

LOW COST DISTRIBUTED FILE SYSTEM

A PROJECT REPORT

Submitted by

MANIKANDAN R (923317104029)

ARUNPANDIYAN P (923317104004)

JAHEEN AFSAR SYED K S (923317104022)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

GOVERNMENT COLLEGE OF ENGINEERING

BODINAYAKANNUR – 625 582

ANNA UNIVERSITY : CHENNAI 600 025

MARCH-2021

ANNA UNIVERSITY : CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report “ **LOW COST DISTRIBUTED FILE SYSTEM** ” is the bonafide work of “ **MANIKANDAN R (923317104029)** and **ARUNPANDIYAN P (923317104004)** and **JAHEEN AFSAR SYED K S (923317104022)** ” who carried out the project work under my supervision during the Academic Year 2020-2021.

SIGNATURE

Dr.D.Mary Sugantharathnam, M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Computer Science and Engineering
Government College of Engineering
Bodinayakannur – 625 582

SIGNATURE

B. SIVARANJANI

SUPERVISOR

Computer Science and Engineering
Government College of Engineering
Bodinayakannur – 625 582

Submitted for **PROJECT (CS8811)** Viva Voice Examination held on..... at Government College of Engineering,Bodinayakannur.

INTERNALEXAMINER

EXTERNALEXAMINER

ACKNOWLEDGEMENT

We acknowledge with great pleasure, deep satisfaction and gratitude the contribution of many individuals in the successful completion of this project.

We express our profound gratitude to **Dr.S.JAYANTHI, Principal, Government College of Engineering, Bodinayakannur** for all the support and encouragement given to us throughout our project work.

We thank **Dr.D. MARY SUGANTHARATHNAM, Head of the Department, Department of Computer Science and Engineering, Government College of Engineering, Bodinayakannur** for her valuable suggestion throughout our project.

We express our gratitude to our Guide **B. SIVARANJANI, Assistant professor, Department of Computer Science and Engineering, Government College of Engineering, Bodinayakannur** for her guidance and help in doing this project work successfully.

ABSTRACT

Recently, the concept of distributed file system has been widely used in many Internet applications. A distributed file system integrates many file servers on the network and becomes an efficiency file system with large storage space. Users can access their files anytime and anywhere. However, the cost for building and maintaining these storage systems is high for many small enterprises and organizations. Thus, a reliable file system with low cost and large storage space is necessary for many users. In this project, we proposed a low cost distributed file system. The proposed file system stores the file replications in different network area. We also design a load detection module in the proposed system. The module will detect the load of each file servers and improve the overall system performance. The main purpose of the Distributed File System (DFS) is to allows users of physically distributed systems to share their data and resources by using a Common File System.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
I	ABSTRACT	ii
II	LIST OF FIGURES	vi
III	LIST OF SYMBOLS	vii
1	INTRODUCTION	
	1.1 INTRODUCTION TO DISTRIBUTED FILE SYSTEM	1
	1.1.1 Objectives	1
	1.1.2 Characteristics	1
	1.1.3 Basics	2
	1.1.4 Features of Proposed System	2
	1.1.5 Scope of the Project	2
2	LITERATURE REVIEW	3
3	SYSTEM ANALYSIS	
	3.1 EXISTING SYSTEM	5
	3.2 PROPOSED SYSTEM	6
4	SYSTEM DESIGN	
	4.1 SYSTEM ARCHITECTURE	7
5	SYSTEM SPECIFICATION	
	5.1 HARDWARE REQUIREMENT	8
	5.2 SOFTWARE REQUIREMENT	8

6	MODULE DESCRIPTION	9
	6.1 Connection Module	9
	6.2 Database for File System Mapping	9
	6.3 File Transfer Module	9
	6.4 Metadata Manager	10
7	SYSTEM IMPLEMENTATION	11
	7.1 MODULES IMPLEMENTATION	11
	7.1.1 Connection Phase	11
	7.1.2 Verification Phase	11
	7.1.3 Data Transfer Phase	11
	7.1.4 Data Integration Phase	12
	7.2 PROTOCOLS USED	13
	7.2.1 HTTP Overview	13
	7.2.2 HTTP Authentication	16
8	SOFTWARE TESTING	20
	8.1 TESTING METHODOLOGIES	20
	8.1.1 Unit Testing	20
	8.1.2 Integration Testing	20
	8.1.3 Validation Testing	21
	8.1.4 User Acceptance Testing	21
	8.1.5 White Box Testing	21
	8.1.6 Block Box Testing	22
9	SCREENSHOTS	23

10	CONCLUSION	48
11	FUTURE ENHANCEMENT	49
12	REFERENCES	50

LIST OF FIGURES

S. No	TITLE	PAGE NO
1	Existing System	6
2	Proposed System	7
3	System Architecture	8

LIST OF SYMBOLS

S.NO	ACRONYM	ABBREVIATION
1.	DFS	D istributed F ile S ystem
2.	OS	O perating S ystem
3.	HDFS	H adoop D istributed F ile S ystem
4.	GFS	G oogle F ile S ystem
5.	RAM	R andom A ccess M emory
6.	HTTP	H yper T ext T ransfer P rotocol

CHAPTER 1

INTRODUCTION

Distributed file system is a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources. The purpose is to promote sharing of dispersed files. A server interface is the set of file operations: create, read, update, delete etc on files. Client application provides interface to access files on servers. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

1.1.1 Objectives

The main objective of this research project will be to examine the critical features that make a strong Distributed file systems. Such features include Data consistency, Uniform access, Security, Reliability, Performance. To develop a distributed file system for small scale application using low cost hardware. To reach maximum efficiency for small scale applications. Developing simple interface to use the distributed storage. To reuse old computer hardware.

1.1.2 Characteristics

Unique Characteristics of Distributed File System:

Scalability

Security

Fault Tolerances

Performance and Data integrity

1.1.3 Basics

In this current distributed file system has high time consuming,high workload pressure and low security. But this file sharing process provide the easy interface, low time consuming,high security and no workload pressure. This online file sharing process is comfortable for many user.

1.1.4 Features of proposed system

Admin can add / update the details servers.

Only Admin can start the file sharing process.

User must enter their personal details of server before add servers.

The verification process must for add server process.

The Result will be client can access upload the files on the servers,create folders on the servers,delete files on the servers.

1.1.5 Scope of the project

The project scope is to provide a better and secure process for distributed file system. That project will be very useful for clients. The file is divide into server and saved to hash values.It will improve the Security of the process. By using this Security we can protect from any illegal activities. So, the proposed system is must need for the entire society.

CHAPTER 2

LITERATURE REVIEW

Chao et al [1] proposed a distributed file system, the General File System, which integrates free web spaces. Users may build a personal network file system by using their e-mail and FTP accounts. The General file system utilizes the convenience of web space. It integrates several heterogeneous spaces to reduce the maintenance cost. Although the storage space of the file system is provided by the public web space, the General file system guarantees the reliability and the security of user files. The Hadoop Distributed File System (HDFS) is integrated by many low cost computers. It is a distributed file system with high performance. To increase the access performance of the file system, HDFS replicates the files and stores most of these replications in the same network that is closest to the client. However, the crash of whole network will lose all of the files stored in this network. The HDFS namespace is hierarchy of files and directories. Everything is stored on the NameNode (metadata server) with attributes like permission, modification and access times including the locations of replicas of each block of the file. Everything is kept in RAM for fast servicing the client. Each block replica on a DataNode is represented by two files in the localhost's native file system. The first file contains the data itself and the second file is block's metadata including checksums. DataNode performs a handshake while startup and informs the NameNode about its presence. It sends the block report containing details of all the blocks maintained by it during handshake. Heartbeats are periodically sent to NameNode providing information about the capacity of the DataNode. These details are used while making allocation and load balancing decisions by the NameNode. It replies to the heartbeats in case it wants the DataNode to perform any specific operation.

Liu Jiang and et al [2] proposed the optimization of filesystem which provides a better performance based on small files. Farag Azzedin proposed a similar architecture with HDFS, but Name Node is distributed. Name Server is responsible for data classification and allocation of storage. Name Server used to support the Name Server works. Data Server placed under the Name Server, in a Data server is a type of image to save it. Data Server are not related, and do not know what kind of video is stored on others. To manage images, a fundamental approach to describe the data and then use the descriptive information for the implementation of the operation. Raw data, basic features, low-level features and semantic features are used to describe the image data. A tree structure is used to display image data. The root node, raw data refers to data files stored images. In the middle child of the root node, low-level features and key characteristics of states. Low-level features include image data features, such as color, texture, shape of images. Basic features include attributes such as name, type, author, and creation time. The leaf node, using the expression of semantic features that include good writer, explaining the topic, and a low-level features. HDFS implements single writer multiple reader model. Lease mechanism is used to avoid multiple clients to write at the same time. Lease is renewed through HeartBeats. Servers are stateful and uses buffer of size 64KB while writing. A pipeline is setup from client to the DataNodes. TCP like mechanism is implemented to achieve reliable writes. It takes some time before the data is visible to other clients to read. hflush operation is provided if data is required to be made visible instantly. Checksums are stored to ensure the integrity of the data on the local disk.

Aditya B. Patel, Manashvi Birla, Ushma Nair et al [3] HDFS provides a reliable, high performance distributed file system over many cheaper computers. It assumes that failure probability of these computers is very high. Thus, the design of the system has to consider the fault-tolerance issues. The system architecture of DHFS is distributed. It consists of one Name node and many Data node. The Name node manages metadata and file name space. The centralized management mechanism simplifies the entire filesystem structure. Besides, there are also much research that improve the HDFS

Metadata server stores the partition information, metadata, data server information, file attributes and the location information of replications. The metadata server also maintains the consistency of the replications. To improve the reliability of the proposed system, metadata servers are also divided into the active metadata server and the standby metadata server. The Google file system (GFS) is for distributed computing; and it is used for storing the large number of web files. The file update and consistence maintenance of GFS is based on the primary chunk server. To reduce the loading of the master server, the master server will assign a primary chunk server to update files and maintain the consistence of file copies when a client writes a file. The consistence checking will be done when the loading of the master server becomes light. A GFS cluster consists of single master node and multiple chunk servers. The master maintains all the system metadata including namespace, access control information, mapping from files to chunk and current location of chunks. It controls chunk lease management, garbage collection of orphaned chunks and chunk migration between chunk servers. The master periodically communicates with each chunk server in Heartbeat messages to give instructions and collect its state. Neither the client nor the ChunkServer caches the file data. Client caches offer little benefit because most applications streams through huge files. Not having them simplifies the client. ChunSservers need not cache file data because chunks are stored as local files and so Linux's buffer cache already keeps frequently accessed data in memory.

H.C. Chao, T.J. Liu, K.H. Chen and C.-R. Dow et al [4] NFS provides file services in a heterogeneous environment of different machines, operating systems and network architectures. This is achieved through the use of RPC primitives built on top of External Data Representation (XDR) protocol. It is mostly divided into two parts- mount protocol and fileaccess protocol. Mount protocol allows user to treat the remote file system locally whereas file access protocol enables reading, writing in a file, searching in a directory etc. NFS has 3 layered architecture as shown in figure 1. The top layer provides a clean interface to users to perform operations on files. Middle layer is Virtual File System (VFS). It activates filesystem specific operations for local requests and NFS protocol procedures

for handling remote requests. The bottom layer implements the NFS protocol. Every system has its own view of logical name structure. For I/O operations, each client communicates to the server directly. Pathname traversal is also performed client to server with no mediator. A directory lookup cache is maintained for efficient traversal. For security reasons, each server maintains an export list that specifies the local file systems it allows to export (get mount) along with the names of machines permitted to mount them. Cascading mounts does not exhibit transitive property and the security of the system is still preserved. The list is also used to notify the servers if any of the connecting server goes down. Only administrator/s has rights to change the export list. The prominent feature of NFS servers is that servers are stateless. Caching is used for efficient file operations but it is handled such that stateless property is preserved. The changes, therefore, may take some time to be visible to others. Overall it provides network transparency, fault tolerance to some extent but fail to provide location transparency, reliability.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTINGSYSTEM

In this section, we discuss the existing system. As it was shown in Figure , the existing system contains three components: metadata servers, data servers and access servers. Metadata servers record the partition information of the file system and the metadata of files. Data servers store the user file blocks. Access servers handle the user requests. Furthermore, to balance the loading of the file system, we design a load detection module. The load detection module detects the data server with high loading; and it checks if this high loading data server will decrease the overall system performance or not.

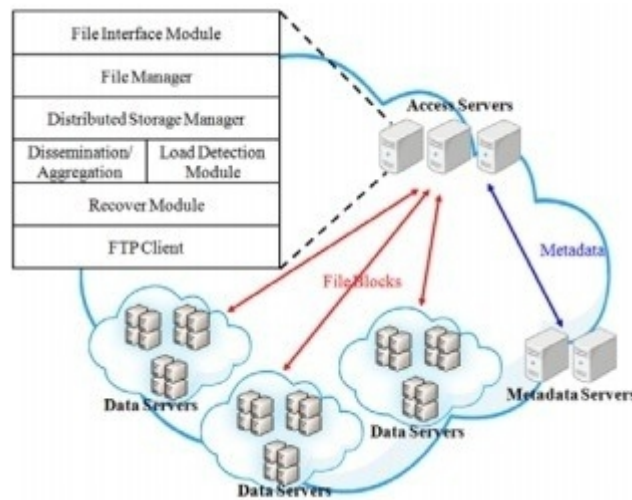


Fig 3.1 EXISTING SYSTEM

Disadvantages:

It requires three minimum types servers(metadata ,access,data server).

It is expensive and inefficient than the proposed system.

3.2 PROPOSEDSYSTEM

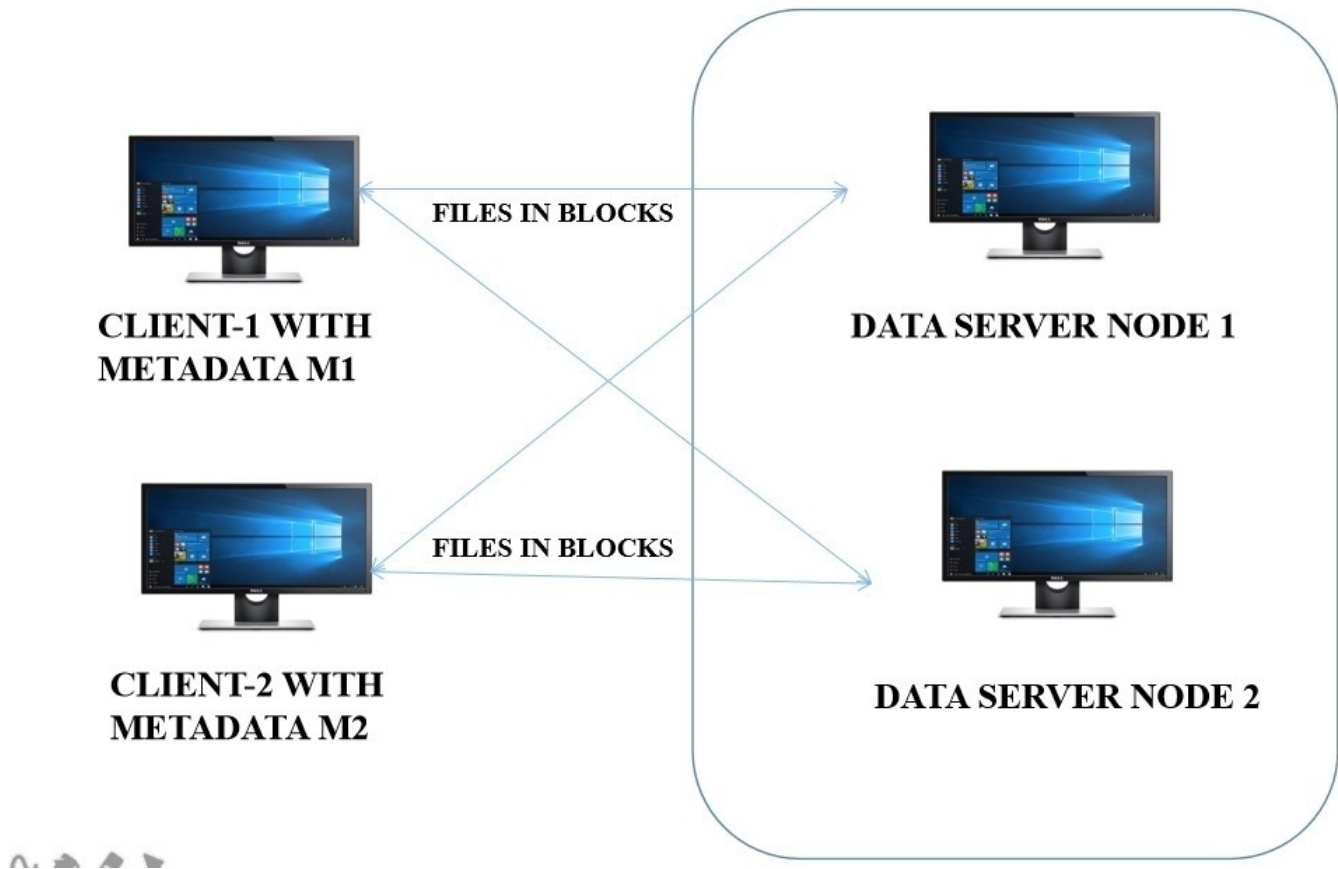


Fig 3.2 PROPOSED SYSTEM

Advantages:

It is secure and safe.

It has high flexible.

It has less number of server needed.

It easily scalable.

It less expensive.

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

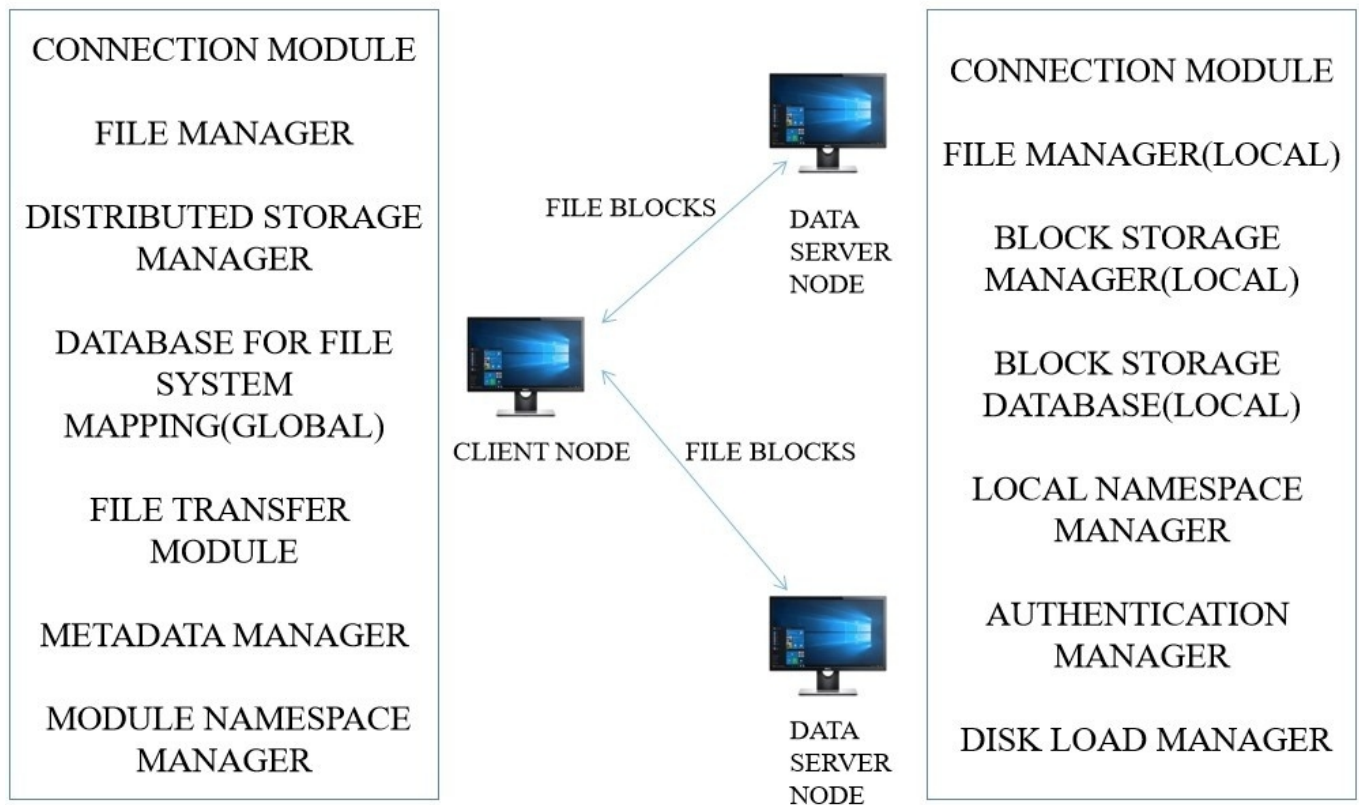


Fig 4.1 SYSTEM ARCHITECTURE

CHAPTER 5

SYSTEM SPECIFICATION

5.1 HARDWARE REQUIREMENT

Processor	: Intel Pentium and higher
Speed	: Network connection (10mbps minimum)
Ram	: 2GB and higher
Harddisk	: Free hard disk storage

5.2 SOFTWARE REQUIREMENT

Operating System

- Linux debianderivatives (recommended)
- Windows 10 edition (recommended)
- Mac Os High sierra and above (recommended)

CHAPTER 6

MODULE DESCRIPTION

6.1 CONNECTION MANAGER

This module is responsible for connecting server and client. This module maintain connection between server and client. All network traffic is handle by this module.

6.2 DATABASE FOR FILE SYSTEM MAPPING

This is database that maintains records of which blocks belong to which file and relations between files and folders. This module maintain 2 tables in the database (files and blocks tables). Blocks table have a foreign key on files table primary key.

6.3 FILE TRANSFER MODULE

The file transfer module is used to send or receive files between client and server using HTTP. In order to transfer as file blocks split into blocks of 1MB size and must be transferred to server through HTTP

6.4 METADATA MANAGER

Metadata server stores the partition information, metadata, data server information, file attributes and the location information of replications. The metadata server also maintains the consistency of the replications. To improve the reliability of the proposed system, metadata servers are also divided into the active metadata server and the standby metadataserver.

CHAPTER 7

SYSTEM IMPLEMENTATION

7.1 MODULE IMPLEMENTATION

7.1.1 ConnectionPhase

This is the initial phase of the system. During this phase the client initiates the connection to the server and checks its availability and capability. This phase is very essential for data transfer to take place.

7.1.2 VerificationPhase

This phase happens before files are transferred to servers as well as to clients. During this phase the client sends the access token sent by the server during authentication to verify the client. This securely allows access only to respective clients only.

7.1.3 Data TransferPhase

When the client sends the files, the client splits the file into blocks of 1MB size and sends it to the server with maximum storage size. The server on the other hand stores the files on the directory specific to that user.

7.1.4 Data Integration Phase

On counting form only election commission authorized user can login with the secure id and password if both id and password are correct then voting process will be continuing. At the end of the day the result will be announced.

7.2 PROTOCOLSUSED

7.2.1 HTTP OVERVIEW

The Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a webbrowser.

Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989. Development of early HTTP Requests for Comments (RFCs) was a coordinated effort by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), with work later moving to the IETF.

HTTP/1.1 was first documented in RFC 2068 in 1997, and as of 2021, it (plus older versions) is less popular (used by less than 37% of web sites; it's always a backup protocol) for web serving than its successors. That specification was obsoleted by RFC 2616 in 1999, which was likewise replaced by the RFC 7230 family of RFCs in 2014.

HTTP/2 is a more efficient expression of HTTP's semantics "on the wire", and was published in 2015, and is used by over 50% of websites; it is now supported by virtually all web browsers and major web servers over Transport Layer Security (TLS) using an Application-Layer Protocol Negotiation (ALPN) extension where TLS 1.2 or newer is required.

HTTP/3 is the proposed successor to HTTP/2, which is already in use by 13.3% of websites; and is used by over 7.3% of desktop computers (enabled by default in latest macOS), using UDP instead of TCP for the underlying transport protocol. Like HTTP/2, it does not obsolete previous major versions of the protocol. Support for HTTP/3 was added to Cloudflare and Google Chrome in September 2019 and can be enabled in the stable versions of Chrome and Firefox.

HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the client and an application running on a computer hosting a website may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its messagebody.

A web browser is an example of a user agent (UA). Other types of user agent include the indexing software used by search providers (web crawlers), voice browsers, mobile apps, and other software that accesses, consumes, or displays web content.

HTTP is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time. Web browsers cache previously accessed web resources and reuse them, when possible, to reduce network traffic. HTTP proxy servers at private network boundaries can facilitate communication for clients without a globally routable address, by relaying messages with external servers.

HTTP is an application layer protocol designed within the framework of the Internet protocol suite. Its definition presumes an underlying and reliable transport layer protocol, and Transmission Control Protocol (TCP) is commonly used. However, HTTP can be adapted to use unreliable protocols such as the User Datagram Protocol (UDP), for example in HTTPU and Simple Service Discovery Protocol(SSDP).

HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes http and https. As defined in RFC 3986, URIs are encoded as hyperlinks in HTML documents, so as to form interlinked hypertext documents.

HTTP/1.1 is a revision of the original HTTP (HTTP/1.0). In HTTP/1.0 a separate connection to the same server is made for every resource request. HTTP/1.1 can reuse a connection multiple times to download images, scripts, stylesheets, etc after the page has been delivered. HTTP/1.1 communications therefore experience less latency as the establishment of TCP connections presents considerable overhead.

7.2.2 HTTP AUTHENTICATION

HTTP supports the use of several authentication mechanisms to control access to pages and other resources. These mechanisms are all based around the use of the 401 status code and the WWW-Authenticate response header.

The most widely used HTTP authentication mechanisms are:

Basic

The client sends the user name and password as unencrypted base64 encoded text. It should only be used with HTTPS, as the password can be easily captured and reused over HTTP.

Digest

The client sends a hashed form of the password to the server. Although, the password cannot be captured over HTTP, it may be possible to replay requests using the hashed password.

NTLM

This uses a secure challenge/response mechanism that prevents password capture or replay attacks over HTTP. However, the authentication is per connection and will only work with HTTP/1.1 persistent connections. For this reason, it may not work through all HTTP proxies and can introduce large numbers of network roundtrips if connections are regularly closed by the webserver.

TOKEN BASED AUTHENTICATION

A token is a piece of data that has no meaning or use on its own, but combined with the correct tokenization system, becomes a vital player in securing your application. Token based authentication works by ensuring that each request to a server is accompanied by a signed token which the server verifies for authenticity and only then responds to the request. JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained method for securely transmitting information between parties encoded as a JSON object. JWT has gained mass popularity due to its compact size which allows tokens to be easily transmitted via query strings, header attributes and within the body of a POST request.

The use of tokens has many benefits compared to traditional methods such as cookies.

1. Tokens are stateless. The token is self-contained and contains all the information it needs for authentication. This is great for scalability as it frees your server from having to store session state.
2. Tokens can be generated from anywhere. Token generation is decoupled from token verification allowing you the option to handle the signing of tokens on a separate server or even through a different company such as Auth0.
3. Fine-grained access control. Within the token payload you can easily specify user roles and permissions as well as resources that the user can access.

These are just some of the benefits JSON Web Tokens provide. To learn more check out this blog post that takes a deeper dive and compares tokens to cookies for managing authentication.

ANATOMY OF A JSON WEB TOKEN

A JSON Web Token consists of three parts: Header, Payload and Signature. The header and payload are Base64 encoded, then concatenated by a period, finally the result is algorithmically signed producing a token in the form of header.claims.signature. The header consists of metadata including the type of token and the hashing algorithm used to sign the token. The payload contains the claims data that the token is encoding

The final result looks like:

**eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJtZXNzYWdlIjoiaSldUIFJ1bGVzISI
sImldhdCI6MTQ1OTQ0ODExOSwiZXhwIjoxNDU5NDU0NTE5fQ.VBD5b73C75os
bmwwshQNRC7frWUYrqaTjTpza2y4.**

Tokens are signed to protect against manipulation, they are not encrypted. What this means is that a token can be easily decoded and its contents revealed. If we navigate over the jwt.io, and paste the above token, we'll be able to read the header and payload – but without the correct secret, the token is useless and we see the message “Invalid Signature.” If we add the correct secret, in this example, the string `L3@RNJWT`, we'll now see a message saying “SignatureVerified.”

In a real world scenario, a client would make a request to the server and pass the token with the request. The server would attempt to verify the token and, if successful, would continue processing the request. If the server could not verify the token, the server would send a 401 Unauthorized and a message saying that the request could not be processed as authorization could not be verified.

JSON WEB TOKEN BEST PRACTICES

Before we actually get to implementing JWT, let's cover some best practices to ensure token based authentication is properly implemented in your application.

1. Keep it secret. Keep it safe. The signing key should be treated like any other credentials and revealed only to services that absolutely need it.

2. Do not add sensitive data to the payload. Tokens are signed to protect against manipulation and are easily decoded. Add the bare minimum number of claims to the payload for best performance and security.

3. Give tokens an expiration. Technically, once a token is signed – it is valid forever – unless the signing key is changed or expiration explicitly set. This could pose potential issues so have a strategy for expiring and/or revoking tokens.

4. Embrace HTTPS. Do not send tokens over non-HTTPS connections as those requests can be intercepted and tokens compromised.

5. Consider all of your authorization use cases. Adding a secondary token verification system that ensure tokens were generated from your server, for example, may not be common practice, but may be necessary to meet your requirements.

USE CASES FOR TOKEN BASED AUTHENTICATION

We've seen how easy it is to implement JWT authentication and secure our API. To conclude, let's examine use cases where token based authentication is best suited for.

1. Platform-as-a-Service Applications – exposing RESTful APIs that will be consumed by a variety of frameworks and clients.

2. Mobile Apps and web apps – implementing native or hybrid mobile and web apps that interact with your services.

CHAPTER 8

SOFTWARE TESTING

8.1 TESTING METHODOLOGIES

Testing is the stage of implementation of which aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct the goal will be achieved. The candidates system subject to a variety of tests. Online response, volume, stress, recovery, security and usability tests. A series of testing are performed for the proposed system before the system is ready for user acceptance testing.

8.1.1 Unit Testing

The procedure level testing is made first. By giving improper inputs, the errors occurred are noted and eliminated. Then the web form level is made.

8.1.2 Integration Testing

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. Thus the system testing is a confirmation that all its correct and an opportunity to show the user that the system works.

8.1.3 ValidationTesting

The final step involves validation testing which determines whether the software function as the user expected. The end-user rather than the system developer conduct this test most software developers as a process called “Alpha and Beta test” to uncover that only the end user seems able to find. The compilation of the entire project is based on the full satisfaction of the end users.

8.1.4 AcceptanceTesting

Acceptance testing can be defined in many ways, but a simple definition is the succeed when the software functions in a manner that can be reasonable expected by the customer. After the acceptance test has been conducted, one of the two possible conditions exists. This is to fine whether the inputs are accepted by the database or other validations. For example accept only numbers in the numeric field, date format data in the date field. Also the null check for the not null fields. If any error occurs then show the error messages. The function of performance characteristics to specification and is accepted. A deviation from specification is uncovered and a deficiency list iscreated.

8.1.5 White BoxTesting

White box testing, sometimes called “Glass-box testing”. Using white box testing methods, the following tests were made on the system,All independent paths with in a module have been exercised at least once.All logical decisions were checked for the true and false side of the values. All loops were executed to check their boundary values.Internal data-structure was tested for their validity.

8.1.6 Black Box Testing

Black box testing focuses on the functional requirements of the software. That is black box testing enables the software engineer to drive a set of input conditions that will fully exercise the requirements for a program. Black box testing is not an alternative for white box testing techniques. Rather, it is a complementary approach that is likely to uncover different class of errors. Black box testing attempts to find errors in the following categories:

- Interface errors.

- Performances in data structures or external database access. Performance errors.

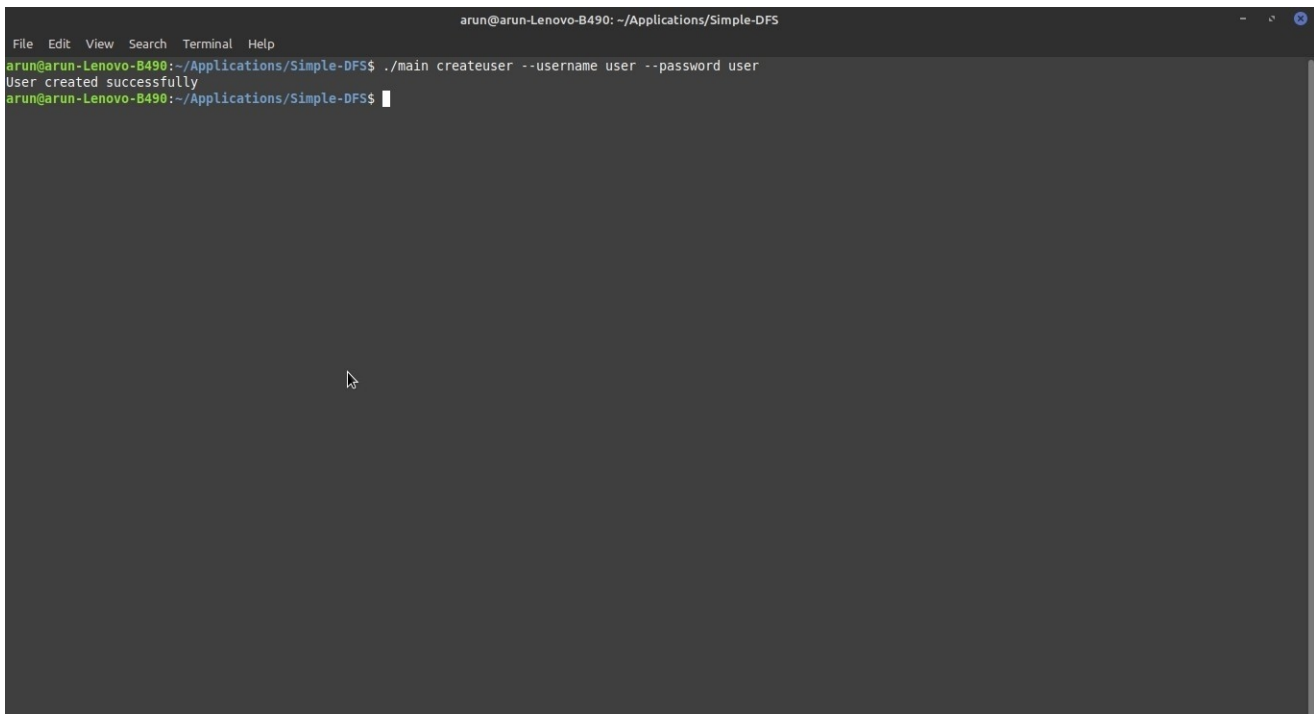
- Initialization and termination errors. Incorrect or missing functions.

CHAPTER 9

SCREENSHOTS

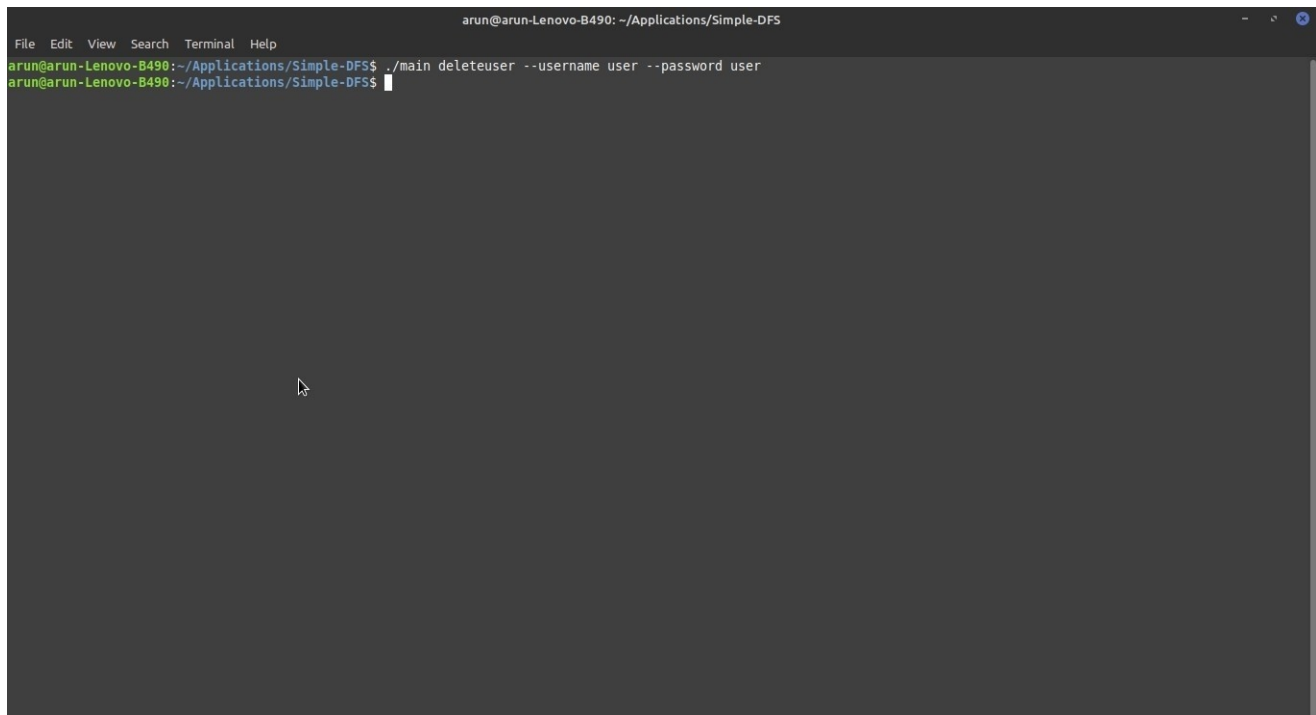
BACKEND SERVER

CREATE USER

A terminal window titled 'arun@arun-Lenovo-B490: ~/Applications/Simple-DFS' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'arun@arun-Lenovo-B490:~/Applications/Simple-DFS\$'. The command './main createuser --username user --password user' is entered. The output is 'User created successfully'. The prompt is then 'arun@arun-Lenovo-B490:~/Applications/Simple-DFS\$' with a cursor.

```
arun@arun-Lenovo-B490:~/Applications/Simple-DFS$ ./main createuser --username user --password user
User created successfully
arun@arun-Lenovo-B490:~/Applications/Simple-DFS$
```

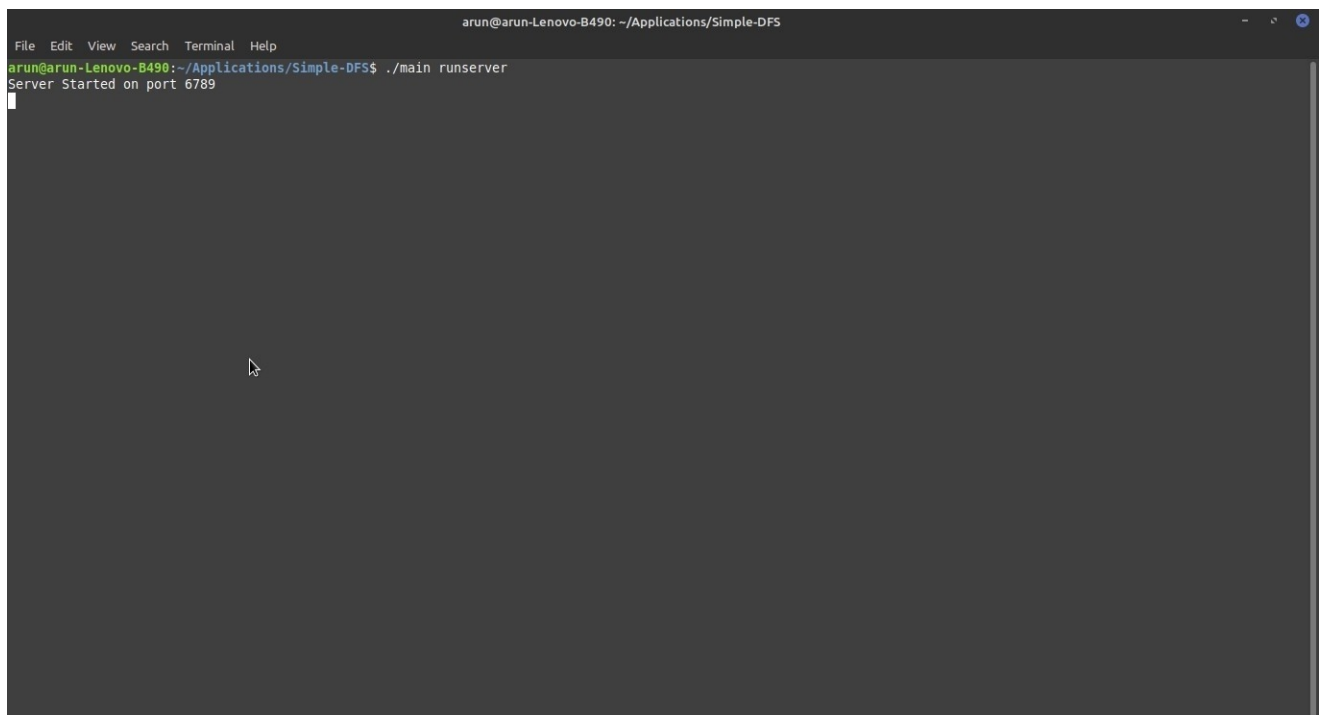

DELETE USER



A terminal window titled "arun@arun-Lenovo-B490: ~/Applications/Simple-DFS" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command `./main deleteuser --username user --password user` being executed. The prompt is `arun@arun-Lenovo-B490:~/Applications/Simple-DFS$`.

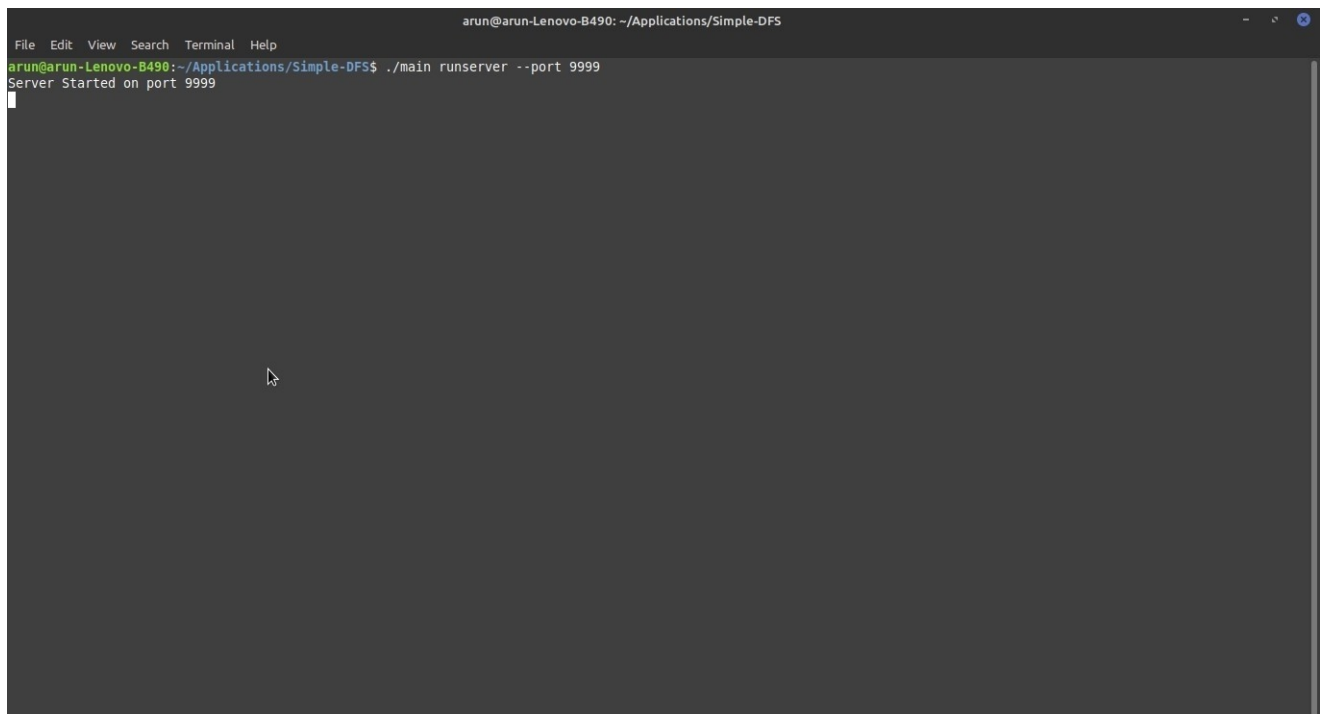
```
arun@arun-Lenovo-B490:~/Applications/Simple-DFS$ ./main deleteuser --username user --password user
arun@arun-Lenovo-B490:~/Applications/Simple-DFS$
```

RUN SERVER

A terminal window with a dark background and light text. The title bar at the top reads 'arun@arun-Lenovo-B490: ~/Applications/Simple-DFS'. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The main area of the terminal shows a green prompt 'arun@arun-Lenovo-B490:~/Applications/Simple-DFS\$' followed by the command './main runserver' in white. Below the command, the output 'Server Started on port 6789' is displayed in white. A white cursor is visible on the line following the output.

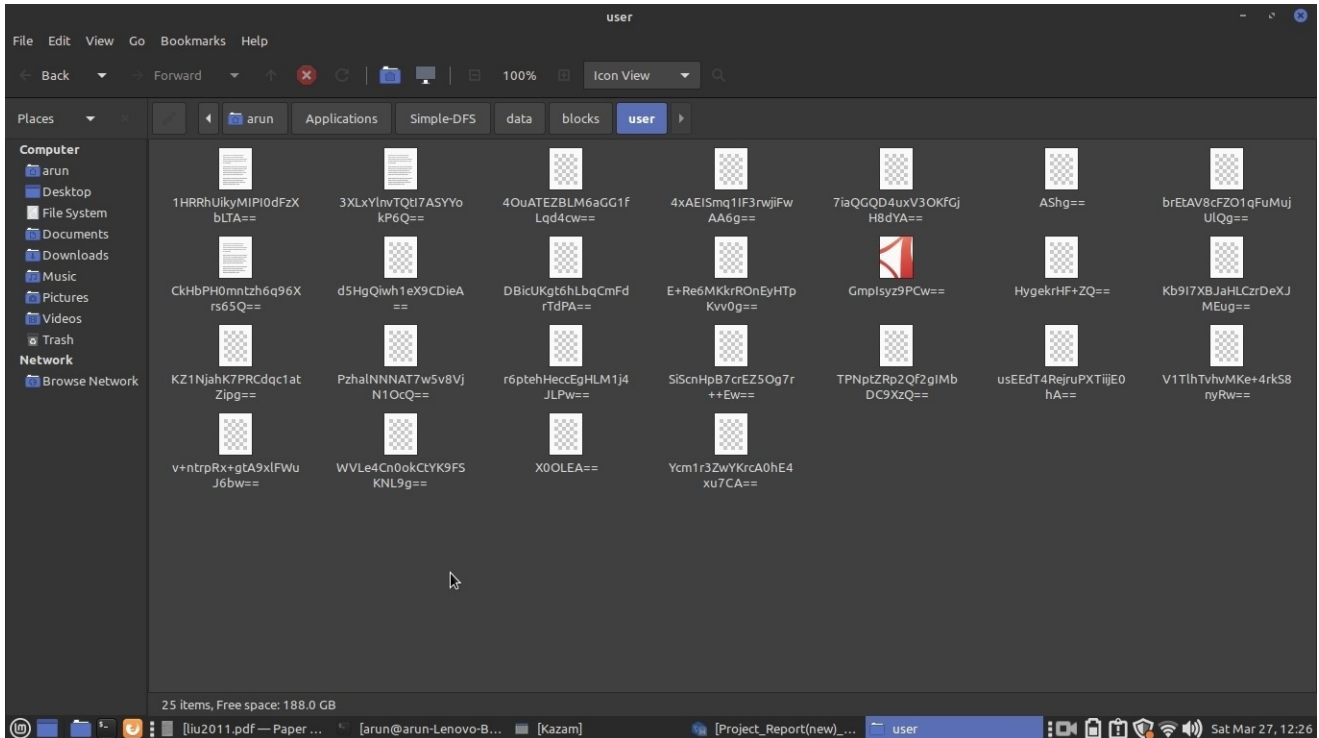
```
arun@arun-Lenovo-B490: ~/Applications/Simple-DFS
File Edit View Search Terminal Help
arun@arun-Lenovo-B490:~/Applications/Simple-DFS$ ./main runserver
Server Started on port 6789
```

RUN SERVER CUSTOM PORT NUMBER

A terminal window titled 'arun@arun-Lenovo-B490: ~/Applications/Simple-DFS'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The prompt is 'arun@arun-Lenovo-B490:~/Applications/Simple-DFS\$'. The command entered is './main runserver --port 9999'. The output is 'Server Started on port 9999'.

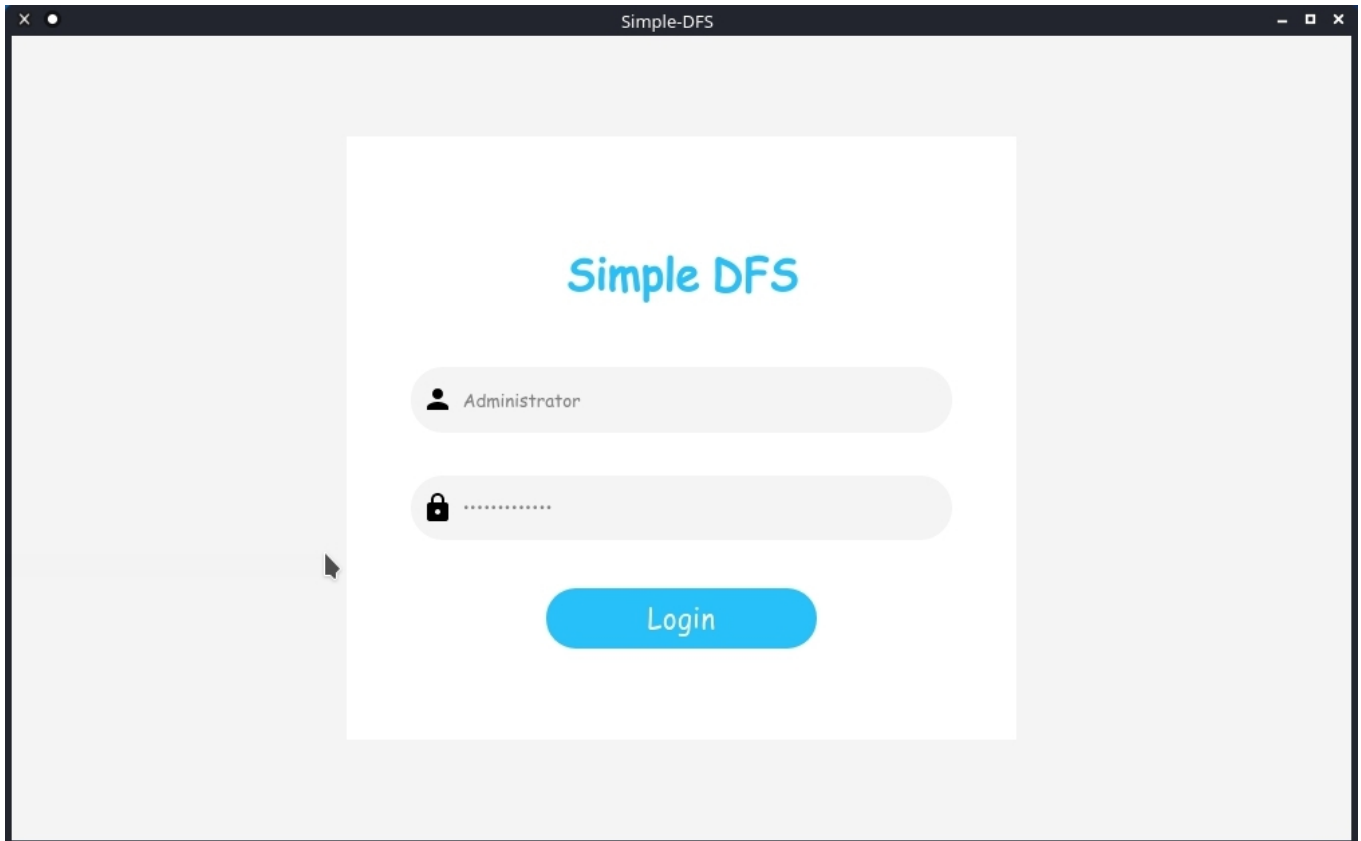
```
arun@arun-Lenovo-B490: ~/Applications/Simple-DFS
File Edit View Search Terminal Help
arun@arun-Lenovo-B490:~/Applications/Simple-DFS$ ./main runserver --port 9999
Server Started on port 9999
```

STORAGE OF BLOCK ON SERVER

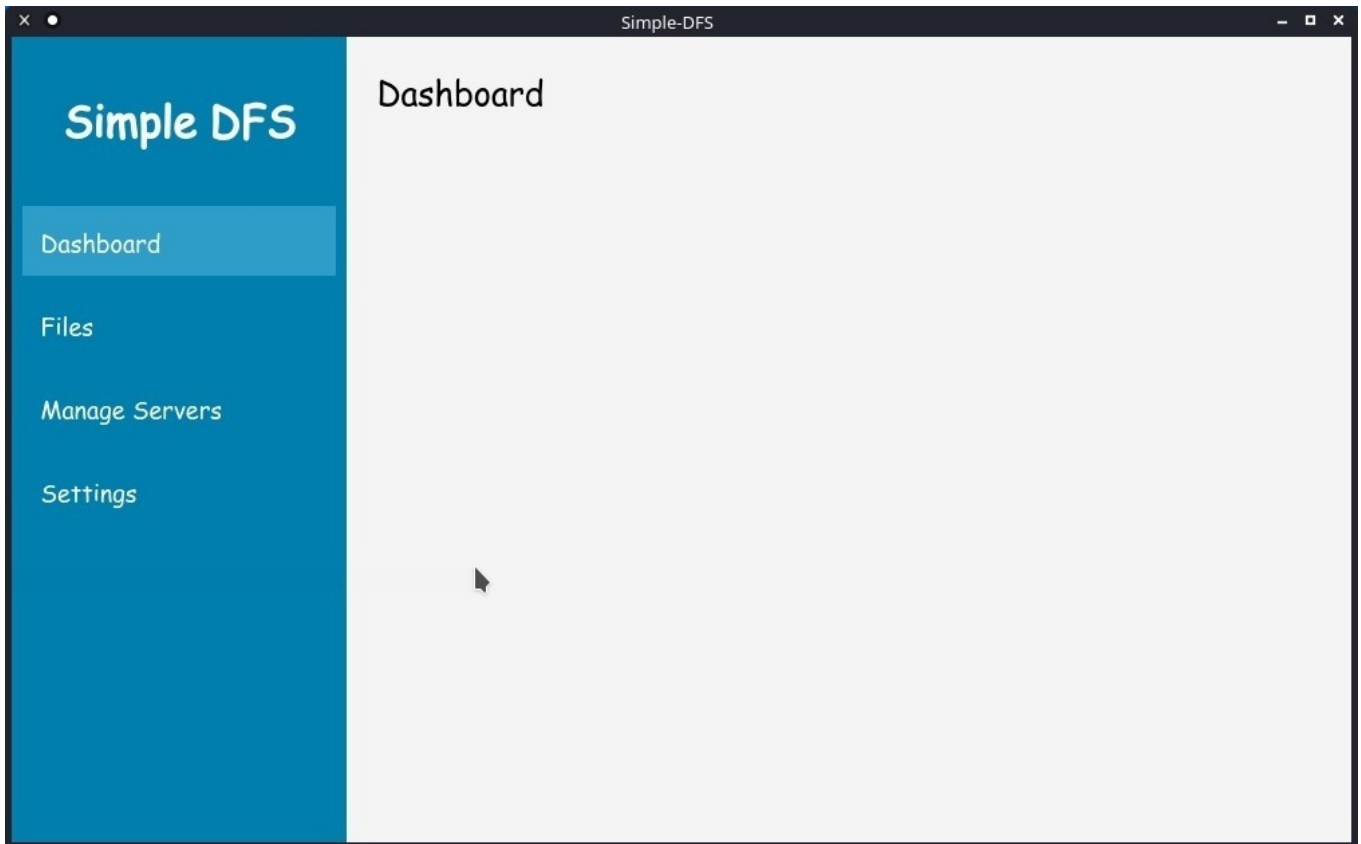


FRONTEND CLIENT

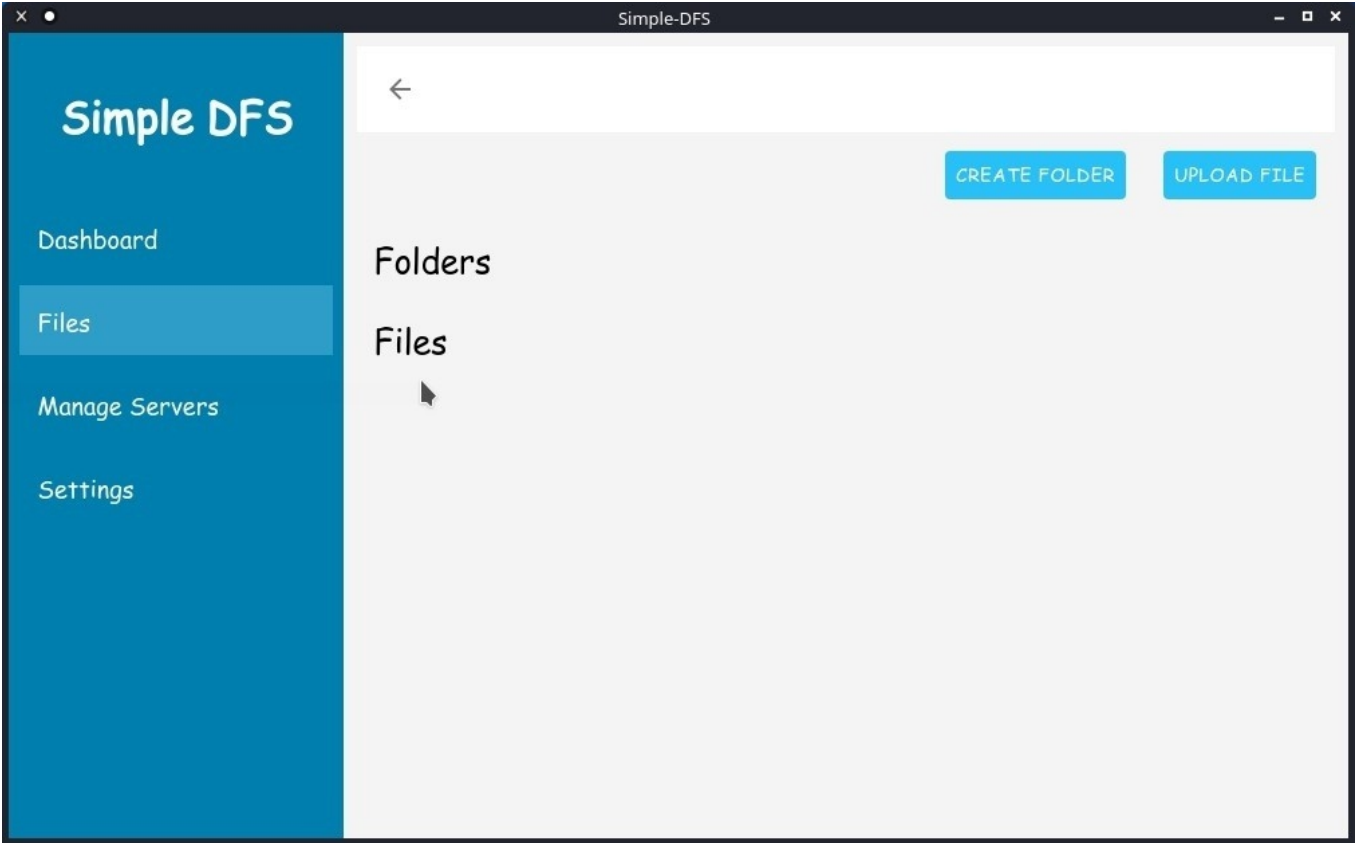
LOGIN SCREEN



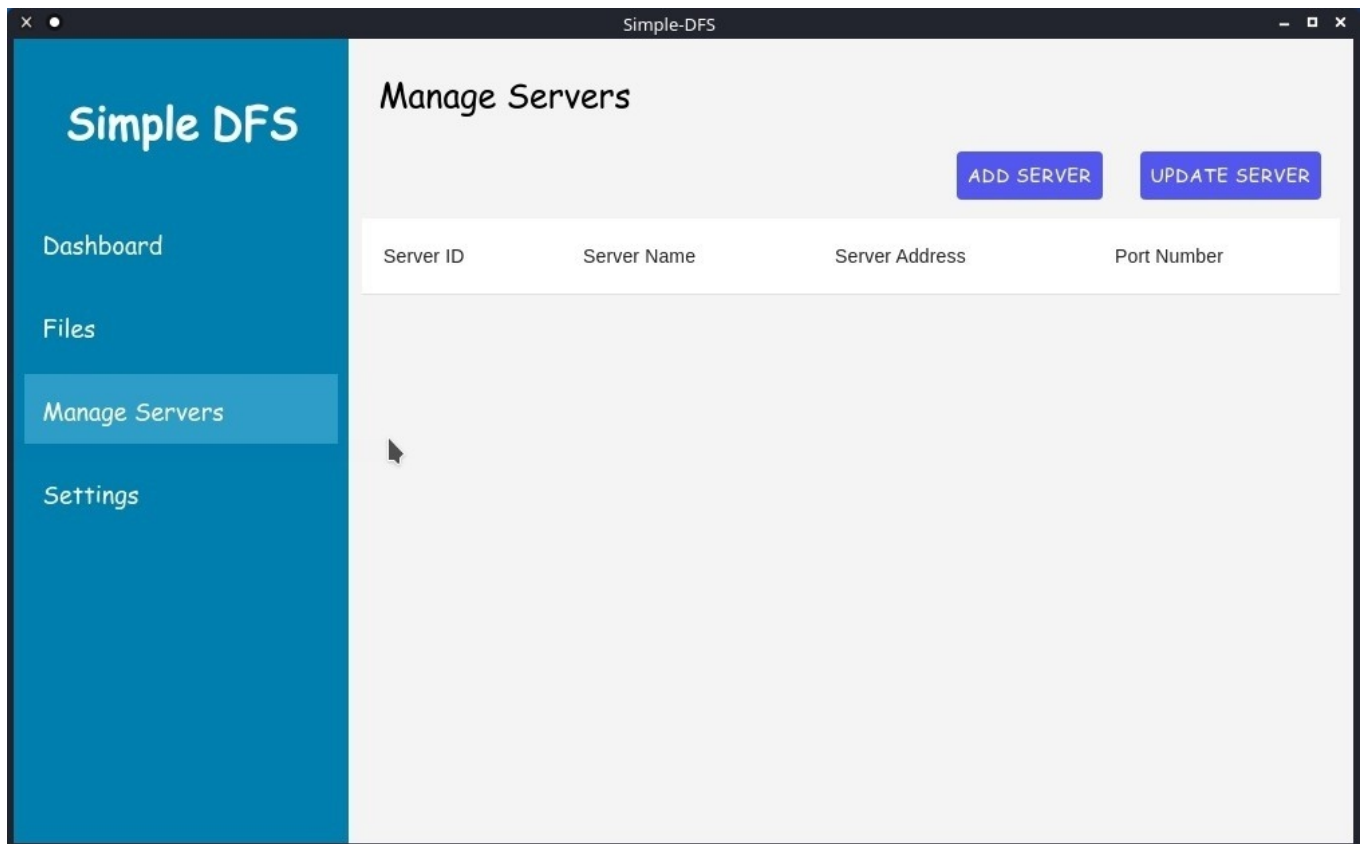
DASHBOARD



FILE BROWSER



MANAGE SERVER



SETTINGS

Simple DFS

Dashboard

Files

Manage Servers

Settings

Change Username

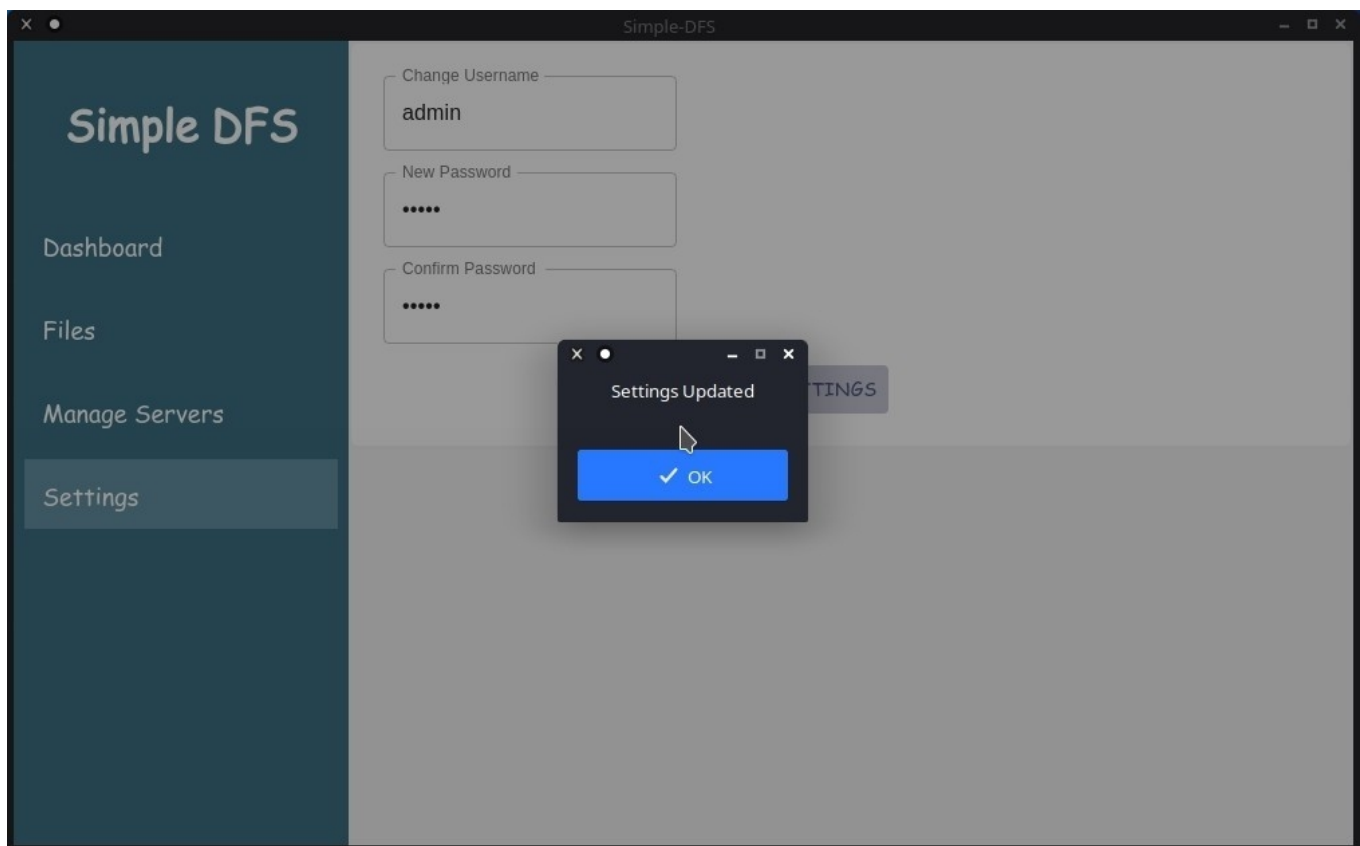
admin

New Password

Confirm Password

SAVE SETTINGS

SETTINGS UPDATE NOTIFICATIONS



ADD A NEW SERVER DIALOGUE

The image shows a web application window titled 'Simple-DFS' with a sidebar menu containing 'Dashboard', 'Files', 'Manage Servers', and 'Settings'. The 'Manage Servers' section is active. A modal dialog box titled 'Add a New DFS Server' is open, prompting the user to 'Enter details of the server'. The dialog contains the following fields:

- Server Name
- Server Address
- PORT Number
- Enter credentials for the server
- Server Username
- Server Password

A 'PROCEED' button is located at the bottom right of the dialog. In the background, an 'UPDATE SERVER' button and a 'Port Number' label are visible.

ENTER DETAILS OF THE SERVER

The screenshot shows a web application titled "Simple DFS" with a sidebar menu containing "Dashboard", "Files", "Manage Servers", and "Settings". The "Manage Servers" page is active, displaying a table with columns for "Server Name", "Server Address", "PORT Number", and "Server Username". A modal window titled "Add a New DFS Server" is open, prompting the user to "Enter details of the server". The form includes fields for "Server Name" (filled with "Arun Pandiyan Lenovo Laptop"), "Server Address" (filled with "192.168.225.122"), "PORT Number" (filled with "6789"), "Server Username" (filled with "user"), and "Server Password" (masked with dots). A "PROCEED" button is at the bottom right of the modal. In the background, an "UPDATE SERVER" button and a "Port Number" label are visible.

Simple DFS

Manage Servers

Dashboard

Files

Manage Servers

Settings

Update Server

Port Number

Add a New DFS Server

Enter details of the server

Server Name

Arun Pandiyan Lenovo Laptop

Server Address

192.168.225.122

PORT Number

6789

Enter credentials for the server

Server Username

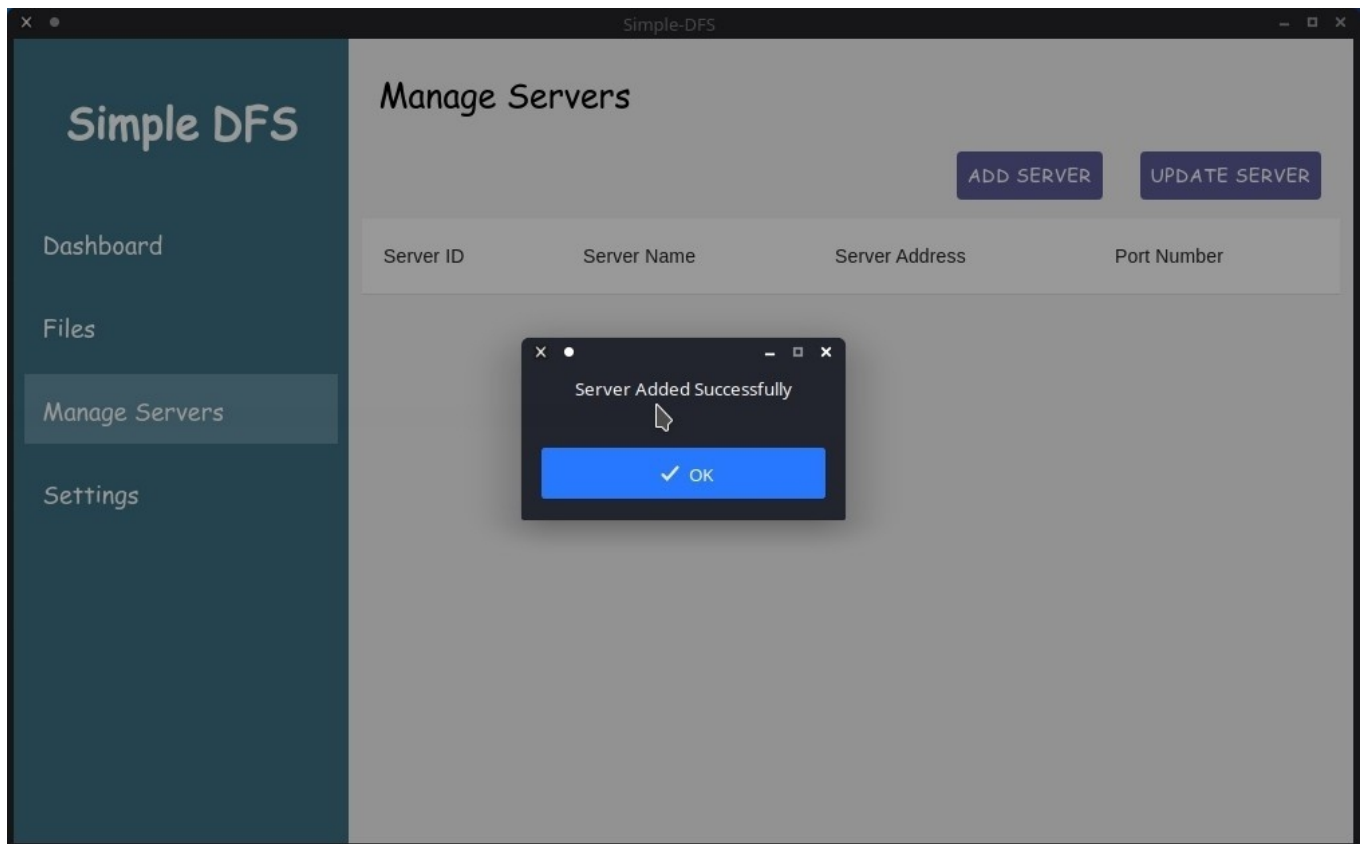
user

Server Password

....

PROCEED

SERVER ADDED NODIFICATIONS

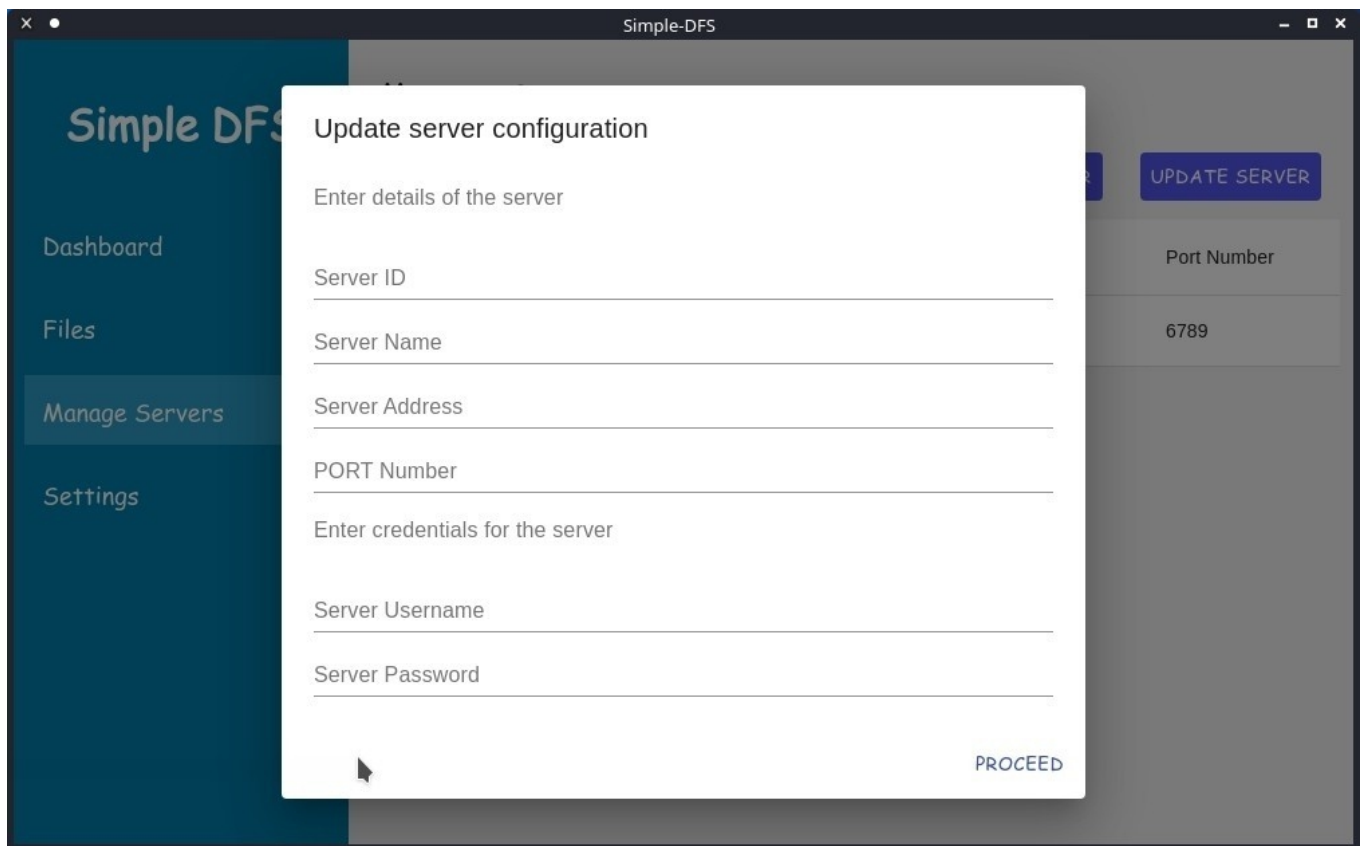


MANAGE SERVER SECTION

The screenshot shows a web application window titled "Simple-DFS". On the left is a blue sidebar with the logo "Simple DFS" and a menu with four items: "Dashboard", "Files", "Manage Servers" (which is highlighted), and "Settings". The main content area is titled "Manage Servers" and contains two blue buttons: "ADD SERVER" and "UPDATE SERVER". Below these buttons is a table with the following data:

Server ID	Server Name	Server Address	Port Number
1	Arun Pandiyan Lenovo Laptop	192.168.225.122	6789

UPDATE SERVER CONFIGURATION DIALOGUE



Simple DFS

Update server configuration

Enter details of the server

Server ID

Server Name

Server Address

PORT Number

Enter credentials for the server

Server Username

Server Password

PROCEED

Dashboard

Files

Manage Servers

Settings

UPDATE SERVER

Port Number

6789

ENTERING DETAILS UPDATE SERVER

The screenshot shows a web application titled 'Simple DFS' with a dark blue sidebar containing navigation links: 'Dashboard', 'Files', 'Manage Servers', and 'Settings'. The main content area is grey and displays a form for updating server configuration. A modal dialog box is open, titled 'Update server configuration', with the instruction 'Enter details of the server'. The form fields are as follows:

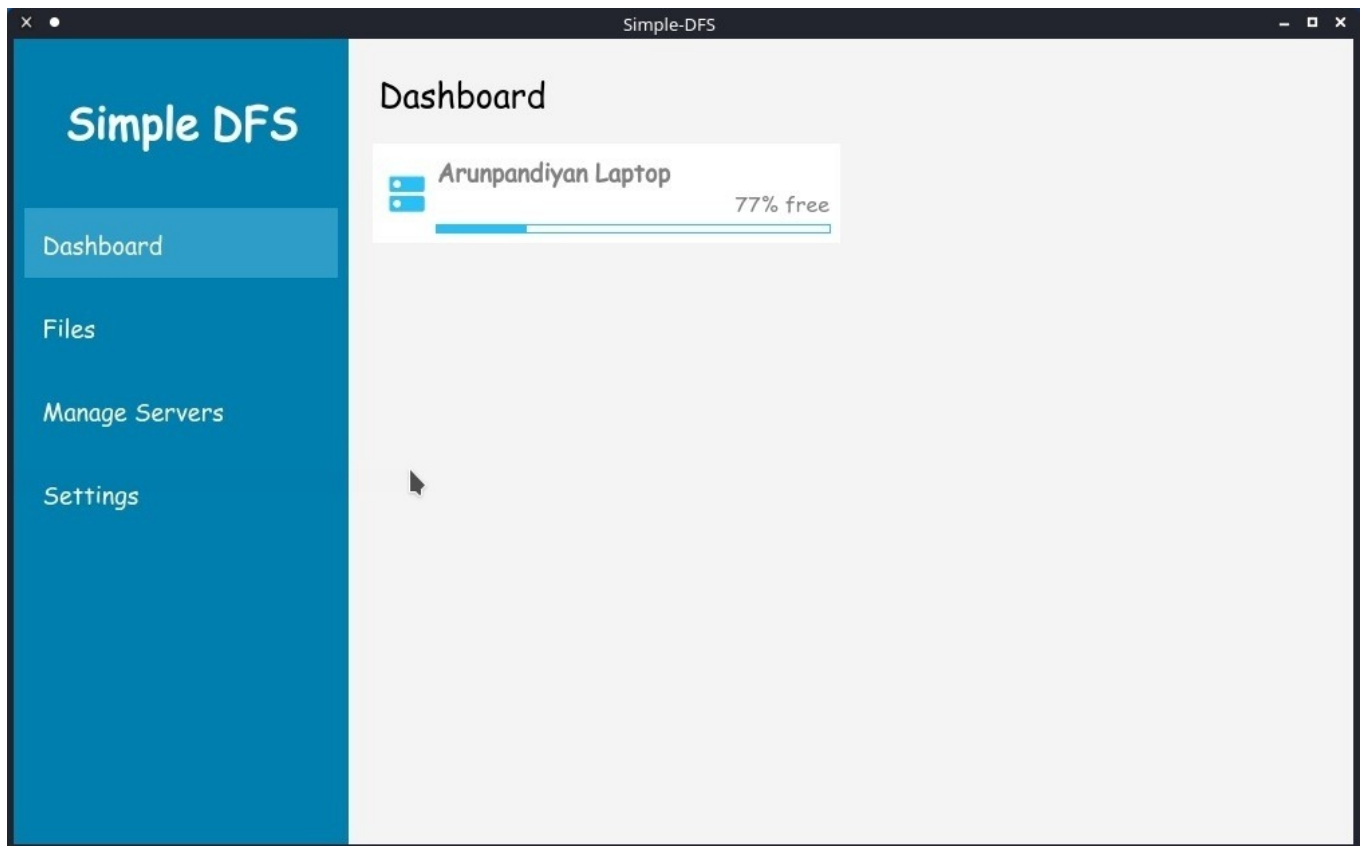
Field Label	Value
Server ID	1
Server Name	Arunpandiyan Laptop
Server Address	192.168.225.122
PORT Number	6789

Below these fields is a section titled 'Enter credentials for the server' with two more fields:

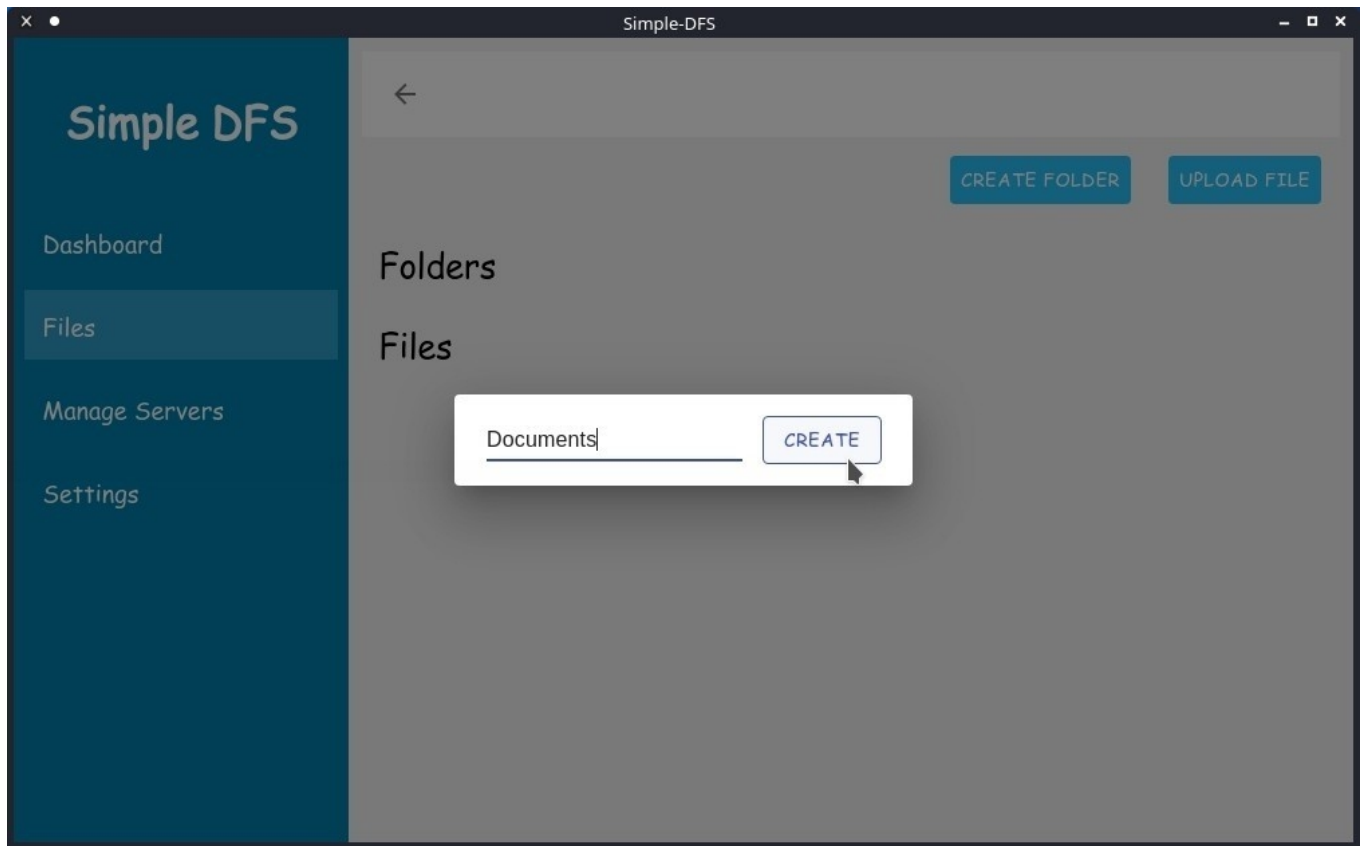
Field Label	Value
Server Username	user
Server Password	••••

A 'PROCEED' button is located at the bottom right of the dialog box. In the background, a blue 'UPDATE SERVER' button is visible on the right side of the main form.

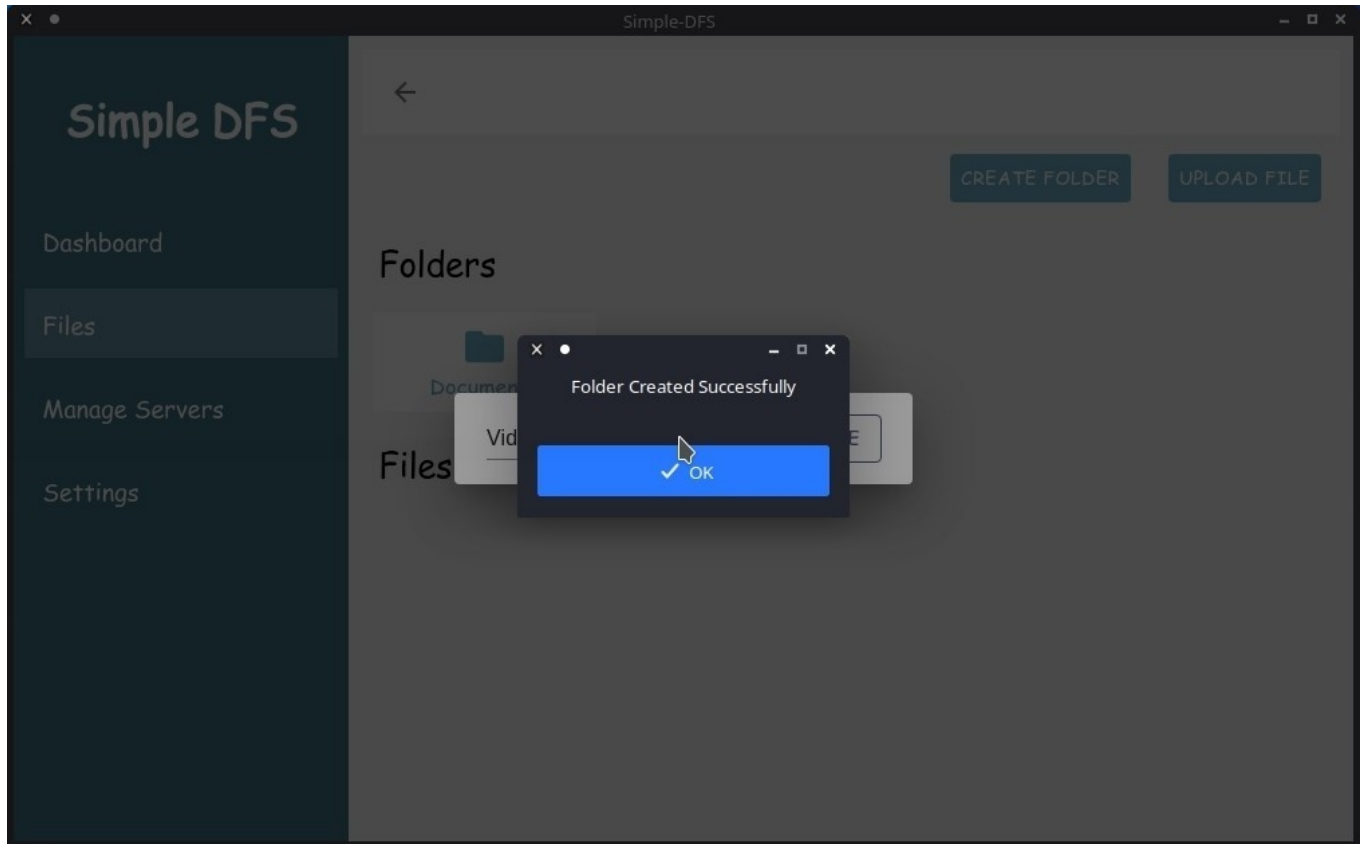
DASHBOARD WITH NEWLY ADDED AND UPDATED SERVER



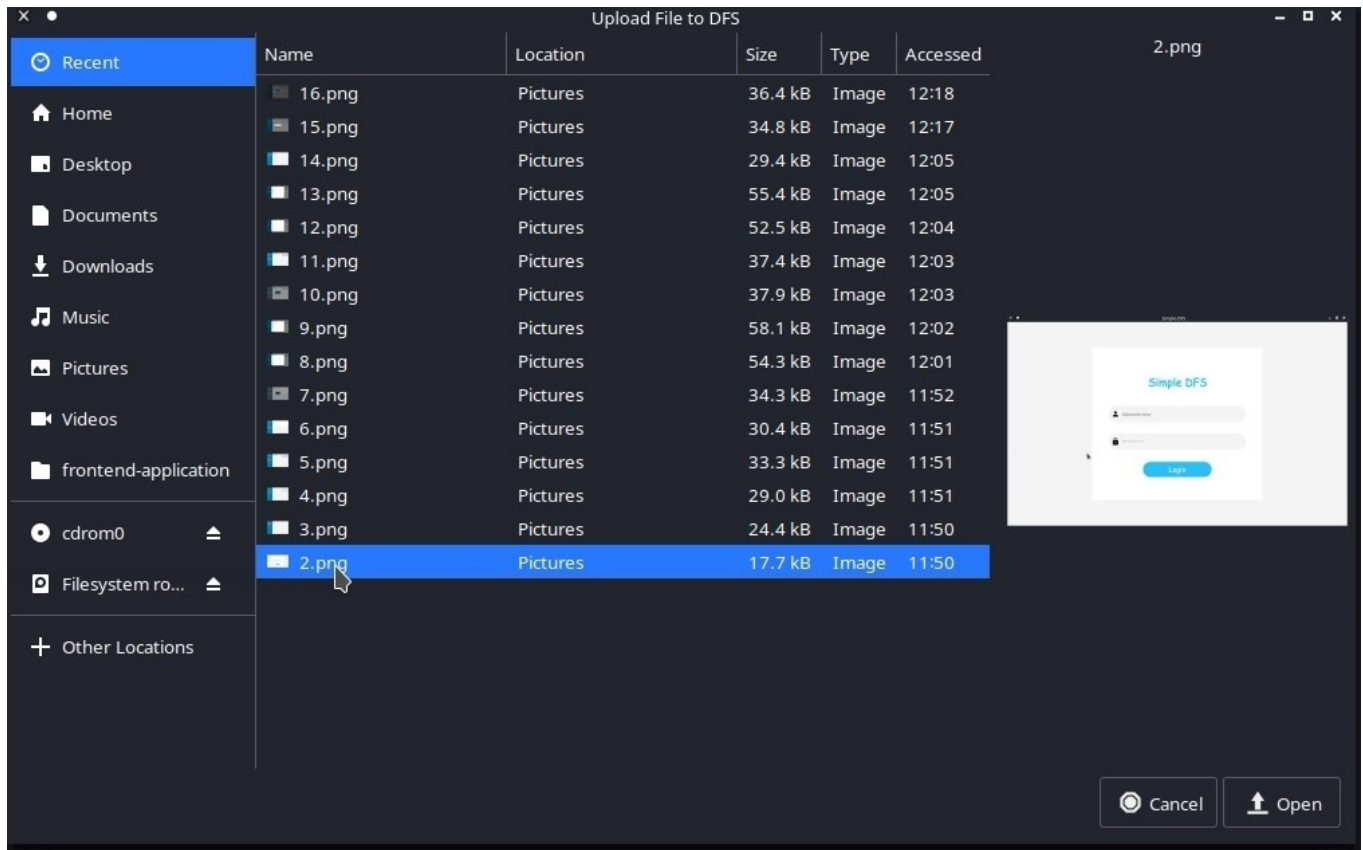
CREATE FOLDER DIALOG



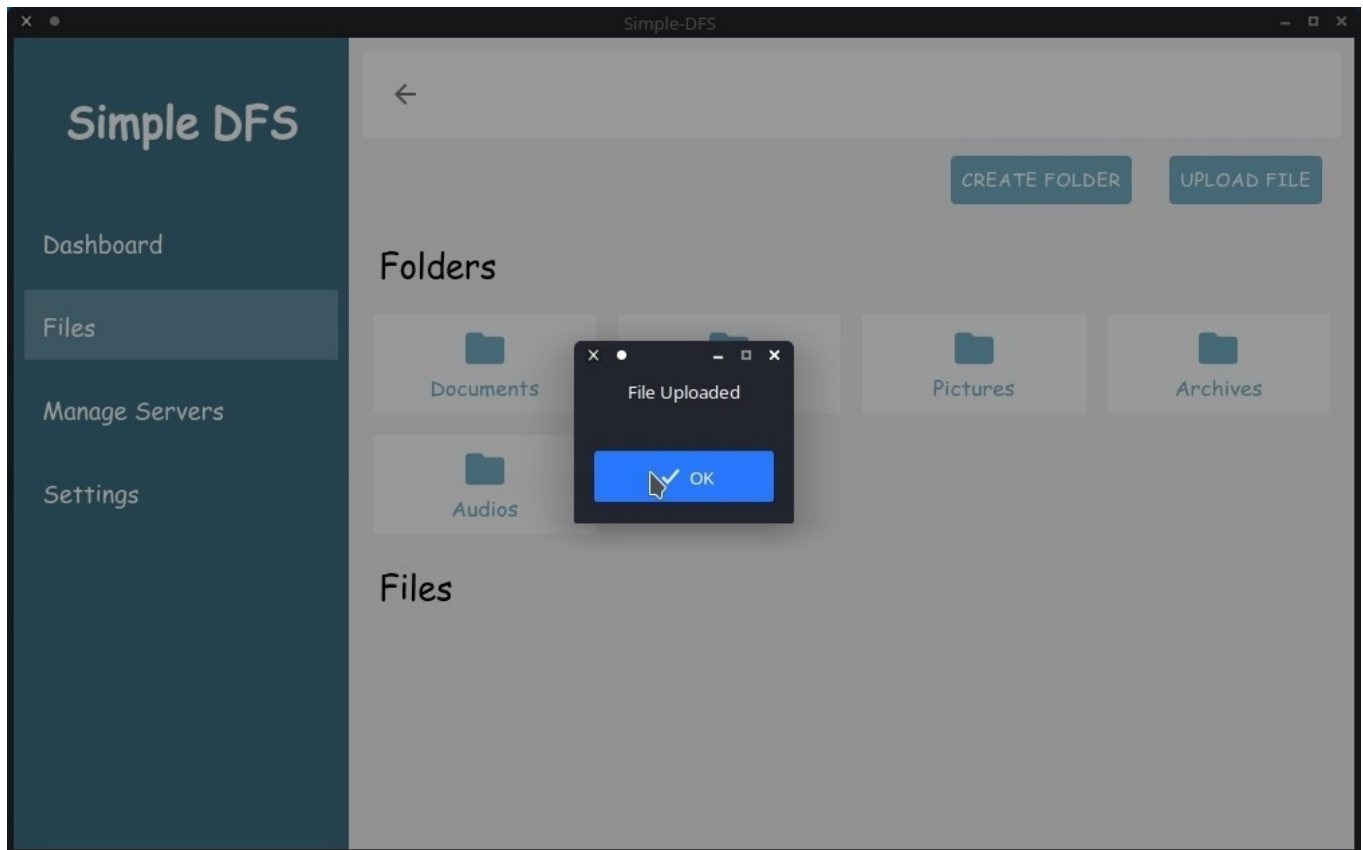
FOLDER CREATED NOTIFICATION



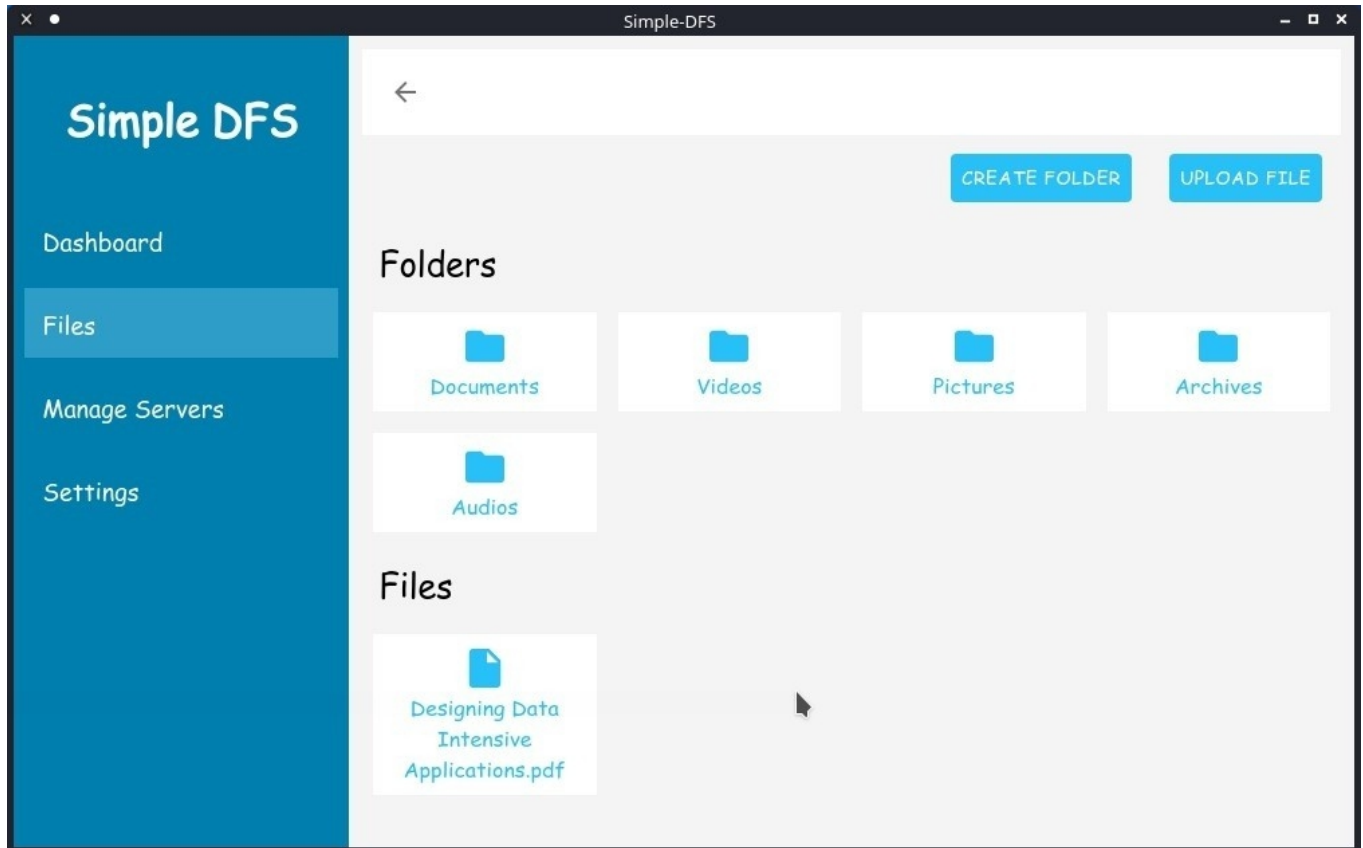
OPEN FILE DIALOG



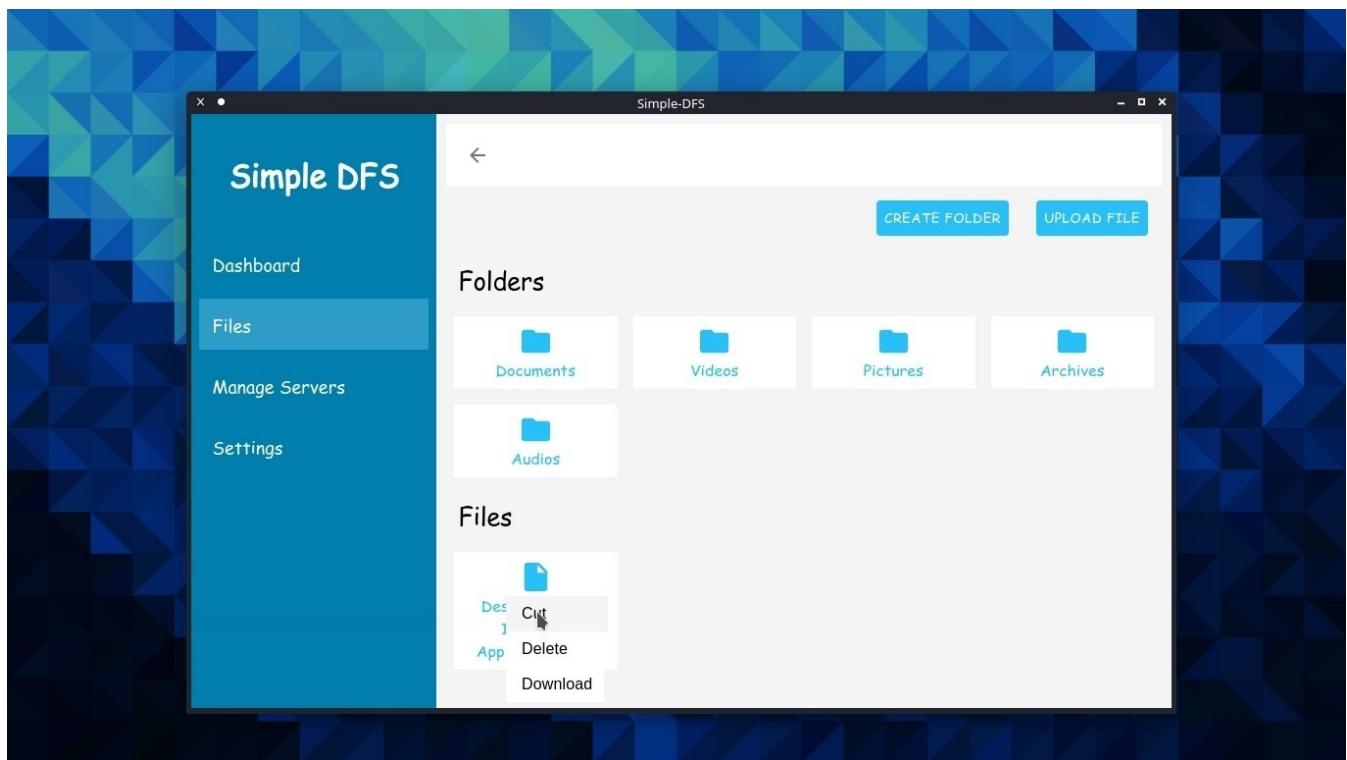
FILE UPLOADED NOTIFICATION



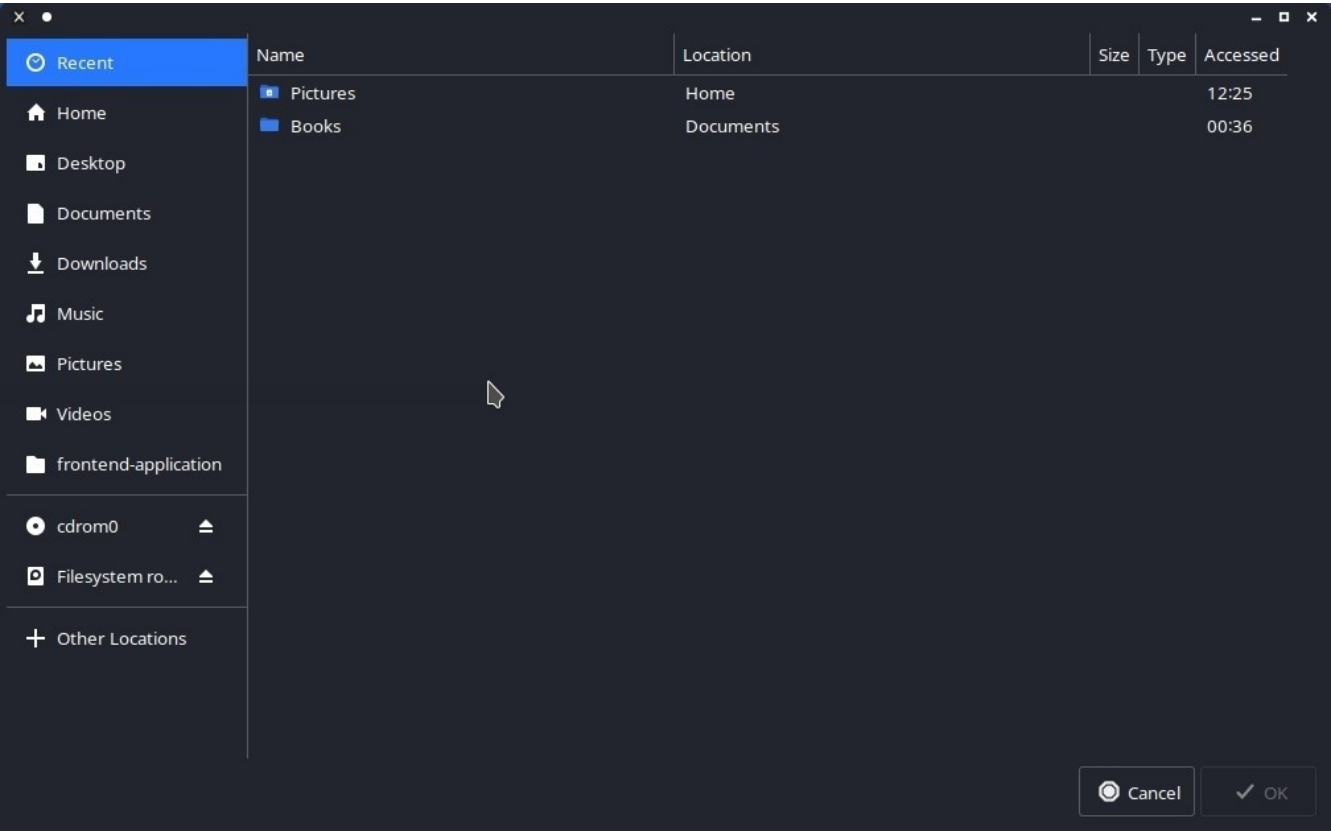
LIST OF UPLOADED FILES



FILE OPERATIONS



FILE DESTINATION TO STORE FILE



CHAPTER10

CONCLUSION

This project proposes a high performance and low cost distributed file system. The proposed system integrates several commodity computers into a large capability distributed file system. The propose system increase the reliability by replicating files in different data servers. To prevent the decreasing of the overall system performance when the loading of some data server is high, we design a load detection module. The experimental result shows that the load detection module can improve the overall system performance when the loading of some data servers is high.

This project describes a distributed file system designed for serving JD's e-commerce business. DFS has a few unique features that differentiates itself from other open source solutions. For example, it has a general-purpose storage engine with scenario-aware replications to accommodate different file access patterns with optimized performance. In addition,it employs a separate cluster to store the file metadata in a distributed fashion with a simple but efficient metadata placement strategy. Moreover, it provides POSIX-compliant APIs with relaxed POSIX semantics and metadata atomicity to obtain better system performance.We have implemented most of the operations by following the POSIX standard,and are actively working on the remaining ones,such as xattr, fcntl, ioctl, mknod and readdirplus.

CHAPTER 11

FUTURE ENHANCEMENT

11.1 FUTURE ENHANCEMENT

This system will be extended to use cloud servers as storage servers.

Replication feature will be added to prevent data loss.

Modules for automatically selecting servers based on performance metrics will be added to improve data transfer and it's efficiency.

CHAPTER 12

REFERENCES

1. Krish K.R., Ali Anwar, Ali R. Butt, “hatS: A Heterogeneity-Aware Tiered Storage for Hadoop”, 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, DOI 10.1109/CCGrid.51, pp.502-511,IEEE,2014.
2. F. Azzedin. “Towards a scalable HDFS architecture”, in IEEE International Conference on Collaboration Technologies and Systems (CTS), pp. 155-161,2013.
3. Shaojian Zhuo, Xing Wu, Wu Zhang and Wanchun Dou, “Distributed file system and classification for small images”, IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, DOI 10.1109/GreenCom-iThings-CPSCom.422, pp. 2231 2234,201.
4. Konstantin Shvachko, Hairong Kunag, Sanjay Radia and Robert Chansler, ”The Hadoop Distributed File System” Sunnyvale, CaliforniaUSA.

5. Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Wu, J., Simitci, H., Haridas, J., Uddaraju, C., Khatri, H., Edwards, A., Bedekar, V., Mainali, S., Abbasi, R., Agarwal, A., Haq, M. F. u., Haq, M. I. u., Bhardwaj, D., Dayanand, S., Adusumilli, A., McNett, M., Sankaran, S., Manivannan, K., and Rigas, L. Windows azure storage: A highly available cloud storage service with strong consistency. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (New York, NY, USA, 2011), SOSP '11, ACM, pp. 143– 157.

6. Dennis Fetterly, Maya Haridasan, Michael Isard and Swaminathan Sundararaman, "TidyFS: A Simple and Small Distributed File System". Microsoft Research Technical Report, MSR-TR-20110-24.