Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia - Social and Behavioral Sciences 47 (2012) 2031 – 2035

Procedia
Social and Behavioral Sciences

CY-ICER 2012

Effective testing with JSFUnit for educational applications

Abdulhadi Celenlioglu[a] *, Birsen G. Ozdemir[b]

[a]*Department of Information Systems Engineering, Doğuş University, İstanbul, 34722, Turkey*
[b]*Department of Computer Engineering Doğuş University,İstanbul, 34722, Turkey*

Abstract

Software testing is one of the steps of software quality assurance. Even testing is a crucial element in the software process, it can be seen by developers as "time consuming" or destructive instead of constructive work. However, it should be considered as a "must" of a project, in the case of taking failure rate of software projects into account. After the decision-making on following testing procedure, the first question that comes to mind is "How do I test my application?" Java Server Faces (JSF) can offer a Java-based Web framework for educational applications. It assists to simplify development of web-based user interfaces. While doing testing for the applications utilizing JSF Framework, developers can use JSFUnit, which is a test framework allowing complete integration testing and unit testing.This paper focuses on sharing experiences with developers who intend to use JSFUnit for testing educational applications utilizing JSF Framework.

Keywords: JSF, Unit testing, JSFUnit, Java;

1. Introduction

Testing is an important aspect for each application including educational ones. Moreover, applications should not be deployed unless they are not thoroughly tested. Unit tests are written programs running in test classes. (Cheon Y. et.al., 2002)They send a settled message to a class to verify the returned value with the expected answer. Unit tests can aim to check any believably inoperative feature of a class. (Zhiming, Z. et.al., 2011) Tests can be written for each produced class in an application. Before deploying the application all unit tests should be run. In case of encountering any test failure, the problem should be realized and rectified. Generally, applications' business logic is tried to be tested by unit tests. (Tremblay G. et.al., 2003) After unit testing different modules of an application, integration testing can also be planned to be applied on the aggregated group of application modules.

JSFUnit framework is originated from JUnit framework. JSFUnit allows you to perform integration and unit test of your JSF application through the container structure. (Silvert S., 2011) The container involves managed beans as a controller, XHTML pages written in expression language (EL) for views, navigation flow control and application configuration. (Figure 2) JSFUnit tests can access the each view state of your application (Dewitt & Siraj, 2010).

*Abdülhadi Çelenlioğlu Tel.: +90-535-316-3166
*E-mail address:*hadican.re@gmail.com

The reason behind writing this paper is to highlight the importance of unit testing and touch on few key points of testing by benefitting experiences. Furthermore, not many frameworks exist in the market for testing JSF applications that encourages developers. We intended to aim making developer's job easy.

The paper simply involves the usage pattern of JSFUnit, key points that developers should follow while testing educational web applications utilizing JSF Framework.

2. JSFUnit Usage Pattern

JSFUnit, being software of JBoss, is a test framework for Java Server Faces (JSF) applications. It provides complete integration testing and unit testing of JSF applications. (Silvert, S., 2011) The general usage pattern of JSFUnit starts with sending HTTP request to JSF application and then, according to received HTTP response JSFUnit can arrange HTML output and JSF internals in order to use in testing process of the JSF application. (Figure 1)

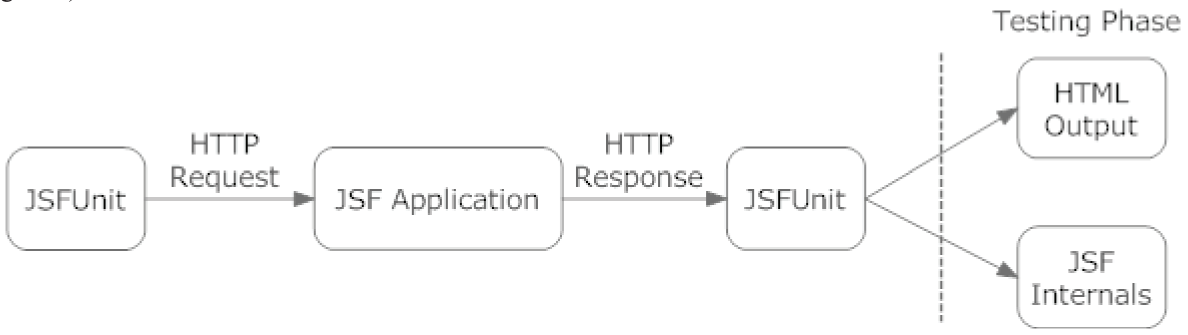

Figure 1. JSFUnit working mechanism

JSF (Java Server Faces) can offer a Java-based Web framework for educational applications. (Hong-qing G. et.al, 2009) Each JSF application employs the container structure. The container involves managed beans as a controller and defines references for each user inputs and data. JSF Framework works with managed beans to interact with XHTML pages using Expression Languages (EL). (Kurniawan B. et.al., 2004) Page navigation control can be done by the managed beans, according to navigation rules and cases defined in application configuration. Furthermore, validation messages generated by the managed beans are added into Faces Context and can be demonstrated in views. (Figure 2) If you are not familiar with JSF, you can first read "JavaServer Faces" book written by Hans Bergsten. (Bergsten H., 2004)

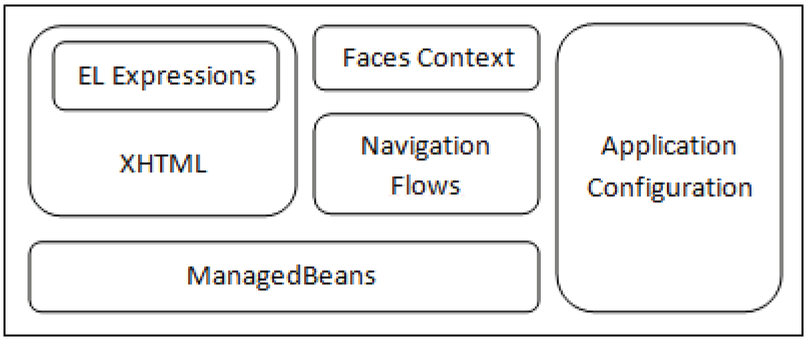

Figure 2. The container structure in JSF Framework

Through running inside the container, JSFUnit allows wider coverage for testing like accessing the FacesContext, managed beans and states, EL Expressions, navigation flows, application configuration, the component tree, HTML output of each client request etc.

A test case defines the fixture to run multiple tests. Each test case, which is defined as a test case class in JSF application, runs in its own channel so there can be no interaction between test runs. (Klein H., 2004) Each new JSF application hosted by web servers accessed via a request-response programming model accompanied with a Servlet, which is a class utilized to extend the capabilities of servers. (Sun, 2011) The Servlet Runner is a functional utility for debugging and testing Java servlets before deploying them to web server. (Alexander A., 2009)

The first important "should do" thing is to extend our all the test case classes with org.apache.cactus.ServetTestCase so that the Servlet Runner can run the test case class written for a web application. In the setUp() method of our test case class, we should establish a session for a JSF page through JSFSession API. By calling the methods getJSFClientSession() and getJSFServerSession() over JSFSession, we can obtain a client side session and a server side one. (Master The Boss, 2011) One thing to notice is that the argument we have given to the constructor of JSFSession is the "/index.xhtml" and we have configured the web application's deployment descriptor (i.e. web.xml) to map the page to "/index.jsf".

```
@Override
public void setUp() throws IOException{
  JSFSession jsfSession = new JSFSession("/index.xhtml");
  client = jsfSession.getJSFClientSession();
  server = jsfSession.getJSFServerSession();
}
```

The test method testGetCurrentViewId() can be used for testing the identifier of the current application view. After establishing a session to "/index.xhtml", the final view displayed will be "/index.jsf" and can assert whether the view identifier is correct.

```
public void testGetCurrentViewId() throws IOException{
      assertEquals("/index.jsf", server.getCurrentViewID());
}
```

To identify components inside an application form, "formId:componentId" identifier can be used. For example, if the form id is "testForm" and the identification number of a component displayed in a form is "testComponent", then for identifying this component "testForm:testComponent" should be passed to the method findComponent() for proper identification.

```
public void testFormComponent() throws IOException{

  UIComponent formComponent = server.findComponent("testForm");
  assertNotNull(formComponent);
  assertTrue(formComponent instanceof HtmlForm);

  UIComponent textComponent = server.findComponent("testForm:testComponent");
  assertNotNull(textComponent);
  assertTrue(textComponent instanceof HtmlInputText);
}
```

To run the test-cases and see the test results on the browser, we have to make use of Servlet Test Case Runner. The Servlet Test Case Runner class expects the test case class name as the parameter “suite” and the xsl file name for formatting the test results as the parameter “xsl” where the xsl file is used for formatting the test results.

The combined query string becomes http://localhost:8080/(project)/ServletTestRunner?suite=(package).(TestClass)&xsl=(fileName).xsl where words in parentheses should be your own specific data. It can be used to run the test cases and the results will get displayed in the browser.

Users can manually carry out tests without using JSFUnit, which saves your time with its automated testing mechanism (Figure 4) from carrying out tests manually (Figure 3).

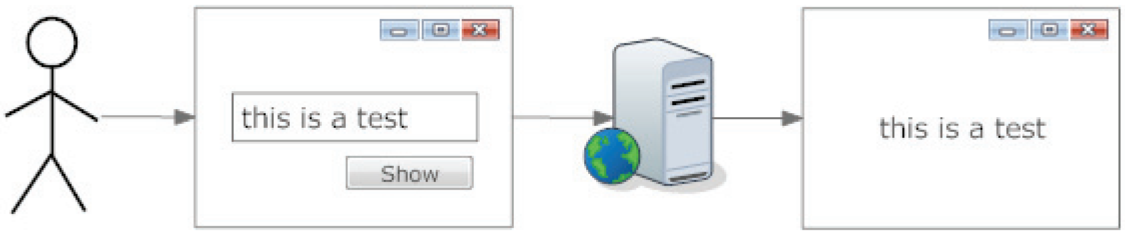

Figure 3. User carrying out tests manually

As simply mentioned before, the setUp() method of a test case class should be overridden for acquiring a session for the application’s page like "/index.xhtml". The application is displaying the index page as above given example (Figure 3) and in the first test case, you can set values of components by calling setValue(parameterName, parameterValue) that will simulate as if the user is manually typing the values for the given parameter fields. Subsequently, the request submission can be simulated by calling the method click() defined on the client session object. As soon as the request is submitted and the response is received, the client can call the method getPageAsText() which will return a raw HTML page text. Finally, the page content is validated for correctness by comparing it with the returned value from getPageAsText() method. (Figure 4)

```
//XHTML Code Snippet
<h:inputText value=”#{Test.text}” id=”testTextBox”/>
<h:commandButton value=”Show” id=”showButton” action=”#{Test.showAction}”/>

//Test Case Class Code Snippet
JSFClientSession client = jsfSession.getJSFClientSession();
client.setValue(“testTextBox”, “this is a test”);
client.click(“showButton”);
String pageText = client.getPageAsText();
assertTrue(pageText.contains(“this is a test”));
```

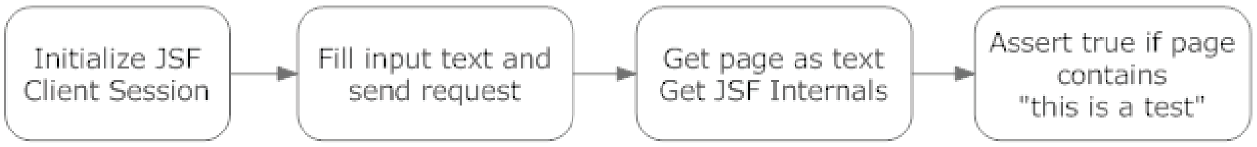
Initialize JSF Client Session
Fill input text and send request
Get page as text Get JSF Internals
Assert true if page contains "this is a test"

Figure 4. JSFUnit automatic test steps

One of the other API support of JSFUnit framework is adding request listener. A Request Listener intercepts client request and allows you to customize request by injecting, before and after the invocation of client request.

```
class OwnListener implements OwnListener{
       public void afterRequest(WebResponse request) {
              System.out.println("After Request.");
       }

       public void beforeRequest(WebRequestresponse) {
              System.out.println("Before Request.");
       }
}
```

A JSF application has built-in notification system, can display messages that can be warning, information, fatal or an error. JSFUnit framework provides support for iterating over the messages upon response completion for validating its contents.

```
public void testGetFacesMessages(){
  Iterator<FacesMessage> itr = server.getFacesMessages("testComponentId");
       while(iterator.hasNext()){
              FacesMessage facesMessage = itr.next();
              Assert.assertNotNull(facesMessage);
              Assert.assertEquals(FacesMessage.SEVERITY_ERROR,
              facesMessage.getSeverity());
       }
}
```

3. Conclusion

Testing is not daunting software process if you apply a pattern or guideline. Having highlighted the importance of testing, we tried to describe how JSFUnit can be easily integrated into an educational web application utilizing JSF Framework by following some steps. We also mentioned how to write simple JSF test cases with a simple example which will display a JSF page that will return static HTML content to the browser. Finally, basic discussions and experiences were given with respect to configuring and running test cases.

References

Alexander, A. (2009). Testing your Java servlets with the Servlet Runner, available from: http://www.devdaily.com/java/edu/pj/pj010025/

Bergsten, H. (2004). Java Server Faces, O'Reilly Media, ISBN 978-0-596-00539-9

Cheon Y. &Leavens G.T. (2002) A simple and practical approach to unit testing: The JML and JUnit way, 16th European Conference on Object Oriented Programming (ECOOP 2002); LECTURE NOTES IN COMPUTER SCIENCE 2002, Volume:2374, 231-255.

Dewitt, D., & Siraj, S. (2010). Learners perceptions of technology for design of a collaborative mLearning module. *World Journal on Educational Technology, 2*(3), 169-185.

Hong-qing G., & Hao W. (2009) Design and Implementation of Educational Resources Grid Portal Based on JSF, 2nd International Symposium on Knowledge Acquisition and Modeling: KAM 2009, VOL 3, 11-15

Klein, H. "ColfFusion unit testing framework: Paul Kenney's version.", ColdFusion Developer's Journal, Sept 2004 Issue

Kurniawan B. & Xue J.L. (2004) A comparative study of web application design models using the Java technologies, Advanced Web Technologies And Applications, Volume: 3007, pp. 711-721

Master The Boss (2011) . JSFUnit tutorial, available from: http://www.mastertheboss.com/

Shunmuga, R. (2007). Introduction to Java Server Faces, available from: http://www.javabeat.net/

Silvert, S. (2011). JSFUnit Overview, available from: http://www.jboss.org/jsfunit

Sun, (2011). What is Servlet, available from: http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets2.html

Tremblay G. &Labonte T. (2003) Semi-automatic marking of Java programs using JUnit, International Conference on Education and Information Systems - Technologies and Applications (EISTA 03) Proceedings, pp. 42-47.

Zhiming, Z., Jiangang, D. &Changgen J. (2011). Design and implementation of S2SH unit testing based on JUnit, 2nd International Conference on Education and Sports Education, ESE 2011, VOL III, 493-496.