Fundamentals of Software Testing

Assignment Report

Lydin Camilleri [492196M], Steve Attard [171497M]

Lecturer: Dr.Mark Micallef

Faculty of ICT

University of Malta

2018

CPS 3230

Fundamentals of Software Testing

Bachelor of Science (Honours) in Computing Science – 3rd 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).

______________________________	______________________________
Student Name	Signature
______________________________	______________________________
Student Name	Signature
______________________________	______________________________
Student Name	Signature
______________________________	______________________________
Student Name	Signature
________________	____
Course Code	Title of work submitted

Table of Contents

Unit Testing and Test Driven Development 3
System Implementation 3
Agent Class 3
Mailbox Class 4
Message Class 4
MessagingSystem Class 4
SupervisorAgent Class 4
Unit Test Implementations and Techniques Used 5
Test Coverage Analysis 5
Cucumber and Automated Web Testing 6
Web Application Development 6
Java Server Pages 6
Java Servlets 7
Tomcat Server 8
Web Application Testing 9
Scenarios 9
Step Definitions 9
Results 10
Model-Based Testing 11
Paperwork 11
Implementation 11
Initial Setup 11
Guards and Transitions 12
Test Runner Implementation 13
Changes performed 14

Unit Testing and Test Driven Development

System Implementation

Using a Test-Driven Development approach, where failing tests are written first and then enough code is written to pass the test, the system was built-up and designed in the following classes as explained in the subsections below.

Agent Class

The Agent class represents the agents making use of the system. Attributes of this class include *agentID* which uniquely identifies each agent, *mailbox* which contains all the messages that the agent receives, *messageCount* which keeps track of the number of messages sent and a *supervisorID* that links the agent to his supervisor. Other attributes which make part of this class are *loginKey* and *sessionKey* which store a 10 and 50 character key respectively, and two time stamp attributes *loginTime* and *loginKeyAcquiredTime*. The latter are set when an agent logs into the system and obtains a login key from his supervisor. In this section, a brief explanation shall be given to the some of the fundamental methods found in the Agent class. All of these methods were designed to follow the specification rules provided in the assignment sheet. The *retrieveLoginKeyFromSupervisor()* method is used to ask the supervisor of agent invoking the method call for a login key. Once the key is obtained, the *loginKeyAcquiredTime* attribute is set to *System.currentTimeMillis()*, so that a check can be made when the agent tries to log in, with the intention of verifying that the login key being used has not yet expired. The *login()* method is used after a login key is obtained. If this method is invoked with a non-expired login key, the agent will be allowed to use the system and the *loginTime* attribute is set to *System.currentTimeMillis()*, with the intention of checking whether or not a logout is required later on. The *sendMessage()* method is used to communicate with agents in the system. Several checks are performed in this method such as exceeded message quota and whether or not a log out is required due to exceeded session time. The *readNextMessage()* method is used to consume a message from the mailbox of the agent.

Mailbox Class

Mailbox instances are used to store messages that an agent receives. A mailbox is essentially made up of two ArrayLists, both containing messages (instances of the Message class). These two ArrayLists store the unread and read messages of its owner. Once a message is consumed, it is moved to the *readMessages* ArrayList. Messages are consumed in a FIFO order using the method *consumeNextMessage()*. This method returns *null* if the no unread messages are in the mailbox or thenext message instance otherwise

Message Class

Each message instance includes the *agentID* of both the sender and receiver, the message content and a reference timestamp that caches the creation time of the message. The *timestamp* attribute is important as a message which remains unread for 30 minutes is deleted, thus a record of when the message was created has to be kept.

MessagingSystem Class

This class is the central hub of the system where every class interacts with another. The implementation includes an ArrayList of agents that are logged in the system, an ArrayList consisting of the system blocked words, and two HashMaps, *agentLoginKeys* and *agentSessionKeys*, which are populated once the keys are given to the agent respectively. The following method descriptions explain the most important methods which make up this class. The *login()* method is used to generate a session key for an agent in possession of a valid login key. The *registerLoginKey()* method is used to put a valid login key in the *agentLoginKeys* HashMap when a login key is acquired by an agent. The *sendMessage()* method is used to perform some checks on a message which an agent wishes to send to another, such as word limit, filtering blocked words and a full recipient's mailbox. Should a message not meet the specified requirements, it is not sent and an appropriate error is returned. Otherwise, a new message instance is created and placed in the recipient's inbox.

SupervisorAgent Class

The SupervisorAgent class extends the Agent Class and implements the Supervisor Interface. The main purpose of this class is to generate the login key for agents to login in the system, which is made up of 10 random characters including numbers, letter and symbols.

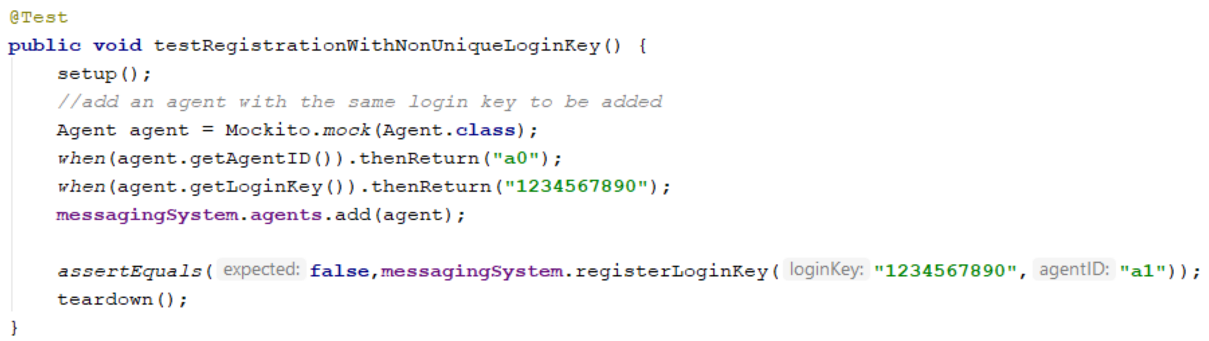
Furthermore, a supervisor agent can do anything that a normal agent is allowed to, which is why this class is a subclass of the Agent class.

Unit Test Implementations and Techniques Used

JUnit and Mockito used are mainly used to create the required unit tests. Tests are written for almost every method in each class. Tests are written to test every possible branch of the code to ensure that every path taken during runtime is functioning as expected. This will be verified later in this report in the Code Coverage section.

As some methods require the use of other external methods found in other classes, Mockito is used to mock these classes. In this way, the developer is provided with the ability to mimic the return values of external methods used in the method under test, which make the result of the test completely independent from the correctness of these external methods. In other words, bugs and faults in external methods will not affect the results obtained in this test, helping to identify the source of any incorrect procedures with ease.

An example of this is shown in the snippet below. The registerLoginKey() method found makes use of the getAgentID() and the getLoginKey() found in the Agent class. The return values of these methods are hardcoded as shown below using when().thenReturn().

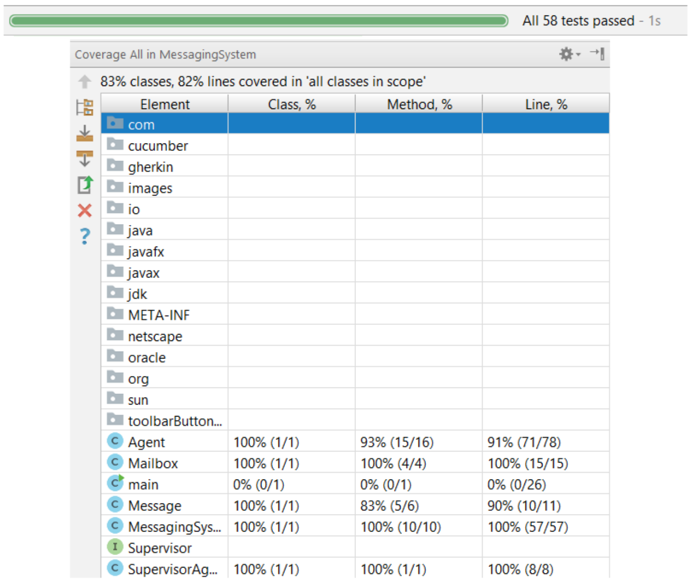


```
@Test
public void testRegistrationWithNonUniqueLoginKey() {
    setup();
    //add an agent with the same login key to be added
    Agent agent = Mockito.mock(Agent.class);
    when(agent.getAgentID()).thenReturn("a0");
    when(agent.getLoginKey()).thenReturn("1234567890");
    messagingSystem.agents.add(agent);

    assertEquals( expected: false,messagingSystem.registerLoginKey( loginKey: "1234567890", agentID: "a1"));
    teardown();
}
```

The same procedure is used for every test regarding a method which makes use of external methods. Next we analyse the code coverage obtained following the write up and execution of all the unit tests.

Test Coverage Analysis

A total of 58 tests were written to develop the system in a TTD approach. The code coverage obtained was satisfactory enough, with having tested almost every method and line in each class. Having obtained this kind of code coverage, it was decided that enough unit testing was done and it was time to build a web application which would serve as an application layer on top of the developed system.



All 58 tests passed - 1s

Coverage All in MessagingSystem

83% classes, 82% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
com			
cucumber			
gherkin			
images			
io			
java			
javafx			
javax			
jdk			
META-INF			
netscape			
oracle			
org			
sun			
toolbarButton...			
Agent	100% (1/1)	93% (15/16)	91% (71/78)
Mailbox	100% (1/1)	100% (4/4)	100% (15/15)
main	0% (0/1)	0% (0/1)	0% (0/26)
Message	100% (1/1)	83% (5/6)	90% (10/11)
MessagingSys...	100% (1/1)	100% (10/10)	100% (57/57)
Supervisor			
SupervisorAg...	100% (1/1)	100% (1/1)	100% (8/8)

Cucumber and Automated Web Testing

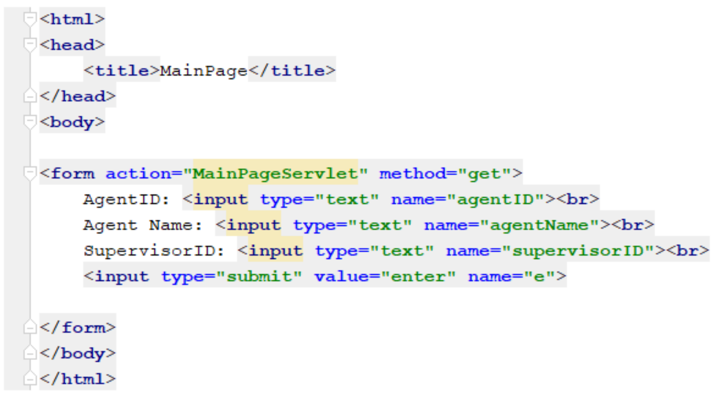
Web Application Development

The web application was developed using an Apache Tomcat 8 server, Java Server Pages and Java Servlets. JSP files were used to write web pages in HTML, which will provide an interface for the user to make use of the system. The servlets' main job is to link the JSP files to the Messaging System developed in part 1 and perform redirections from one JSP file to another. Each JSP file has its own servlet and it is responsible of handling the actions of the user when the JSP file is being used in the browser.

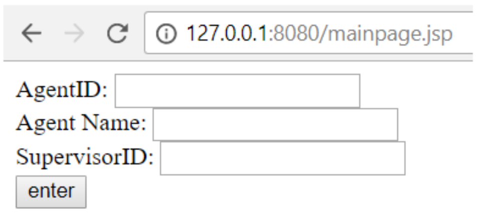
Java Server Pages

As already pointed out, the main reasons why JSP files are used in this case is to display a user interface on the web by writing HTML code in these files.

The agentID, agentName and supervisorID inputs will be used in the corresponding Main Page Servlet where these values will be fed as inputs to the underlying messaging system. The tasks performed by the servlet are discussed in the following section and in the below snippet, the main page JSP file and the resultant interface on the browser are shown.

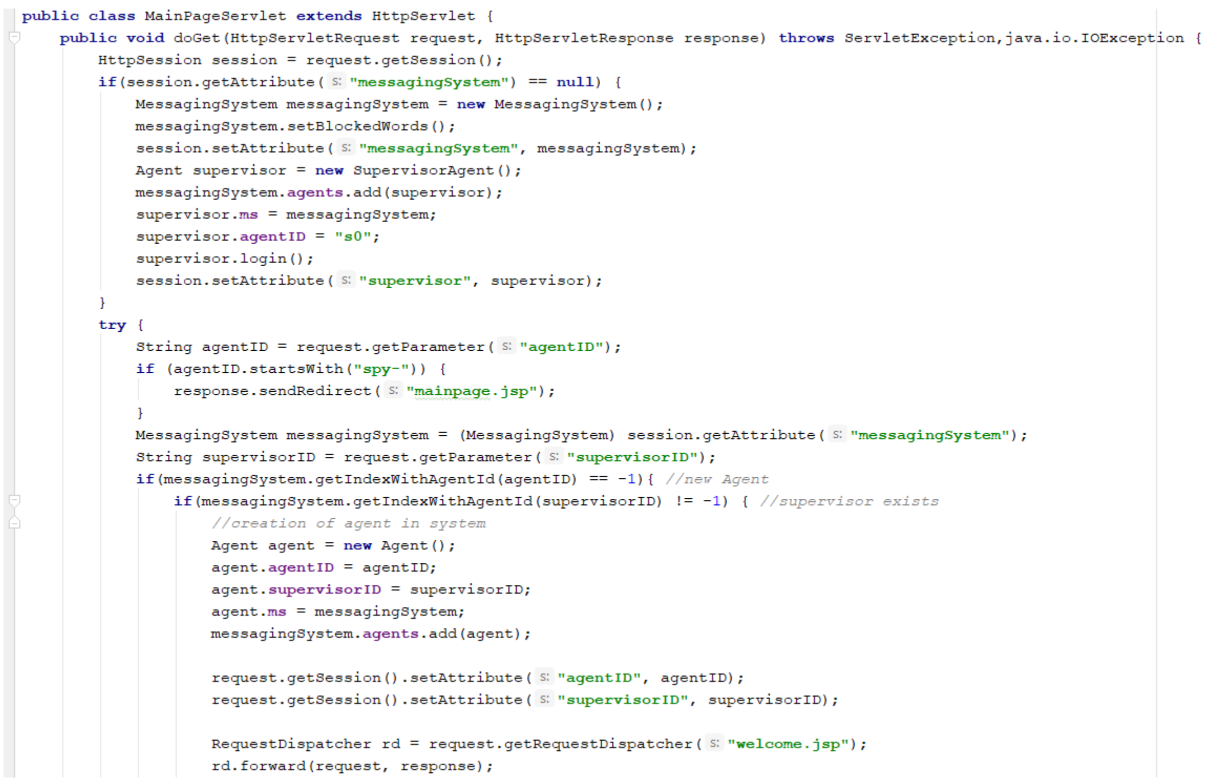


```
<html>
<head>
    <title>MainPage</title>
</head>
<body>

<form action="MainPageServlet" method="get">
    AgentID: <input type="text" name="agentID"><br>
    Agent Name: <input type="text" name="agentName"><br>
    SupervisorID: <input type="text" name="supervisorID"><br>
    <input type="submit" value="enter" name="e">

</form>
</body>
</html>
```



Java Servlets

Since the main page is the first page that will be loaded when the web app is being used, the role of the main page servlet is to create a system if one does not exist. Furthermore, it must interact with the inputs entered by the user in the main page JSP file described above. To do these tasks, an if statement is used to check if a session attribute with the name messagingSystem exists. If no such attribute is found, a new messaging system is initialized and set as a session attribute. The following step is to retrieve the inputs entered by the user, namely agentID and supervisorID. This is done through the request.getParameter() method, which is called to retrieve the input and store it inside a String variable. At this point, methods defined during the creation of the system in part 1 are used to determine whether an agent is anew agent or not, whether the specified supervisor exists or not etc. After these checks and manipulations are made, the servlet redirects the user to another JSP file, i.e. another web page. In the following page, a figure with a section of the main page servlet is shown.

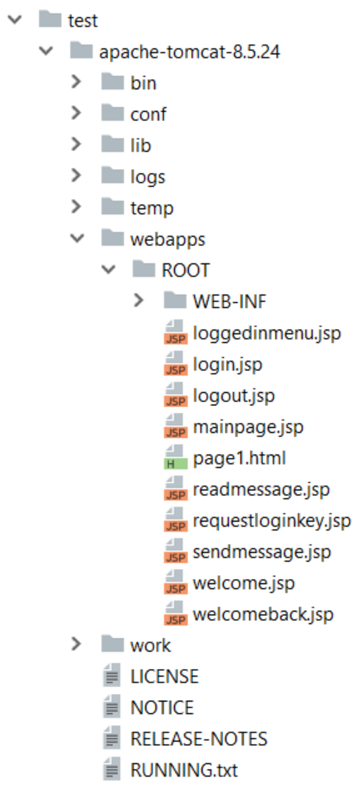


```
public class MainPageServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,java.io.IOException {
        HttpSession session = request.getSession();
        if(session.getAttribute( s: "messagingSystem") == null) {
            MessagingSystem messagingSystem = new MessagingSystem();
            messagingSystem.setBlockedWords();
            session.setAttribute( s: "messagingSystem", messagingSystem);
            Agent supervisor = new SupervisorAgent();
            messagingSystem.agents.add(supervisor);
            supervisor.ms = messagingSystem;
            supervisor.agentID = "s0";
            supervisor.login();
            session.setAttribute( s: "supervisor", supervisor);
        }
        try {
            String agentID = request.getParameter( s: "agentID");
            if (agentID.startsWith("spy-")) {
                response.sendRedirect( s: "mainpage.jsp");
            }
            MessagingSystem messagingSystem = (MessagingSystem) session.getAttribute( s: "messagingSystem");
            String supervisorID = request.getParameter( s: "supervisorID");
            if(messagingSystem.getIndexWithAgentId(agentID) == -1){ //new Agent
                if(messagingSystem.getIndexWithAgentId(supervisorID) != -1) { //supervisor exists
                    //creation of agent in system
                    Agent agent = new Agent();
                    agent.agentID = agentID;
                    agent.supervisorID = supervisorID;
                    agent.ms = messagingSystem;
                    messagingSystem.agents.add(agent);

                    request.getSession().setAttribute( s: "agentID", agentID);
                    request.getSession().setAttribute( s: "supervisorID", supervisorID);

                    RequestDispatcher rd = request.getRequestDispatcher( s: "welcome.jsp");
                    rd.forward(request, response);
```

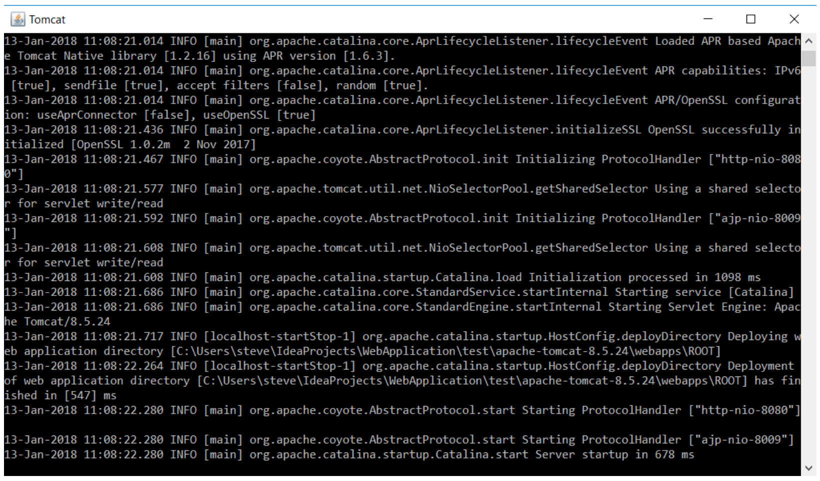
The same technique of creating a web page using a JSP file and a corresponding servlet was used to build the whole application.

Tomcat Server

The Apache Tomcat server was used to have a host for the application to run on. The directory structure of the server is shown in the following snippet. In the ROOT folder, all JSP files are stored and in the subfolder WEB-INF, one can find a folder for all the classes and the web.xml file. The server is run by going to the bin directory in the command prompt and run the command startup.bat. This will load the server and once it is up and running, users can access the main page by typing the URL 127.0.0.1:8080/mainpage.jsp. The following window is shown after the start up command is executed.



```
test
  apache-tomcat-8.5.24
    bin
    conf
    lib
    logs
    temp
    webapps
      ROOT
        WEB-INF
        loggedinmenu.jsp
        login.jsp
        logout.jsp
        mainpage.jsp
        page1.html
        readmessage.jsp
        requestloginkey.jsp
        sendmessage.jsp
        welcome.jsp
        welcomeback.jsp
    work
    LICENSE
    NOTICE
    RELEASE-NOTES
    RUNNING.txt
```



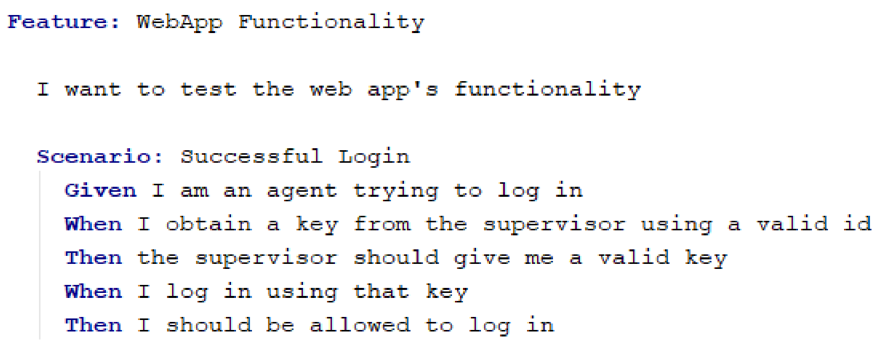
```
13-Jan-2018 11:08:21.014 INFO [main] org.apache.catalina.core.AprLifecycleListener.lifecycleEvent Loaded APR based Apach
e Tomcat Native library [1.2.16] using APR version [1.6.3].
13-Jan-2018 11:08:21.014 INFO [main] org.apache.catalina.core.AprLifecycleListener.lifecycleEvent APR capabilities: IPv6
 [true], sendfile [true], accept filters [false], random [true].
13-Jan-2018 11:08:21.014 INFO [main] org.apache.catalina.core.AprLifecycleListener.lifecycleEvent APR/OpenSSL configurat
ion: useAprConnector [false], useOpenSSL [true]
13-Jan-2018 11:08:21.436 INFO [main] org.apache.catalina.core.AprLifecycleListener.initializeSSL OpenSSL successfully in
itialized [OpenSSL 1.0.2m  2 Nov 2017]
13-Jan-2018 11:08:21.467 INFO [main] org.apache.coyote.AbstractProtocol.init Initializing ProtocolHandler ["http-nio-808
0"]
13-Jan-2018 11:08:21.577 INFO [main] org.apache.tomcat.util.net.NioSelectorPool.getSharedSelector Using a shared selecto
r for servlet write/read
13-Jan-2018 11:08:21.592 INFO [main] org.apache.coyote.AbstractProtocol.init Initializing ProtocolHandler ["ajp-nio-8009
"]
13-Jan-2018 11:08:21.608 INFO [main] org.apache.tomcat.util.net.NioSelectorPool.getSharedSelector Using a shared selecto
r for servlet write/read
13-Jan-2018 11:08:21.608 INFO [main] org.apache.catalina.startup.Catalina.load Initialization processed in 1098 ms
13-Jan-2018 11:08:21.686 INFO [main] org.apache.catalina.core.StandardService.startInternal Starting service [Catalina]
13-Jan-2018 11:08:21.686 INFO [main] org.apache.catalina.core.StandardEngine.startInternal Starting Servlet Engine: Apac
he Tomcat/8.5.24
13-Jan-2018 11:08:21.717 INFO [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deploying w
eb application directory [C:\Users\steve\IdeaProjects\WebApplication\test\apache-tomcat-8.5.24\webapps\ROOT]
13-Jan-2018 11:08:22.264 INFO [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment
of web application directory [C:\Users\steve\IdeaProjects\WebApplication\test\apache-tomcat-8.5.24\webapps\ROOT] has fin
ished in [547] ms
13-Jan-2018 11:08:22.280 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]

13-Jan-2018 11:08:22.280 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-nio-8009"]
13-Jan-2018 11:08:22.280 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in 678 ms
```

Web Application Testing

Once the application was ready, some scenarios had to be tested to verify that the web app was working as expected. The scenarios described in the assignment sheet where written in a features file using Cucumber and each step definition was written in a class called WebAppStepDefs. In these step definitions, Selenium is used to instruct the browser what inputs it should enter.

Scenarios

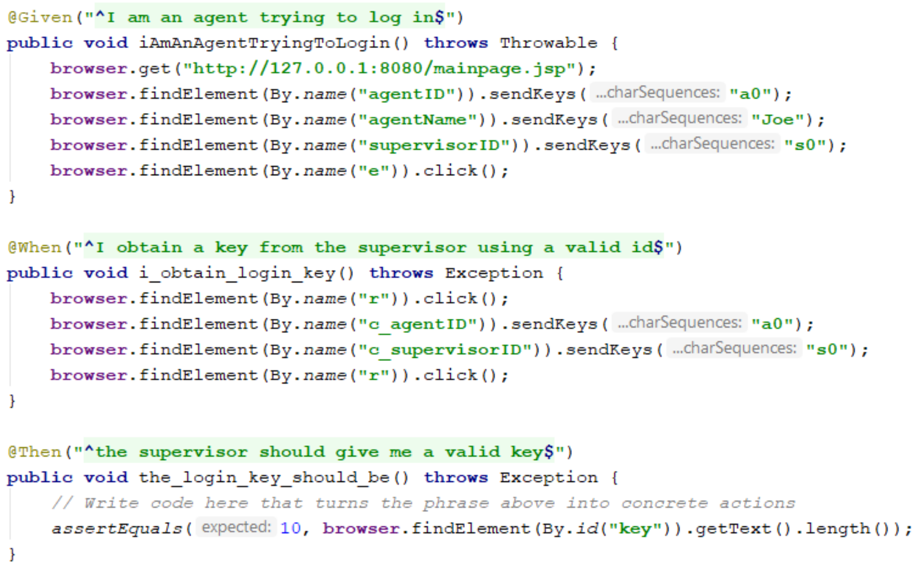
The scenarios are written exactly as specified in the assignment sheet in the WebApp.features file. Each scenario is made up of statements written in English, and each statement will have a respective step definition in the WebAppStepDefs Java class. The figure below shows the first scenario.



```
Feature: WebApp Functionality

  I want to test the web app's functionality

  Scenario: Successful Login
    Given I am an agent trying to log in
    When I obtain a key from the supervisor using a valid id
    Then the supervisor should give me a valid key
    When I log in using that key
    Then I should be allowed to log in
```

Step Definitions

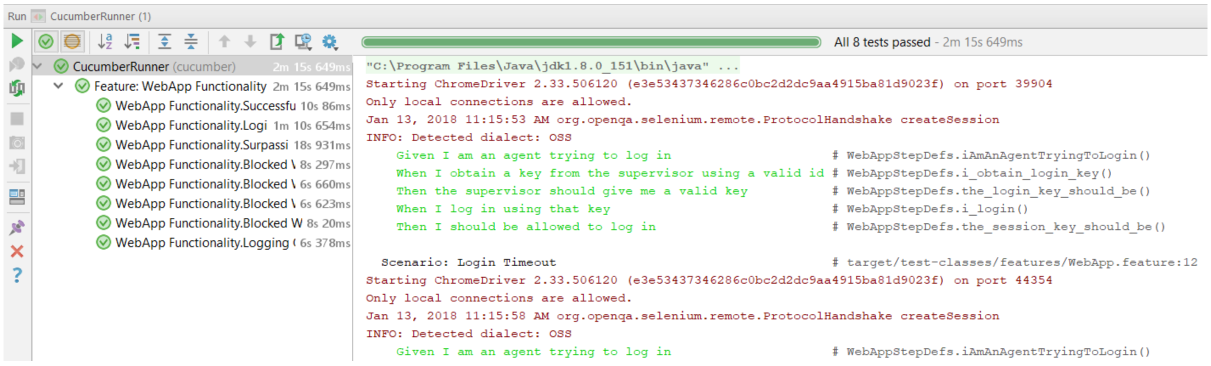
Step definitions make use of Selenium and Cucumber to drive the web browser. As the following snippet shows assertions are used when a method starts with a @Then annotation. Step definitions which start with the annotations @Given, @When and @And contain statements which locate elements in the web pages by name or id and perform action on these elements, such as clicks or filling in. In the case shown here, a valid login key is being asserted by making sure that it is exactly 10 characters long.



```
@Given("^I am an agent trying to log in$")
public void iAmAnAgentTryingToLogin() throws Throwable {
    browser.get("http://127.0.0.1:8080/mainpage.jsp");
    browser.findElement(By.name("agentID")).sendKeys( ...charSequences: "a0");
    browser.findElement(By.name("agentName")).sendKeys( ...charSequences: "Joe");
    browser.findElement(By.name("supervisorID")).sendKeys( ...charSequences: "s0");
    browser.findElement(By.name("e")).click();
}

@When("^I obtain a key from the supervisor using a valid id$")
public void i_obtain_login_key() throws Exception {
    browser.findElement(By.name("r")).click();
    browser.findElement(By.name("c_agentID")).sendKeys( ...charSequences: "a0");
    browser.findElement(By.name("c_supervisorID")).sendKeys( ...charSequences: "s0");
    browser.findElement(By.name("r")).click();
}

@Then("^the supervisor should give me a valid key$")
public void the_login_key_should_be() throws Exception {
    // Write code here that turns the phrase above into concrete actions
    assertEquals( expected: 10, browser.findElement(By.id("key")).getText().length());
}
```

Results

As the next figure shows, all eight tests passed. There are eight tests in all since the fourth scenario is an outline meaning that it takes different inputs and expects different outputs for the same test.



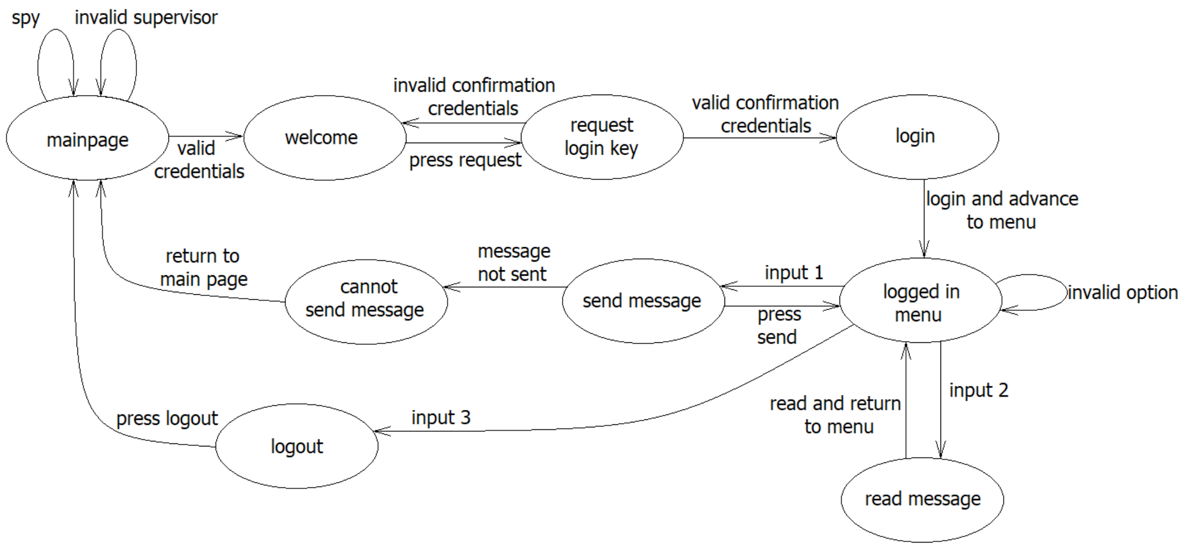
```
Run  CucumberRunner (1)
All 8 tests passed - 2m 15s 649ms

CucumberRunner (cucumber)                 2m 15s 649ms
  Feature: WebApp Functionality           2m 15s 649ms
    WebApp Functionality.Successfu        10s 86ms
    WebApp Functionality.Logi             1m 10s 654ms
    WebApp Functionality.Surpassi         18s 931ms
    WebApp Functionality.Blocked \        8s 297ms
    WebApp Functionality.Blocked \        6s 660ms
    WebApp Functionality.Blocked \        6s 623ms
    WebApp Functionality.Blocked W        8s 20ms
    WebApp Functionality.Logging (        6s 378ms
```

```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
Starting ChromeDriver 2.33.506120 (e3e53437346286c0bc2d2dc9aa4915ba81d9023f) on port 39904
Only local connections are allowed.
Jan 13, 2018 11:15:53 AM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
    Given I am an agent trying to log in                          # WebAppStepDefs.iAmAnAgentTryingToLogin()
    When I obtain a key from the supervisor using a valid id # WebAppStepDefs.i_obtain_login_key()
    Then the supervisor should give me a valid key            # WebAppStepDefs.the_login_key_should_be()
    When I log in using that key                                  # WebAppStepDefs.i_login()
    Then I should be allowed to log in                            # WebAppStepDefs.the_session_key_should_be()

  Scenario: Login Timeout                                         # target/test-classes/features/WebApp.feature:12
Starting ChromeDriver 2.33.506120 (e3e53437346286c0bc2d2dc9aa4915ba81d9023f) on port 44354
Only local connections are allowed.
Jan 13, 2018 11:15:58 AM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
    Given I am an agent trying to log in                          # WebAppStepDefs.iAmAnAgentTryingToLogin()
```

Model-Based Testing

Paperwork

The system plan was first analysed and an abstraction model view of it was recognized. A Finite-State Machine model was built by creating a list of states and transitions that the system should contain in order to have the functionality that the requirements instruct. The below FSM diagram shows the model of the built system.



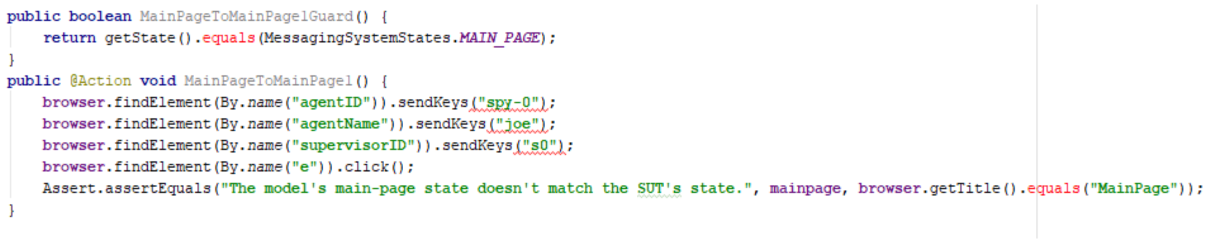
Implementation

Initial Setup

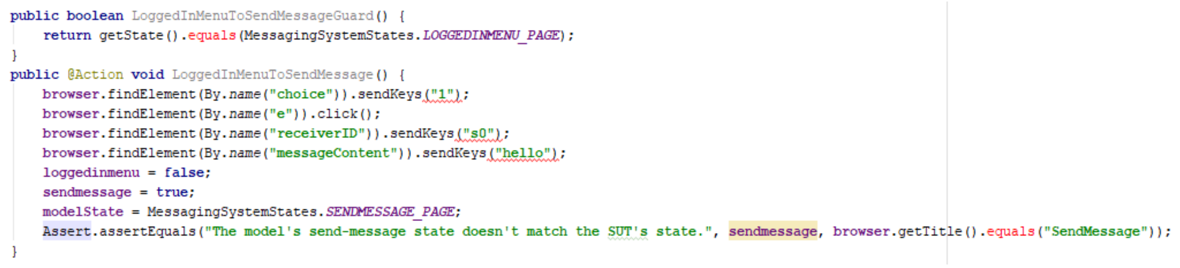
Implementing the model started off with creating the ENUM class containing every state presented in the FSM model. Then, a MessagingSystemModelTest class was created which implements the FsmModel interface and an instance of the system was initiated. The states were then declared and the MainPage state was set as the initial state when running the model. A *getState()* method was also implemented to get the state that the model is at any time.

Guards and Transitions

In order to indicate the system automation when it is possible to choose a particular action according to the state it is in, guards were introduced. For every transition designed in the FSM model, two methods were implemented: a Boolean Guard method and an @Action method. The Guard method simply checks that the state that the system is in allows the transition before applying it whilst the @Action method actually implements the transition and is responsible for updating the system under test by executing the relevant methods, updates the model's state variables and asserts consistency between the model and the actual system. Note that the guard method was always named the same as the @Action method concatenated with the word "Guard" at the end.



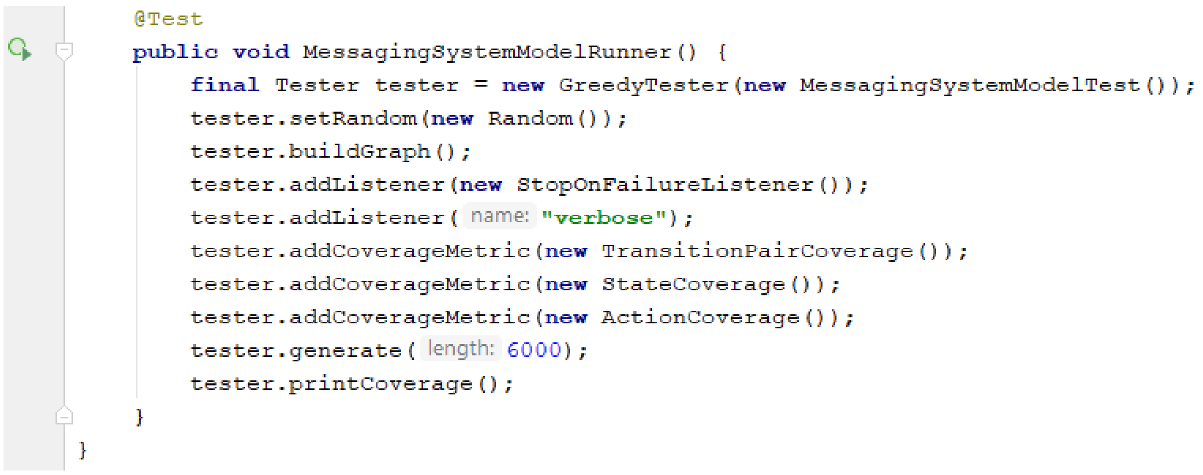
```java
public boolean MainPageToMainPage1Guard() {
    return getState().equals(MessagingSystemStates.MAIN_PAGE);
}
public @Action void MainPageToMainPage1() {
    browser.findElement(By.name("agentID")).sendKeys("spy-0");
    browser.findElement(By.name("agentName")).sendKeys("joe");
    browser.findElement(By.name("supervisorID")).sendKeys("s0");
    browser.findElement(By.name("e")).click();
    Assert.assertEquals("The model's main-page state doesn't match the SUT's state.", mainpage, browser.getTitle().equals("MainPage"));
}
```



```java
public boolean LoggedInMenuToSendMessageGuard() {
    return getState().equals(MessagingSystemStates.LOGGEDINMENU_PAGE);
}
public @Action void LoggedInMenuToSendMessage() {
    browser.findElement(By.name("choice")).sendKeys("1");
    browser.findElement(By.name("e")).click();
    browser.findElement(By.name("receiverID")).sendKeys("s0");
    browser.findElement(By.name("messageContent")).sendKeys("hello");
    loggedinmenu = false;
    sendmessage = true;
    modelState = MessagingSystemStates.SENDMESSAGE_PAGE;
    Assert.assertEquals("The model's send-message state doesn't match the SUT's state.", sendmessage, browser.getTitle().equals("SendMessage"));
}
```

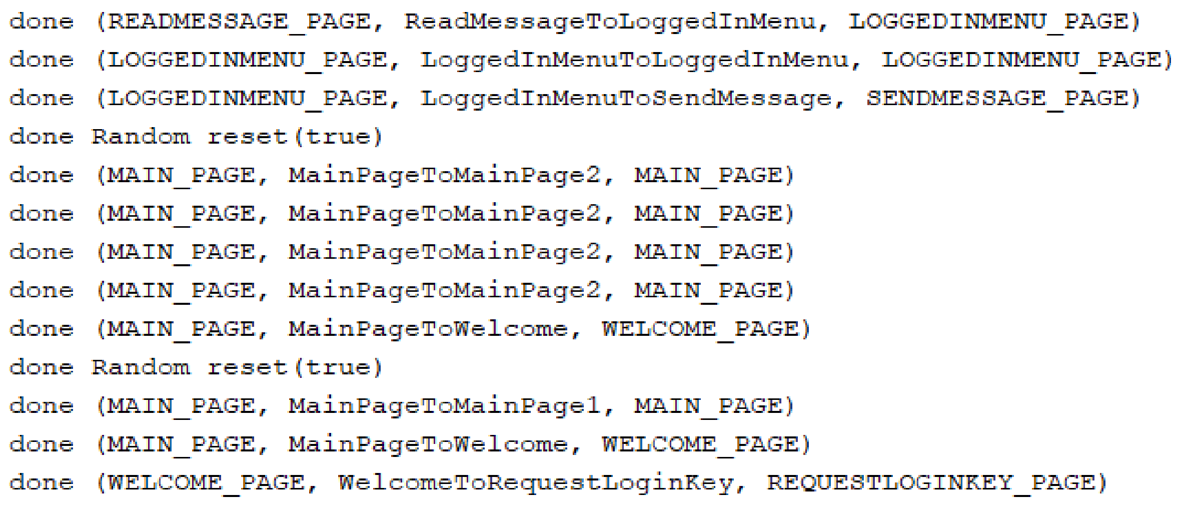
As it can be seen in the snippet above, the Guard is simply a two liner implementation. In the @Action, which name is identical to the Guard method excluding "Guard" as explained earlier, the Web Page System is driven using Selenium commands (in this case *browser.findElement()*) to behave as expected and then the state variable of the model is changed by converting the Boolean values of the states according to the transition. Finally, an assert command is implemented to ensure that the Webpage behaves as expected and therefore changing the page accordingly by using the Selenium commands such as *browser.getTitle()* and *browser.getCurrentUrl()* accordingly.

Test Runner Implementation

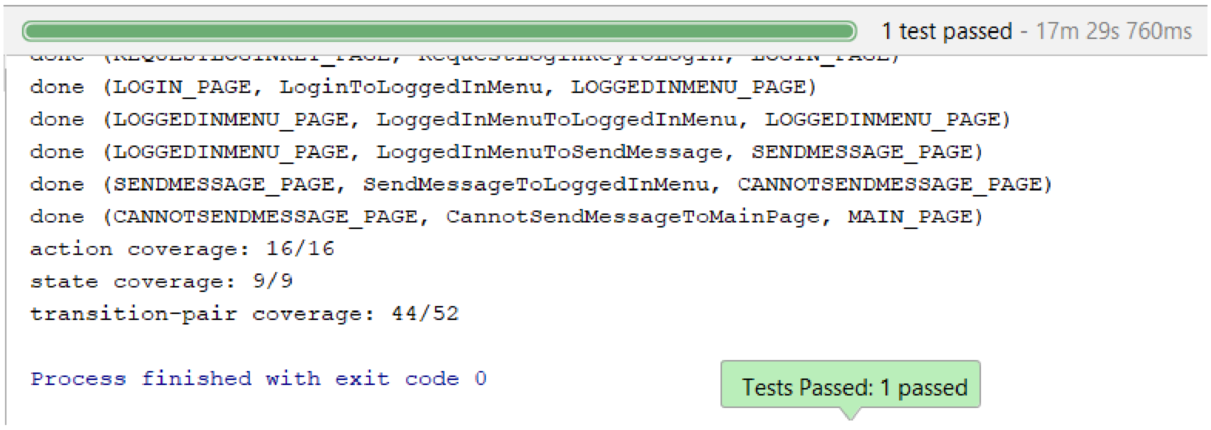


```
    @Test
    public void MessagingSystemModelRunner() {
        final Tester tester = new GreedyTester(new MessagingSystemModelTest());
        tester.setRandom(new Random());
        tester.buildGraph();
        tester.addListener(new StopOnFailureListener());
        tester.addListener( name: "verbose");
        tester.addCoverageMetric(new TransitionPairCoverage());
        tester.addCoverageMetric(new StateCoverage());
        tester.addCoverageMetric(new ActionCoverage());
        tester.generate( length: 6000);
        tester.printCoverage();
    }
}
```

For the automation testing of the model and the Messaging System, a Greedy Tester was used. This tester decides, according to the current state, which transitions are yet to be visited and randomly traverses the model in different ways. A *StopOnFailureListener()* was added to inform us that the system didn't behave as expected and coverage metrics were also added to identify the progress running of the system, the states reached and actions performed. This tester was set to run for 6000 transitions which amounted up to a 17 minutes 29 seconds sequence of tests that passed without failing. The below screenshots represent the Greedy Test Runner execution together with the coverage analysis results.



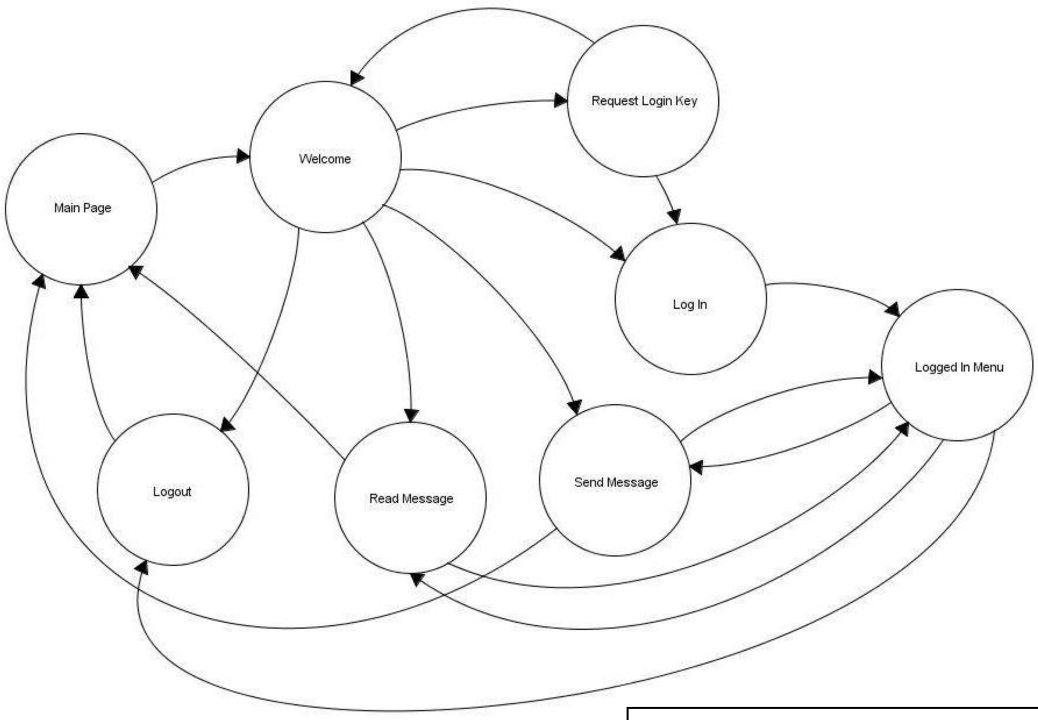
```
done (READMESSAGE_PAGE, ReadMessageToLoggedInMenu, LOGGEDINMENU_PAGE)
done (LOGGEDINMENU_PAGE, LoggedInMenuToLoggedInMenu, LOGGEDINMENU_PAGE)
done (LOGGEDINMENU_PAGE, LoggedInMenuToSendMessage, SENDMESSAGE_PAGE)
done Random reset(true)
done (MAIN_PAGE, MainPageToMainPage2, MAIN_PAGE)
done (MAIN_PAGE, MainPageToMainPage2, MAIN_PAGE)
done (MAIN_PAGE, MainPageToMainPage2, MAIN_PAGE)
done (MAIN_PAGE, MainPageToMainPage2, MAIN_PAGE)
done (MAIN_PAGE, MainPageToWelcome, WELCOME_PAGE)
done Random reset(true)
done (MAIN_PAGE, MainPageToMainPage1, MAIN_PAGE)
done (MAIN_PAGE, MainPageToWelcome, WELCOME_PAGE)
done (WELCOME_PAGE, WelcomeToRequestLoginKey, REQUESTLOGINKEY_PAGE)
```



```
done (LOGIN_PAGE, LoginToLoggedInMenu, LOGGEDINMENU_PAGE)
done (LOGGEDINMENU_PAGE, LoggedInMenuToLoggedInMenu, LOGGEDINMENU_PAGE)
done (LOGGEDINMENU_PAGE, LoggedInMenuToSendMessage, SENDMESSAGE_PAGE)
done (SENDMESSAGE_PAGE, SendMessageToLoggedInMenu, CANNOTSENDMESSAGE_PAGE)
done (CANNOTSENDMESSAGE_PAGE, CannotSendMessageToMainPage, MAIN_PAGE)
action coverage: 16/16
state coverage: 9/9
transition-pair coverage: 44/52

Process finished with exit code 0
```

Changes performed

Some changes had to be done to the Web Application since during Model Testing it was noticed that certain transitions did not make sense. These include transitions from the Welcome state to the Logout state directly without forcing a logging in procedure. Due to such transitions, the system was updated to only allow a *LoginKeyRequest* procedure from the Welcome state. Additional transitions form the MainPage state to itself were included to handle cases for when the Supervisor does not exist or the AgentID starts with "spy". (*MainPageToMainPage1()* and *MainPageToMainPage2()*).



Erroneous Model before