CPS3230 – Fundamentals of Software Testing

ASSIGNMENT PART 2 OF 3

UNIVERSITY OF MALTA
L-Università ta' Malta

Robert Fenech Adami (0165802L)
B.Sc. (Hons.) Information Technology (Software Development)
December 2022

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published" (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), 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 making a false declaration may include, but are not limited to, 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).

Robert Fenech Adami

_____ _____

Student Name Signature

_____ _____

Student Name Signature

CPS3230 Assignment Part 2 of 3

_____ _____

Course Code Title of work submitted

2/1/2023

_____

Date

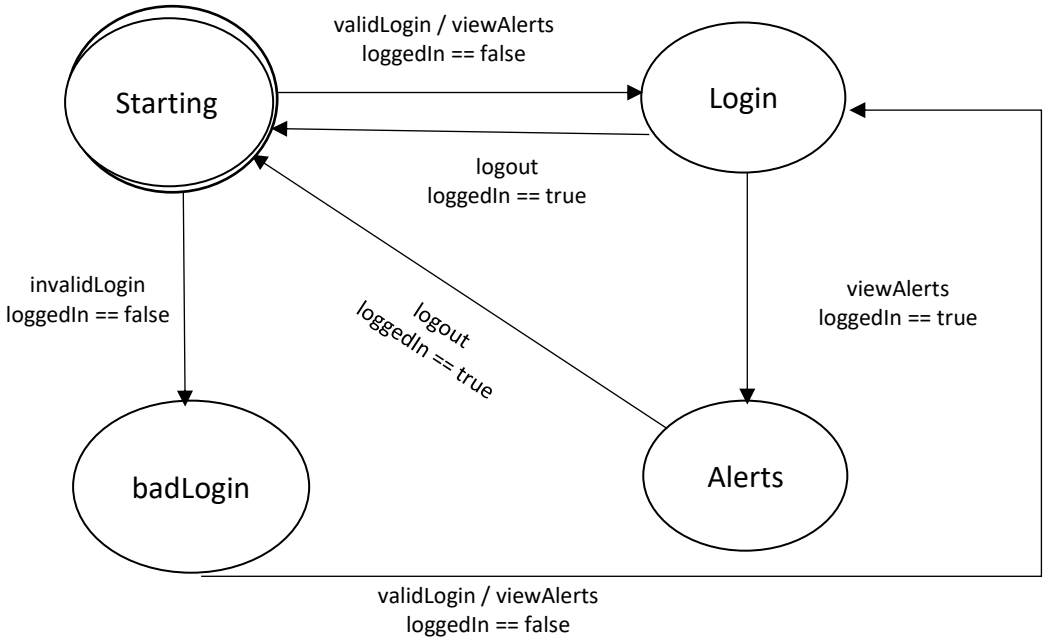
GitHub repository: https://github.com/robbyfa/CPS3230-Assignment-2

Google Drive:
https://drive.google.com/drive/folders/1spSVI73AnIQiFWhRiWkZNkZe_5dhQsIA?usp=sharing

Task 2.1

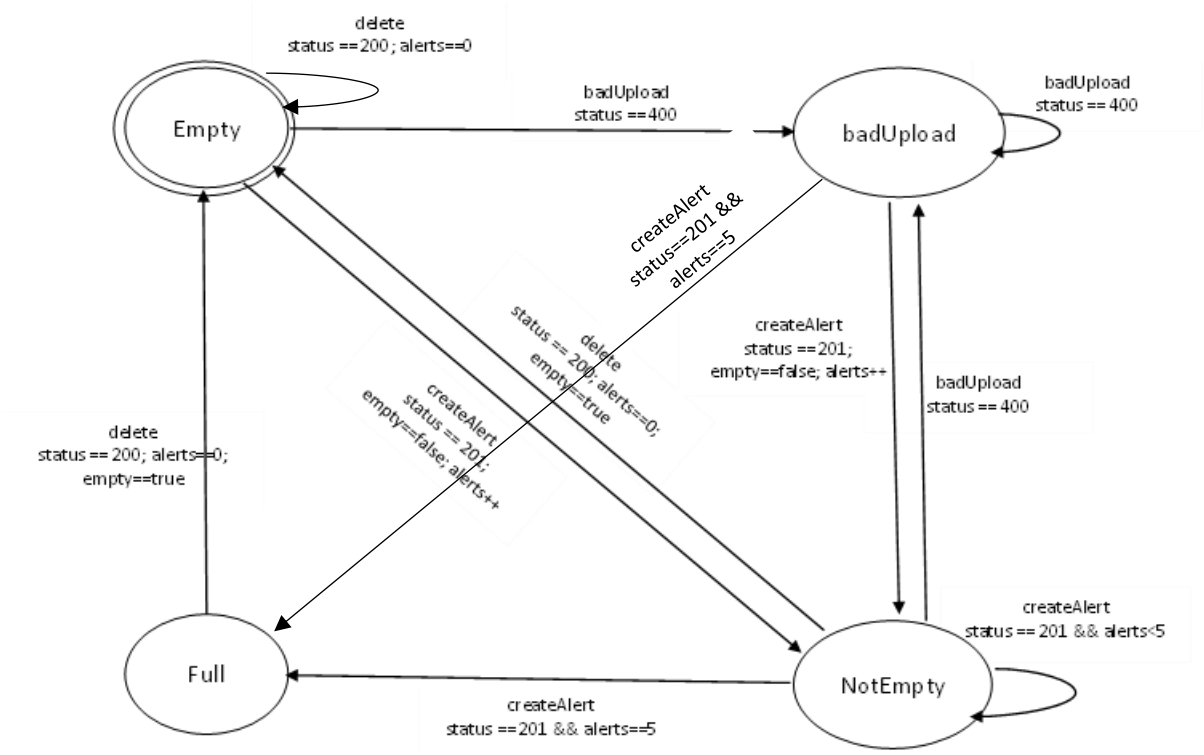
The system specification is broken down into 2 automata corresponding to the Logging in property of the system and the alerts handling property of the system respectively.

Login Property:



Before logging in to the system, the user is at the starting state (Home Page). Upon valid login, the system is now transferred to the logged in state (LoggedIn). From here, the user can either view already published alerts via the viewAlerts event or else logout and return to the starting state. When in the Alerts state, the user may also opt to log out and return to the starting, logged out state.

Alerts Property:

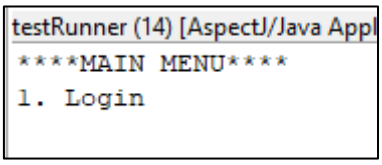


The starting state of the system for this property refers to an empty alerts catalogue. Upon creating an alert, the system is either moved to a NotEmpty state (Good upload) or a badUpload state. These are verified using the response codes by the corresponding HTTP requests (201 = good, 400 = bad). When in the NotEmpty state, new alerts may be created until reaching the alert limit condition; when the number of alerts reaches 5, the system moves onto the Full state. Alerts may be deleted, and this will cause the system to revert to the original empty starting state.

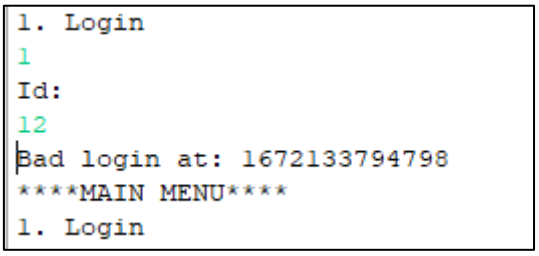
All actions are also monitored via their corresponding event log type.

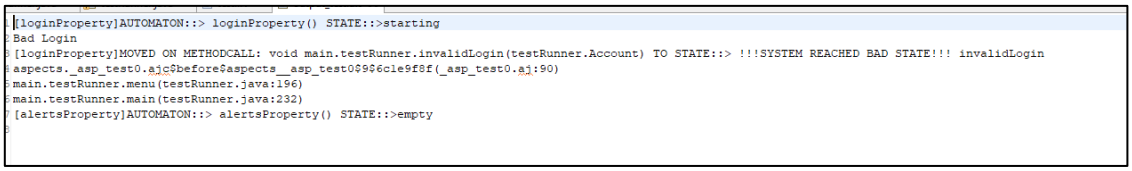
Task 2.2

The SUT makes use of a test Account which contains zero alerts. Upon compilation, the user may navigate through the system through a displayed menu.

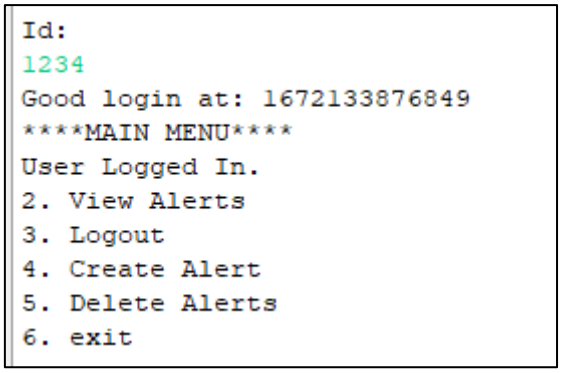


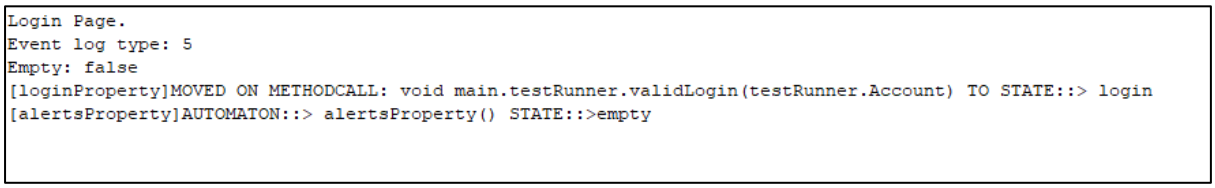
Upon invalid login, the system state is moved to the invalidLogin state.



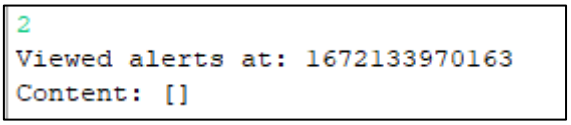


Upon valid login, the system moves onto the login state.





When viewing alerts, the system's Login property moves to the Alerts state.



```
Alerts Page
Event log type: 7
[loginProperty]MOVED ON METHODCALL: void main.testRunner.viewAlerts(testRunner.Account) TO STATE::> alerts
[alertsProperty]AUTOMATON::> alertsProperty() STATE::>empty
```

The Login Property then returns to the starting state upon logout.

```
3
User logged out at: 1672134215259
****MAIN MENU****
1. Login
```

```
Logged out
Event log type: 6
[loginProperty]MOVED ON METHODCALL: void main.testRunner.logout(testRunner.Account) TO STATE::> starting
```

When creating an alert, the Alerts property moves from the empty to the notEmpty state.

```
4
Alert created at: 1672134302238
Code:201
```

```
Alert created. Number of alerts: 1. Event log type: 0
Empty: false
[alertsProperty]MOVED ON METHODCALL: void main.testRunner.createAlert(testRunner.Account) TO STATE::> notEmpty
[loginProperty]AUTOMATON::> loginProperty() STATE::>login
[alertsProperty]AUTOMATON::> alertsProperty() STATE::>notEmpty
```

In the case of an unsuccessful upload, the state is moved to the badUpload state.

```
4
Bad upload
```

```
Bad upload. Status: 400
[alertsProperty]MOVED ON METHODCALL: void main.testRunner.createAlert(testRunner.Account) TO STATE::> !!!SYSTEM REACHED BAD STATE!!! badUpload
aspects._asp_test0.ajc$before$aspects__asp_test0$8$2a28a79(_asp_test0.aj:81)
main.testRunner.menu(testRunner.java:209)
main.testRunner.main(testRunner.java:238)
```

When deleting alerts, the Alert property state is always reverted to the starting, Empty state.

```
5
Alerts deleted at: 1672135021610
Code: 200
```

```
Alerts deleted. Event log type: 1
Empty: true
[alertsProperty]MOVED ON METHODCALL: void main.testRunner.deleteAlerts(testRunner.Account) TO STATE::> empty
```

Once the limit of Alerts is reached, the system should move onto the Full state.

```
Alerts full. Number of alerts: 6
Status: 201
EventLogType: 0
[alertsProperty]MOVED ON METHODCALL: void main.testRunner.createAlert(testRunner.Account) TO STATE::> full
[loginProperty]AUTOMATON::> loginProperty() STATE::>login
[alertsProperty]AUTOMATON::> alertsProperty() STATE::>full
```

```
EVENTS {
    validLogin() = {*.validLogin()}
    viewAlerts() = {*.viewAlerts()}
    logout() = {*.logout()}
    invalidLogin() = {*.invalidLogin()}
    logIn(boolean loggedIn) = {*.setLogged(loggedIn)}
    setEmpty(boolean empty) = {*.setEmpty(empty)}
    setStatus(int status) = {*.setStatus(status)}
    logType(int eventLogType) = {*.setEventLogType(eventLogType)}
    createAlert() = {*.createAlert()}
    deleteAlerts() = {*.deleteAlerts()}
}

PROPERTY loginProperty(
    STATES{
        BAD{
            invalidLogin
        }
        NORMAL{
            login
            alerts
        }
        STARTING{
            starting
        }
    }
    TRANSITIONS{


        starting -> invalidLogin [invalidLogin\loggedIn == false\System.out.println("Bad Login");]
        invalidLogin -> login [validLogin\loggedIn == true\System.out.println("Login Page. \nEvent log type: "+eventLogType+"\nEmpty: "+empty);]
        starting -> login [validLogin\loggedIn == true\System.out.println("Login Page. \nEvent log type: "+eventLogType+"\nEmpty: "+empty);]
        starting -> login [viewAlerts\loggedIn == true\ System.out.println("Login Page \nEvent log type: "+eventLogType);]
        starting -> alerts [viewAlerts\loggedIn == true\ System.out.println("Viewing Alerts. Event log type: "+eventLogType);]
        login -> alerts [viewAlerts\loggedIn == true\ System.out.println("Alerts Page \nEvent log type: "+eventLogType);]
        login -> starting [logout\loggedIn == false\  System.out.println("Logged out \nEvent log type: "+eventLogType);]
        alerts -> starting [logout\loggedIn == false\ System.out.println("Logged out \nEvent log type: "+eventLogType);]



    }
```

```
PROPERTY alertsProperty(
    STATES{
        BAD{
            badUpload
        }
        NORMAL{
            notEmpty
            full
        }
        STARTING{
            empty
        }
    }
    TRANSITIONS{
        empty -> notEmpty [createAlert\empty==false && status==201\alerts=0; alerts++;System.out.println("Alert created. Number of alerts: "+alerts+". Event log type: "+eventLogType+"\nEmpty: "+empty);]
        badUpload -> notEmpty [createAlert\empty==false && status==201\ alerts++;System.out.println("Alert created. Number of alerts: "+alerts+". Event log type: "+eventLogType+"\nEmpty: "+empty);]
        notEmpty -> notEmpty [createAlert\alerts<5 && status==201\  alerts++;System.out.println("Alert created. Number of alerts: "+alerts+"\nStatus: "+status+"\nEventLogType: "+eventLogType);]
        notEmpty -> full [createAlert\alerts>=5 && status==201\  alerts++;System.out.println("Alerts full. Number of alerts: "+alerts+"\nStatus: "+status+"\nEventLogType: "+eventLogType);]
        empty -> badUpload [createAlert\status==400\System.out.println("Bad upload. Status: "+status);]
        notEmpty -> empty [deleteAlerts\empty==true && status==200\ alerts=0; System.out.println("Alerts deleted. Event log type: "+eventLogType+"\nEmpty: "+empty);]
        full -> empty [deleteAlerts\empty==true && status==200\ alerts = 0; System.out.println("Alerts deleted. Event log type: "+eventLogType+"\nEmpty: "+empty);]
        notEmpty ->badUpload [createAlert\status==400\ System.out.println("Bad upload. Number of alerts: "+alerts+"\nStatus: "+status);]
        empty -> empty [deleteAlerts\empty==true && status==200\ alerts=0; System.out.println("Alerts deleted. Event log type: "+eventLogType+"\nEmpty: "+empty);]
        badUpload -> badUpload [createAlert\status==400\ System.out.println("Bad upload. Number of alerts: "+alerts+"\nStatus: "+status);]
        badUpload -> full [createAlert\alerts>=5 && status==201\  alerts++;System.out.println("Alerts full. Number of alerts: "+alerts+"\nStatus: "+status+"\nEventLogType: "+eventLogType);]
    }
}
```

Assumptions:

When creating a new alert, a random number generator is used to simulate a Good or Bad upload with 201 and 400 status codes respectively.

Task 2.3

The model-based test was split into two tests, one for the Alerts property and the other for the login property. The guards and their respective actions may be seen below.

Login Property:

```
    public Object getState() { return modelAlert; }

    public boolean getStatus() { return loggedIn; }

    public void reset(boolean var1) {...}

    // BadLogin
    public boolean invalidLoginGuard(){ return getState().equals(AlertsStates.STARTING); }
    public @Action void invalidLogin(){...}


    // Login
    public boolean loginGuard(){ return getState().equals(AlertsStates.STARTING) || getState().equals(AlertsStates.BADLOGIN); }
    public @Action void login(){...}

    // Logout
    public boolean logoutGuard() {...}
    public @Action void logout() {...}


    // View Alerts
    public boolean viewAlertsGuard() {...}
    public @Action void viewAlerts() throws IOException {...}

    @Test
    public void AlertModelRunner() {...}
}
```

Alerts Property:

```
    public Object getState() { return modelAlert; }


    public void reset(boolean var1) {...}


    // Create Alert
    public boolean createAlertGuard() {
        return !(getState().equals(AlertsStates.FULL)) && (getState().equals(AlertsStates.EMPTY) || getState().equals(AlertsStates.NOTEMPTY) || getState().equals(AlertsStates.BADUPLOAD));
    }
    public @Action void createAlert() throws IOException {...}

    // Bad Upload
    public boolean badUploadGuard(){...}
    public @Action void badUpload() throws IOException {...}

    // Fill Alerts
    public boolean fillAlertsGuard(){...}
    public @Action void fillAlerts() throws IOException {...}


    // Delete Alerts
    public boolean deleteAlertsGuard() {...}
    public @Action void deleteAlerts() throws IOException {...}

    @Test
    public void AlertsRunner() {...}
}
```

The reset method in the Login Property returns the system to the initial state by resetting the number of alerts to 0 and logged in status to false.

```
public void reset(boolean var1) {
    if (var1) {
        systemUnderTest = new Alerts();
    }
    modelAlert = AlertsStates.STARTING;
    noOfAlerts = 0;
    eventLogType = 10;
    loggedIn = false;
}
```

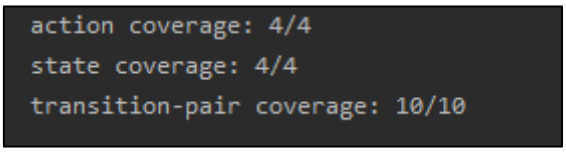
The reset method in the Alerts Property returns the system to the initial, empty state and resets the number of alerts to 0, empty to false, full to false and status code to 0.

```
public void reset(boolean var1) {
    if (var1) {
        systemUnderTest = new Alerts();
    }
    modelAlert = AlertsStates.EMPTY;
    noOfAlerts = 0;
    eventLogType = 10;
    status = 0;
    empty = true;
    full = false;
}
```

ACTION	GUARD
Login	State = Starting or state = BadLogin
InvalidLogin	State = Starting
Logout	State =LoggedIn or state = Alerts
CreateAlert	State != Full and (State = Empty/NotEmpty/BadUpload)
BadUpload	State = Empty/NotEmpty/BadUpload
FillAlerts	(State = Empty/NotEmpty) and state != Full
DeleteAlerts	State = Empty/NotEmpty/Full
ViewAlerts	State = LoggedIn and LoggedIn = true

Both tests were measured using a Greedy Tester with 500 randomly generated transitions. The below figures show the coverage metrics observed when running both tests.

Login Property Test:



Alerts Property Test: