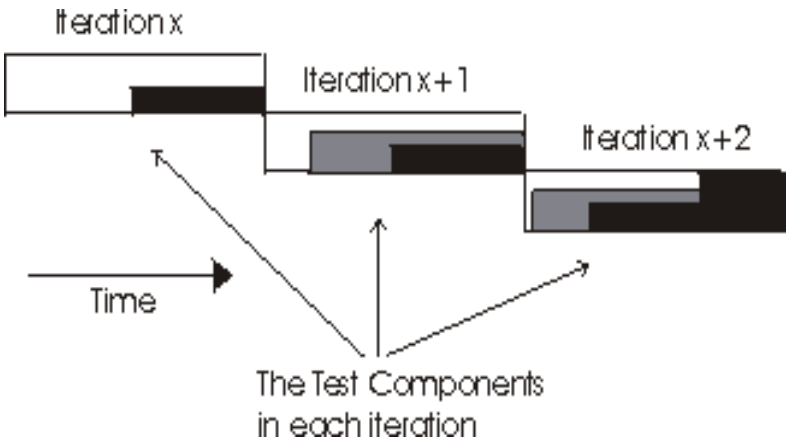
ABSTRACT

Bug can be defined as the abnormal behavior of the software. No software exists without a bug. The elimination of bugs from the software depends upon the efficiency of testing done on the software. A bug is a specific concern about the quality of the Application under Test (AUT).

The Lifecycle of Testing:

In the software development lifecycle, software is refined through iterations. In this environment, the testing lifecycle must also have an iterative approach with each build being a target for testing. Additions and refinements are made to the tests that are executed for each build, accumulating a body of tests, which are used for regression testing at later stages. This approach implies that it causes reworking the tests throughout the process, just as the software itself is revised. There is no frozen software specification and there are no frozen tests.



This iterative approach gives a high focus on regression test. Most tests of iteration X are used as regression tests in iteration X+1. In iteration X+2, you would use most tests from iteration X and iteration X+1 as regression tests, and the same principle would be followed in subsequent iterations.

Because the same test is repeated several times, it is well worth the effort to automate the tests. It becomes necessary to effectively automate your tests to meet your deadlines.

The testing lifecycle is a part of the software lifecycle; they should start at the same time. If not started early enough, the tests will either be deficient, or cause a long testing and bug-fixing schedule to be appended to the development schedule, which defeats the goals of iterative development. Furthermore, the test planning and design activities can expose faults or flaws in the application definition. The earlier these are resolved, the lower the impact on the overall schedule. Problems found during evaluation can be solved within this iteration, or postponed to the next iteration. There is always some "requirements creep" from iteration to iteration, something you need to be aware of and able to manage.

Bug Life Cycle:

In software development process, the bug has a life cycle. The bug should go through the life cycle to be closed. A specific life cycle ensures that the process is standardized. The bug attains different states in the life cycle. The life cycle of the bug can be shown diagrammatically as follows:

The different states of a bug can be summarized as follows:

- New
- Open
- Assign
- Test
- Verified
- Deferred
- Reopened
- Duplicate
- Rejected
- Closed

LIST OF FIGURES

1. INTRODUCTION

Vision: The purpose of Bug Tracking for improving software reliability is to provide better service to the administrator or useful for applications developed in an organization.

Scope: The Bug Tracking for Improving Software Reliability is a web based application that can be accessed throughout the organization. This system can be used for logging Bugs against an application/module, assigning Bugs to team members and tracking the Bugs to resolution. There are features like email notifications, user maintenance, user access control, report generators etc in this system.

Definition, Acronyms, Abbreviations

Bug - A software Bug (or just "Bug") is an error, flaw, mistake, failure, or fault in a computer program that prevents it from behaving as intended (e.g., producing an incorrect result). Most Bugs arise from mistakes and errors made by people in either a program's source code or its design, and a few are caused by compilers producing incorrect code.

Overview

Bug Tracking is the process of reporting and tracking the progress of Bugs from discovery through to resolution, where a Bug is defined as a deviation from requirements. Other terminology frequently used to describe this process include

- problem tracking
- change management
- fault management

Bug Tracking systems are most commonly used in the coding and testing phases of the software development process. However, tracking systems can in fact be used for many other purposes such as general issue tracking, simple task lists, help desk situations or contact management, where the focus is on the tracking aspect rather than what is being tracked. Even in software development, tracking systems are quite often not limited to simply tracking Bugs, but extended to track feature requests or enhancements as well as enquiries.

2. LITERATURE REVIEW

2.1 Problem Definition

Existing System

- The existing system consists of entering the details in the Microsoft Excel Sheets for the storing of the data.
- When a manager needs information of the employee he searches for the specified file in the file system.
- He opens the file and takes the information.
- Report Generation done manually by copying the content of the different files into another file.
- The Manually generated report was then printed.

Limitations in Existing System

- Information retrieval is a very big process.
- Lack of organization of the files may porn to information loss due to accidental deletion of files.
- No security because the files are visible to the users.
- Report generation will be a big task.

2.2 Present System

The existing system consists of entering the details in the Microsoft Excel Sheets for the storing of the data. When a manager needs information of the employee he searches for the specified file in the file system. He opens the file and takes the information. Report Generation done manually by copying the content of the different files into another file. The Manually generated report was then printed.

2.3 Proposed System

The Proposed system is a browser which is completely related to online system, which provides the centralized database. It stores Bugs data and description of the particular Bug data. It can also create Excel reports and PDF documents based on the information in its database.

Advantages over Existing System

- The performance is increased due to well designed database.
- Security is increased
- Time saving in report generation
- Easy to update the details

Feasibility Study

Economic Feasibility:

Economic feasibility attempts 2 weigh the costs of developing and implementing a new system, against the benefits that would accrue from having the new system in place. This feasibility study gives the top management the economic justification for the new system.

A simple economic analysis which gives the actual comparison of costs and benefits are much more meaningful in this case. In addition, this proves to be a useful point of reference to compare actual costs as the project progresses. There could be various types of intangible benefits on account of automation. These could include increased customer satisfaction, improvement in product quality better decision making timeliness of information, expediting activities, improved accuracy of operations, better documentation and record keeping, faster retrieval of information, better employee morale.

Operational Feasibility

Proposed project is beneficial only if it can be turned into information systems that will meet the organizations operating requirements. Simply stated, this test of feasibility asks if the system will work when it is developed and installed. Are there major barriers to Implementation? Here are questions that will help test the operational feasibility of a project: Is there sufficient support for the project from management from users? If the current system is well liked and used to the extent that persons will not be able to see reasons for change, there may be resistance. Are the current business methods acceptable to the user? If they are not, Users may welcome a change that will bring about a more operational and useful systems.

Have the user been involved in the planning and development of the project?
Early involvement reduces the chances of resistance to the system and in general and increases the likelihood of successful project. Since the proposed system was to help reduce the hardships encountered. In the existing manual system, the new system was considered to be operational feasible.

Technical Feasibility

Evaluating the technical feasibility is the trickiest part of a feasibility study. This is because, .at this point in time, not too many detailed design of the system, making it difficult to access issues like performance, costs on (on account of the kind of technology to be deployed) etc. A number of issues have to be considered while doing a technical analysis.

Understand the different technologies involved in the proposed system before commencing the project we have to be very clear about what are the technologies that are to be required for the development of the new system. Find out whether the organization currently possesses the required technologies. Is the required technology available with the organization?

3. SYSTEM REQUIREMENT SPECFICATION

3.1 Hardware and Software Specification

3.1.1 Operation System

Windows Operating System Is Used.

3.1.1.1 Supported Operating System

Windows Operating System

3.2 Minimum Software and Hardware

3.2.1 Minimum Software

Operating System	:	Windows XP/98/2003
User Interface	:	HTML, JSP
Client-side Scripting	:	JavaScript
Programming Language	:	Java
Web Applications	:	JDBC, JNDI, Servlets, JSP
IDE/Workbench	:	Editplus
Database	:	Oracle/Access
Server Deployment	:	Tomcat

3.2.2 Minimum Hardware

Processor	:	Pentium IV
Hard Disk	:	40GB
RAM	:	256MB

3.3 Technology Overview

HTML

HTML, an initialism of Hypertext Markup Language, is the predominant markup language for web pages. It provides a means to describe the structure of text-based information in a document — by denoting certain text as headings, paragraphs, lists, and so on — and to supplement that text with interactive forms, embedded images, and other objects. HTML is written in the form of labels (known as tags), surrounded by angle brackets. HTML can also describe, to some degree, the appearance and semantics of a document, and can include embedded scripting language code which can affect the behavior of web browsers and other HTML processors.

HTML is also often used to refer to content of the MIME type text/html or even more broadly as a generic term for HTML whether in its XML-descended form (such as XHTML 1.0 and later) or its form descended directly from SGML
Hyper Text Markup Language Hypertext Markup Language (HTML), the languages of the World Wide Web (WWW), allows users to produces Web pages that include text, graphics and pointer to other Web pages (Hyperlinks).

HTML is not a programming language but it is an application of ISO Standard 8879, SGML (Standard Generalized Markup Language), but specialized to hypertext and adapted to the Web. The idea behind Hypertext is that instead of reading text in rigid linear structure, we can easily jump from one point to another point. We can navigate through the information based on our interest and preference. A markup language is simply a series of elements, each delimited with special characters that define how text or other items enclosed within the elements should be displayed. Hyperlinks are underlined or emphasized works that load to other documents or some portions of the same document. HTML can be used to display any type of document on the host computer, which can be geographically at a different location. It is a versatile language and can be used on any platform or desktop.

HTML provides tags (special codes) to make the document look attractive. HTML tags are not case-sensitive. Using graphics, fonts, different sizes, color, etc., can enhance the presentation of the document. Anything that is not a tag is part of the document itself.

Basic HTML Tags:

<! -- -->	specifies comments
<A>…….</A>	Creates hypertext links
<B>…….</B>	Formats text as bold
<BIG>…….</BIG>	Formats text in large font.
<BODY>…</BODY>	Contains all tags and text in the HTML document
<CENTER>...</CENTER>	Creates text
<DD>…</DD>	Definition of a term
<DL>...</DL>	Creates definition list
<FONT>…</FONT>	Formats text with a particular font
<FORM>...</FORM>	Encloses a fill-out form
<FRAME>...</FRAME>	Defines a particular frame in a set of frames
<H#>…</H#>	Creates headings of different levels(1 – 6)
<HEAD>...</HEAD>	Contains tags that specify information about a document
<HR>...</HR>	Creates a horizontal rule
<HTML>…</HTML>	Contains all other HTML tags
<META>...</META>	Provides meta-information about a document
<SCRIPT>…</SCRIPT>	Contains client-side or server-side script
<TABLE>…</TABLE>	Creates a table
<TD>…</TD>	Indicates table data in a table
<TR>…</TR>	Designates a table row
<TH>…</TH>	Creates a heading in a table

Attributes

The attributes of an element are name-value pairs, separated by "=", and written within the start label of an element, after the element's name. The value should be enclosed in single or double quotes, although values consisting of certain characters can be left unquoted in HTML (but not XHTML).Leaving attribute values unquoted is considered unsafe.

Most elements take any of several common attributes: id, class, style and title. Most also take language-related attributes: lang and dir. The id attribute provides a document-wide unique identifier for an element. This can be used by stylesheets to provide presentational properties, by browsers to focus attention on the specific element or by scripts to alter the contents or presentation of an element. The class attribute provides a way of classifying similar elements for presentation purposes. For example, an HTML document (or a set of documents) may use the designation class="notation" to indicate that all elements with this class value are all subordinate to the main text of the document (or documents). Such notation classes of elements might be gathered together and presented as footnotes on a page, rather than appearing in the place where they appear in the source HTML.

An author may use the style non-attributal codes presentational properties to a particular element. It is considered better practice to use an element's son- id page and select the element with a stylesheet, though sometimes this can be too cumbersome for a simple ad hoc application of styled properties. The title is used to attach subtextual explanation to an element. In most browsers this title attribute is displayed as what is often referred to as a tooltip. The generic inline span element can be used to demonstrate these various non-attributes.

The preceding displays as HTML (pointing the cursor at the abbreviation should display the title text in most browsers).

Advantages

- A HTML document is small and hence easy to send over the net. It is small because it does not include formatted information.
- HTML is platform independent.
- HTML tags are not case-sensitive.

3.3.1 Introduction to JAVA

JavaScript is a script-based programming language that was developed by Netscape Communication Corporation. JavaScript was originally called Live Script and renamed as JavaScript to indicate its relationship with Java. JavaScript supports the development of both client and server components of Web-based applications. On the client side, it can be used to write programs that are executed by a Web browser within the context of a Web page. On the server side, it can be used to write Web server programs that can process information submitted by a Web browser and then update the browser's display accordingly

Even though JavaScript supports both client and server Web programming, we prefer JavaScript at Client side programming since most of the browsers supports it. JavaScript is almost as easy to learn as HTML, and JavaScript statements can be included in HTML documents by enclosing the statements between a pair of scripting tags

```
<SCRIPTS>.. </SCRIPT>.
<SCRIPT LANGUAGE = "JavaScript">
JavaScript statements
</SCRIPT>
```

Here are a few things we can do with JavaScript:

- Validate the contents of a form and make calculations.
- Add scrolling or changing messages to the Browser's status line.
- Animate images or rotate images that change when we move the mouse over them.
- Detect the browser in use and display different content for different browsers.
- Detect installed plug-ins and notify the user if a plug-in is required.

We can do much more with JavaScript, including creating entire application. JavaScript and Java are entirely different languages. A few of the most glaring differences are:

- Java applets are generally displayed in a box within the web document; JavaScript can affect any part of the Web document itself.
- While JavaScript is best suited to simple applications and adding interactive features to Web pages; Java can be used for incredibly complex applications.

There are many other differences but the important thing to remember is that JavaScript and Java are separate languages. They are both useful for different things; in fact they can be used together to combine their advantages.

Advantages

- JavaScript can be used for Sever-side and Client-side scripting.
- It is more flexible than VBScript.
- JavaScript is the default scripting languages at Client-side since all the browsers supports it.

Java Technology

Initially the language was called as "oak" but it was renamed as "Java" in 1995. The primary motivation of this language was the need for a platform-independent (i.e., architecture neutral) language that could be used to create software to be embedded in various consumer electronic devices.

- Java is a programmer's language.
- Java is cohesive and consistent.
- Except for those constraints imposed by the Internet environment, Java gives the programmer, full control.
- Finally, Java is to Internet programming where C was to system programming.

Importance of Java to the Internet

Java has had a profound effect on the Internet. This is because; Java expands the Universe of objects that can move about freely in Cyberspace. In a network, two categories of objects are transmitted between the Server and the Personal computer. They are: Passive information and Dynamic active programs. The Dynamic, Self-executing programs cause serious problems in the areas of Security and probability. But, Java addresses those concerns and by doing so, has opened the door to an exciting new form of program called the Applet.

Java can be used to create two types of programs

Applications and Applets: An application is a program that runs on our Computer under the operating system of that computer. It is more or less like one creating using C or C++. Java's ability to create Applets makes it important. An Applet is an application designed to be transmitted over the Internet and executed by a Java – compatible web browser. An applet is actually a tiny Java program, dynamically downloaded across the network, just like an image. But the difference is, it is an intelligent program, not just a media file. It can react to the user input and dynamically change.

Features of Java Security

Every time you that you download a "normal" program, you are risking a viral infection. Prior to Java, most users did not download executable programs frequently, and those who did scan them for viruses prior to execution. Most users still worried about the possibility of infecting their systems with a virus. In addition, another type of malicious program exists that must be guarded against. This type of program can gather private information, such as credit card numbers, bank account balances, and passwords. Java answers both these concerns by providing a "firewall" between a network application and your computer. When you use a Java-compatible Web browser, you can safely download Java applets without fear of virus infection or malicious intent.

Portability

For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed .As you will see, the same mechanism that helps ensure security also helps create portability. Indeed, Java's solution to these two problems is both elegant and efficient.

The Byte code

The key that allows the Java to solve the security and portability problems is that the output of Java compiler is Byte code. Byte code is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreter for byte code. Translating a Java program into byte code helps makes it much easier to run a program in a wide variety of environments. The reason is, once the run-time package exists for a given system, any Java program can run on it.

Although Java was designed for interpretation, there is technically nothing about Java that prevents on-the-fly compilation of byte code into native code. Sun has just completed its Just In Time (JIT) compiler for byte code. When the JIT compiler is a part of JVM, it compiles byte code into executable code in real time, on a piece-by-piece, demand basis. It is not possible to compile an entire Java program into executable code all at once, because Java performs various run-time checks that can be done only at run time. The JIT compiles code, as it is needed, during execution.

Java Virtual Machine (JVM)

Beyond the language, there is the Java virtual machine. The Java virtual machine is an important element of the Java technology. The virtual machine can be embedded within a web browser or an operating system. Once a piece of Java code is loaded onto a machine, it is verified. As part of the loading process, a class loader is invoked and does byte code verification makes sure that the code that's has been generated by the compiler will not corrupt the machine that it's loaded on. Byte code verification takes place at the

end of the compilation process to make sure that is all accurate and correct. So byte code verification is integral to the compiling and executing of Java code.

Overall Description

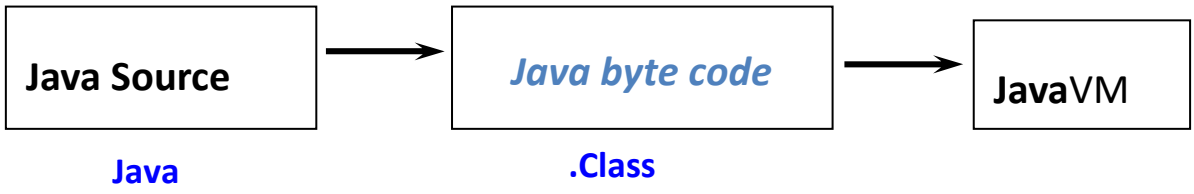


Figure: The development process of JAVA Program

Java programming uses to produce byte codes and executes them. The first box indicates that the Java source code is located in a. Java file that is processed with a Java compiler called javac. The Java compiler produces a file called a. class file, which contains the byte code. The .Class file is then loaded across the network or loaded locally on your machine into the execution environment is the Java virtual machine, which interprets and executes the byte code.

Java Database Connectivity

What Is JDBC?

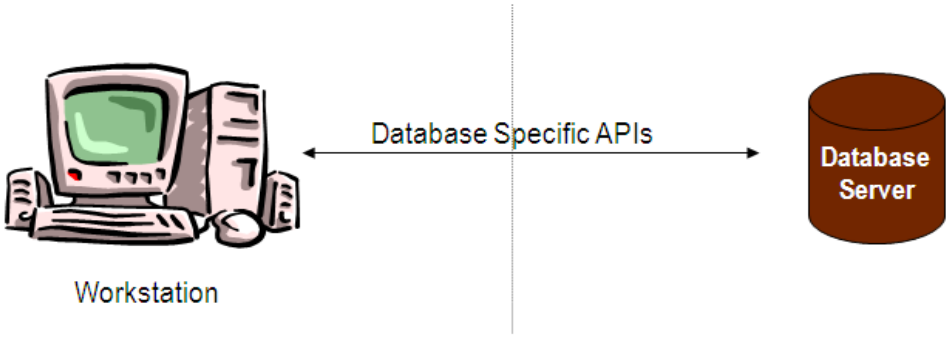
JDBC is a Java API for executing SQL statements. (As a point of interest, JDBC is a trademarked name and is not an acronym; nevertheless, JDBC is often thought of as standing for Java Database Connectivity. It consists of a set of classes and interfaces written in the Java programming language. JDBC provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.

Using JDBC, it is easy to send SQL statements to virtually any relational database. One can write a single program using the JDBC API, and the program will be able to send SQL statements to the appropriate database. The combinations of Java and JDBC lets a programmer write it once and run it anywhere.

What Does JDBC Do?

Simply put, JDBC makes it possible to do three things:

- Establish a connection with a database
- Send SQL statements
- Process the results.



- Presentation, business and data model processing logic into client application
- Server is typically a database server
- Client sends SQL statements, retrieves raw data

Figure: JDBC Connection

3.3.2 Introduction to MySQL

Structured Query Language (SQL) is the language used to manipulate relational databases. SQL is tied very closely with the relational model. In the relational model, data is stored in structures called relations or tables. SQL statements are issued for the purpose of:

Data definition: Defining tables and structures in the database (DDL used to create, alter and drop schema objects such as tables and indexes).

Data manipulation: Used to manipulate the data within those schema objects (DML Inserting, Updating, Deleting the data, and Querying the Database).

A schema is a collection of database objects that can include: tables, views, indexes and sequences

4. SYSTEM DESIGN

4.1 Data Flow Diagram

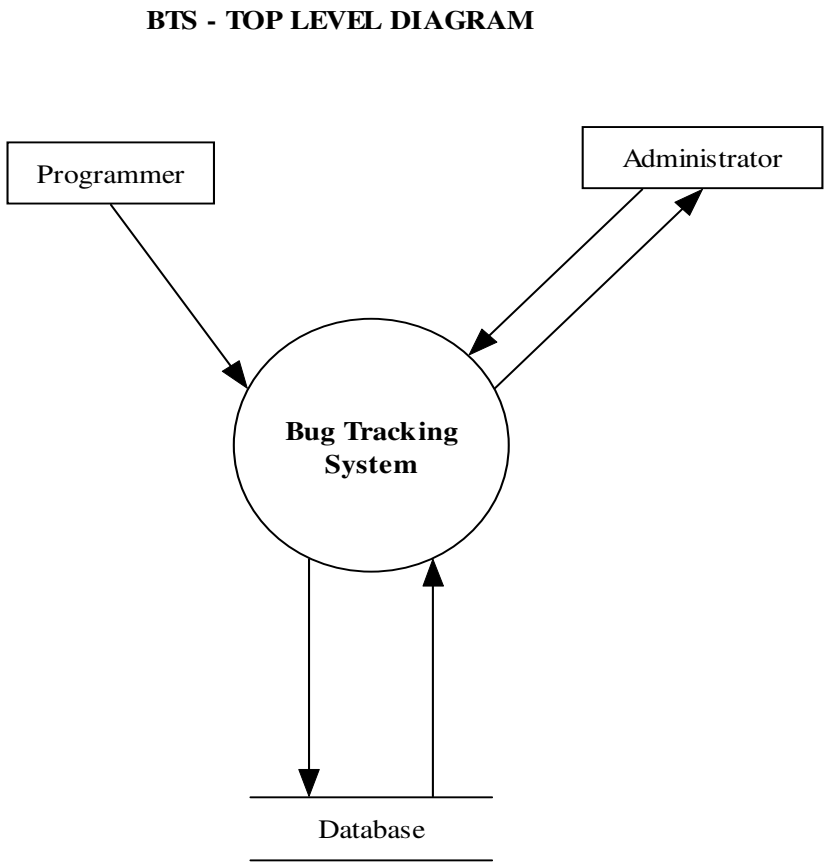


Figure: BTS Top Level Diagram

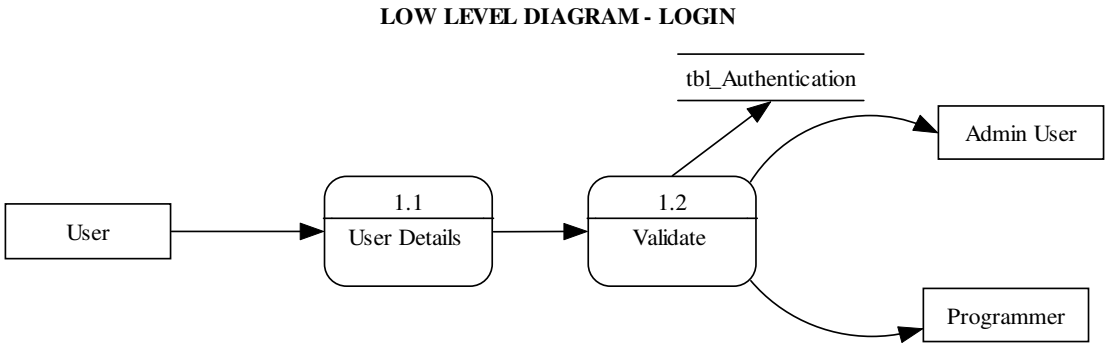
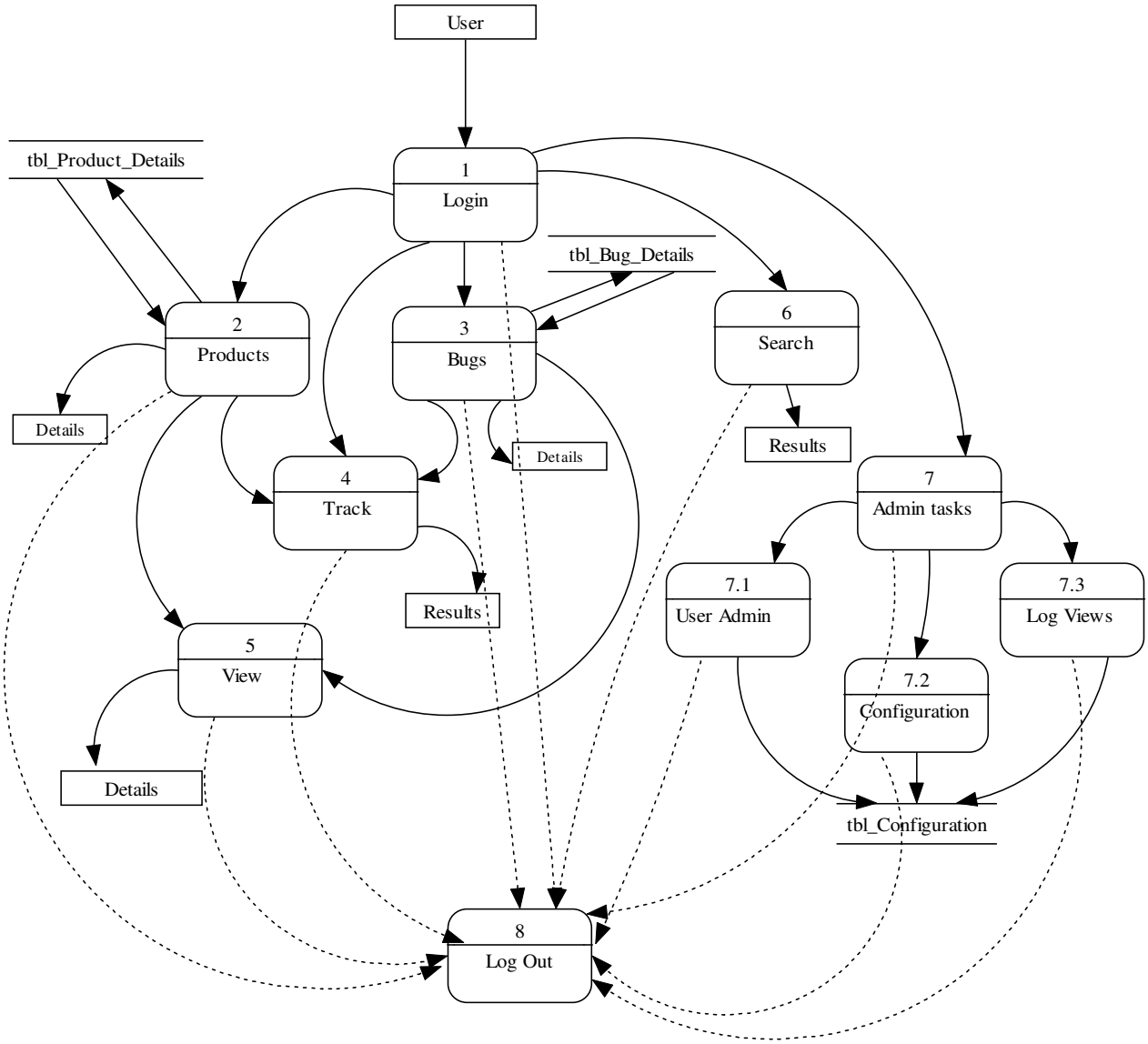
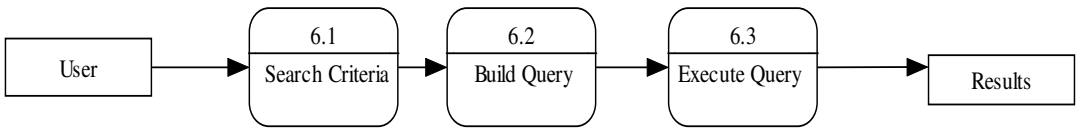


Figure: BTS Low Level Diagram

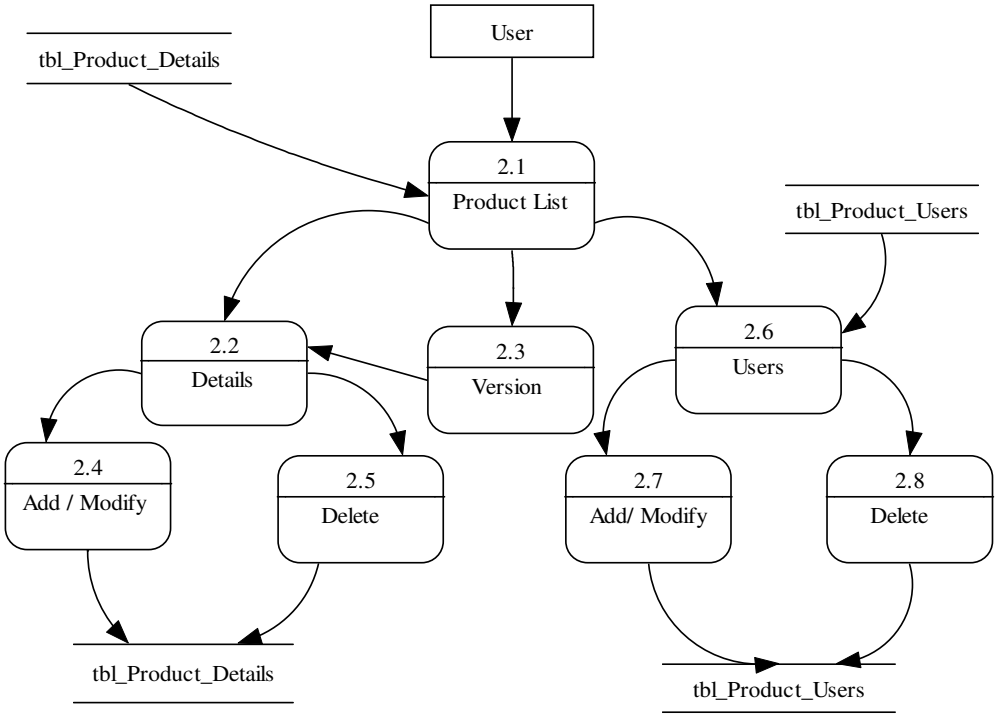
BTS - TOP LEVEL DIAGRAM



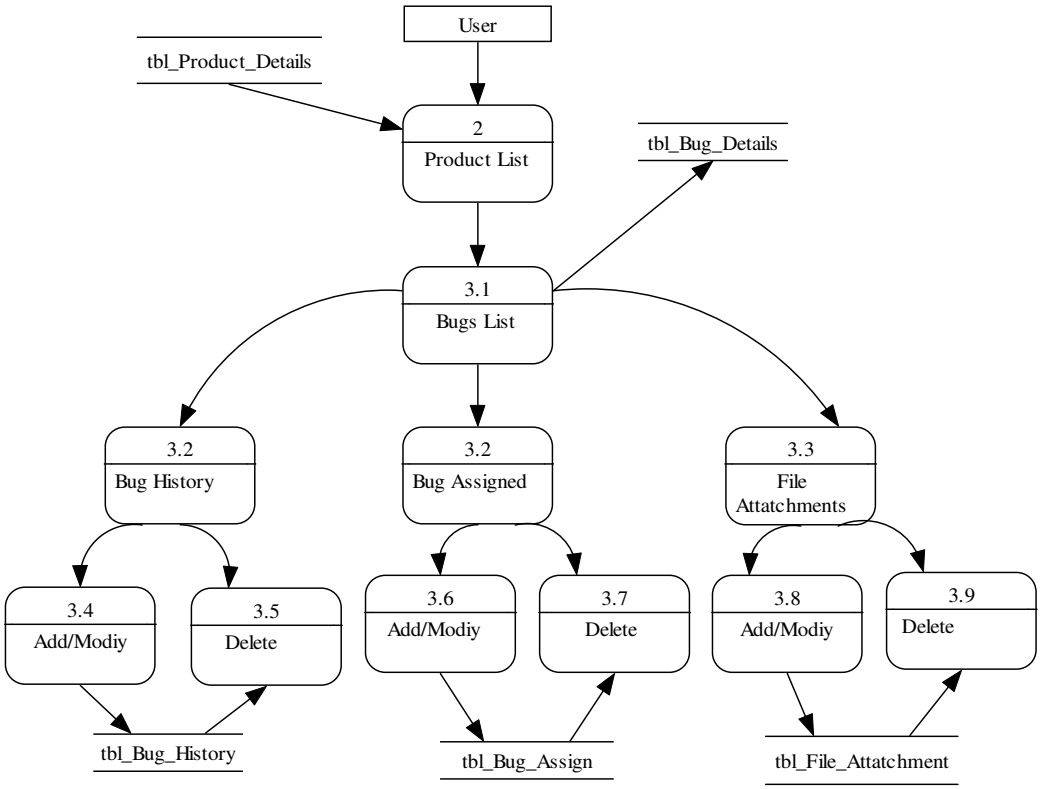
LOW LEVEL DIAGRAM - SEARCH



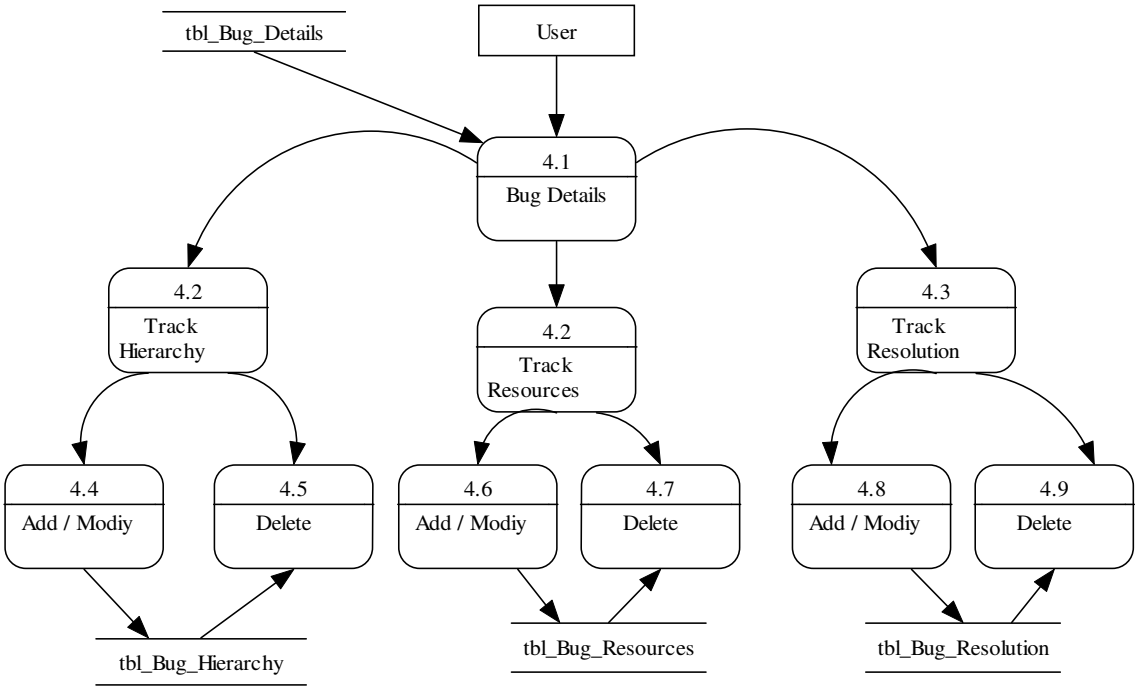
LOW LEVEL DIAGRAM - PRODUCTS



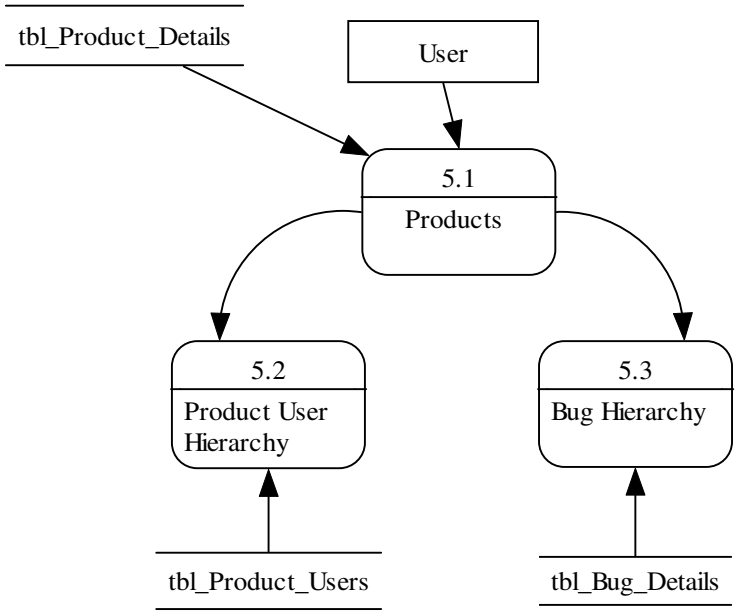
LOW LEVEL DIAGRAM - BUGS



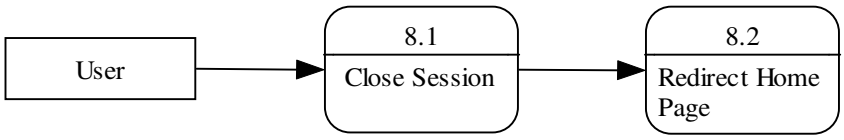
LOW LEVEL DIAGRAM - TRACKING



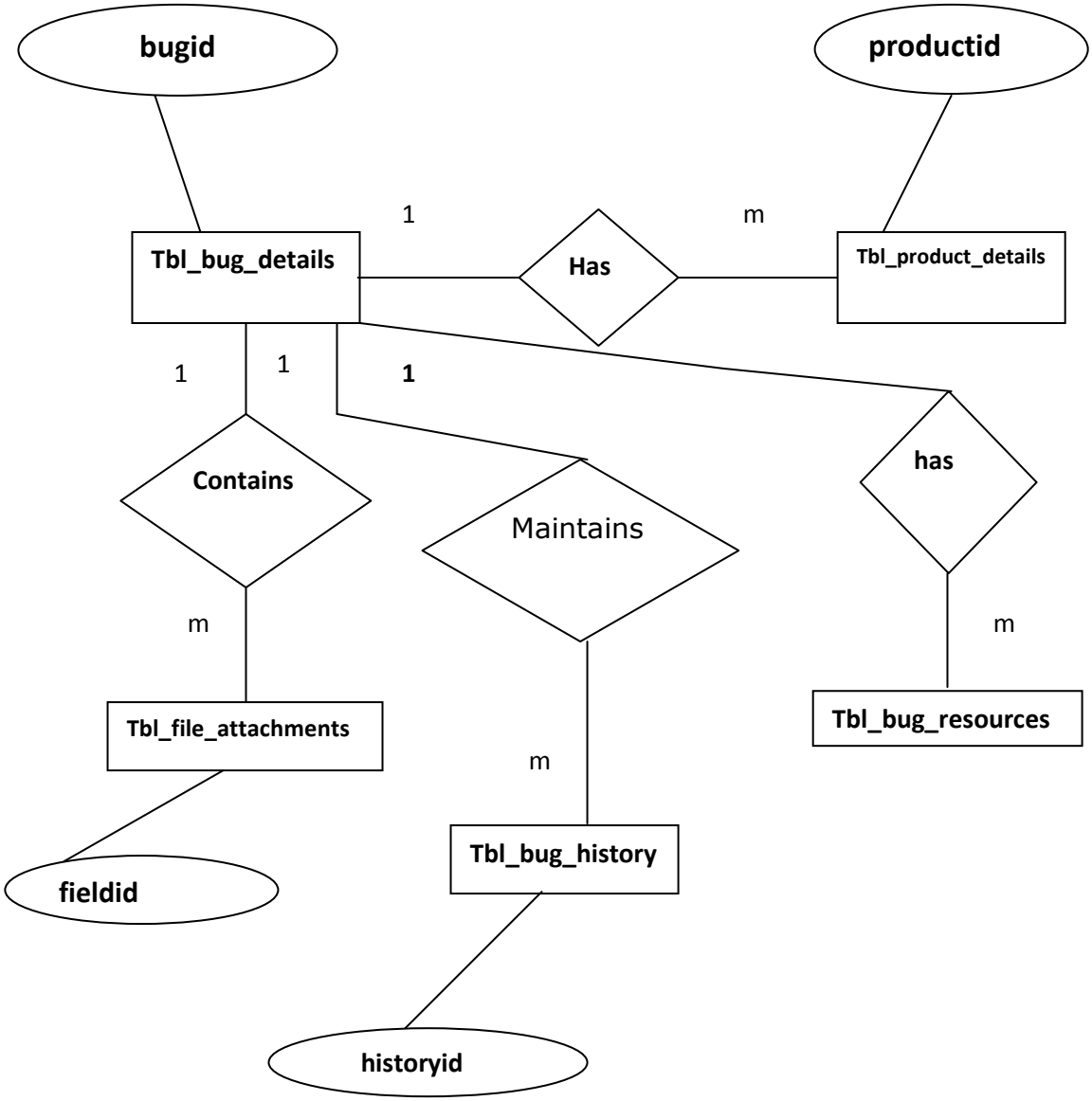
LOW LEVEL DIAGRAM - VIEW



LOW LEVEL DIAGRAM - LOGOUT



ER Diagram



5. IMPLEMENTATION

5.1 Sample Code

Database.java

```
package database;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Database {

   public static final String DRIVER = "com.mysql.jdbc.Driver";
   public static final String URL = "jdbc:mysql://localhost:3306/Bugtrackingsystem";
   public static final String USERNAME = "root";
   public static final String PASSWORD = "root";
   private static Connection connection;

   public Database() {
      super();
   }

   public static Connection getConnection() {
      try {
         if (connection == null || connection.isClosed()) {
            try {
               Class.forName(DRIVER);
               connection = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
            } catch (SQLException ex) {
               Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, ex);
            } catch (ClassNotFoundException ex) {
               Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, ex);
            }
         }
         return connection;
      } catch (SQLException ex) {
         Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, ex);
      }
      return null;
   }

   public static ResultSet getTableData(String tableName) {
```

```
        ResultSet resultSet = null;
        String query;
        connection = getConnection();
        if (connection != null) {
            try {
                query = "select * from " + tableName;
                PreparedStatement preparedStatement = (PreparedStatement)
connection.prepareStatement(query);
                resultSet = preparedStatement.executeQuery();
            } catch (SQLException ex) {
                Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        return resultSet;
    }

    public static ResultSet executeQuery(String sql) throws SQLException {
        ResultSet resultSet = null;
        Connection connection = Database.getConnection();
        PreparedStatement prepareStatement = connection.prepareStatement(sql);
        resultSet = prepareStatement.executeQuery();
        return resultSet;
    }

    public static void freeDatabase(ResultSet resultSet) throws SQLException {
        if (resultSet != null && resultSet.isClosed()) {
            resultSet.getStatement().close();
            resultSet.close();
        }
    }
}
```

LoginForm.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
        <script language="javascript">
            function validate(form){
                var userName=form.elements['userNameTextField'].value;
                if(userName.length==0){
                    alert('User Name cannot be Empty');
                    return ;
                }
```

```
            var passwordName=form.elements['passwordTextField'].value;
            if(passwordName.length==0){
               alert('Password cannot be Empty');
               return ;
            }
            form.submit();
         }
      </script>
      <link href="../default.css" rel="stylesheet" type="text/css" media="screen" />
      <style type="text/css">
      </style>
   </head>
   <body bgcolor="#a9b3bb">
      <form action="LoginAction.jsp">
         <table>
            <tr>
               <td></td>
            </tr>
            <tr>
               <td></td>
            </tr>
         </table>
         <table style="border:solid #eec352" width="40%" align=center cellpadding="0"
cellspacing="0">

            <tr style="font-family:calibri;background:#eec352">
               <td style="padding:10px 10px 10px 10px;border-bottom:1px solid #eec352"
colspan="2">Login</td>
            </tr>
            <tr style="font-family:calibri">
               <td style="padding:10px 10px 10px 10px">User Name : </td>
               <td><input type="text" size="40" name="userNameTextField"/></td>
            </tr>
            <tr style="font-family:calibri">
               <td style="padding:10px 10px 10px 10px;border-bottom:1px solid
#eec352">Password : </td>
               <td style="border-bottom:1px solid #eec352"><input type="password"
size="42" name="passwordTextField"/></td>
            </tr>
            <tr>
               <td style="padding:10px 10px 10px 10px;" colspan="2"
align="center"><input type="button" value="Submit" onclick="validate(this.form);">
</td>
            </tr>
         </table>
      </form>
   </body>
</html>
```

Welcome.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body bgcolor="#dfdfdf">
      <center><h4 style="font-family:calibri;">Hello My Dear Friend
<%=session.getAttribute("name")%> Welcome to Bug Tracking System.</h4></center>
      <center><h4 style="font-family:calibri;">You are logged in as
<%=session.getAttribute("role")%>. </h4></center>
   </body>
</html>
```

CreateBug.jsp

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body bgcolor="#dfdfdf">
      <form action="CreateBugHandler.jsp">
         <table align="center" width="80%">
            <tr>
               <td width="15%"  style="font-family:calibri;padding:10px 10px 10px
10px;">
                  Bud ID:
               </td>
               <td width="25%">
                  <input type="text" size="25" name="bugIdTextField"/>
               </td>
               <td width="15%"  style="font-family:calibri;padding:10px 10px 10px
10px;">
                  Bug Details :
               </td>
               <td rowspan="3" width="25%">
                  <textarea cols="40" rows="10" name="bugDetailsTextArea"></textarea>
               </td>
            </tr>
            <tr>
               <td width="15%"  style="font-family:calibri;padding:10px 10px 10px
10px;">
                  Bug Name:
```

```
                </td>
                <td width="25%">
                    <input type="text" size="25" name="bugNameTextField"/>
                </td>

            </tr>
            <tr>
                <td width="15%"  style="font-family:calibri;padding:10px 10px 10px
10px;">
                    Project ID :
                </td>
                <td width="25%">
                    <%
        ResultSet projectResultSet = Database.getTableData("projecttable");
        out.print("<select name=projectIdCombo>");
        if (projectResultSet != null) {
            try {
                while (projectResultSet.next()) {
                    out.print("<option value=" + projectResultSet.getString("projectid") + ">" +
projectResultSet.getString("projectid") + "</option>");
                }
            } catch (SQLException sqle) {
            }
            projectResultSet.close();
        }
        out.print("</select>");
                    %>
                </td>

            </tr>
            <tr>
                <td width="15%"  style="font-family:calibri;padding:10px 10px 10px
10px;">
                    Priority :
                </td>
                <td width="25%">
                    <%
        ResultSet resultSet = Database.getTableData("bugprioritytable");
        out.print("<select name=bugPriorityCombo>");
        if (resultSet != null) {
            try {
                while (resultSet.next()) {
                    out.print("<option value=" + resultSet.getString("bugpriorityid") + ">" +
resultSet.getString("bugpriority") + "</option>");
                }
            } catch (SQLException sqle) {
            }
            resultSet.close();
        }
        out.print("</select>");
```

```
                %>
              </td>
              <td width="15%"  style="font-family:calibri;padding:10px 10px 10px
10px;">
                Bug Status :
              </td>
              <td width="25%">
                <%
      ResultSet bugResultSet = Database.getTableData("bugstatustable");
      out.print("<select name=bugStatusCombo>");
      if (bugResultSet != null) {
         try {
            while (bugResultSet.next()) {
               out.print("<option value=" + bugResultSet.getString("bugstatusid") + ">" +
bugResultSet.getString("bugstatus") + "</option>");
            }
         } catch (SQLException sqle) {
         }
         bugResultSet.close();
      }
      out.print("</select>");
                %>
              </td>
            </tr>
            <tr>
              <td align="center" colspan="4">
                <input type="submit" value="Submit"/>
              </td>
            </tr>
         </table>
      </form>
   </body>
</html>
```

ViewBug.jsp

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body bgcolor="#dfdfdf">
      <%

      ArrayList<BugDTO> bugList = null;
      ResultSet resultSet = Database.getTableData("bugtable");
      if (resultSet != null) {
         bugList = new ArrayList<BugDTO>();
         try {
```

```
            while (resultSet.next()) {
               String bugId = resultSet.getString("bugid");
               String bugName = resultSet.getString("bugname");
               String projectId = resultSet.getString("projectid");
               bugList.add(new BugDTO(bugId, bugName, projectId, null, null, null));
            }
         } catch (SQLException sqle) {
         }
      }
      %>
      <table style="border:2px solid #363636;" width="80%" align=center
cellpadding="0" cellspacing="0">
         <tr style="background-color:#eec352;">
            <td width="25" align="center" style="font-family:calibri;padding:10px 10px
10px 10px;border-bottom:1px solid #363636">
               Bug ID
            </td>
            <td align="center" style="font-family:calibri;padding:10px 10px 10px
10px;border-bottom:1px solid #363636">
               Bug
            </td>
            <td align="center" style="font-family:calibri;padding:10px 10px 10px
10px;border-bottom:1px solid #363636">
               Project Name
            </td>
         </tr>
         <%
      for (BugDTO bug : bugList) {
         out.print("<tr style=font-family:calibri;border: 3px solid blue>");
         out.print("<td width=25 align=center style=\"font-family:calibri;padding:10px
10px 10px 10px;border-bottom:1px solid #363636\" width=\"40%\">");
         out.print(bug.getBugId());
         out.print("</td>");
         out.print("<td align=center style=\"font-family:calibri;padding:10px 10px 10px
10px;border-bottom:1px solid #363636\" width=\"40%\">");
         out.print("<a href=BugDetails.jsp?bugid=" + bug.getBugId() + ">");
         out.print(bug.getBugName());
         out.print("</a>");
         out.print("</td>");
         out.print("<td align=center style=\"font-family:calibri;padding:10px 10px 10px
10px;border-bottom:1px solid #363636\" width=\"40%\">");
         out.print(bug.getProjectId());
         out.print("</td>");
         out.print("</tr>");
      }
         %>
      </table>
   </body>
</html>
```

CreateProject.jsp

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>

   <body bgcolor="#dfdfdf">
      <form action="CreateProjectHandler.jsp">
         <table align="center" width="100%">
            <tr>
               <td valign="top" align="right" style="font-family:calibri;padding:10px 10px
10px 10px;">Project ID : </td>
               <td valign="top" style="padding:10px 10px 10px 10px;"><input type="text"
size="50" name="projectIdTextField"/></td>
               <td valign="top" align="right" style="font-family:calibri;padding:10px 10px
10px 10px;">Project Details : </td>
               <td rowspan="3" style="padding:10px 10px 10px 10px;"><textarea
cols="40" rows="10" name="projectDetailsTextArea"></textarea></td>
            </tr>
            <tr>
               <td align="right" style="font-family:calibri;padding:10px 10px 10px
10px;">Project Name : </td>
               <td style="padding:10px 10px 10px 10px;"><input type="text" size="50"
name="projectNameTextField"/></td>
            </tr>
            <tr>
               <td align="right" style="font-family:calibri;padding:10px 10px 10px
10px;">Assigned to : </td>
               <td style="padding:10px 10px 10px 10px;"><input type="text" size="50"
name="assignedToTextField"/></td>
            </tr>
            <tr>
               <td colspan="4" align="center"> <input type="submit"
value="Submit"/></td>
            </tr>
         </table>
      </form>
   </body>
</html>
```

5.2 Screen Shots

.

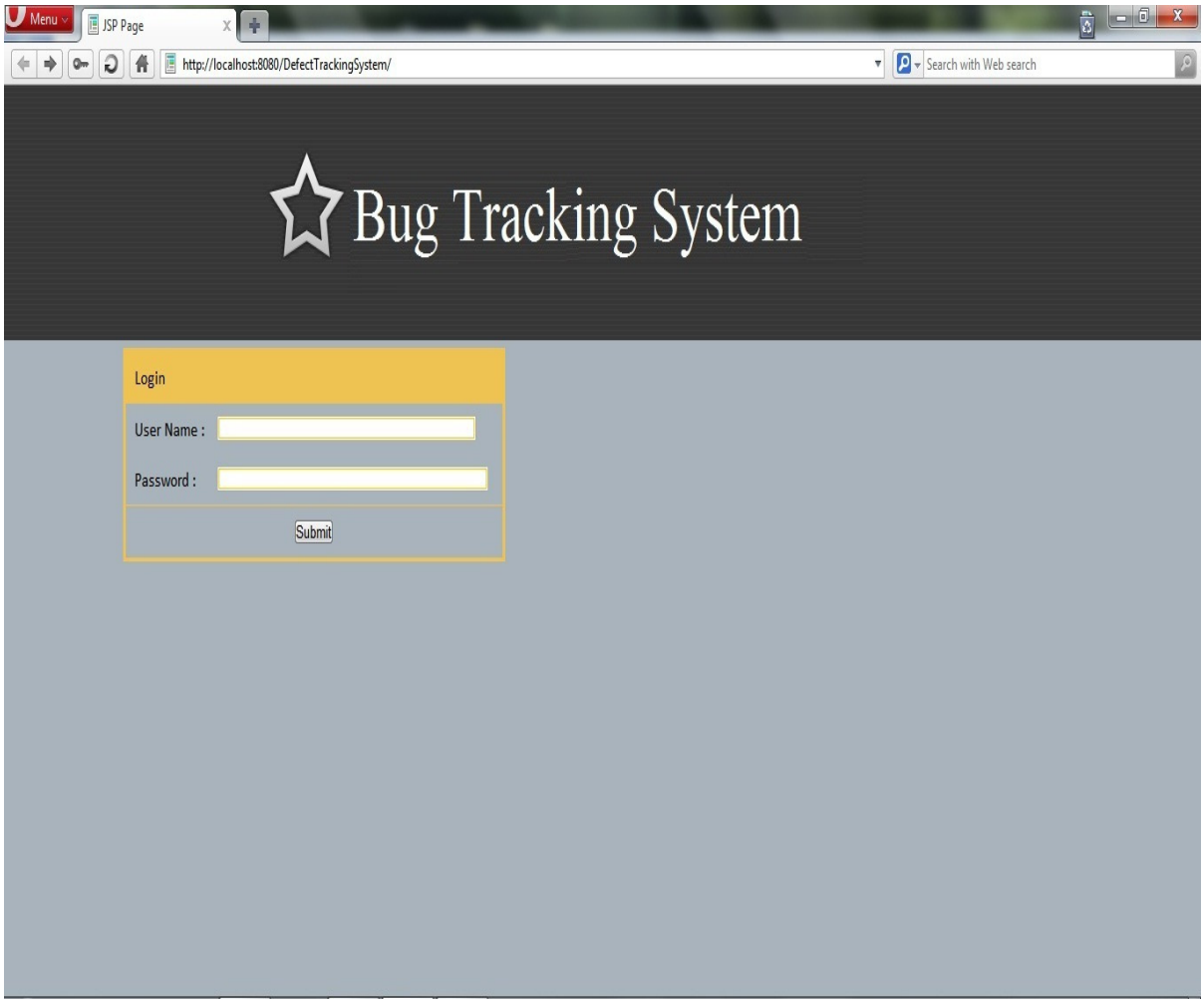


Figure: login

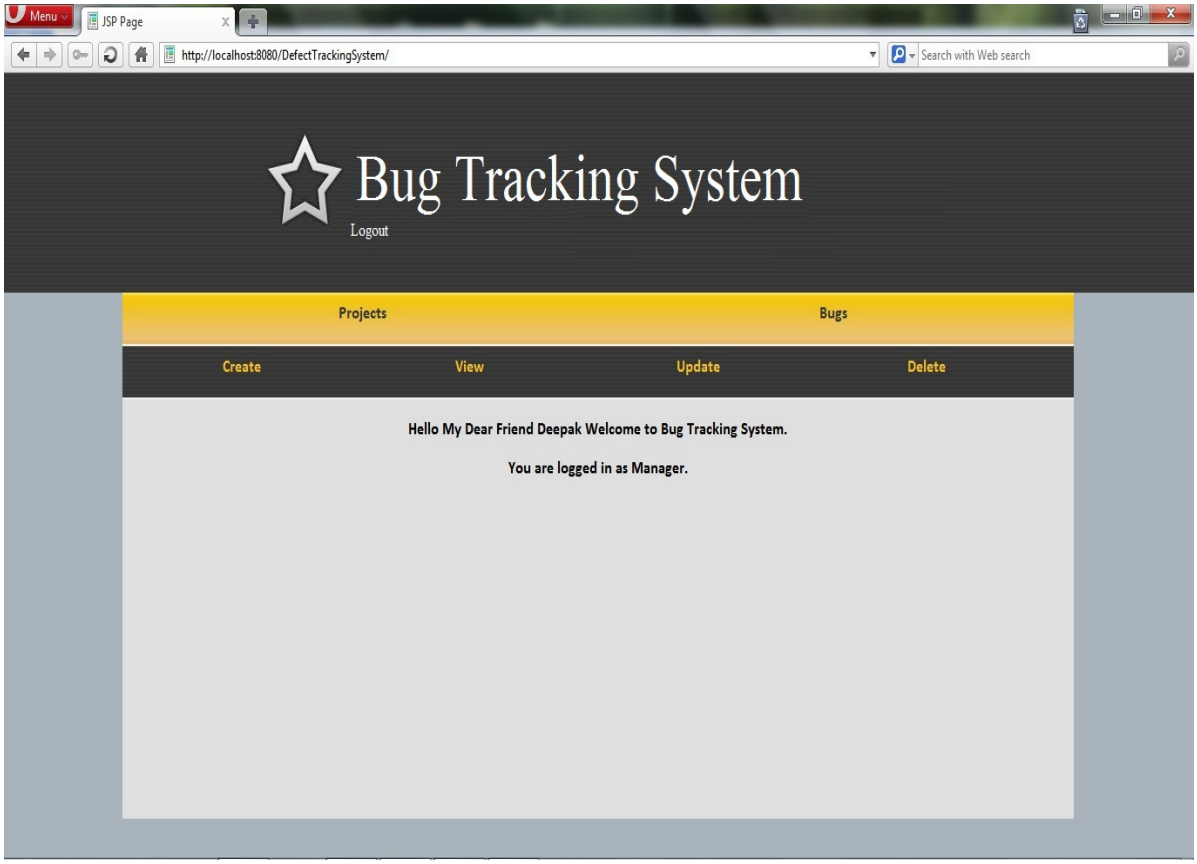


Figure: manager log in

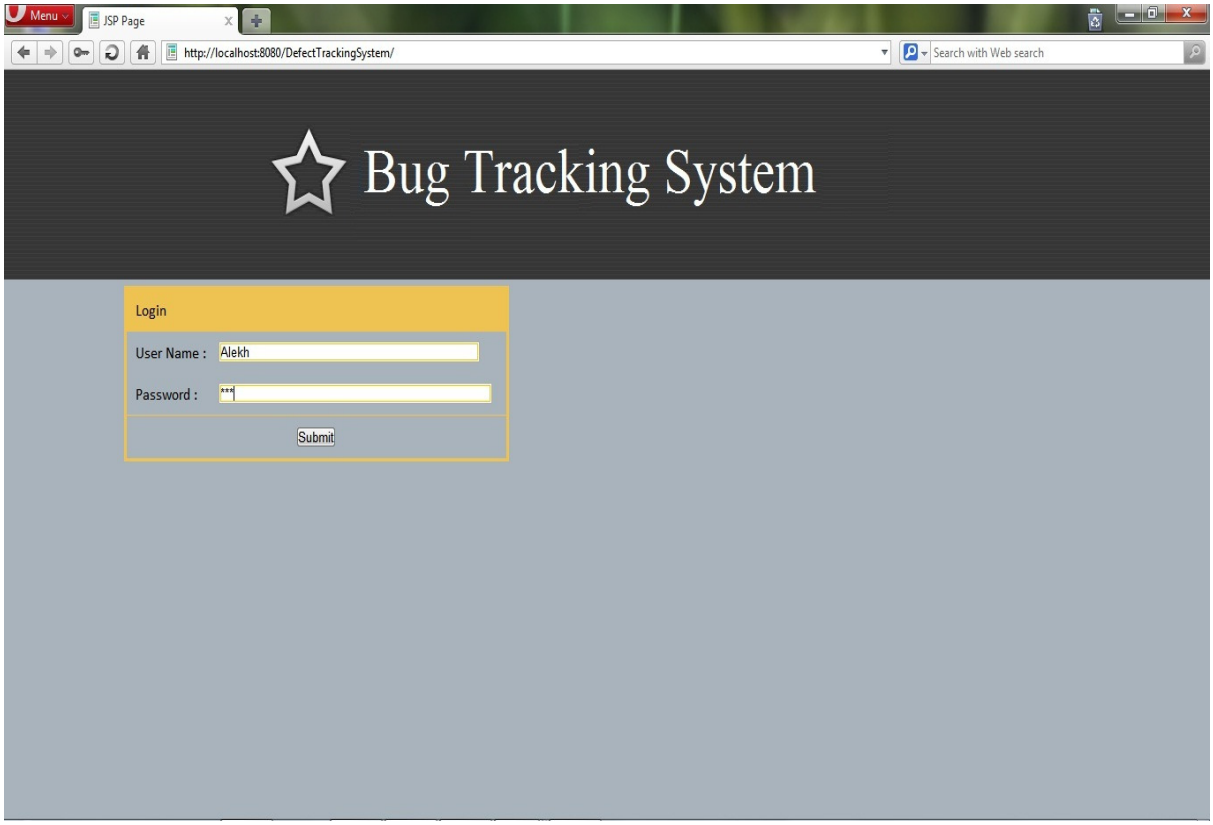


Figure: tester

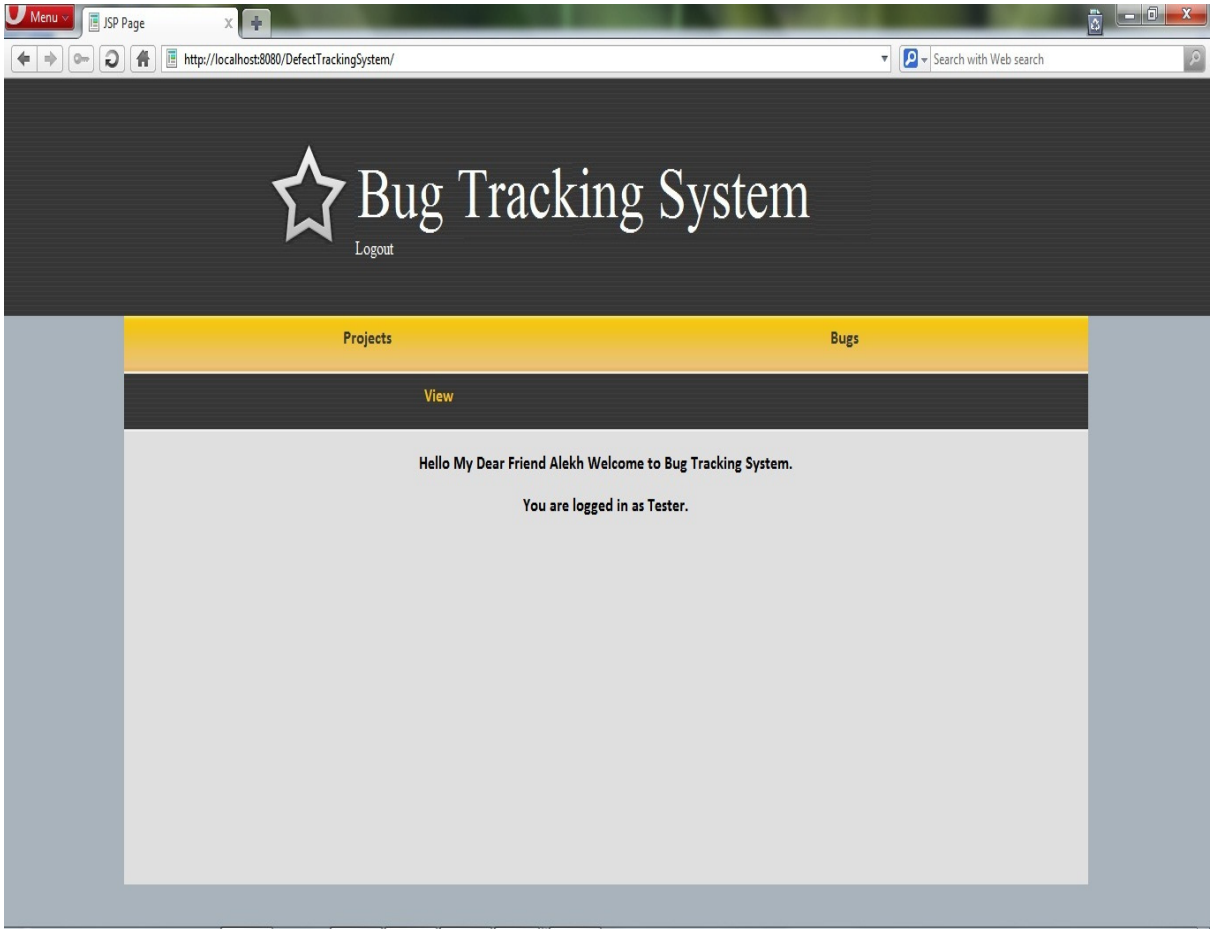


Figure: Tester login

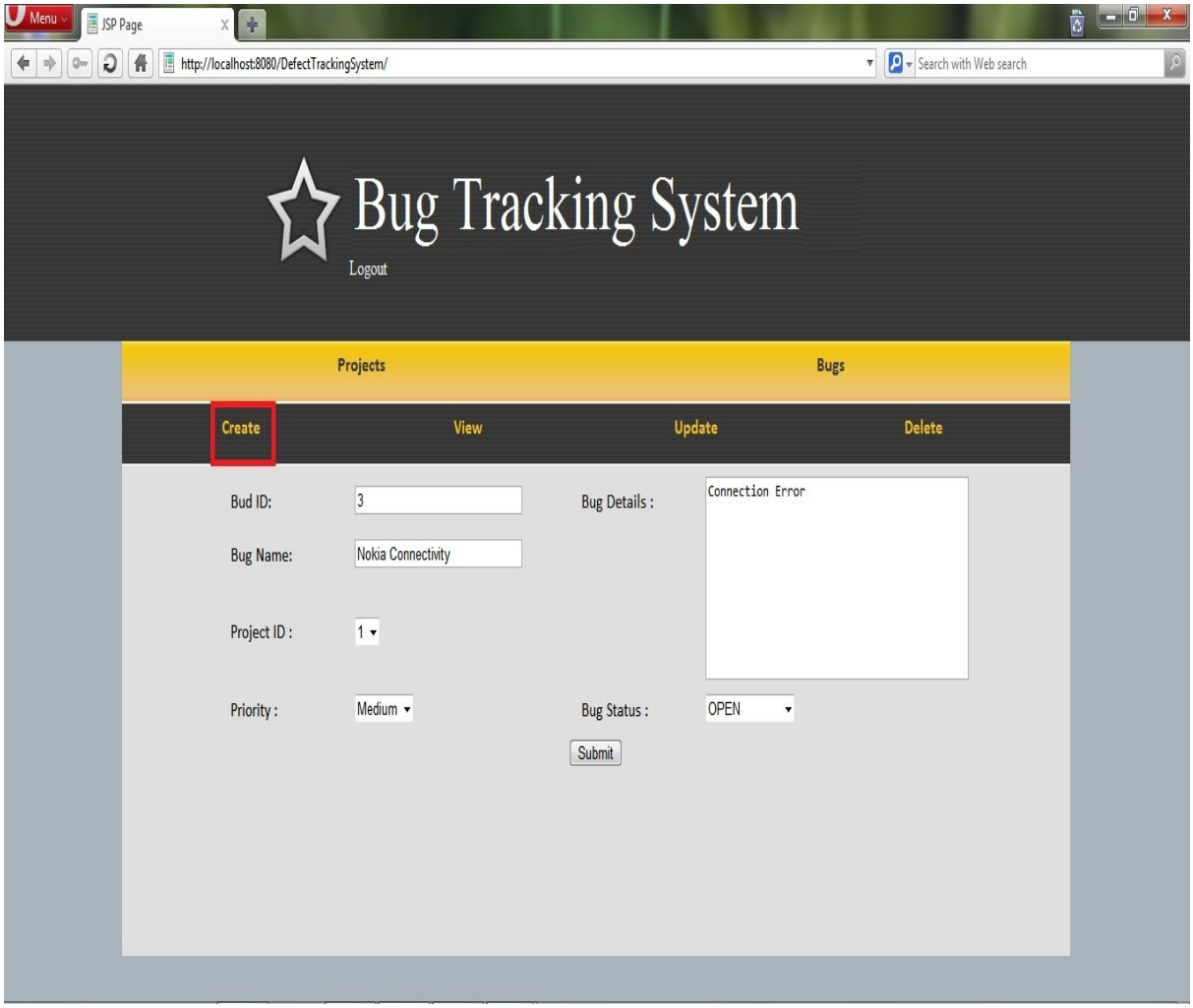


Figure: tester create

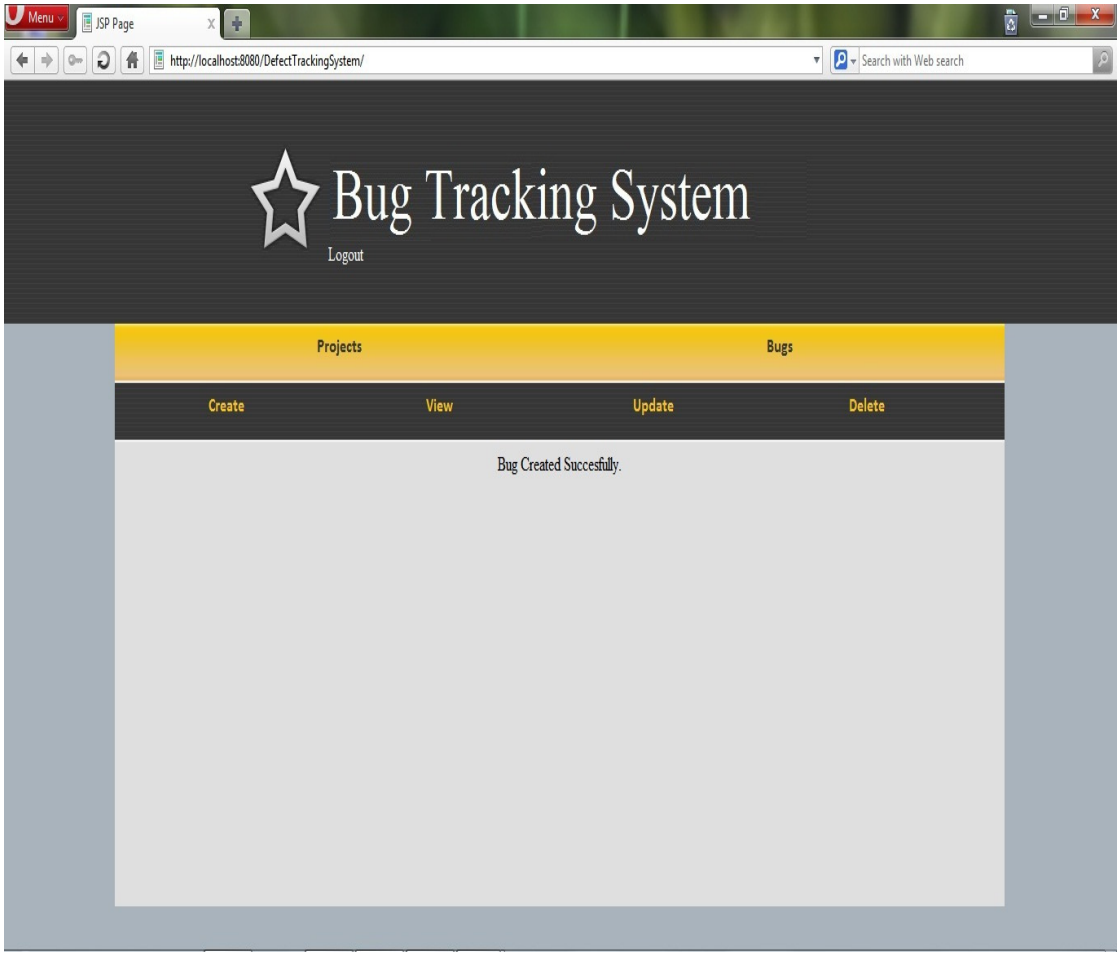


Figure:bug create succesfully

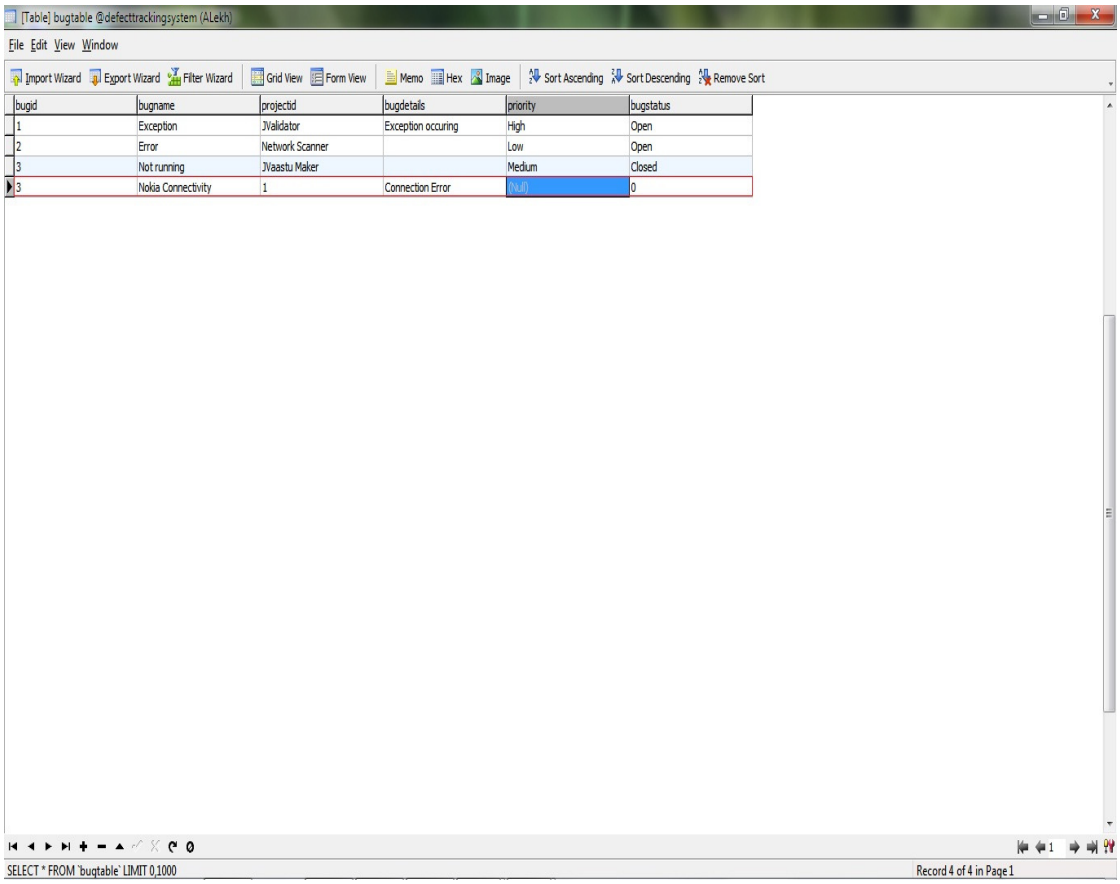


Figure:bug table

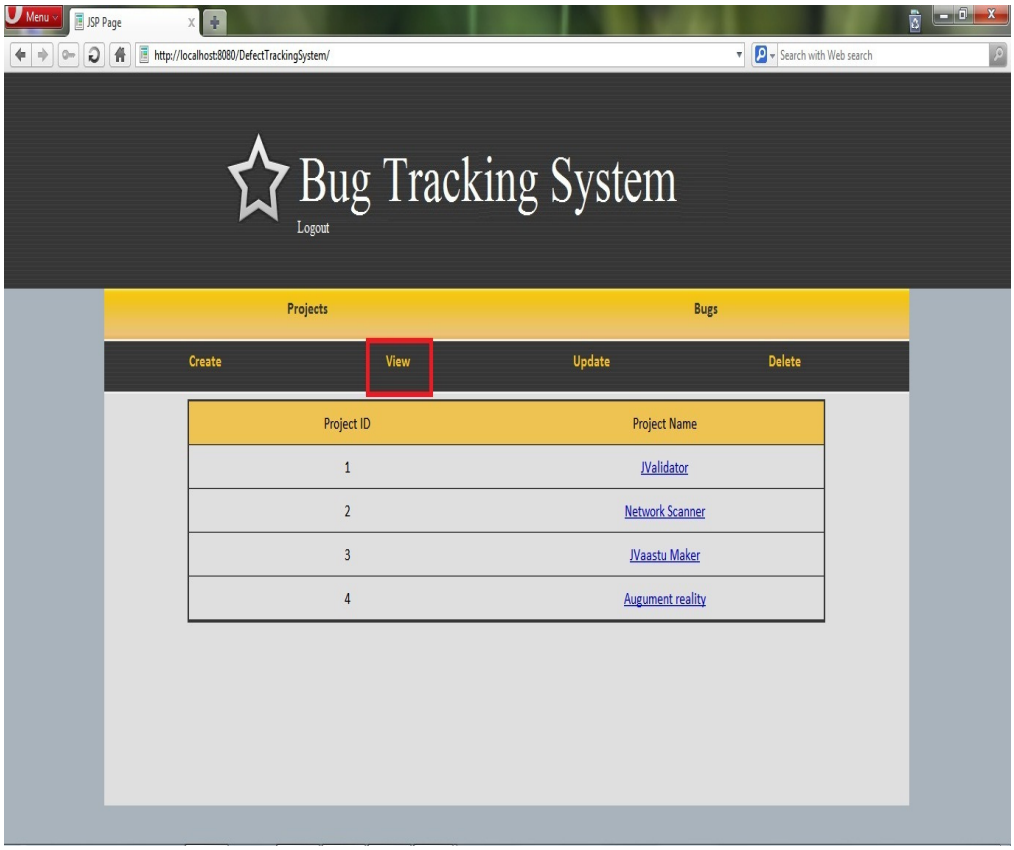


Figure:bug view

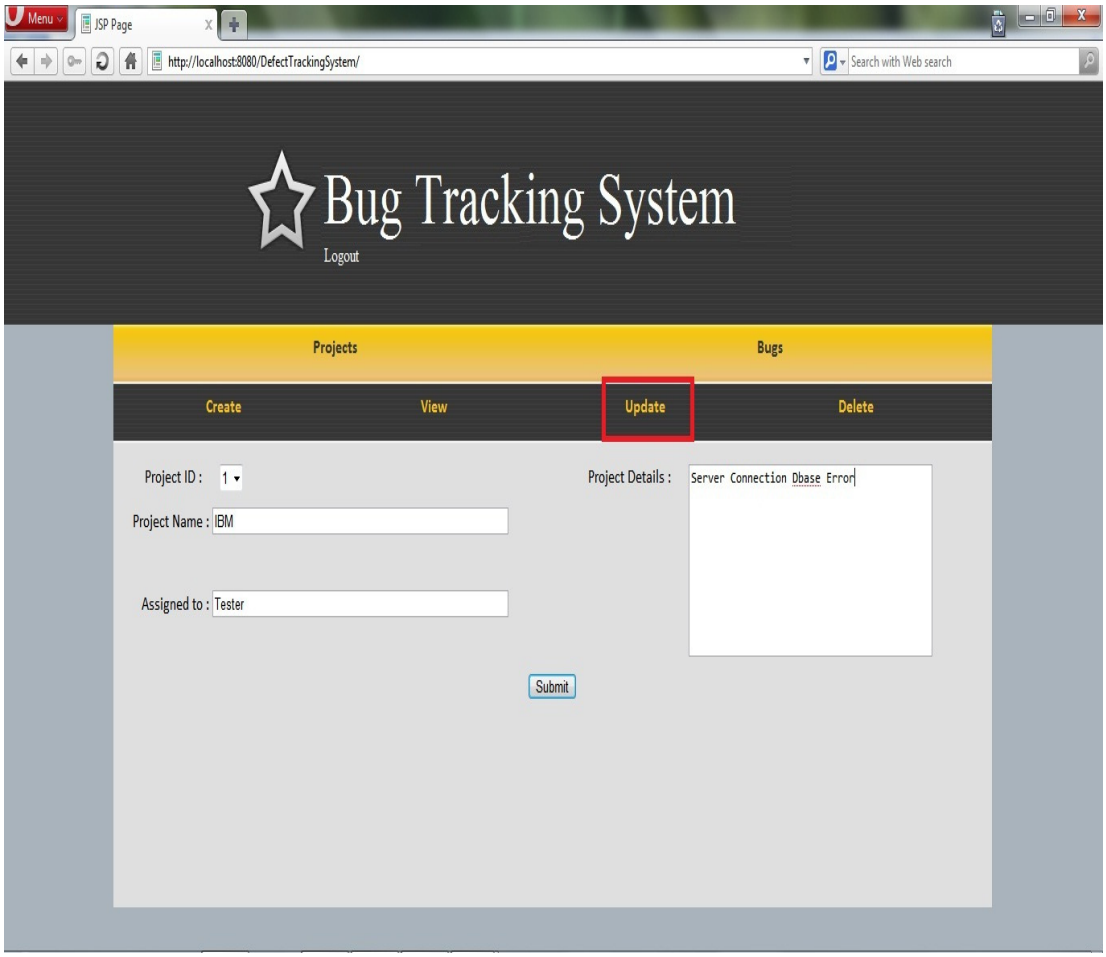


Figure:bug update

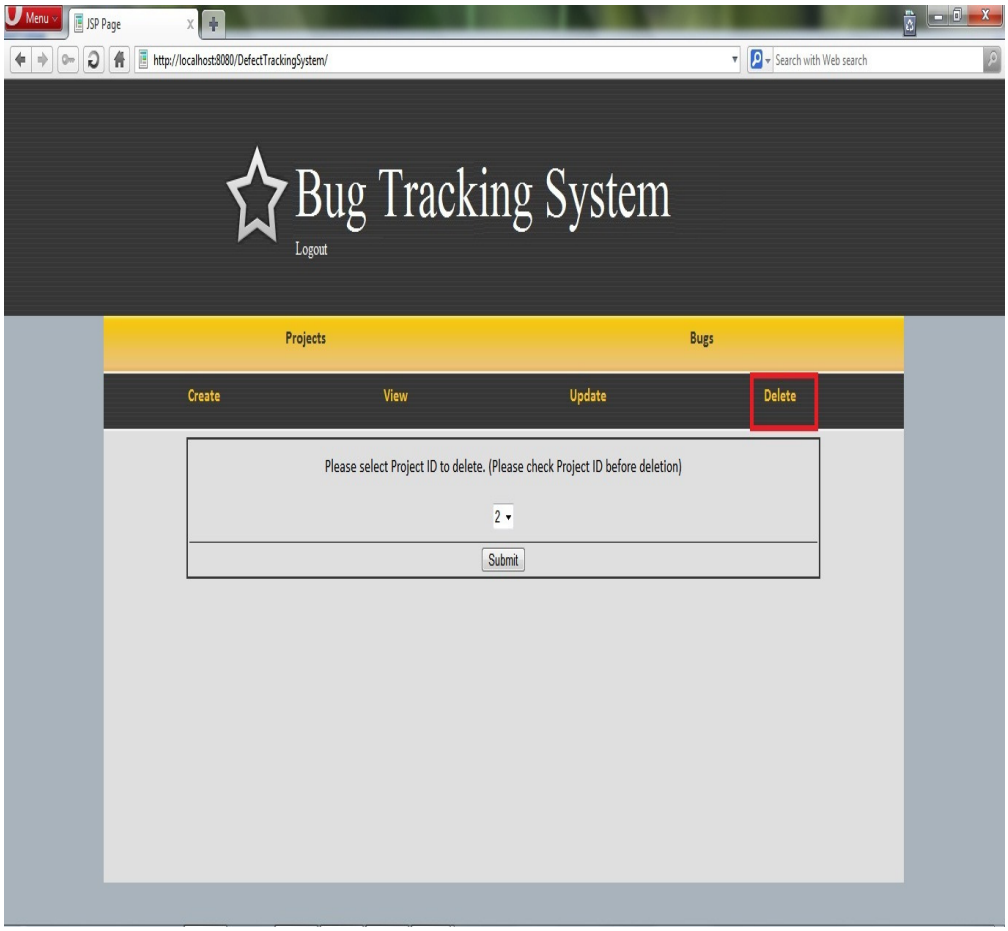


Figure:bug delete

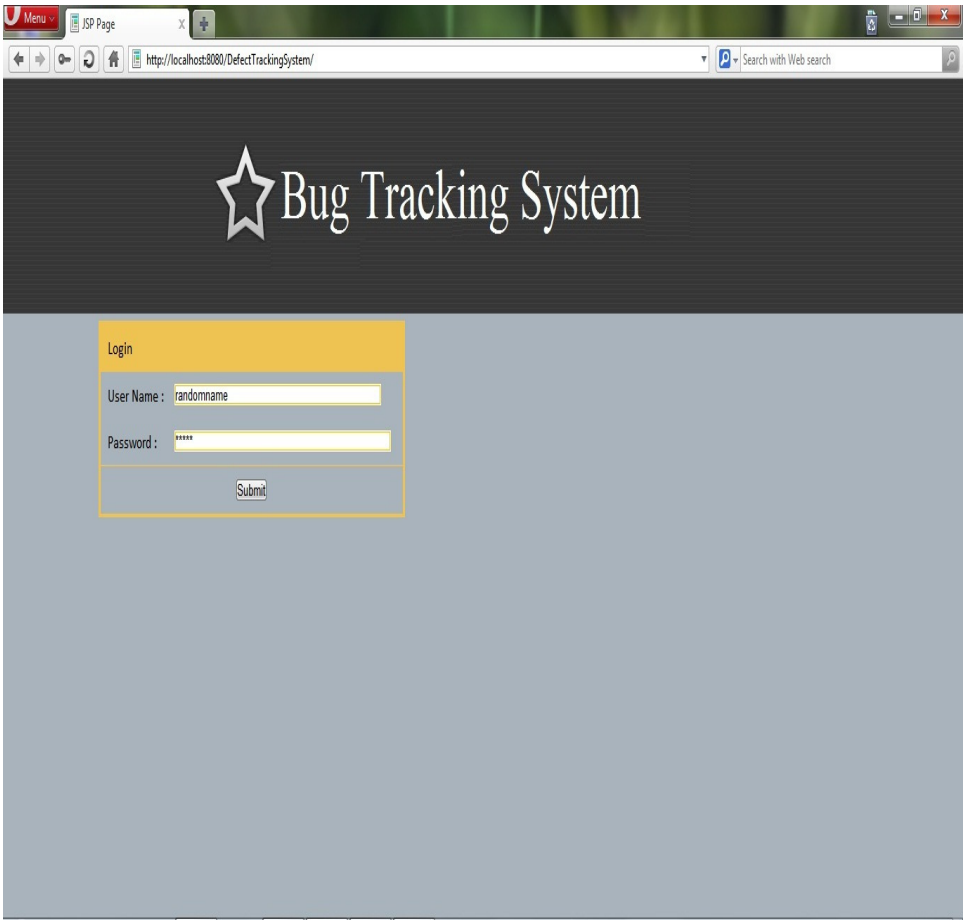


Figure: improper login

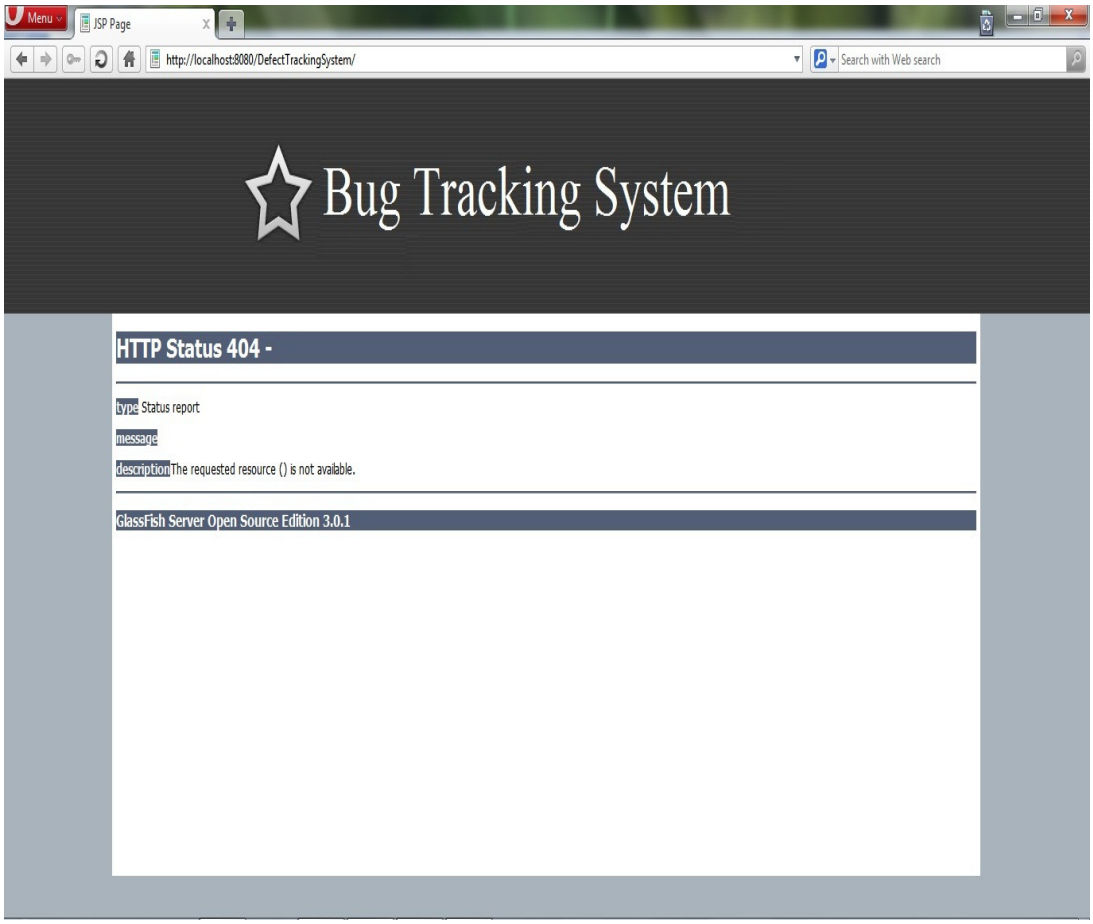


Figure: error page

6. TESTING

6.1 Introduction to Testing

During the software development process, errors are inevitably introduced and some of them are even amplified as a project progresses. To detect these errors so that they can be removed, we need to test the software.

The success of testing depends on the test cases used. Considering that most software projects are delivered in a limited time span using fixed resources, effective test cases must be used to optimize the available resources.

Software Testing is the process of executing a program or system with the intent of finding errors. In addition, it is important to plan for testing, otherwise it is likely to be skipped or performed in a haphazard fashion. Planning for software testing involves organizing testing at three levels—unit, integration, and high-order. The intent and scope of testing varies for these three levels.

Planning for software testing also involves procuring tools to automate testing and identifying the people who will perform testing. A problem with software testing is that testing all combinations of inputs and preconditions is not feasible when testing anything other than a simple product. This means that the number of Bugs in a software product can be very large and Bugs that occur infrequently are difficult to find in testing.

More significantly, par functional dimensions of quality--for example, usability, scalability, performance, compatibility, reliability--can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another. Software bugs will almost always exist in any software module with moderate size: not because programmers are careless or irresponsible, but because the complexity of software is generally intractable -- and humans have only limited ability to manage complexity. It is also true that for any complex systems, design Bugs can never be completely ruled out.

When Testing Is Carried Out?

A common practice of software testing is that it is performed by an independent group of testers after the functionality is developed but before it is shipped to the customer. This practice often results in the testing phase being used as project buffer to compensate for project delays, thereby compromising the time devoted to testing. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes.

Another common practice is for test suites to be developed during technical support escalation procedures. Such tests are then maintained in regression testing suites to ensure that future updates to the software don't repeat any of the known mistakes.

Types of software testing

ACCEPTANCE TESTING

Testing to verify a product meets customer specified requirements. A customer usually does this type of testing on a product that is developed externally.

BLACK BOX TESTING

Testing without knowledge of the internal workings of the item being tested. Tests are usually functional.

COMPATIBILITY TESTING

Testing to ensure compatibility of an application or Web site with different browsers, OSs, and hardware platforms. Compatibility testing can be performed manually or can be driven by an automated functional or regression test suite.

CONFORMANCE TESTING

Verifying implementation conformance to industry standards. Producing tests for the behavior of an implementation to be sure it provides the portability, interoperability, and/or compatibility a standard defines.

FUNCTIONAL TESTING

Validating an application or Web site conforms to its specifications and correctly performs all its required functions. This entails a series of tests which perform a feature by feature validation of behavior, using a wide range of normal and erroneous input data. This can involve testing of the product's user interface, APIs, database management, security, installation, networking; etc.

INTEGRATION TESTING

Testing in which modules are combined and tested as a group. Modules are typically code modules, individual applications, client and server applications on a network, etc. Integration Testing follows unit testing and precedes system testing.

LOAD TESTING

Load testing is a generic term covering Performance Testing and Stress Testing.

PERFORMANCE TESTING

Performance testing can be applied to understand your application or WWW site's scalability, or to benchmark the performance in an environment of third party products such as servers and middleware for potential purchase. This sort of testing is particularly useful to identify performance bottlenecks in high use applications. Performance testing generally involves an automated test suite as this allows easy simulation of a variety of normal, peak, and exceptional load conditions.

REGRESSION TESTING

Similar in scope to a functional test, a regression test allows a consistent, repeatable validation of each new release of a product or Web site. Such testing ensures reported product Bugs have been corrected for each new release and that no new quality problems were introduced in the maintenance process. Though regression testing can be performed manually an automated test suite is often used to reduce the time and resources needed to perform the required testing.

SMOKE TESTING

A quick-and-dirty test that the major functions of a piece of software work without bothering with finer details. Originated in the hardware testing practice of turning on a new piece of hardware for the first time and considering it a success if it does not catch on fire.

STRESS TESTING

Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements to determine the load under which it fails and how. A graceful degradation under load leading to non-catastrophic failure is the desired result. Often Stress Testing is performed using the same process as Performance Testing but employing a very high level of simulated load.

SYSTEM TESTING

Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of

black box testing, and as such, should require no knowledge of the inner design of the code or logic.

UNIT TESTING

Functional and reliability testing in an Engineering environment. Producing tests for the behavior of components of a product to ensure their correct behavior prior to system integration.

WHITE BOX TESTING

Testing based on an analysis of internal workings and structure of a piece of software. Includes techniques such as Branch Testing and Path Testing. Also known as Structural Testing and Glass Box Testing.

When to stop testing?

Testing is potentially endless. We cannot test till all the Bugs are unearthed and removed it is simply impossible. At some point, we have to stop testing and ship the software. The question is when.

Realistically, testing is a trade-off between budget, time and quality. It is driven by profit models. The pessimistic and unfortunately most often used approach is to stop testing whenever some or any of the allocated resources -- time, budget, or test cases -- are exhausted. The optimistic stopping rule is to stop testing when either reliability meets the requirement, or the benefit from continuing testing cannot justify the testing cost. This will usually require the use of reliability models to evaluate and predict reliability of the software under test. Each evaluation requires repeated running of the following cycle: failure data gathering -- modeling -- prediction. This method does not fit well for ultra-dependable systems, however, because the real field failure data will take too long to accumulate.

7. CONCLUSION

This project Bug Tracking System for Improving Software Quality and Reliability is to keep track of employee skills and based on the skills assigning of the task is done to an employee. Employee does Bugs capturing. It can be done on daily basis. Various Reports are generated by this System for an employee and as well as to a manager.

This project will be accessible to all developers and its facility allows developers to focus on creating the database schema and while letting the application server define table based on the fields in JSP and relationships between them.

This application software has been computed successfully and was also tested successfully by taking “test cases”. It is user friendly, and has required options, which can be utilized by the user to perform the desired operations.

7.1 Work Done

The software is developed using Java as front end and MYSQL as back end in Windows environment. The goals that are achieved by the software are:

- ✓ Instant access.
- ✓ Improved productivity.
- ✓ Optimum utilization of resources.
- ✓ Efficient management of records.
- ✓ Simplification of the operations.
- ✓ Less processing time and getting required information.
- ✓ User friendly.
- ✓ Portable and flexible for further enhancement.

7.2 Future Enhancements

It is not possible to develop a system that makes all the requirements of the user. User requirements keep changing as the system is being used. Some of the future enhancements that can be done to this system are:

- As the technology emerges, it is possible to upgrade the system and can be adaptable to desired environment.
- Because it is based on object-oriented design, any further changes can be easily adaptable.
- Based on the future security issues, security can be improved using emerging technologies.
- Attendance module can be added
- sub admin module can be added

Limitations of the system:

- Only the permanent employees can access the system.
- System works with windows'98 and its compatible environments.
- Advanced techniques are not used to check the authorization.
- Once the employee is registered to a course cannot drop, without completing.

8. BIBLOGRAPHY

Core Java™ 2 Volume I – Fundamentals 7th Edition - Cay S. Hortsman
Pearson Education – Sun Microsystems Gary Cornell

Core Java™ 2 Volume II – Advanced - Cay S. Hortsman
Pearson Education – Sun Microsystems Gary Cornell

Head First Servlets & JSP - Eric Freeman
O'Reilly – SPD Elisabeth Freeman

The Book of JavaScript 2nd Edition - Thau
SPD

Effective Java – Programming Language Guide - Joshua Bloch
Pearson Education – Sun Microsystems

Java Database Best Practices - George Reese
O'Reilly – SPD

JBoss – A Developers Notebook - Norman Richards
O'Reilly – SPD Sam Griffith