**CONTINUOUS IMPROVEMENT PROCESS IMPLEMENATION FOR PERSHING SECURITY FRAMEWORK**

A PROJECT REPORT

*Submitted by*

**Saranya.P (111712104089)**

**Vaishni.R (111712104113)**

***In partial fulfilment for the award of the degree***

*Of*

BACHELOR OF ENGINEERING

IN

**COMPUTER SCIENCE AND ENGINEERING**

RMK ENGINEERING COLLEGE, KAVARAIPETTAI

ANNA UNIVERSITY :: CHENNAI 600 025

APRIL 2016

**ANNA UNIVERSITY::CHENNAI – 600 025**

**BONAFIDE CERTIFICATE**

Certified that this project report “**CONTINUOUS IMPROVEMENT PROCESS IMPLEMENTATION FOR PERSHING SECURITY FRAMEWORK**” is the bonafide work of  **SARANYA[111712104089] , VAISHNI[111712104113]** who carried out the project under my supervision.

|  |  |
| --- | --- |
| **SIGNATURE**  **DR. K.L.SHANMUGANATHAN, M.E.,M.S.,Ph.D.,** HEAD OF THE DEPARTMENT  PROFESSOR  Department of Computer Science and  Engineering,  R.M.K Engineering College,  Kavaraipettai - 601 206. | **SIGNATURE**  **Mrs. C.GEETHA, M.E., (Ph. D)**  SUPERVISOR  ASSOCIATE PROFESSOR  Department of Computer Science and Engineering,  R.M.K Engineering College,  Kavaraipettai - 601 206. |

Submitted to Project Viva Voce Examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **EXTERNAL EXAMINER** | **INTERNAL EXAMINER** |

**ABSTRACT**

The objective of the project is to understand the pershing security framework and provide continuous improvement and implementation support. Having that ,we would need to develop an application that is going to consume this framework which checks for user authentication and authorization and provide access to resources based on their roles alloted. For continuous improvement and implementation we would need to come up with the junit modules for different use cases supported by the pershing security framework which would provide faster result execution and feedback.This will be automated in such a way that these modules will be invoked during build time and will check for any gaps programmatically . It will publish the results and also notify through email the results of every execution. Built on top of Continuous Integration are Continuous Deployment/Delivery where after a successful test run your instantly and automatically release the latest version of your codebase. Makes deployment a non-issue and helps you speed up your development.

**ACKNOWLEDGEMENT**

Support on demand, encouragement at the needed moment and guidance in the right direction are in dispensable for the success of any project. We have received these in excess from all corners from various people, we are glad to submit our gratitude to them.

We thank **Shri. R.S.Munirathinam, Chairman,** and **Shri. R.M.Kishore , Vice Chairman** of RMK group of Institutions for extending a generous hand in providing the best of resources to the college. Dr**. Elwin Chandra Monnie, the esteemed Head of our Institution** has been a source of motivation to all the staffs and students of our college. We are so much thankful to him. Our sincere thanks to **Dr. Mohammed Junaid, the Vice Principal** for giving the continuous support for the completion of our project.

Our sincere thanks to **Dr. K.L.Shanmuganathan, the Head of the Department** for his continuous support and motivation throughout our project.

We extend our profound gratitude to **Dr. K.Sethukarasi our Project Coordinator** and **Mrs. C.GEETHA our Guide** for her guidance, who has indeed been a polestar throughout the course of the project, we thank him for giving us full support to complete the project successfully.

Last, but not the least, we take this opportunity to thank all the **staff members** of the Department of Computer Science and Engineering. Regards to **our family, classmates and friends** who offered an unflinching moral support for completion of this project.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | **iii** |
|  | **LIST OF FIGURES** | **ix** |
|  | **LIST OF ABBREVIATIONS** | **x** |
| **1.** | **INTRODUCTION** |  |
|  | 1.1 Objective of the Project |  |
|  | 1.2 System Study |  |
|  | 1.2.1 Existing System  1.2.2 Proposed System |  |
| **2.** | **SYSTEM ANALYSIS** |  |
|  | 2.1 Scope  2.2 Purpose  2.3 Resources Required  2.4 Feasibility Study |  |
|  | 2.4.1 Technical Feasibility |  |
|  | 2.4.2 Economic Feasibility  2.4.3 Operational Feasibility |  |
|  | 2.5 System Specification |  |
|  | 2.5.1 Hardware Specification |  |
|  | 2.5.2 Software Specification |  |
| **3.** | **SYSTEM DESIGN** |  |
|  | 3.1 System Architecture |  |
|  | 3.2 Use Case Diagram |  |
|  | 3.3 Activity Diagram |  |
|  | 3.4 Collaboration Diagram |  |
| **4.** | **TECHNOLOGIES USED** |  |
|  | 4.1 Javascript  4.2 Maven  4.3 Junit  4.4 Java7/J2EE  4.5 Eclipse  4.6 Jenkins |  |
| **5.** | **PROJECT DESCRIPTION**  5.1 Modules Description |  |
| **6.** | **TESTING OF THE PROJECT**  6.1 Testing  6.2 Unit Testing  6.3 Running unit test with Maven |  |
| **7.** | **IMPLEMENTATION AND MAINTENANCE**  7.1 Implementation  7.2 Maintenance |  |
| **8.** | **CONCLUSION**  8.1 Appendix(Screenshots)  8.2 References |  |
|  |  |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **TITLE** | **PAGE NO.** |
| 3.1 | System Architecture |  |
| 3.2 | Use Case Diagram |  |
| 3.3 | Activity Diagram |  |
| 3.4 | Collaboration Diagram |  |
| 5.1 | Sample login page |  |
| 5.2 | Login failure |  |
| 5.3 | Login success |  |

**LIST OF ABBREVIATIONS**

|  |  |  |
| --- | --- | --- |
| 1. | SDLC | Software Development Life Cycle |
| 2. | PSF | Pershing Security Framework |
| 3. | API | Application Program Interface |
| 4. | JDBC | Java Database Connectivity |
| 5. | IDE | Integrated Development Environment |
| 6. | POM | Project Object Model |
| 7. | XML | Extensible Markup Language |
| 8. | JSP | Java server page |
| 9. | EJB | Enterprise JavaBeans |
| 10. | GUI | Graphical User Interface |
|  |  |  |

**CHAPTER 1**

**INTRODUCTION**

**1.1 OBJECTIVE OF THE PROJECT**

The main objective of the project is to develop a sample application that uses pershing security framework by using Maven and developing junit modules for the Pershing Security Framework to provide continuous improvement and implementation .Continuous improvement and implementation is is the practice of running your tests on a non-developer machine automatically every time someone pushes new code into the source repository. This has the tremendous advantage of always knowing if all tests work and getting fast feedback. The results of the test execution are also automatically emailed to the recipients mail address. We can also schedule to run the test for a defined interval of time continuously thus it provides continuous improvement and implementatiuon support.

* 1. **SYSTEM STUDY**

The system study phase involves the understanding of the Pershing security framework to provides continuous improvement and implantation support. Writing up junit modules such that they are automatically build and configure the test and provides faster feedback (i.e) the results of execution whether it is a success or failure.

**1.2.1 EXISTING SYSTEM**

Authorization Framework for intelligently controlling [access](http://whatis.techtarget.com/definition/access) to computer resources, enforcing policies, auditing usage, and providing the information necessary for services. These combined processes are considered important for effective network management and security.  The server compares a user's authentication credentials with other user credentials stored in a database. If the credentials match, the user is granted access to the network. If the credentials are at variance, authentication fails and network access is denied.

The application will always have some improvements and releases for further enhancements. We cannot ensure that all the methods work without any gap after some changes have been updated. Whenever there is a gap, it causes failure and it requires a manual attention to correct them.This kind of manual testing becomes very tedious when the point of error cannot be encountered which is the major drawback of the existing system.

1.2.2 **PROPOSED SYSTEM**

With the good understanding of pershing security framework we have come up with a sample application for login using Maven that consumes the framework.For all the methods employed in the we are going to write junit modules . This will be automated in such a way that these modules will be invoked during build time and will check for any gaps programmatically.It provides continuous improvement and implementation support in such a way that whenever a user pushes into the repository , they are build automatically and runs the test cases .

The testcases are checked for the assertion condition and based on it success or failure is determined. It helps the developer to know that the code changed has the following effects and the changes made are reflected then and then. The results of the execution are also automatically mailed to the user and thus provides faster feedback. We can also schedule to run the test for a defined interval of time continuously thus providing continuous implementation support. This also avoids any human intervention for validation since it is automated and provides accurate results and feedback.

**CHAPTER 2**

**SYSTEM ANALYSIS**

Systems are created to solve problems. One can think of the systems approach as an organized way of dealing with a problem.The analysis phase is the second phase of the SDLC and to describe if the project should go ahead with the resources available.This also includes looking at any existing system to see what it is doing for the organization and how well that system is doing its job .The feasibility of the project is also considered and the group has to ask questions such as

* Can this system be created with the resources we have?
* Will this system significantly improve the organization?
* Does the old system necessarily need to be enhanced?

**2.1 SCOPE**

Perform continuous improvement process for Pershing Security Framework by developing junit modules for all the uses cases involved. This can be used for providing an automated process.

* 1. **PURPOSE**

This project is to develop an automated process in such a way that these modules will be invoked during build time and will check for any gaps programmatically. Hence avoiding any human intervention for validation.

* 1. **RESOURCES REQUIRED:**

This phase we need to analyze the availability of the resources that are required to design, develop and implement and test the project. The resources need to be analyzed are manpower, time and the system requirements. Teams of individual member are involved in the entire SDLC lifecycle except the testing phase.The testing phase is guided by implementing junit modules which will be invoked during build time for checking any gaps programmatically.

* 1. **FEASIBILTY STUDY**

Feasibility study is an evaluation of system regarding to its workability , impact on organization, ability to meet user needs and effective use of resources. All projects are feasible given unlimited resources and infinite time.The feasibility study is to determine whether the solution is achievable,given the organizations resource and constraints. By performing the feasibility study the scope of the system will be defined completely. It is both necessary and prudent to evaluate the feasibility of the project at the earliest possible time. Feasibility and risk analysis is related in many ways. If project risk is great the feasibility listed below is equally important.

The following feasibility techniques has been used in this project

* Technical feasibility
* Operational feasibility
* Economic feasibility

**2.4.1Technical Feasibilty**

Technical feasibility analysis makes a comparison between the levels of technology available that is needed for the development of the project. The level of technology consists of factors like software tools, machine environment ,platform developed and so on. Technical feasibility centers around the hardware and software and to what extent it can be deployed to run successfully. The tools that are used to develop the application are the best tools available in the technological scenario and hence it requires efficient and versatile programmers and programming skills .Maven , junit plugin ,oracle database and an IDE are sufficient for the project to be technically feasible.

**2.4.2 Economic Feasibilty**

Economic feasibility is used for evaluating the effectiveness of the system. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/ benefits analysis.All the softwares involved in the project are open source. So it can be easily downloaded fom the internet.So no high cost is involved. Regarding the maintenance , since the source code will be with us , any small and necessary changes can be done with minimum maintenance cost involved in it. Hence the system is economically feasible and the risk of finance doesn’t exist.

**2.4. 3 Operational Feasibilty**

Operational feasibility is necessary as it ensures the success of the project. Certain tests have been carried out to ensure the operational feasibility of the system. The proposed system operates well with all types of connection. All the capabilities work well if proper functioning environment is provided. The developed system is operatable by a range of people and it is consistent .Hence the system is operationally feasible.

**2.5 SYSTEM SPECIFICATION**

**2.5.1 HARDWARE SPECIFICATIONS**

Processor : Any processor above 500 MHz

Ram : 1 GB

Hard Disk : 50 GB

Input device : Standard Keyboard and Mouse.

Output device : Display Screen

**2.5.2 SOFTWARE SPECIFICATIONS:**

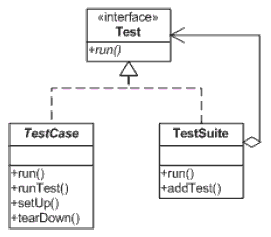
* Eclipse IDE
* Maven
* Junit plugin
* Oracle database
* Jenkins

**CHAPTER 3**

**SYSTEM DESIGN**

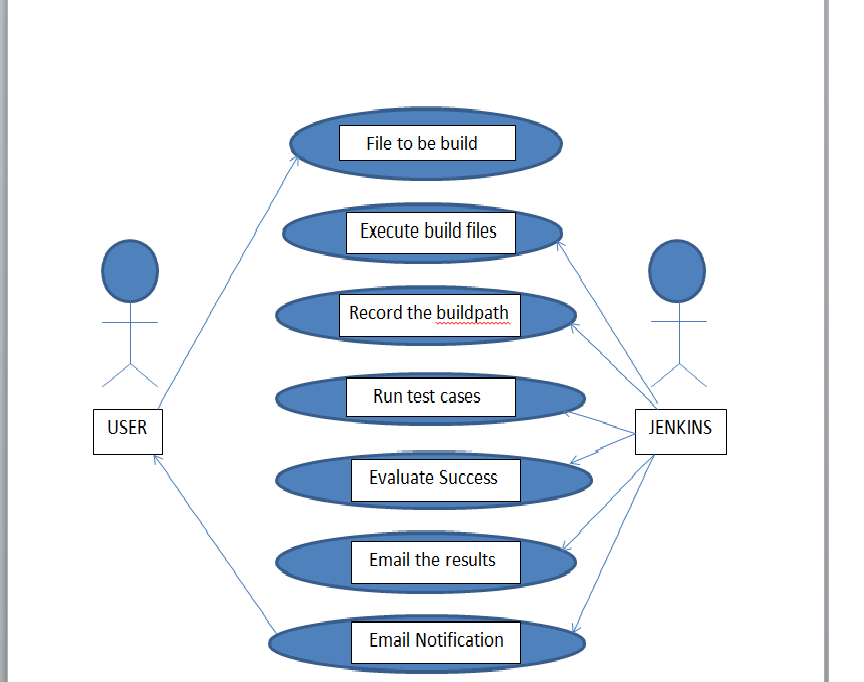
System design is the process or art of defining the hardware and software architecture, components , modules, interfaces and data for a computer system to satisfy specified requirements. One could see it as an application of system theories to computing. Some overlap with the discipline of system analysis appears inevitable.Design tools such as Unified Modelling Language (UML), now addresses some of the issues of computer systems design and interfacing.

**JUNIT ARCHITECTURE**

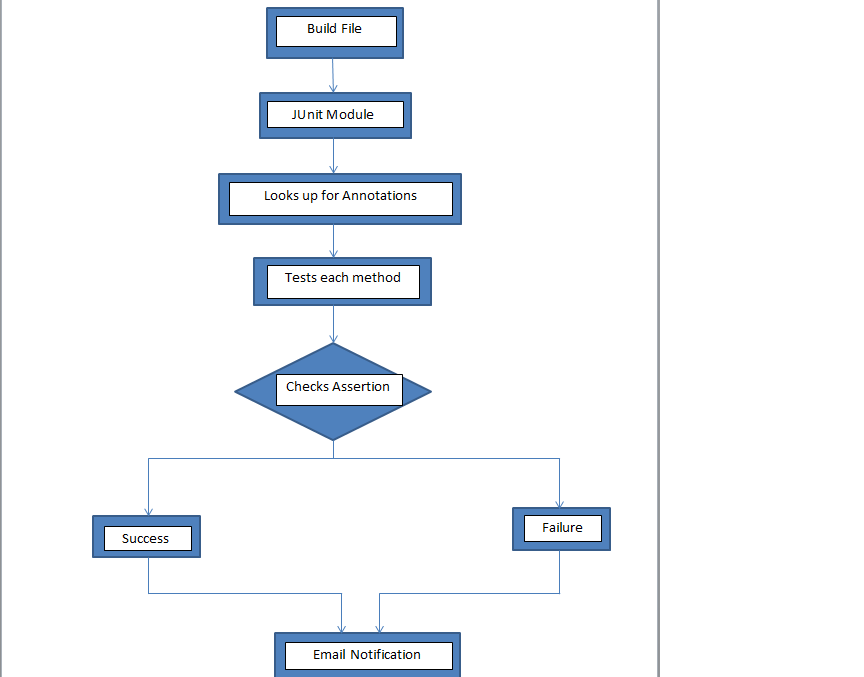


* 1. **Use Case Diagram**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

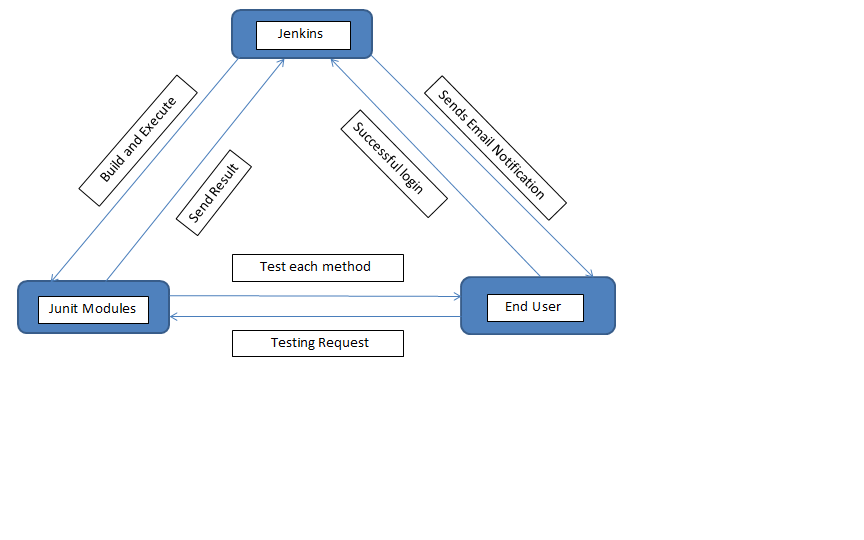


**3.2ACTIVITY DIAGRAM**

****

**3.3Collaboration Diagram**

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between the objects are shown as lines connecting the rectangles. The messages between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing.



**CHAPTER 4**

**TECHNOLOGIES USED**

**4.1 Javascript**

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform. JavaScript ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on website

**4.2 MAVEN:**

Apache maven is a software management and comprehension tool. Based on the concept of Project Object Model (POM) model. Maven can manage a project’s build ,reporting and documentation from a central piece of information.

**Mavens Objectives**:

Maven’s primary goal is to allow a developer to comprehend to comprehend the complete state of the development effort in the shortest period of time.In order to attain this goal there are several areas of concern that Maven attempts to deal with:

* Making the building process easy.
* Provide a uniform build system.
* Providing a quality project information.
* Providing guideline for best practices in development.
* Allowing transparent migration for new features.

**Features of Maven:**

* Simple project setup that follows best practices-get a new project or module started in seconds.
* Consistent usage across all project means that ramp up time for new developers coming onto a project.
* Super dependency management including automatic updating,dependency closures(also knowns as transitive dependencies).
* Able to work with multiple projects at the same time.
* A large and growing repository of libraries and metadata to use out of box, and arrangements in place with large open source projects for real time availability of their latest releases.

**4.3 JUNIT**:

JUnit promotes the idea of "first testing then coding", which emphasis on setting up the test data for a piece of code which can be tested first and then can be implemented . This approach is like "test a little, code a little, test a little, code a little..." which increases programmer productivity and stability of program code that reduces programmer stress and the time spent on debugging.

**Features**

* JUnit is an open source framework which is used for writing & running tests.
* Provides Annotation to identify the test methods.
* Provides Assertions for testing expected results.
* Provides Test runners for running tests.
* JUnit tests allow you to write code faster which increasing quality
* JUnit is elegantly simple. It is less complex & takes less time.
* JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
* JUnit tests can be organized into test suites containing test cases and even other test suites.
* Junit shows test progress in a bar that is green if test is going fine and it turns red when a test fails.

The packages in junit are

* **org.hamcrest**-The stable API defining matcher and its associated interfaces and classes.
* **org.hamcrest.core**-Fundamental matchers of objects and values and composite matchers
* **org.junit**-Provides junit core classes and annotations.
* **org.junit.runner**: Provides classes used to describe ,collect,run and analyse multiple tests.
* **org.junit.runner.manipulation**:Provide classes to filter or sort tests.
* **org.junit.runner.notification**:Provides information about a test run.
* **org.junit.matchers**:Provides useful matchers for use with the *Assert.assertThat(Object,org.hamcrest.Matcher)* statement.
* **org.junit.runners**: Provide standard runner implementations.

**Assertions**:

Junit provides overloaded assertion methods for all primitive types and Objects and arrays (of primitives or Objects). The parameter order is expected value followed by actual value. Optionally the first parameter can be a String message that is output on failure. There is a slightly different assertion,  assertThat  that has parameters of the optional failure message, the actual value, and aMatcher  object. Note that expected and actual are reversed compared to the other assert methods.

Some of the methods in junit are

* assertEquals(expected, actual)
* assertEquals(message, expected, actual)
* assertEquals(expected, actual, delta)
* assertEquals(message, expected, actual, tolerance ) -Test that float or double values match. The tolerance is the number of decimals which must be the same.
* assertFalse(condition)
* assertFalse(message, condition)
* Assert(Not)Null(object)
* Assert(Not)Null(message, object)
* Assert(Not)Same(expected, actual)
* Assert(Not)Same(message, expected, actual)
* assertTrue(condition)
* assertTrue(message, condition)
* assertArrayEquals(expectedArray, resultArray)

**JUNIT ANNOTATIONS:**

**@Test**: The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case. To run the method, JUnit first constructs a fresh instance of the class then invokes the annotated method. Any exceptions thrown by the test will be reported by JUnit as a failure. If no exceptions are thrown, the test is assumed to have succeeded

**@Test(timeout=100):** Somethimes we need to mesure the performance interms of time. The @Test annotations provides an optional parameter called 'timeout', which causes a test to fail if it takes longer than a specified amount of clock time (measured in milliseconds).

**@Before:** When writing tests, it is common to find that several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before the Test method. The @Before methods of super classes will be run before those of the current class.

**@After:** If you allocate external resources in a Before method you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method. All @After methods are guaranteed to run even if a Before or Test method throws an exception. The @After methods declared in superclasses will be run after those of the current class.

**@Ignore:** Sometimes you want to temporarily disable a test or a group of tests. Methods annotated with Test that are also annotated with @Ignore will not be executed as tests. Also, you can annotate a class containing test methods with @Ignore and none of the containing tests will be executed. Native JUnit 4 test runners should report the number of ignored tests along with the number of tests that ran and the number of tests that failed.

You can also use @Ignore annotation at class level.

**Features of Junit**:

* Fixtures
* Test Suite
* Test runners
* Junit Classes

**Fixtures**:

Fixtures are the fixed state of a set of objects used as a base line for running tests.

setup() method which runs before every test invocation.

teardown() method which runs after every test method.

**Test Suite**:

Test Suite is a bundle of few unit test cases and run it together.

@RunWith and @Suite annotation are used to run the suite test.

**Test Runner**:

Test runner used to run the test cases.

Eg: JUnitCore.runClasses(TestJunit.class)

**Junit Classes**:

Assert which contain a set of assert methods.

TestCase which contain a test case defines the fixture to run multiple tests.

TestResult which contains a method to collect the result of executing a test case.

**Junit using Maven**:

Dependency to be added to use junit in maven is

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

**4.4 Java 7/J2EE:**

Developers today increasingly recognize the need for distributed, transactional, and portable applications that leverage the speed, security, and reliability of server-side technology. **Enterprise applications** provide the business logic for an enterprise. They are centrally managed and often interact with other enterprise software. In the world of information technology, enterprise applications must be designed, built, and produced for less money, with greater speed, and with fewer resources.

With the Java Platform, Enterprise Edition (Java EE), development of Java enterprise applications has never been easier or faster. The aim of the Java EE platform is to provide developers with a powerful set of APIs while shortening development time, reducing application complexity, and improving application performance.

The Java EE platform is developed through the Java Community Process (JCP), which is responsible for all Java technologies. Expert groups composed of interested parties have created Java Specification Requests (JSRs) to define the various Java EE technologies. The work of the Java Community under the JCP program helps to ensure Java technology's standards of stability and cross-platform compatibility.

The Java EE platform uses a simplified programming model. XML deployment descriptors are optional. Instead, a developer can simply enter the information as an **annotation** directly into a Java source file, and the Java EE server will configure the component at deployment and runtime. These annotations are generally used to embed in a program data that would otherwise be furnished in a deployment descriptor. With annotations, you put the specification information in your code next to the program element affected.

In the Java EE platform, dependency injection can be applied to all resources a component needs, effectively hiding the creation and lookup of resources from application code. Dependency injection can be used in Enterprise JavaBeans (EJB) containers, web containers, and application clients. Dependency injection allows the Java EE container to automatically insert references to other required components or resources, using annotations.

**4.5 Eclipse**

Eclipse is an integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages through the use of plugins

**4.6 Jenkins**

Jenkins is an open-source continuous integration software tool written in the Java programming language for testing and reporting on isolated changes in a larger code base in real time. It is a server-based system running in a servlet Container such as Apache Tomcat It supports SCM tools including Accurev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase  and can execute Apache Ant and Apache Maven based projects as well as arbitrary shel scripts and Windows batch commands. The software enables developers to find and solve defects in a code base rapidly and to automate testing of their builds.Basically Continuous Integration is the practice of running your tests on a non-developer machine automatically everytime someone pushes new code into the source repository.

Jenkins offers the following major features out of the box, and many more can be added through  plugins

1. **Easy installation**: Just run java -jar jenkins.war, deploy it in a servlet container. No additional install, no database. Prefer an installer or native package? We have those as well.
2. **Easy configuration**: Jenkins can be configured entirely from its friendly web GUI with extensive on-the-fly error checks and inline help.
3. **Rich plugin ecosystem**: Jenkins integrates with virtually every SCM or build tool that exists.
4. **Extensibility**: Most parts of Jenkins can be extended and modified, and it's easy to create new Jenkins plugins. This allows you to customize Jenkins to your needs.
5. **Distributed builds**: Jenkins can distribute build/test loads to multiple computers with different operating systems. Building software for OS X, Linux, and Windows? No problem.

**CHAPTER 5**

**PROJECT DESCRIPTION**

Basically Continuous Integration is the practice of running your tests on a non-developer machine automatically everytime someone pushes new code into the source repository.This has the tremendous advantage of always knowing if all tests work and getting fast feedback.

The fast feedback is important so you always know right after you broke the build (introduced changes that made either the compile/build cycle or the tests fail) what you did that failed and how to revert it.

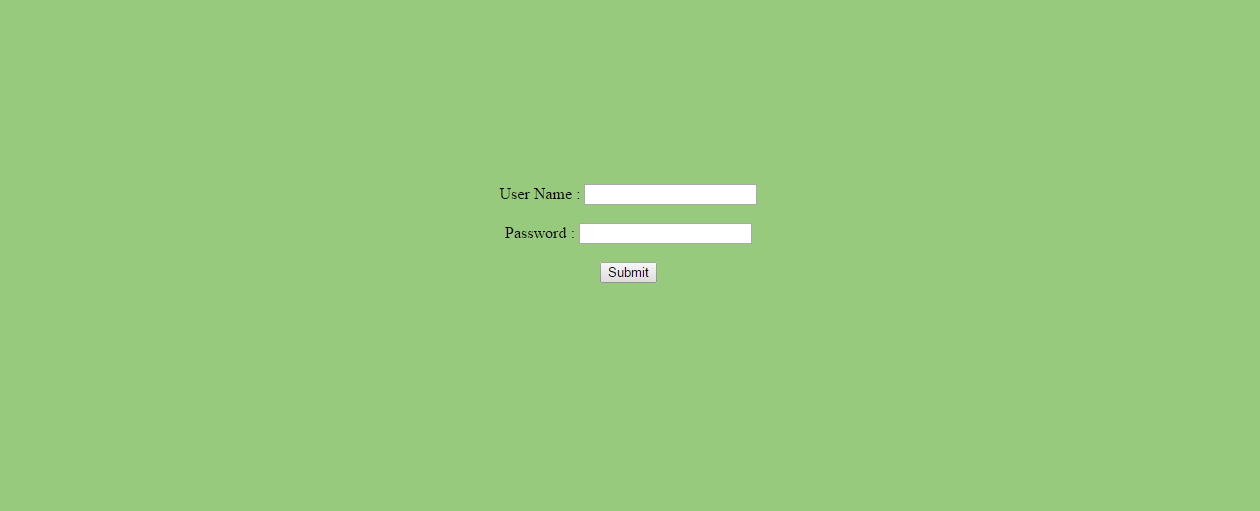
If you only run your tests occasionally the problem is that a lot of code changes may have happened since the last time and it is rather hard to figure out which change introduced the problem. When it is run automatically on every push then it is always pretty obvious what and who introduced the problem.

Built on top of Continuous Integration are Continuous Deployment/Delivery where after a successful test run your instantly and automatically release the latest version of your codebase. Makes deployment a non-issue and helps you speed up your development.

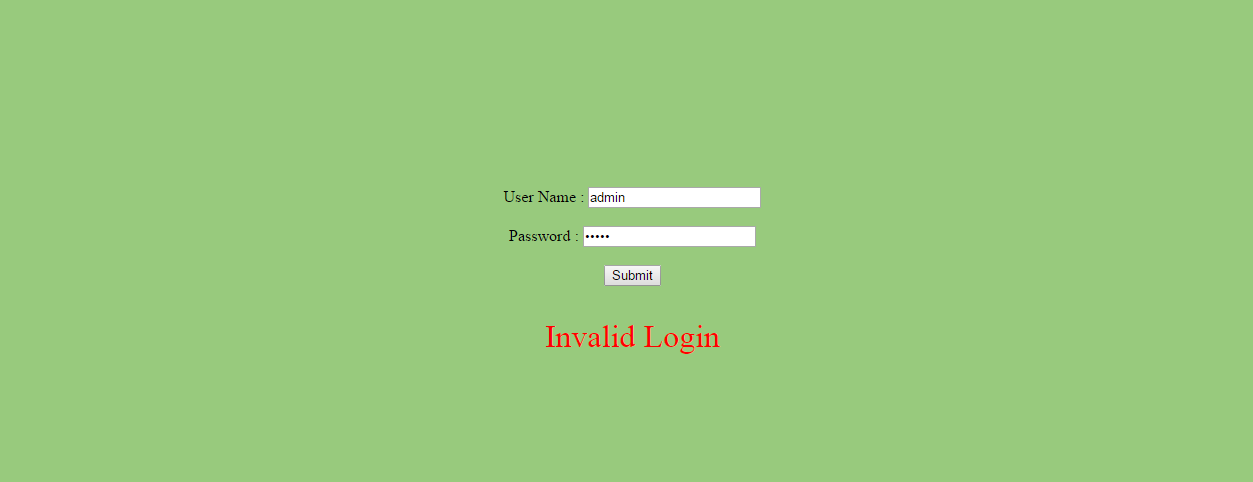
In this project we perform the following tasks

* A sample application for user login is created.
* While user authentication, (e.g. submitting a login form) an Authentication Object is created which holds the login credentials.
* In the backend ,The authentication object is passed to an AuthenticationManager, which can be thought of as the controller in the authentication process.
* The methods which were used in the authentication login are tested in junit and automated with the help of Jenkins such that it provides continuous support on execution when someone pushes into the repository they are build and the results are emailed.

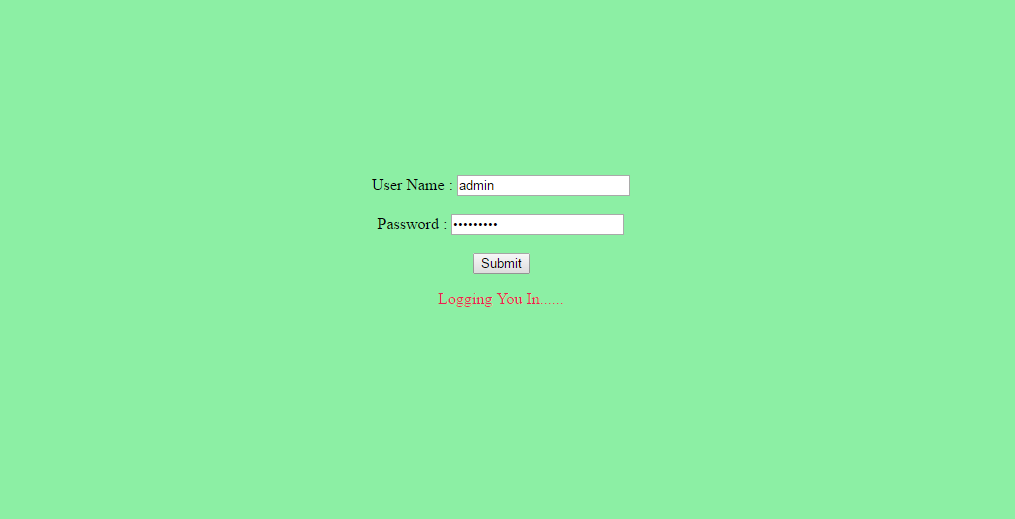
SAMPLE LOGIN FORM



**ON INVALID LOGIN NO ACCESS IS GRANTED:**



**SUCCESFUL LOGIN :**



**FETCHING ROLES OF USER**

You can't modify the server roles or add new server roles. A login can belong to more than one role and acquires a combination of the privileges of the roles it belongs to. This API is used to fetch the roles of a user by invoking the service. It will return a List of roles and the access granted by the roles.

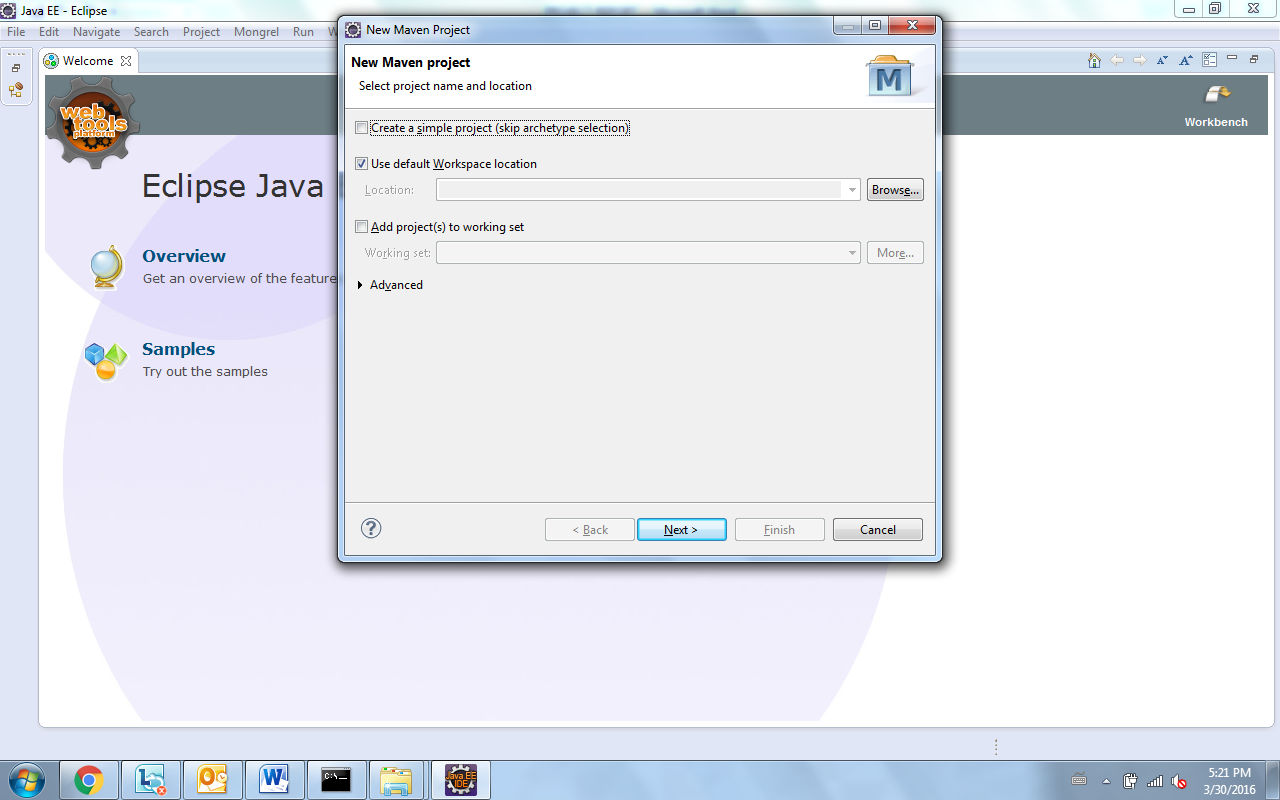
**REMOVING LOGIN OF A USER**

You'll eventually need to remove logins from the Server. Server prevents you from removing a login if the login is associated with the user name of a user who owns objects in any database. This rule prevents objects from being orphaned, which would be a violation of referential integrity in the database. If you follow good programming practices, only the database owner (dbo) can own objects, so deleting a user and login account won't cause any conflicts. Also, you can't remove a login that's in user.

**SESSION MANAGEMENT**

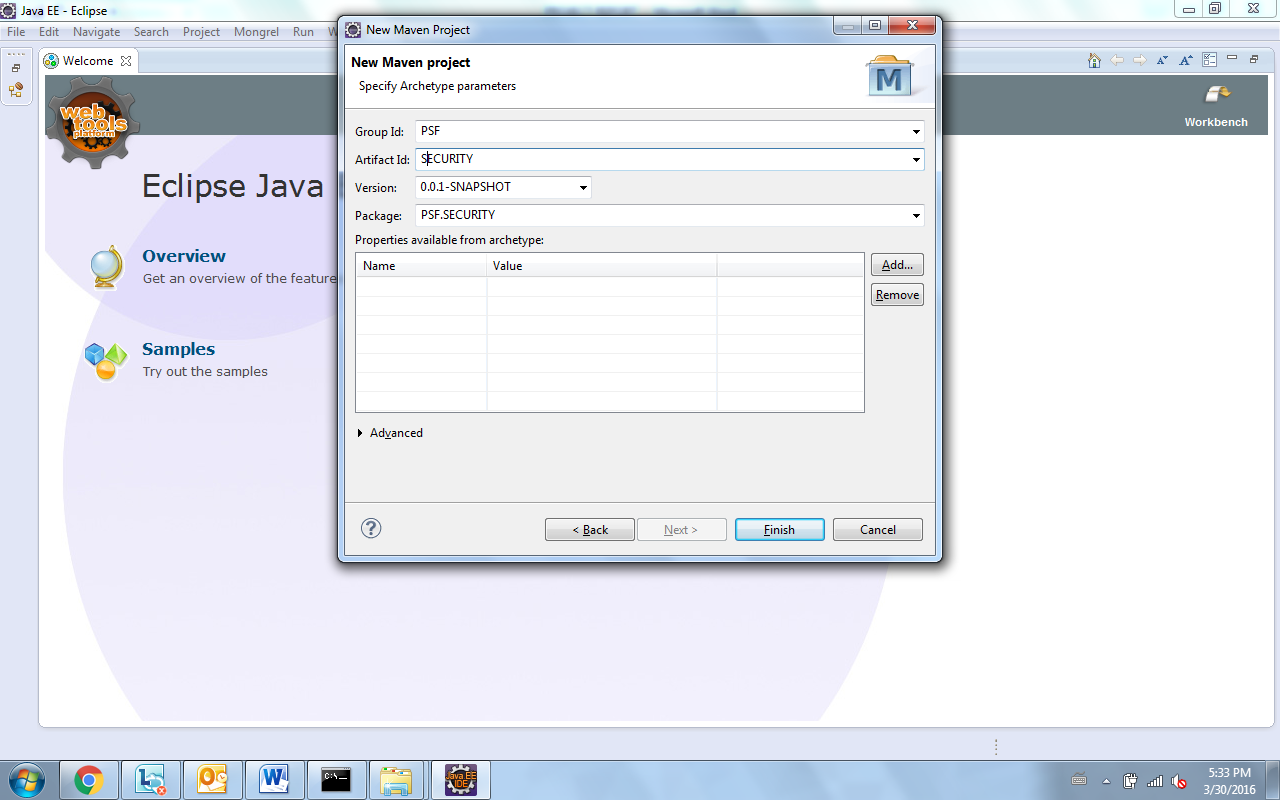
Once the user logins with the correct credentials a session is created for the user and it remains active until the session is closed. You will need to identify the maximum age of any given session ID as well as a timeout for sessions. There is often the requirement to re-authenticate users during a session for security reasons

**CREATING A MAVEN PROJECT :**



File-> new->Maven project is used to create a maven project. Click create a simple project (skip archetype selection) and then next and select webapp .

In the next screen enter the necessary details such as groupid, Artifactid, version and package.Click Finsh the maven project will be created.



**METHODS INVOLVED IN USER AUTHENTICATION AND SESSION MANAGEMAENT:**

**USER AUTHENTICATION CLASS**

* 1. fetchRoles()- It will fetch the roles for the valid user and checks for validation in the database.
  2. deleteRoles()- It will delete the role of the user from the database.
  3. settingRolesRequest()- It sends the request for setting up the roles and they are updated in the database.
  4. authenticateUser()- authenticates user based on the valid login credentials.

**USER SESSION MANAGEMENT CLASS**

1. createSession()- It is used for creating a session for the user when the user logins.
2. refreshSession()- It is used for refreshing the session for a given interval of time.
3. InvalidateSession()- Once the user logs out,the session is invalidated.

**WRITING JUNIT TESTCASES**

For each of the above discussed methods,we have to write testcases to check their functionality.

In the Junit testcase ,we have to create an object check for the class and with the help of the object we invoke the corresponding methods.

**OBJECT CREATION FOR USER AUTHENTICATION**

UserAuthentication abc = new UserAuthentication();

The junit annotations such as @test,@before , @after,@beforeClass,@afterClass are created . In @Test, we have to provide sample inputs parameters for testing the functionality . After providing the sample input parameters we check the functionality with the help of assertion methods.

The various assertion methods to be checked are

assertTrue(“Fetching Roles of the User”, abc.fetchRoles());

assertNotNull (“Fetching Roles of the User”, abc.fetchRoles());

assertNotNull (“Deleting Roles of the User”, abc. deleteRoles());

assertNull (“Deleting Roles of the User”, abc. deleteRoles());

**OBJECT CREATION FOR USER SESSION MANAGEMENT**

UserSessionManagement abc = new UserSessionManagement ();

The junit annotations such as @test,@before , @after,@beforeClass,@afterClass are created . In @Test, we have to provide sample inputs parameters for testing the functionality . After providing the sample input parameters we check the functionality with the help of assertion methods.

The various assertion methods to be checked are

assertTrue(“Fetching Roles of the User”, abc.CreateSessions());

assertNotNull (“Fetching Roles of the User”, abc.CreateSession());

assertNotNull (“Deleting Roles of the User”, abc. invalidateSessions());

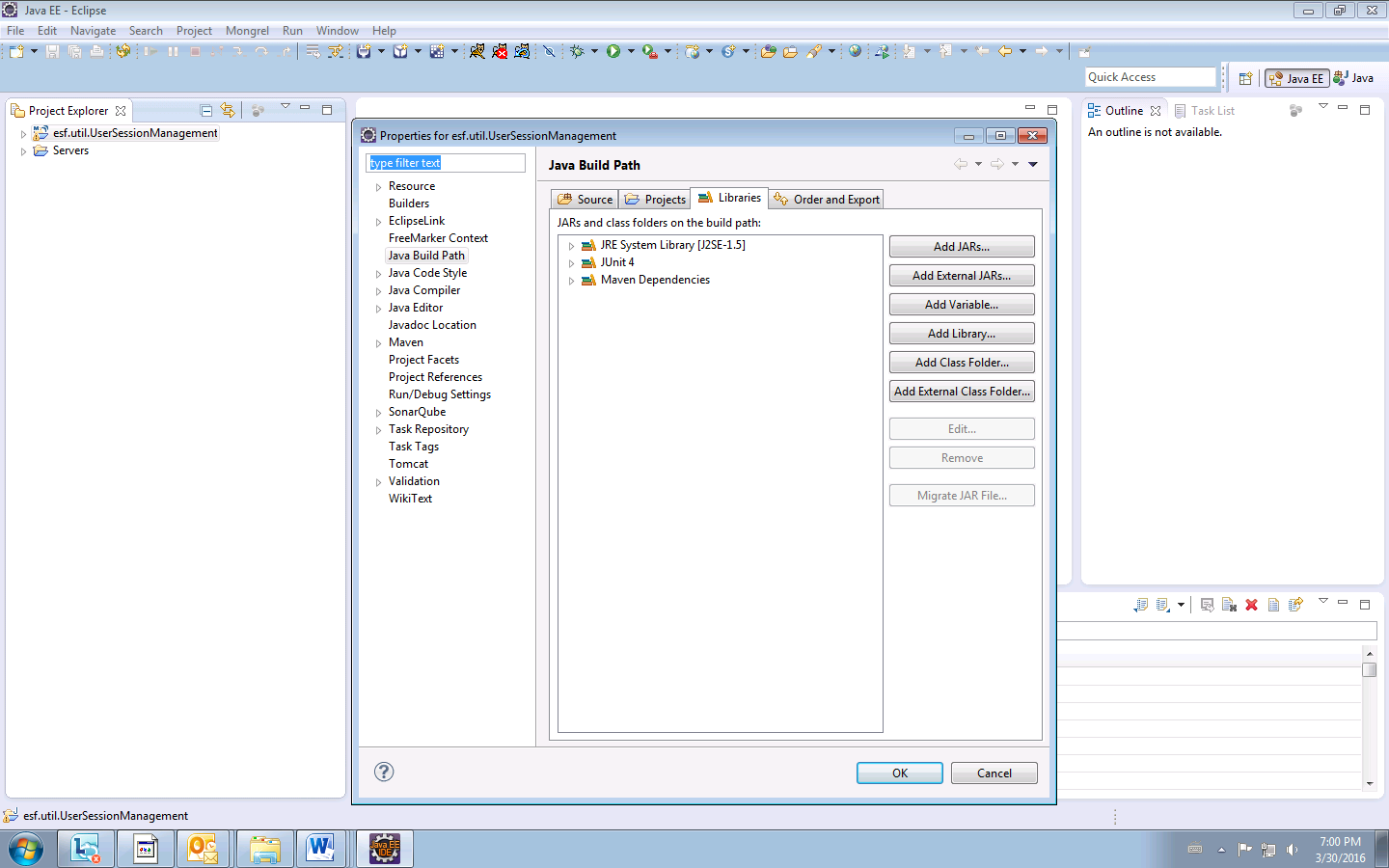
assertNull (“Deleting Roles of the User”, abc.invalidateSessions());

assertTrue(“Refreshing session”,abc.refreshSession());

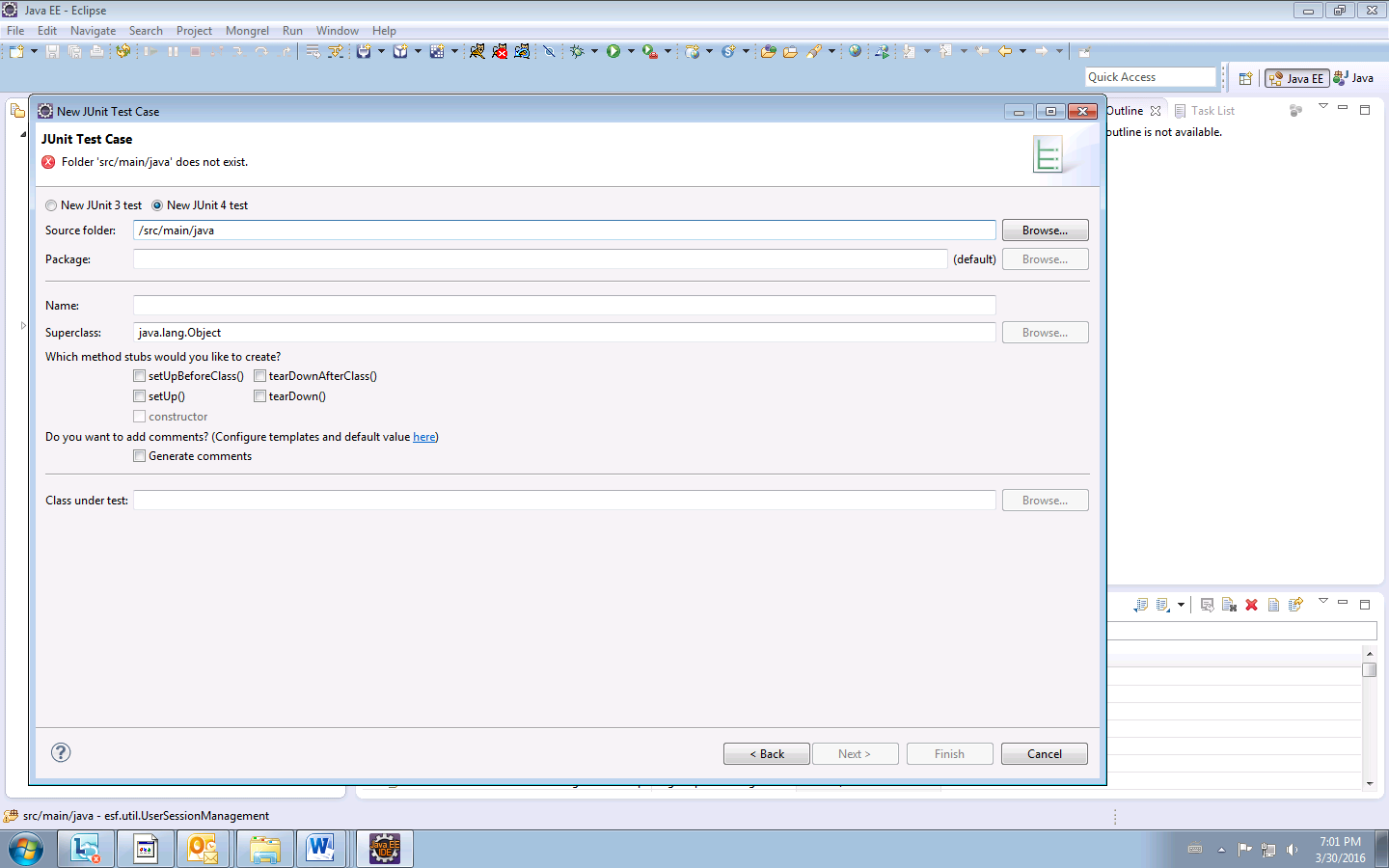
These assertions methods are validated and the result whether it is success or failure is obtained.

**Junit modules**

To write junit modules in eclipse, First configure Build path and ad external jars and select junit.jar



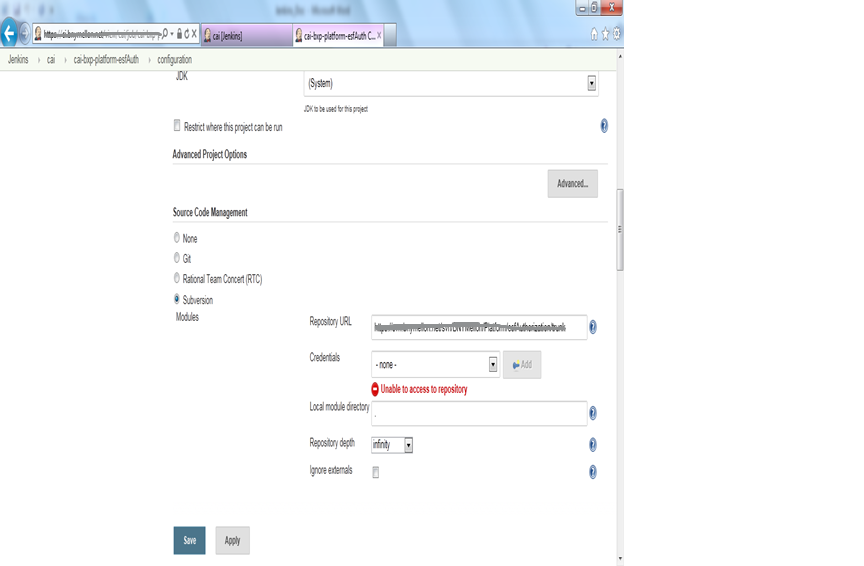
Once added click on the maven package and select File->New ->junit TestCases



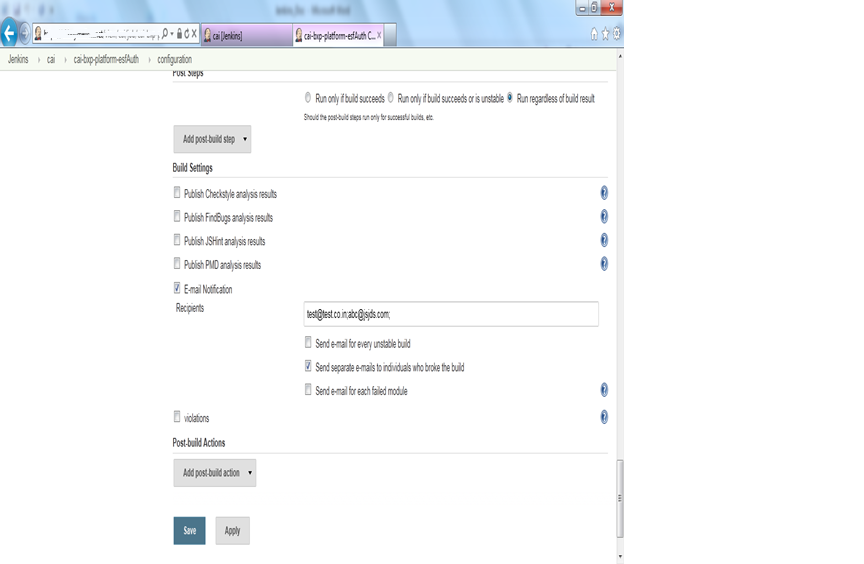
We are going to write junit testcase for an authentication login ,On successful execution of the testcase or failure of the testcase ,an email is generated and send automatically.This is automated in such a way that while running the junit test case, the result is emailed and the test case is made to run for a defined interval of time continuously with the help of jenkins.

**Jenkins tool for automation**:

Using this we can give the repositrory address ,it will automatically build and configure the junit modules and send an email on confirmation.



SENDING EMAILSTHE RESULTS OF EXECUTION:



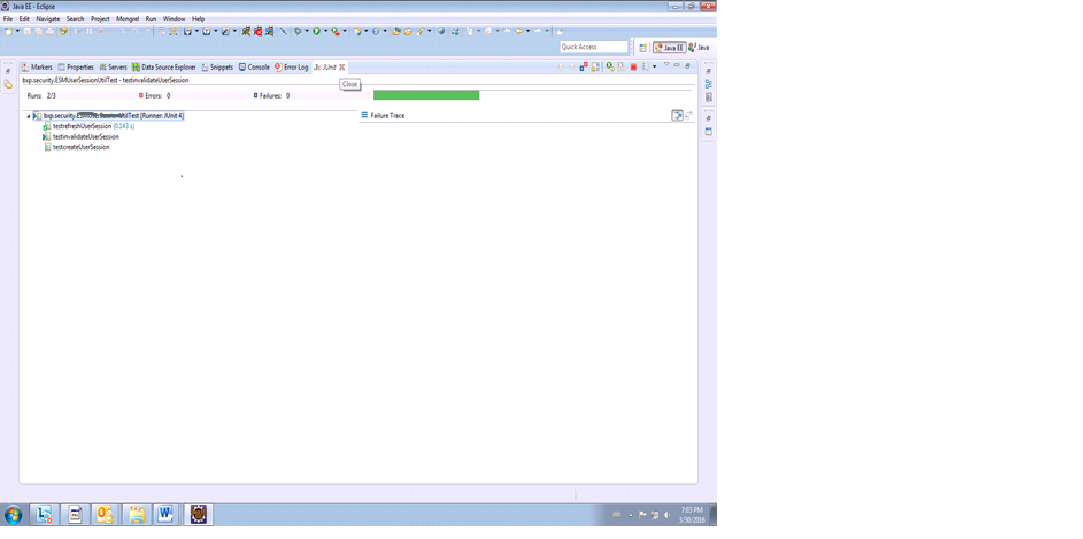
**JUNIT TEST CASES:**

For each of the method in the authentication package, we write testcases to check all the methods are working

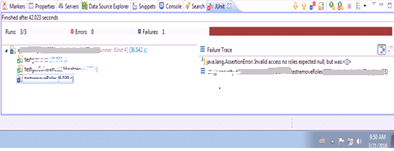
For usercreation and authentication we will pass the parameters of username and password .The session of each user and check whether they are executing and the results of execution are automatically mailed.

The fast feedback is important so you always know right after you broke the build (introduced changes that made either the compile/build cycle or the tests fail) what you did that failed and how to revert it.

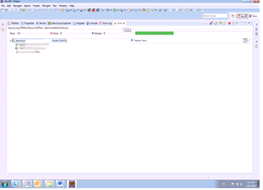
**EXECUTION OF JUNIT TESTCASES:**



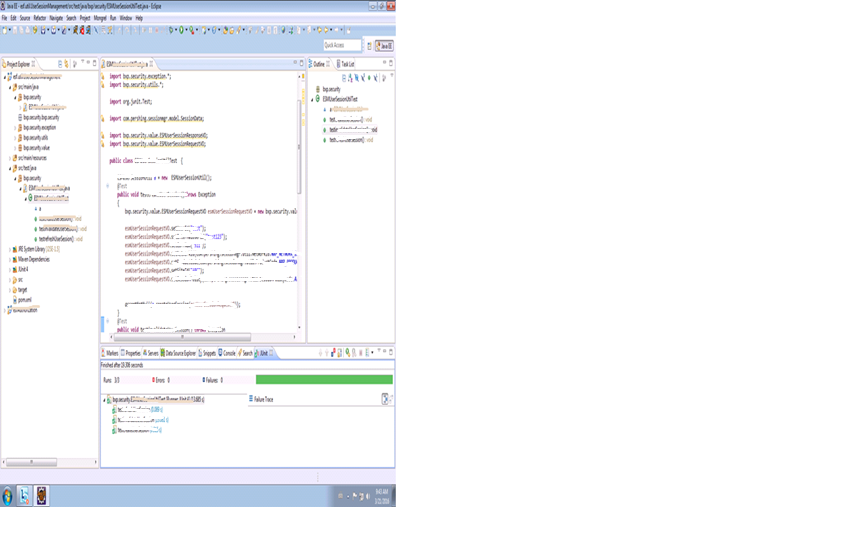
FAILURE OF A METHOD:



SUCCESS RUN OF A METHOD:

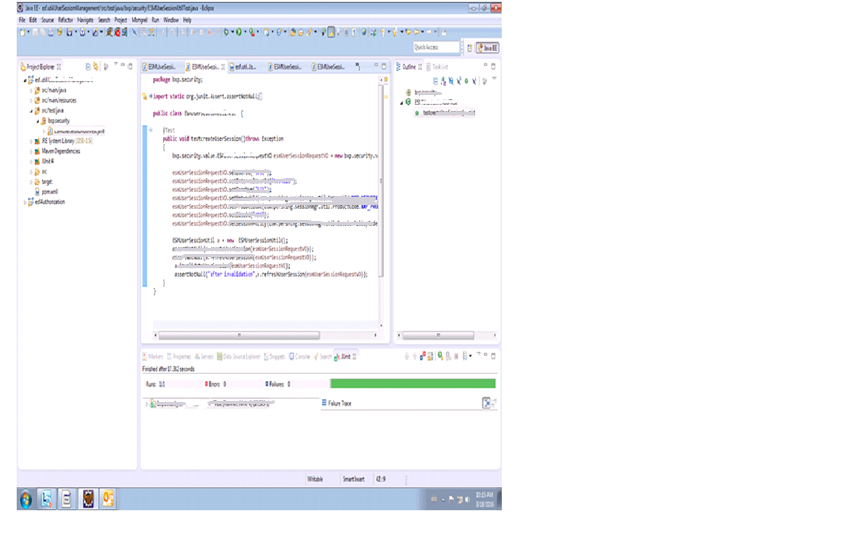
****

**Junit modules for session management**

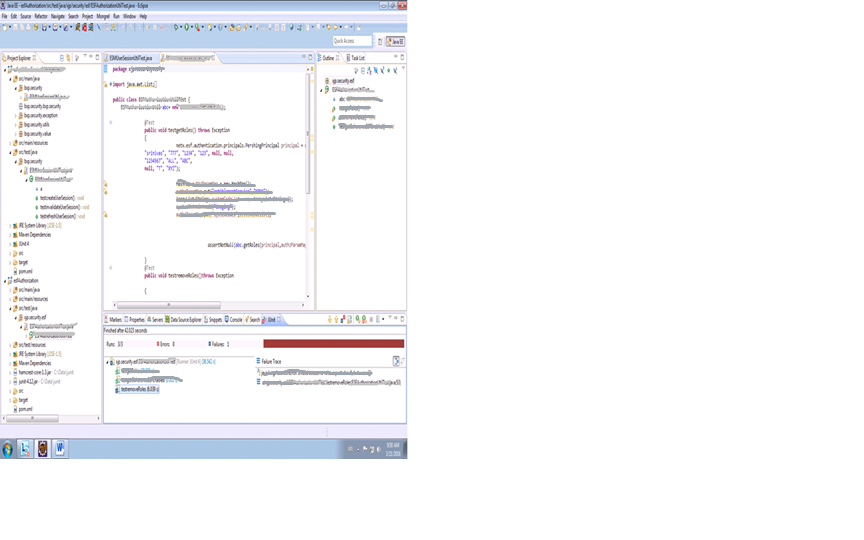


**JUNIT MODULES FOR AUTHORIZATION**

EXECUTING JUNIT TEST CASE FOR AUTHORIZATION AND EVALUATING THE SUCCESS AND FAILURE OF THE EXECUTION.



JNUIT MODULES FAILURE SCENARIO ON REMOVING USER :



Thus the results of the execution are emailed automatically and whenever there is a change in the source code , the junit test modules are automated and the reults are known fast.

**CHAPTER 6**

**TESTING OF THE PROJECT**

**6.1 TESTING**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies and/or a finished product. It is the process of exercising software with the intent to ensure that the software system meets its requirements and user expectations and does not fail unacceptable manner.

**6.2 UNIT TESTING**:

Unit testing is a level of software testing where individual units of software are tested.The purpose is to validate that each unit of the software performs as designed.A unit is the small testable part of a software .It usually has one or few inputs and a single output.In object oriented programming,the smallest unit is a method which may belong to a base/ super class,abstract class.Unit testing framework ,drivers, stubs are used to assist in unit testing.Unit testing is a first level of testing and performs prior to integration testing.Unit testing is normally performed by software developers themseleves or their peers or in some cases independent software testers.

**BENEFITS:**

* Unit testing increases confidence in changing/maintaining code.If a good unit tests are written and if they run every time any code is changed,we will be able to promptly catch any defects introduced due to the change.
* Also,if codes are already made less interdependent to make unit testing possible,the unintended impact of changes to any code is less.
* Codes are more reusable.Inorder to make unit testing possible,codes need to be modular.This means codes are easier to reuse
* The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.
* Debugging is easy.When a test fails,only the latest changes need to be debugged.With testing at higher levels,changes made over a span of several days/weeks/months need to be scanned.
* Codes are more reliable.

**6.3 Running unit test with Maven**

To run unit test via Maven, issue this command **: mvn test**

$ **mvn test**

[INFO] Scanning for projects...

[INFO] ------------------------------------------------------------------------

[INFO] Building junit-tests

[INFO] task-segment: [test]

[INFO] ------------------------------------------------------------------------

[INFO] [resources:resources {execution: default-resources}]

[WARNING] Using platform encoding (MacRoman actually) to copy filtered

resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO] [compiler:compile {execution: default-compile}]

[INFO] Compiling 1 source file to /Users/Tim/Library/Code/sonatype/

maven-cookbook/mcookbook-examples/unit/junit-tests/target/classes

[INFO] [resources:testResources {execution: default-testResources}]

[WARNING] Using platform encoding (MacRoman actually) to copy filtered

resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO] [compiler:testCompile {execution: default-testCompile}]

[INFO] Compiling 1 source file to /Users/Tim/Library/Code/sonatype/

maven-cookbook/mcookbook-examples/unit/junit-tests/target/test-classes

[INFO] [surefire:test {execution: default-test}]

[INFO] Surefire report directory: /Users/Tim/Library/Code/sonatype/

maven-cookbook/mcookbook-examples/unit/junit-tests/target/surefire-reports

-------------------------------------------------------

T E S T S

-------------------------------------------------------

Running org.sonatype.mcookbook.SeriousComponentTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.065 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 5 seconds

[INFO] Finished at: Wed Nov 25 14:42:39 CST 2009

[INFO] Final Memory: 17M/80M

[INFO] ------------------------------------------------------------------------

If the unit test fails the same file will contain a stack trace that points to a failed assertion on a specific line of the unit test as shown below:

-------------------------------------------------------------------------------

Test set: org.sonatype.mcookbook.SeriousComponentTest

-------------------------------------------------------------------------------

Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.062 sec

<<< FAILURE!

testSeriousness(org.sonatype.mcookbook.SeriousComponentTest) Time elapsed:

0.017 sec <<< FAILURE!

junit.framework.AssertionFailedError: null

at junit.framework.Assert.fail(Assert.java:47)

at junit.framework.Assert.assertTrue(Assert.java:20)

at junit.framework.Assert.assertTrue(Assert.java:27)

at org.sonatype.mcookbook.SeriousComponentTest.testSeriousness(

SeriousComponentTest.java:15)

**CHAPTER 7**

**IMPLEMENTATION AND MAINTENANCE**

**7.1 IMPLEMENATION**

This project is implemented in an organization to provide authorized user access to resources. It fetches the necessary information for different kinds of authorized user based on their roles and validate them .

**7. 2 MAINTENANCE**

System maintenance is the phase of software engineering that helps the user and the developer to maintain the software in an environment required by the user. . It is this phase in which any sort of implementation notification are made to incorporate the changes that the user need. The Maintenance Phase occurs once the system is operational. It includes implementation of changes that software might undergo over a period of time, or implementation of new requirements after the software is deployed at the customer location. The maintenance phase also includes handling the residual errors that may exist in the software even after the testing phase. This phase also monitors system performance, rectifies bugs and requested changes are made.

**CHAPTER 8**

**CONCLUSION**

The proposed system is found to be working efficiently and effectively. The system is developed with much attention over its quality and reliability. This system is found to be satisfactorily running under real environment.The system is perfectly tested for its efficiency,flexibility,accuracy and user friendly.The new system makes the task more secure and faster.

Any errors in the program will be detected during the running of junit testcase and the necessary changes are made it to run successfully. Automation is provided in such a way that these modules will be invoked during build time and will check for any gaps programmatically and the results are emailed simultaneously and provides faster feedback.

**REFERENCES :**

1. <https://books.sonatype.com/mcookbook/reference/unit-sect-junit>.
2. <http://faculty.washington.edu/stepp/courses/2005spring/tcss360/handouts/6-junit_eclipse_2>.
3. [https://www.informit.com/guides/content.aspx/java - junit reference 4](https://www.informit.com/guides/content.aspx/java%20-%20junit%20reference%204)
4. <http://frightanic.com/software-development/adding-references-junit-tests/>
5. <http://automatic-generation/methods-of-unit-test-cases-for-net-and-java>
6. <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>