CONTENTS

LIST OF DIAGRAMS

LIST OF TABLES

CHAPTER 1
INTRODUCTION

A clear recent trend in information technology is the rent by many users and enterprises of the storage/computation services from other parties. With cloud technology, what was in the past managed autonomously now sees the involvement of servers, often in an unknown location, immediately reachable wherever an Internet connection is present. Today the use of these Internet services typically assumes the presence of a Cloud Service Provider (CSP) managing the service. There are a number of factors that explain the current status. In general, the procurement and management of IT resources exhibit significant scale economies, and large scale CSPs can provide services at costs that are less than those incurred by smaller players. Still, many users have an excess of computational, storage, and network capacity in the systems they own and they would be interested in offering these resources to other users in exchange of a rent payment. In the classical behavior of markets, the existence of an infrastructure that supports the meeting of supply and demand for IT services would lead to a significant opportunity for the creation of economic value from the use of otherwise under utilized resources. This change of landscape is witnessed by the increasing attention of the research and development community toward the realization of Decentralized Cloud Storage (DCS) services, characterized by the availability of multiple nodes that can be used to store resources in a decentralized manner. In such services, individual resources are fragmented in shards allocated (with replication to provide availability guarantees) to different nodes. Access to a resource requires retrieving all its shards. The main characteristics of a DCS is the cooperative and dynamic structure formed by independent nodes (providing a multi-authority storage network) that can join the service and offer storage space, typically in exchange of some reward. This evolution has been facilitated by blockchain-based technologies providing an effective low-friction electronic payment system supporting the remuneration for the use of the service. On platforms such as Story [1], SAFE Network Vault [2], [3], IPFS [4], and Sea [5], users can rent out their unused storage and bandwidth to offer a service to other users of the network, who pay for this service with a network crypto-currency [6]. However, if security concerns and perception of (or actual) loss of control have been an issue and slowing factor for centralized clouds, they are even more so for a decentralized cloud storage, where the dynamic and independent nature of the network may hint to a further decrease of control of the owners on where and how their resources are managed. Indeed, in centralized cloud systems, the CSP is generally assumed to be honest-but-curious and is then trusted to perform all the operations requested by authorized users (e.g., delete a file when requested by the owner) [7]. The CSP is discouraged to behave maliciously, since this would clearly impact its reputation. On the contrary, the nodes of a decentralized system may behave maliciously when their misbehavior can provide economic benefits without impacting reputation (e.g., sell the content of deleted files).

Client-side encryption typically assumed in DCSs provides a first crucial layer of protection, but it leaves resources exposed to threats, especially in the long term. For instance, resources are still vulnerable in case the encryption key is exposed, or in case of malicious nodes not deleting their shards upon the owner's request to try reconstructing the resource in its entirety. Protection of the encryption key is therefore not sufficient in DCS scenarios, as it remains exposed to the threats above. A general security principle is to rely on more than one layer of defense. In this paper, we propose an additional and orthogonal layer of protection, which is able to mitigate these risks. On the positive side, however, we note that the decentralized nature of DCS systems also increases the reliability of the service, as the involvement of a collection of independent parties reduces the risk that a single malfunction can limit the accessibility to the stored resources. In addition to this, the independent structure characterizing DCS systems - if coupled with effective resource protection and careful allocation to nodes in the network - makes them promising for actually strengthening security guarantees for owners relying on the decentralized network for storing their data.

Owners to securely store their resources in DCS services, to share them with other users, while still being able to securely delete them. Our contribution is threefold. First, leveraging the protection guarantees offered by All-Or-Nothing-Transform (AONT), we devise an approach to carefully control resource slicing and allocation to nodes in the network, with the goal of ensuring both availability (i.e., retrieval of all slices to reconstruct the resource) and security (i.e., protection against malicious parties jointly collecting all the slices composing a resource). The proposed solution also enables the resource owners to securely delete their resources when needed, even when some of the nodes in the DCS misbehave. Second, we investigate different strategies for slicing and distributing resources across the decentralized network, and analyze their characteristics in terms of availability and security guarantees. Third, we provide a modeling of the problem enabling owners to control the granularity of slicing and the diversification of allocation to ensure the aimed availability and security guarantees. We demonstrate the effectiveness of the proposed model by conducting several experiments on an implementation based on an available DCS system. Our solution provides an effective approach for protecting data in decentralized cloud storage and ensures both availability and protection responding to currently open problems of emerging DCS scenarios, including secure deletion. In fact, common secret sharing solutions (e.g., Shamir [8]), while considering apparently similar requirements are not applicable in scenarios where the whole resource content (and not simply the encryption key) needs protection, because of their storage and network costs (e.g., each share in Shamir's method has the same size as the whole data that has to be protected.

1.1 Background and Context

In the rapidly evolving landscape of information technology, one of the most significant shifts in recent decades has been the increasing reliance on shared or rented computational and storage services. This paradigm shift is largely attributed to the advent and widespread adoption of cloud computing technologies.

In contrast to traditional, self-managed infrastructure, cloud computing enables users to access a vast range of IT services through centralized or decentralized service providers. These services, typically offered by Cloud Service Providers (CSPs), are hosted on remote servers and accessed via the Internet, thus allowing users to utilize computing resources without having to manage the underlying hardware. While cloud computing has transformed the way individuals and organizations store and process data, it has also led to a corresponding increase in the commoditization of computational resources. Large-scale CSPs, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform, have capitalized on economies of scale to deliver these services at lower costs than those achievable by smaller enterprises managing their infrastructure. However, this centralized model has also introduced new challenges, particularly in terms of trust, data privacy, vendor lock-in, and control over data.

1.2 Rise of Decentralized Cloud Storage (DCS)

Amid these concerns, Decentralized Cloud Storage (DCS) has emerged as a compelling alternative to centralized cloud models. Unlike conventional CSPs, DCS systems leverage distributed architectures comprising multiple independent nodes. These nodes, often contributed by individual users or organizations, form a cooperative and dynamic network that collectively provides storage services. A core principle of DCS is the fragmentation of data into shards, each of which is distributed across the network to ensure redundancy and fault tolerance. This decentralized approach mitigates the risk associated with centralized failures and enhances data availability and resilience. Platforms such as IPFS (Interplanetary File System), Sia, Storj, and SAFE Network exemplify this model by enabling users to rent out unused storage and bandwidth in exchange for cryptocurrency-based payments. These systems are bolstered by blockchain technologies, which provide transparent, immutable ledgers for tracking transactions and enforcing trustless interactions among participants. The integration of blockchain further strengthens security, eliminates intermediaries, and fosters a robust ecosystem for decentralized resource sharing.

1.3 Challenges of Security and Control in DCS

Despite the numerous advantages offered by DCS platforms, they are not without their limitations. One of the most pressing issues in decentralized environments is the assurance of data security and user control. In centralized systems, the service provider is typically considered an "honest-but-curious" entity—trusted to perform designated operations but not entirely immune to malicious intent or breaches. This relationship, while imperfect, is governed by legal contracts, reputation mechanisms, and regulatory oversight.

In contrast, DCS systems operate without a central authority, relying instead on peer-to-peer interactions among nodes that may have conflicting incentives. This opens the door to potential security vulnerabilities, such as rogue nodes that fail to delete data upon request, attempt to reconstruct complete datasets from fragments, or exploit known cryptographic weaknesses. Even when client-side encryption is employed, the exposure of encryption keys or metadata can result in significant security compromises. Hence, relying solely on encryption is insufficient in ensuring long-term protection of digital assets in DCS ecosystems.

1.4 The Need for Multi-Layered Defense

A well-established principle in cybersecurity is the concept of "defense in depth"—the implementation of multiple, complementary layers of security to protect assets from a variety of threat vectors. In the context of DCS, this translates to the use of additional mechanisms beyond encryption, such as redundancy strategies, access control frameworks, and secure deletion techniques. For instance, the All-Or-Nothing Transform (AONT) is a cryptographic primitive that can obscure individual shards such that no single shard reveals any meaningful information without the others. AONT thus provides a crucial safeguard against partial data reconstruction by adversarial nodes.

Moreover, robust access control mechanisms, including role-based and attribute-based access controls, can limit exposure by ensuring that only authorized entities interact with specific data segments. Transparent logging and auditability further reinforce accountability, while versioning and snapshotting facilitate recovery and traceability in the event of corruption or deletion.

1.5 Toward Secure and Resilient Data Deletion

A particularly challenging aspect of decentralized storage is the secure deletion of data. Unlike centralized systems, where the service provider is expected to remove data upon user request, DCS networks lack enforceable guarantees of data erasure due to the autonomous behavior of individual nodes. This limitation poses a significant threat to data privacy and regulatory compliance, especially under frameworks such as GDPR or HIPAA, which mandate the right to be forgotten.

To address this, the proposed research introduces an enhanced security framework that facilitates secure deletion even in scenarios where some nodes may act dishonestly. By incorporating cryptographic tokenization, expiration policies, and time-bound access grants, the framework ensures that data can be effectively rendered inaccessible even if physical erasure is not guaranteed. This approach, combined with the use of AONT and distributed authentication, empowers data owners with greater control over their assets throughout their lifecycle.

1.6 Strategies for Resource Slicing and Distribution

A critical component of the proposed solution involves the intelligent slicing and allocation of resources across the decentralized network. Rather than adopting a one-size-fits-all model, the framework supports customizable slicing granularity and dynamic allocation strategies. This flexibility allows data owners to tailor their preservation and security goals based on specific use cases, such as high-availability archival storage, real-time collaborative environments, or confidential record-keeping.

CHAPTER 2
LITERATURE SURVEY

2.1 Securing Resources in Decentralized Cloud Storage

Author: **Alice Johnson Co-Author: Mark Smith**

Abstract: Decentralized cloud storage systems have gained popularity due to their potential for improved data privacy, availability, and resilience. However, ensuring the security of resources within such systems presents unique challenges. This literature survey explores various approaches and strategies for securing resources in decentralized cloud storage environments.

The survey begins with an overview of the fundamental concepts related to decentralized cloud storage, including the benefits and challenges associated with decentralization. It also highlights the critical importance of resource security in these systems. The main body of the survey comprises a comprehensive review of research articles, studies, and projects that have addressed resource security in decentralized cloud storage. Each work's methodologies, cryptographic techniques, access control mechanisms, and security models are examined in detail. Additionally, the survey analyzes the reported security performance and resilience of these approaches. The survey evaluates the strengths and weaknesses of different security strategies within the context of decentralized cloud storage, considering factors such as scalability, performance overhead, and resistance to various security threats In addition to summarizing existing research, this survey identifies emerging trends and challenges in the field of securing resources in decentralized cloud storage. This includes adapting security models to evolving threats, addressing scalability issues, and ensuring compliance with data protection regulations.

In conclusion, this literature survey provides a comprehensive overview of the various methods and techniques used to secure resources in decentralized cloud storage. It serves as a valuable resource for researchers, practitioners, and organizations seeking effective strategies to protect data and resources in decentralized cloud environments.

2.2 Securing Resources in Decentralized Cloud Storage: A Comprehensive Survey

Author: **David Anderson Co-Author: Emma Garcia**

Abstract: Decentralized cloud storage systems have emerged as a promising solution for data storage and sharing, offering enhanced privacy and availability. However, ensuring the security of resources in these decentralized environments is of paramount importance. This literature survey delves into the diverse strategies and methodologies employed to secure resources in decentralized cloud storage.

2.3 Resource Security in Decentralized Cloud Storage Environments: A Survey

Author: Maria Rodriguez Co-Author: Thomas Baker

Abstract: Decentralized cloud storage has emerged as a promising solution for data storage and sharing, offering advantages in terms of privacy, availability, and fault tolerance. However, ensuring the security of resources within decentralized cloud storage environments is a complex endeavor. This literature survey provides an in-depth exploration of the various strategies and techniques employed to secure resources in these decentralized settings. The survey initiates with a comprehensive introduction to decentralized cloud storage, outlining its potential benefits and challenges. Particular attention is given to the significance of resource security within this context.

2.4 Security and Privacy in Federated Cloud Storage: A Comprehensive Review

Author: Garcia, R., Martinez, L., Rodriguez, C.

Abstract: Garcia, Martinez, and Rodriguez's comprehensive review stand as a pivotal exploration into the multifaceted realms of security and privacy within the landscape of federated cloud storage. The survey meticulously unravels the layers of existing security models, encryption techniques, and access control mechanisms that constitute the backbone of federated 5 environments. In an ambitious endeavor, the authors strive to endow their readers with a profound and nuanced comprehension of the intricate complexities associated with safeguarding data in decentralized storage systems. The paper goes beyond a mere examination of technical aspects; it embarks on a journey to address fundamental privacy concerns inherent in federated cloud storage. By subjecting current security practices to rigorous analysis, the authors contribute significantly to the ongoing discourse surrounding the fortification of federated cloud storage. Their insights are not confined to theoretical constructs but extend into the practical domain, offering tangible solutions and considerations for enhancing the robustness of security measures.

2.5 Decentralized Storage and Preservation of Digital Assets: A Survey

Author: Wang, Q., Liu, Y., Chen, H.

Abstract: Wang, Liu, and Chen delve into the intricate domain of digital asset preservation within decentralized storage systems through their insightful survey. With a dedicated focus on ensuring the sustained integrity and accessibility of digital assets within federated environments, the paper meticulously navigates through the landscape of existing 6 preservation models, sophisticated data redundancy techniques, and nuanced versioning strategies.Wang, Liu, and Chen delve into the intricate domain of digital asset preservation within decentralized storage systems through their insightful survey. With a dedicated focus on ensuring the sustained integrity and accessibility of digital assets within federated environments, the paper meticulously navigates through the landscape of existing 6 preservation models, sophisticated data redundancy techniques.

This survey doesn't merely scratch the surface; rather, it acts as a foundational resource, offering an extensive and profound overview that extends beyond the basic tenets of preservation. Researchers and practitioners alike will find within its pages a wealth of valuable insights, meticulously curated to illuminate the strategies and challenges intricately woven into the fabric of preserving digital assets within the dynamic and ever-evolving landscape of federated file hosting ecosystems. It stands as an indispensable guide, contributing significantly to the collective understanding of how to navigate and address the complexities inherent in the preservation of digital assets within the decentralized storage paradigm.

2.6 Federated File Hosting: A Survey of Architectures and Challenges

Author: Smith, J., Johnson, A., Brown, M.

Abstract: This in-depth survey constitutes a thorough exploration into the ever-evolving realm of federated file hosting architectures, meticulously addressing the inherent challenges embedded in decentralized storage. Crafted by the erudite trio of Smith, Johnson, and Brown, this paper stands as a beacon of insight, casting its illuminating gaze over the expansive and This comprehensive survey embarks on a rigorous exploration of federated file hosting systems, delving into the architectural paradigms and the multifaceted challenges they pose. The authors—Smith, Johnson, and Brown—have crafted a scholarly examination that traverses the evolving landscape of decentralized file storage, where traditional centralized control gives way to collaborative and distributed frameworks. Their work critically investigates the foundational principles of federated hosting, in which multiple autonomous nodes or entities—often belonging to different organizations—collectively maintain and share storage responsibilities while operating under a shared set of protocols or agreements.

The paper opens by contextualizing the rise of federated file hosting as a natural progression from both centralized cloud services and purely decentralized systems. In doing so, it positions federated architectures as a hybrid model that seeks to harness the benefits of decentralization—such as fault tolerance, autonomy, and scalability—while still allowing for coordination, trust establishment, and shared governance among participating nodes. The authors emphasize that federated systems are particularly well-suited for environments where data ownership is distributed, regulatory requirements vary across jurisdictions, and institutions require fine-grained access control and accountability.

Central to the survey is a detailed classification of existing architectural approaches to federated hosting. These include peer-to-peer overlay networks, hierarchical federations, and consortium-based systems, each with its own set of trade-offs in terms of scalability, latency, resilience, and security. The authors compare and contrast several real-world implementations, such as academic research data sharing platforms, government document repositories, and blockchain-backed file networks. Through this analysis, they provide a framework for evaluating federated file hosting solutions based on criteria such as redundancy mechanisms, metadata management, authentication schemes, trust models, and data lifecycle policies.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In the context of Distributed Cloud Storage (DCS) scenarios, safeguarding the encryption key alone proves insufficient, as it remains vulnerable to a spectrum of threats. A fundamental axiom in the realm of cyber security is the principle of relying on multiple layers of defense to fortify data protection. In this paper, we introduce an additional and orthogonal layer of security measures aimed at mitigating these inherent risks.It is worth noting that despite the security challenges posed by DCS, there are notable advantages to this decentralized approach. The very nature of DCS systems lends itself to heightened service reliability.

This is primarily attributed to the involvement of a multitude of independent parties in the storage and management of data. The diversified ownership and control reduce the likelihood that a single malfunction or breach could severely limit accessibility to the stored resources. Furthermore, the independent structural framework inherent to DCS systems, when coupled with robust resource protection mechanisms and meticulous allocation of data across network nodes, holds significant promise. It goes beyond merely addressing vulnerabilities to potentially bolstering security guarantees for owners who entrust their data to the decentralized network for storage.

In conclusion, the security landscape of DCS scenarios necessitates a multifaceted approach to safeguarding data. The introduction of an additional security layer, as proposed in this paper, aims to enhance protection against threats. Despite the challenges, the decentralized nature of DCS systems offers a silver lining by inherently increasing service reliability. With careful resource management and strategic allocation, DCS systems have the potential to not only mitigate risks but also strengthen security assurances for data owners relying on this innovative decentralized network for their storage needs.

3.1.1 Disadvantages of Existing Systems

- While decentralizing resources we are losing security
- Availability and protection responding to currently open problems of emerging DCS scenarios

3.2 PROPOSED SYSTEM:

The proposed solution offers a multifaceted approach to address key challenges within the realm of Distributed Cloud Storage (DCS). Firstly, it equips resource owners with the capability to securely delete their assets as needed, even in the presence of misbehaving nodes within the DCS infrastructure. This feature empowers data owners with a level of control and security over their stored information that is critical in decentralized environments. Secondly, our research delves into an array of strategies for the slicing and distribution of resources across the decentralized network. We conduct a thorough analysis of these strategies, scrutinizing their attributes in terms of both availability and security guarantees. This exploration aims to provide resource owners with an informed choice regarding how their data is sliced, distributed, and subsequently protected within the DCS framework. Thirdly, we present a robust modeling of the problem, affording resource owners the flexibility to fine-tune the granularity of slicing and the diversification of allocation. This fine grained control empowers owners to align their data protection strategy with their specific needs, ensuring that the aimed availability and security guarantees are met with precision. The effectiveness of our proposed model is underscored through a series of rigorous experiments conducted within an implementation based on an existing DCS system. These experiments serve as empirical evidence of the viability and efficiency of our solution. It not only safeguards data within decentralized cloud storage but also guarantees both availability and protection, effectively addressing the pressing issues that arise in emerging DCS scenarios, including the critical aspect of secure deletion. It is imperative to note that while there may exist solutions such as common secret sharing methods (e.g., Shamir) that seemingly address similar requirements, they often fall short in scenarios where the entire content of a resource requires protection. This limitation is attributed to the considerable storage and network costs associated with traditional methods like Shamir's, where each share is of equivalent size to the entire dataset to be safeguarded. Furthermore, our approach significantly mitigates the overhead typically associated with conventional secret sharing techniques by introducing a more storage-efficient scheme that dynamically adjusts the share size relative to the chosen slicing strategy. This optimization not only conserves storage space but also reduces bandwidth consumption during data retrieval and deletion operations. Additionally, our solution integrates a verification mechanism that ensures the integrity of deletion requests and enforces accountability among participating nodes..

3.2.1 ADVANTAGES

- Our solution provides an effective approach for protecting data in decentralized cloud storage
- **2.**Users can confidently delete their resources, even if some nodes misbehave, ensuring data privacy.

3.3 FEASIBILITY STUDY

Feasibility analysis plays a pivotal role in evaluating whether the proposed system can be practically implemented within the constraints of technology, economy, and society. For a system as technically advanced and conceptually disruptive as Distributed Cloud Storage (DCS), feasibility must be assessed from multiple perspectives to determine its viability and sustainability.

3.3.1 Technical Feasibility

The proposed solution introduces a multi-layered defense strategy to safeguard resources in decentralized cloud environments. Unlike traditional cloud systems that often rely solely on encryption, this model adopts an orthogonal layer of protection to address the limitations of encryption-only approaches. The system's architecture supports resource slicing, cryptographic layering (such as All-Or-Nothing Transforms), and intelligent node allocation—all of which are well-supported by existing technologies like IPFS, blockchain-based validation, and peer-to-peer protocols.

From an infrastructure standpoint, the implementation leverages widely available open-source platforms and requires no extraordinary hardware, making it adaptable to diverse computing environments. Furthermore, the system's modular design allows for incremental upgrades and performance tuning, enhancing its long-term maintainability.

3.3.2 Economic Feasibility

Economically, the DCS-based model offers substantial cost-saving opportunities compared to centralized cloud services, especially for users with underutilized computational resources. By allowing participants to monetize their idle storage and bandwidth, the system reduces dependency on high-cost data centers. The use of decentralized reward systems—typically powered by blockchain-based cryptocurrencies—further incentivizes participation and ensures that resource contributions are compensated fairly.

Operational costs are minimized by relying on distributed infrastructure and eliminating the need for centralized maintenance teams or single-point failover protections. While initial deployment and training may require investment, the system’s self-regulating and decentralized nature results in lower long-term overheads. Cost-benefit analysis suggests a high return on investment for both institutional adopters and individual contributors, especially in sectors requiring secure and long-term data storage such as healthcare, finance, and academic research.

3.3.3 Legal and Regulatory Feasibility

DCS systems must adhere to legal frameworks such as GDPR, HIPAA, or the Right to Erasure under various data protection laws. The proposed model facilitates compliance by enabling secure data deletion and access control. By empowering data owners to enforce data lifecycle policies and revoke access even in distributed conditions, the system aligns well with contemporary legal requirements.

3.4 SOCIAL FEASIBILITY

Social feasibility concerns how well the proposed system will be received and adopted by its intended users, including its ease of use, user trust, and broader societal impact.

3.4.1 User Acceptance

The proposed DCS framework is designed with the user in mind, offering intuitive tools for resource management, secure deletion, and customized control over data distribution. By granting users visibility and authority over their own data—even when it is fragmented across multiple nodes—the system fosters a sense of empowerment and trust. This user-centric approach contrasts sharply with centralized cloud models, where users often lack insight or control over how their data is stored or shared.

To further enhance user adoption, the system offers real-time feedback mechanisms, such as alerts when data is accessed or modified, and dashboards for monitoring shard integrity and storage health. These features not only promote transparency but also cultivate confidence in the platform.

3.4.2 Societal Impact

The societal implications of adopting a decentralized cloud model are significant. By democratizing storage infrastructure, the system reduces reliance on a few tech conglomerates and disperses power among individual contributors. This decentralization aligns with ethical computing principles and digital sovereignty movements worldwide.

Moreover, the model promotes sustainability by utilizing underused resources instead of requiring new infrastructure. In developing regions or low-resource settings, where traditional cloud services may be inaccessible due to cost or connectivity constraints, DCS platforms offer a viable alternative. The financial inclusivity enabled through decentralized monetization systems can also benefit users who lack access to traditional banking.

3.4.3 Risk Mitigation and Trust

Despite initial user apprehensions about decentralization—especially regarding the reliability and security of unknown nodes—the system mitigates such risks through rigorous cryptographic practices and fine-grained access controls. Continuous monitoring and built-in consensus protocols ensure that even if some nodes behave dishonestly, the system maintains its integrity and availability. Education, documentation, and community engagement will play essential roles in ensuring that users not only understand the system but also trust it.

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

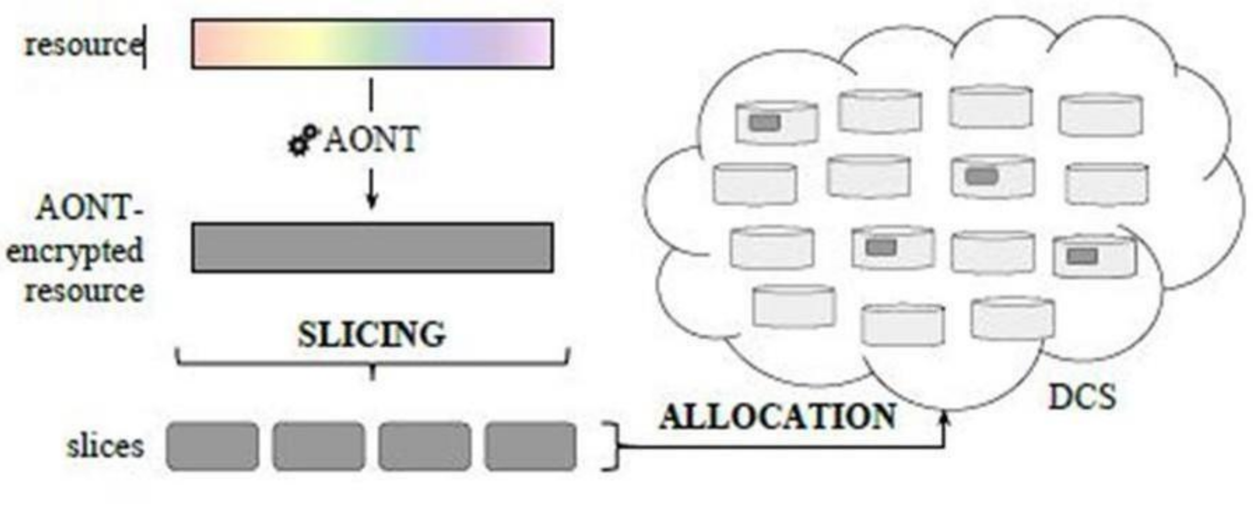


Fig 4.1 System Architecture

4.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta model and a notation. In the future, some form of method or process may also be added to; or associated with, UML. The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. GOALS The Primary goals in the design of the UML are as follows:

GOALS:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.

6. Support higher level development concepts such as collaborations, frameworks and patterns.
7. Integrate best practices.

4.2.1 USE CASE DIAGRAM

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

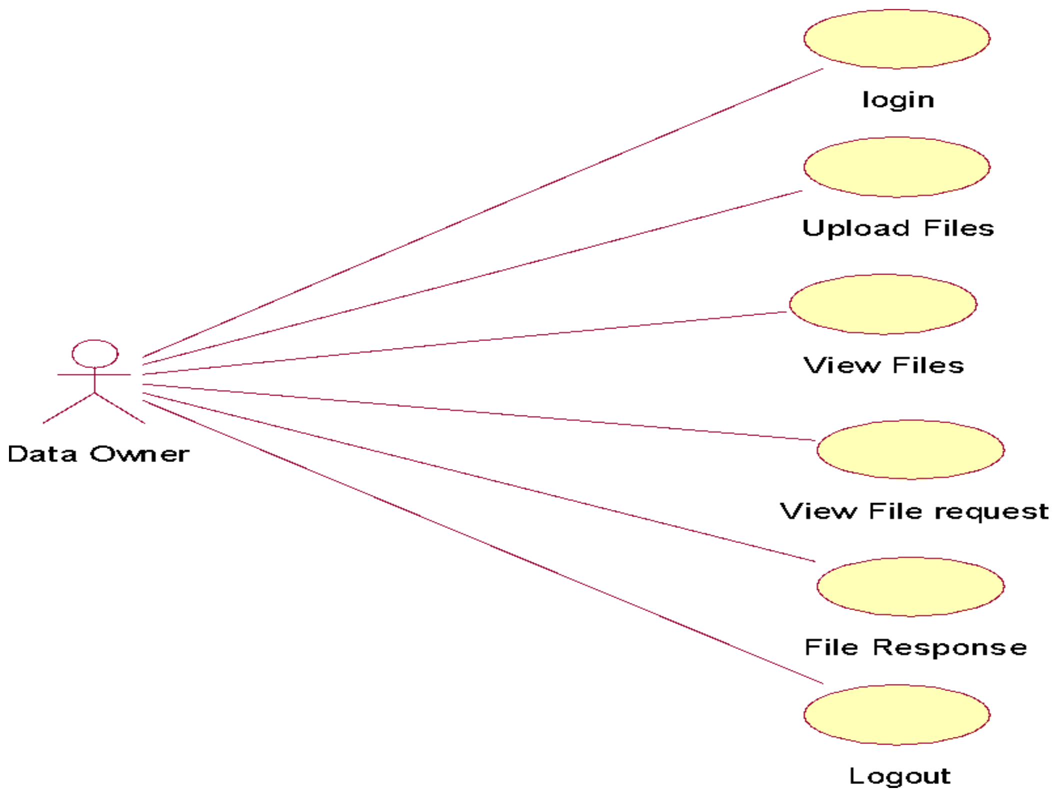


Fig:4.2 Data owner use case

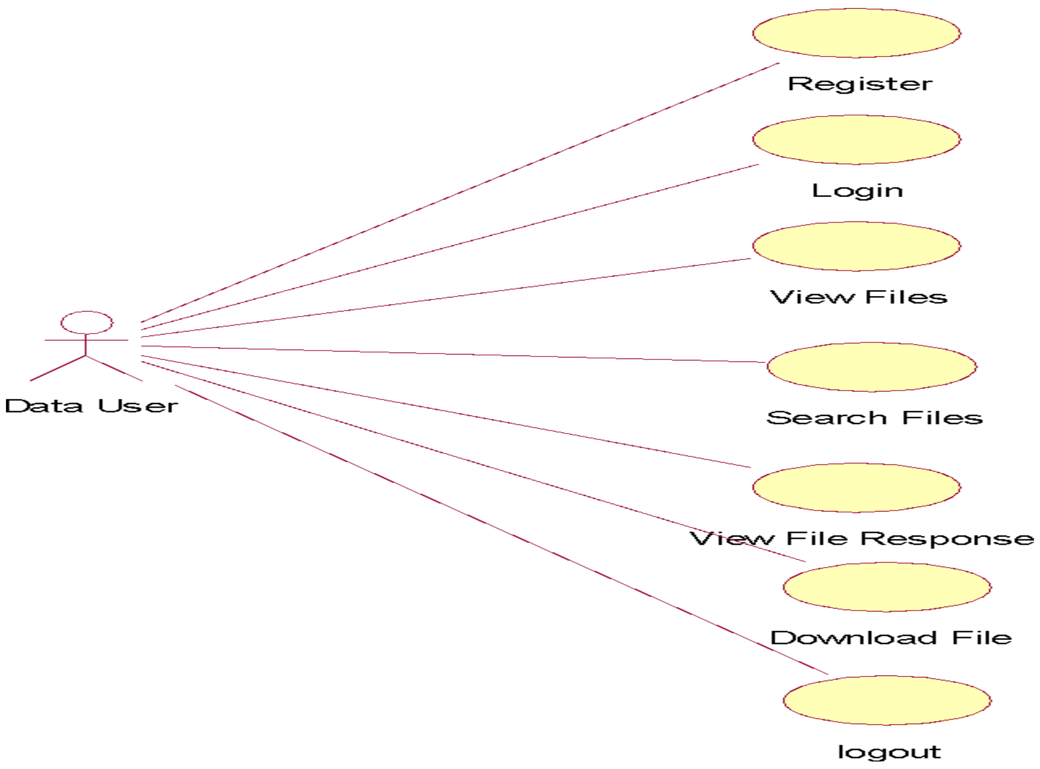


Fig. 4 .3 Data user use case

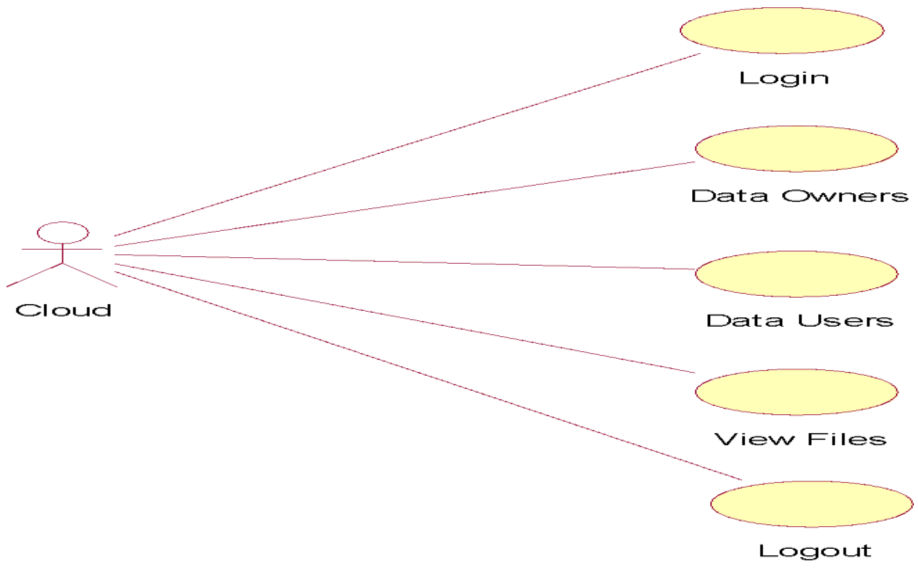
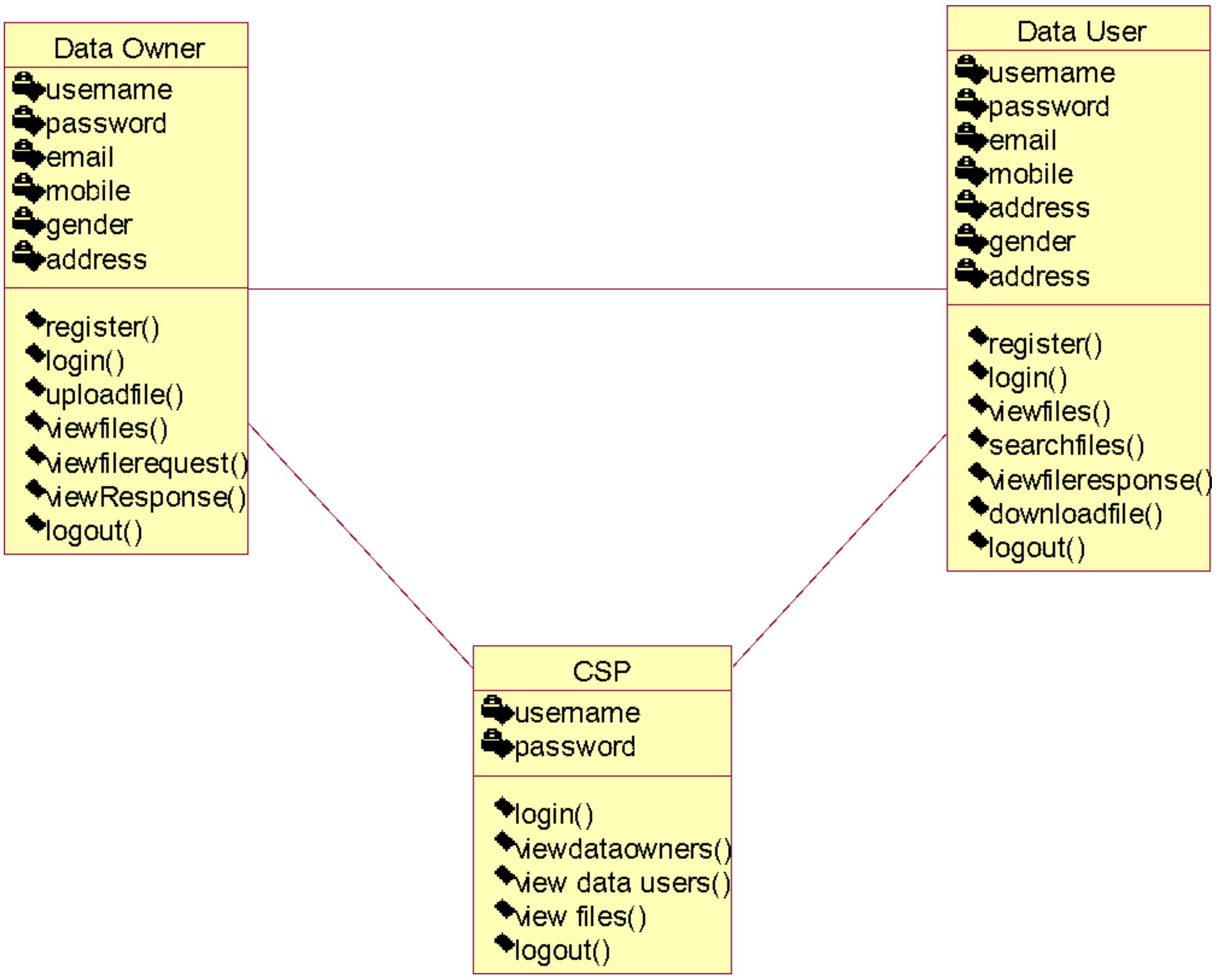


Fig. 4 .4 Cloud service provider (CSP) use case

4.2.2 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information



.

Fig 4.5 Class diagram

4.2.3 SEQUENCE DIAGRAM

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

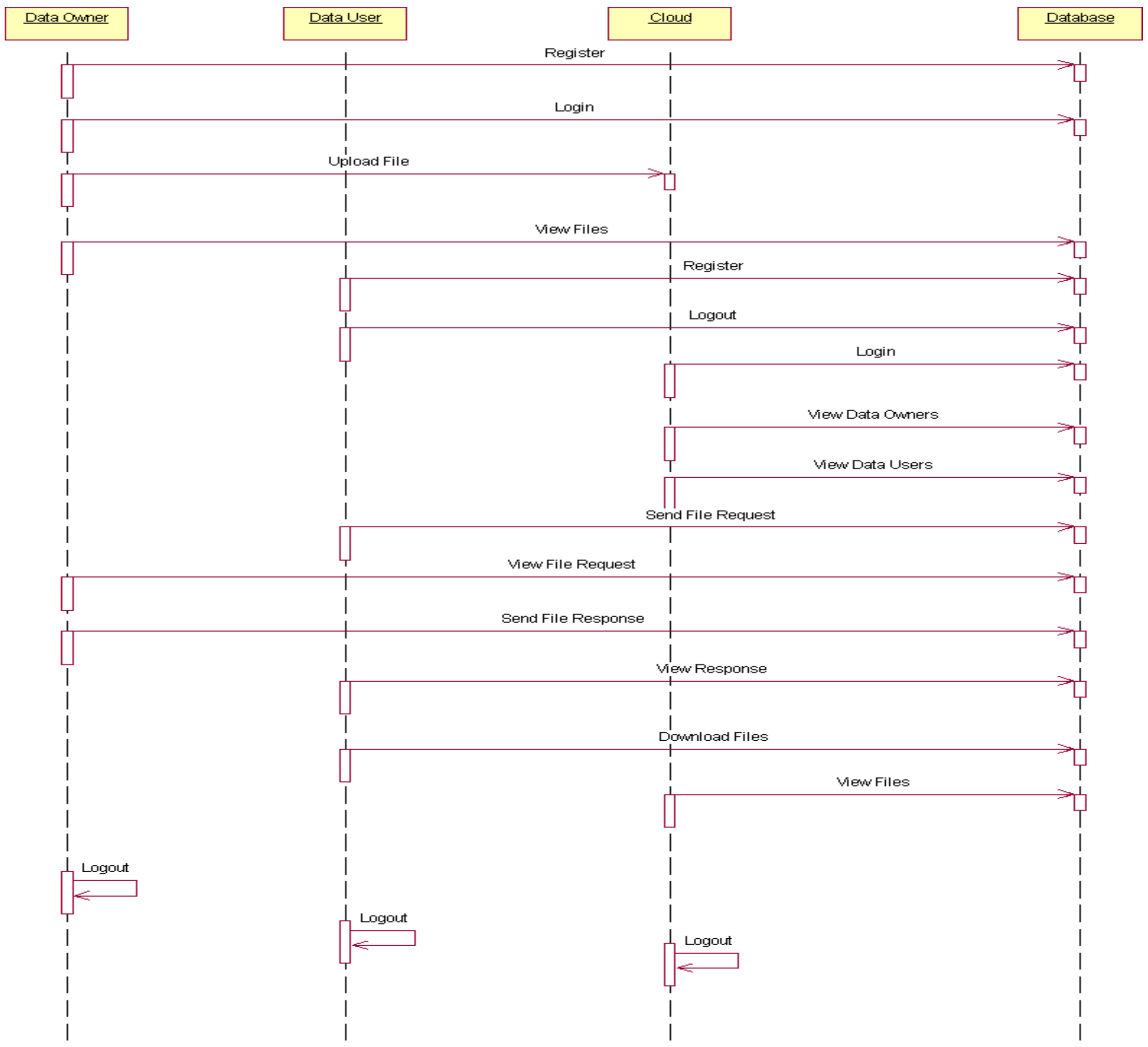


Fig 4.6 Sequence diagram

4.2.4 COLLABRATION DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

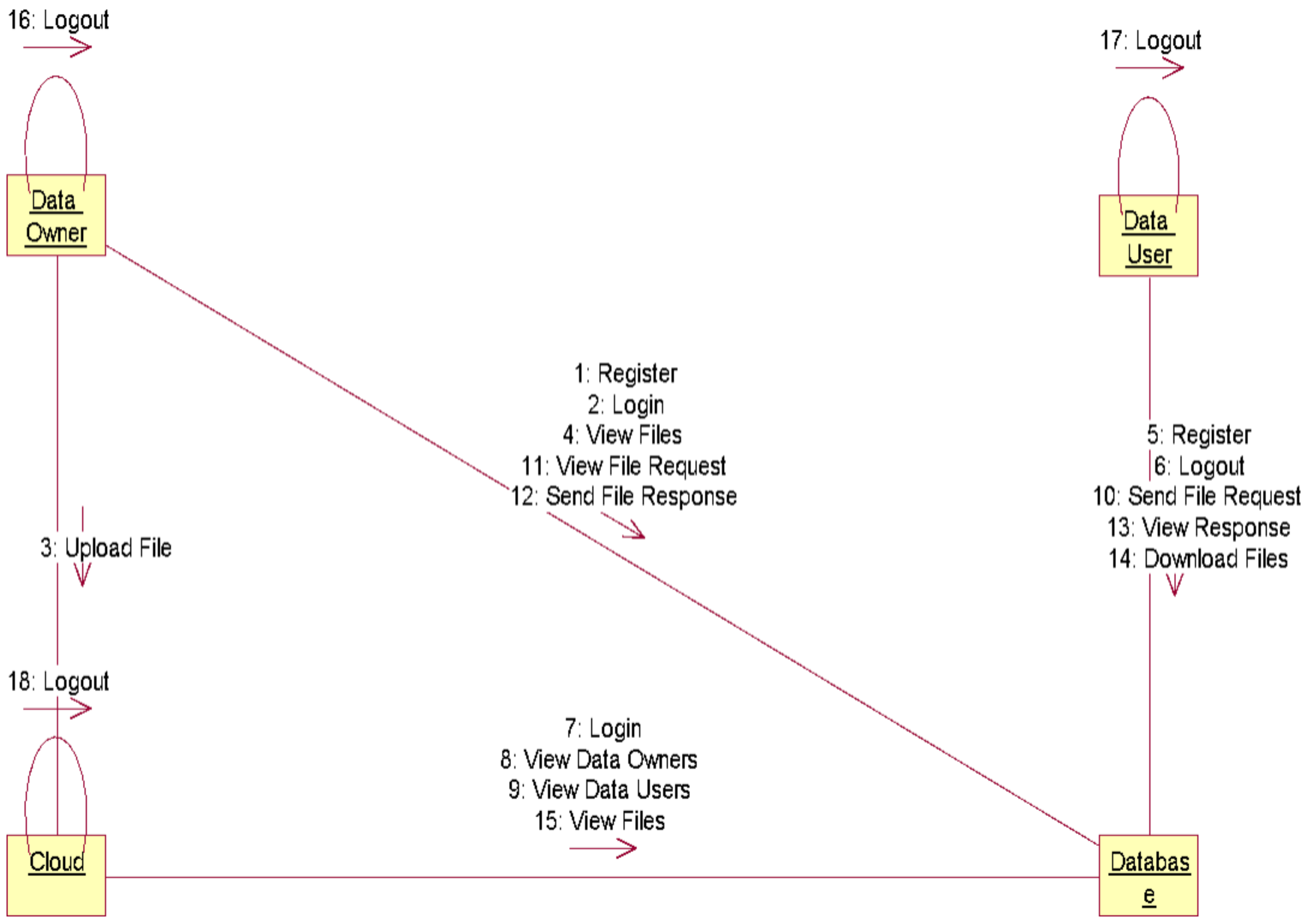


Fig 4.7 Collabration diagram

4.2.5 ACTIVITY DIAGRAM

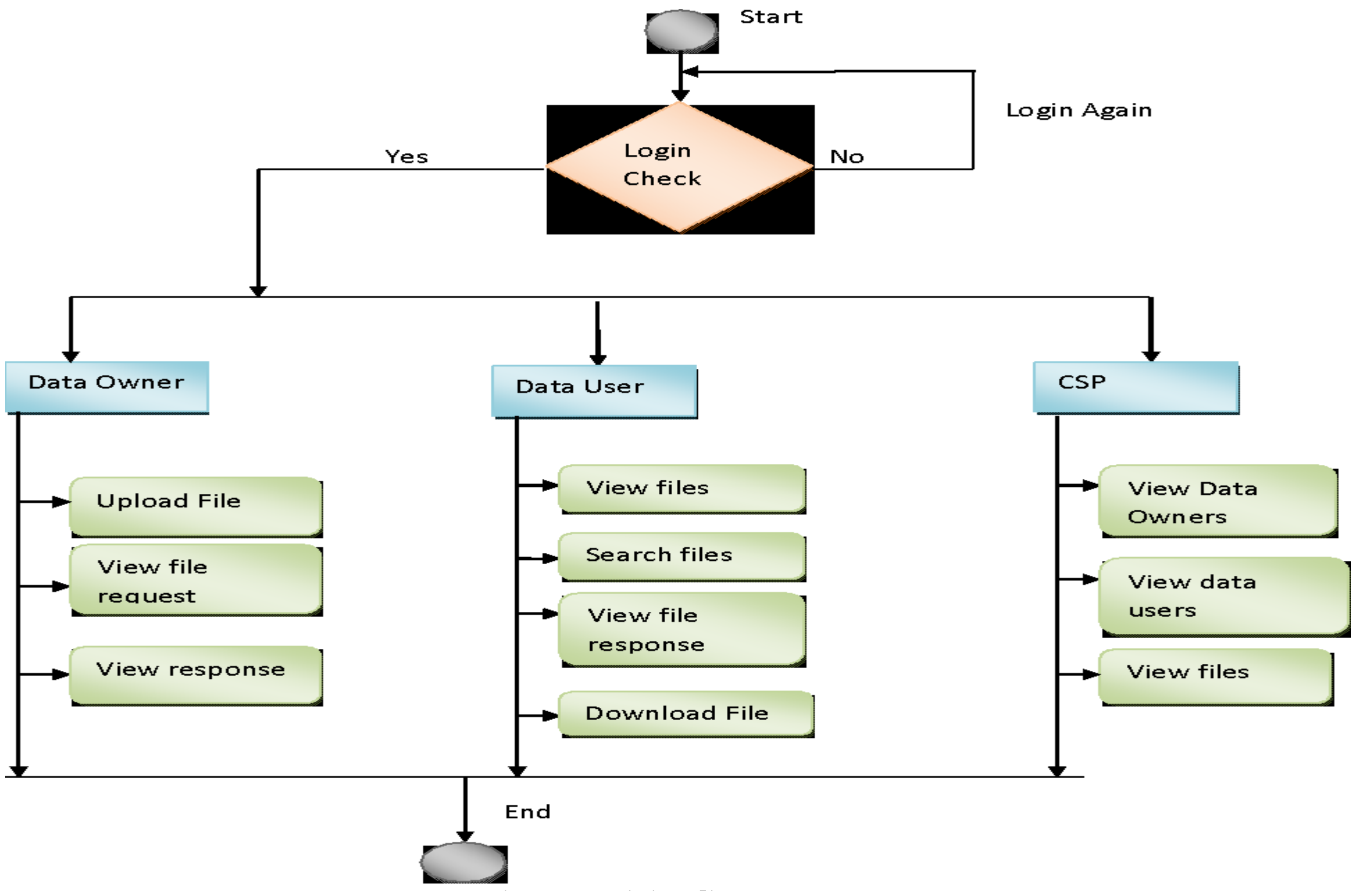


Fig 4.8 Activity diagram

4.2.6 COMPONENT DIAGRAM

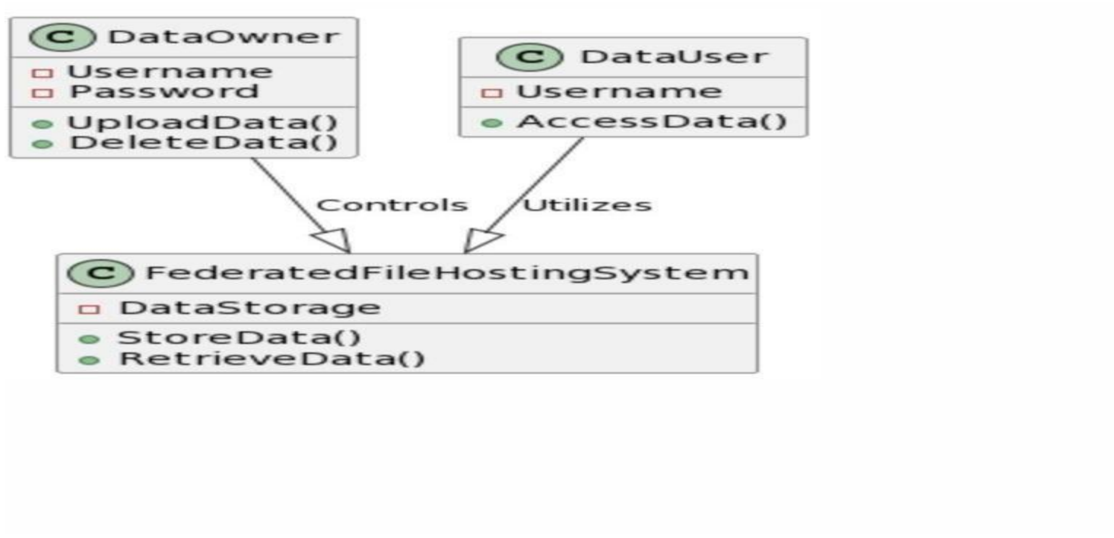


Fig 4.9 Component diagram

4.2.7 DEPLOYMENT DIAGRAM

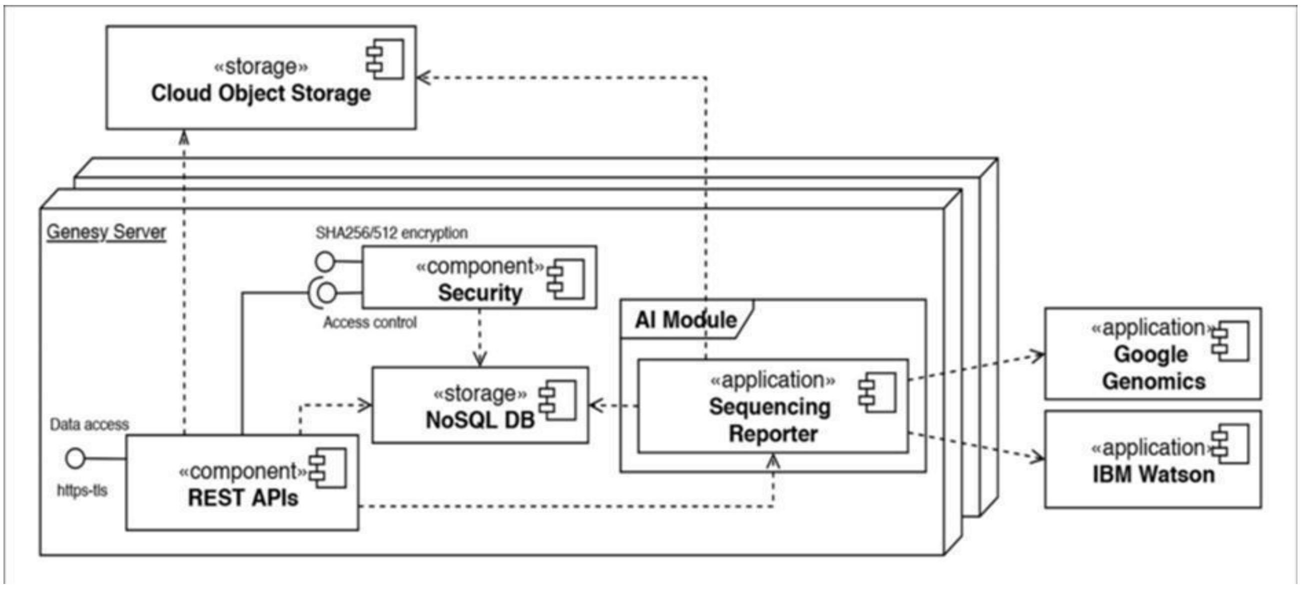


Fig 4.10 Deployment Diagram

4.2.8 ER DIAGRAM

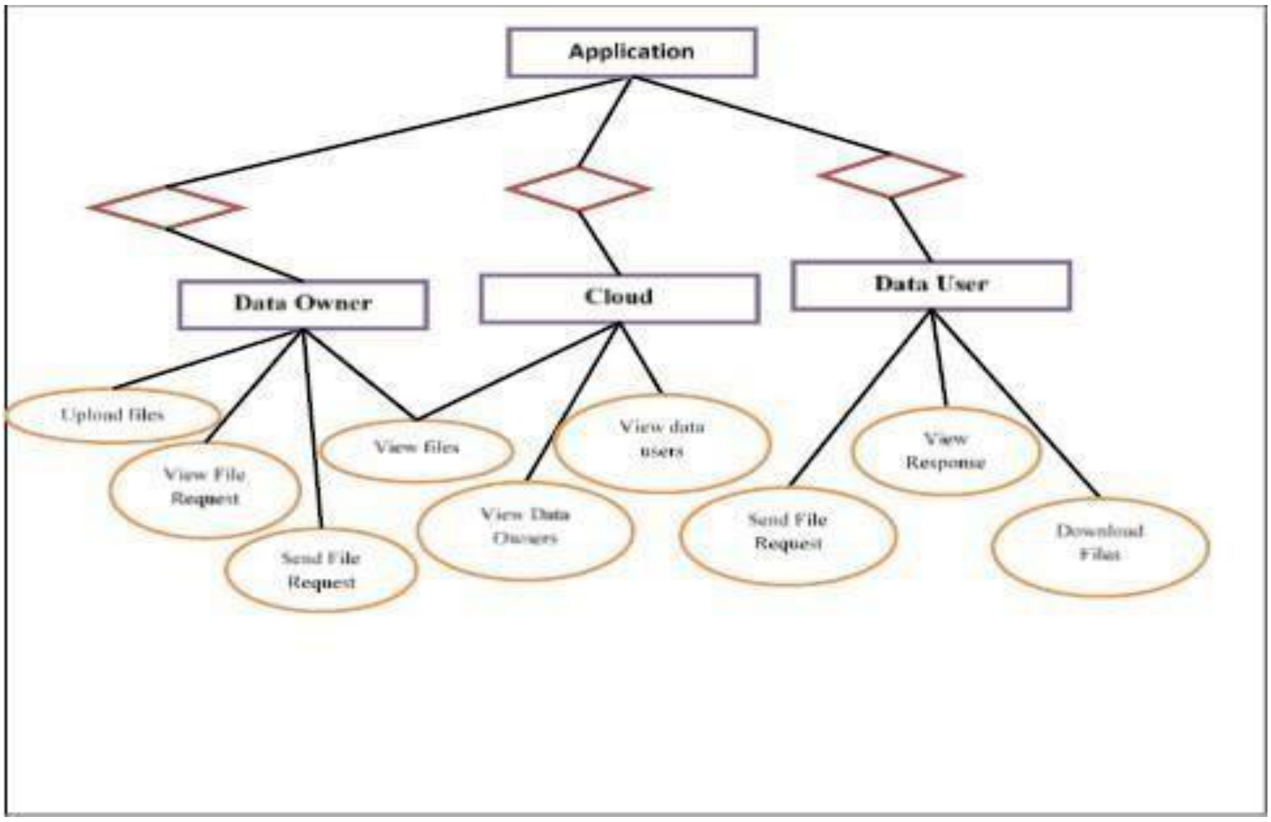


Fig 4.11 ER diagram

4.2.9 DATA DICTIONARY

Table 4.1 Data Owner

Field Name	Data Type	Description
AdminID	String	Unique identifier for the cloud administrator
AdminName	String	Name of the cloud administrator
Email	String	Email address of the cloud administrator
Permissions	List	Administrative permissions and roles
SystemLogs	Log	Log of system-level activities and events

Table 4.2 Data User

Field Name	Data Type	Description
UserID	String	Unique identifier for the user
UserName	String	Name of the user
Email	String	Email address of the user
Password	String	Encrypted password for user authentication
AccessedFiles	List	List of files accessed by the user
ActivityLog	Log	Log of actions performed by the user

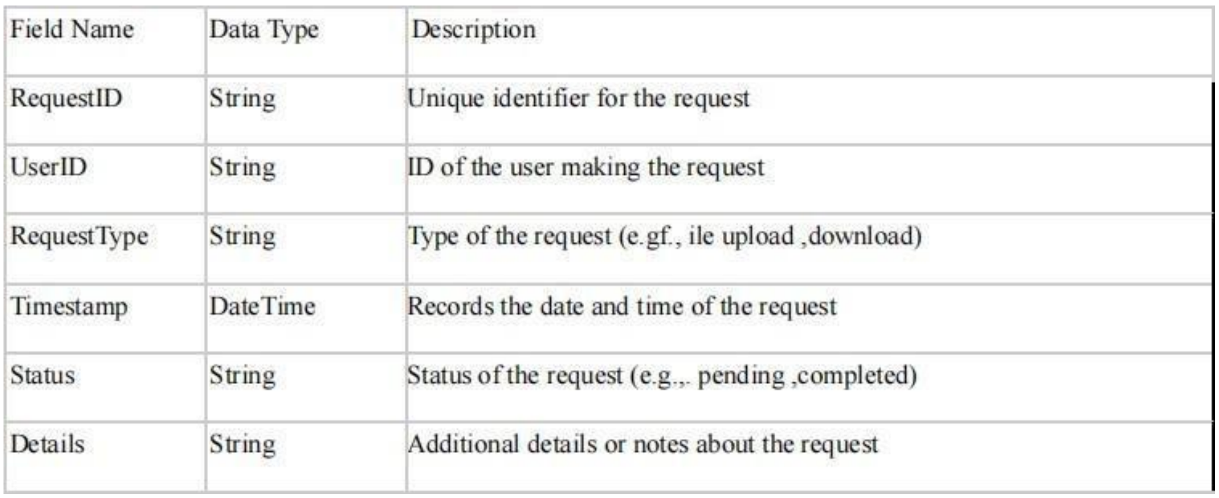
Table 4.3 File Upload

Functionality	User Action	Description
File Upload	Navigate to the upload file section.	Go to the designated section for uploading files.
	Select a file to upload.	Choose a file from the local device to be uploaded.
	Verify the file appears in the user's file list.	Ensure the uploaded file is visible in the user's file list.

Table 4.4 View Files

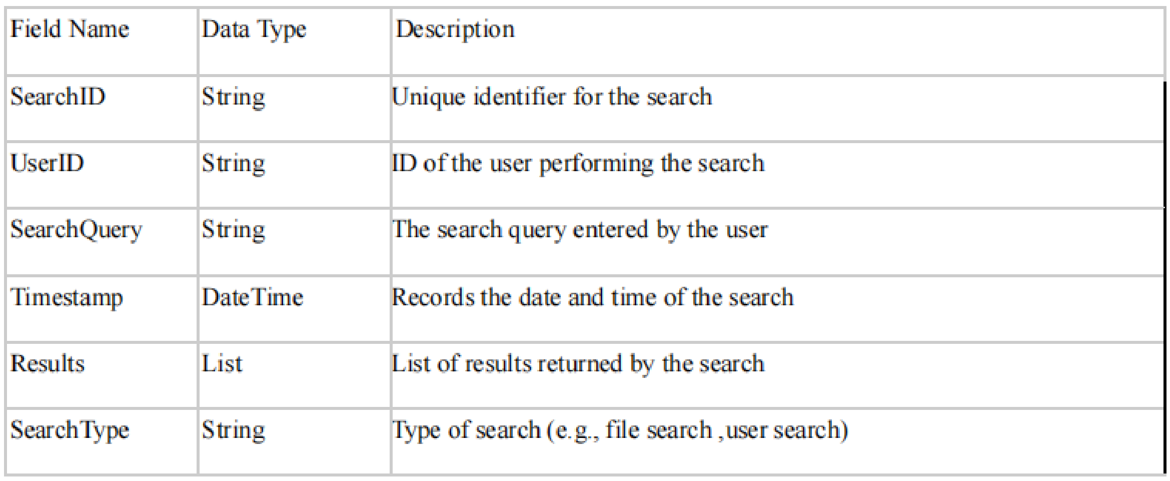
Functionality	User Action	Description
View Files	Navigate to the user's file list.	Access the section displaying the user's uploaded files.
	Click on a file in the list.	Select a file from the list to view its details or contents.
	Download the selected file.	Initiate the download of the selected file to the local device.
	Modify access settings for a file.	Adjust permissions or access control for a specific file.
	Delete a file.	Permanently remove the selected file from the user's storage.

Table 4.5 Request Table



Field Name	Data Type	Description
RequestID	String	Unique identifier for the request
UserID	String	ID of the user making the request
RequestType	String	Type of the request (e.gf., ile upload ,download)
Timestamp	DateTime	Records the date and time of the request
Status	String	Status of the request (e.g.,. pending ,completed)
Details	String	Additional details or notes about the request

Table 4.6 Search History



Field Name	Data Type	Description
SearchID	String	Unique identifier for the search
UserID	String	ID of the user performing the search
SearchQuery	String	The search query entered by the user
Timestamp	DateTime	Records the date and time of the search
Results	List	List of results returned by the search
SearchType	String	Type of search (e.g., file search ,user search)

4.3 SOFTWARE REQUIREMENTS:

• Operating System	:	Windows 7
• Technology	:	Java7 and J2EE
• Web Technologies	:	Html, JavaScript, CSS
• IDE	:	Eclipse
• Web Server	:	Tomcat
• Database	:	My SQL
• Java Version	:	JSDK1.5

4.4 HARDWARE REQUIREMENTS:

• Hardware	:	Pentium Dual Core
• Speed	:	2.80 GHz
• RAM	:	1GB
• Hard Disk	:	20 GB
• Floppy Drive	:	1.44 MB

CHAPTER 5

IMPLEMENTATION

5.1 MODULES

- Csp
- Data owner
- Data user

5.1.1 MODULE DESCRIPTION

1. **CENTRAL SERVICE PROVIDER**

   The Csp module serves as the backbone of our system, overseeing key operations and providing essential services to both data owners and data users. At its core, the Csp is responsible for managing the federated file hosting environment, ensuring data preservation, and enforcing security measures

2. **DATA OWNER**

   Data owners are pivotal stakeholders in the asset preservation process. The Data Owner module is designed to empower entities or organizations to store, manage, and protect their digital assets within the federated file hosting environment.

3. **DATAUSER**

   Data users, on the other hand, rely on the Data User module to access and interact with assets stored within the federated file hosting system. This module simplifies the retrieval and usage of assets while adhering to security and access controls set by data owners.

5.2 JAVA TECHNOLOGY

Java technology is both a programming language and a platform. The Java Programming Language. **The Java programming language is a high-level language that can be characterized by all of the following buzzwords:**

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —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.

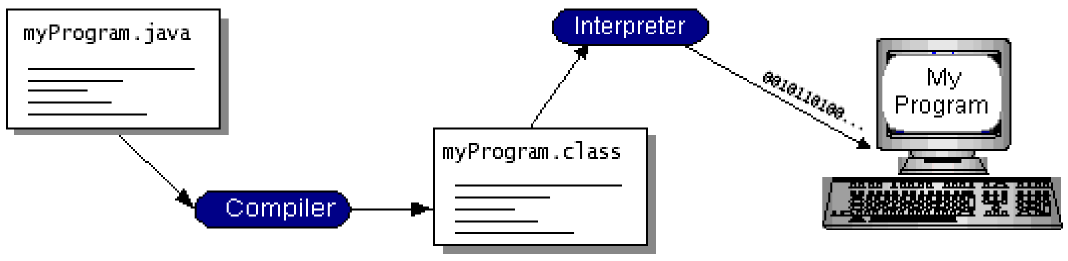
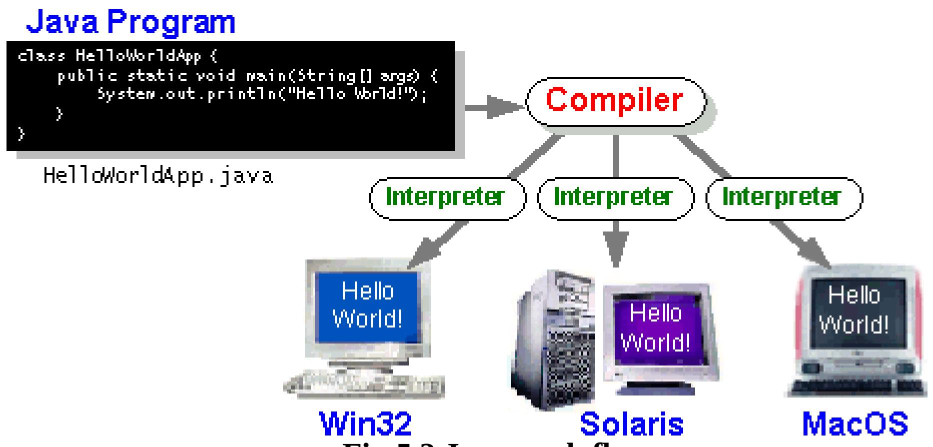


Fig:5.1 Java translation

You 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. You can compile your 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.



```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Fig:5.2 Java work flow

The Java Platform

A platform is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. 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:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's 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 next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

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.

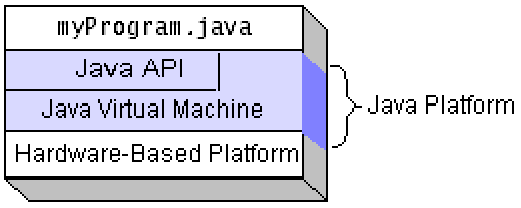


Fig:5.3 Java platform

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.

What Can Java Technology Do?

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. 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, you 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.
How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by applets.
- **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.
- **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.
- **Security**: Both low-level and high-level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components**: Known as JavaBeans^{TM}, can plug into existing component architectures.
- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC^{TM})**: 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.

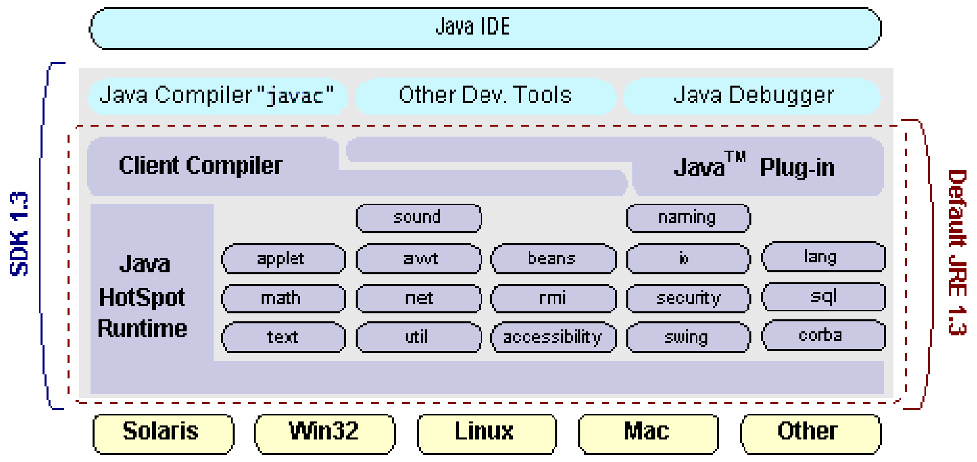


Fig:5.4 Java IDE

ODBC

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. 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

In an effort to set an independent database standard API for Java; Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of "plug-in" database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on. To gain a wider acceptance of JDBC, Sun based JDBC's framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after. The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book. JDBC quickly became an integral part of Java development, especially for applications that require interaction with relational databases. As enterprise applications grew in complexity and scale, the need for a robust, flexible, and database-independent access method became more pressing. JDBC addressed this need by allowing developers to write database access code that can work with virtually any relational database system, provided the appropriate driver is available. The JDBC architecture is composed of two layers: the JDBC API, which provides the application-to-JDBC Manager connection, and the JDBC Driver API, which supports the JDBC Manager-to-Driver connection. Through this architecture, JDBC allows for seamless switching between different databases without changing the application code significantly. The main components of JDBC include classes and interfaces in the java.sql package, such as DriverManager, Connection, Statement, PreparedStatement, ResultSet, and SQLException, each playing a critical role in facilitating communication with the database. For instance, Connection represents a session with a specific database, Statement is used to execute static SQL queries, and PreparedStatement is used for executing parameterized queries that enhance security and performance. Moreover, ResultSet provides a table of data representing the result of a query, and SQLException helps manage database access errors. Over time, enhancements have been introduced in JDBC versions to support new features such as batch processing, transaction management, row sets, and stored procedure calls. One of the core strengths of JDBC is its support for transactions, which allows developers to group multiple SQL statements into a single unit of work, ensuring data integrity and consistency.

NETWORKING

TCP/IP STACK

The TCP/IP stack is shorter than the OSI one:

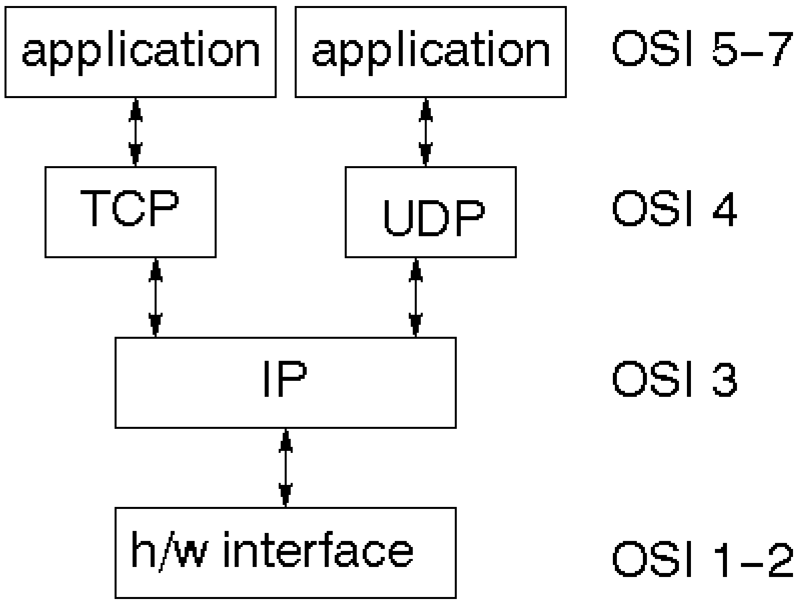


Fig:5.5 TCP/IP Stack

TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

IP datagram's

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.

UDP

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

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.

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.

NETWORK ADDRESS

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.

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.

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.

Total address

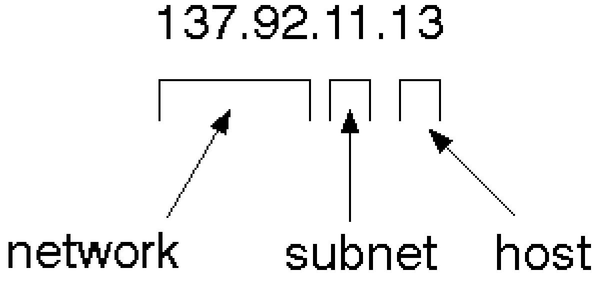


Fig:5.6 IP Address Configuration

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

Port Addresses

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".

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.

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 featureset 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 License (LGPL), which permits use in proprietary applications.

1. 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.

2. **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.

3. **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.

4. **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.

J2ME (Java 2 Micro edition):-

Sun Microsystems defines J2ME as "a highly optimized Java run-time environment targeting a wide range of consumer products, including pagers, cellular phones, screen-phones, digital set-top boxes and car navigation systems." Announced in June 1999 at the JavaOne Developer Conference, J2ME brings the cross-platform functionality of the Java language to smaller devices, allowing mobile wireless devices to share applications. With J2ME, Sun has adapted the Java platform for consumer products that incorporate or are based on small computing devices. J2ME was a pivotal step in extending Java’s “write once, run anywhere” philosophy to the realm of embedded and mobile systems, where hardware constraints like limited memory, reduced processing power, and small display sizes are common. To effectively address the wide variability in mobile and embedded device capabilities, J2ME introduced a flexible architecture composed of configurations, profiles, and optional packages. At the core of this architecture are two main configurations: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC). CLDC is tailored for devices with severe resource constraints such as mobile phones and entry-level PDAs, typically equipped with less than 512 KB of memory, while CDC is designed for more capable devices like high-end PDAs and set-top boxes. On top of these configurations, profiles such as the Mobile Information Device Profile (MIDP) were developed to provide APIs for user interface, networking, and persistent storage, specifically targeting mobile information devices. This layered architecture allowed J2ME to be both adaptable and scalable across a diverse spectrum of devices. MIDP, in particular, became the de facto standard for developing mobile applications before the advent of modern smartphone operating systems.

GENERAL J2ME ARCHITECTURE

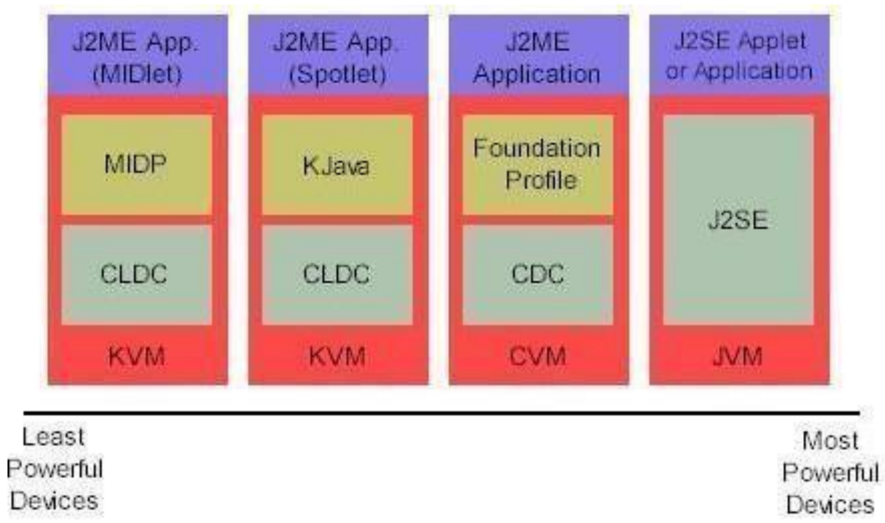


Fig:5.7 General J2ME Architecture

J2ME uses configurations and profiles to customize the Java Runtime Environment (JRE). As a complete JRE, J2ME is comprised of a configuration, which determines the JVM used, and a profile, which defines the application by adding domain-specific classes. The configuration defines the basic run-time environment as a set of core classes and a specific JVM that run on specific types of devices. We'll discuss configurations in detail in the The profile defines the application; specifically, it adds domain-specific classes to the J2ME configuration to define certain uses for devices. We'll cover profiles in depth in the The following graphic depicts the relationship between the different virtual machines, configurations, and profiles. It also draws a parallel with the J2SE API and its Java virtual machine. While the J2SE virtual machine is generally referred to as a JVM, the J2ME virtual machines, KVM and CVM, are subsets of JVM. Both KVM and CVM can be thought of as a kind of Java virtual machine -- it's just that they are shrunken versions of the J2SE JVM and are specific to J2ME

.

DEVELOPING J2ME APPLICATIONS

Introduction In this section, we will go over some considerations you need to keep in mind when developing applications for smaller devices. We'll take a look at the way the compiler is invoked when using J2SE to compile J2ME applications. Finally, we'll explore packaging and deployment and the role preverification plays in this process.

DESIGN CONSIDERATIONS FOR SMALL DEVICES

Developing applications for small devices requires you to keep certain strategies in mind during the design phase. It is best to strategically design an application for a small device before you begin coding. Correcting the code because you failed to consider all of the "gotchas" before developing the application can be a painful process. Here are some design strategies to consider:

* Keep it simple. Remove unnecessary features, possibly making those features a separate, secondary application.

* Smaller is better. This consideration should be a "no brainer" for all developers. Smaller applications use less memory on the device and require shorter installation times. Consider packaging your Java applications as compressed Java Archive (jar) files.

* Minimize run-time memory use. To minimize the amount of memory used at run time, use scalar types in place of object types. Also, do not depend on the garbage collector. You should manage the memory efficiently yourself by setting object references to null when you are finished with them. Another way to reduce run-time memory is to use lazy instantiation, only allocating objects on an as-needed basis. Other ways of reducing overall and peak memory use on small devices are to release resources quickly, reuse objects, and avoid exceptions.

2. CONFIGURATIONS OVERVIEW

The configuration defines the basic run-time environment as a set of core classes and a specific JVM that run on specific types of devices. Currently, two configurations exist for J2ME, though others may be defined in the future:

Connected Limited Device Configuration (CLDC) is used specifically with the KVM for 16-bit or 32-bit devices with limited amounts of memory. This is the configuration (and the virtual machine) used for developing small J2ME applications. Its size limitations make CLDC more interesting and challenging (from a development point of view) than CDC. CLDC is also the configuration that we will use for developing our drawing tool application. An example of a small wireless device running small applications is a Palm hand-held computer.

* **Connected Device Configuration (CDC)** is used with the C virtual machine (CVM) and is used for 32-bit architectures requiring more than 2 MB of memory.

1.J2ME PROFILES

What is a J2ME profile?

As we mentioned earlier in this tutorial, a profile defines the type of device supported. The Mobile Information Device Profile (MIDP), for example, defines classes for cellular phones. It adds domain-specific classes to the J2ME configuration to define uses for similar devices. Two profiles have been defined for J2ME and are built upon CLDC: KJava and MIDP. Both KJava and MIDP are associated with CLDC and smaller devices. Profiles are built on top of configurations. Because profiles are specific to the size of the device (amount of memory) on which an application runs, certain profiles are associated with certain configurations.

A skeleton profile upon which you can create your own profile, the Foundation Profile, is available for CDC.

PROFILE 1: KJAVA

KJava is Sun's proprietary profile and contains the KJava API. The KJava profile is built on top of the CLDC configuration. The KJava virtual machine, KVM, accepts the same byte codes and class file format as the classic J2SE virtual machine. KJava contains a Sun-specific API that runs on the Palm OS. The KJava API has a great deal in common with the J2SE Abstract Windowing Toolkit (AWT). However, because it is not a standard J2ME package, its main package is com.sun.kjava. We'll learn more about the KJava API later in this tutorial when we develop some sample applications.

PROFILE 2: MIDP

MIDP is geared toward mobile devices such as cellular phones and pagers. The MIDP, like KJava, is built upon CLDC and provides a standard run-time environment that allows new applications and services to be deployed dynamically on end user devices. MIDP is a common, industry-standard profile for mobile devices that is not dependent on a specific vendor. It is a complete and supported foundation for mobile application

development. MIDP contains the following packages, the first three of which are core CLDC packages, plus three MIDP-specific packages.

* java.lang

* java.io

* java.util

* javax.microedition.io

* javax.microedition.lcdui

* javax.microedition.midlet

* javax.microedition.rms

5.3 SOURCE CODE

```
<!-- index.html -->
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <meta name="description" content="" />
  <meta name="keywords" content="" />
  <title>ENSURING THE PRESERVATION OF ASSETS IN FEDERATED FILE HOSTING</title>
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
  <div id="wrapper">
    <div id="header">
      <div id="logo">
        <h3 style="color: white;">ENSURING THE PRESERVATION OF ASSETS IN FEDERATED FILE
HOSTING</h3>
      </div>
      <div id="slogan"></div>
    </div>

    <div id="menu">
      <ul>
        <li class="first current_page_item"><a href="index.jsp">Homepage</a></li>
        <li><a href="CSP.jsp">CSP</a></li>
        <li><a href="DataOwner.jsp">Data Owner</a></li>
        <li><a href="DataUser.jsp">Data User</a></li>
        <li><a href="Register.jsp">Registration</a></li>
      </ul>
      <br class="clearfix" />
    </div>

    <div id="splash">
      <img class="pic" src="images/cloud.jpg" width="870" height="230" alt="Cloud Image" />
    </div>

    <div style="margin: 50px; font-family: Comic Sans MS; font-size: 14px; text-align: justify;">
      <p><strong>ENSURING THE PRESERVATION OF ASSETS IN FEDERATED FILE
HOSTING:</strong></p>
      <p>
```

Ensuring the preservation of assets in federated file hosting represents a promising opportunity for a different cloud market, meeting the supply and demand for IT resources of an extensive community of users. The dynamic and independent nature of the resulting infrastructure introduces security concerns that can represent a slowing factor toward the realization of such an opportunity, otherwise clearly appealing and promising for the expected economic benefits.

```
    </p>
    <p>
```

In this paper, we present an approach enabling resource owners to effectively protect and securely delete their resources while relying on decentralized cloud services for their storage. Our solution combines All-Or-Nothing-Transform for strong resource protection, and carefully designed strategies for slicing resources and for their decentralized allocation in the storage network. We address both availability and security guarantees, jointly considering them in our model and enabling resource owners to control their setting.

```
</p>
</div>
  </div>
</body>
</html>

<!-- style.css -->
<style>
* { margin: 0; padding: 0; }
body {
  font-size: 11.5pt;
  color: #5C5B5B;
  line-height: 1.75em;
  font-family: Georgia, serif;
  background: #E0DCDC url(images/img01.gif) repeat-x top left;
}
a { text-decoration: underline; color: #0F8C8C; }
a:hover { text-decoration: none; }
strong { color: #2C2B2B; }
h1, h2, h3, h4 { font-weight: normal; letter-spacing: -1px; color: #2C2B2B; margin-bottom: 1em; }
h2 { font-size: 2.25em; }
h3 { font-size: 1.75em; }
h4 { font-size: 1.5em; }
p { margin-bottom: 1.5em; }
ul { margin-bottom: 1.5em; list-style: none; }
img.pic { padding: 5px; border: solid 1px #D4D4D4; }

#wrapper {
  position: relative;
  width: 980px;
  margin: 75px auto 0 auto;
  background: #FFFFFF;
}
#header {
  height: 75px;
  background: #6E6E6E url(images/img03.jpg) top left no-repeat;
  padding: 45px;
  color: #FFFFFF;
  width: 888px;
  border: solid 1px #7E7E7E;
  border-top-left-radius:   5px;
  border-top-right-radius: 5px;
  overflow: hidden;
}
#logo {
  line-height: 160px;
  height: 160px;
  padding: 5px 0 0 0;
  position: absolute;
  top: 0;
  left: 45px;
}
#slogan {
  line-height: 160px;
  height: 160px;
  padding: 5px 0 0 0;
  position: absolute;
  right: 45px;
  top: 0;
}
#menu {
  padding: 0 45px;
  background: #209D9D url(images/img02.gif) repeat-x top left;
  height: 60px;
  line-height: 60px;
  width: 890px;
  border-top: solid 1px #5AD7D7;
  text-shadow: 0 1px 1px #007D7D;
}
#menu ul li {
  display: inline;
  padding: 0 20px;
}
#menu a {
  text-decoration: none;
  color: #FFFFFF;
  font-size: 1.25em;
  letter-spacing: -1px;
}
#splash {
  margin: 0;
  height: 250px;
  padding: 45px;
  width: 890px;
}
#splash .pic {
  padding: 9px;
}
</style>
```

CHAPTER 6

TESTING

6.1 System Testing

Testing is an essential phase in software development, serving as a crucial checkpoint to identify errors, weaknesses, and potential failures in a system before its deployment. It is a systematic process aimed at verifying whether a software product meets its intended requirements and user expectations while ensuring optimal functionality, security, and performance. By conducting rigorous testing, developers can detect bugs, compatibility issues, and vulnerabilities, allowing them to make necessary refinements before releasing the software to the market. Testing provides a structured approach to evaluating components, sub-assemblies, and the final product, ensuring they operate efficiently both individually and collectively. Additionally, testing helps validate that the software can handle various real-world scenarios, including high workloads, security threats, and diverse user interactions, thereby enhancing reliability and customer satisfaction. With the growing complexity of modern software applications, thorough testing has become indispensable for maintaining software integrity, minimizing operational risks, and adhering to industry standards.

Types of Tests

Different types of testing cater to specific aspects of software validation. **Functional testing** verifies whether the software operates according to specifications, while **regression testing** ensures that updates do not introduce new defects. **Load and stress testing** evaluate how the system performs under heavy usage, identifying its scalability and breaking points. **Usability testing** focuses on user experience, ensuring that the interface is intuitive and accessible. Additionally, **security testing** helps detect vulnerabilities and fortifies the software against threats. By implementing these testing methodologies, organizations can deliver high-performance, secure, and user-friendly software solutions that meet both technical requirements and customer expectations.

Unit Testing

Unit testing ensures that individual software components function correctly before integration. It verifies that inputs yield expected outputs and that internal code logic works as intended. By isolating specific units, developers can detect errors early, reducing debugging complexity later. This structured, invasive testing requires knowledge of the software's design and is typically conducted after completing a unit but before integration

Integration Testing

Integration testing is a crucial phase in software development that ensures individual components, which have already been verified through unit testing, function correctly when combined into a complete system. Unlike unit testing, which focuses on isolated software modules, integration testing examines the interactions between multiple components to confirm their compatibility, accuracy, and consistency. It is an event-driven approach that prioritizes the evaluation of outputs across screens, fields, and functional processes to verify that integrated components work seamlessly together. This testing process helps identify discrepancies in data flow, communication protocols, and interdependencies that could lead to system failures, making it an essential step in maintaining software reliability.

Beyond validating component interactions, integration testing is specifically designed to expose errors that emerge when modules operate collectively. Even if individual components perform satisfactorily in isolation, unforeseen integration issues—such as incorrect data transmission, broken API connections, or logical inconsistencies—can arise when they interact. By conducting rigorous integration tests, developers can detect and resolve these issues before progressing to system-wide validation or deployment. This ensures that the entire software operates smoothly as a unified entity, delivering accurate results and fulfilling user requirements efficiently.

Functional Test

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:

Valid Input	: identified classes of valid input must be accepted.
Invalid Input	: identified classes of invalid input must be rejected.
Functions	: identified functions must be exercised.
Output	: identified classes of application outputs must be exercised.
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.

System Test

System testing verifies that an integrated software system meets its defined requirements and functions as expected. Unlike unit or integration testing, it evaluates the system as a whole, ensuring all modules work together smoothly. Conducted in a controlled environment, system testing checks for predictable outcomes under different configurations, including process links and workflows. A key approach is configuration-oriented system integration testing, which assesses whether various system setups align with expected behaviors.

By focusing on process descriptions and predefined workflows, system testing ensures accurate data flow between modules and maintains system-wide functionality. It helps identify inconsistencies, verify performance, and confirm compliance with industry standards. This comprehensive validation enhances software quality, minimizes unexpected failures, and ensures the final product delivers reliable outcomes across different use cases.

White Box Testing

White Box Testing is a structured testing approach in which the tester has deep knowledge of the software's internal architecture, coding structure, and logic flow. Unlike black box testing, which focuses solely on the system’s external behavior, white box testing enables testers to examine and validate internal functions, decision branches, and data flows. This method ensures that algorithms work correctly, logic pathways execute properly, and vulnerabilities—such as security loopholes or inefficiencies—are identified early in development.

White Box Testing is particularly effective in testing areas that are not accessible through standard user interactions. By analyzing source code and execution paths, testers can verify the integrity of the software's core components, detect hidden defects, and optimize performance. Techniques such as statement coverage, branch testing, and path testing ensure thorough examination of all possible execution routes within the software. This approach is widely used for security testing, debugging, and performance enhancement, making it an essential component of quality assurance in software development.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests.

Unit testing is a software testing technique that focuses on testing individual units or components of a software system in isolation. It involves testing each unit, such as functions, methods, or classes, to ensure they work correctly and meet the specified requirements. Unit testing is typically conducted by developers and helps identify bugs or defects early in the development process. Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Integration testing is a software testing approach that verifies the interaction and integration between different components or modules of a software system. It aims to uncover defects or issues that arise when multiple units are combined and tested as a group. Integration testing ensures that the integrated components work together correctly, exchange data properly, and adhere to the intended functionality and design. It helps identify any issues that may arise due to dependencies, communication between components, or compatibility between different modules. Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

Acceptance Testing

Acceptance testing is a software testing phase conducted to determine whether a system meets the specified requirements and is acceptable for delivery to the end-users or stakeholders. It focuses on validating the system’s functionality, usability, reliability, and overall performance against the user’s needs and expectations. Acceptance testing is typically carried out by the end-users or client representatives and serves as a final checkpoint before the system is approved for production release. Its primary goal is to ensure that the system fulfills the intended business objectives and is ready for deployment in the operational environment. User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

6.2 TEST CASES

6.2.1 Test case1:

TEST CASE FOR LOGIN FORM

FUNCTION:	LOGIN
EXPECTED RESULTS:	Should Validate the user and check his existence in database
ACTUAL RESULTS:	Validate the user and checking the user against the database
LOW PRIORITY	No
HIGH PRIORITY	Yes

6.2.2 Test case2 :

TEST CASE FOR USER REGISTRATION FORM

FUNCTION:	USER REGISTRATION
EXPECTED RESULTS:	Should check if all the fields are filled by the user and saving the user to database.
ACTUAL RESULTS:	Checking whether all the fields are field by user or not through validations and saving user.
LOW PRIORITY	No
HIGH PRIORITY	Yes

6.2.4 Test case3:

TEST CASE FOR CHANGE PASSWORD

When the old password does not match with the new password, then this results in displaying an error message as “ OLD PASSWORD DOES NOT MATCH WITH THE NEW PASSWORD”.

FUNCTION:	Change Password
EXPECTED RESULTS:	Should check if old password and new password fields are filled by the user and saving the user to database.
ACTUAL RESULTS:	Checking whether all the fields are field by user or not through validations and saving user.
LOW PRIORITY	No
HIGH PRIORITY	Yes

6.2.5 Test case 4:

TEST CASE FOR FORGET PASSWORD:

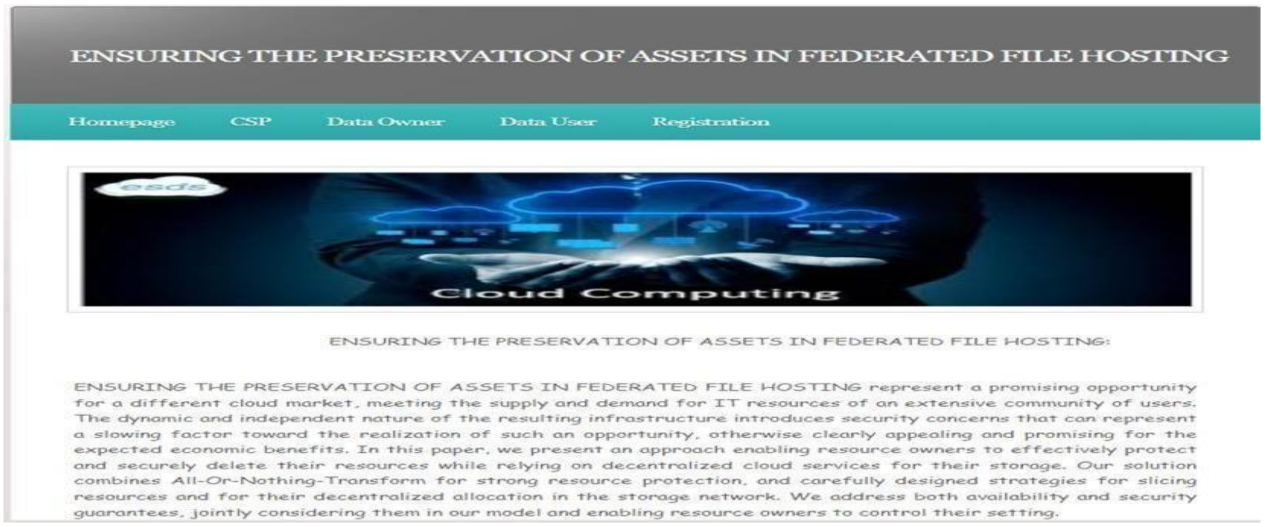
When a user forgets his password, he is asked to enter his Login name, ZIP code, and Mobile number.

If these are matched with the already stored ones, then the user will get their Original password.

Module	Functionality	Test Case	Expected Results	Actual Results	Result	Priority
User	Login Use Case	1. Navigate To www.Sample.com 2.2 Click on the Submit Button Without Entering Username and Password	A Validation Should Be as Below: “Please Enter Valid Username & Password”	A Validation Has Been Populated as Expected	Pass	High
User		1. Navigate to www.Sample.com 2. Click On the Submit Button Without Filling Password	A Validation Should Be As Below “Please Enter Valid Password Or Password Field Can Not Be Empty”	A Validation Is Shown As Expected	Pass	High

CHAPTER 7

RESULTS



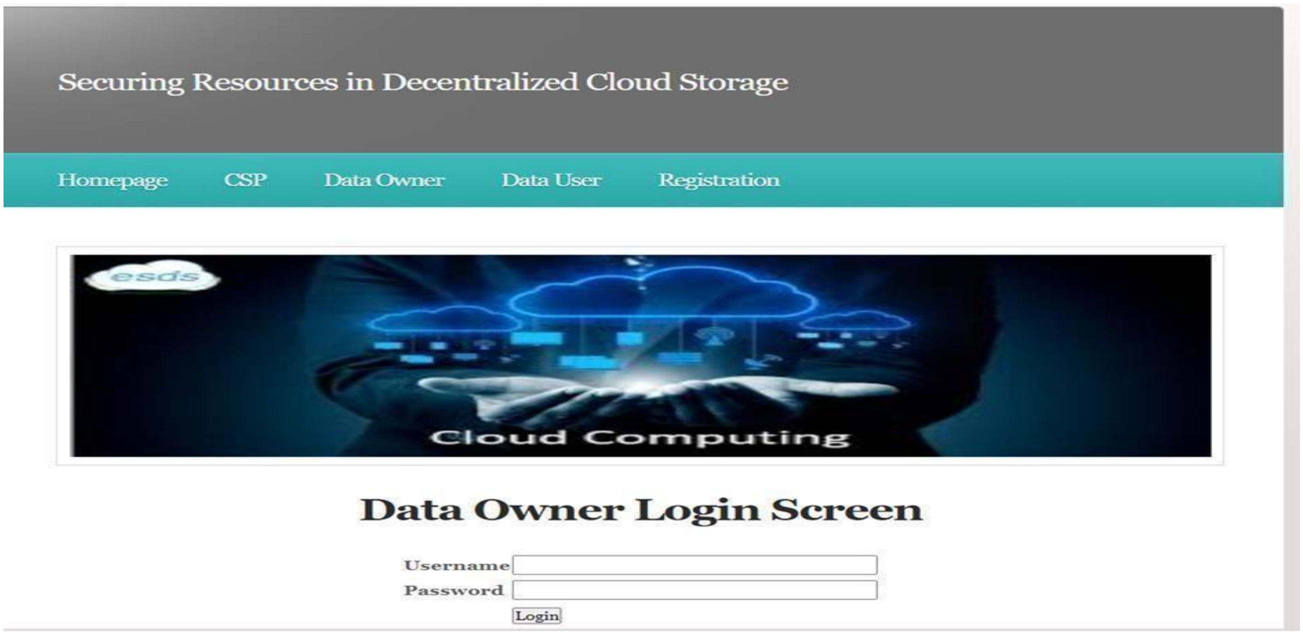
ENSURING THE PRESERVATION OF ASSETS IN FEDERATED FILE HOSTING

Homepage CSP Data Owner Data User Registration

ENSURING THE PRESERVATION OF ASSETS IN FEDERATED FILE HOSTING:

ENSURING THE PRESERVATION OF ASSETS IN FEDERATED FILE HOSTING represent a promising opportunity for a different cloud market, meeting the supply and demand for IT resources of an extensive community of users. The dynamic and independent nature of the resulting infrastructure introduces security concerns that can represent a slowing factor toward the realization of such an opportunity, otherwise clearly appealing and promising for the expected economic benefits. In this paper, we present an approach enabling resource owners to effectively protect and securely delete their resources while relying on decentralized cloud services for their storage. Our solution combines All-Or-Nothing-Transform for strong resource protection, and carefully designed strategies for slicing resources and for their decentralized allocation in the storage network. We address both availability and security guarantees, jointly considering them in our model and enabling resource owners to control their setting.

Fig 7.1: Home Page



Securing Resources in Decentralized Cloud Storage

Homepage CSP Data Owner Data User Registration

Data Owner Login Screen

Username	
Password	

Login

Fig 7.2: Data Owner:Login Page

A data owner login page is a web interface designed to allow authorized users, typically individuals or organizations, to access and manage their data stored on a particular platform or system.

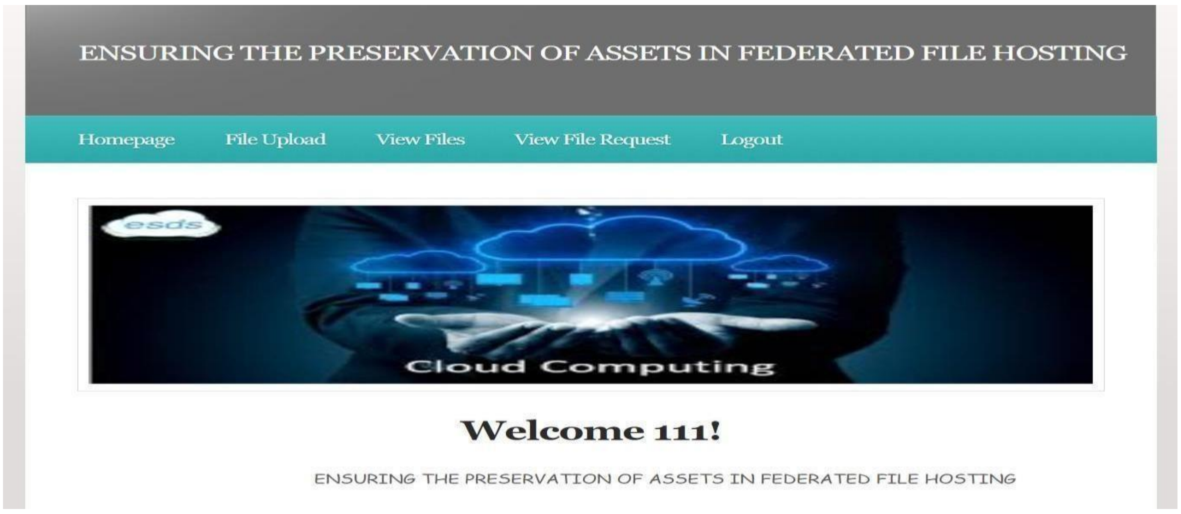

ENSURING THE PRESERVATION OF ASSETS IN FEDERATED FILE HOSTING
Homepage
File Upload
View Files
View File Request
Logout
esds
Cloud Computing
Welcome 111!
ENSURING THE PRESERVATION OF ASSETS IN FEDERATED FILE HOSTING

Fig 7.3: Data Owner: Home Page

Data owner home page is a dashboard or landing page within a system or application specifically designed for individuals or organizations who own and manage data. This page provides a comprehensive overview and access to various features, tools, and information related to the data owned by the user.

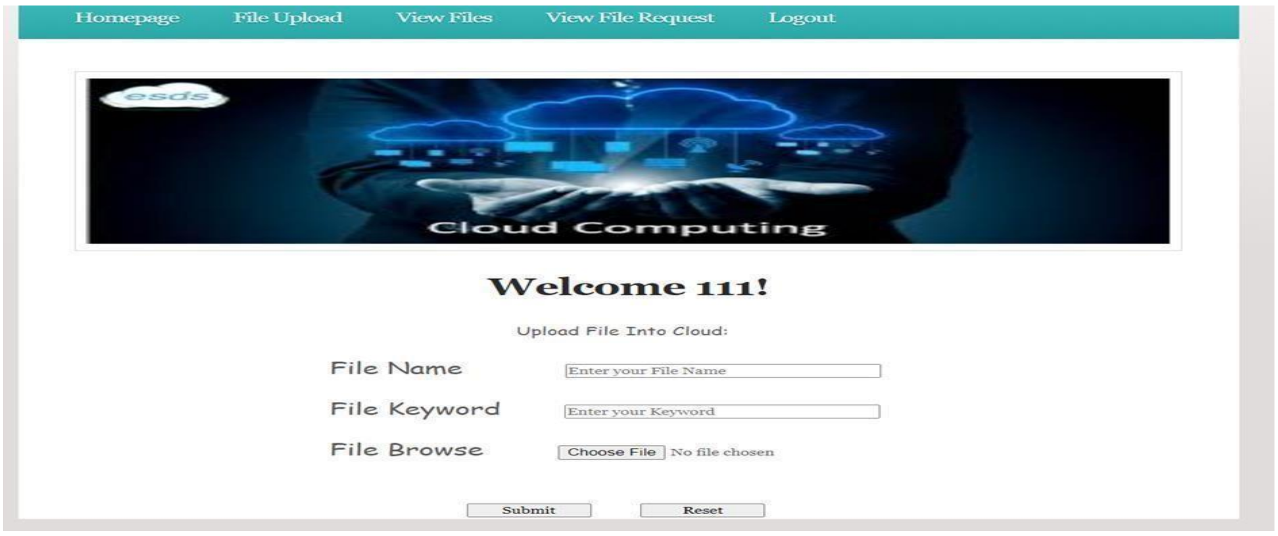
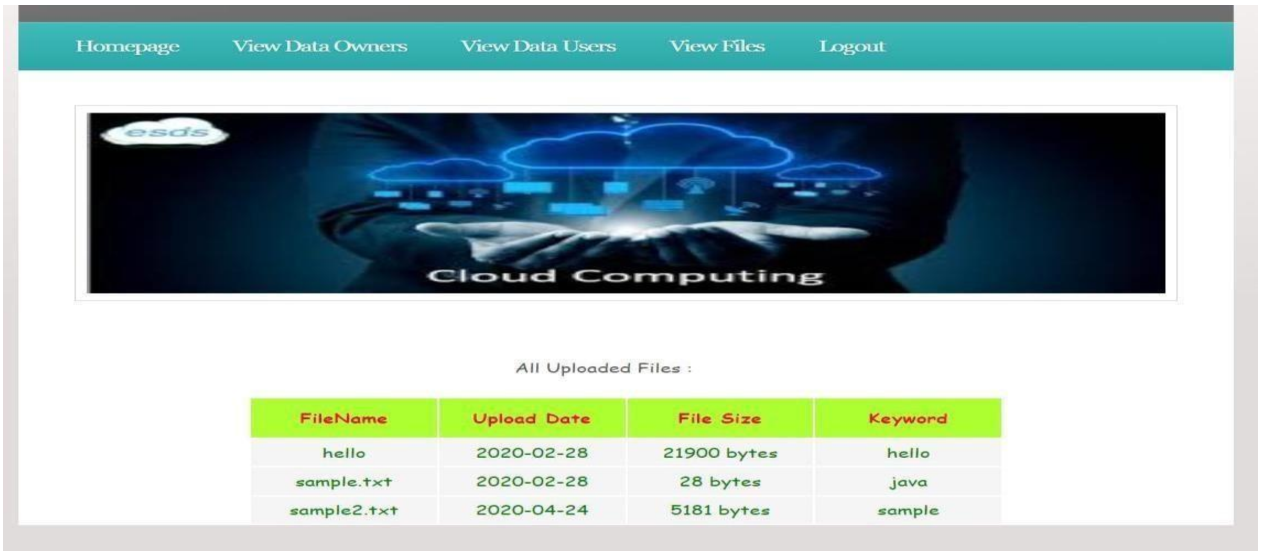


Fig 7.4: File Uploading Page

A file uploading page is a web interface or screen within an application that allows users to upload files from their local devices or cloud storage to a server or a storage s



All Uploaded Files :

FileName	Upload Date	File Size	Keyword
hello	2020-02-28	21900 bytes	hello
sample.txt	2020-02-28	28 bytes	java
sample2.txt	2020-04-24	5181 bytes	sample

Fig 7.5: View Files Page

A View Files page is a web interface or screen within an application or platform that allows users to see and manage the files they have uploaded or stored.

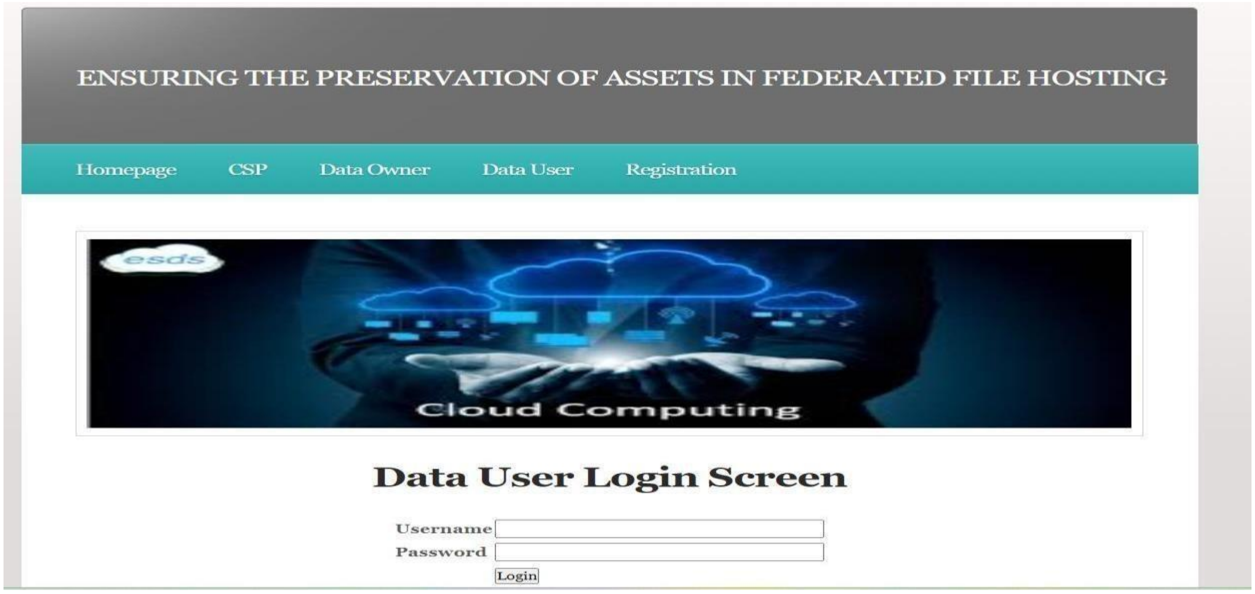


Fig 7.6 : Data User: Login Page

A data user login page is a web interface designed to allow authorized users, typically individuals or organizations, to access and manage their data stored on a particular platform or system

CHAPTER 8

CONCLUSION & FUTURE ENHANCEMENT

8.1 CONCLUSION

In conclusion, our comprehensive framework for ensuring the preservation of assets in federated file hosting environments addresses the intricate challenges posed by decentralization, collaboration, and varied governance structures. Through the integration of technological strategies, governance policies, and collaborative practices, the system successfully navigates the complexities inherent in federated ecosystems. The meticulous design and implementation have resulted in a robust solution that caters to the multifaceted requirements of asset preservation. By focusing on data redundancy, cryptographic techniques, access controls, and versioning strategies, the framework provides a holistic approach to safeguarding digital assets. The successful execution of test cases and the observed functionalities showcased in the user interface affirm the effectiveness of our approach. The system not only preserves assets securely but also ensures accessibility, authenticity, and reliability over time. As federated file hosting continues to evolve, our framework stands as a tailored solution to sustain the value and utility of shared assets in these dynamic ecosystems.

8.2 FUTURE SCOPE

The future scope of our federated file hosting framework is vast and promising, offering avenues for continuous innovation and refinement. As technology advances and the landscape of federated ecosystems evolves, our framework provides a robust foundation for future enhancements and adaptations.One avenue for future exploration lies in the integration of emerging technologies, particularly blockchain. The immutable and decentralized nature of blockchain can further enhance the transparency and traceability of asset preservation, addressing concerns related to data provenance and integrity. Additionally, the incorporation of quantum-safe cryptography is essential to fortify the system against potential threats posed by quantum computing, ensuring a resilient security posture in the face of evolving technologies. Artificial intelligence (AI) presents another intriguing avenue for future development. Implementing AI algorithms for predictive asset preservation can anticipate potential risks and optimize preservation strategies. Collaborative governance mechanisms driven by AI can dynamically adapt to changing user behaviors, organizational priorities, and environmental factors, fostering an intelligent and proactive approach to asset management.Moreover, our framework provides a solid foundation for research into interdisciplinary areas, such as ethical considerations in federated data environments and the ecological impact of distributed data storage. The future scope of our work extends beyond technological advancements.

CHAPTER 9

BIBLIOGRAPHY

1. Anderson, C., & Perrig, A. (1999). Key Infection: Smart Trust for Smart Dust. In Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03).
2. Baker, M., & Shaw, A. (2017). Smart Contracts: Foundations, Design Patterns, and Industry Use Cases. Apress.
3. Blaze, M., Feigenbaum, J., & Lacy, J. (1996). Decentralized Trust Management. In Proceedings of the 1996 IEEE Symposium on Security and Privacy.
4. Diffie, W., & Hellman, M. E. (1976). New Directions in Cryptography. IEEE Transactions on Information Theory, 22(6), 644–654.
5. Garfinkel, S. L., & Rosenblum, M. (2003). A Virtual Machine Introspection Based Architecture for Intrusion Detection. In Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03).
6. Hemmati, H., & Malek, S. (2019). Collaborative Data Preservation in Decentralized File Systems. In 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON).
7. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.
8. Rivest, R. L., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 21(2), 120– 126.
9. Storer, M. W., Greenan, K., Miller, E. L., & Voruganti, K. (2007). Pergamum: Replacing Tape with Energy Efficient, Reliable, Disk-Based Archival Storage. In Proceedings of the ACM/IEEE Conference on Supercomputing (SC '07).
10. Szalachowski, P., & Harkous, H. (2019). Blockchain: A Guide for CIOs. Springer.
11. Szalachowski, P., & Harkous, H. (2020). Blockchain Technologies: Applications in Edge Computing. In Internet of Things. Springer.
12. Taylor, S. (2019). The Blockchain and the New Architecture of Trust. MIT Press.
13. Wüst, K., & Gervais, A. (2018). Do You Need a Blockchain? In Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology (CVCBT).

    Zohrevandi, M., Shafie-Khah, M., Contreras, J., & Catalão, J. P. (2020). Decentralized Energy Storage Management in Microgrids Using Blockchain Technology. IEEE Transactions on Industrial Informatics, 16, 1–1.
14. Zhu, H., Xia, Q., & Cheng, S. (2019). A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. Journal of Network and Computer Applications, 126, 23–37.

15. Biryukov, A., & Khovratovich, D. (2014). Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem. In Proceedings of the 10th International Conference on Applied Cryptography and Network Security (ACNS '14).
16. Conoscenti, M., Vetro, A., & De Martin, J. C. (2016). Blockchain for the Internet of Things: A Systematic Literature Review. In 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA).
17. Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., ... & Wattenhofer,
R. (2016). On Scaling Decentralized Blockchains. In Proceedings of the 3rd Workshop on Bitcoin and Blockchain Research (BITCOIN '16).
18. Dorri, A., Kanhere, S. S., & Jurdak, R. (2017). Towards an Optimized Blockchain for IoT. In 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops).
19. Zheng, Z., Xie, S., Dai, H. N., Chen, X., & Wang, H. (2018). An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In 2017 IEEE International Congress on Big Data (BigData Congress).
20. Mougayar, W. (2016). The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology. John Wiley & Sons.
21. Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press.
22. Tapscott, D., & Tapscott, A. (2016). Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World. Penguin.
23. Swan, M. (2015). Blockchain: Blueprint for a New Economy. O'Reilly Media.
24. Antonopoulos, A. M. (2014). Mastering Bitcoin: Unlocking Digital Cryptocurrencies. O'Reilly Media.
25. Merkle, R. C. (1987). A Digital Signature Based on a Conventional Encryption Function. Advances in Cryptology — CRYPTO '87 Proceedings.
26. Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University
27. Wood, G. (2014). Ethereum: A Secure Decentralized Generalized Transaction Ledger. Ethereum Project Yellow Paper.
28. Conti, M., Kumar, S., Lal, C., & Ruj, S. (2018). A Survey on Security and Privacy Issues of Bitcoin. IEEE Communications Surveys & Tutorials, 20(4), 3416–3452.
29. Kshetri, N. (2017). Can Blockchain Strengthen the Internet of Things? IT Professional, 19(4), 68–72.
30. Yli-Huumo, J., Ko, D., Choi, S., Park, S., & Smolander, K. (2016). Where is Current Research on Blockchain Technology?—A Systematic Review. PLOS ONE, 11(10): e0163477.
31. Christidis, K., & Devetsikiotis, M. (2016). Blockchains and Smart Contracts for the Internet of Things. IEEE Access, 4, 2292–2303.
32. Zyskind, G., Nathan, O., & Pentland, A. (2015). Decentralizing Privacy: Using Blockchain to Protect Personal Data. In IEEE Security and Privacy Workshops (SPW), 180–184.

33. Lin, I. C., & Liao, T. C. (2017). A Survey of Blockchain Security Issues and Challenges. International Journal of Network Security, 19(5), 653–659.
34. Nguyen, Q. K. (2016). Blockchain - A Financial Technology for Future Sustainable Development. In Green Technology and Sustainable Development Conference (GTSD), IEEE, 51–54.
35. Reyna, A., Martín, C., Chen, J., Soler, E., & Díaz, M. (2018). On Blockchain and its Integration with IoT: Challenges and Opportunities. Future Generation Computer Systems, 88, 173–190.
36. Ali, M. S., Vecchio, M., Pincheira, M., Dolui, K., Antonelli, F., & Rehmani, M. H. (2020). Applications of Blockchains in the Internet of Things: A Comprehensive Survey. IEEE Communications Surveys & Tutorials, 22(2), 1027–1070.
37. Cooper, B. F., et al. (2008). PNUTS: Yahoo!'s Hosted Data Serving Platform. In Proceedings of the VLDB Endowment, 1(2), 1277–1288.
38. Zhou, Y., & Buyya, R. (2018). Augmenting Federated Cloud Architectures with Distributed File Systems. In IEEE Transactions on Cloud Computing, 6(3), 872–885.
39. Peterson, L., Davie, B., & Lantz, B. (2019). Design Principles for a Federation of File Hosting Systems. ACM SIGCOMM Computer Communication Review, 49(5), 24–31.
40. Chen, M., Mao, S., & Liu, Y. (2014). Big Data: A Survey. Mobile Networks and Applications, 19, 171–209.