourceAgent = sourceAgent;
4.      this.targetAgent = targetAgent;
5.      this.timestamp = timestamp;
6.      this.message = message;
7.  }
```

Setter Injection

Private variables are set through a method which can be called at any point during runtime. The example provided is from the Mailbox class.

```
1.  // Getters and setters
2.  public String getAgentID() {
3.      return agentID;
4.  }
5.  public void setAgentID(String agentID) {
6.      this.agentID = agentID;
7.  }
```

Parameter Injection

Variables are passed as a parameter to the method being called. Therefore, in this case, if a test object requires the use of a variable from outside the object itself, it can be passed when the method is called. The example being provided is from the Agent class.

```
1.  // Initiate contact with a supervisor to get a login key and subsequently logs into
     the system
2.  public boolean contactSupervisor(Supervisor supervisor){
3.
4.      // Contact the supervisor
5.      this.key = supervisor.getLoginKey(this);
6.
7.      // Return response
8.      if(this.key != null){
9.          return true;
10.     }
11.     else{
12.         return false;
13.     }
14. }
```

Test Doubles

Test doubles are objects which are installed in place of the real component for express purpose of running a test. During almost all tests performed, the use of mock objects was very essential. This is because at this stage, the system still did not have an implementation of a supervisor, only an interface. The mocking framework that was used in this project was Mockito since it is a powerful tool which enables the dynamic creation and management of mock objects. An example of how Mockito was used to mock a supervisor is:

```
1.  // Properties
2.  private final String VALID_KEY = "0000000000";
3.
4.  @Mock
5.  private Supervisor supervisor;
6.  @InjectMocks
7.  private Agent agent;
8.
9.  @Before
10. public void setup() {
11.     StaticVariables.Erase();
12.     agent = new Agent("001", "Ryan");
13.
14.     // Initiate mockito
15.     MockitoAnnotations.initMocks(this);
16. }
17.
18. @Test
19. // Testing contactSupervisor method in agent with valid ID
20. public void testContactSupervisorValidID(){
21.     // Specify the return of the method without even having an implementation
22.     when(supervisor.getLoginKey(agent)).thenReturn(new LoginKey(VALID_KEY, System.c
    urrentTimeMillis()));
23.
24.     // Exercise
25.     boolean check = agent.contactSupervisor(supervisor);
26.     assertEquals(true, check);
27. }
```

Cucumber and Automated Web Testing

Web Application

The web application was built by using Java Server Pages (JSP) for the front end. These JSP files would then communicate to servlets whose aim is to handle HTTP requests from the web application. Finally, these servlets would communicate accordingly with the classes developed in task 1 of this assignment. At this stage, an implementation of the Supervisor interface mentioned in task 2 was developed and unit tests to test the methods in the class were written.

Element	Class, %	Method, %	Line, %
Agent	100% (1/1)	100% (16/16)	100% (47/47)
JohnDoe	100% (1/1)	100% (2/2)	93% (14/15)
LoginKey	100% (1/1)	100% (5/5)	100% (10/10)
Mailbox	100% (1/1)	100% (10/10)	100% (27/27)
Message	100% (1/1)	100% (9/9)	100% (18/18)
MessagingSystem	100% (1/1)	100% (7/7)	100% (46/46)

The JohnDoe unit tests passed successfully and yielded a 100% for both class and method coverage, and a 93% for line coverage. The reason that the tests did not result in 100%-line coverage is because in the Supervisor implementation, the method 'getLoginKey' return a unique 10-character login key. Therefore, the possibility of generating a non-unique login key is very low, thus the tests did not manage to enter the else statement of the method. This can be shown in the code below.

```
1.  // Method to generate a session key
2.  public String key(int counter) {
3.
4.      Random random = new Random();
5.      String sessionKey = "";
6.
7.      // Generate a session key according to the length passed to the method
8.      for (int i = 0; i < counter; i++) {
9.          sessionKey += alphanumerical.charAt(random.nextInt(alphanumerical.length())
    );
10.     }
11.
12.     // Check if login key is unique
13.     if(StaticVariables.keys.contains(sessionKey)){
14.         return key(counter);
15.     }
16.     else{
17.         return sessionKey;
18.     }
19. }
```

Assumptions

Some assumptions that were taken to make the website work are the following:

- Messages did not enter an agent's mailbox on their own, meaning, that the agent needed to press the 'Get Next Message' button which returned the next valid message in the mailbox;
- An agent's session did not terminate automatically. The system checks if the user's session is valid every time the user interacts with the system;
- A login key can only be used once, even if it still has not yet expired;

Web Testing

Page objects were used to write the tests since they helped keep the tests more maintainable. Moreover, by using page objects, the tests resulted to be more readable because an abstraction mechanism would be in place.

As web testing tools, Selenium and Chrome Driver were used to automate tests. Selenium has two main objects, the driver and the web element. The driver allows interaction with the browser such as navigation and finding elements. On the other hand, web elements allow encapsulation of elements within a page. Each element written in the JSP files, was given a unique name which were used to query the page to find a specific element as can be seen below.

```
1.  // Method to return a web element
2.  public WebElement find(String name){
3.      return browser.findElement(By.name(name));
4.  }
```

Tests were written by using Cucumber, a tool which Behaviour Driven Development. Tests were written in a Given-When-Then forma. The tests were divided into three files:

- Feature files – Contains narratives of the tests that were needed to be performed;
- Step Definition Files – Turned the narratives that were written in the feature files into code by using regular expressions;
- Page Objects – Acted as an abstraction mechanism for the web application in the step definition files;

Model-Based Testing

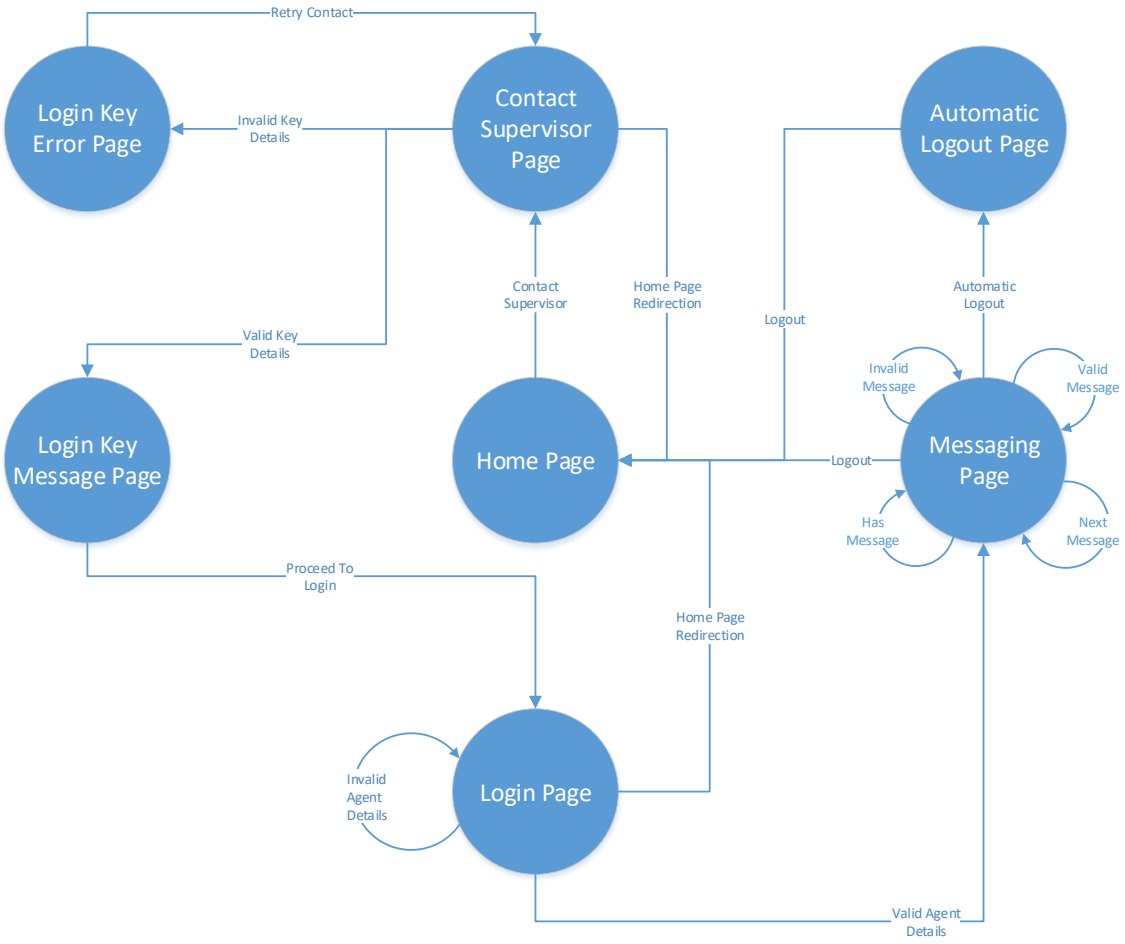
Model Overview

Our model consists of various states and transitions connecting them. Each web page in the web application represents a particular state, while the various operations such as submission of forms and pressing of buttons are represented by the actions and transitions. Whenever an action is performed in the model, the state is updated and the current page title is asserted to confirm that the transition was performed successfully. The list of states in the model are:

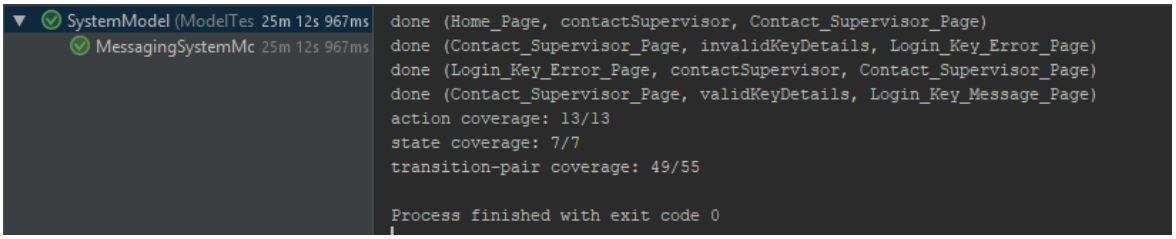
- Home Page;
- Contact Supervisor Page;
- Login Key Page;
- Login Key Error Page;
- Login Page;
- Messaging Page;
- Automatic Logout Page;

Action	Description
Contact supervisor	Moves from state Home Page to state Contact Supervisor Page.
Valid key details	Provides a valid agent name and agent ID so that he/she can move from state Contact Supervisor Page to state Login Key Page.
Invalid key details	Providing invalid agent information will result in a transition from state Contact Supervisor Page to state Login Key Error Page.
Proceed to login	Takes the agent from the Login Key Page to the Login Page.
Redirection to Home Page	Redirects the agent from the current state to the Home Page state.
Invalid agent details	Inputting invalid agent details will replace the agent in the Login Page.
Valid agent details	Entering valid details will take the agent from the Login Page to the Messaging Page.
Logout	Redirects the agent from the current state to the Home Page state.
Valid message	A valid message entry will replace the agent in the Login Page with a message sent response.
Invalid message	An invalid message will replace the agent in the Login Page with an error message as a response.
Next message	The ‘next message’ transition will keep the agent in the same state but will display the next message in the mailbox.
Has messages	The ‘has message’ transition will keep the agent in the same state but will display whether the mailbox has unread messages.
Automatic logout	An automatic logout will redirect the agent from the Message Page state to the Automatic Logout Page state.

Finite State Machine



Results



Mobile Browser Testing

Environment Setup

Before we started writing the test for mobile browser testing, first we needed to setup the environment. The following programs were installed to aid in the mobile browser testing:

- Appium – This is a server that connects the android mobile emulator with the selenium driver;
- Android Studio – Even though Android Studio was not used for coding, we made use of its built in Android Virtual Device Manager (AVD) to run an emulator of an android phone;

We left Appium's server address to its default value which is 0.0.0.0:4723. We used two emulators for this test which are:

- Nexus 6 running Android Lollipop 5.1 and OS official browser;
- Google Pixel 2 XL running Android Oreo 8.1 and Google Chrome Mobile;

The reason we used two devices to test our web application was to ensure that the web application would run easily on the majority of the devices in the market. The Nexus 6 ensured that older devices which do not have Google Chrome available were able to run the web application. On the other hand, with the tests running on the Google Pixel 2 XL we ensure that newer devices were able to handle the web application as well. To switch between devices, we had to change a piece of code in the step definition file were we were setting the emulator properties. The code below depicts this.

Nexus 6

```
1.  @Before
2.  public void setup() throws MalformedURLException {
3.      DesiredCapabilities capabilities = new DesiredCapabilities();
4.      capabilities.setCapability("platformName", "Android");
5.      capabilities.setCapability("deviceName", "Nexus_6_API_22");
6.      capabilities.setCapability("platformVersion", "5.1");
7.      capabilities.setCapability("browserName", "Browser");
8.      capabilities.setCapability("deviceOrientation", "portrait");
9.      capabilities.setCapability("appiumVersion", "1.7.2");
10.     capabilities.setCapability("newCommandTimeout", 60 * 5);
11.     System.setProperty("java.net.preferIPv4Stack" , "true");
12.     device = new AndroidDriver(new URL("http://0.0.0.0:4723/wd/hub"), capabilities)
    ;
13. }
```

Google Pixel 2 XL

```
1.  @Before
2.  public void setup() throws MalformedURLException {
3.      DesiredCapabilities capabilities = new DesiredCapabilities();
4.      capabilities.setCapability("platformName", "Android");
5.      capabilities.setCapability("deviceName", "Pixel_2_XL_API_27");
6.      capabilities.setCapability("platformVersion", "8.1");
7.      capabilities.setCapability("browserName", "Chrome");
8.      capabilities.setCapability("deviceOrientation", "portrait");
9.      capabilities.setCapability("appiumVersion", "1.7.2");
10.     capabilities.setCapability("newCommandTimeout", 60 * 5);
11.     System.setProperty("java.net.preferIPv4Stack" , "true");
12.     device = new AndroidDriver(new URL("http://0.0.0.0:4723/wd/hub"), capabilities);
13. }
```

Running The Tests

To test how a mobile browser handles the web application, the same scenarios used in the automated web tests were used. The only difference is that instead of the ChromeDriver that was being used in that task, in this task we made use of the AndroidDriver. To run the tests, we had to follow these steps:

- Start Appium server on 0.0.0.0:4723;
- Boot the android mobile emulator;
- Start tomcat server from intelliJ on localhost:8080;
- Run the class MobileCucumberRunner;

We had two issues while running these automated tests:

1. After being idle for some time, the emulator would stop the tests. This was being caused due to the timeout Appium had on the emulator. In one of the scenarios, the test was instructing the emulator to stay idle for 65 seconds. This cause a problem since it was quitting the test before the 65 seconds were over. We overcome this issue by increasing the timeout for Appium;
2. Another problem we encountered was that in some instances of the tests, some of the elements were not being found during runtime. However, we found that this was a common problem for people using Appium;

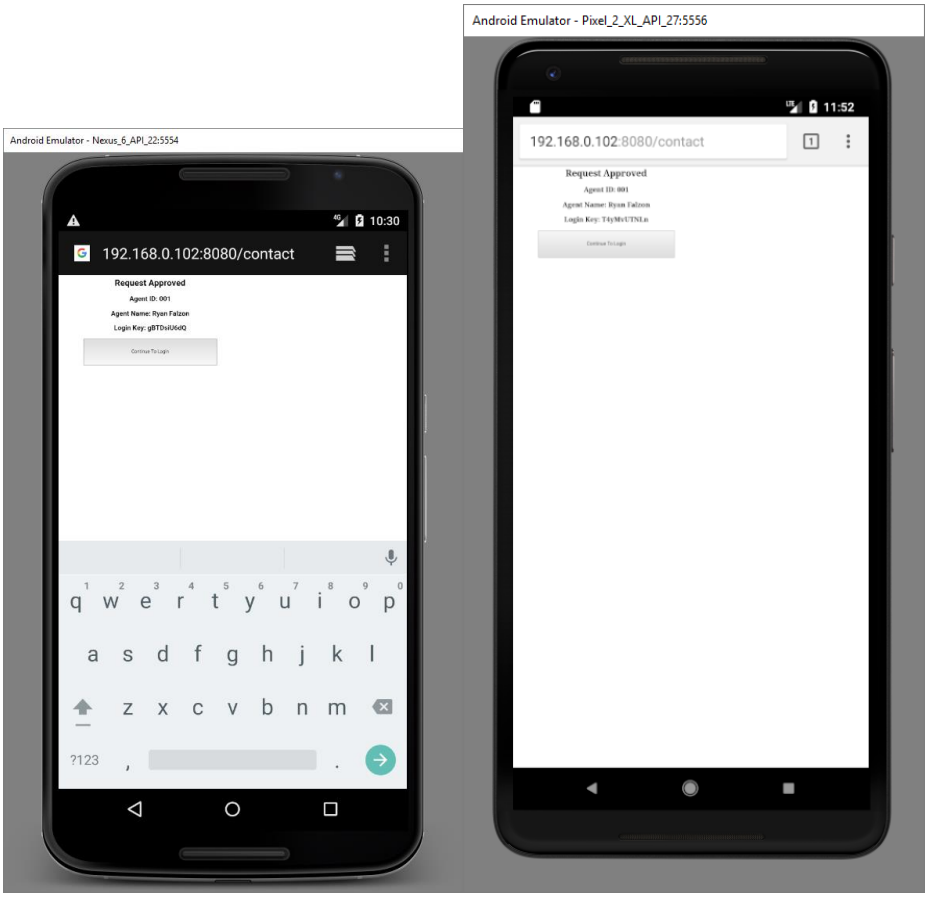


Figure 2: Nexus 6 (Left) & Google Pixel 2 XL (Right)

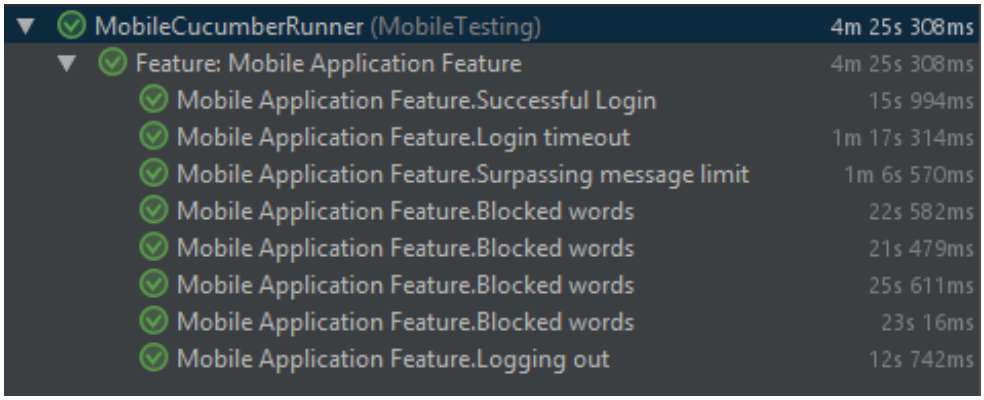


Figure 3: Results from Nexus 6

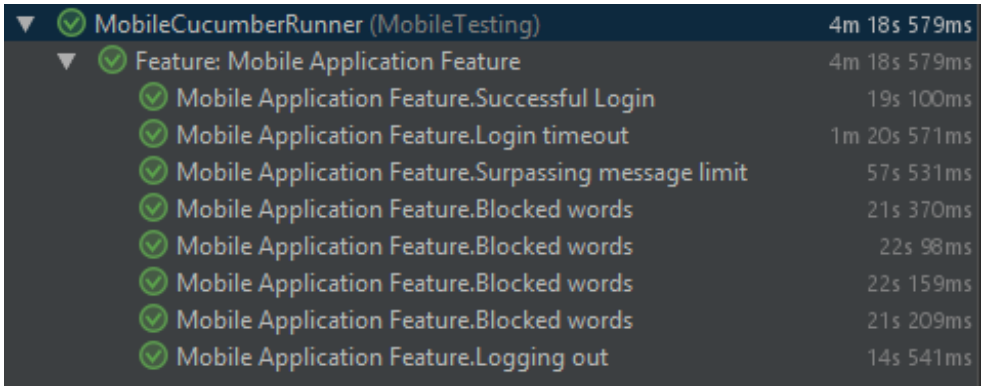


Figure 4: Results from Google Pixel 2 XL