

# COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE

A Major Project Report Submitted to the Faculty of  
Computer Science and Engineering

**Geethanjali College of Engineering & Technology**

(Affiliated to J.N.T.U.H, Approved by AICTE, NEW DELHI)



In partial fulfillment of the requirements

For the award of degree of

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

By

B.PadmaPriya (10R11A0505)

D.NaveenKanth (10R11A0510)

B.Kirthi (10R11A0526)

KompellaKaushik (10R11A0527)

Under the Esteemed Guidance of

Mrs. M.Ashwini

Associate Professor

Department of Computer Science and Engineering

**Geethanjali College of Engineering & Technology**

Cheeryal (V), Keesara (M), R.R. Dist. Hyderabad-501301, A.P

2013-2014

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**Geethanjali College of Engineering & Technology**

(Affiliated to J.N.T.U.H, Approved by AICTE, NEWDELHI.)



**CERTIFICATE**

This is to certify that the Major Project report entitled “COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE” is a bonafide work done by B.PadmaPriya(10R11A0505), D.NaveenKanth (10R11A0510), B.Kirthi (10R11A0526),KompellaKaushik(10R11A0527), in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in “Computer Science and Engineering” from Jawaharlal Nehru Technological University, Hyderabad during the year 2013-2014.

Internal Guide

Mrs. M. Ashwini

Associate Professor

Dept of C.S.E

H.O.D

Dr. D. RatnaDeepthi

Professor & Head

Dept of C.S.E

External Examiner

## ACKNOWLEDGEMENT

We are greatly indebted to the authorities of Geethanjali College of Engineering and Technology, Cheeryal, R.R Dist, for providing us the necessary facilities to successfully carry out this major project work titled “**COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE**”.

Firstly, we thank and express our sincere gratitude to **Prof. Dr. D.RATNADEEPTHI**, HOD, CSE department, Geethanjali College of Engineering and Technology, for her invaluable help and support which helped us a lot in successfully completing our major project.

Secondly, we express our gratitude to **Associate Prof. Mrs. M.ASHWINI**, Internal guide, Geethanjali College of Engineering and Technology for her suggestions and encouragement which helped us in the successful completion of our major project.

We would like to express our sincere gratitude to our Principal **Dr.S.UDAYA KUMAR** for providing the necessary infrastructure to complete our project.

Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our set goals.

B.PadmaPriya (10R11A0505)

D.NaveenKanth (10R11A0510)

B.Kirthi (10R11A0526)

KompellaKaushik (10R11A0527)

## **ABSTRACT**

Provable data possession (PDP) is a technique for ensuring the integrity of data in storage outsourcing. In this project, we construct an efficient PDP scheme for distributed cloud storage to support the scalability of service and data migration, in which we consider the existence of multiple Cloud Service Providers to cooperatively store and maintain the clients' data. A cooperative PDP (CPDP) scheme is presented which is based on homomorphic verifiable response and hash index hierarchy. The security of CPDP is proved based on multi-prover zero-knowledge proof system, which can satisfy completeness, knowledge soundness, and zero-knowledge properties. In addition, performance optimization mechanisms are articulated for CPDP, and in particular an efficient method for selecting optimal parameter values to minimize the computation costs of clients and storage service providers is presented. Experiments show that CPDP introduces lower computation and communication overheads in comparison with non-cooperative approaches.

## LIST OF FIGURES/DIAGRAMS

<b>S.No.</b>	<b>Figure</b>	<b>Description</b>	<b>Page No.</b>
1	Figure 3.1	Java Program Execution	13
2	Figure 3.2	Program running on Java Platform	14
3	Figure 3.3	Java 2 SDK	16
4	Figure 3.4	JSP Model 2 Architecture	17
5	Figure 3.5	Servlet Architecture	25
6	Figure 3.6	TCP/IP Stack	44
7	Figure 3.7	Tomcat Web Server	58
8	Figure 4.1	System Architecture	60
9	Figure 4.2	Usecase Diagram	64
10	Figure 4.3	Sequence Diagram	65
11	Figure 4.4	Class Diagram	66
12	Figure 4.5	Activity Diagram	67
13	Figure 7.1	Home Page	92
14	Figure 7.2	User Login Page	93
15	Figure 7.3	User Registration Page	94
16	Figure 7.4	TPA Login Page	95
17	Figure 7.5	Admin Login Page	96
18	Figure 7.6	User Home Page	97
19	Figure 7.7	File Upload Page	98
20	Figure 7.8	File Update Page	99
21	Figure 7.9	TPA Home Page	100
22	Figure 7.10	TPA All Files Page	101

23	Figure 7.11	TPA File Alert Page	102
24	Figure 7.12	TPA All File Alerts Page	103
25	Figure 7.13	Admin Home Page	104
26	Figure 7.14	Admin All Files Page	105

## LIST OF TABLES

S.No.	Table	Description	Page No.
1	Table 2.1	Software Requirements	7
2	Table 2.2	Hardware Requirements	7
3	Table 6.1	Test Cases	91

# TABLE OF CONTENTS

S.No.	Contents	Page No.
<b>1</b>	<b>Introduction.....</b>	<b>1</b>
	1.1 About the Project.....	1
	1.2 Objective.....	2
<b>2</b>	<b>System Analysis.....</b>	<b>3</b>
	2.1 Existing System.....	3
	2.2 Proposed System.....	3
	2.3 Scope of the Project.....	4
	2.4 Modules Description.....	5
	2.4.1 Multi Cloud Storage.....	5
	2.4.2 Cooperative PDP.....	5
	2.4.3 Data Integrity.....	5
	2.4.4 Third Party Auditor.....	6
	2.4.5 Cloud User.....	6
	2.5 System Configuration.....	6
	2.5.1 Software Requirements.....	6
	2.5.2 Hardware Requirements.....	7
	2.5.3 Feasibility Study.....	8
	2.5.3.1 Economical Feasibility.....	8
	2.5.3.2 Technical Feasibility.....	9
	2.5.3.3 Social Feasibility.....	9
<b>3</b>	<b>Literature Overview.....</b>	<b>10</b>
	3.1 Technology Used.....	10
	3.1.1 The Java Platform.....	13
	3.2 Java Tools.....	16
	3.2.1 JSP.....	17
	3.2.2 JDBC.....	18



3.2.2.1 Single Tier Architecture.....	20
3.2.2.2 Two Tier Architecture.....	20
3.2.2.3 Three Tier and N-Tier Architecture.....	20
3.2.2.4 JDBC Goals.....	20
3.2.3 J2EE.....	22
3.2.4 Servlets.....	25
3.2.5 HTML.....	26
3.2.6 XML.....	31
3.2.7 Networking.....	40
3.2.7.1 TCP/IP Stack.....	45
3.2.7.2 IP Datagram.....	53
3.2.7.3 UDP.....	54
3.2.7.4 TCP.....	54
3.2.7.5 Internet Addresses.....	55
3.2.7.6 Network Address.....	55
3.2.7.7 Subnet Address.....	56
3.2.7.8 Host Address.....	57
3.2.7.9 Total Address.....	57
3.2.7.10 Port Address.....	57
3.2.7.11 Sockets.....	58
3.2.8 Tomcat 6.0 Web Server.....	58
3.2.9 JFree Chart.....	59
<b>4 System Design.....</b>	<b>61</b>
4.1 System Architecture.....	61
4.2 Input Design.....	62
4.2.1 Objectives of Input Design.....	62
4.3 Output Design.....	62
4.3.1 Objectives of Output Design.....	63
4.4 UML Diagrams.....	63
4.4.1 Usecase Diagram.....	65

4.4.2	Sequence Diagram.....	66
4.4.3	Class Diagram.....	67
4.4.4	Activity Diagram.....	68
4.5	Database Design.....	69
<b>5</b>	<b>Sample Code.....</b>	<b>70</b>
5.1	Coding.....	70
5.1.1	Object Oriented Programming and Java.....	70
5.1.2	The Object Oriented Approach.....	70
5.1.3	Characteristics of Object-Oriented Languages.....	70
5.1.3.1	Objects.....	70
5.1.3.2	Abstraction.....	71
5.1.3.3	Encapsulation.....	72
5.1.3.4	Inheritance.....	72
5.1.3.5	Polymorphism.....	73
5.1.4	Sample Code.....	74
<b>6</b>	<b>Testing.....</b>	<b>81</b>
6.1	Testing.....	81
6.1.1	White-Box Testing.....	82
6.1.2	Black-Box Testing.....	83
6.2	Testing Levels.....	85
6.2.1	Unit Testing.....	85
6.2.2	Integration Testing.....	86
6.2.3	Validation Testing.....	87
6.2.4	System Testing.....	87
6.2.5	Functional Testing.....	88
6.2.6	Performance Testing.....	88
6.2.7	Regression Testing.....	89
6.2.8	Alpha Testing.....	90
6.2.9	Beta Testing.....	90
6.3	Test Cases.....	90

<b>7</b>	<b>Output Screens.....</b>	<b>93</b>
	7.1 Home Page.....	93
	7.2 User Login Page.....	94
	7.3 User Registration Page.....	95
	7.4 TPA Login Page.....	96
	7.5 Admin Login Page.....	97
	7.6 User Home Page.....	98
	7.7 File Upload Page.....	99
	7.8 File Update Page.....	100
	7.9 TPA Home Page.....	101
	7.10 TPA All Files Page.....	102
	7.11 TPA File Alert Page.....	103
	7.12 TPA All File Alerts Page.....	104
	7.13 Admin Home Page.....	105
	7.14 Admin All Files Page.....	106
<b>8</b>	<b>Conclusion.....</b>	<b>107</b>
	8.1 Conclusion.....	107
	8.2 Further Enhancements.....	107
<b>9</b>	<b>Bibliography.....</b>	<b>108</b>
	9.1 Books References.....	108
	9.2 Website References.....	109
	9.3 Technical Publication References.....	110

# 1. INTRODUCTION

## 1.1 ABOUT THE PROJECT

In recent years, cloud storage service has become a faster profit growth point by providing a comparably low-cost, scalable, position-independent platform for clients' data. Since cloud computing environment is constructed based on open architectures and interfaces, it has the capability to incorporate multiple internal and/or external cloud services together to provide high interoperability. Such a distributed cloud environment is called multi-Cloud (or hybrid cloud). Often, by using virtual infrastructure management (VIM), a multi-cloud allows clients to easily access his/her resources remotely through interfaces such as Web services provided by Amazon EC2.

There exist various tools and technologies for multi-cloud, such as Platform VM Orchestrator, VMware vSphere, and Ovirt. These tools help cloud providers construct a distributed cloud storage platform (DCSP) for managing clients' data. However, if such an important platform is vulnerable to security attacks, it would bring irretrievable losses to the clients. For example, the confidential data in an enterprise may be illegally accessed through a remote interface provided by a multi-cloud, or relevant data and archives may be lost or tampered-with when they are stored into an uncertain storage pool outside the enterprise. Therefore, it is indispensable for cloud service providers (CSPs) to provide security techniques for managing their storage services.

Provable data possession (PDP) (or proofs of retrievability (POR)) is such a probabilistic proof technique for a storage provider to prove the integrity and ownership of clients' data without downloading data. The proof-checking without downloading makes it especially important for large-size files and folders (typically including many clients' files) to check whether these data have been tampered with or deleted without downloading the latest version of data. Thus, it is able to replace traditional hash and signature functions in storage outsourcing.

Various PDP schemes have been recently proposed, such as Scalable PDP and Dynamic PDP. However, these schemes mainly focus on PDP issues at untrusted servers in a single cloud storage provider and are not suitable for a multi-cloud environment.

## 1.2 OBJECTIVE

The objective of COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE is to provide security for managing storage services provided by the Cloud Service Providers (CSP). Provable data possession (PDP) is a technique for ensuring the integrity of data in storage outsourcing.

In this project, we construct an efficient PDP scheme for distributed cloud storage to support the scalability of service and data migration, in which we consider the existence of multiple Cloud Service Providers to cooperatively store and maintain the clients' data. A cooperative PDP (CPDP) scheme is presented which is based on homomorphic verifiable response and hash index hierarchy. The security of CPDP is proved based on multi-prover zero-knowledge proof system, which can satisfy completeness, knowledge soundness, and zero-knowledge properties.

In addition, performance optimization mechanisms are articulated for CPDP, and in particular an efficient method for selecting optimal parameter values to minimize the computation costs of clients and storage service providers is presented. Experiments show that CPDP introduces lower computation and communication overheads in comparison with non-cooperative approaches.

## **2. SYSTEM ANALYSIS**

### **2.1 EXISTING SYSTEM**

There exist various tools and technologies for multi-cloud, such as Platform VM Orchestrator, VMwarevSphere, and Ovirt. These tools help Cloud Service Providers construct a distributed cloud storage platform for managing clients' data. However, if such an important platform is vulnerable to security attacks, it would bring irretrievable losses to the clients. For example, the confidential data in an enterprise may be illegally accessed through a remote interface provided by a multi-cloud, or relevant data and archives may be lost or tampered-with when they are stored into an uncertain storage pool outside the enterprise. Therefore, it is indispensable for Cloud Service Providers to provide security techniques for managing their storage services.

### **2.2 PROPOSED SYSTEM**

To check the availability and integrity of outsourced data in cloud storages, researchers have proposed two basic approaches called Provable Data Possession and Proofs of Retrievability. Ateniese et al first proposed the PDP model for ensuring possession of files on untrusted storages and provided an RSA-based scheme for a static case that achieves the communication cost. They also proposed a publicly verifiable version, which allows anyone, not just the owner, to challenge the server for data possession. They proposed a lightweight PDP scheme based on cryptographic hash function and symmetric key encryption, but the servers can deceive the owners by using previous metadata or responses due to the lack of randomness in the challenges. The numbers of updates and challenges are limited and fixed in advance and users cannot perform block insertions anywhere.

In order to prove the integrity of data stored in a multi-cloud environment, a framework for CPDP based on interactive proof system (IPS) and multi prove zero-knowledge proof system (MPZKPS) is defined.

To support distributed cloud storage, a representative architecture is illustrated in CPDP. The architecture has a hierarchy structure which resembles a natural representation of file storage. This hierarchical structure consists of three layers to represent relationships among all blocks for stored resources. They are described as follows:

- a) **Express Layer:** Offers an abstract representation of the stored resources;
- b) **Service Layer:** Offers and manages cloud storage services; and
- c) **Storage Layer:** Realizes data storage on many physical devices.

This simple hierarchy is used to organize data blocks from multiple CSP services into a large size file by shading their differences among these cloud storage systems. For example the resources in Express Layer are split and stored into three CSP's that are indicated by different colors in Service Layer. These colors are used to distinguish between CSP's. Moreover, logically the data blocks are organized in the Storage Layer. This architecture also provides special functions for data storage and management. For example, there may exist overlaps among data blocks and discontinuous blocks but these functions may increase the complexity of storage management.

## 2.3 SCOPE OF THE PROJECT

This project is basically aimed at providing security for managing storage services provided by the Cloud Service Providers (CSP). An efficient PDP scheme for distributed cloud storage to support the scalability of service and data migration, in which we consider the existence of multiple Cloud Service Providers to cooperatively store and maintain the clients' data is constructed. A cooperative PDP (CPDP) scheme is presented which is based on homomorphic verifiable response and hash index hierarchy. The security of CPDP is proved based on multi-prover zero-knowledge proof system, which can satisfy completeness, knowledge soundness, and zero-knowledge properties.

## **2.4 MODULES DESCRIPTION**

This project has the following modules:

- a) Multi-cloud Storage
- b) Cooperative PDP
- c) Data Integrity
- d) Third Party Auditor
- e) Cloud User

### **2.4.1 MULTI CLOUD STORAGE**

Distributed computing is used to refer to any large collaboration in which many individual personal computer owners allow some of their computer's processing time to be put at the service of a large problem. In this project, each cloud admin consists of data blocks. The cloud user uploads the data into multi cloud. Cloud computing environment is constructed based on open architectures and interfaces, it has the capability to incorporate multiple internal and/or external cloud services together to provide high interoperability. Such a distributed cloud environment is called multi-Cloud. A multi-cloud allows clients to easily access his/her resources remotely through interfaces.

### **2.4.2 COOPERATIVE PDP**

Cooperative PDP (CPDP) schemes adopt zero-knowledge property and three-layered index hierarchy. In particular, efficient method for selecting the optimal number of sectors in each block to minimize the computation costs of clients and storage service providers is embedded. Cooperative PDP (CPDP) scheme without compromising data privacy based on modern cryptographic techniques is also included.

### **2.4.3 DATA INTEGRITY**

Data Integrity is very important in database operations in particular, Data warehousing and Business intelligence in general. Data Integrity ensures that data is of high quality, correct, consistent and accessible.



#### **2.4.4 THIRD PARTY AUDITOR**

Trusted Third Party (TTP) is the one who is trusted to store verification parameters and offer public query services for these parameters. In this project, the Trusted Third Party can view the user data blocks and upload to the distributed cloud. In distributed cloud environment each cloud has user data blocks. If any modification is tried by cloud owner, an alert is sent to the Trusted Third Party.

#### **2.4.5 CLOUD USER**

The Cloud User is the one who has a large amount of data to be stored in multiple clouds and has the permissions to access and manipulate stored data. The user's data is converted into data blocks. The data blocks are uploaded to the cloud. The TPA views the data blocks and uploads in multi cloud. The user can update the uploaded data. If the user wants to download his/her files, the data in multi cloud is integrated and downloaded.

### **2.5 SYSTEM CONFIGURATION**

The system configuration defines how the system should be configured in order to run the project being developed. The system should be properly configured in order to avoid nasty resource conflict problems and other strange errors. System Configuration includes the software requirements of the project, the hardware requirements of the project and the feasibility study. The feasibility study shows how well the project can work under limited resources and limited time.

#### **2.5.1 SOFTWARE REQUIREMENTS**

The Software requirements enlist all the requirements and the environment that are necessary for the project development. The software requirements primarily include the front end and the back end of the project. The front end application is the one with which the users interact directly. The back end application serves indirectly in support of the front end services by communicating with the required resource. The intermediate program mediates front end and back end activities.

Below is a list of Software Requirements for the development of “COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE” system.

Operating System	Windows 95/98/2000/XP onwards
Application Server	Tomcat 5.0/6.X
Front End	HTML, Java, JSP
Scripts	Javascript
Server Side Script	Java Server Pages
Database	Mysql
Database Connectivity	JDBC

Table 2.1: Software requirements

## 2.5.2 HARDWARE REQUIREMENTS

The hardware requirements represent all the usage requirements for the project. This shows the minimum and recommended requirements in the hardware of the system to run the project efficiently.

Below is a list of Hardware Requirements for the development of “COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE” system.

Processor	Pentium-III
Speed	1.1 Ghz
RAM	256 MB(min)
Hard Disk	20GB
Monitor	SVGA
Keyboard	Standard Windows Keyboard

Table 2.2 Hardware requirements

### **2.5.3 FEASIBILITY STUDY**

A feasibility study evaluates the project's potential for success. All projects are feasible under unlimited resources and infinite time. But the development of software is plagued by the scarcity of resources and difficult delivery rates. It is both necessary and prudent to evaluate the feasibility of a project at the earliest possible time.

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential. Three key considerations involved in the feasibility analysis are:

- a) ECONOMICAL FEASIBILITY
- b) TECHNICAL FEASIBILITY
- c) SOCIAL FEASIBILITY

#### **2.5.3.1 ECONOMICAL FEASIBILITY**

This procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs. If benefits outweigh costs, then the decision is made to design and implement the system. Otherwise, further justification or alterations in proposed system will have to be made if it is to have a chance of being approved. This is an ongoing effort that improves in accuracy at each phase of the system life cycle.

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system remains within the budget because most of the technologies used are freely available. Only the customized products will have to be purchased.

### **2.5.3.2 TECHNICAL FEASIBILITY**

A large part of determining resources has to do with assessing technical feasibility. It considers the technical requirements of the proposed project. The technical requirements are then compared to the technical capability. The project is considered technically feasible if the internal technical capability is sufficient to support the project requirements. Technical feasibility centers on the existing computer system (hardware, software, etc.,) and to what extent it can support the proposed addition. If the budget is a serious constraint, then the project is judged not feasible.

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

### **2.5.3.3 SOCIAL FEASIBILITY**

Social feasibility is dependent on human resources available for the project and involves projecting whether the system will be used if it is developed and implemented. Social feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

### **3. LITERATURE OVERVIEW**

Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy and company strength. Once these things are satisfied next steps are to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool, the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system the above consideration are taken into account for developing the proposed system.

#### **3.1 TECHNOLOGY USED**

The technology used in this system is **JAVA**.

Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at SUN Microsystems Incorporation in the year 1991. It took 18 months to develop the 1st working version. This language was initially called “OAK”, but was renamed “JAVA” in 1995, many more contributed to the design and evolution of the language.

Java is a powerful but lean object-oriented programming language. It has generated a lot of excitement because it makes it possible to program for Internet by creating Applets, programs that can be embedded in web page. The context of an applet can be an animation with sound, an interactive game or a ticker tape. Applets can be just little decorations to liven up web page, or they can be serious applications like Word processor or Spreadsheet.

Java is being used more for writing standalone applications as well. It is becoming so popular that many people believe it will become standard language for both general purpose and Internet programming.

Java builds on the strength of C++. It has taken the best features of C++ and discarded the more problematic and error prone parts. To this lean core, it has added garbage collection (automatic memory management), multithreading (the capacity for one program

to do more than one thing at a time), security capabilities. This result is that Java is simple, elegant, and powerful and easy-to-use. This system uses various JAVA tools.

Programs written in Java have a reputation for being slower and requiring more memory than those written in C++. However, Java programs' execution speed improved significantly with the introduction of Just-in-time compilation in 1997/1998 for Java 1.1, the addition of language features supporting better code analysis (such as inner classes, the `StringBuilder` class, optional assertions, etc.), and optimizations in the Java virtual machine itself, such as HotSpot becoming the default for Sun's JVM in 2000.

Some platforms offer direct hardware support for Java; there are microcontrollers that can run Java in hardware instead of a software Java virtual machine, and ARM based processors can have hardware support for executing Java bytecode through their Jazelle option.

Java uses an automatic garbage collector to manage memory in the object lifecycle. The programmer determines when objects are created, and the Java runtime is responsible for recovering the memory once objects are no longer in use. Once no references to an object remain, the unreachable memory becomes eligible to be freed automatically by the garbage collector. Something similar to a memory leak may still occur if a programmer's code holds a reference to an object that is no longer needed, typically when objects that are no longer needed are stored in containers that are still in use. If methods for a nonexistent object are called, a "null pointer exception" is thrown.

One of the ideas behind Java's automatic memory management model is that programmers can be spared the burden of having to perform manual memory management. In some languages, memory for the creation of objects is implicitly allocated on the stack, or explicitly allocated and de-allocated from the heap. In the latter case the responsibility of managing memory resides with the programmer. If the program does not de-allocate an object, a memory leak occurs. If the program attempts to access or de-allocate memory that has already been de-allocated, the result is undefined and difficult to predict, and the program is likely to become unstable and/or crash. This can be partially remedied by the

use of smart pointers, but these add overhead and complexity. Note that garbage collection does not prevent "logical" memory leaks, i.e. those where the memory is still referenced but never used.

Garbage collection may happen at any time. Ideally, it will occur when a program is idle. It is guaranteed to be triggered if there is insufficient free memory on the heap to allocate a new object; this can cause a program to stall momentarily. Explicit memory management is not possible in Java.

Java does not support C/C++ style pointer arithmetic, where object addresses and unsigned integers (usually long integers) can be used interchangeably. This allows the garbage collector to relocate referenced objects and ensures type safety and security.

As in C++ and some other object-oriented languages, variables of Java's primitive data types are not objects. Values of primitive types are either stored directly in fields (for objects) or on the stack (for methods) rather than on the heap, as is commonly true for objects (but see escape analysis). This was a conscious decision by Java's designers for performance reasons. Because of this, Java was not considered to be a pure object-oriented programming language. However, as of Java 5.0, autoboxing enables programmers to proceed as if primitive types were instances of their wrapper class.

Java contains multiple types of garbage collectors. By default, HotSpot uses the concurrent mark sweep collector, also known as the CMS garbage collector or CMS. However, there are also several other garbage collectors that can be used to manage the heap. For 90% of applications in Java, the CMS garbage collector is sufficient. Oracle aims to replace CMS with the Garbage-first collector (G1).

With most programming languages, we either compile or interpret a program so that we can run it on our computer. Java programming language is unusual, in this a program is both compiled and interpreted. With the compiler, first we translate a program into an intermediate language called Java byte code —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code

instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.

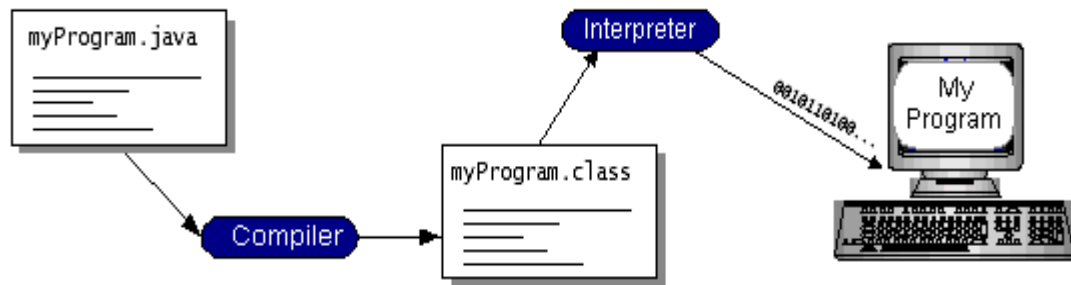


Figure 3.1: Java Program Execution

We can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. We can compile our program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.

### 3.1.1 THE JAVA PLATFORM

A platform is the hardware or software environment in which a program runs. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms, in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- a) The Java Virtual Machine (Java VM)



b) The Java Application Programming Interface (Java API)

Java VM is the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.

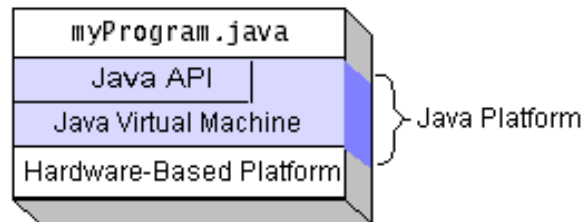


Figure 3.2: Program running on Java Platform

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

The most common types of programs written in the Java programming language are applets and applications. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, we can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a servlet. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

The API supports all these kind of programs with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- a) **The essentials:** Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- b) **Applets:** The set of conventions used by applets.
- c) **Networking:** URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, and IP (Internet Protocol) addresses.
- d) **Internationalization:** Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- e) **Security:** Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- f) **Software components:** Known as JavaBeans™, can plug into existing component architectures.

- g) **Object serialization:** Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- h) **Java Database Connectivity (JDBC™):** Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.

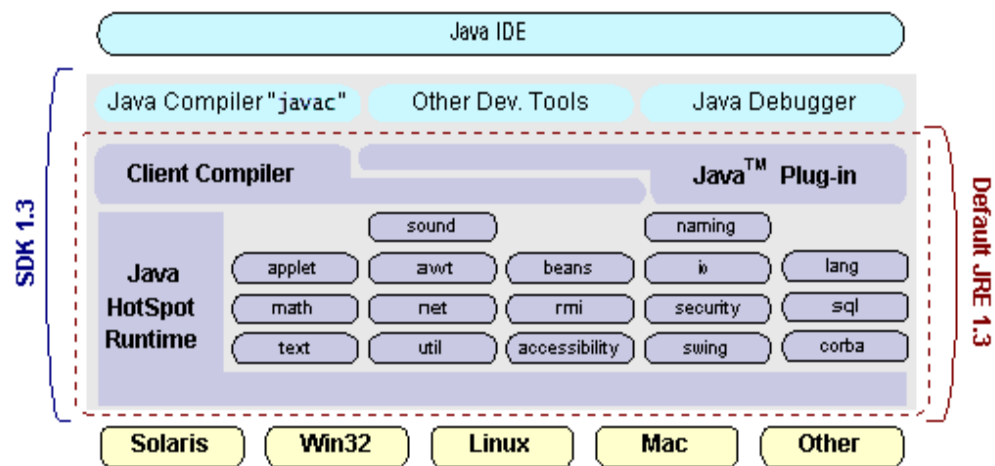


Figure 3.3: Java 2 SDK

The JDK forms an extended subset of a software development kit (SDK). It includes "tools for developing, debugging, and monitoring Java applications". Oracle strongly suggests that they now use the term "JDK" to refer to the Java SE Development Kit. The Java EE SDK is available with or without the "JDK", by which they specifically mean the Java SE 7 JDK.

## 3.2 JAVA TOOLS

Given below are the various Java Tools used for the development of the "COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE" system.

### 3.2.1 JSP

Java Server Pages or JSP for short is Sun's solution for developing dynamic web sites. JSP provides excellent server side scripting support for creating database driven web applications. JSP enables the developers to directly insert java code into jsp file, this makes the development process very simple and its maintenance also becomes very easy. JSP pages are efficient, it loads into the web server's memory on receiving the request very first time and the subsequent calls are served within a very short period of time. JavaServer Pages are text files that combine standard HTML and new scripting tags. JSPs look like HTML, but they get compiled into Java servlets the first time they are invoked. The resulting servlet is a combination of HTML from the JSP file and embedded dynamic content specified by the new tags.

Architecturally, JSP may be viewed as a high-level abstraction of Java servlets. JSPs are translated into servlets at runtime; each JSP's servlet is cached and re-used until the original JSP is modified.

JSP can be used independently or as the view component of a server-side model–view–controller design, normally with JavaBeans as the model and Java servlets (or a framework such as Apache Struts) as the controller. This is a type of Model 2 architecture.

JSP allows Java code and certain pre-defined actions to be interleaved with static web markup content, with the resulting page being compiled and executed on the server to deliver a document. The compiled pages, as well as any dependent Java libraries, use Java bytecode rather than a native software format. Like any other Java program, they must be executed within a Java virtual machine (JVM) that integrates with the server's host operating system to provide an abstract platform-neutral environment.

JSPs are usually used to deliver HTML and XML documents, but through the use of OutputStream, they can deliver other types of data as well. The Web container creates JSP implicit objects like pageContext, servletContext, session, request & response.

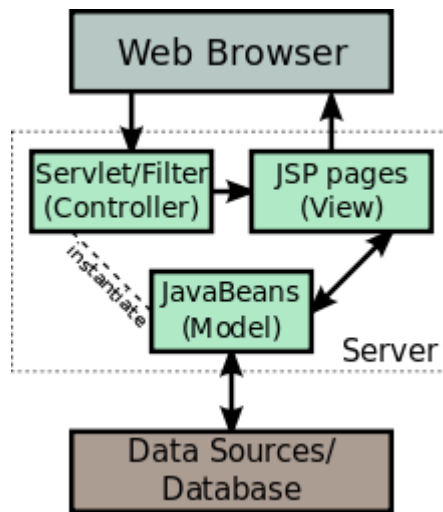


Figure 3.4: JSP Model 2 Architecture

### 3.2.2 JDBC

JDBC (Java Database connectivity) is a front-end tool for connecting to a server to ODBC in that respect, however JDBC can connect only java client and it uses ODBC for the connectivity.

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for application developers and database systems providers. Before ODBC became a de facto standard for Windows programs to interface with database systems, programmers had to use proprietary languages for each database they wanted to connect to. Now, ODBC has made the choice of the database system almost irrelevant from a coding perspective, which is as it should be. Application developers have much more important things to worry about than the syntax that is needed to port their program from one database to another when business needs suddenly change.

Through the ODBC Administrator in Control Panel, you can specify the particular database that is associated with a data source that an ODBC application program is written to use. Think of an ODBC data source as a door with a name on it. Each door will lead you to a particular database. For example, the data source named Sales Figures might be a SQL Server database, whereas the Accounts Payable data source could refer to an Access

database. The physical database referred to by a data source can reside anywhere on the LAN.

The ODBC system files are not installed on your system by Windows 95. Rather, they are installed when you setup a separate database application, such as SQL Server Client or Visual Basic 4.0. When the ODBC icon is installed in Control Panel, it uses a file called ODBCINST.DLL. It is also possible to administer your ODBC data sources through a stand-alone program called ODBCADM.EXE. There is a 16-bit and a 32-bit version of this program and each maintains a separate list of ODBC data sources.

From a programming perspective, the beauty of ODBC is that the application can be written to use the same set of function calls to interface with any data source, regardless of the database vendor. The source code of the application doesn't change whether it talks to Oracle or SQL Server. We only mention these two as an example. There are ODBC drivers available for several dozen popular database systems. Even Excel spreadsheets and plain text files can be turned into data sources. The operating system uses the Registry information written by ODBC Administrator to determine which low-level ODBC drivers are needed to talk to the data source (such as the interface to Oracle or SQL Server). The loading of the ODBC drivers is transparent to the ODBC application program. In a client/server environment, the ODBC API even handles many of the network issues for the application programmer.

The advantages of this scheme are so numerous that you are probably thinking there must be some catch. The only disadvantage of ODBC is that it isn't as efficient as talking directly to the native database interface. ODBC has had many detractors make the charge that it is too slow. Microsoft has always claimed that the critical factor in performance is the quality of the driver software that is used. In our humble opinion, this is true. The availability of good ODBC drivers has improved a great deal recently. And anyway, the criticism about performance is somewhat analogous to those who said that compilers would never match the speed of pure assembly language. Maybe not, but the compiler (or ODBC) gives you the opportunity to write cleaner programs, which means you finish sooner. Meanwhile, computers get faster every year.

JDBC is essentially a low-level API since any data manipulation, storage and retrieval has to be done by the program itself. Based on number of intermediate server through the request should go it is named as single tier, two tier and multi tier architecture

#### **3.2.2.1 SINGLE TIER ARCHITECTURE**

In a single tier the server and client are the same, in the sense that a client program that needs information (client) and the source of this type of architecture is also possible in java, in case flat files are used to store the data. However this is useful only in case of small applications. The advantage with this is the simplicity and portability of the application developed.

#### **3.2.2.2 TWO TIER ARCHITECTURE**

In two tier architecture the database resides in one machine the network. In this type of architecture a database management takes control of the database and provides access to clients in a network. This software bundle is also called as the server. Software in different machines, requesting for information are called as the clients.

#### **3.2.2.3 THREE TIER AND N-TIER ARCHITECTURE**

In the three-tier architecture, any number servers can access the database that resides on server, which in turn serve clients in a network. The intermediate server acts as a two-way communication channel also. This is the information or data from the database is passed on to the applet that is requesting it. This can be extended to make n tiers of servers, each server carrying to specific type of request from clients; however in practice only 3 tiers architecture is popular.

The four types of JDBC Drivers are, jdbc-odbc Bridge plus odbc driver, native api partly-java driver, jdbc-net all-java driver, and native protocol all-java driver.

#### **3.2.2.4 JDBC GOALS**

Few software packages are designed with goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

- a) **SQL Level API:**The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to “generate” JDBC code and to hide many of JDBC’s complexities from the end user.
- b) **SQL Conformance:**SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.
- c) **JDBC must be implemental on top of common database interfaces:**The JDBC SQL API must “sit” on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.
- d) **Provide a Java interface that is consistent with the rest of the Java system:**Because of Java’s acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.
- e) **Keep it simple:**This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.
- f) **Use strong, static typing wherever possible:**Strong typing allows for more error checking to be done at compile time; also, less error appear at runtime.
- g) **Keep the common cases simple:**Because more often than not, the usual SQL calls used by the programmer are simple SELECT’s, INSERT’s, DELETE’s and



UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

### 3.2.3 J2EE

Java Platform, Enterprise Edition or Java EE is Oracle's enterprise Java computing platform. Java EE extends the Java Platform, Standard Edition (Java SE), providing an API for object-relational mapping, distributed and multi-tier architectures, and web services. Software for Java EE is primarily developed in the Java programming language. The platform emphasizes Convention over configuration and annotations for configuration. Optionally XML can be used to override annotations or to deviate from the platform defaults.

The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications. The platform incorporates a design based largely on modular components running on an application server.

Java EE is defined by its specification. As with other Java Community Process specifications, providers must meet certain conformance requirements in order to declare their products as Java EE compliant. Java EE includes several API specifications, such as JDBC, RMI, e-mail, JMS, web services, XML, etc., and defines how to coordinate them. Java EE also features some specifications unique to Java EE for components. These include Enterprise JavaBeans, Connectors, servlets, JavaServer Pages and several web service technologies. This allows developers to create enterprise applications that are portable and scalable, and that integrate with legacy technologies. A Java EE application server can handle transactions, security, scalability, concurrency and management of the components it is deploying, in order to enable developers to concentrate more on the business logic of the components rather than on infrastructure and integration tasks.

The Java EE APIs includes several technologies that extend the functionality of the base Java SE APIs.

- a) **javax.servlet.\***: The servlet specification defines a set of APIs to service mainly HTTP requests. It includes the JavaServer Pages (JSP) specification.

- b) **javax.websocket.\***:The Java API for WebSocket specification defines a set of APIs to service WebSocket connections.
- c) **javax.faces.\***:This package defines the root of the JavaServer Faces (JSF) API. JSF is a technology for constructing user interfaces out of components.
- d) **javax.faces.component.\***:This package defines the component part of the JavaServer Faces (JSF) API. Since JSF is primarily component oriented, this is one of the core packages. The package overview contains a UML diagram of the component hierarchy.
- e) **javax.el.\***:This package defines the classes and interfaces for Java EE's Expression Language. The Expression Language (EL) is a simple language originally designed to satisfy the specific needs of web application developers. It's used specifically in JSF to bind components to (backing) beans and in CDI to name beans, but can be used throughout the entire platform.
- f) **javax.enterprise.inject.\***:These packages define the injection annotations for the contexts and Dependency Injection (CDI) APIs.
- g) **javax.enterprise.context.\***:These packages define the context annotations and interfaces for the Contexts and Dependency Injection (CDI) API.
- h) **javax.ejb.\***: The Enterprise JavaBean (EJB) specification defines a set of lightweight APIs that an object container (the EJB container) will support in order to provide transactions (using JTA), remote procedure calls (using RMI or RMI-IIOP), concurrency control, dependency injection and access control for business objects. This package contains the Enterprise JavaBeans classes and interfaces that define the contracts between the enterprise bean and its clients and between the enterprise bean and the EJB container.
- i) **javax.validation.\***: This package contains the annotations and interfaces for the declarative validation support offered by the Bean Validation API. Bean Validation provides a unified way to provide constraints on beans (e.g. JPA model classes) that can be enforced cross-layer. In Java EE, JPA honors bean validation constraints in the persistence layer, while JSF does so in the view layer.

- j) **javax.persistence.\***: This package contains the classes and interfaces that define the contracts between a persistence provider and the managed classes and the clients of the Java Persistence API (JPA).
- k) **javax.transaction.\***: This package provides the Java Transaction API (JTA) that contains the interfaces and annotations to interact with the transaction support offered by Java EE. Even though this API abstracts from the really low-level details, the interfaces are also considered somewhat low-level and the average application developer in Java EE is either assumed to be relying on transparent handling of transactions by the higher level EJB abstractions, or using the annotations provided by this API in combination with CDI managed beans.
- l) **javax.security.auth.message.\***: This package provides the core of the Java Authentication SPI (JASPIC) that contains the interfaces and classes to build authentication modules for secure Java EE applications. Authentication modules are responsible for the interaction dialog with a user (e.g. redirecting to a Form or to an OpenID provider), verifying the user's input (e.g. by doing an LDAP lookup, database query or contacting the OpenID provider with a token) and retrieving a set of groups/roles that the authenticated user is in or has (e.g. by again doing an LDAP lookup or database query).
- m) **javax.enterprise.concurrent.\***: This package provides the interfaces for interacting directly with Java EE's platform default managed thread pool. A higher-level executor service working on this same thread pool can be used optionally. The same interfaces can be used for user-defined managed thread pools, but this relies on vendor specific configuration and is not covered by the Java EE specification.
- n) **javax.jms.\***: This package defines the Java Message Service (JMS) API. The JMS API provides a common way for Java programs to create, send, receive and read an enterprise messaging system's messages.
- o) **javax.batch.api.\***: This package defines the entry AP for Java EE Batch Applications. The Batch Applications API provides the means to run long running background tasks that possibly involve a large volume of data and which may need to be periodically executed.

- p) **javax.resource.\***: This package defines the Java EE Connector Architecture (JCA) API. Java EE Connector Architecture (JCA) is a Java-based technology solution for connecting application servers and enterprise information systems (EIS) as part of enterprise application integration (EAI) solutions. This is a low-level API aimed at vendors that the average application developer typically does not come in contact with.

### 3.2.4 SERVLETS

Java Servlets are programs that run on a Web or Application server and act as a middle layer between requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server. Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically. Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI. Performance is significantly better. Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.

Servlets are platform-independent because they are written in Java. Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted. The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

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 which can be plugged into a Java-based server. Servlets are to the server-side where as applets are 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, plug gable helper byte code objects on the server side that can be used to dynamically extend server-side functionality.

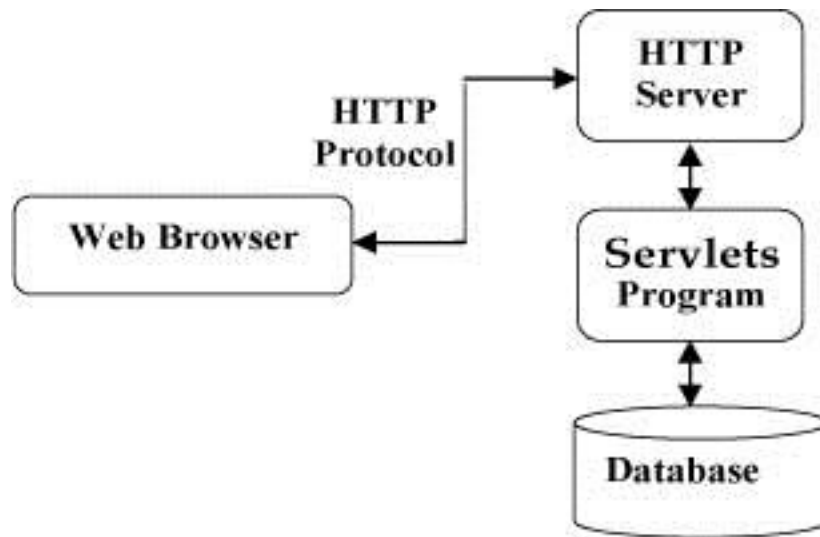


Figure 3.5: Servlet Architecture

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification. Servlets can be created using the `javax.servlet` and `javax.servlet.http` packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects. These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

### 3.2.5 HTML

The hypertext markup language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML (Standard generalized markup language) documents with generic semantics that are appropriate for representing information from a wide range of applications. An HTML document consists of text, which comprises the content of the document and tags, which, defines the structure, and appearance of the document. Each document has a head and body delimited by the `<HEAD>` and `<BODY>` tag. The head is where we give our HTML

document a title and where we indicate other parameters the browser may use when displaying the document.

HTML is written in the form of HTML elements consisting of tags enclosed in angle brackets (like `<html>`), within the web page content. HTML tags most commonly come in pairs like `<h1>` and `</h1>`, although some tags represent empty elements and so are unpaired, for example `<img>`. The first tag in a pair is the start tag, and the second tag is the end tag (they are also called opening tags and closing tags). In between these tags web designers can add text, further tags, comments and other types of text-based content.

The purpose of a web browser is to read HTML documents and compose them into visible or audible web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page.

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts written in languages such as JavaScript which affect the behavior of HTML web pages.

Web browsers can also refer to Cascading Style Sheets (CSS) to define the look and layout of text and other material. The W3C, maintainer of both the HTML and the CSS standards, encourages the use of CSS over explicit presentational HTML.

HTML documents imply a structure of nested HTML elements. These are indicated in the document by HTML tags, enclosed in angle brackets thus: `<p>`

In the simple, general case, the extent of an element is indicated by a pair of tags: a 'start tag' `<p>` and 'end tag'. The text content of the element, if any, is placed between these tags. Tags may also enclose further tag markup between the start and end, including a mixture of tags and text. This indicates further, nested, elements, as children of the parent element. The start tag may also include attributes within the tag. These indicate other information, such as identifiers for sections within the document, identifiers used to bind style information to the presentation of the document, and for some tags such as the `<img>` used to embed images, the reference to the image resource.

Some elements, such as the line break `<br>`, do not permit any embedded content, either text or further tags. These require only a single empty tag (akin to a start tag) and do not use an end tag.

Many tags, particularly the closing end tag for the very commonly-used paragraph element `<p>`, are optional. An HTML browser or other agent can infer the closure for the end of an element from the context and the structural rules defined by the HTML standard. These rules are complex and not widely understood by most HTML coders.

The general form of an HTML element is therefore: `<tag attribute1="value1" attribute2="value2">content</tag>`. Some HTML elements are defined as empty elements and take the form `<tag attribute1="value1" attribute2="value2" >`. Empty elements may enclose no content, for instance, the BR tag or the inline IMG tag. The name of an HTML element is the name used in the tags. Note that the end tag's name is preceded by a slash character, `/`, and that in empty elements the end tag is neither required nor allowed. If attributes are not mentioned, default values are used in each case.

Semantic HTML is a way of writing HTML that emphasizes the meaning of the encoded information over its presentation (look). HTML has included semantic markup from its inception, but has also included presentational markup, such as `<font>`, `<i>` and `<center>` tags. There are also the semantically neutral `span` and `div` tags. Since the late 1990s when Cascading Style Sheets were beginning to work in most browsers, web authors have been encouraged to avoid the use of presentational HTML markup with a view to the separation of presentation and content.

In a 2001 discussion of the Semantic Web, Tim Berners-Lee and others gave examples of ways in which intelligent software 'agents' may one day automatically crawl the web and find, filter and correlate previously unrelated, published facts for the benefit of human users. Such agents are not commonplace even now, but some of the ideas of Web 2.0, mash-ups and price comparison websites may be coming close. The main difference between these web application hybrids and Berners-Lee's semantic agents lies in the fact that the current aggregation and hybridization of information is usually designed in by web developers, who already know the web locations and the API semantics of the specific data they wish to mash, compare and combine.

An important type of web agent that does crawl and read web pages automatically, without prior knowledge of what it might find, is the web crawler or search-engine spider. These software agents are dependent on the semantic clarity of web pages they find as they use various techniques and algorithms to read and index millions of web pages a day and provide web users with search facilities without which the World Wide Web's usefulness would be greatly reduced.

In order for search-engine spiders to be able to rate the significance of pieces of text they find in HTML documents, and also for those creating mashups and other hybrids as well as for more automated agents as they are developed, the semantic structures that exist in HTML need to be widely and uniformly applied to bring out the meaning of published text.

Presentational markup tags are deprecated in current HTML and XHTML recommendations and are illegal in HTML5.

Good semantic HTML also improves the accessibility of web documents (see also Web Content Accessibility Guidelines). For example, when a screen reader or audio browser can correctly ascertain the structure of a document, it will not waste the visually impaired user's time by reading out repeated or irrelevant information when it has been marked up correctly.

One difference in the latest HTML specifications lies in the distinction between the SGML-based specification and the XML-based specification. The XML-based specification is usually called XHTML to distinguish it clearly from the more traditional definition. However, the root element name continues to be 'html' even in the XHTML-specified HTML. The W3C intended XHTML 1.0 to be identical to HTML 4.01 except where limitations of XML over the more complex SGML require workarounds. Because XHTML and HTML are closely related, they are sometimes documented in parallel. In such circumstances, some authors conflate the two names as (X)HTML or X(HTML). Like HTML 4.01, XHTML 1.0 has three sub-specifications: strict, transitional and frameset.

Aside from the different opening declarations for a document, the differences between an HTML 4.01 and XHTML 1.0 document—in each of the corresponding DTDs—are largely syntactic. The underlying syntax of HTML allows many shortcuts that XHTML



does not, such as elements with optional opening or closing tags, and even EMPTY elements which must not have an end tag. By contrast, XHTML requires all elements to have an opening tag and a closing tag. XHTML, however, also introduces a new shortcut: an XHTML tag may be opened and closed within the same tag, by including a slash before the end of the tag like this: `<br/>`. The introduction of this shorthand, which is not used in the SGML declaration for HTML 4.01, may confuse earlier software unfamiliar with this new convention. A fix for this is to include a space before closing the tag, as such: `<br />`. To understand the subtle differences between HTML and XHTML, consider the transformation of a valid and well-formed XHTML 1.0 document that adheres to Appendix C (see below) into a valid HTML 4.01 document. To make this translation requires the following steps:

- a) The language for an element should be specified with a lang attribute rather than the XHTML `xml:lang` attribute. XHTML uses XML's built in language-defining functionality attribute.
- b) Remove the XML namespace (`xmlns=URI`). HTML has no facilities for namespaces.
- c) Change the document type declaration from XHTML 1.0 to HTML 4.01.
- d) If present, remove the XML declaration. (Typically this is: `<?xml version="1.0" encoding="utf-8"?>`).
- e) Ensure that the document's MIME type is set to `text/html`. For both HTML and XHTML, this comes from the HTTP Content-Type header sent by the server.
- f) Change the XML empty-element syntax to an HTML style empty element (`<br/>` to `<br>`).

Those are the main changes necessary to translate a document from XHTML 1.0 to HTML 4.01. To translate from HTML to XHTML would also require the addition of any omitted opening or closing tags. Whether coding in HTML or XHTML it may just be best to always include the optional tags within an HTML document rather than remembering which tags can be omitted.

A well-formed XHTML document adheres to all the syntax requirements of XML. A valid document adheres to the content specification for XHTML, which describes the document structure.

The W3C recommends several conventions to ensure an easy migration between HTML and XHTML (see HTML Compatibility Guidelines). The following steps can be applied to XHTML 1.0 documents only: Include both `xml:lang` and `lang` attributes on any elements assigning language. Use the empty-element syntax only for elements specified as empty in HTML. Include an extra space in empty-element tags: for example `<br />` instead of `<br/>`. Include explicit close tags for elements that permit content but are left empty (for example, `<div></div>`, not `<div />`).

For documents that are XHTML 1.0 and have been made compatible in this way, the W3C permits them to be served either as HTML (with a `text/html` MIME type), or as XHTML (with an `application/xhtml+xml` or `application/xml` MIME type). When delivered as XHTML, browsers should use an XML parser, which adheres strictly to the XML specifications for parsing the document's contents.

### **3.2.6 XML**

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for the languages of the world. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

Many application programming interfaces (APIs) have been developed to aid software developers with processing XML data, and several schema systems exist to aid in the definition of XML-based languages.

As of 2009, hundreds of document formats using XML syntax have been developed, including RSS, Atom, SOAP, and XHTML. XML-based formats have become the default for many office-productivity tools, including Microsoft Office (Office Open XML),

OpenOffice.org and LibreOffice (OpenDocument), and Apple's iWork. XML has also been employed as the base language for communication protocols, such as XMPP. Applications for the Microsoft .NET Framework use XML files for configuration. Apple has an implementation of a registry based on XML.

XML has come into common use for the interchange of data over the Internet. RFC 3023 gives rules for the construction of Internet Media Types for use when sending XML. It also defines the media types application/xml and text/xml, which say only that the data are in XML, and nothing about its semantics. The use of text/xml has been criticized as a potential source of encoding problems and it has been suggested that it should be deprecated.

RFC 3023 also recommends that XML-based languages be given media types ending in +xml; for example image/svg+xml for SVG. Further guidelines for the use of XML in a networked context may be found in RFC 3470, also known as IETF BCP 70; this document is very wide-ranging and covers many aspects of designing and deploying an XML-based language.

The XML specification defines an XML document as a well-formed text – meaning that it satisfies a list of syntax rules provided in the specification. Some key points in the fairly lengthy list include:

- a) The document contains only properly encoded legal Unicode characters
- b) None of the special syntax characters such as < and & appear except when performing their markup-delineation roles
- c) The begin, end, and empty-element tags that delimit the elements are correctly nested, with none missing and none overlapping
- d) The element tags are case-sensitive; the beginning and end tags must match exactly. Tag names cannot contain any of the characters !"#\$%&'()\*+,-/;<=>?@[\\]^\_`{|}~ , nor a space character, and cannot start with -, ., or a numeric digit.
- e) A single "root" element contains all the other elements

The definition of an XML document excludes texts that contain violations of well-formedness rules; they are simply not XML. An XML processor that encounters such a violation is required to report such errors and to cease normal processing. This policy,

occasionally referred to as "draconian error handling," stands in notable contrast to the behavior of programs that process HTML, which are designed to produce a reasonable result even in the presence of severe markup errors. XML's policy in this area has been criticized as a violation of Postel's law ("Be conservative in what you send; be liberal in what you accept").

The XML specification defines a valid XML document as a well-formed XML document which also conforms to the rules of a Document Type Definition (DTD). By extension, the term can also refer to documents that conform to rules in other schema languages, such as XML Schema (XSD). This term should not be confused with a well-formed XML document, which is defined as an XML document that has correct XML syntax according to W3C standards.

In addition to being well-formed, an XML document may be valid. This means that it contains a reference to a Document Type Definition (DTD), and that its elements and attributes are declared in that DTD and follow the grammatical rules for them that the DTD specifies.

XML processors are classified as validating or non-validating depending on whether or not they check XML documents for validity. A processor that discovers a validity error must be able to report it, but may continue normal processing.

A DTD is an example of a schema or grammar. Since the initial publication of XML 1.0, there has been substantial work in the area of schema languages for XML. Such schema languages typically constrain the set of elements that may be used in a document, which attributes may be applied to them, the order in which they may appear, and the allowable parent/child relationships.

**Document Type Definition:** The oldest schema language for XML is the Document Type Definition (DTD), inherited from SGML.

DTDs have the following benefits:

- a) DTD support is ubiquitous due to its inclusion in the XML 1.0 standard.
- b) DTDs are terse compared to element-based schema languages and consequently present more information in a single screen.
- c) DTDs allow the declaration of standard public entity sets for publishing characters.

- d) DTDs define a document type rather than the types used by a namespace, thus grouping all constraints for a document in a single collection.

DTDs have the following limitations:

- a) They have no explicit support for newer features of XML, most importantly namespaces.
- b) They lack expressiveness. XML DTDs are simpler than SGML DTDs and there are certain structures that cannot be expressed with regular grammars. DTDs only support rudimentary datatypes.
- c) They lack readability. DTD designers typically make heavy use of parameter entities (which behave essentially as textual macros), which make it easier to define complex grammars, but at the expense of clarity.
- d) They use a syntax based on regular expression syntax, inherited from SGML, to describe the schema. Typical XML APIs such as SAX do not attempt to offer applications a structured representation of the syntax, so it is less accessible to programmers than an element-based syntax may be.

Two peculiar features that distinguish DTDs from other schema types are the syntactic support for embedding a DTD within XML documents and for defining entities, which are arbitrary fragments of text and/or markup that the XML processor inserts in the DTD itself and in the XML document wherever they are referenced, like character escapes. DTD technology is still used in many applications because of its ubiquity.

**XML Schema:** A newer schema language, described by the W3C as the successor of DTDs, is XML Schema, often referred to by the initialism for XML Schema instances, XSD (XML Schema Definition). XSDs are far more powerful than DTDs in describing XML languages. They use a rich datatyping system and allow for more detailed constraints on an XML document's logical structure. XSDs also use an XML-based format, which makes it possible to use ordinary XML tools to help process them.

The design goals of XML include, "It shall be easy to write programs which process XML documents." Despite this, the XML specification contains almost no information about how programmers might go about doing such processing. The XML Infoset specification provides a vocabulary to refer to the constructs within an XML document, but also does not

provide any guidance on how to access this information. A variety of APIs for accessing XML have been developed and used, and some have been standardized.

Existing APIs for XML processing tend to fall into these categories:

- a) Stream-oriented APIs accessible from a programming language, for example SAX and StAX.
- b) Tree-traversal APIs accessible from a programming language, for example DOM.
- c) XML data binding, which provides an automated translation between an XML document and programming-language objects.
- d) Declarative transformation languages such as XSLT and XQuery.

Stream-oriented facilities require less memory and, for certain tasks which are based on a linear traversal of an XML document, are faster and simpler than other alternatives. Tree-traversal and data-binding APIs typically require the use of much more memory, but are often found more convenient for use by programmers; some include declarative retrieval of document components via the use of XPath expressions.

XSLT is designed for declarative description of XML document transformations, and has been widely implemented both in server-side packages and Web browsers. XQuery overlaps XSLT in its functionality, but is designed more for searching of large XML databases.

Simple API for XML (SAX) is a lexical, event-driven interface in which a document is read serially and its contents are reported as callbacks to various methods on a handler object of the user's design. SAX is fast and efficient to implement, but difficult to use for extracting information at random from the XML, since it tends to burden the application author with keeping track of what part of the document is being processed. It is better suited to situations in which certain types of information are always handled the same way, no matter where they occur in the document.

Pull parsing treats the document as a series of items which are read in sequence using the Iterator design pattern. This allows for writing of recursive-descent parsers in which the structure of the code performing the parsing mirrors the structure of the XML being parsed, and intermediate parsed results can be used and accessed as local variables within the methods performing the parsing, or passed down (as method parameters) into lower-level

methods, or returned (as method return values) to higher-level methods. Examples of pull parsers include StAX in the Java programming language, XMLReader in PHP, ElementTree.iterparse in Python, System.Xml.XmlReader in the .NET Framework, and the DOM traversal API (NodeIterator and TreeWalker).

A pull parser creates an iterator that sequentially visits the various elements, attributes, and data in an XML document. Code which uses this iterator can test the current item (to tell, for example, whether it is a start or end element, or text), and inspect its attributes (local name, namespace, values of XML attributes, value of text, etc.), and can also move the iterator to the next item. The code can thus extract information from the document as it traverses it. The recursive-descent approach tends to lend itself to keeping data as typed local variables in the code doing the parsing, while SAX, for instance, typically requires a parser to manually maintain intermediate data within a stack of elements which are parent elements of the element being parsed. Pull-parsing code can be more straightforward to understand and maintain than SAX parsing code.

The Document Object Model (DOM) is an interface-oriented application programming interface that allows for navigation of the entire document as if it were a tree of node objects representing the document's contents. A DOM document can be created by a parser, or can be generated manually by users (with limitations). Data types in DOM nodes are abstract; implementations provide their own programming language-specific bindings. DOM implementations tend to be memory intensive, as they generally require the entire document to be loaded into memory and constructed as a tree of objects before access is allowed.

Another form of XML processing API is XML data binding, where XML data are made available as a hierarchy of custom, strongly typed classes, in contrast to the generic objects created by a Document Object Model parser. This approach simplifies code development, and in many cases allows problems to be identified at compile time rather than run-time. Example data binding systems include the Java Architecture for XML Binding (JAXB) and XML Serialization in .NET.

XML has appeared as a first-class data type in other languages. The ECMAScript for XML (E4X) extension to the ECMAScript/JavaScript language explicitly defines two

specific objects (XML and XMLList) for JavaScript, which support XML document nodes and XML node lists as distinct objects and use a dot-notation specifying parent-child relationships. E4X is supported by the Mozilla 2.5+ browsers (though now deprecated) and Adobe Actionscript, but has not been adopted more universally. Similar notations are used in Microsoft's LINQ implementation for Microsoft .NET 3.5 and above, and in Scala (which uses the Java VM). The open-source xmlsh application, which provides a Linux-like shell with special features for XML manipulation, similarly treats XML as a data type, using the `<[ ]>` notation. The Resource Description Framework defines a data type `rdf:XMLLiteral` to hold wrapped, canonical XML.

XML is an application profile of SGML (ISO 8879). The versatility of SGML for dynamic information display was understood by early digital media publishers in the late 1980s prior to the rise of the Internet. By the mid-1990s some practitioners of SGML had gained experience with the then-new World Wide Web, and believed that SGML offered solutions to some of the problems the Web was likely to face as it grew. Dan Connolly added SGML to the list of W3C's activities when he joined the staff in 1995; work began in mid-1996 when Sun Microsystems engineer Jon Bosak developed a charter and recruited collaborators. Bosak was well connected in the small community of people who had experience both in SGML and the Web.

XML was compiled by a working group of eleven members, supported by a (roughly) 150-member Interest Group. Technical debate took place on the Interest Group mailing list and issues were resolved by consensus or, when that failed, majority vote of the Working Group. A record of design decisions and their rationales was compiled by Michael Sperberg-McQueen on December 4, 1997. James Clark served as Technical Lead of the Working Group, notably contributing the empty-element `<empty />` syntax and the name "XML". Other names that had been put forward for consideration included "MAGMA" (Minimal Architecture for Generalized Markup Applications), "SLIM" (Structured Language for Internet Markup) and "MGML" (Minimal Generalized Markup Language). The co-editors of the specification were originally Tim Bray and Michael Sperberg-McQueen. Halfway through the project Bray accepted a consulting engagement with Netscape, provoking vociferous protests from Microsoft. Bray was temporarily asked to



resign the editorship. This led to intense dispute in the Working Group, eventually solved by the appointment of Microsoft's Jean Paoli as a third co-editor.

The XML Working Group never met face-to-face; the design was accomplished using a combination of email and weekly teleconferences. The major design decisions were reached in a short burst of intense work between August and November 1996, when the first Working Draft of an XML specification was published. Further design work continued through 1997, and XML 1.0 became a W3C Recommendation on February 10, 1998.

XML is a profile of an ISO standard SGML, and most of XML comes from SGML unchanged. From SGML comes the separation of logical and physical structures (elements and entities), the availability of grammar-based validation (DTDs), the separation of data and metadata (elements and attributes), mixed content, the separation of processing from representation (processing instructions), and the default angle-bracket syntax. Removed were the SGML declaration (XML has a fixed delimiter set and adopts Unicode as the document character set).

Other sources of technology for XML were the Text Encoding Initiative (TEI), which defined a profile of SGML for use as a "transfer syntax"; and HTML, in which elements were synchronous with their resource, document character sets were separate from resource encoding, the `xml:lang` attribute was invented, and (like HTTP) metadata accompanied the resource rather than being needed at the declaration of a link. The Extended Reference Concrete Syntax (ERCS) project of the SPREAD (Standardization Project Regarding East Asian Documents) project of the ISO-related China/Japan/Korea Document Processing expert group was the basis of XML 1.0's naming rules; SPREAD also introduced hexadecimal numeric character references and the concept of references to make available all Unicode characters. To support ERCS, XML and HTML better, the SGML standard IS 8879 was revised in 1996 and 1998 with WebSGML Adaptations. The XML header followed that of ISO HyTime.

Ideas that developed during discussion which were novel in XML included the algorithm for encoding detection and the encoding header, the processing instruction target, the `xml:space` attribute, and the new close delimiter for empty-element tags. The notion of well-formedness as opposed to validity (which enables parsing without a schema) was first

formalized in XML, although it had been implemented successfully in the Electronic Book Technology "Dynatext" software; the software from the University of Waterloo New Oxford English Dictionary Project; the RISP LISP SGML text processor at Uniscope, Tokyo; the US Army Missile Command IADS hypertext system; Mentor Graphics Context; Interleaf and Xerox Publishing System.

There are two current versions of XML. The first (XML 1.0) was initially defined in 1998. It has undergone minor revisions since then, without being given a new version number, and is currently in its fifth edition, as published on November 26, 2008. It is widely implemented and still recommended for general use.

The second (XML 1.1) was initially published on February 4, 2004, the same day as XML 1.0 Third Edition, and is currently in its second edition, as published on August 16, 2006. It contains features (some contentious) that are intended to make XML easier to use in certain cases. The main changes are to enable the use of line-ending characters used on EBCDIC platforms, and the use of scripts and characters absent from Unicode 3.2. XML 1.1 is not very widely implemented and is recommended for use only by those who need its unique features.

Prior to its fifth edition release, XML 1.0 differed from XML 1.1 in having stricter requirements for characters available for use in element and attribute names and unique identifiers: in the first four editions of XML 1.0 the characters were exclusively enumerated using a specific version of the Unicode standard (Unicode 2.0 to Unicode 3.2.) The fifth edition substitutes the mechanism of XML 1.1, which is more future-proof but reduces redundancy. The approach taken in the fifth edition of XML 1.0 and in all editions of XML 1.1 is that only certain characters are forbidden in names, and everything else is allowed, in order to accommodate the use of suitable name characters in future versions of Unicode. In the fifth edition, XML names may contain characters in the Balinese, Cham, or Phoenician scripts among many others which have been added to Unicode since Unicode 3.2.

Almost any Unicode code point can be used in the character data and attribute values of an XML 1.0 or 1.1 document, even if the character corresponding to the code point is not defined in the current version of Unicode. In character data and attribute values, XML 1.1

allows the use of more control characters than XML 1.0, but, for "robustness", most of the control characters introduced in XML 1.1 must be expressed as numeric character references (and #x7F through #x9F, which had been allowed in XML 1.0, are in XML 1.1 even required to be expressed as numeric character references). Among the supported control characters in XML 1.1 are two line break codes that must be treated as whitespace. Whitespace characters are the only control codes that can be written directly.

There has been discussion of an XML 2.0, although no organization has announced plans for work on such a project. XML-SW (SW for skunkworks), written by one of the original developers of XML, contains some proposals for what an XML 2.0 might look like: elimination of DTDs from syntax, integration of namespaces, XML Base and XML Information Set (infoset) into the base standard.

The World Wide Web Consortium also has an XML Binary Characterization Working Group doing preliminary research into use cases and properties for a binary encoding of the XML infoset. The working group is not chartered to produce any official standards. Since XML is by definition text-based, ITU-T and ISO are using the name Fast Infoset for their own binary infoset to avoid confusion (see ITU-T Rec. X.891 | ISO/IEC 24824-1).

### **3.2.7 NETWORKING**

In the world of computers, networking is the practice of linking two or more computing devices together for the purpose of sharing data. Networks are built with a mix of computer hardware and computer software.

Computer networks also differ in their design. The two basic forms of network design are called client/server and peer-to-peer. Client-server networks feature centralized server computers that store email, Web pages, files and or applications. On a peer-to-peer network, conversely, all computers tend to support the same functions. Client-server networks are much more common in business and peer-to-peer networks much more common in homes. A network topology represents its layout or structure from the point of view of data flow. In so-called bus networks, for example, all of the computers share and communicate across one common conduit, whereas in a star network, all data flows through one centralized device. Common types of network topologies include bus, star, ring networks and mesh networks.

Communication languages used by computer devices are called network protocol. Yet another way to classify computer networks is by the set of protocols they support. Networks often implement multiple protocols with each supporting specific applications. Popular protocols include TCP/IP, the most common protocol found on the Internet and in home networks.

A computer network or data network is a telecommunications network that allows computers to exchange data. In computer networks, networked computing devices pass data to each other along data connections. The connections (network links) between nodes are established using either cable media or wireless media. The best-known computer network is the Internet.

Network computer devices that originate, route and terminate the data are called network nodes. Nodes can include hosts such as personal computers, phones, servers as well as networking hardware. Two such devices are said to be networked together when one device is able to exchange information with the other device, whether or not they have a direct connection to each other.

Computer networks support applications such as access to the World Wide Web, shared use of application and storage servers, printers, and fax machines, and use of email and instant messaging applications. Computer networks differ in the physical media used to transmit their signals, the communications protocols to organize network traffic, the network's size, topology and organizational intent.

A communications protocol is a set of rules for exchanging information over network links. In a protocol stack (also see the OSI model), each protocol leverages the services of the protocol below it. An important example of a protocol stack is HTTP running over TCP over IP over IEEE 802.11. (TCP and IP are members of the Internet Protocol Suite. IEEE 802.11 is a member of the Ethernet protocol suite.) This stack is used between the wireless router and the home user's personal computer when the user is surfing the web.

Whilst the use of protocol layering is today ubiquitous across the field of computer networking, it has been historically criticized by many researchers for two principle reasons. Firstly, abstracting the protocol stack in this way may cause a higher layer to duplicate functionality of a lower layer, a prime example being error recovery on both a

per-link basis and an end-to-end basis. Secondly, it is common that a protocol implementation at one layer may require data, state or addressing information that is only present at another layer, thus defeating the point of separating the layers in the first place. For example, TCP uses the ECN field in the IPv4 header as an indication of congestion; IP is a network layer protocol whereas TCP is a transport layer protocol.

Communication protocols have various characteristics. They may be connection-oriented or connectionless, they may use circuit mode or packet switching, and they may use hierarchical addressing or flat addressing.

A network can be characterized by its physical capacity or its organizational purpose. Use of the network, including user authorization and access rights, differ accordingly.

**Personal area network:** A personal area network (PAN) is a computer network used for communication among computer and different information technological devices close to one person. Some examples of devices that are used in a PAN are personal computers, printers, fax machines, telephones, PDAs, scanners, and even video game consoles. A PAN may include wired and wireless devices. The reach of a PAN typically extends to 10 meters. A wired PAN is usually constructed with USB and FireWire connections while technologies such as Bluetooth and infrared communication typically form a wireless PAN.

**Local area network:** A local area network (LAN) is a network that connects computers and devices in a limited geographical area such as a home, school, office building, or closely positioned group of buildings. Each computer or device on the network is a node. Wired LANs are most likely based on Ethernet technology. Newer standards such as ITU-T G.hn also provide a way to create a wired LAN using existing wiring, such as coaxial cables, telephone lines, and power lines.

A LAN is depicted in the accompanying diagram. All interconnected devices use the network layer (layer 3) to handle multiple subnets (represented by different colors). Those inside the library have 10/100 Mbit/s Ethernet connections to the user device and a Gigabit Ethernet connection to the central router. They could be called Layer 3 switches, because they only have Ethernet interfaces and support the Internet Protocol. It might be more correct to call them access routers, where the router at the top is a distribution router that connects to the Internet and to the academic networks' customer access routers.

The defining characteristics of a LAN, in contrast to a wide area network (WAN), include higher data transfer rates, limited geographic range, and lack of reliance on leased lines to provide connectivity. Current Ethernet or other IEEE 802.3 LAN technologies operate at data transfer rates up to 10 Gbit/s. The IEEE investigates the standardization of 40 and 100 Gbit/s rates. A LAN can be connected to a WAN using a router.

**Home area network:** A home area network (HAN) is a residential LAN which is used for communication between digital devices typically deployed in the home, usually a small number of personal computers and accessories, such as printers and mobile computing devices. An important function is the sharing of Internet access, often a broadband service through a cable TV or digital subscriber line (DSL) provider.

**Storage area network:** A storage area network (SAN) is a dedicated network that provides access to consolidated, block level data storage. SANs are primarily used to make storage devices, such as disk arrays, tape libraries, and optical jukeboxes, accessible to servers so that the devices appear like locally attached devices to the operating system. A SAN typically has its own network of storage devices that are generally not accessible through the local area network by other devices. The cost and complexity of SANs dropped in the early 2000s to levels allowing wider adoption across both enterprise and small to medium sized business environments.

**Campus area network:** A campus area network (CAN) is made up of an interconnection of LANs within a limited geographical area. The networking equipment (switches, routers) and transmission media (optical fiber, copper plant, Cat5 cabling, etc.) are almost entirely owned by the campus tenant / owner (an enterprise, university, government, etc.).

For example, a university campus network is likely to link a variety of campus buildings to connect academic colleges or departments, the library, and student residence halls.

**Backbone network:** A backbone network is part of a computer network infrastructure that provides a path for the exchange of information between different LANs or sub-networks. A backbone can tie together diverse networks within the same building, across different buildings, or over a wide area.

For example, a large company might implement a backbone network to connect departments that are located around the world. The equipment that ties together the departmental networks constitutes the network backbone. When designing a network backbone, network performance and network congestion are critical factors to take into account. Normally, the backbone network's capacity is greater than that of the individual networks connected to it.

Another example of a backbone network is the Internet backbone, which is the set of wide area networks (WANs) and core routers that tie together all networks connected to the Internet.

**Metropolitan area network:** A Metropolitan area network (MAN) is a large computer network that usually spans a city or a large campus.

**Wide area network:** A wide area network (WAN) is a computer network that covers a large geographic area such as a city, country, or spans even intercontinental distances. A WAN uses a communications channel that combines many types of media such as telephone lines, cables, and air waves. A WAN often makes use of transmission facilities provided by common carriers, such as telephone companies. WAN technologies generally function at the lower three layers of the OSI reference model: the physical layer, the data link layer, and the network layer.

**Enterprise private network:** An enterprise private network is a network built by a single organization to interconnect its office locations (e.g., production sites, head offices, remote offices, shops) in order to share computer resources.

**Virtual private network:** A virtual private network (VPN) is an overlay network in which some of the links between nodes are carried by open connections or virtual circuits in some larger network (e.g., the Internet) instead of by physical wires. The data link layer protocols of the virtual network are said to be tunneled through the larger network when this is the case. One common application is secure communications through the public Internet, but a VPN need not have explicit security features, such as authentication or content encryption. VPNs, for example, can be used to separate the traffic of different user communities over an underlying network with strong security features. VPN may have best-effort performance, or may have a defined service level agreement (SLA) between the

VPN customer and the VPN service provider. Generally, a VPN has a topology more complex than point-to-point.

**Global area network:** A global area network (GAN) is a network used for supporting mobile across an arbitrary number of wireless LANs, satellite coverage areas, etc. The key challenge in mobile communications is handing off user communications from one local coverage area to the next. In IEEE Project 802, this involves a succession of terrestrial wireless LANs.

### 3.2.7.1 TCP/IP STACK

The TCP/IP stack is shorter than the OSI one. TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

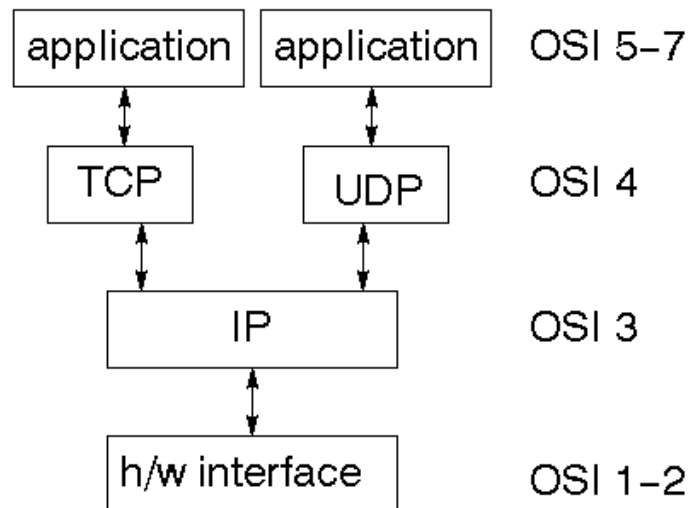


Figure 3.6: TCP/IP Stack

The Internet protocol suite is the networking model and a set of communications protocols used for the Internet and similar networks. It is commonly known as TCP/IP, because its most important protocols, the Transmission Control Protocol (TCP) and the Internet Protocol (IP), were the first networking protocols defined in this standard. It is occasionally known as the DoD model, because the development of the networking model was funded by DARPA, an agency of the United States Department of Defense.



TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. This functionality has been organized into four abstraction layers which are used to sort all related protocols according to the scope of networking involved. From lowest to highest, the layers are the link layer, containing communication technologies for a single network segment (link), the internet layer, connecting independent networks, thus establishing internetworking, the transport layer handling process-to-process communication, and the application layer, which interfaces to the user and provides support services.

The TCP/IP model and related protocols are maintained by the Internet Engineering Task Force (IETF).

The Internet protocol suite uses encapsulation to provide abstraction of protocols and services. Encapsulation is usually aligned with the division of the protocol suite into layers of general functionality. In general, an application (the highest level of the model) uses a set of protocols to send its data down the layers, being further encapsulated at each level. The layers of the protocol suite near the top are logically closer to the user application, while those near the bottom are logically closer to the physical transmission of the data. Viewing layers as providing or consuming a service is a method of abstraction to isolate upper layer protocols from the details of transmitting bits over, for example, Ethernet and collision detection, while the lower layers avoid having to know the details of each and every application and its protocol.

Even when the layers are examined, the assorted architectural documents—there is no single architectural model such as ISO 7498, the Open Systems Interconnection (OSI) model—have fewer and less rigidly defined layers than the OSI model, and thus provide an easier fit for real-world protocols. One frequently referenced document, RFC 1958, does not contain a stack of layers. The lack of emphasis on layering is a major difference between the IETF and OSI approaches. It only refers to the existence of the internetworking layer and generally to upper layers; this document was intended as a 1996 snapshot of the architecture: "The Internet and its architecture have grown in evolutionary fashion from modest beginnings, rather than from a Grand Plan. While this process of

evolution is one of the main reasons for the technology's success, it nevertheless seems useful to record a snapshot of the current principles of the Internet architecture."

RFC 1122, entitled Host Requirements, is structured in paragraphs referring to layers, but the document refers to many other architectural principles not emphasizing layering. It loosely defines a four-layer model, with the layers having names, not numbers, as follows:

**Application layer (user interface services and support services):** This is the scope within which applications create user data and communicate this data to other applications on another or the same host. The communications partners are often called peers. This is where the higher level protocols such as SMTP, FTP, SSH, HTTP, etc. operate.

**Transport layer (process-to-process):** The transport layer constitutes the networking regime between two network processes, on either the same or different hosts and on either the local network or remote networks separated by routers. Processes are addressed via "ports," and the transport layer header contains the port numbers. UDP is the basic transport layer protocol, providing communication between processes via port addresses in the header. Also, some OSI session layer services such as flow-control, error-correction, and connection establishment and teardown protocols belong at the transport layer. In the Internet protocol suite, TCP provides flow-control, connection establishment, and reliable transmission of data.

**Internet layer:** The internet layer has the task of exchanging datagrams across network boundaries. It provides a uniform networking interface that hides the actual topology (layout) of the underlying network connections. It is therefore also referred to as the layer that establishes internetworking, indeed, it defines and establishes the Internet. This layer defines the addressing and routing structures used for the TCP/IP protocol suite. The primary protocol in this scope is the Internet Protocol, which defines IP addresses. Its function in routing is to transport datagrams to the next IP router that has the connectivity to a network closer to the final data destination.

**Link layer:** This layer defines the networking methods within the scope of the local network link on which hosts communicate without intervening routers. This layer describes the protocols used to describe the local network topology and the interfaces needed to effect transmission of Internet layer datagrams to next-neighbor hosts.

The Internet protocol suite and the layered protocol stack design were in use before the OSI model was established. Since then, the TCP/IP model has been compared with the OSI model in books and classrooms, which often results in confusion because the two models use different assumptions and goals, including the relative importance of strict layering. This abstraction also allows upper layers to provide services that the lower layers do not provide. While the original OSI model was extended to include connectionless services (OSIRM CL), IP is not designed to be reliable and is a best effort delivery protocol. This means that all transport layer implementations must choose whether or how to provide reliability. UDP provides data integrity via a checksum but does not guarantee delivery; TCP provides both data integrity and delivery guarantee by retransmitting until the receiver acknowledges the reception of the packet.

This model lacks the formalism of the OSI model and associated documents, but the IETF does not use a formal model and does not consider this a limitation, as illustrated in the comment by David D. Clark, "We reject: kings, presidents and voting. We believe in: rough consensus and running code." Criticisms of this model, which have been made with respect to the OSI model, often do not consider ISO's later extensions to that model. For multi-access links with their own addressing systems (e.g. Ethernet) an address mapping protocol is needed. Such protocols can be considered to be below IP but above the existing link system. While the IETF does not use the terminology, this is a sub-network dependent convergence facility according to an extension to the OSI model, the internal organization of the network layer (IONL).

ICMP & IGMP operate on top of IP but do not transport data like UDP or TCP. Again, this functionality exists as layer management extensions to the OSI model, in its Management Framework (OSIRM MF).

The SSL/TLS library operates above the transport layer (uses TCP) but below application protocols. Again, there was no intention, on the part of the designers of these protocols, to comply with OSI architecture.

The link is treated like a black box. The IETF explicitly does not intend to discuss transmission systems, which is a less academic but practical alternative to the OSI model.

The following is a description of each layer in the TCP/IP networking model starting from the lowest level.

- a) **Link layer:** The link layer has the networking scope of the local network connection to which a host is attached. This regime is called the link in TCP/IP literature. It is the lowest component layer of the Internet protocols, as TCP/IP is designed to be hardware independent. As a result TCP/IP may be implemented on top of virtually any hardware networking technology. The link layer is used to move packets between the Internet layer interfaces of two different hosts on the same link.

The processes of transmitting and receiving packets on a given link can be controlled both in the software device driver for the network card, as well as on firmware or specialized chipsets. These perform data link functions such as adding a packet header to prepare it for transmission, then actually transmit the frame over a physical medium. The TCP/IP model includes specifications of translating the network addressing methods used in the Internet Protocol to data link addressing, such as Media Access Control (MAC).

All other aspects below that level, however, are implicitly assumed to exist in the link layer, but are not explicitly defined. This is also the layer where packets may be selected to be sent over a virtual private network or other networking tunnel. In this scenario, the link layer data may be considered application data which traverses another instantiation of the IP stack for transmission or reception over another IP connection. Such a connection, or virtual link, may be established with a transport protocol or even an application scope protocol that serves as a tunnel in the link layer of the protocol stack.

Thus, the TCP/IP model does not dictate a strict hierarchical encapsulation sequence. The TCP/IP model's link layer corresponds to the Open Systems Interconnection (OSI) model physical and data link layers, layers one and two of the OSI model.

- b) **Internet layer:** The internet layer has the responsibility of sending packets across potentially multiple networks. Internetworking requires sending data from the

source network to the destination network. This process is called routing. In the Internet protocol suite, the Internet Protocol performs two basic functions:

- i. **Host addressing and identification:** This is accomplished with a hierarchical IP addressing system.
- ii. **Packet routing:** This is the basic task of sending packets of data (datagrams) from source to destination by forwarding them to the next network router closer to the final destination.

The internet layer is not only agnostic of application data structures at the transport layer, but it also does not distinguish between operations of the various transport layer protocols. IP can carry data for a variety of different upper layer protocols. These protocols are each identified by a unique protocol number: for example, Internet Control Message Protocol (ICMP) and Internet Group Management Protocol (IGMP) are protocols 1 and 2, respectively.

Some of the protocols carried by IP, such as ICMP which is used to transmit diagnostic information, and IGMP which is used to manage IP Multicast data, are layered on top of IP but perform internetworking functions. This illustrates the differences in the architecture of the TCP/IP stack of the Internet and the OSI model. The TCP/IP model's internet layer corresponds to layer three of the Open Systems Interconnection (OSI) model, where it is referred to as the network layer.

The internet layer provides only an unreliable datagram transmission facility between hosts located on potentially different IP networks by forwarding the transport layer datagrams to an appropriate next-hop router for further relaying to its destination. With this functionality, the internet layer makes possible internetworking, the interworking of different IP networks, and it essentially establishes the Internet. The Internet Protocol is the principal component of the internet layer, and it defines two addressing systems to identify network hosts computers, and to locate them on the network. The original address system of the ARPANET and its successor, the Internet, is Internet Protocol version 4 (IPv4). It uses a 32-bit IP address and is therefore capable of identifying approximately four billion hosts. This limitation was eliminated by the standardization of Internet

Protocol version 6 (IPv6) in 1998, and beginning production implementations in approximately 2006.

- c) **Transport layer:** The transport layer establishes a basic data channel that an application uses in its task-specific data exchange. The layer establishes process-to-process connectivity, meaning it provides end-to-end services that are independent of the structure of user data and the logistics of exchanging information for any particular specific purpose.

Its responsibility includes end-to-end message transfer independent of the underlying network, along with error control, segmentation, flow control, congestion control, and application addressing (port numbers). End to end message transmission or connecting applications at the transport layer can be categorized as either connection-oriented, implemented in TCP, or connectionless, implemented in UDP. For the purpose of providing process-specific transmission channels, the layer establishes the concept of the port. This is a numbered logical construct allocated specifically for each of the communication channels an application needs.

For many types of services, these port numbers have been standardized so that client computers may address specific services of a server computer without the involvement of service announcements or directory services. Because IP provides only a best effort delivery, the transport layer is the first layer of the TCP/IP stack to offer reliability. However, IP can run over a reliable data link protocol such as the High-Level Data Link Control (HDLC). For example, the TCP is a connection-oriented protocol that addresses numerous reliability issues in providing a reliable byte stream:

- i. data arrives in-order
- ii. data has minimal error (i.e. correctness)
- iii. duplicate data is discarded
- iv. lost or discarded packets are resent
- v. includes traffic congestion control

The newer Stream Control Transmission Protocol (SCTP) is also a reliable, connection-oriented transport mechanism. It is message-stream-oriented — not

byte-stream-oriented like TCP — and provides multiple streams multiplexed over a single connection. It also provides multi-homing support, in which a connection end can be represented by multiple IP addresses (representing multiple physical interfaces), such that if one fails, the connection is not interrupted. It was developed initially for telephony applications (to transport SS7 over IP), but can also be used for other applications.

The User Datagram Protocol is a connectionless datagram protocol. Like IP, it is a best effort, "unreliable" protocol. Reliability is addressed through error detection using a weak checksum algorithm. UDP is typically used for applications such as streaming media (audio, video, Voice over IP etc.) where on-time arrival is more important than reliability, or for simple query/response applications like DNS lookups, where the overhead of setting up a reliable connection is disproportionately large. Real-time Transport Protocol (RTP) is a datagram protocol that is designed for real-time data such as streaming audio and video.

The applications at any given network address are distinguished by their TCP or UDP port. By convention certain well known ports are associated with specific applications. The TCP/IP model's transport or host-to-host layer corresponds to the fourth layer in the Open Systems Interconnection (OSI) model, also called the transport layer.

- d) **Application layer:** The application layer protocols used by most applications for providing user services over a network and for some basic network support services. Examples of application layer protocols include the File Transfer Protocol (FTP), the Simple Mail Transfer Protocol (SMTP), and Dynamic Host Configuration Protocol (DHCP). Data coded according to application layer protocols are then encapsulated into one or (occasionally) more transport layer protocols (such as TCP or UDP), which in turn use lower layer protocols to effect actual data transfer.

As the IP model does not consider the specifics of formatting and presenting data, it defines no layers between the application and transport layers as in the OSI

model (presentation and session layers). Such functions are the realm of libraries and application programming interfaces.

Application layer protocols generally treat the transport layer (and lower) protocols as black boxes which provide a stable network connection across which to communicate, although the applications are usually aware of key qualities of the transport layer connection such as the end point IP addresses and port numbers. As noted above, layers are not necessarily clearly defined in the Internet protocol suite. Application layer protocols are most often associated with client–server applications, and the commoner servers have specific ports assigned to them by the IANA: HTTP has port 80; Telnet has port 23; etc. Clients, on the other hand, tend to use ephemeral ports, i.e. port numbers assigned at random from a range set aside for the purpose.

Transport and lower level layers are largely unconcerned with the specifics of application layer protocols. Routers and switches do not typically "look inside" the encapsulated traffic to see what kind of application protocol it represents, rather they just provide a conduit for it. However, some firewall and bandwidth throttling applications do try to determine what's inside, as with the Resource Reservation Protocol (RSVP). It is also sometimes necessary for network address translation (NAT) facilities to take account of the needs of particular application layer protocols.

The TCP/IP model's application layer encompasses the fifth, sixth, and seventh layers of the Open Systems Interconnection (OSI) model, which are the session layer, presentation layer, and application layer, respectively.

### **3.2.7.2 IP DATAGRAM**

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.



### **3.2.7.3 UDP**

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model.

UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as UDP/IP. Like the Transmission Control Protocol, UDP uses the Internet Protocol to actually get a data unit (called a datagram) from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets (datagrams) and reassembling it at the other end. Specifically, UDP doesn't provide sequencing of the packets that the data arrives in. This means that the application program that uses UDP must be able to make sure that the entire message has arrived and is in the right order. Network applications that want to save processing time because they have very small data units to exchange (and therefore very little message reassembling to do) may prefer UDP to TCP. The Trivial File Transfer Protocol (TFTP) uses UDP instead of TCP.

UDP provides two services not provided by the IP layer. It provides port numbers to help distinguish different user requests and, optionally, a checksum capability to verify that the data arrived intact.

In the Open Systems Interconnection (OSI) communication model, UDP, like TCP, is in layer 4, the Transport Layer.

### **3.2.7.4 TCP**

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

TCP (Transmission Control Protocol) is a set of rules (protocol) used along with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. While IP takes care of handling the actual delivery of the data, TCP takes care of keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

For example, when an HTML file is sent to you from a Web server, the Transmission Control Protocol (TCP) program layer in that server divides the file into one or more packets, numbers the packets, and then forwards them individually to the IP program layer. Although each packet has the same destination IP address, it may get routed differently through the network. At the other end (the client program in your computer), TCP reassembles the individual packets and waits until they have arrived to forward them to you as a single file.

TCP is known as a connection-oriented protocol, which means that a connection is established and maintained until such time as the message or messages to be exchanged by the application programs at each end have been exchanged. TCP is responsible for ensuring that a message is divided into the packets that IP manages and for reassembling the packets back into the complete message at the other end. In the Open Systems Interconnection (OSI) communication model, TCP is in layer 4, the Transport Layer.

#### **3.2.7.5 INTERNET ADDRESSES**

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

#### **3.2.7.6 NETWORK ADDRESS**

A network address serves as a unique identifier for a computer on a network. When set up correctly, computers can determine the addresses of other computers on the network and use these addresses to send messages to each other.

One of the best known form of network addressing is the Internet Protocol (IP) address. IP addresses consist of four bytes (32 bits) that uniquely identify all computers on the public Internet.

Another popular form of address is the Media Access Control (MAC) address. MAC addresses are six bytes (48 bits) that manufacturers of network adapters burn into their products to uniquely identify them.

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

### **3.2.7.7 SUBNET ADDRESS**

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

A sub-network, or subnet, is a logically visible subdivision of an IP network. The practice of dividing a network into two or more networks is called subnetting.

All computers that belong to a subnet are addressed with a common, identical, most-significant bit-group in their IP address. This results in the logical division of an IP address into two fields, a network or routing prefix and the rest field or host identifier. The rest field is an identifier for a specific host or network interface.

The routing prefix is expressed in CIDR notation. It is written as the first address of a network, followed by a slash character (/), and ending with the bit-length of the prefix. For example, 192.168.1.0/24 is the prefix of the Internet Protocol Version 4 network starting at the given address, having 24 bits allocated for the network prefix, and the remaining 8 bits reserved for host addressing. The IPv6 address specification 2001:db8::/32 is a large address block with 296 addresses, having a 32-bit routing prefix. In IPv4 the routing prefix is also specified in the form of the subnet mask, which is expressed in quad-dotted decimal representation like an address. For example, 255.255.255.0 is the network mask for the 192.168.1.0/24 prefix. Traffic between sub-networks is exchanged or routed with special gateways called routers which constitute the logical or physical boundaries between the subnets.

The benefits of subnetting vary with each deployment scenario. In the address allocation architecture of the Internet using Classless Inter-Domain Routing (CIDR) and in large organizations, it is necessary to allocate address space efficiently. It may also enhance routing efficiency, or have advantages in network management when sub-networks are administratively controlled by different entities in a larger organization. Subnets may be arranged logically in a hierarchical architecture, partitioning an organization's network address space into a tree-like routing structure.

### **3.2.7.8 HOST ADDRESS**

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

In network addressing, the host address, or the host ID portion of an IP address, is the portion of the address used to identify hosts (any device requiring a Network Interface Card, such as a PC or networked printer) on the network. The network ID, by contrast, is the portion of the address that refers to the network itself.

### **3.2.7.9 TOTAL ADDRESS**

The 32 bit address is usually written as 4 integers separated by dots.

### **3.2.7.10 PORT ADDRESSES**

In computer networking, a port is an application-specific or process-specific software construct serving as a communications endpoint in a computer's host operating system. A port is associated with an IP address of the host, as well as the type of protocol used for communication. The purpose of ports is to uniquely identify different applications or processes running on a single computer and thereby enable them to share a single physical connection to a packet-switched network like the Internet.

The protocols that primarily use ports are the Transport Layer protocols, such as the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) of the Internet Protocol Suite. A port is identified for each address and protocol by a 16-bit number, commonly known as the port number. The port number, added to a computer's IP address, completes the destination address for a communications session. That is, data packets are routed across the network to a specific destination IP address, and then, upon reaching the destination computer, are further routed to the specific process bound to the destination port number.

Note that it is the combination of IP address and port number together that must be globally unique. Thus, different IP addresses or protocols may use the same port number for communication; e.g., on a given host or interface UDP and TCP may use the same port number, or on a host with two interfaces, both addresses may be associated with a port having the same number.

Of the thousands of enumerated ports, about 250 well-known ports are reserved by convention to identify specific service types on a host. In the client-server model of application architecture, ports are used to provide a multiplexing service on each server-side port number that network clients connect to for service initiation, after which communication can be reestablished on other connection-specific port numbers.

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

### **3.2.7.11 SOCKETS**

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call `socket`. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with Read File and Write File functions.

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int family, int type, int protocol);
```

Here "family" will be `AF_INET` for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

### **3.2.8 TOMCAT 6.0 WEB SERVER**

Tomcat is an open source web server developed by Apache Group. Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process. Web Servers like Apache Tomcat support only web components while an application server supports web components as well as business components (BEAs Weblogic, is one of the popular application server). To develop a web application with jsp/servlet install any web server like JRun, Tomcat etc to run your application.

Apache Tomcat (or simply Tomcat, formerly also Jakarta Tomcat) is an open source web server and servlet container developed by the Apache Software Foundation (ASF).

Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP web server environment for Java code to run in. Apache Tomcat includes tools for configuration and management, but can also be configured by editing XML configuration files.

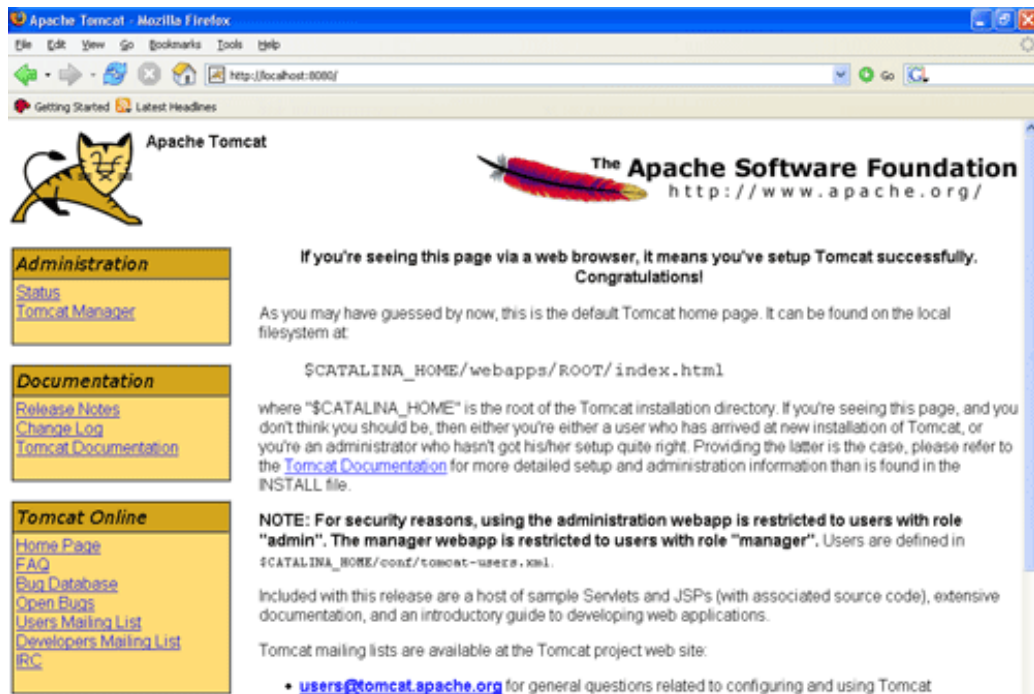


Figure 3.7: Tomcat Web Server

### 3.2.9 JFREE CHART

JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. JFreeChart's extensive feature set includes: A consistent and well-documented API, supporting a wide range of chart types; A flexible design that is easy to extend, and targets both server-side and client-side applications; Support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG).

JFreeChart is "open source" or, more specifically, free software. It is distributed under the terms of the GNU Lesser General Public Licence (LGPL), which permits use in proprietary applications.

- a) **Map Visualizations:**Charts showing values that relate to geographical areas. Some examples include: (a) population density in each state of the United States, (b) income per capita for each country in Europe, (c) life expectancy in each country of the world. The tasks in this project include:Sourcing freely redistributable vector outlines for the countries of the world, states/provinces in particular countries (USA in particular, but also other areas); Creating an appropriate dataset interface (plus default implementation), a rendered, and integrating this with the existing XYPlot class in JFreeChart; Testing, documenting, testing some more, documenting some more.
- b) **Time Series Chart Interactivity:**Implement a new (to JFreeChart) feature for interactive time series charts --- to display a separate control that shows a small version of ALL the time series data, with a sliding "view" rectangle that allows you to select the subset of the time series data to display in the main chart.
- c) **Dashboards:**There is currently a lot of interest in dashboard displays. Create a flexible dashboard mechanism that supports a subset of JFreeChart chart types (dials, pies, thermometers, bars, and lines/time series) that can be delivered easily via both Java Web Start and an applet.
- d) **Property Editors:**The property editor mechanism in JFreeChart only handles a small subset of the properties that can be set for charts. Extend (or reimplement) this mechanism to provide greater end-user control over the appearance of the charts.

## 4. SYSTEM DESIGN

### 4.1 SYSTEM ARCHITECTURE

A system architecture or systems architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures of the system.

A system architecture can comprise system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them. It can provide a plan from which products can be procured, and systems developed, that will work together to implement the overall system. There have been efforts to formalize languages to describe system architecture, collectively these are called architecture description languages (ADLs).

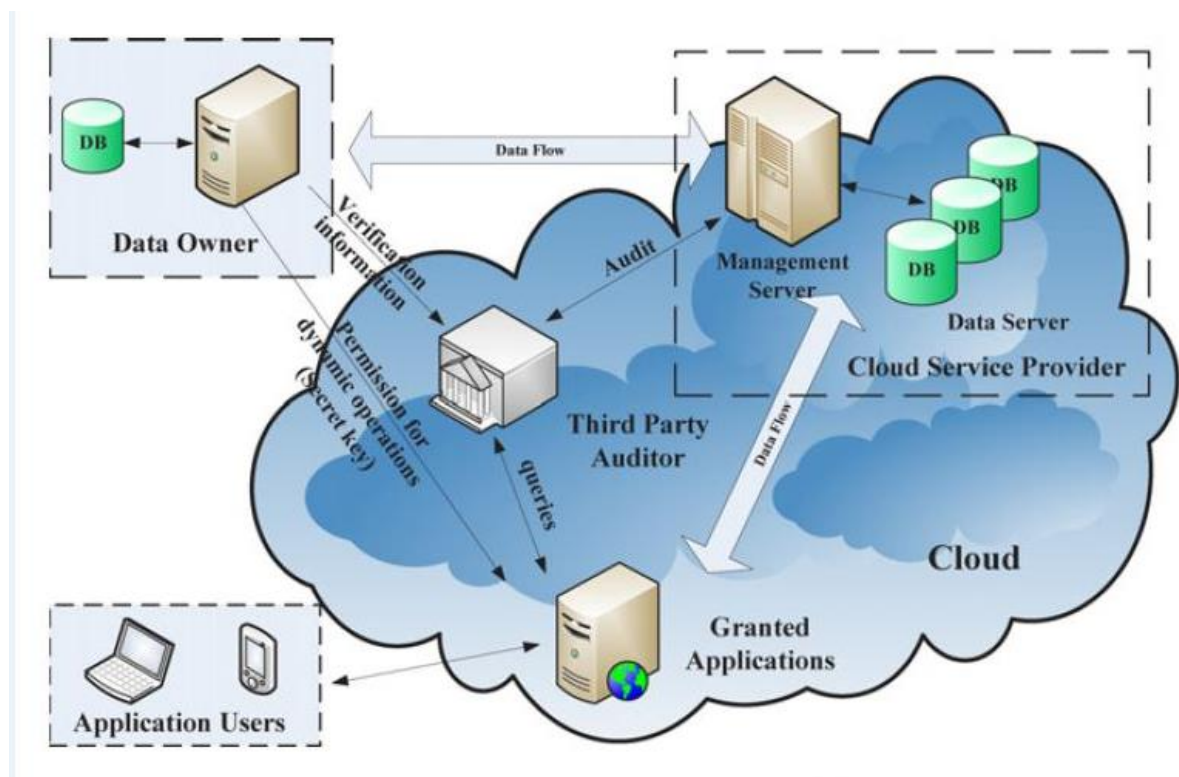


Figure 4.1: System Architecture



## **4.2 INPUT DESIGN**

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy.

### **4.2.1 OBJECTIVES OF INPUT DESIGN**

- a) Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
- b) It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
- c) When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

## **4.3 OUTPUT DESIGN**

- a) A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

- b) Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
- c) Select methods for presenting information.
- d) Create document, report, or other formats that contain information produced by the system.

#### **4.3.1 OBJECTIVES OF OUTPUT DESIGN**

- a) Convey information about past activities, current status or projections of the future.
- b) Signal important events, opportunities, problems, or warnings.
- c) Trigger an action.
- d) Confirm an action.

### **4.4 UML DIAGRAMS**

The Unified Modeling Language (UML) is a standard visual modeling language intended to be used for modeling business and similar processes and to analysis, design, and implement a software-based systems.

Unified Modeling Language (UML) combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies.

The Unified Modeling Language (UML) offers a standard way to visualize a system's architectural blueprints, including elements such as:

- a) Activities
- b) Actors
- c) Business processes
- d) Database schemas

- e) (Logical) components
- f) Programming language statements
- g) Reusable software components

UML is a common language for business analysts, software architects and developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems.

UML can be applied to diverse application domains (e.g., banking, finance, internet, etc.) It can be used with all major object and component software development methods and for various implementation platforms (e.g., J2EE,). The various UML diagrams are listed below.

UML is intentionally process independent and could be applied in the context of different processes. Still, it is most suitable for use case driven, iterative and incremental development processes. An example of such process is Rational Unified Process (RUP). UML is not complete and it is not completely visual. Given some UML diagram, we can't be sure to understand depicted part or behavior of the system from the diagram alone

For example, semantics of multiplicity of actors and multiplicity of use cases on use case diagrams is not defined precisely in the UML specification and could mean either concurrent or successive usage of use cases.

The Unified Modeling Language - UML - is OMG's most-used specification, and the way the world models not only application structure, behavior, and architecture, but also business process and data structure.

UML, along with the Meta Object Facility (MOF), also provides a key foundation for OMG's Model-Driven Architecture, which unifies every step of development and integration from business modeling, through architectural and application modeling, to development, deployment, maintenance, and evolution.

#### 4.4.1 USECASE DIAGRAM

A Use Case Model describes the proposed functionality of a new system. A Use Case represents a discrete unit of interaction between a user (human or machine) and the system. This interaction is a single unit of meaningful work, such as Create Account or View Account Details. Each Use Case describes the functionality to be built in the proposed system, which can include another Use Case's functionality or extend another Use Case with its own behavior.

The USECASE diagram of the Administrator in the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTICLOUD STORAGE system is given as

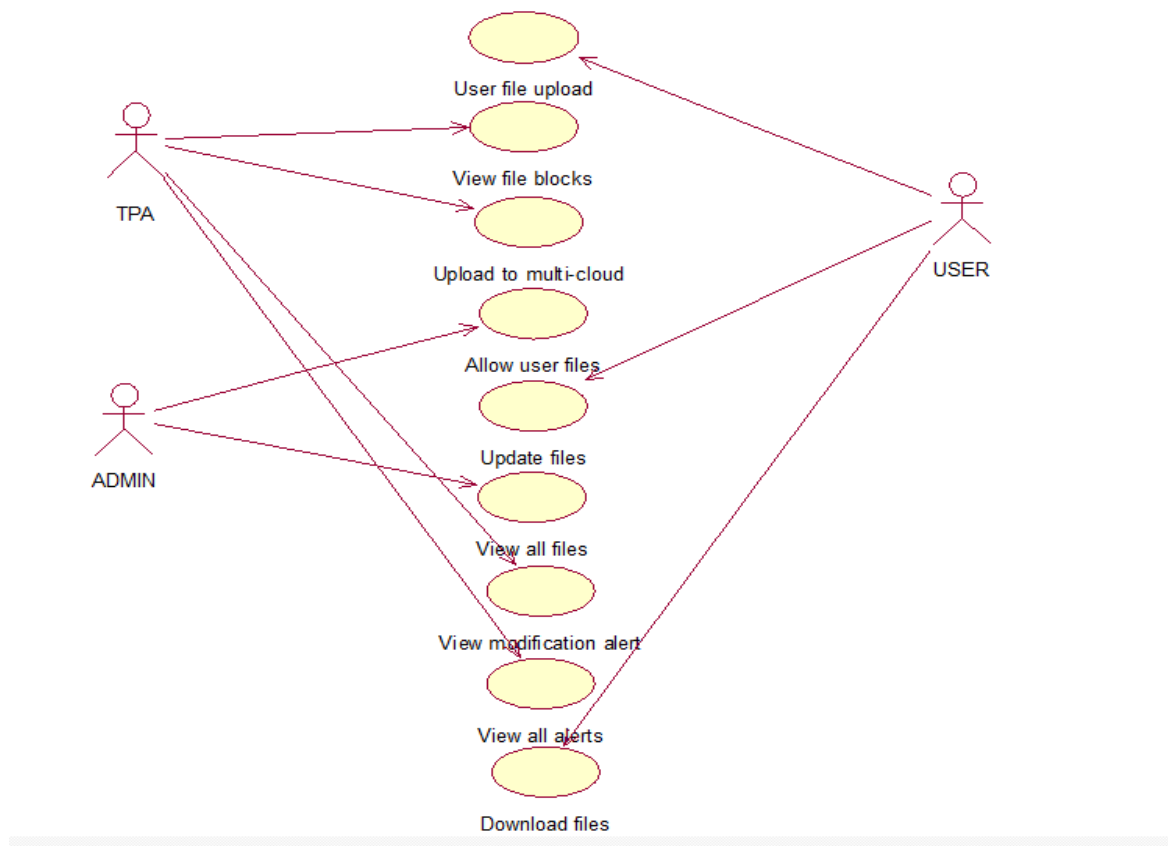


Figure 4.2: Usecase Diagram

#### 4.4.2 SEQUENCE DIAGRAM

Sequence diagrams provide a graphical representation of object interactions over time. These typically show a user or actor, and the objects and components they interact with in the execution of a use case. One sequence diagram typically represents a single Use Case 'scenario' or flow of events. Sequence diagrams are an excellent way of documenting usage scenarios and both capturing required objects early in analysis and verifying object use later in design. The diagrams show the flow of messages from one object to another, and as such correspond to the methods and events supported by a class/object.

The SEQUENCE diagram of the Administrator in the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system is given as

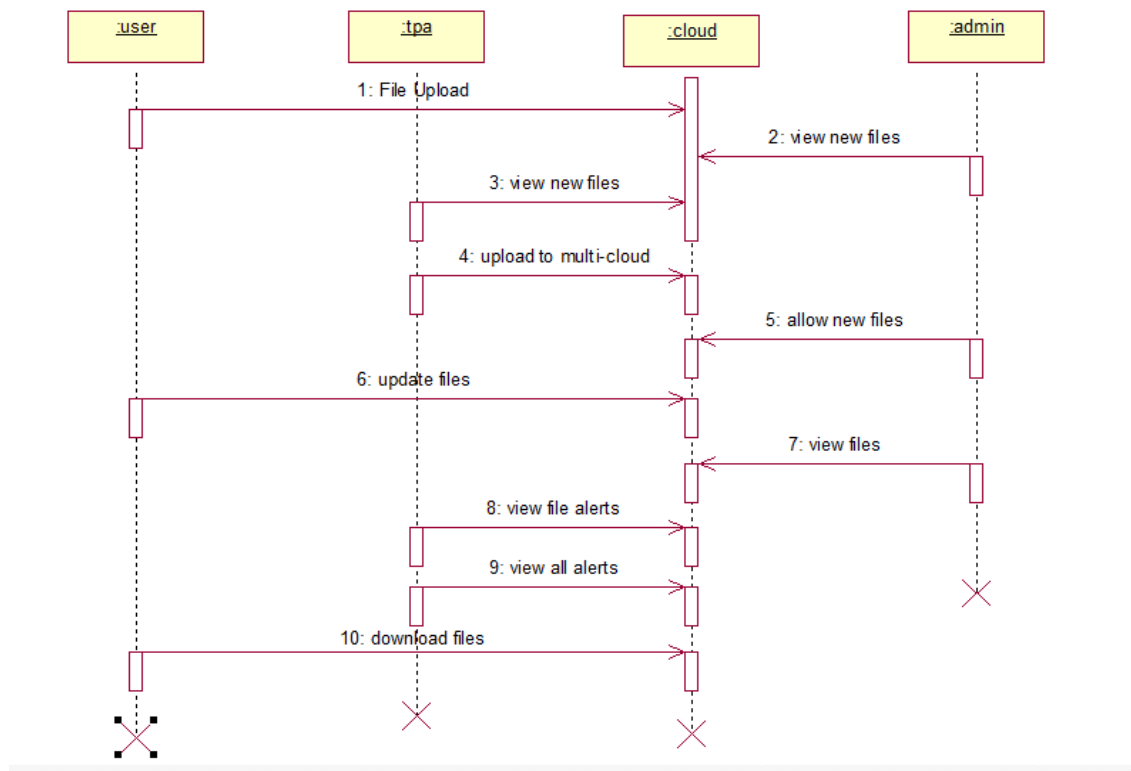


Figure 4.3: Sequence Diagram

#### 4.4.3 CLASS DIAGRAM

The class diagram is a static diagram. It represents the static view of an application. The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram. The purpose of the class diagram is to Analyze and design the static view of an application, Describe responsibilities of a system; provide base for component and deployment diagrams, forward and reverse engineering.

The CLASS diagram of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULIT-CLOUD STORAGE system is given as

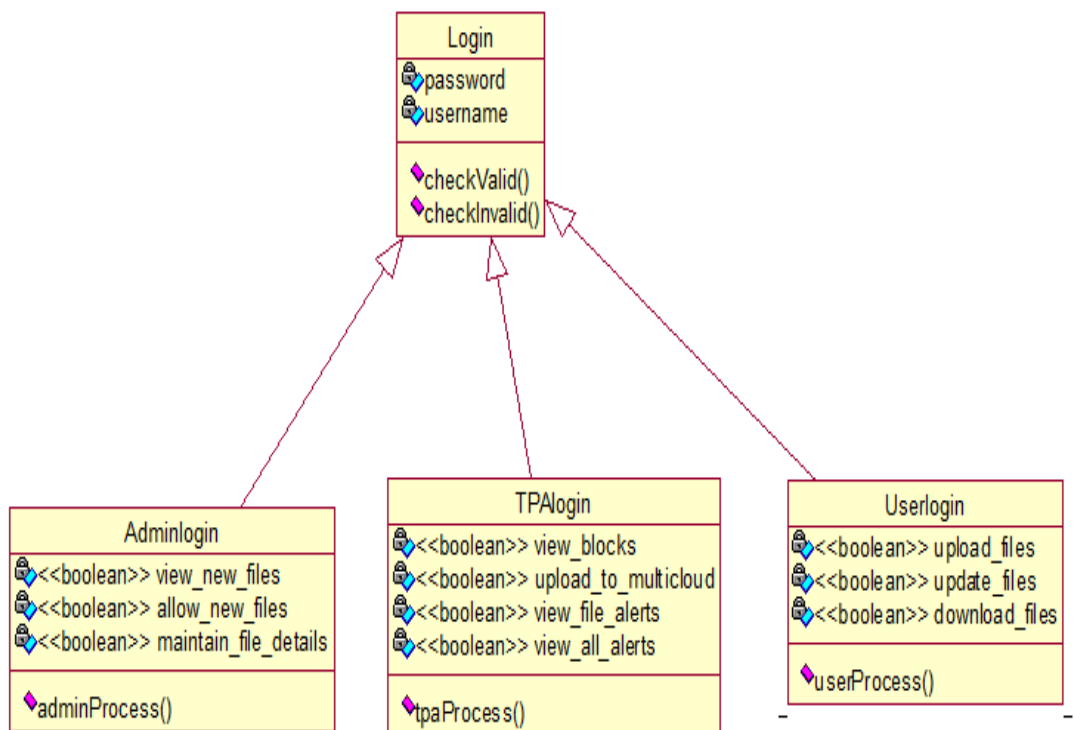


Figure 4.4: Class Diagram

#### 4.4.4 ACTIVITY DIAGRAM

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deal with all type of flow control by using different elements like fork, join etc.

The ACTIVITY diagram for the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system is given as

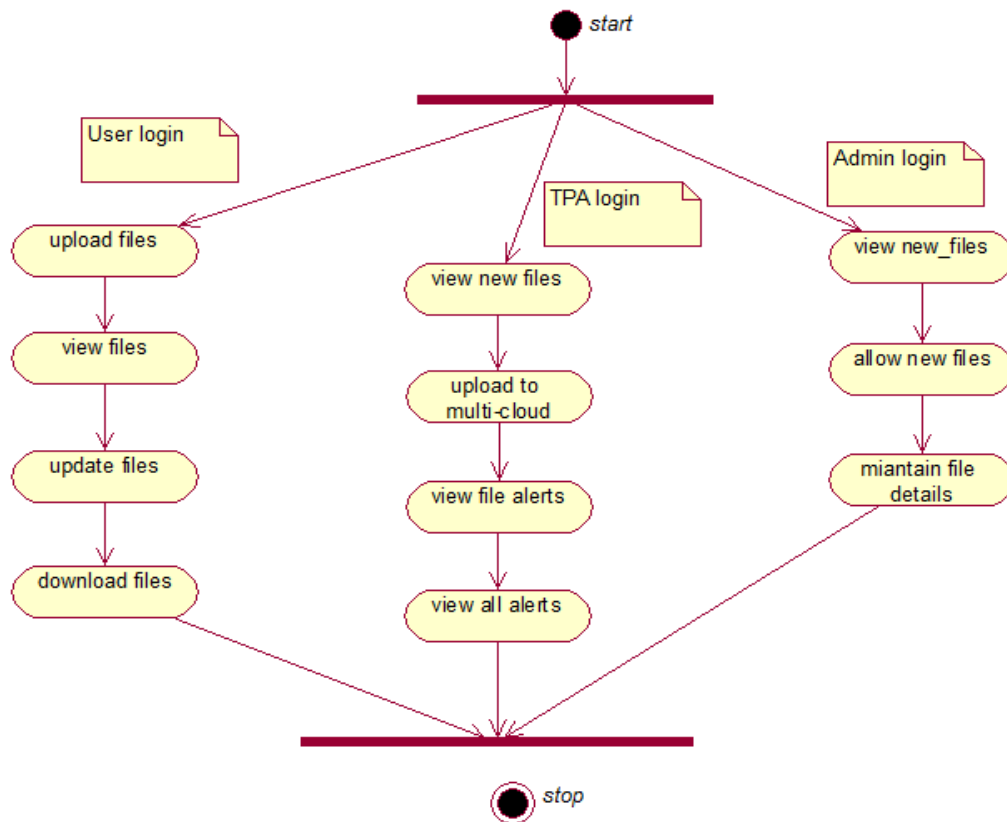


Figure 4.5: Activity Diagram

## **4.5 DATABASE DESIGN**

In the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system MySQL is used to store the data. MySQL is a database system used on the web. Basically, a MySQL database allows us to create a relational database structure on a web-server somewhere in order to store data or automate procedures.



## **5. SAMPLE CODE**

### **5.1 CODING**

#### **5.1.1 OBJECT ORIENTED PROGRAMMING AND JAVA**

Object-oriented Programming was developed because of limitations found in earlier approaches of programming. To appreciate what OOP does, we need to understand what these limitations are and how they arose from traditional programming.

#### **5.1.2 THE OBJECT ORIENTED APPROACH**

The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object.

An object's functions, called member methods in Java, typically provide the only way to access its data. If you want to read the item and return the value to you, you call a member function in the object. It will read the item and return the value to you. You can't access the data directly. The data is hidden, so it is safe from accidental modification. Data and its functions are said to be encapsulated into a single entity. Data encapsulation and data hiding are key terms in the description of object oriented languages. No other functions can access the data. This simplifies writing, debugging, and maintaining the program. A Java program typically consists of a number of objects, which communicate with each other by calling one another's members functions.

#### **5.1.3 CHARACTERISTICS OF OBJECT-ORIENTED LANGUAGES**

##### **5.1.3.1 OBJECTS**

When you approach a programming problem in an object oriented language, you no longer ask how the problem will be divided into functions, but how it will be divided into objects. Thinking in terms of objects, rather than functions, has a surprisingly helpful effect on how easily programs can be designed and objects in the real world.

An object is a location in memory having a value and referenced by an identifier. An object can be a variable, function, or data structure. In the object-oriented programming paradigm, "object," refers to a particular instance of a class where the object can be a combination of variables, functions, and data structures. In relational Database

management an object can be a table or column, or an association between data and a database entity (such as relating a person's age to a specific person).

### **5.1.3.2 ABSTRACTION**

An essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behavior. This abstraction allows people to use a car to drive to the grocery store without being overwhelmed by the complexity of the parts that form the car. They can ignore the details of how the engine, transmission, and braking systems work. Instead they are free to utilize the object as a whole.

Abstraction is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics. In object-oriented programming, abstraction is one of three central principles (along with encapsulation and inheritance). Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency. In the same way that abstraction sometimes works in art, the object that remains is a representation of the original, with unwanted detail omitted. The resulting object itself can be referred to as an abstraction, meaning a named entity made up of selected attributes and behavior specific to a particular usage of the originating entity. Abstraction is related to both encapsulation and data hiding.

In the process of abstraction, the programmer tries to ensure that the entity is named in a manner that will make sense and that it will have all the relevant aspects included and none of the extraneous ones. A real-world analogy of abstraction might work like this: You (the object) are arranging to meet a blind date and are deciding what to tell them so that they can recognize you in the restaurant. You decide to include the information about where you will be located, your height, hair color, and the color of your jacket. This is all data that will help the procedure (your date finding you) work smoothly. You should include all that information. On the other hand, there are a lot of bits of information about you that aren't relevant to this situation: your social security number, your admiration for obscure films, and what you took to "show and tell" in fifth grade are all irrelevant to this particular

situation because they won't help your date find you. However, since entities may have any number of abstractions, you may get to use them in another procedure in the future.

### **5.1.3.3 ENCAPSULATION**

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper. Access to the code and data inside the wrapper is tightly controlled through a well-defined interface. To relate this to the real world, consider the automatic transmission on an automobile. It encapsulates hundreds of bits of information about your engine, such as how much you are accelerating, the pitch of the surface you are on, and the position of the shift lever.

In general, encapsulation is the inclusion of one thing within another thing so that the included thing is not apparent. Decapsulation is the removal or the making apparent a thing previously encapsulated.

- a) In object-oriented programming, encapsulation is the inclusion within a program object of all the resources need for the object to function - basically, the methods and the data. The object is said to "publish its interfaces." Other objects adhere to these interfaces to use the object without having to be concerned with how the object accomplishes it. The idea is "don't tell me how you do it; just do it." An object can be thought of as a self-contained atom. The object interface consists of public methods and instantiated data.
- b) In telecommunication, encapsulation is the inclusion of one data structure within another structure so that the first data structure is hidden for the time being. For example, a TCP/IP-formatted data packet can be encapsulated within an ATM frame (another kind of transmitted data unit). Within the context of transmitting and receiving the ATM frame, the encapsulated packet is simply a stream of bits between the ATM data that describes the transfer.

### **5.1.3.4 INHERITANCE**

Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. As

mentioned earlier, most knowledge is made manageable by hierarchical (that is, top-down) classifications. Without the use of hierarchies, each object would need to define all of its Characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

In object-oriented programming (OOP), inheritance is when an object or class is based on another object or class, using the same implementation; it is a mechanism for code reuse. The relationships of objects or classes through inheritance give rise to a hierarchy. Inheritance was invented in 1967 for Simula.

Inheritance should not be confused with subtyping. In some languages inheritance and subtyping agree, while in others they differ; in general sub-typing establishes an is-a relationship, while inheritance only reuses implementation and establishes a syntactic relationship, not necessarily a semantic relationship (inheritance does not ensure behavioral subtyping). To distinguish these concepts, subtyping is also known as interface inheritance, while inheritance as defined here is known as implementation inheritance.

Inheritance is contrasted with object composition, where one object contains another object (or objects of one class contain objects of another class); see composition over inheritance. Composition implements a has-a relationship, in contrast to the is-a relationship of subtyping.

### **5.1.3.5 POLYMORPHISM**

Polymorphism (from the Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. Consider a stack (which is a last-in, first-out list). You might have a program that requires three types of stack. One stack is used for integer values, one for floating-point values, and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs. In a non-object-oriented language, you would be required to create three difference sets of stack routines, with each set using different names. However, because of polymorphism, in Java you can specify a general set of stack routines that all share the same names.

Generally, the ability to appear in many forms. In object-oriented programming, polymorphism refers to a programming language's ability to process objects differently depending on their data type or class. More specifically, it is the ability to redefine methods for derived classes. For example, given a base class shape, polymorphism enables the programmer to define different area methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the area method to it will return the correct results. Polymorphism is considered to be a requirement of any true object-oriented programming language (OOPL).

#### **5.1.4 SAMPLE CODE**

##### **databasecon.java**

```
packagedatabaseconnection;
importjava.sql.*;
public class databasecon
{
    static Connection con;
    public static Connection getconnection()
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            con=DriverManager.getConnection("jdbc:mysql://localhost:3306/multicloud","root","root"
            );//multi-cloud
        }
        catch(Exception e)
        {
            System.out.println("class error");
        }
        return con;
    }
}
```

### cloud1\_home.jsp

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
<title>Multi Cloud</title>
<style type="text/css">
.b:hover{
border-size:3px;
border-color:red;
}
.big:hover{
color:red;
font-weight:bold;
}
.b1{
background-color: #color;
border-bottom:solid;
border-left: #FFEEEE;
border-right:solid;
border-top: #EEEEEE;
color: brown;
font-family: Verdana, Arial
}
</style>
<meta name="keywords" content="" />
<meta name="description" content="" />
<link rel="stylesheet" type="text/css" href="default.css" />
</head>
<body>
<%
```









```

{
fid=rs.getString("fid");
uid=rs.getString("uid");
uname=rs.getString("name");
fname=rs.getString("fname");
fsize=rs.getString("fsize");
date=rs.getString("date");
%>
<tr bgcolor="#FFFFCC">
<td align="center"><strong><font color="#6300C6"><%=uid%></font></strong></td>
<td align="center"><strong><font
color="#6300C6"><%=uname%></font></strong></td>
<td align="center"><strong><font color="#6300C6"><%=date%></font></strong></td>
<td align="center"><strong><font color="#FF3300"><%=fid%></font></strong></td>
<td align="center"><strong><font color="#6300C6"><%=fname%></font></strong>
</td>
<td align="center"><strong><font color="#6300C6"><%=fsize%>&nbsp;&nbsp;&nbsp;<font
color="#333333">KB</font>
</font></strong></td>
<td align="center"><a href="cloud1_allow.jsp?<%=fid%>"><strong><font
color="#6300C6">allow</font></strong></a></td>
</tr>
<%
}
}
catch(Exception e1)
{
out.println(e1.getMessage());
}
%>

```

```

</table>
</form>
<br><br><br><br><br><br><br>
</div>
<!-- Primary content area end -->
</div>
<!-- Secondary content: Stuff that goes in the secondary content column (by default, the
narrower right column) -->
</div>
<div id="footer">
<!--<div                class="right">Design                by                <a
href="http://www.nodethirtythree.com/">NodeThirtyThree Design</a></div>-->
</div>
</div>
</body>
</html>

```

## **6. TESTING**

### **6.1 TESTING**

Once the source code has been generated, the software must be tested to uncover as many errors as possible before delivery to the customer. Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and code generation.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to the process of executing a program or application with the intent of finding software bugs (errors or other defects).

Software testing can be stated as the process of validating and verifying that a computer program/application/product:

- a) meets the requirements that guided its design and development,
- b) works as expected,
- c) can be implemented with the same characteristics,
- d) Satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the software development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the agile approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology.

There are two basics of software testing, WHITE BOX TESTING and BLACK BOX TESTING.

### 6.1.1 WHITE-BOX TESTING

This is performed knowing the internal workings of a product. Tests are conducted to ensure that “all gears mesh”, that is, that internal operation performs according to specification and all internal components have been adequately exercised. This can be done on close examination of procedural detail. Using the white-box testing we can derive test cases that,

- a) Guarantee that all independent paths within a module have been exercised at least once.
- b) Exercise all logical decisions on their true and false sides.
- c) Execute all loops at their boundaries and within their operational bounds and
- d) Exercise internal data structures to assure their validity.

White-box testing (also known as clear box testing, glass box testing, and transparent box testing and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements. Techniques used in white-box testing include:

- a) **API testing (application programming interface):** Testing of the application using public and private APIs.

- b) **Code coverage:** Creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once).
- c) **Fault injection methods:** Intentionally introducing faults to gauge the efficacy of testing strategies.
- d) **Mutation testing methods**
- e) **Static testing methods**

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for:

- a) Function coverage, which reports on functions executed.
- b) Statement coverage, which reports on the number of lines executed to complete the test.

100% statement coverage ensures that all code paths, or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

### **6.1.2 BLACK-BOX TESTING**

Knowing the specified function that a product has been designed to perform tests can be conducted that demonstrates each function is fully operational, at the same time searching for errors in each function. It enables us to derive sets of input conditions that will fully exercise all function requirements for documents. It attempts to find errors in,

- a) Incorrect or missing functions
- b) Interface errors
- c) Errors in data structures or external database access
- d) Performance errors
- e) Initialization and termination errors

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing.

Specification-based testing aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.

Specification-based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations.

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight." Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case, or leaves some parts of the program untested.

This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

## **6.2 TESTING LEVELS**

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

### **6.2.1 UNIT TESTING**

Unit testing focuses verification effort on the smallest unit of software design (i.e.), the module. Unit Testing exercise specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually; ensure that it functions properly as a unit. Hence, the name is unit testing.

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.



Unit testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it. Unit testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process. Depending on the organization's expectations for software development, unit testing might include static code analysis, data flow analysis metrics analysis, peer code reviews, code coverage analysis and other software verification practices.

### **6.2.2 INTEGRATION TESTING**

Integration Testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of High-order tests are conducted. The main objectives in this testing process are to take unit-tested modules and build a program structure that has been dictated by design. Top-down integration and bottom-up integration are the two different types of integration testing.

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

### **6.2.3 VALIDATION TESTING**

At the end of Integration Testing, software is completely assembled as a package, interfacing errors have been uncovered and correction testing begins.

Verification and validation testing are two important tests which are carried out on a software before it has been handed over to the customer. The aim of both verification and validation is to ensure that the product is made according to the requirements of the client, and does indeed fulfill the intended purpose. While verification is a quality control process, the quality assurance process carried out before the software is ready for release is known as validation testing. Its goal is to validate and be confident about the product or system, and that it fulfills the requirements given by the customer. The acceptance of the software from the end customer is also its part. Often, testing activities are introduced early in the software development life cycle.

### **6.2.4 SYSTEM TESTING**

System testing is series of different tests whose primary purpose is to fully exercise the computer based system. Although each test has a different purpose, all the work should verify that all system elements have been properly integrated and perform allocated functions.

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

System testing, or end-to-end testing, tests a completely integrated system to verify that it meets its requirements. For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

In addition, the software testing should ensure that the program, as well as working as expected, does not also destroy or partially corrupt its operating environment or cause other processes within that environment to become inoperative (this includes not corrupting shared memory, not consuming or locking up excessive resources and leaving any parallel processes unharmed by its presence).

### **6.2.5 FUNCTIONAL TESTING**

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- a) Valid Input: Identified classes of valid input must be accepted.
- b) Invalid Input: Identified classes of invalid input must be rejected.
- c) Functions: Identified functions must be exercised.
- d) Output: Identified classes of application outputs must be exercised.
- e) Systems/Procedures: Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work."

### **6.2.6 PERFORMANCE TESTING**

Performance testing is the testing to assess the speed and effectiveness of the system and to make sure it is generating results within a specified time as in performance requirements. It falls under the class of black box testing.

Performance testing is generally executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Load testing is primarily concerned with testing that the system can continue to operate under a specific load, whether that be large quantities of data or a large number of users. This is generally referred to as software scalability. The related load testing activity of when performed as a non-functional activity is often referred to as endurance testing. Volume testing is a way to test software functions even when certain components (for example a file or database) increase radically in size. Stress testing is a way to test reliability under unexpected or rare workloads. Stability testing (often referred to as load or endurance testing) checks to see if the software can continuously function well in or above an acceptable period.

There is little agreement on what the specific goals of performance testing are. The terms load testing, performance testing, scalability testing, and volume testing, are often used interchangeably.

Real-time software systems have strict timing constraints. To test if timing constraints are met, real-time testing is used.

#### **6.2.7 REGRESSION TESTING**

Regression testing is the testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. It falls under the class of black box testing.

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, as degraded or lost features, including old bugs that have come back. Such regressions occur whenever software functionality that was previously working, correctly, stops working as intended. Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code. Common methods of regression testing include re-running previous sets of test-cases and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features. They can either be complete, for changes added late in the release or deemed to be risky, or be very shallow, consisting of positive tests on each feature, if the changes are early in the release or deemed to be of low

risk. Regression testing is typically the largest test effort in commercial software development, due to checking numerous details in prior software features, and even new software can be developed while using some old test-cases to test parts of the new design to ensure prior functionality is still supported.

#### **6.2.8 ALPHA TESTING**

Alpha testing is a preliminary software field test carried out by a team of users in order to find bugs that were not found previously through other tests. The main purpose of alpha testing is to refine the software product by finding (and fixing) the bugs that were not discovered through previous tests.

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

#### **6.2.9 BETA TESTING**

Beta testing is the testing which is done by end users, a team outside development, or publicly releasing full pre-version of the product which is known as beta version. The aim of beta testing is to cover unexpected errors. It falls under the class of black box testing.

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

### **6.3 TEST CASES**

Test cases are specific executable tests that examine all aspects including inputs and outputs of a system and then provide a detailed description of the steps that should be taken, the results that should be achieved, and other elements that should be identified. Steps explained in a test case include all details even if they are assumed to be common knowledge. Test cases are used as a technical explanation and reference guide for systems.

A test case, in software engineering, is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do. The mechanism for determining whether a software program or system has passed or failed such a test is known as a test oracle. In some settings, an oracle could be a requirement or use case, while in others it could be a heuristic. It may take many test cases to determine that a software program or system is considered sufficiently scrutinized to be released. Test cases are often referred to as test scripts, particularly when written - when they are usually collected into test suites.

In order to fully test that all the requirements of an application are met, there must be at least two test cases for each requirement: one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases. Keeping track of the link between the requirement and the test is frequently done using a traceability matrix. Written test cases should include a description of the functionality to be tested, and the preparation required to ensure that the test can be conducted.

A formal written test-case is characterized by a known input and by an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

For applications or systems without formal requirements, test cases can be written based on the accepted normal operation of programs of a similar class. In some schools of testing, test cases are not written at all but the activities and results are reported after the tests have been run.

In scenario testing, hypothetical stories are used to help the tester think through a complex problem or system. These scenarios are usually not written down in any detail. They can be as simple as a diagram for a testing environment or they could be a description written in prose. The ideal scenario test is a story that is motivating, credible, complex, and easy to evaluate. They are usually different from test cases in that test cases are single steps while scenarios cover a number of steps of the key.

Given is a list of various test cases of the “COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE” system.

S.No.	Field	Input	Expected Result	Actual Result	Status Pass/Fail	Remarks
1	Name	NULL	Enter Name	Enter Name	Pass	-
2	Name	Ram	No Error	No Error	Pass	-
3	Email	NULL	Enter Email ID	Enter Email ID	Pass	-
4	Email	ram@gmail.com	No Error	No Error	Pass	-
5	Mobile	NULL	Enter valid Number	Enter valid Number	Pass	-
6	Mobile	Asdf	Enter valid Number	Enter valid Number	Pass	-
7	Mobile	9923667788	No Error	No Error	Pass	-

Table 6.1: Test Cases

## 7. OUTPUT SCREENS

### 7.1 HOME PAGE

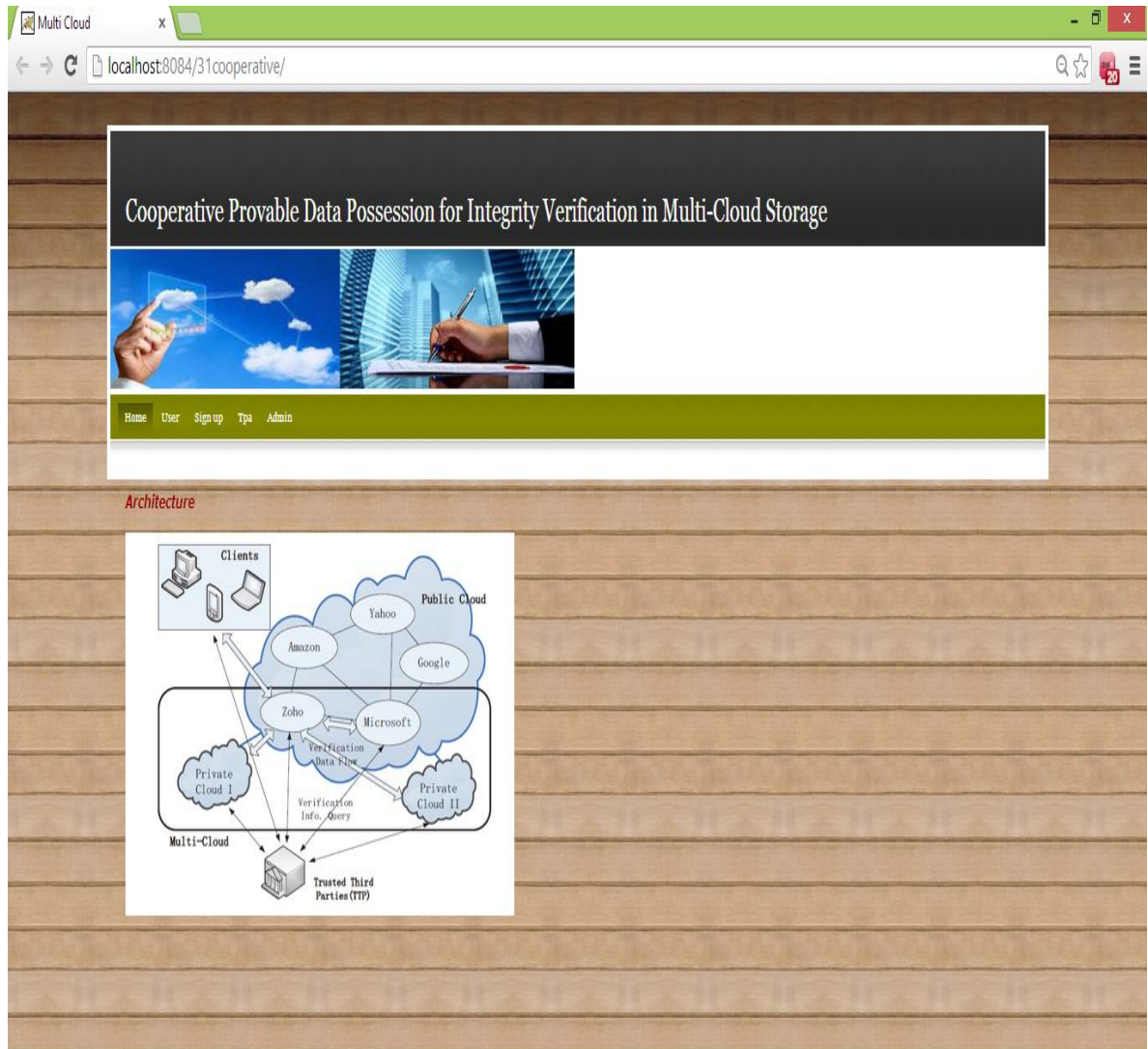


Figure 7.1: Home Page

This is the Home Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. The above section has got buttons to “Sign Up”, login as “User”, “Tpa” and “Admin”. The system architecture is represented in a diagrammatical way in the Home Page.



## 7.2 USER LOGIN PAGE

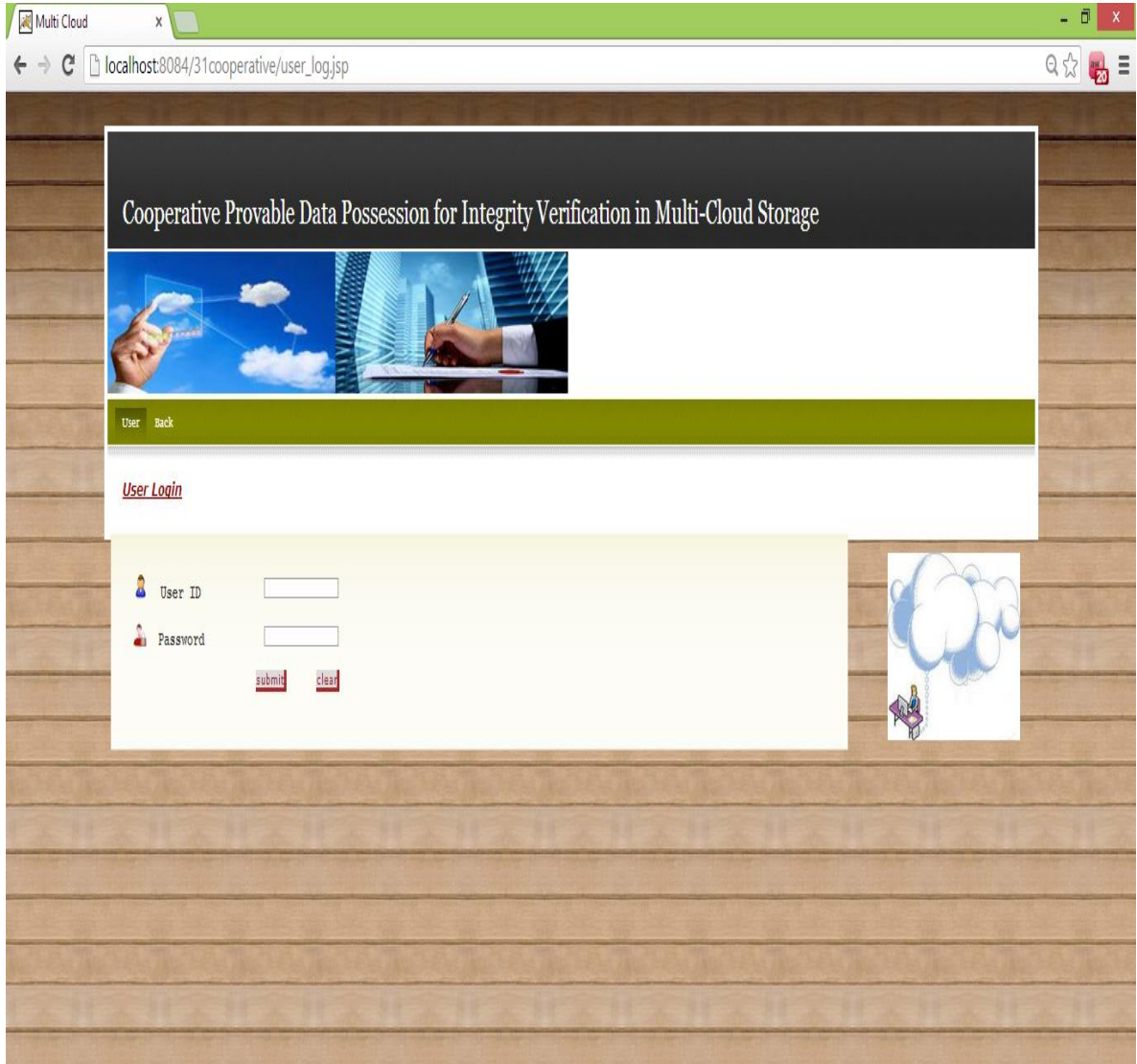


Figure 7.2: User Login Page

This is the User Login Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, the registered user should enter his/her “User ID” and “Password” and then click on “Submit” button. User can even clear his/her data by clicking the “Clear” button. To navigate back to the “Home Page”, the User has to click the “Back” button in the above section.

### 7.3 USER REGISTRATION PAGE

Multi Cloud x

localhost:8084/31cooperative/signup.jsp

Cooperative Provable Data Possession for Integrity Verification in Multi-Cloud Storage

Signup Back

[User Register](#)

ID 16

Name

User ID

Password

Mobile

Email ID

Date 17/09/2014

submit clear

Figure 7.3: User Registration Page

This is the User Registration Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. The User has to fill the form by entering all the values and then he/she should click the “Submit” button. He/she can even clear the entered data by clicking the “Clear” button. To navigate back to the Home Page the User can click the “Back” button in the above section.

## 7.4 TPA LOGIN PAGE

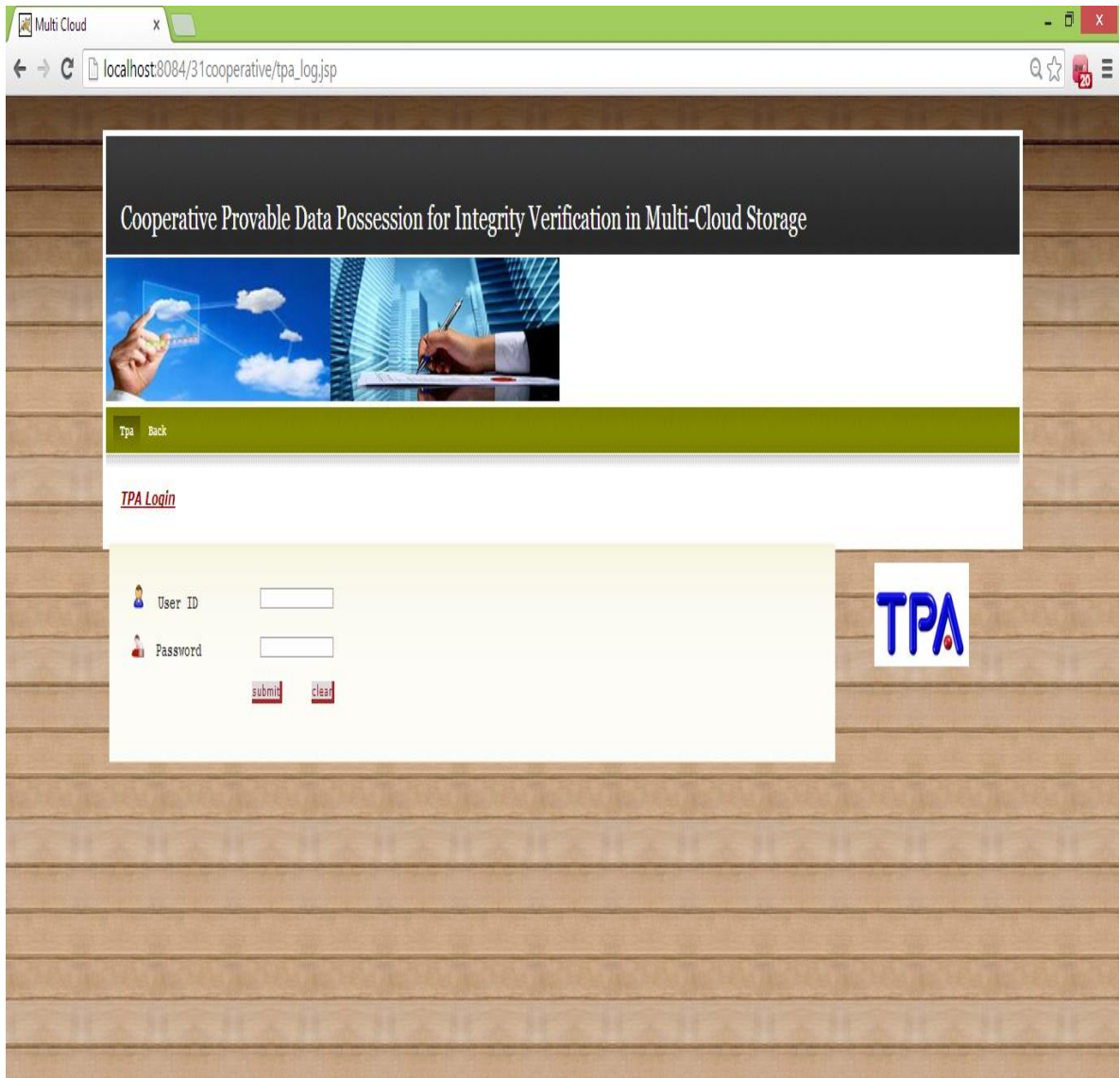


Figure 7.4: TPA Login Page

This is the TPA Login Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, the TPA enters his “User ID”, “Password” and then on clicking the “Submit” button he can log into the system. He can clear his data by clicking the “Clear” button. To navigate back to the Home Page, the User can click the “Back” button in the above section.

## 7.5 ADMIN LOGIN PAGE

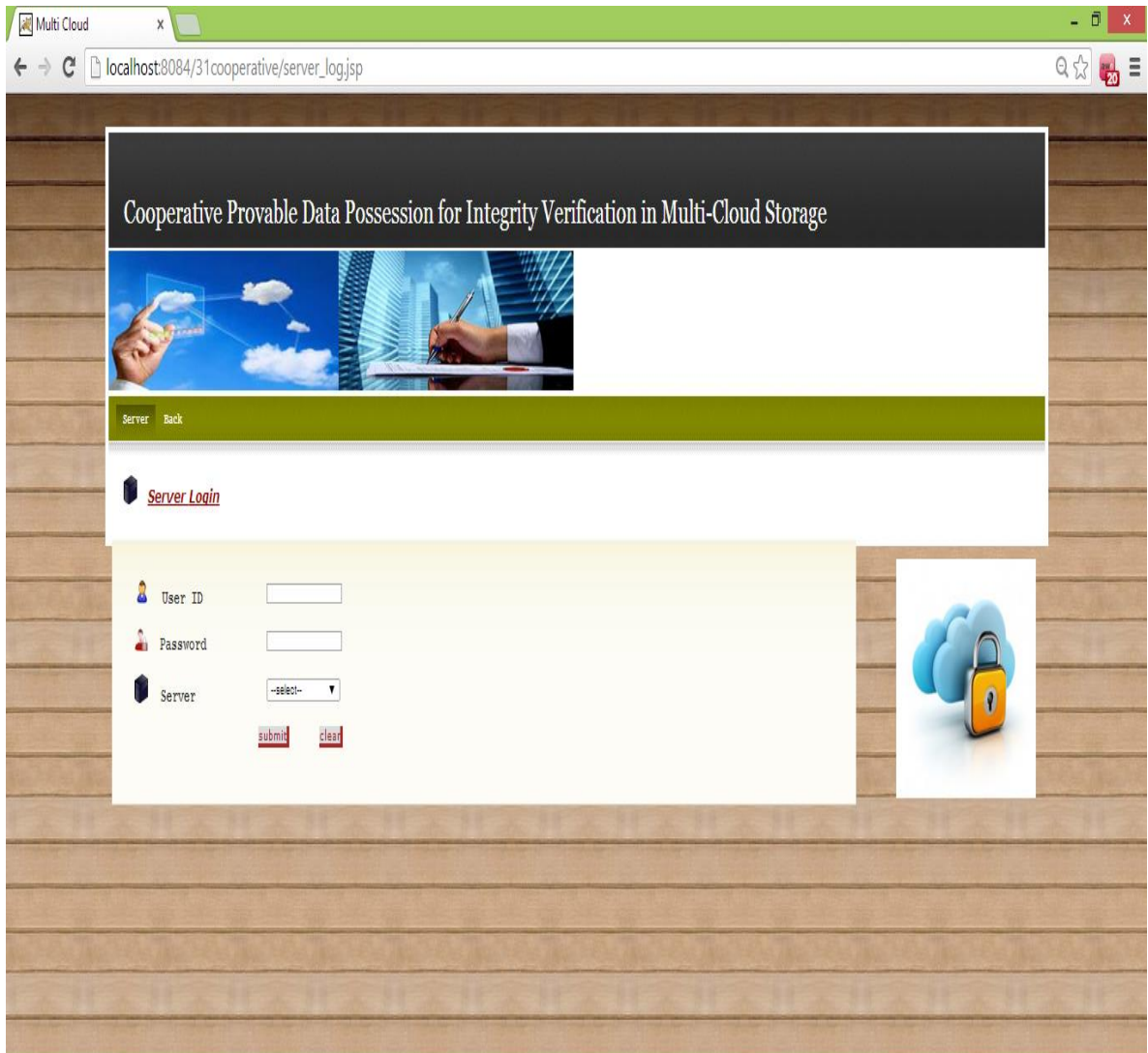


Figure 7.5: Admin Login Page

This is the Admin/Server Login Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, the Admin enters his “User ID”, “Password”, selects the “Server” and then clicks the “Submit” button. He/she can even clear the data by clicking the “Clear” button. To navigate back to the Home Page the Admin has to click the “Back” button in the above section.



## 7.6 USER HOME PAGE

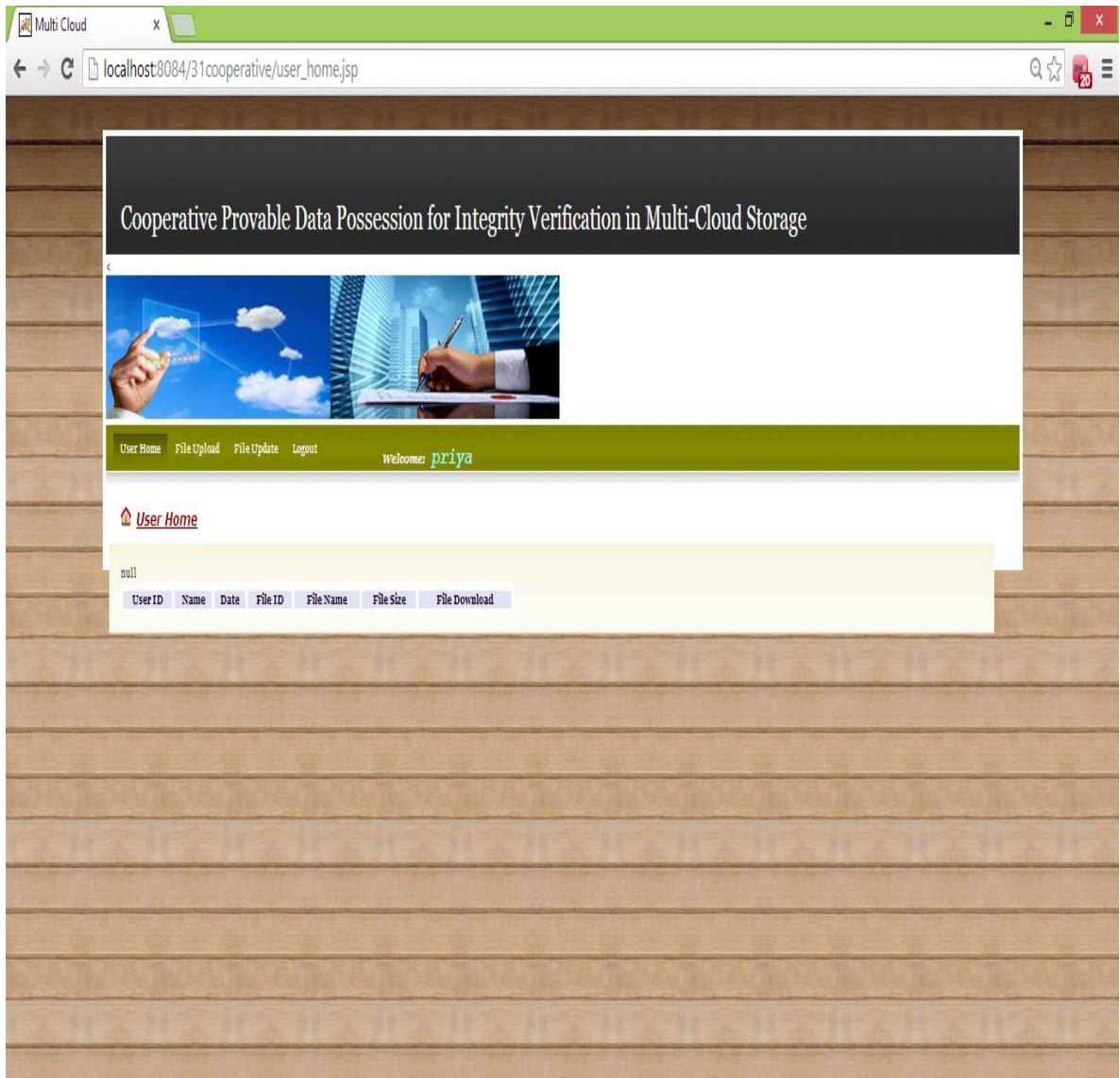


Figure 7.6: User Home page

This is the User Home Page of the COOPEARTIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, the User can view his/her details. He can upload a file by clicking the “File Upload” button, he can update a file by clicking the “File Update” button and he can even logout from the system by clicking the “Logout” button in the above section.

## 7.7 FILE UPLOAD PAGE

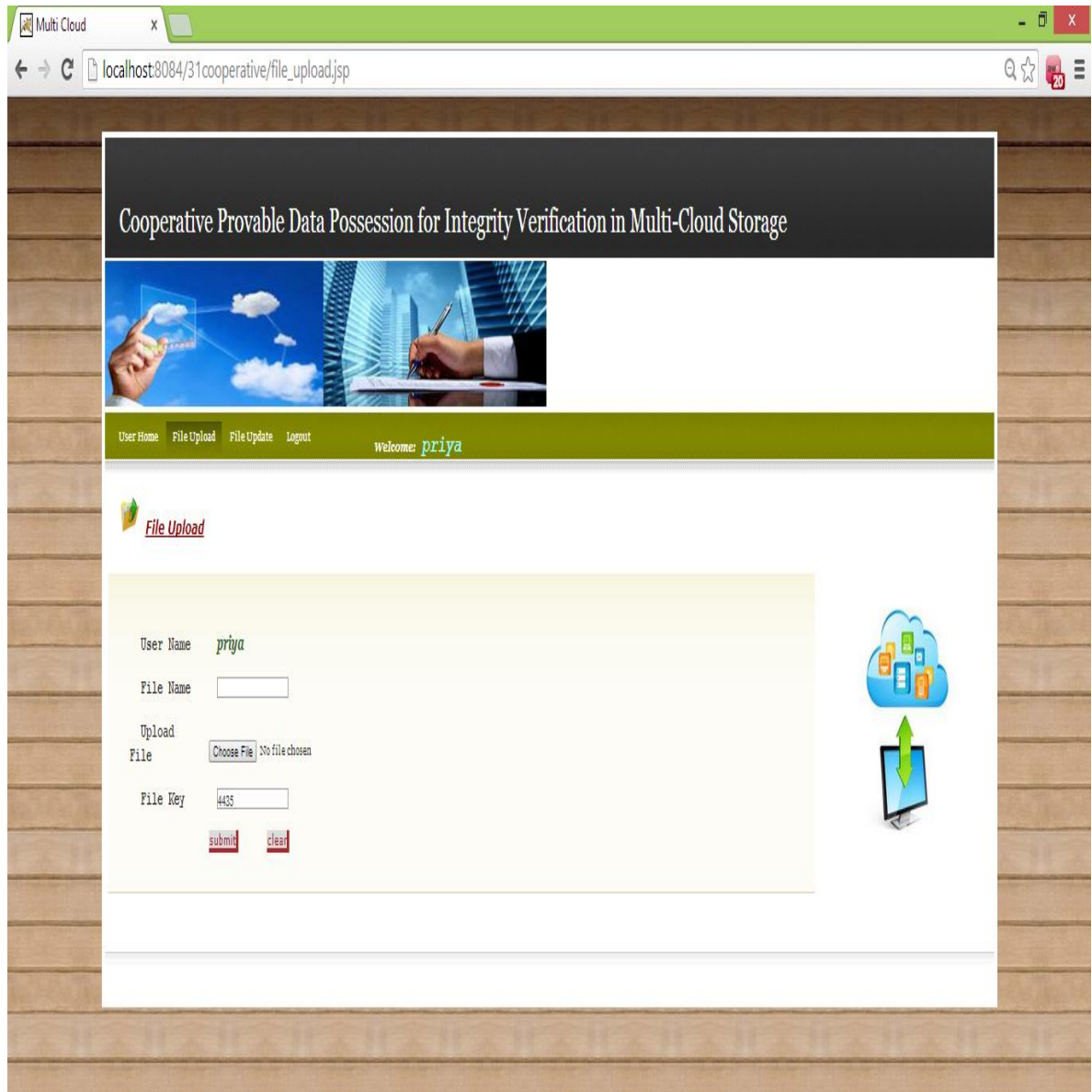


Figure 7.7: File Upload Page

This is the File Upload Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. The registered User can upload files on to the cloud by clicking “Choose File” button and then clicking the “Submit” button. The uploaded file is stored in the cloud.

## 7.8 FILE UPDATE PAGE

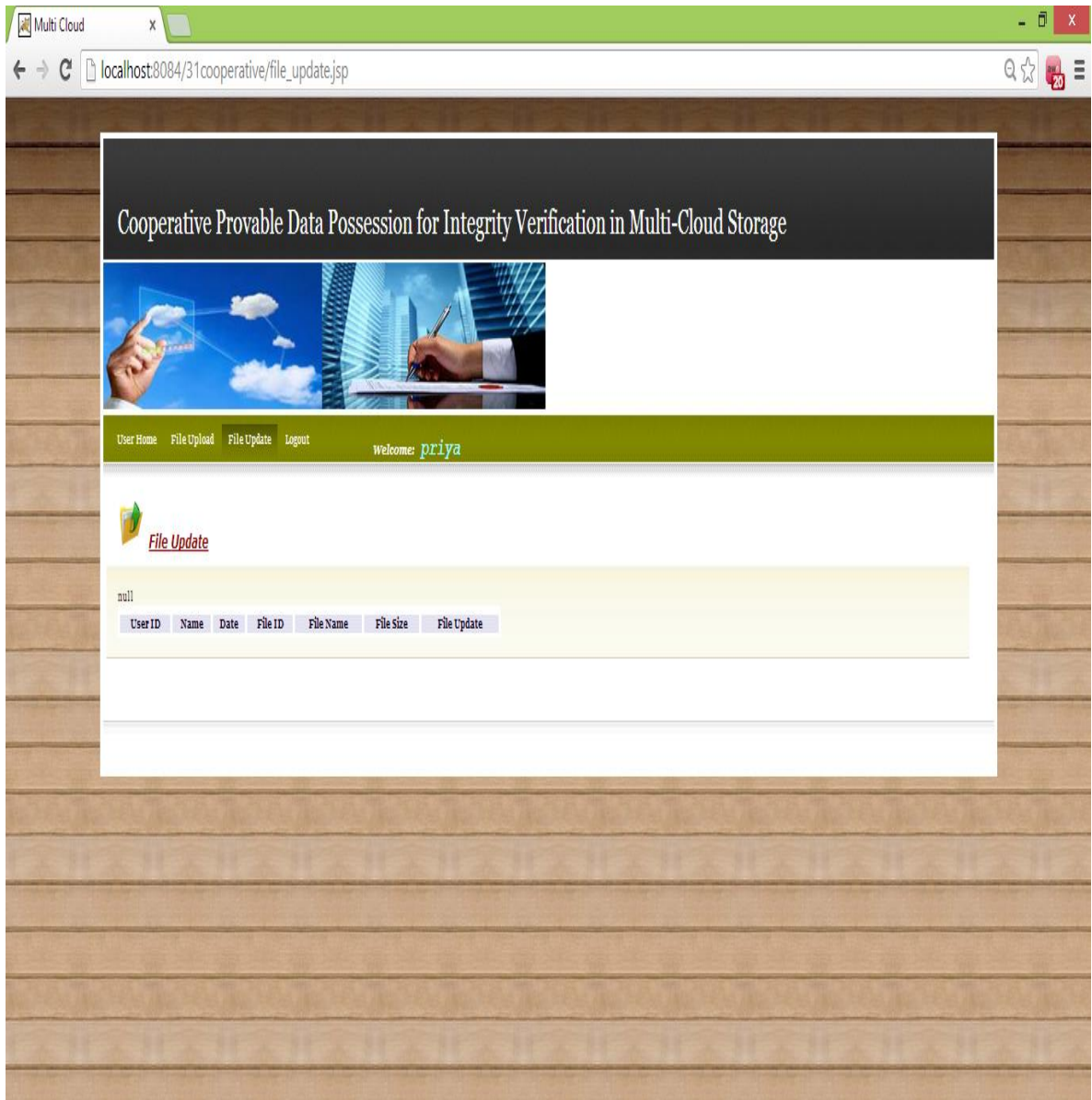


Figure 7.8: File Update Page

This is the File Update Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, the list of User files are displayed and the User can update the files he wishes to by selecting the file and clicking the “Update” link beside the selected file.

## 7.9 TPA HOME PAGE

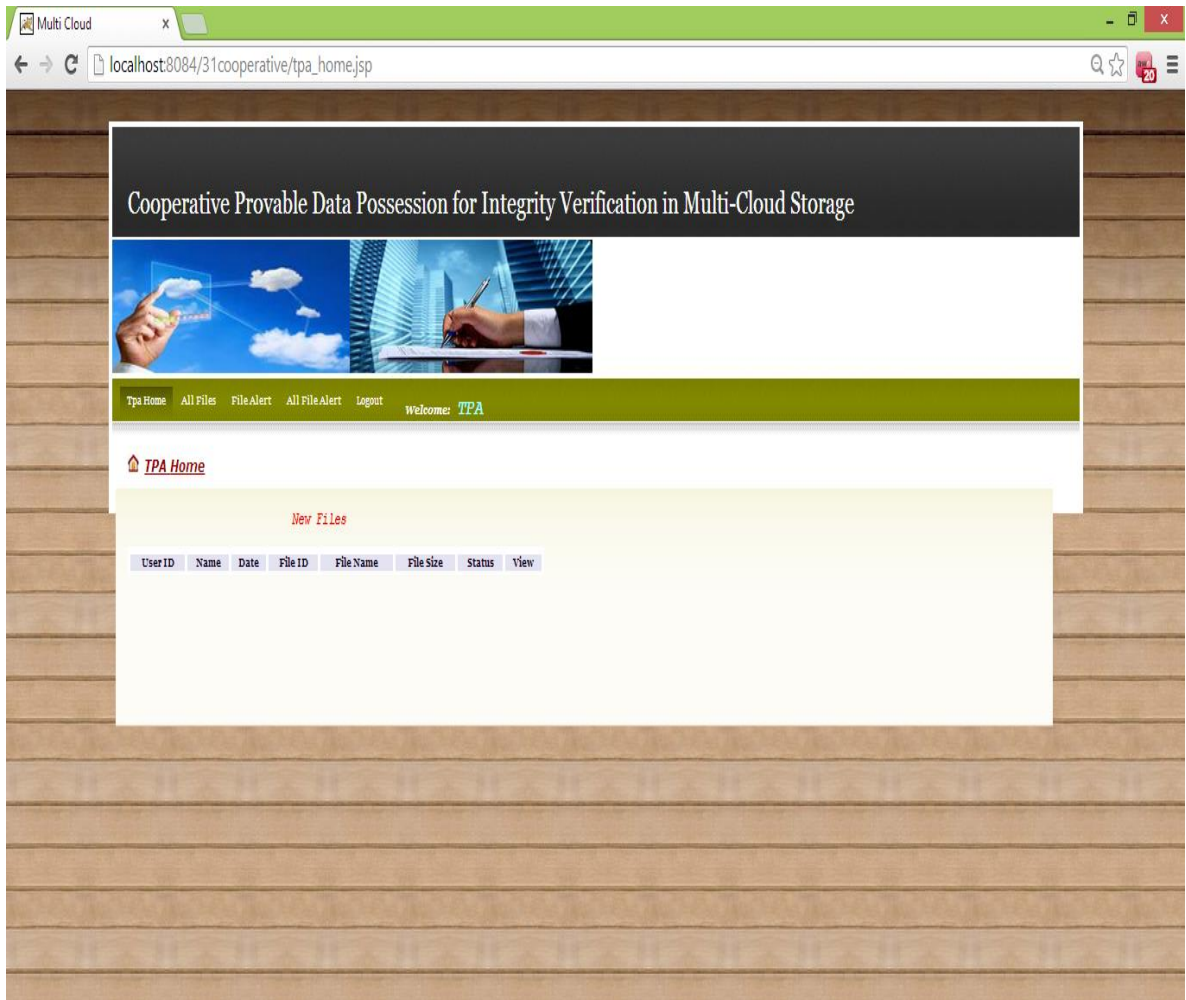


Figure 7.9: TPA Home Page

This is the TPA Home Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. This page is displayed when the TPA logs into the system. Here all the New Files are displayed to the TPA. TPA is the one who stores the verification parameters. He can view and upload the files into the cloud. The TPA can view the files, file alerts, and even log out of the system by clicking the “All Files”, “File Alert”, “All File Alerts” and “Logout” buttons respectively in the above section.



## 7.10 TPA ALL FILES PAGE

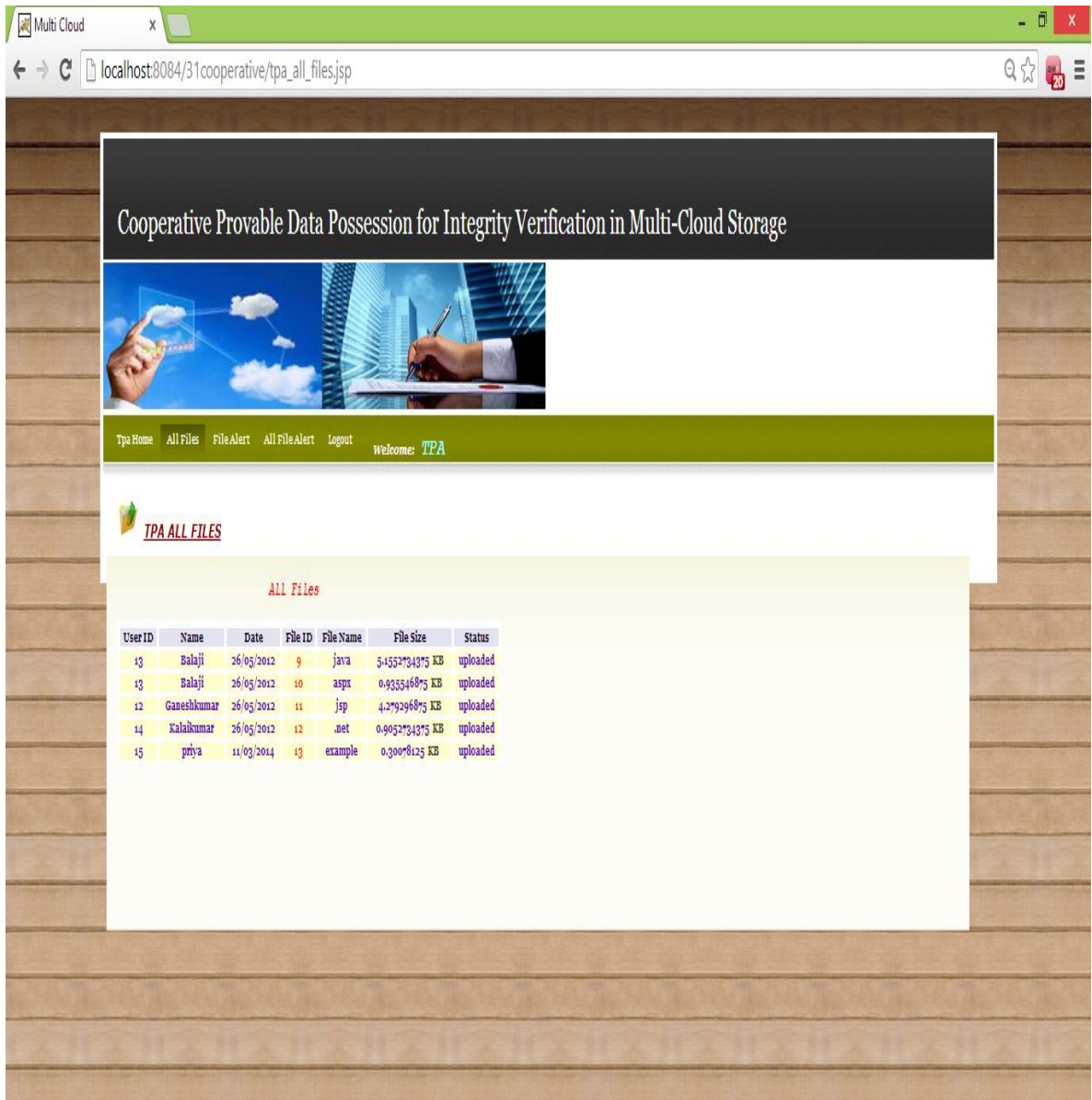


Figure 7.10: TPA All Files Page

This is the TPA All Files Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, all the files in the cloud are displayed to the TPA as the TPA has the permissions to view all the files and upload them to the cloud.

## 7.11 TPA FILE ALERT PAGE

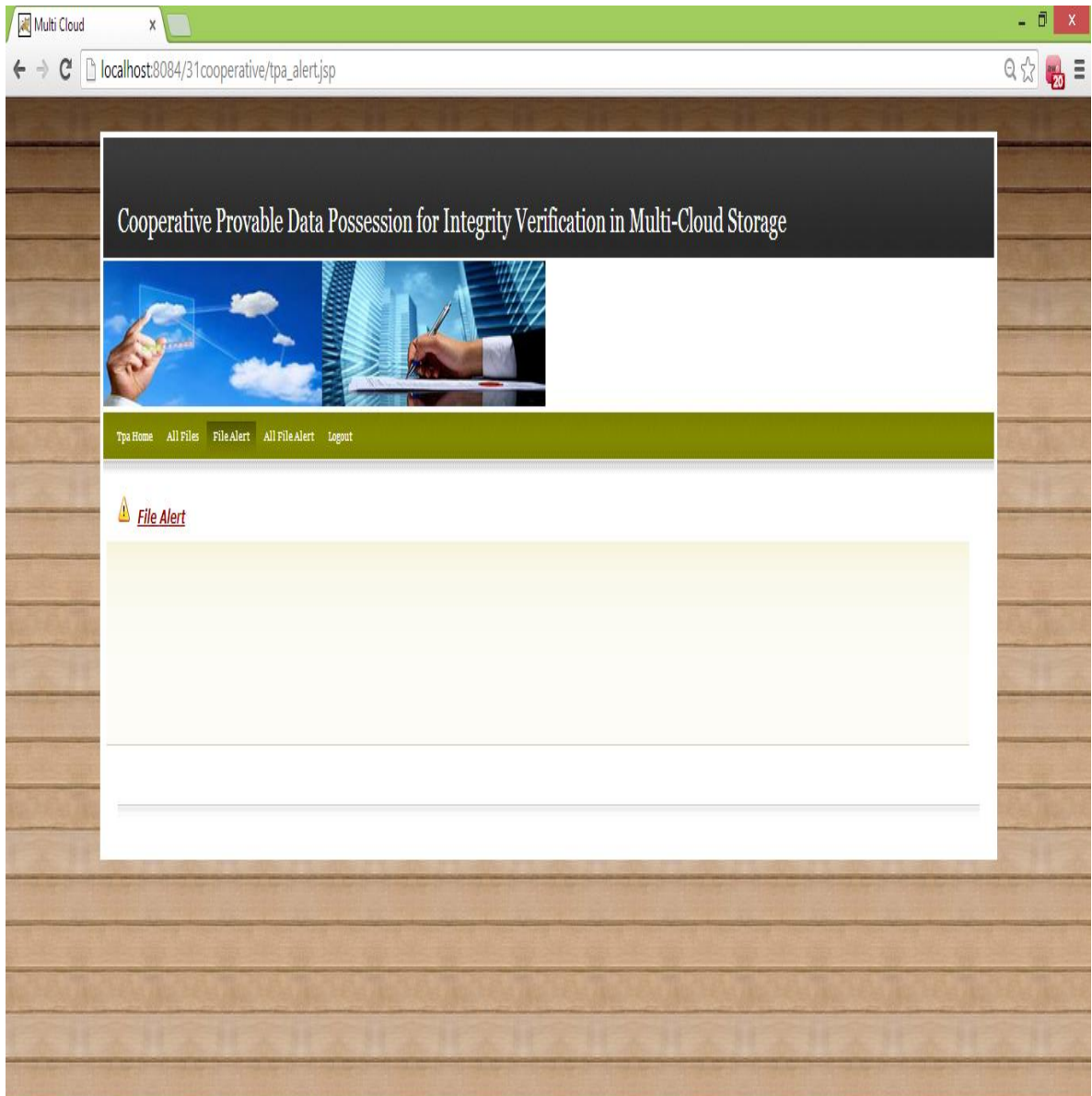
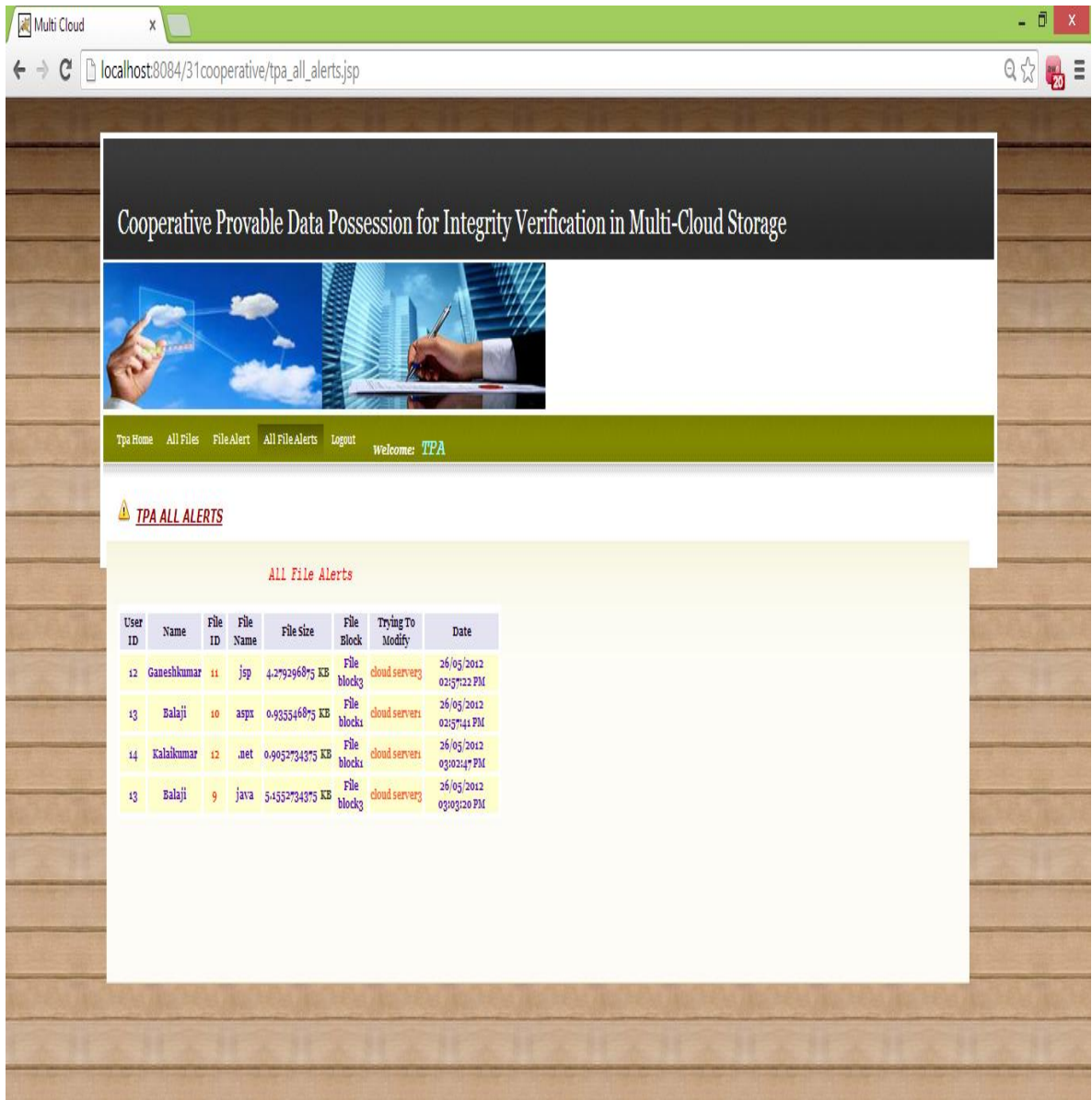


Figure 7.11: TPA File Alert Page

This is the TPA File Alert Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. The TPA receives File Alerts when any Cloud User tries to modify the data in the files. In this page the current File Alert will be displayed to the TPA.

## 7.12 TPA ALL FILE ALERTS PAGE



Cooperative Provable Data Possession for Integrity Verification in Multi-Cloud Storage

Tpa Home All Files File Alert All File Alerts Logout Welcome: TPA

**TPA ALL ALERTS**

All File Alerts

User ID	Name	File ID	File Name	File Size	File Block	Trying To Modify	Date
12	Ganeshkumar	11	jsp	4.279296875 KB	File block3	cloud server3	26/05/2012 02:57:22 PM
13	Balaji	10	aspx	0.933546875 KB	File block1	cloud server1	26/05/2012 02:57:41 PM
14	Kalaidkumar	12	.net	0.9952734375 KB	File block1	cloud server1	26/05/2012 03:02:14 PM
13	Balaji	9	java	5.4552734375 KB	File block3	cloud server3	26/05/2012 03:03:20 PM

Figure 7.12: TPA All File Alerts Page

This is the TPA All File Alerts Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, all the File Alerts that the TPA received will be displayed to the TPA with all the necessary particulars for each file individually.

## 7.13 ADMIN HOME PAGE

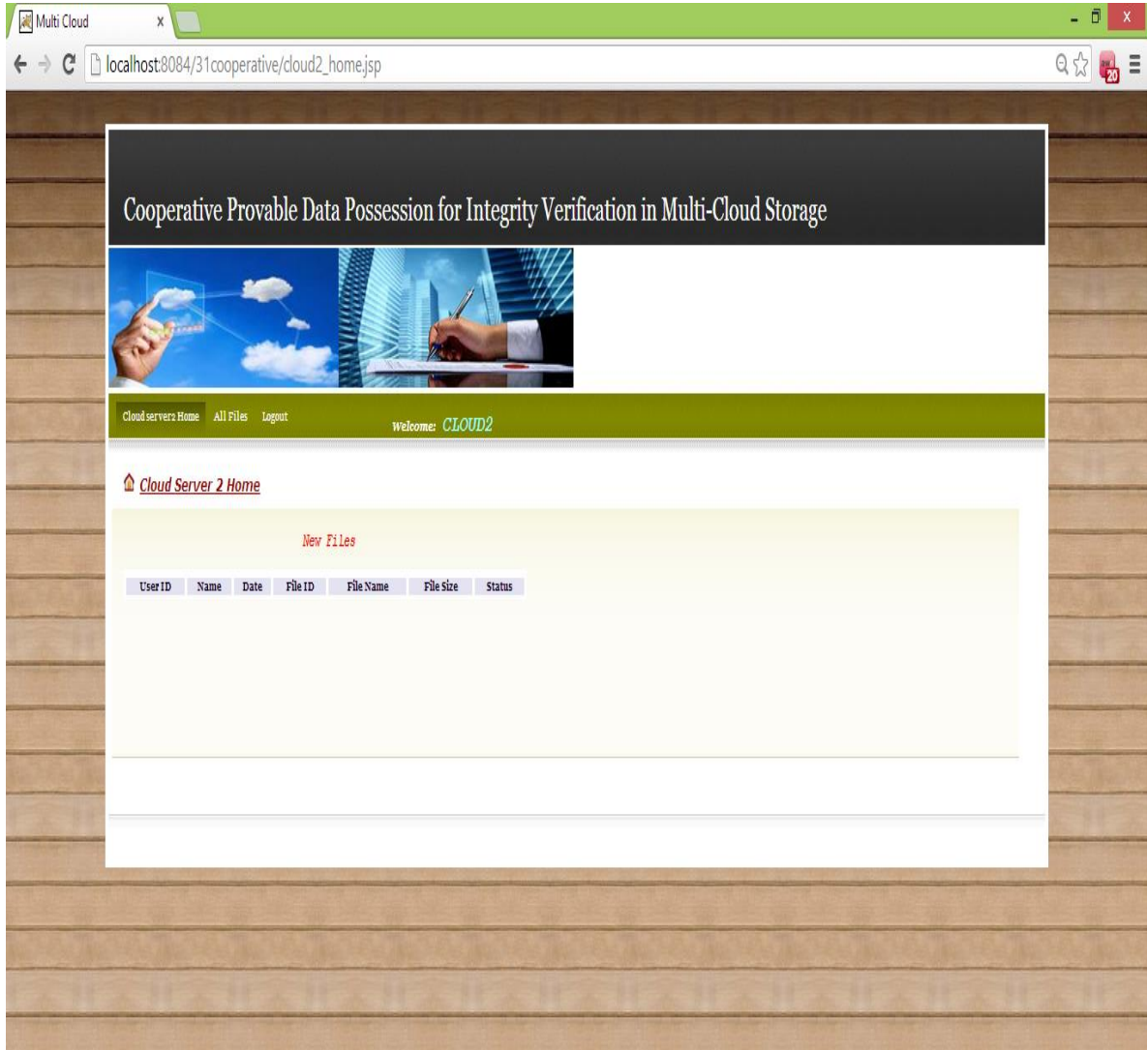


Figure 7.13: Admin Home Page

This is the Admin Home Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, all the New Files are displayed to the Admin in an order. The Admin can view, update and upload the files to the cloud. The admin can view all the files in the cloud and he can even logout from the system by clicking the “All Files” and “Logout” buttons respectively in the above section.



## 7.14 ADMIN ALL FILES PAGE

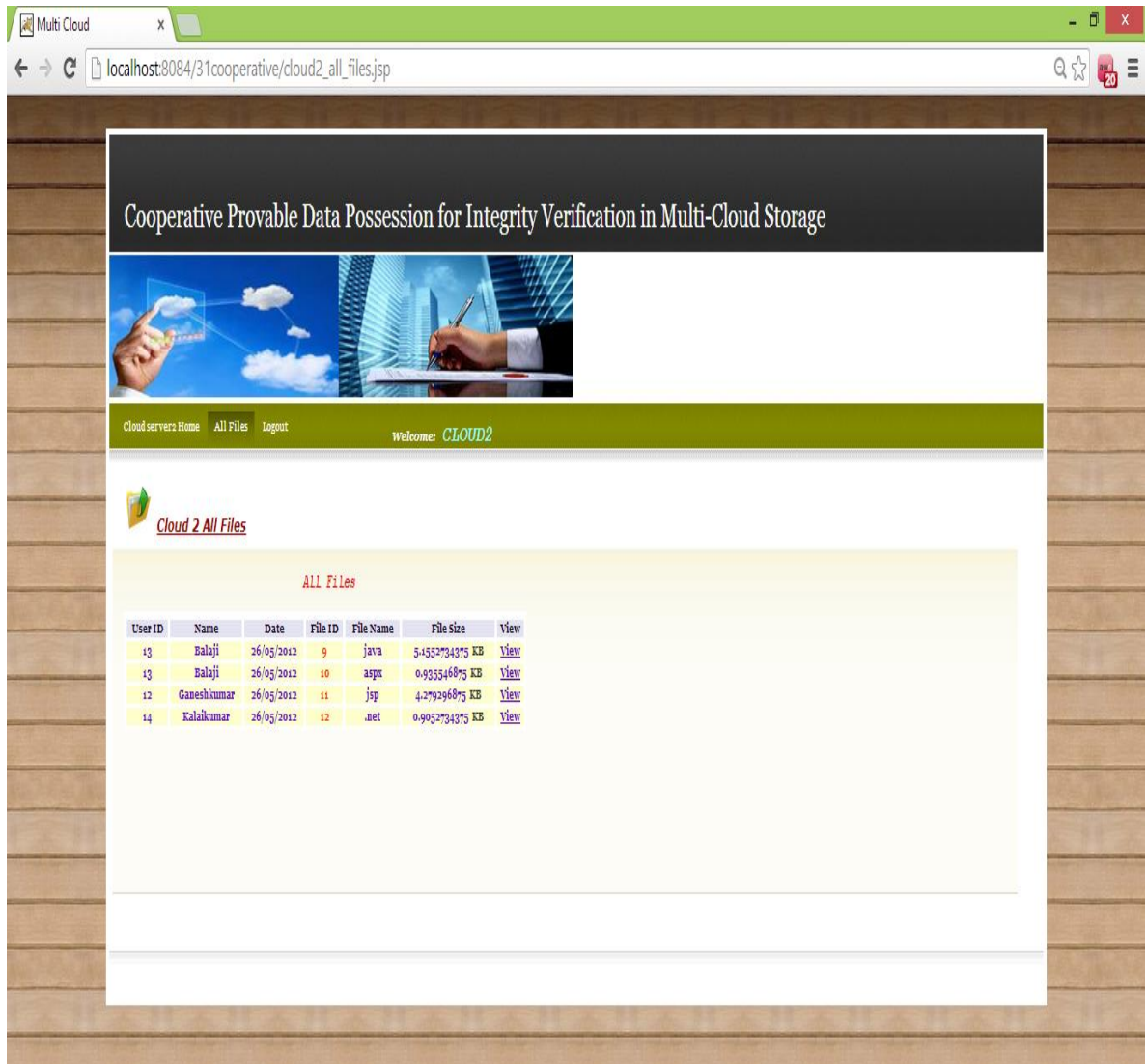


Figure 7.14: Admin All Files Page

This is the Admin All Files Page of the COOPERATIVE PROVABLE DATA POSSESSION FOR INTEGRITY VERIFICATION IN MULTI-CLOUD STORAGE system. Here, the admin can view all the files in the cloud. These files are displayed in order and the admin can view them by clicking the “View” link corresponding to each of them.

## **8. CONCLUSION**

### **8.1 CONCLUSION**

An efficient PDP scheme for distributed cloud storage has been constructed. Based on homomorphic verifiable response and hash index hierarchy, a cooperative PDP scheme to support dynamic scalability on multiple storage servers has been proposed. It has been proved that CPDP provides all security properties required by zero-knowledge interactive proof system, so that it can resist various attacks even if it is deployed as a public audit service in clouds. Furthermore, the probabilistic query and periodic verification are optimized to improve the audit performance. Experiments clearly demonstrated that CPDP introduced a small amount of computation and communication overheads. Therefore, CPDP can be treated as a new candidate for data integrity verification in outsourcing data storage systems.

### **8.2 FURTHER ENHANCEMENTS**

As part of future work, we can extend the project to explore more effective CPDP constructions. Finally, it is still a challenging problem for the generation of tags with the length irrelevant to the size of data blocks. We can explore this issue to provide the support of variable-length block verification.

## 9. BIBLIOGRAPHY

### 9.1 BOOKS REFERENCES

- a) B. Sotomayor, R. S. Montero, I. M. Llorente, and I. T. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.
- b) G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song, "Provable data possession at untrusted stores," in *ACM Conference on Computer and Communications Security*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds. ACM, 2007, pp. 598–609.
- c) A. Juels and B. S. K. Jr., "Pors: proofs of retrievability for large files," in *ACM Conference on Computer and Communications Security*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds. ACM, 2007, pp. 584–597.
- d) G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication networks*, SecureComm, 2008, pp. 1–10.
- e) C. C. Erway, A. K. Upc, "u, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 213–222.
- f) H. Shacham and B. Waters, "Compact proofs of retrievability," in *ASIACRYPT*, ser. Lecture Notes in Computer Science, J. Pieprzyk, Ed., vol. 5350. Springer, 2008, pp. 90–107.
- g) Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *ESORICS*, ser. Lecture Notes in Computer Science, M. Backes and P. Ning, Eds., vol. 5789. Springer, 2009, pp. 355–370.
- h) Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Dynamic audit services for integrity verification of outsourced storages in clouds," in *SAC*, W. C. Chu, W. E. Wong, M. J. Palakal, and C.-C. Hung, Eds. ACM, 2011, pp. 1550–1557.

- i) K. D. Bowers, A. Juels, and A. Oprea, “Hail: a high-availability and integrity layer for cloud storage,” in ACM Conference on Computer and Communications Security, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 187–198.
- j) Y. Dodis, S. P. Vadhan, and D. Wichs, “Proofs of retrievability via hardness amplification,” in TCC, ser. Lecture Notes in Computer Science, O. Reingold, Ed., vol. 5444. Springer, 2009, pp. 109–127.
- k) L. Fortnow, J. Rompel, and M. Sipser, “On the power of multiprover interactive protocols,” in Theoretical Computer Science, 1988, pp. 156–161.
- l) Y. Zhu, H. Hu, G.-J. Ahn, Y. Han, and S. Chen, “Collaborative integrity verification in hybrid clouds,” in IEEE Conference on the 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom, Orlando, Florida, USA, October 15-18, 2011, pp. 197–206.
- m) M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the clouds: A Berkeley view of cloud computing,” EECS Department, University of California, Berkeley, Tech. Rep., Feb 2009.
- n) D. Boneh and M. Franklin, “Identity-based encryption from the weil pairing,” in Advances in Cryptology (CRYPTO’2001), vol. 2139 of LNCS, 2001, pp. 213–229.
- o) O. Goldreich, Foundations of Cryptography: Basic Tools. Cambridge University Press, 2001.

## 9.2 WEBSITE REFERENCES

- a) <http://java.sun.com>
- b) <http://www.sourceforge.de>
- c) <http://www.networkcomputing.com/>
- d) <http://www.roseindia.com/>
- e) <http://www.java2s.com/>
- f) [www.avantetech.com](http://www.avantetech.com)
- g) [www.metrokc.gov](http://www.metrokc.gov)
- h) [www.premierelections.com](http://www.premierelections.com)



- i) [www.wikipedia.com](http://www.wikipedia.com)
- j) <http://www.freejavaguide.com/corejava.htm>
- k) <http://www.learnjavaonline.org/>

### **9.3 TECHNICAL PUBLICATION REFERENCES**

- a) Java, Object Oriented Analysis and Design, UML By Addison-Wesley Professional
- b) Writing “Real” Java-Not Just C in Java By Oracle
- c) Building Secure Software in Java By Aonix
- d) Real-Time Core Extensions for the Java Platform By NewMonics