**Courier Services System** **INDEX**

1. Introduction

2. System Analysis

a. Existing System

b. proposed System

3. Feasibility Report

a. Technical Feasibility

b. Operational Feasibility

c. Economical Feasibility

4. System Requirement Specification Document

a. Overview

b. Modules Description

c. Process Flow

d. SDLC Methodology

e. Software Requirements

f. Hardware Requirements

5. System Design

a. E-R diagram

b. UML

c. Data Dictionary

6. Technology Description

7. Coding

8. Testing & Debugging Techniques

9. Output Screens

10. Reports

11. Future Enhancements

12. Conclusion

13. Bibliography

INTRODUCTION:-

ABSTRACT

**Objective:**

**Existing System:**

**Proposed System:**

**FEASIBILITY REPORT**

**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.

1. **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.

1. **Find out whether the organization currently possesses the required technologies:**
   * Is the required technology available with the organization?
   * If so is the capacity sufficient?

For instance –

“Will the current printer be able to handle the new reports and forms required for the new system?”

**OPERATIONAL FEASIBILITY:**

Proposed projects are beneficial only if they 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.

**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.

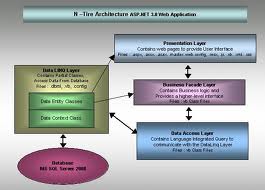
**SYSTEM REQUIREMENT SPECIFICATION**

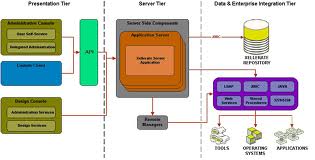
**OVERVIEW**

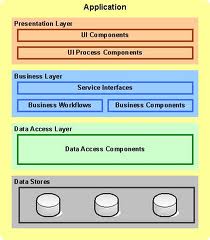
**STUDY OF THE SYSTEM**

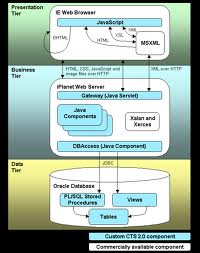
**PROCESS FLOW**

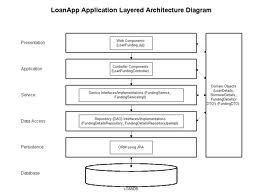
**ARCHITECTURE DIAGRAM**

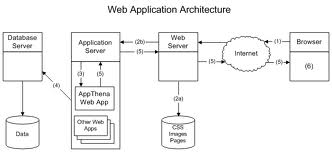
****

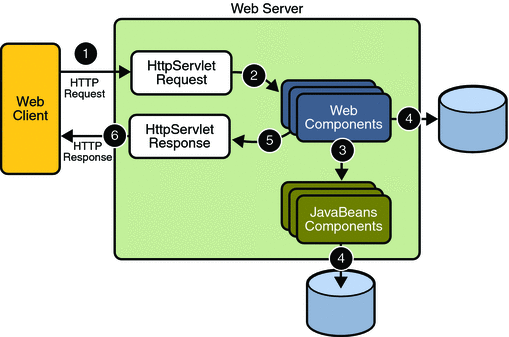
****

****

****

****

****



1. **THE PRESENTATION LAYER**

Also called as the client layer comprises of components that are dedicated to presenting the data to the user. For example: Windows/Web Forms and buttons, edit boxes, Text boxes, labels, grids, etc.

1. **THE BUSINESS RULES LAYER**

This layer encapsulates the Business rules or the business logic of the encapsulations. To have a separate layer for business logic is of a great advantage. This is because any changes in Business Rules can be easily handled in this layer. As long as the interface between the layers remains the same, any changes to the functionality/processing logic in this layer can be made without impacting the others. A lot of client-server apps failed to implement successfully as changing the business logic was a painful process

1. **THE DATA ACCESS LAYER**

This layer comprises of components that help in accessing the Database. If used in the right way, this layer provides a level of abstraction for the database structures. Simply put changes made to the database, tables, etc do not affect the rest of the application because of the Data Access layer. The different application layers send the data requests to this layer and receive the response from this layer.

1. **THE DATABASE LAYER**

This layer comprises of the Database Components such as DB Files, Tables, Views, etc. The Actual database could be created using SQL Server, Oracle, Flat files, etc.   
In an n-tier application, the entire application can be implemented in such a way that it is independent of the actual Database. For instance, you could change the Database Location with minimal changes to Data Access Layer. The rest of the Application should remain unaffected.

**SDLC METHODOLOGIES**

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
2. Defining the requirements of the second prototype.
3. Planning an designing the second prototype.
4. Constructing and testing the second prototype.

* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

**The following diagram shows how a spiral model acts like:**



**Fig 1.0-Spiral Model**

**ADVANTAGES**

* Estimates(i.e. budget, schedule etc .) become more relistic as work progresses, because important issues discoved earlier.
* It is more able to cope with the changes that are software development generally entails.
* Software engineers can get their hands in and start woring on the core of a project earlier.

**SOFTWARE REQUIREMENT AND**

**HARDWARE REQUIREMENT**

**Software Requirements:-**

* Technology : J2SE and J2EE
* Web-Technologies : HTML ,JavaScript ,CSS
* Web Server : Tomcat 6.0
* Java Version : JDK1.6
* Backend Database : Oracle 10G
* IDE : My Eclipse 8.6

**Hardware Requirements:-**

* Processor : Pentium IV
* RAM Capacity : 1GB
* Hard Disk : 40GB

**SYSTEM DESIGN**

##### Data Flow Diagrams

A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know as a data flow graph or a bubble chart.

DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system’s structure charts. The Basic Notation used to create a DFD’s are as follows:

**1. Dataflow:** Data move in a specific direction from an origin to a destination.

**2. Process:** People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.

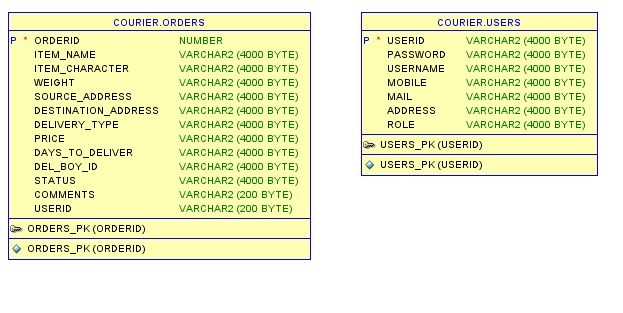
**3. Source:** External sources or destination of data, which may be People, programs, organizations or other entities.

**4. Data Store:** Here data are stored or referenced by a process in the System.

**Authentication Data Flow Diagram:**



**E-R DIAGRAM**

****

**UML DIAGRAMS**

**UNIFIED MODELING LANGUAGE DIAGRAMS**

The unified modeling language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

**USER MODEL VIEW**

This view represents the system from the users perspective.

The analysis representation describes a usage scenario from the end-users perspective.

**STRUCTURAL MODEL VIEW**

In this model the data and functionality are arrived from inside the system.

This model view models the static structures.

**BEHAVIORAL MODEL VIEW**

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

**IMPLEMENTATION MODEL VIEW**

In this the structural and behavioral as parts of the system are represented as they are to be built.

**ENVIRONMENTAL MODEL VIEW**

In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

UML Analysis modeling, which focuses on the user model and structural model views of the system.

UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

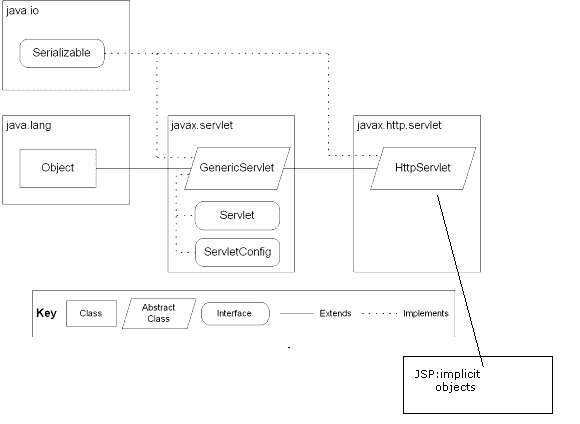
Use case Diagrams represent the functionality of the system from a user’s point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

**Class Diagram**

**CLASS DIAGRAM**

Class diagrams describe the structure of the system in terms of classes and objects. The servlet api class diagram will be as follows.



Use case diagrams:-



Admin use case diagram:-



BOY USE CASE DIAGRAM:-

****

**Customer use case diagram:-**

****

**SEQUENCE DIAGRAMS:-**

****

****

****

**Collaboration diagram:-**

****

****

****

**Activity Diagrams:-**

****

****

****

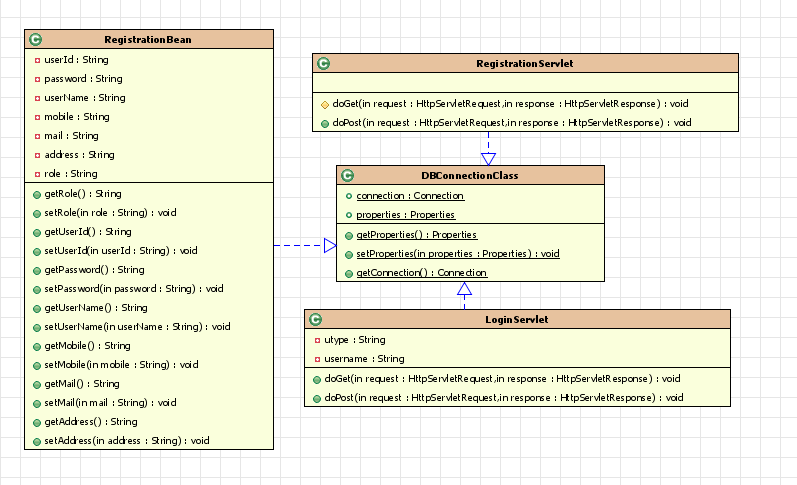
**Component Diagram:**

****

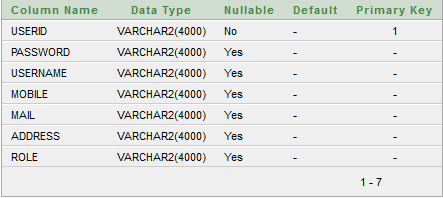
**Deployment Diagram**

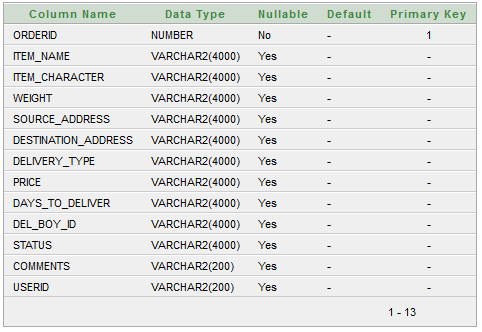
****

**Class diagram:-**

****

**Data Dictionary:-**

****

****

**TECHNOLOGY DESCRIPTION:-**

**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.

**JavaScript**

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 Vs Java

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

# Java Source

## Java byte code

# JavaVM

**Java**

**.Class**

Picture showing 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 Architecture

Java architecture provides a portable, robust, high performing environment for development. Java provides portability by compiling the byte codes for the Java Virtual Machine, which is then interpreted on each platform by the run-time environment. Java is a dynamic system, able to load code when needed from a machine in the same room or across the planet.

#### Compilation of code

When you compile the code, the Java compiler creates machine code (called byte code) for a hypothetical machine called Java Virtual Machine (JVM). The JVM is supposed to execute the byte code. The JVM is created for overcoming the issue of portability. The code is written and compiled for one machine and interpreted on all machines. This machine is called Java Virtual Machine.

Compiling and interpreting Java Source Code

During run-time the Java interpreter tricks the byte code file into thinking that it is running on a Java Virtual Machine. In reality this could be a Intel Pentium Windows 95 or SunSARC station running Solaris or Apple Macintosh running system and all could receive code from any computer through Internet and run the Applets.

**Source**

**Code**

**………..**

**………..**

**………..**

**…………**

# PC Compiler

**Macintosh**

**Compiler**

**SPARC**

###### Compiler

**Java**

**Byte code**

**(Platform**

**Independent)**

**Java**

**Interpreter**

**(PC)**

**Java**

**Interpreter**

**(Macintosh)**

**Java**

**Interpreter**

**(Spare)**

Simple

Java was designed to be easy for the Professional programmer to learn and to use effectively. If you are an experienced C++ programmer, learning Java will be even easier. Because Java inherits the C/C++ syntax and many of the object oriented features of C++. Most of the confusing concepts from C++ are either left out of Java or implemented in a cleaner, more approachable manner. In Java there are a small number of clearly defined ways to accomplish a given task.

**Object-Oriented**

Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean usable, pragmatic approach to objects. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non-objects.

**Robust**

The multi-platform environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. The ability to create robust programs was given a high priority in the design of Java. Java is strictly typed language; it checks your code at compile time and run time.

Java virtually eliminates the problems of memory management and de-allocation, which is completely automatic. In a well-written Java program, all run time errors can –and should –be managed by your program.

**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.

JDBC versus ODBC and other APIs

At this point, Microsoft's ODBC (Open Database Connectivity) API is that probably the most widely used programming interface for accessing relational databases. It offers the ability to connect to almost all databases on almost all platforms.

So why not just use ODBC from Java? The answer is that you can use ODBC from Java, but this is best done with the help of JDBC in the form of the JDBC-ODBC Bridge, which we will cover shortly. The question now becomes "Why do you need JDBC?" There are several answers to this question:

1. ODBC is not appropriate for direct use from Java because it uses a C interface. Calls from Java to native C code have a number of drawbacks in the security, implementation, robustness, and automatic portability of applications.
2. A literal translation of the ODBC C API into a Java API would not be desirable. For example, Java has no pointers, and ODBC makes copious use of them, including the notoriously error-prone generic pointer "void \*". You can think of JDBC as ODBC translated into an object-oriented interface that is natural for Java programmers.
3. ODBC is hard to learn. It mixes simple and advanced features together, and it has complex options even for simple queries. JDBC, on the other hand, was designed to keep simple things simple while allowing more advanced capabilities where required.
4. A Java API like JDBC is needed in order to enable a "pure Java" solution. When ODBC is used, the ODBC driver manager and drivers must be manually installed on every client machine. When the JDBC driver is written completely in Java, however, JDBC code is automatically installable, portable, and secure on all Java platforms from network computers to mainframes.

Two-tier and Three-tier Models

The JDBC API supports both two-tier and three-tier models for database access.

In the two-tier model, a Java applet or application talks directly to the database. This requires a JDBC driver that can communicate with the particular database management system being accessed. A user's SQL statements are delivered to the database, and the results of those statements are sent back to the user. The database may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the database as the server. The network can be an Intranet, which, for example, connects employees within a corporation, or it can be the Internet.

**JAVA**

**Application**

### JDBC

### DBMS

**Client machine**

**DBMS-proprietary protocol**

**Database server**

**Java applet or**

**Html browser**

**Application**

**Server (Java)**

**JDBC**

## DBMS

**Client machine (GUI)**

**HTTP, RMI, or CORBA calls**

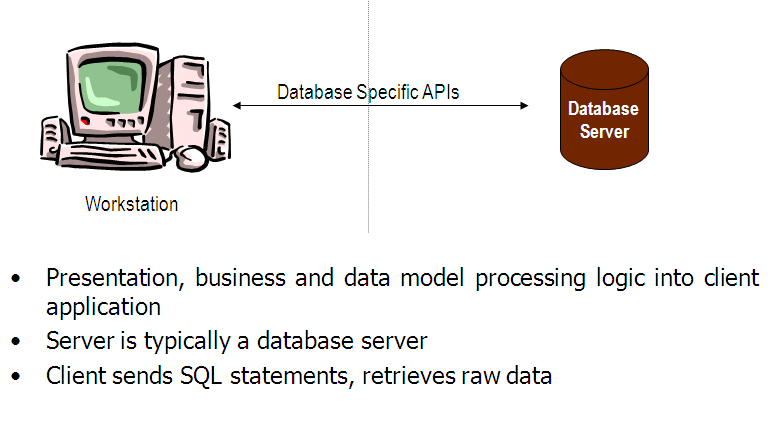
**Server machine (business Logic)**

**DBMS-proprietary protocol**

**Database server**

In the three-tier model, commands are sent to a "middle tier" of services, which then send SQL statements to the database. The database processes the SQL statements and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that when there is a middle tier, the user can employ an easy-to-use higher-level API which is translated by the middle tier into the appropriate low-level calls. Finally, in many cases the three-tier architecture can provide performance advantages.

Until now the middle tier has typically been written in languages such as C or C++, which offer fast performance. However, with the introduction of optimizing compilers that translate Java byte code into efficient machine-specific code, it is becoming practical to implement the middle tier in Java. This is a big plus, making it possible to take advantage of Java's robustness, multithreading, and security features. JDBC is important to allow database access from a Java middle tier.

**JDBC Driver Types**

The JDBC drivers that we are aware of at this time fit into one of four categories:

* JDBC-ODBC bridge plus ODBC driver
* Native-API partly-Java driver
* JDBC-Net pure Java driver
* Native-protocol pure Java driver

**JDBC-ODBC Bridge**

If possible, use a Pure Java JDBC driver instead of the Bridge and an ODBC driver. This completely eliminates the client configuration required by ODBC. It also eliminates the potential that the Java VM could be corrupted by an error in the native code brought in by the Bridge (that is, the Bridge native library, the ODBC driver manager library, the ODBC driver library, and the database client library).

**What Is the JDBC- ODBC Bridge?**

The JDBC-ODBC Bridge is a JDBC driver, which implements JDBC operations by translating them into ODBC operations. To ODBC it appears as a normal application program. The Bridge implements JDBC for any database for which an ODBC driver is available. The Bridge is implemented as the

Sun.jdbc.odbc Java package and contains a native library used to access ODBC. The Bridge is a joint development of Innersole and Java Soft.

**JDBC connectivity**

The JDBC provides database-independent connectivity between the J2EE platform and a wide range of tabular data sources. JDBC technology allows an Application Component Provider to:

* Perform connection and authentication to a database server
* Manager transactions
* Move SQL statements to a database engine for preprocessing and execution
* Execute stored procedures
* Inspect and modify the results from Select statements

**Database:**

A database management system (DBMS) is computer software designed for the purpose of managing databases, a large set of structured data, and run operations on the data requested by numerous users. Typical examples of DBMSs include Oracle, DB2, Microsoft Access, Microsoft SQL Server, Firebird, PostgreSQL, MySQL, SQLite, FileMaker and Sybase Adaptive Server Enterprise. DBMSs are typically used by Database administrators in the creation of Database systems. Typical examples of DBMS use include accounting, human resources and customer support systems.

Originally found only in large companies with the computer hardware needed to support large data sets, DBMSs have more recently emerged as a fairly standard part of any company back office.

**Description**

A DBMS is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. A DBMS includes:

* A modeling language to define the schema of each database hosted in the DBMS, according to the DBMS data model.
* The four most common types of organizations are the hierarchical, network, relational and object models. Inverted lists and other methods are also used. A given database management system may provide one or more of the four models. The optimal structure depends on the natural organization of the application's data, and on the application's requirements (which include transaction rate (speed), reliability, maintainability, scalability, and cost).
* The dominant model in use today is the ad hoc one embedded in SQL, despite the objections of purists who believe this model is a corruption of the relational model, since it violates several of its fundamental principles for the sake of practicality and performance. Many DBMSs also support the Open Database Connectivity API that supports a standard way for programmers to access the DBMS.
* Data structures (fields, records, files and objects) optimized to deal with very large amounts of data stored on a permanent data storage device (which implies relatively slow access compared to volatile main memory).
* A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.
* It also controls the security of the database.
* Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data.
* If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.
* A transaction mechanism, that ideally would guarantee the ACID properties, in order to ensure data integrity, despite concurrent user accesses (concurrency control), and faults (fault tolerance).
* It also maintains the integrity of the data in the database.
* The DBMS can maintain the integrity of the database by not allowing more than one user to update the same record at the same time. The DBMS can help prevent duplicate records via unique index constraints; for example, no two customers with the same customer numbers (key fields) can be entered into the database. See ACID properties for more information (Redundancy avoidance).

The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.

Organizations may use one kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by database administrators.

Database servers are specially designed computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with RAID disk arrays used for stable storage. Connected to one or more servers via a high-speed channel, hardware database accelerators are also used in large volume transaction processing environments.

DBMSs are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system.

**SQL**

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

List of SQL statements that can be issued against an Oracle database schema are:

1. **ALTER** - Change an existing table, view or index definition (DDL)
2. **AUDIT** - Track the changes made to a table (DDL)
3. **COMMENT** - Add a comment to a table or column in a table (DDL)
4. **COMMIT** - Make all recent changes permanent (DML - transactional)
5. **CREATE** - Create new database objects such as tables or views (DDL)
6. **DELETE**- Delete rows from a database table (DML)
7. **DROP** - Drop a database object such as a table, view or index (DDL)
8. **GRANT** - Allow another user to access database objects such as tables or views (DDL)
9. **INSERT** - Insert new data into a database table (DML)
10. **No AUDIT** - Turn off the auditing function (DDL)
11. **REVOKE** - Disallow a user access to database objects such as tables and views (DDL)
12. **ROLLBACK** - Undo any recent changes to the database (DML - Transactional)
13. **SELECT** - Retrieve data from a database table (DML)
14. **TRUNCATE** - Delete all rows from a database table (can not be rolled back) (DML)
15. **UPDATE**- Change the values of some data items in a database table (DML)

**Normalization:**

Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.   
  
Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A customer address change is much easier to implement if that data is stored only in the Customers table and nowhere else in the database.   
  
What is an "inconsistent dependency"? While it is intuitive for a user to look in the Customers table for the address of a particular customer, it may not make sense to look there for the salary of the employee who calls on that customer. The employee's salary is related to, or dependent on, the employee and thus should be moved to the Employees table. Inconsistent dependencies can make data difficult to access because the path to find the data may be missing or broken.   
  
There are a few rules for database normalization. Each rule is called a "normal form." If the first rule is observed, the database is said to be in "first normal form." If the first three rules are observed, the database is considered to be in "third normal form." Although other levels of normalization are possible, third normal form is considered the highest level necessary for most applications.   
  
As with many formal rules and specifications, real world scenarios do not always allow for perfect compliance. In general, normalization requires additional tables and some customers find this cumbersome. If you decide to violate one of the first three rules of normalization, make sure that your application anticipates any problems that could occur, such as redundant data and inconsistent dependencies.   
  
The following descriptions include examples.

**First Normal Form :**

Eliminate repeating groups in individual tables.

Create a separate table for each set of related data.

Identify each set of related data with a primary key.

Do not use multiple fields in a single table to store similar data. For example, to track an inventory item that may come from two possible sources, an inventory record may contain fields for Vendor Code 1 and Vendor Code 2.   
  
What happens when you add a third vendor? Adding a field is not the answer; it requires program and table modifications and does not smoothly accommodate a dynamic number of vendors. Instead, place all vendor information in a separate table called Vendors, then link inventory to vendors with an item number key, or vendors to inventory with a vendor code key.

**Second Normal Form :**

Create separate tables for sets of values that apply to multiple records.

Relate these tables with a foreign key.

Records should not depend on anything other than a table's primary key (a compound key, if necessary). For example, consider a customer's address in an accounting system. The address is needed by the Customers table, but also by the Orders, Shipping, Invoices, Accounts Receivable, and Collections tables. Instead of storing the customer's address as a separate entry in each of these tables, store it in one place, either in the Customers table or in a separate Addresses table.

**Third Normal Form :**

Eliminate fields that do not depend on the key.

Values in a record that are not part of that record's key do not belong in the table. In general, any time the contents of a group of fields may apply to more than a single record in the table, consider placing those fields in a separate table.   
For example, in an Employee Recruitment table, a candidate's university name and address may be included. But you need a complete list of universities for group mailings. If university information is stored in the Candidates table, there is no way to list universities with no current candidates. Create a separate Universities table and link it to the Candidates table with a university code key.   
  
EXCEPTION: Adhering to the third normal form, while theoretically desirable, is not always practical. If you have a Customers table and you want to eliminate all possible interfield dependencies, you must create separate tables for cities, ZIP codes, sales representatives, customer classes, and any other factor that may be duplicated in multiple records. In theory, normalization is worth pursing. However, many small tables may degrade performance or exceed open file and memory capacities.   
  
It may be more feasible to apply third normal form only to data that changes frequently. If some dependent fields remain, design your application to require the user to verify all related fields when any one is changed.

**Other Normalization Forms :**

Fourth normal form, also called Boyce Codd Normal Form (BCNF), and fifth normal form do exist, but are rarely considered in practical design. Disregarding these rules may result in less than perfect database design, but should not affect functionality.

**SERVLETS**

Introduction

The Java web server is JavaSoft's own web Server. The Java web server is just a part of a larger framework, intended to provide you not just with a web server, but also with tools. To build customized network servers for any Internet or Intranet client/server system. Servlets are to a web server, how applets are to the browser.

About Servlets

Servlets provide a Java-based solution used to address the problems currently associated with doing server-side programming, including inextensible scripting solutions, platform-specific APIs, and incomplete interfaces.

Servlets are objects that conform to a specific interface that can be plugged into a Java-based server. Servlets are to the server-side what applets are to the client-side - object byte codes that can be dynamically loaded off the net. They differ from applets in that they are faceless objects (without graphics or a GUI component). They serve as platform independent, dynamically loadable, pluggable helper byte code objects on the server side that can be used to dynamically extend server-side functionality.

For example, an HTTP Servlets can be used to generate dynamic HTML content. When you use Servlets to do dynamic content you get the following advantages:

* They’re faster and cleaner than CGI scripts
* They use a standard API (the Servlets API)
* They provide all the advantages of Java (run on a variety of servers without needing to be rewritten).

Attractiveness of Servlets

There are many features of Servlets that make them easy and attractive to use. These include:

* Easily configured using the GUI-based Admin tool
* Can be loaded and invoked from a local disk or remotely across the network.
* Can be linked together, or chained, so that one Servlets can call another Servlets, or several Servlets in sequence.
* Can be called dynamically from within HTML pages, using server-side include tags.
* Are secure - even when downloading across the network, the Servlets security model and Servlets sandbox protect your system from unfriendly behavior.

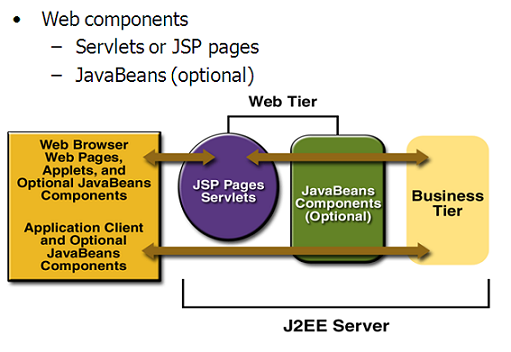
**Advantages of the Servlet API**

One of the great advantages of the Servlet API is protocol independence. It assumes nothing about:

* The protocol being used to transmit on the net
* How it is loaded
* The server environment it will be running in

These qualities are important, because it allows the Servlet API to be embedded in many different kinds of servers. There are other advantages to the Servlet API as well. These include:

* It’s extensible - you can inherit all your functionality from the base classes made available to you.
* It’s simple, small, and easy to use.



**Features of Servlets:**

* Servlets are persistent. Servlet are loaded only by the web server and can maintain services between requests.
* Servlets are fast. Since Servlets only need to be loaded once, they offer much better performance over their CGI counterparts.
* Servlets are platform independent.
* Servlets are extensible. Java is a robust, object-oriented programming language, which easily can be extended to suit your needs
* Servlets are secure.
* Servlets can be used with a variety of clients.

**Loading Servlets:**

Servlets can be loaded from three places

From a directory that is on the CLASSPATH. The CLASSPATH of the JavaWebServer includes service root/classes/ which is where the system classes reside.

From the <SERVICE\_ROOT /Servlets/ directory. This is \*not\* in the server’s class path. A class loader is used to create Servlets from this directory. New Servlets can be added - existing Servlets can be recompiled and the server will notice these changes.

From a remote location, for this a code base like http: // nine.eng / classes / foo / is required in addition to the Servlets class name. Refer to the admin GUI docs on Servlet section to see how to set this up.

**Loading Remote Servlets**

Remote Servlets can be loaded by:

1. Configuring the Admin Tool to setup automatic loading of remote Servlets
2. Setting up server side include tags in. shtml files
3. Defining a filter chain configuration

**Invoking Servlets**

A Servlet invoker is a Servlet that invokes the "service" method on a named Servlet. If the Servlet is not loaded in the server, then the invoker first loads the Servlet (either from local disk or from the network) and the then invokes the "service" method. Also like applets, local Servlets in the server can be identified by just the class name. In other words, if a Servlet name is not absolute, it is treated as local.

A client can invoke Servlets in the following ways**:**

* The client can ask for a document that is served by the Servlet.
* The client (browser) can invoke the Servlet directly using a URL, once it has been mapped using the [Servlet Aliases](../administration/servlet_alias.html) section of the admin GUI.
* The Servlet can be invoked through [server side include](ssinclude.html) tags.
* The Servlet can be invoked by placing it in the Servlets/ directory.
* The Servlet can be invoked by using it in a filter chain.

**Java Server Pages (JSP)**

Java server Pages is a simple, yet powerful technology for creating and maintaining dynamic-content web pages. Based on the Java programming language, Java Server Pages offers proven portability, open standards, and a mature re-usable component model .The Java Server Pages architecture enables the separation of content generation from content presentation. This separation not eases maintenance headaches; it also allows web team members to focus on their areas of expertise. Now, web page designer can concentrate on layout, and web application designers on programming, with minimal concern about impacting each other’s work.

Features of JSP

Portability:

Java Server Pages files can be run on any web server or web-enabled application server that provides support for them. Dubbed the JSP engine, this support involves recognition, translation, and management of the Java Server Page lifecycle and its interaction components.

Components

It was mentioned earlier that the Java Server Pages architecture can include reusable Java components. The architecture also allows for the embedding of a scripting language directly into the Java Server Pages file. The components current supported include Java Beans, and Servlets.

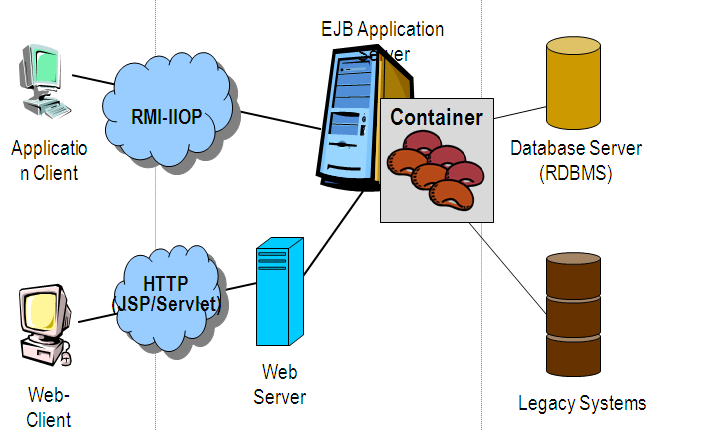
Processing

A Java Server Pages file is essentially an HTML document with JSP scripting or tags. The Java Server Pages file has a JSP extension to the server as a Java Server Pages file. Before the page is served, the Java Server Pages syntax is parsed and processed into a Servlet on the server side. The Servlet that is generated outputs real content in straight HTML for responding to the client.

Access Models:

A Java Server Pages file may be accessed in at least two different ways. A client’s request comes directly into a Java Server Page. In this scenario, suppose the page accesses reusable Java Bean components that perform particular well-defined computations like accessing a database. The result of the Beans computations, called result sets is stored within the Bean as properties. The page uses such Beans to generate dynamic content and present it back to the client.

In both of the above cases, the page could also contain any valid Java code. Java Server Pages architecture encourages separation of content from presentation.



Steps in the execution of a JSP Application:

1. The client sends a request to the web server for a JSP file by giving the name of the JSP file within the form tag of a HTML page.
2. This request is transferred to the JavaWebServer. At the server side JavaWebServer receives the request and if it is a request for a jsp file server gives this request to the JSP engine.
3. JSP engine is program which can under stands the tags of the jsp and then it converts those tags into a Servlet program and it is stored at the server side. This Servlet is loaded in the memory and then it is executed and the result is given back to the JavaWebServer and then it is transferred back to the result is given back to the JavaWebServer and then it is transferred back to the client.

**Eclipse IDE**

Eclipse is an open-source software framework written primarily in Java. In its default form it is an Integrated Development Environment (IDE) for Java developers, consisting of the Java Development Tools (JDT) and the Eclipse Compiler for Java (ECJ). Users can extend its capabilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. Language packs are available for over a dozen languages.

**Architecture**

The basis for Eclipse is the Rich Client Platform (RCP). The following components constitute the rich client platform:

* OSGi - a standard bundling framework
* Core platform - boot Eclipse, run plug-ins
* the Standard Widget Toolkit (SWT) - a portable widget toolkit
* JFace - viewer classes to bring model view controller programming to SWT, file buffers, text handling, text editors
* the Eclipse Workbench - views, editors, perspectives, wizards

Eclipse's widgets are implemented by a widget toolkit for Java called SWT, unlike most Java applications, which use the Java standard Abstract Window Toolkit (AWT) or Swing. Eclipse's user interface also leverages an intermediate GUI layer called JFace, which simplifies the construction of applications based on SWT.

Eclipse employs plug-ins in order to provide all of its functionality on top of (and including) the rich client platform, in contrast to some other applications where functionality is typically hard coded. This plug-in mechanism is a lightweight software componentry framework. In addition to allowing Eclipse to be extended using other programming languages such as C and Python, the plug-in framework allows Eclipse to work with typesetting languages like LaTeX, networking applications such as telnet, and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. Java and CVS support is provided in the Eclipse SDK.

The key to the seamless integration of tools with Eclipse is the plugin. With the exception of a small run-time kernel, everything in Eclipse is a plug-in. This means that a plug-in you develop integrates with Eclipse in exactly the same way as other plug-ins; in this respect, all features are created equal.

The Eclipse SDK includes the Eclipse Java Development Tools, offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. The IDE also makes use of a workspace, in this case a set of metadata over a flat filespace allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards. The Visual Editor project allows interfaces to be created interactively, hence allowing Eclipse to be used as a RAD tool.

The following is a list of notable projects and plugins for the Eclipse IDE.

These projects are maintained by the Eclipse community and hosted by the Eclipse Foundation.

**CODING:-**

**LOGIN SERVELET .JAVA**

**package com.nit.controller;**

**import java.io.IOException;**

**import java.io.PrintWriter;**

**import javax.servlet.RequestDispatcher;**

**import javax.servlet.ServletException;**

**import javax.servlet.http.HttpServlet;**

**import javax.servlet.http.HttpServletRequest;**

**import javax.servlet.http.HttpServletResponse;**

**import javax.servlet.http.HttpSession;**

**import com.nit.beans.RegistrationBean;**

**import com.nit.dao.UserDAO;**

**public class LoginServlet extends HttpServlet {**

**private String utype ="";**

**private String username = "";**

**public void doGet(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException {**

**UserDAO dao = new UserDAO();**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**HttpSession session = request.getSession();**

**RegistrationBean rb = new RegistrationBean();**

**session.setAttribute("userId", request.getParameter("userId"));**

**String target = "login.jsp?status=Internal Proublem Please Try again!";**

**try**

**{**

**rb.setUserId(request.getParameter("userId"));**

**rb.setPassword(request.getParameter("password"));**

**RegistrationBean rb1 = new RegistrationBean();**

**rb1 = dao.loginCheck(rb);**

**utype = rb1.getRole();**

**username = rb1.getUserName();**

**System.out.println("utype===="+utype);**

**if (utype.equals("ADMIN")) {**

**/\*target = "adminhome.jsp";\*/**

**target = "index.jsp";**

**session.setAttribute("user", username);**

**session.setAttribute("role", utype);**

**}**

**else if (utype.equals("BOY")) {**

**target = "index.jsp";**

**/\*target = "BoyHome.jsp";\*/**

**session.setAttribute("user", username);**

**session.setAttribute("role", utype);**

**} else if(utype.equals("CUSTOMER"))**

**{**

**target = "index.jsp";**

**/\* target = "CustomerHome.jsp";\*/**

**session.setAttribute("user", username);**

**session.setAttribute("role", utype);**

**}**

**else**

**{**

**target = "login.jsp?status=Invalid username or password";**

**}**

**}**

**catch(Exception e)**

**{**

**target = "login.jsp?status=Invalid username or password";**

**}**

**RequestDispatcher rd = request.getRequestDispatcher(target);**

**rd.forward(request, response);**

**out.flush();**

**out.close();**

**}**

**public void doPost(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException {**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**doGet(request, response);**

**out.flush();**

**out.close();**

**}**

**}**

**USER DAO.JAVA**

**package com.nit.dao;**

**import java.io.File;**

**import java.io.FileInputStream;**

**import java.io.FileNotFoundException;**

**import java.io.InputStream;**

**import java.sql.Blob;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.util.ArrayList;**

**import java.util.HashMap;**

**import java.util.List;**

**import com.nit.beans.OrderBean;**

**import com.nit.beans.RegistrationBean;**

**import com.nit.util.Constants;**

**import com.nit.util.DBConnectionClass;**

**import com.nit.util.SqlConstants;**

**public class UserDAO extends DBConnectionClass {**

**public static Connection con;**

**PreparedStatement ps,pst;**

**private boolean flag;**

**public UserDAO() {**

**con = getConnection();**

**}**

**public boolean checkLoginIDExisted(String loginid) {**

**boolean f = false;**

**try {**

**con = getConnection();**

**ps = con**

**.prepareStatement(SqlConstants.SQL\_CHECK\_LOGIN);**

**ps.setString(1, loginid);**

**ResultSet rs = ps.executeQuery();**

**while (rs.next()) {**

**f = true;**

**}**

**ps.close();**

**rs.close();**

**}**

**catch (Exception e) {**

**}**

**return f;**

**}**

**public boolean insertUserInfo(RegistrationBean reb) {**

**try {**

**int i = 0;**

**pst = con**

**.prepareStatement(SqlConstants.SQL\_INSERT\_USER\_INFO);**

**pst.setString(1, reb.getUserId());**

**pst.setString(2, reb.getPassword());**

**pst.setString(3, reb.getUserName());**

**pst.setString(4, reb.getMobile());**

**pst.setString(5, reb.getMail());**

**pst.setString(6, reb.getAddress());**

**pst.setString(7, reb.getRole());**

**i = pst.executeUpdate();**

**if (i == 1) {**

**flag = true;**

**con.commit();**

**} else {**

**flag = false;**

**con.rollback();**

**}**

**pst.close();**

**} catch (SQLException ex) {**

**ex.printStackTrace();**

**flag = false;**

**try {**

**con.rollback();**

**} catch (SQLException sex1) {**

**sex1.printStackTrace();**

**}**

**} catch (Exception e) {**

**e.printStackTrace();**

**flag = false;**

**try {**

**con.rollback();**

**} catch (SQLException se) {**

**se.printStackTrace();**

**}**

**}**

**return flag;**

**}**

**public RegistrationBean loginCheck(RegistrationBean regbean)**

**{**

**String loginid = regbean.getUserId();**

**String password = regbean.getPassword();**

**RegistrationBean regbean1 = new RegistrationBean();**

**try {**

**pst = con**

**.prepareStatement(SqlConstants.SQL\_LOGIN\_CHECK);**

**pst.setString(1, loginid);**

**pst.setString(2, password);**

**ResultSet rs = pst.executeQuery();**

**if (rs.next()) {**

**regbean1.setRole(rs.getString(1));**

**regbean1.setUserName(rs.getString(2));**

**} else {**

**flag = false;**

**regbean1.setRole("");**

**regbean1.setUserName("");**

**}**

**pst.close();**

**} catch (SQLException ex) {**

**System.out.println(ex);**

**flag = false;**

**}**

**return regbean1;**

**}**

**public int sendCustomerOrderRequest(OrderBean ob) {**

**int i = 0;**

**try {**

**pst = con**

**.prepareStatement(SqlConstants.SQL\_INSERT\_CUSTOMER\_REQUEST);**

**pst.setString(1, ob.getItemName());**

**pst.setString(2,ob.getItemCharacter());**

**pst.setString(3,ob.getWeight());**

**pst.setString(4,ob.getSourceAddress());**

**pst.setString(5,ob.getDestinationAddress());**

**pst.setString(6,ob.getDeliveryType());**

**pst.setString(7,ob.getStatus());**

**pst.setString(8, ob.getUserId());**

**i=pst.executeUpdate();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return i;**

**}**

**public List<OrderBean> showCustomerOrderRequest()**

**{**

**List<OrderBean> list=new ArrayList<OrderBean>();**

**try**

**{**

**pst=con.prepareStatement("select \* from orders where status=?");**

**pst.setString(1, Constants.PENDING);**

**ResultSet rs=pst.executeQuery();**

**while(rs.next())**

**{**

**OrderBean ob=new OrderBean();**

**ob.setDeliveryType(rs.getString(7));**

**ob.setDestinationAddress(rs.getString(6));**

**ob.setItemCharacter(rs.getString(3));**

**ob.setItemName(rs.getString(2));**

**ob.setOrderId(rs.getInt(1));**

**ob.setSourceAddress(rs.getString(5));**

**ob.setWeight(rs.getString(4));**

**list.add(ob);**

**}**

**pst.close();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return list;**

**}**

**public List<OrderBean> shoBoyOrders(String userId)**

**{**

**List<OrderBean> list=new ArrayList<OrderBean>();**

**try**

**{**

**pst=con.prepareStatement("select \* from orders where del\_boy\_id=? and status=?");**

**pst.setString(1, userId);**

**pst.setString(2, Constants.BOY\_ALLOCATED);**

**ResultSet rs=pst.executeQuery();**

**while(rs.next())**

**{**

**OrderBean ob=new OrderBean();**

**ob.setDeliveryType(rs.getString(7));**

**ob.setDestinationAddress(rs.getString(6));**

**ob.setItemCharacter(rs.getString(3));**

**ob.setItemName(rs.getString(2));**

**ob.setOrderId(rs.getInt(1));**

**ob.setSourceAddress(rs.getString(5));**

**ob.setWeight(rs.getString(4));**

**ob.setNoOfDays(rs.getString(9));**

**ps=con.prepareStatement("select \* from users where userid=?");**

**ps.setString(1, rs.getString(13));**

**ResultSet rss=ps.executeQuery();**

**if(rss.next())**

**{**

**RegistrationBean rb=new RegistrationBean();**

**rb.setAddress(rss.getString(6));**

**rb.setMail(rss.getString(5));**

**rb.setMobile(rss.getString(4));**

**rb.setUserName(rss.getString(3));**

**ob.setRb(rb);**

**}**

**ps.close();**

**list.add(ob);**

**}**

**pst.close();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return list;**

**}**

**public List<OrderBean> showCustomerOrderStatus(String userId)**

**{**

**List<OrderBean> list=new ArrayList<OrderBean>();**

**try**

**{**

**pst=con.prepareStatement("select \* from orders where (status=? or status=? or status=? or status=?) and userId=?");**

**pst.setString(1, Constants.CUSTOMER\_CONFIRM\_PENDING);**

**pst.setString(2, Constants.REJECTED);**

**pst.setString(3, Constants.BOY\_ALLOCATED);**

**pst.setString(4, Constants.DELIVERED);**

**pst.setString(5,userId);**

**ResultSet rs=pst.executeQuery();**

**while(rs.next())**

**{**

**OrderBean ob=new OrderBean();**

**ob.setDeliveryType(rs.getString(7));**

**ob.setDestinationAddress(rs.getString(6));**

**ob.setItemCharacter(rs.getString(3));**

**ob.setItemName(rs.getString(2));**

**ob.setOrderId(rs.getInt(1));**

**ob.setSourceAddress(rs.getString(5));**

**ob.setWeight(rs.getString(4));**

**ob.setPrice(rs.getString(8));**

**ob.setNoOfDays(rs.getString(9));**

**ob.setComment(rs.getString(12));**

**ob.setStatus(rs.getString(11));**

**list.add(ob);**

**}**

**pst.close();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return list;**

**}**

**public List<OrderBean> showCustomerAcceptedOrders()**

**{**

**List<OrderBean> list=new ArrayList<OrderBean>();**

**try**

**{**

**pst=con.prepareStatement("select \* from orders where status=?");**

**pst.setString(1, Constants.CUSTOMER\_ACCEPTED);**

**ResultSet rs=pst.executeQuery();**

**while(rs.next())**

**{**

**OrderBean ob=new OrderBean();**

**ob.setDeliveryType(rs.getString(7));**

**ob.setDestinationAddress(rs.getString(6));**

**ob.setItemCharacter(rs.getString(3));**

**ob.setItemName(rs.getString(2));**

**ob.setOrderId(rs.getInt(1));**

**ob.setSourceAddress(rs.getString(5));**

**ob.setWeight(rs.getString(4));**

**ob.setPrice(rs.getString(8));**

**ob.setNoOfDays(rs.getString(9));**

**ob.setComment(rs.getString(12));**

**ob.setStatus(rs.getString(11));**

**list.add(ob);**

**}**

**pst.close();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return list;**

**}**

**public String acceptAdminOrder(String price, String days,String id,String comment) {**

**String status="";**

**try**

**{**

**pst=con.prepareStatement("update orders set price=? , days\_to\_deliver=?,status=?,comments=? where orderid=?");**

**pst.setString(1, price);**

**pst.setString(2, days);**

**pst.setString(3,Constants.CUSTOMER\_CONFIRM\_PENDING);**

**pst.setString(4, comment);**

**pst.setInt(5, Integer.parseInt(id));**

**int i=pst.executeUpdate();**

**if(i>0)**

**{**

**status="Successfully Accepted";**

**}**

**else**

**{**

**status="Something Went Wrong";**

**}**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return status;**

**}**

**public String rejectAdminOrder(String id, String comment) {**

**String status="";**

**try**

**{**

**pst=con.prepareStatement("update orders set status=?,comments=? where orderid=?");**

**pst.setString(1,Constants.REJECTED);**

**pst.setString(2, comment);**

**pst.setInt(3, Integer.parseInt(id));**

**int i=pst.executeUpdate();**

**if(i>0)**

**{**

**status="Successfully Accepted";**

**}**

**else**

**{**

**status="Something Went Wrong";**

**}**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return status;**

**}**

**public String customerAcept(String parameter) {**

**String status="";**

**try**

**{**

**pst=con.prepareStatement("update orders set status=? where orderid=?");**

**pst.setString(1,Constants.CUSTOMER\_ACCEPTED);**

**pst.setString(2, parameter);**

**int i=pst.executeUpdate();**

**if(i>0)**

**{**

**status="Successfully Accepted";**

**}**

**else**

**{**

**status="Something Went Wrong";**

**}**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return status;**

**}**

**public List<RegistrationBean> getBoys()**

**{**

**List<RegistrationBean> list=new ArrayList<RegistrationBean>();**

**try**

**{**

**pst=con.prepareStatement("select \* from users where userid not in (select del\_boy\_id from orders where status='ALLOCATED') and role='BOY'");**

**ResultSet rs=pst.executeQuery();**

**while(rs.next())**

**{**

**RegistrationBean rb=new RegistrationBean();**

**rb.setUserId(rs.getString(1));**

**rb.setUserName(rs.getString(3));**

**rb.setMobile(rs.getString(4));**

**rb.setMail(rs.getString(5));**

**list.add(rb);**

**}**

**pst.close();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return list;**

**}**

**public String allocateBoy(String user, String oid) {**

**String status="";**

**try**

**{**

**pst=con.prepareStatement("update orders set status=?,del\_boy\_id=? where orderid=?");**

**pst.setString(1,Constants.BOY\_ALLOCATED);**

**pst.setString(2, user);**

**pst.setInt(3,Integer.parseInt(oid));**

**int i=pst.executeUpdate();**

**if(i>0)**

**{**

**status="Successfully Allocated";**

**}**

**else**

**{**

**status="Something Went Wrong";**

**}**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return status;**

**}**

**public RegistrationBean getAllocatedBoy(int id) {**

**RegistrationBean rb=new RegistrationBean();**

**try**

**{**

**pst=con.prepareStatement("select \* from users where userid=(select del\_boy\_id from orders where orderid=?)");**

**pst.setInt(1, id);**

**ResultSet rs=pst.executeQuery();**

**if(rs.next())**

**{**

**rb.setUserId(rs.getString(1));**

**rb.setUserName(rs.getString(3));**

**rb.setMobile(rs.getString(4));**

**rb.setMail(rs.getString(5));**

**}**

**pst.close();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return rb;**

**}**

**public int boyOrderStatus(String orderId, String comment) {**

**try**

**{**

**if(comment.contains(Constants.DELIVERED))**

**{**

**pst=con.prepareStatement("update orders set comments=?,status=? where orderId=?");**

**pst.setString(1, comment);**

**pst.setString(3,orderId);**

**pst.setString(2,Constants.DELIVERED);**

**}**

**else**

**{**

**pst=con.prepareStatement("update orders set comments=? where orderId=?");**

**pst.setString(1, comment);**

**pst.setString(2,orderId);**

**}**

**return pst.executeUpdate();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**return 0;**

**}**

**}**

**REGISTRATION.JAVA**

**package com.nit.controller;**

**import java.io.IOException;**

**import java.io.PrintWriter;**

**import javax.servlet.RequestDispatcher;**

**import javax.servlet.ServletException;**

**import javax.servlet.http.HttpServlet;**

**import javax.servlet.http.HttpServletRequest;**

**import javax.servlet.http.HttpServletResponse;**

**import com.nit.beans.RegistrationBean;**

**import com.nit.dao.UserDAO;**

**public class RegistrationServlet extends HttpServlet**

**{**

**@Override**

**protected void doGet(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException**

**{**

**/\*response.getWriter().print("hello...");\*/**

**UserDAO dao = new UserDAO();**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**RegistrationBean rb = new RegistrationBean();**

**rb.setUserId(request.getParameter("userId"));**

**rb.setUserName(request.getParameter("userName"));**

**rb.setPassword(request.getParameter("password"));**

**rb.setMobile(request.getParameter("mobile"));**

**rb.setMail(request.getParameter("mail"));**

**int i=0;**

**rb.setAddress(request.getParameter("address"));**

**if(request.getParameter("boy")!=null && request.getParameter("boy").equals("1"))**

**{**

**i=1;**

**rb.setRole("BOY");**

**}**

**else**

**{**

**rb.setRole("CUSTOMER");**

**}**

**System.out.println("address---"+rb.getAddress());**

**String target = "registration.jsp?status=Registration Failed !!!!!";**

**boolean flag1 = dao.checkLoginIDExisted(request.getParameter("userId"));**

**if (flag1 == true) {**

**target = "registration.jsp?status=" + request.getParameter("userId")**

**+ " Already Registered";**

**else {**

**boolean f1 = dao.insertUserInfo(rb);**

**if (f1 == true) {**

**if(i!=1)**

**{**

**target = "login.jsp?status="+request.getParameter("userId")+" Account created successfully";**

**}**

**else**

**{**

**target = "uploadboy.jsp?status="+request.getParameter("userId")+" Account created successfully";**

**}**

**} else {**

**target = "registration.jsp?status=Registration Failed !!!!!";**

**}**

**}**

**RequestDispatcher rd = request.getRequestDispatcher(target);**

**rd.forward(request, response);**

**out.flush();**

**out.close();**

**}**

**public void doPost(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException {**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**doGet(request, response);**

**out.flush();**

**out.close();**

**}**

**}**

**TESTING**

Software Testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software. Testing is a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors. Quality is not an absolute; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behavior of the product against a specification. An important point is that software testing should be distinguished from the separate discipline of Software Quality Assurance (SQA), which encompasses all business process areas, not just testing.

There are many approaches to software testing, but effective testing of complex products is essentially a process of investigation, not merely a matter of creating and following routine procedure. One definition of testing is "the process of questioning a product in order to evaluate it", where the "questions" are operations the tester attempts to execute with the product, and the product answers with its behavior in reaction to the probing of the tester[citation needed]. Although most of the intellectual processes of testing are nearly identical to that of review or inspection, the word testing is connoted to mean the dynamic analysis of the product—putting the product through its paces. Some of the common quality attributes include capability, reliability, efficiency, portability, maintainability, compatibility and usability. A good test is sometimes described as one which reveals an error; however, more recent thinking suggests that a good test is one which reveals information of interest to someone who matters within the project community.

**Introduction:**

In general, software engineers distinguish software faults from software failures. In case of a failure, the software does not do what the user expects. A fault is a programming error that may or may not actually manifest as a failure. A fault can also be described as an error in the correctness of the semantic of a computer program. A fault will become a failure if the exact computation conditions are met, one of them being that the faulty portion of computer software executes on the CPU. A fault can also turn into a failure when the software is ported to a different hardware platform or a different compiler, or when the software gets extended. Software testing is the technical investigation of the product under test to provide stakeholders with quality related information.

Software testing may be viewed as a sub-field of Software Quality Assurance but typically exists independently (and there may be no SQA areas in some companies). In SQA, software process specialists and auditors take a broader view on software and its development. They examine and change the software engineering process itself to reduce the amount of faults that end up in the code or deliver faster.

Regardless of the methods used or level of formality involved the desired result of testing is a level of confidence in the software so that the organization is confident that the software has an acceptable defect rate. What constitutes an acceptable defect rate depends on the nature of the software. An arcade video game designed to simulate flying an airplane would presumably have a much higher tolerance for defects than software used to control an actual airliner.

A problem with software testing is that the number of defects in a software product can be very large, and the number of configurations of the product larger still. Bugs that occur infrequently are difficult to find in testing. A rule of thumb is that a system that is expected to function without faults for a certain length of time must have already been tested for at least that length of time. This has severe consequences for projects to write long-lived reliable software.

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. 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.

It is commonly believed that the earlier a defect is found the cheaper it is to fix it.

Unit tests are maintained along with the rest of the software source code and generally integrated into the build process (with inherently interactive tests being relegated to a partially manual build acceptance process).

The software, tools, samples of data input and output, and configurations are all referred to collectively as a test harness.

History

The separation of debugging from testing was initially introduced by Glen ford J. Myers in his 1978 book the "Art of Software Testing". Although his attention was on breakage testing it illustrated the desire of the software engineering community to separate fundamental development activities, such as debugging, from that of verification. Drs. Dave Gelperin and William C. Hetzel classified in 1988 the phases and goals in software testing as follows: until 1956 it was the debugging oriented period, where testing was often associated to debugging: there was no clear difference between testing and debugging. From 1957-1978 there was the demonstration oriented period where debugging and testing was distinguished now - in this period it was shown, that software satisfies the requirements. The time between 1979-1982 is announced as the destruction oriented period, where the goal was to find errors. 1983-1987 is classified as the evaluation oriented period: intention here is that during the software lifecycle a product evaluation is provided and measuring quality. From 1988 on it was seen as prevention oriented period where tests were to demonstrate that software satisfies its specification, to detect faults and to prevent faults. Dr. Gelperin chaired the IEEE 829-1988 (Test Documentation Standard) with Dr. Hetzel writing the book "The Complete Guide of Software Testing". Both works were pivotal in to today's testing culture and remain a consistent source of reference. Dr. Gelperin and Jerry E. Durant also went on to develop High Impact Inspection Technology that builds upon traditional Inspections but utilizes a test driven additive.

Testing Concepts

* ***Testing***
* ***Testing Methodologies***
* Black box Testing:
* White box Testing.
* Gray Box Testing.
* ***Levels of Testing***
  + Unit Testing.
  + Module Testing.
  + Integration Testing.
  + System Testing.
  + User Acceptance Testing.
* ***Types Of Testing***
  + Smoke Testing.
  + Sanitary Testing.
  + Regression Testing.
  + Re-Testing.
  + Static Testing.
  + Dynamic Testing.
  + Alpha-Testing.
  + Beta-Testing.
  + Monkey Testing.
  + Compatibility Testing.
  + Installation Testing.
  + Adhoc Testing.
  + Ext….

***TCD (Test Case Documentation)***

* ***STLC***
  + Test Planning.
  + Test Development.
  + Test Execution.
  + Result Analysis.
  + Bug-Tracing.
  + Reporting.
* ***Microsoft Windows – Standards***
* ***Manual Testing***
* ***Automation Testing (Tools)***
  + Win Runner.
  + Test Director.

**Testing:**

* The process of executing a system with the intent of finding an error.
* Testing is defined as the process in which defects are identified, isolated, subjected for rectification and ensured that product is defect free in order to produce the quality product and hence customer satisfaction.
* Quality is defined as justification of the requirements
* Defect is nothing but deviation from the requirements
* Defect is nothing but bug.
* Testing --- The presence of bugs
* Testing can demonstrate the presence of bugs, but not their absence
* Debugging and Testing are not the same thing!
* Testing is a systematic attempt to break a program or the AUT
* Debugging is the art or method of uncovering why the script /program did not execute properly.

**Testing Methodologies:**

* **Black box Testing**: is the testing process in which tester can perform testing on an application without having any internal structural knowledge of application.

Usually Test Engineers are involved in the black box testing.

* **White box Testing**: is the testing process in which tester can perform testing on an application with having internal structural knowledge.

Usually The Developers are involved in white box testing.

* **Gray Box Testing**: is the process in which the combination of black box and white box tonics’ are used.

**Levels of Testing:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ***Module1*** ***Module2*** ***Module3***   |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ***Units***   |  |  |  | | --- | --- | --- | |  |  |  | |  | ***Units***   |  |  |  | | --- | --- | --- | |  |  |  | |  | ***Units***   |  |  |  | | --- | --- | --- | |  |  |  | |   i/p ***Integration***  o/p i/p ***Integration o/p*** |   ***System Testing: Presentation + business +Databases***  ***🚹UAT: user acceptance testing*** |

##### STLC (SOFTWARE TESTING LIFE CYCLE)

**Test Planning:**

**1.**Test Plan is defined as a strategic document which describes the procedure how to perform various testing on the total application in the most efficient way.

**2.**This document involves the scope of testing,

**3.** Objective of testing,

**4.** Areas that need to be tested,

**5.** Areas that should not be tested,

**6.** Scheduling Resource Planning,

**7.** Areas to be automated, various testing tools

Used….

**Test Development**:

**1.** Test case Development (check list)

**2.** Test Procedure preparation. (Description of the Test cases).

**1.** Implementation of test cases. Observing the result.

**Result Analysis**: **1.** Expected value: is nothing but expected behavior

Of application.

**2.** Actual value: is nothing but actual behavior of

application

**Bug Tracing:**  Collect all the failed cases, prepare documents.

**Reporting:** Prepare document (status of the application)

**Types Of Testing:**

**🚺>** **Smoke Testing**: is the process of initial testing in which tester looks for the availability of all the functionality of the application in order to perform detailed testing on them. (Main check is for available forms)

**🚺>** **Sanity Testing:** is a type of testing that is conducted on an application initially to check for the proper behavior of an application that is to check all the functionality are available before the detailed testing is conducted by on them.

**🚺>** **Regression Testing:** is one of the best and important testing. Regression testing is the process in which the functionality, which is already tested before, is once again tested whenever some new change is added in order to check whether the existing functionality remains same.

**🚺>Re-Testing:** is the process in which testing is performed on some functionality which is already tested before to make sure that the defects are reproducible and to rule out the environments issues if at all any defects are there.

**🚺Static Testing:** is the testing, which is performed on an application when it is not been executed.ex: GUI, Document Testing

**🚺Dynamic Testing:** is the testing which is performed on an application when it is being executed.ex: Functional testing.

**🚺Alpha Testing:** it is a type of user acceptance testing, which is conducted on an application when it is just before released to the customer.

**🚺 Beta-Testing:** it is a type of UAT that is conducted on an application when it is released to the customer, when deployed in to the real time environment and being accessed by the real time users.

**🚺 Monkey Testing:** is the process in which abnormal operations, beyond capacity operations are done on the application to check the stability of it in spite of the users abnormal behavior.

**🚺Compatibility testing:** it is the testing process in which usually the products are tested on the environments with different combinations of databases (application servers, browsers…etc) In order to check how far the product is compatible with all these environments platform combination.

**🚺Installation Testing:** it is the process of testing in which the tester try to install or try to deploy the module into the corresponding environment by following the guidelines produced in the deployment document and check whether the installation is successful or not.

**🚺Adhoc Testing:** Adhoc Testing is the process of testing in which unlike the formal testing where in test case document is used, with out that test case document testing can be done of an application, to cover that testing of the future which are not covered in that test case document. Also it is intended to perform GUI testing which may involve the cosmotic issues.

**TCD (Test Case Document:**

**Test Case Document Contains**

* **Test Scope (or) Test objective**
* **Test Scenario**
* **Test Procedure**
* **Test case**

This is the sample test case document for the Acadamic details of student project:

**Test scope:**

* Test coverage is provided for the screen “ Acadamic status entry” form of a student module of university management system application
* Areas of the application to be tested

**Test Scenario:**

* When the office personals use this screen for the marks entry, calculate the status details, saving the information on student’s basis and quit the form.

**Test Procedure:**

* The procedure for testing this screen is planned in such a way that the data entry, status calculation functionality, saving and quitting operations are tested in terms of Gui testing, Positive testing, Negative testing using the corresponding Gui test cases, Positive test cases, Negative test cases respectively

**Test Cases:**

* Template for Test Case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T.C.No | Description | Exp | Act | Result |
| 1  2 | Enter user name and password  Enter valid date to store in the database | True/false  Accurate/Valid  data | True  Valid date | Home page  Data stored  successfully |

**Guidelines for Test Cases**:

1. **GUI Test Cases:**

* Total no of features that need to be check
* Look & Feel
* Look for Default values if at all any (date & Time, if at all any require)
* Look for spell check

***Example for Gui Test cases***:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T.C.No | Description | Expected value | Actual value | Result |
| 1 | Check for all the features in the screen | The screen must contain all the features |  |  |
| 2 | Check for the alignment of the objects as per the validations | The alignment should be in proper way |  |  |

1. **Positive Test Cases:**

* The positive flow of the functionality must be considered
* Valid inputs must be used for testing
* Must have the positive perception to verify whether the requirements are justified.

***Example for Positive Test cases:***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T.C.No | Description | Expected value | Actual value | Result |
| 1 | Check for the date Time Auto Display | The date and time of the system must be displayed |  |  |
| 2 | Enter the valid Roll no into the student roll no field | It should accept |  |  |

1. **Negative Test Cases:**

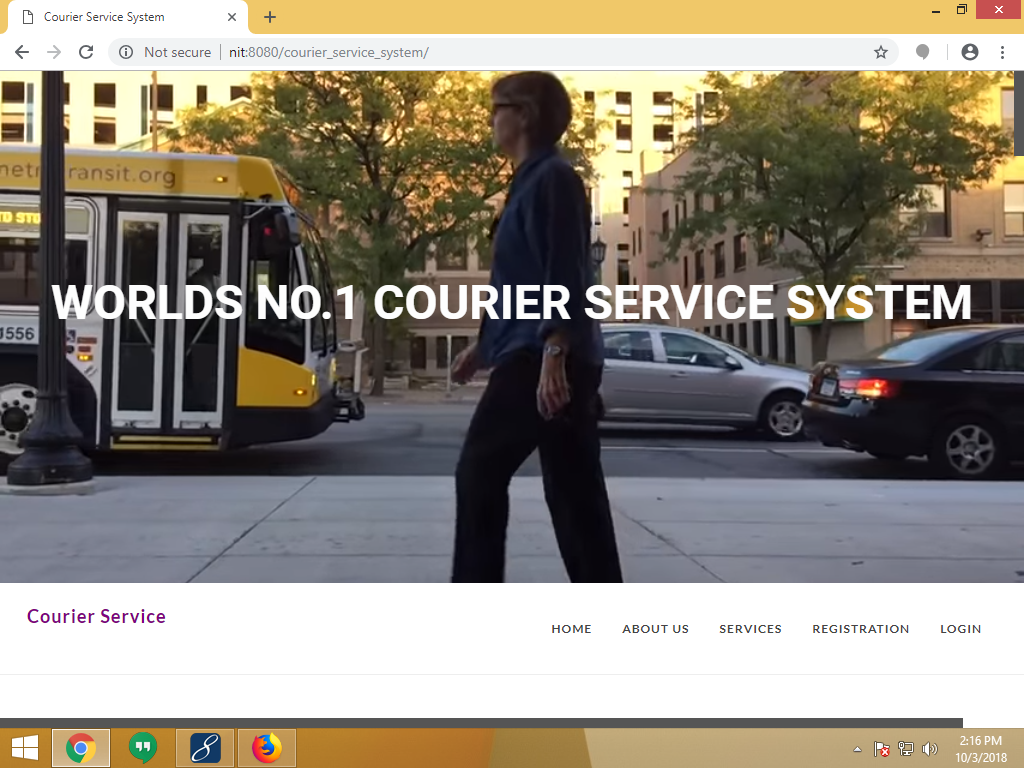
* Must have negative perception.
* Invalid inputs must be used for test.

***Example for Negative Test cases***:

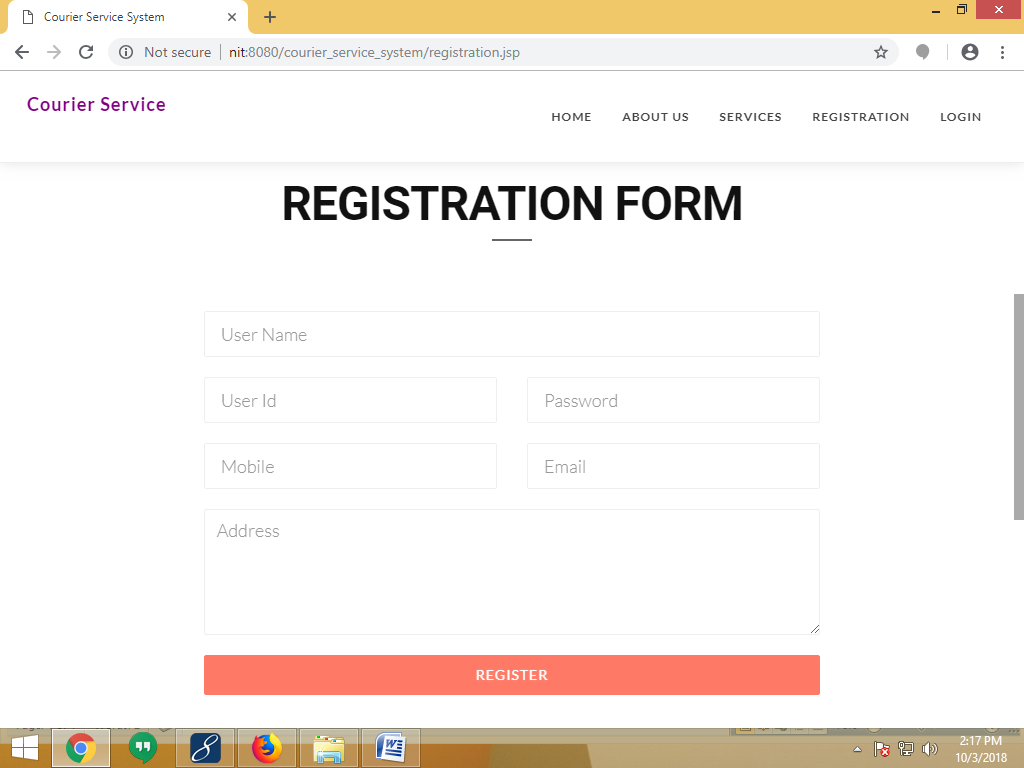
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T.C.No | Description | Expected value | Actual value | Result |
| 1 | Try to modify The information in date and time | Modification should not be allow |  |  |
| 2 | Enter invalid data in to the student details form, click on save | It should not accept invalid data, save should not allow |  |  |

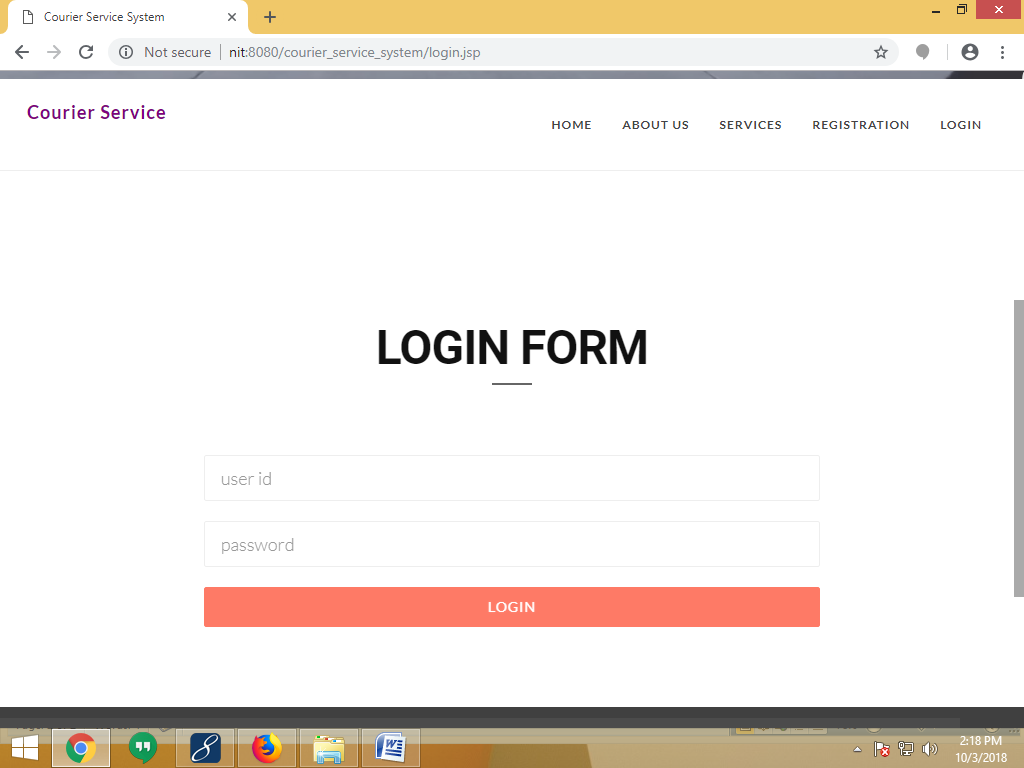
**OUTPUT SCREENS**

Home page:-

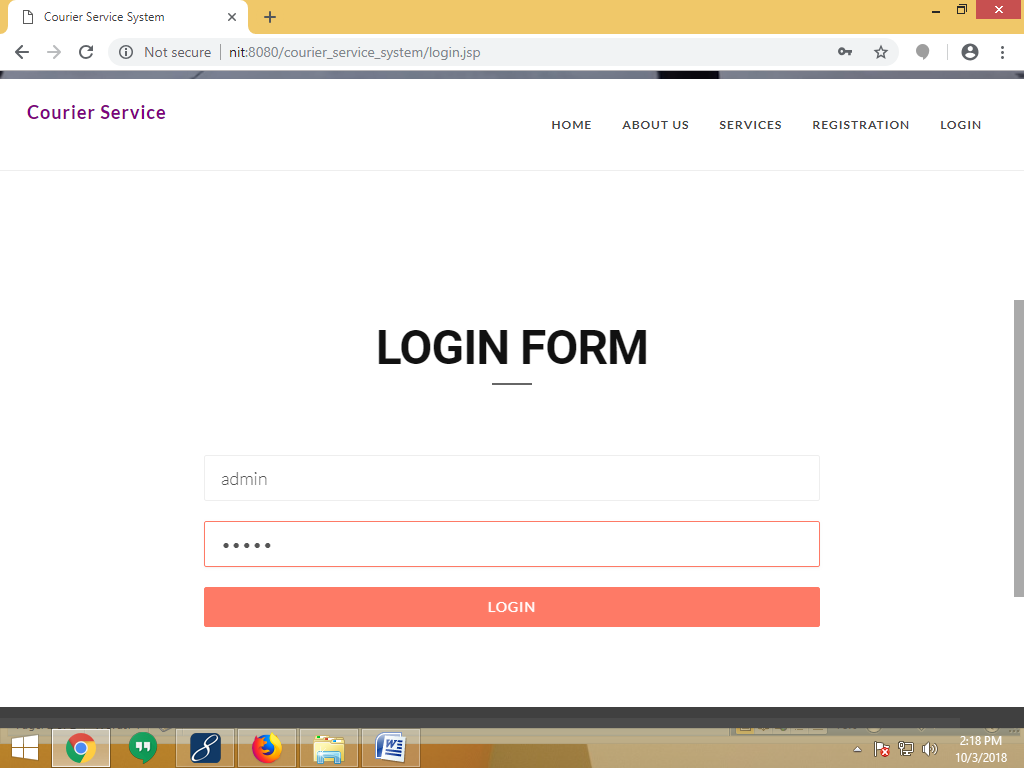


Registration page:-

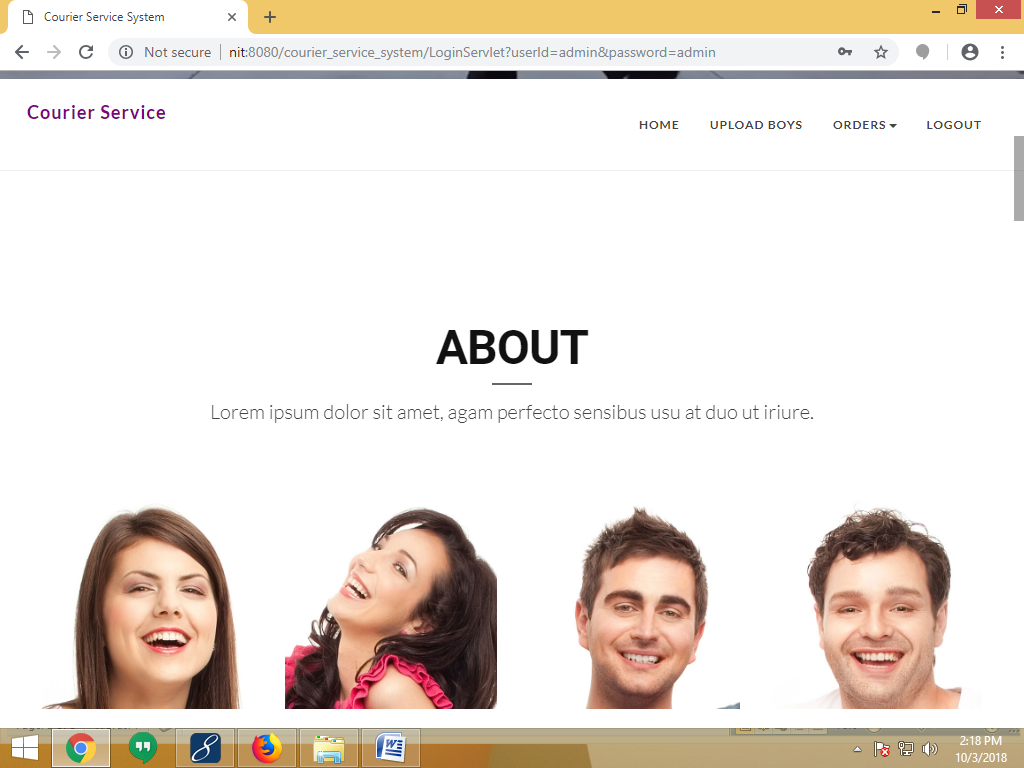




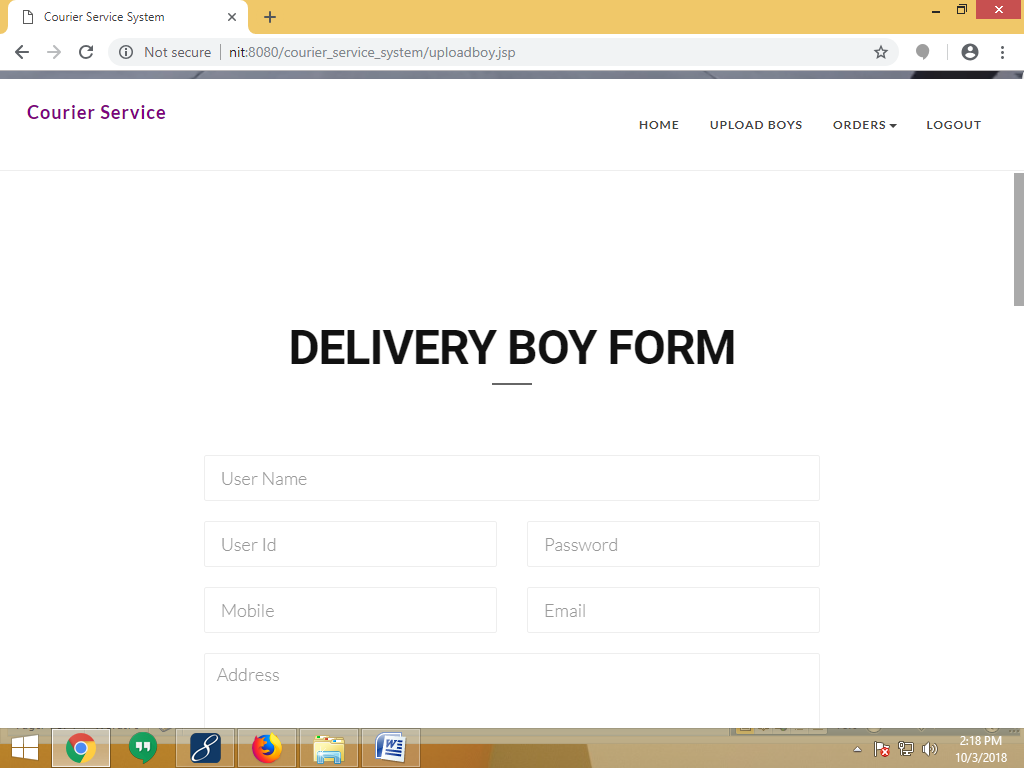
Admin login page



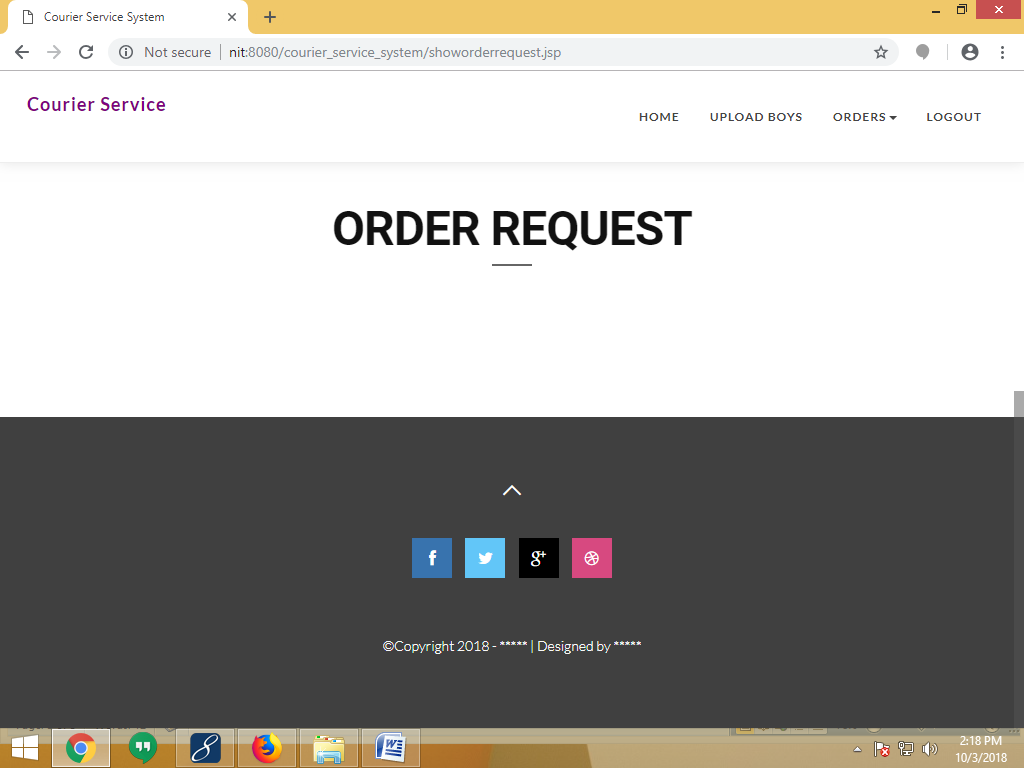
Admin homepage



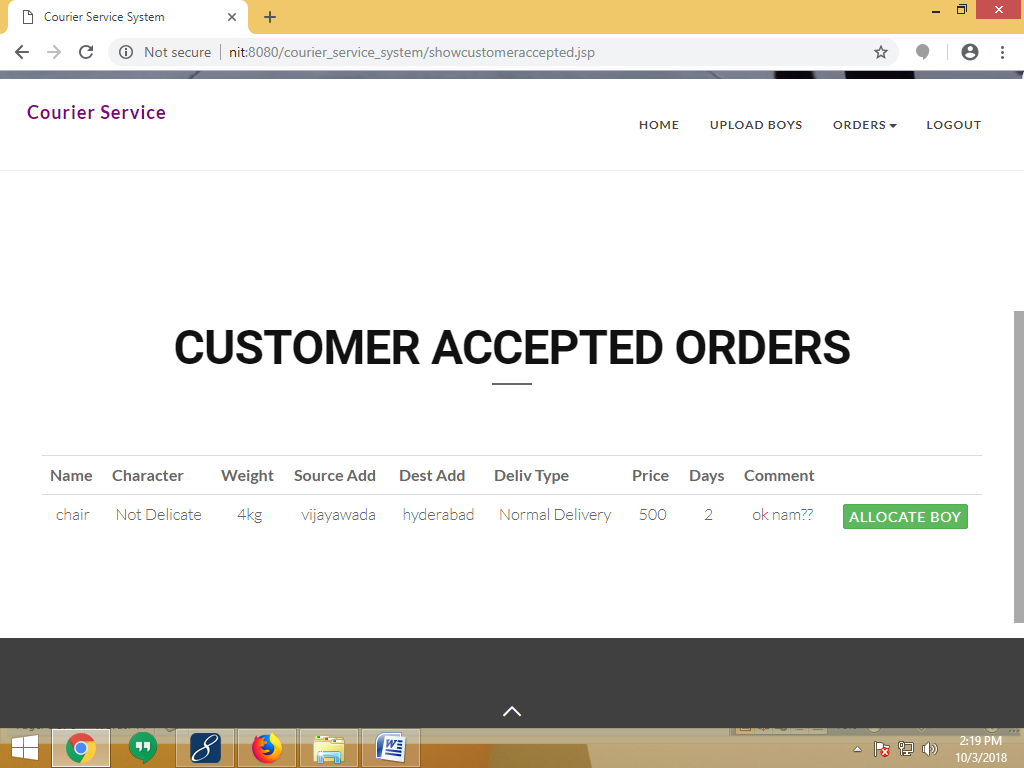
Upload boys info



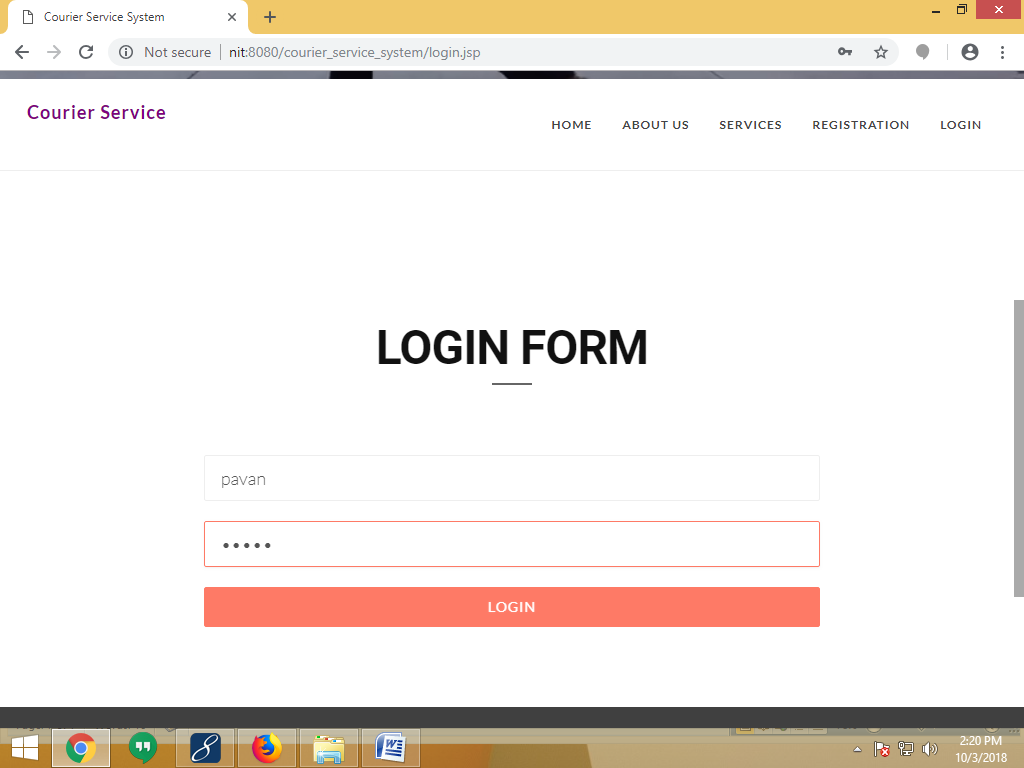
Order requests



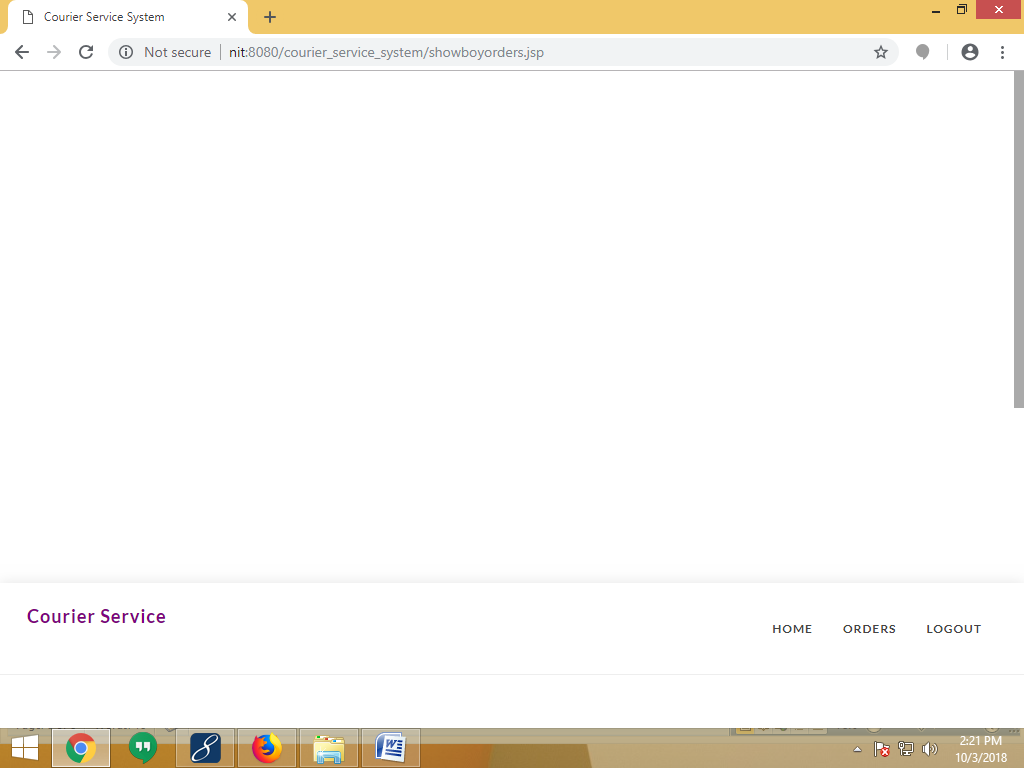
Customer accepted



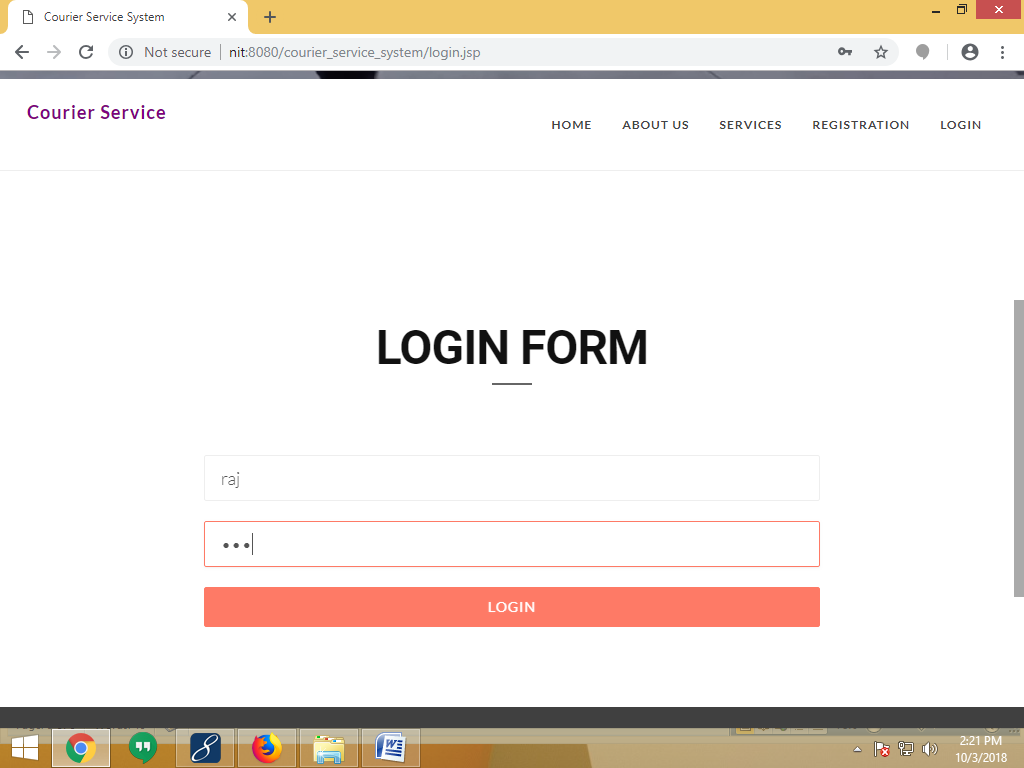
Boy login



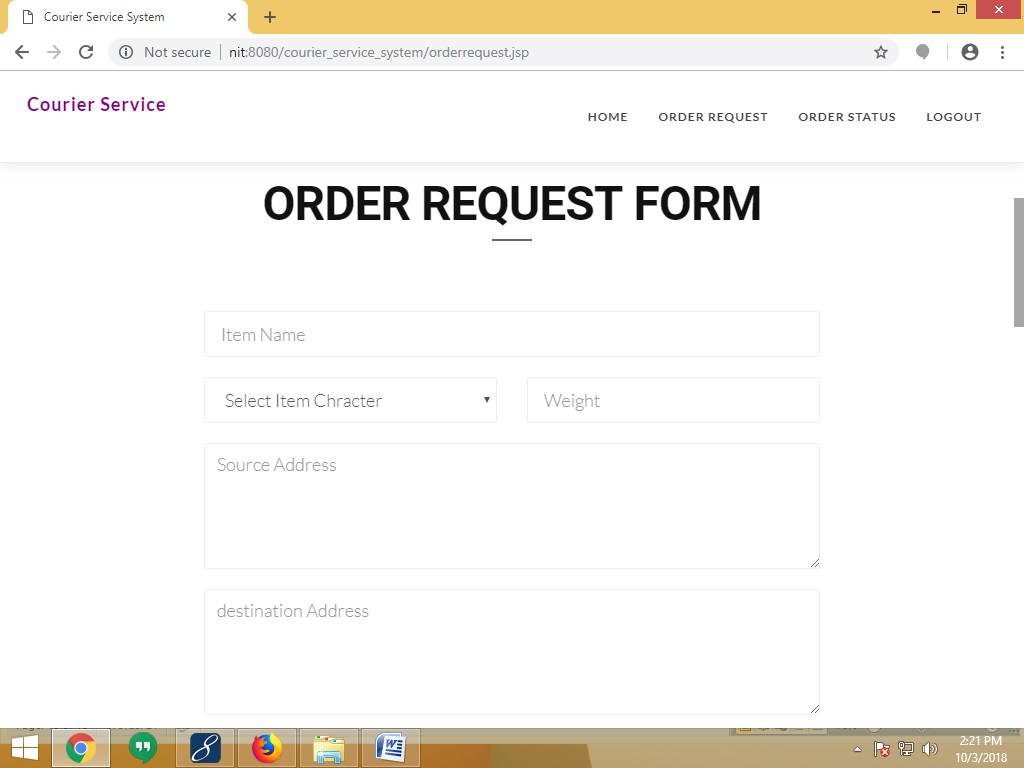
Orders



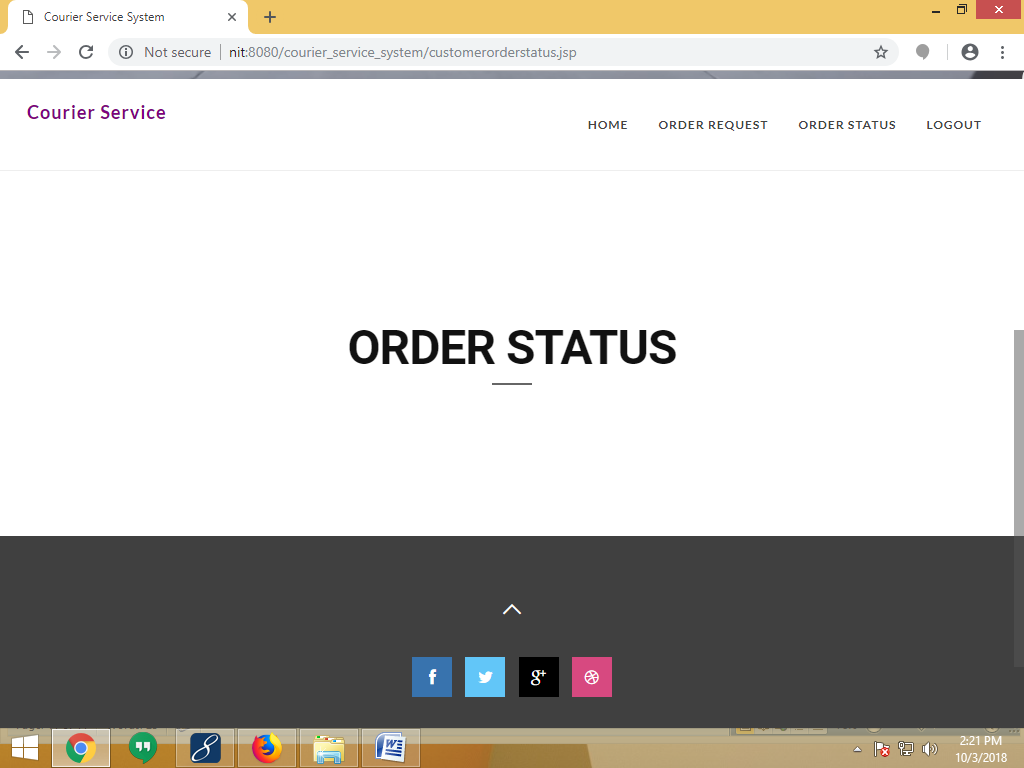
Customer home



Order request



Order status:



**LIMITATIONS AND SCOPE FOR FUTURE ENHANCEMENTS:**

**Limitations of the system:**.

* + System works in all platforms and its compatible environments.
  + Advanced techniques are not used to check the authorization.

**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

**PROJECT SUMMARY**

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.

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

* 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.

## CONCLUSION

**WORK DONE:**

The **COURIER SERVICES SYSTEM** was successfully designed and is tested for accuracy and quality.

During this project we have accomplished all the objectives and this project meets the needs of the organization. The developed will be used in searching, retrieving and generating information for the concerned requests.

**GOALS**

* + Reduced entry work
  + Easy retrieval of information
  + Reduced errors due to human intervention
  + User friendly screens to enter the data
  + Portable and flexible for further enhancement
  + Web enabled.
  + Fast finding of information requested

**BIBILIOGRAPHY**

(1) Java Complete Reference by Herbert Shield

(2) Database Programming with JDBC and Java by George Reese

(3) Java and XML By Brett McLaughlin

(4) Wikipedia, URL: <http://www.wikipedia.org>.

(5) Answers.com, Online Dictionary, Encyclopedia and much more, URL: <http://www.answers.com>

(6) Google, URL: <http://www.google.co.in>

(7)Project Management URL: http://www.startwright.com/project.htm