Scientific Research Publishing

Open Journal of Applied Sciences, 2023, 13, 643-658
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

Distributed File System Based on a Relational Database

Vladyslav Kutsman

Computer Sciences Department, Information Technologies and Computer Engineering Faculty, Vinnytsia National Technical University, Vinnytsia, Ukraine
Email: kutsmanvlad@gmail.com

How to cite this paper: Kutsman, V. (2023) Distributed File System Based on a Relational Database. *Open Journal of Applied Sciences*, **13**, 643-658.
https://doi.org/10.4236/ojapps.2023.135051

Received: March 25, 2023
Accepted: May 13, 2023
Published: May 16, 2023

Abstract

Working with files and the safety of information has always been relevant, especially in financial institutions where the requirements for the safety of information and security are especially important. And in today's conditions, when an earthquake can destroy the floor of a city in an instant, or when a missile hits an office and all servers turn into scrap metal, the issue of data safety becomes especially important. Also, you can't put the cost of the software and the convenience of working with files in last place. Especially if an office worker needs to find the necessary information on a client, a financial contract or a company's financial product in a few seconds. Also, during the operation of computer equipment, failures are possible, and some of them can lead to partial or complete loss of information. In this paper, it is proposed to create another level of abstraction for working with the file system, which will be based on a relational database as a storage of objects and access rights to objects. Also considered are possible protocols for transferring data to other programs that work with files, these can be both small sites and the operating system itself. This article will be especially interesting for financial institutions or companies operating in the banking sector. The purpose of this article is an attempt to introduce another level of abstraction for working with files. A level that is completely abstracted from the storage medium.

Keywords

File System, PostgreSQL, Golang, AWS, HDD, Ldap, Active Directory, AES Encryption, Fintech, Banking

1. Introduction

The storage and use of information is a complicated process, since one has to face a large number of tasks that have to be solved in the process of work. For

examples information can be stored on a storage with different types of file system like NTFS [1] or FAT [2] also it can be AWS [3] storage and each of these storages has own characteristics, depending of the business requirements. In this article operate with the following entities such as COMPANY—PRODUCT—CLIENT. An example of a company is an insurance company, a bank, a leasing company. As an example of a PRODUCT, as example can be insurance, credit, leasing, respectively by the essence of the client need to understand the final recipient of the financial product.

Each entity has a specific set of properties. So in the case of a company, this is a certificate, a license, etc. For a product, this is a description of the product itself, for example, if it is a car leasing, then this is the documentation for the subject of leasing itself. In the case of a client, this is a certain set of documents of the client, which ultimately allows to conclude the financial transaction itself. But for a computer and the operating system [4] (hereinafter OS) installed on it, all these entities are a record in the database and a set of files with different sizes and extensions, such as pdf, png, etc. Also, the OS does not know anything about the importance of this or that document. In the same way, it is impossible to guess in advance how much disk space is needed in order to store information. Lets to consider a real case. For example, if you structure the data of the company's clients by name, where the client's name is a directory on disk N and subdirectories are a specific set of data on the client, and if the information on the logical disk are placed in the form **Figure 1** of a tree.

Then soon may arise a situation when the disk is full and it will be impossible to create directory for a new client or add files to an existing client directory because this will require transferring data for some client directory to another disk. But transferring data on the John Doe 1 client from one disk to another is not as trivial a task as it might seem at first glance. Since the data for this client is most likely stored in some accounting system in which links to documents are stored, and accordingly, when changing the disk, it will be necessary to change the links in the database of the accounting system. The situation is exactly the same when it is necessary to transfer new clients to another drive, which leads to inconvenience when working with data directly for the company manager. Now let's simulate a situation where, for some unknown reason, not enough attention is

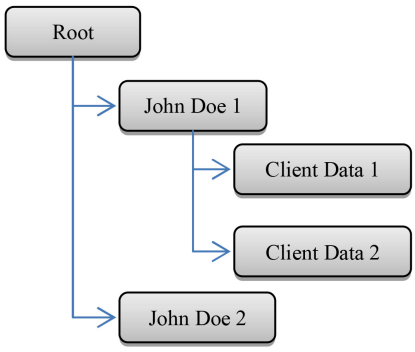

Figure 1. File tree structure client information.

paid to data security. And an unwanted user of the company gets access to our N drive, and as a result, he receives all the information on the company's customers in an open form, downloads what he needs and safely sends the company's customer base to a competitor. In the same way, it may be necessary to separate information within the client directory into access levels by company departments, for example, only Department1 should have access to the ClientData1 directory, and Department2 and Department3 should have access to ClientData2. That is, to create a client directory in the case of windows, its need at least a bat file that will automatically set the rights to the client directory and subdirectories. And it becomes clear that there are a number of tasks that may not be so obvious at first glance, but they exist and often they have to be solved in completely different ways.

This article is a description of the key nodes and the general mechanism of a possible abstraction level based on a relational database for working with files and data stores, that is, a kind of distributed file system based on relational database DFSboRD.

DFSboRD architecture.

On **Figure 2** shows a schematic representation of the main elements of the general architecture of DFSboRD.

This diagram shows the following key elements:

1) Physical storage such as NASS, AWS, HDD.

2) Database DB, within the article the database is implemented on PostgreSQL [5].

3) The SB software block that implements protocols for working with files within the article was released in Golang [6]. The following file exchange protocols have also been implemented, such as WebDav [7], FTP/FTPS [8] [9], a local implementation of file exchange based on the HTTP(S) protocol [10] [11].

4) Admin panel AP is designed for manage the issuance of rights, the creation of logical drives, the generation of connection links, etc.

One of the main elements in this scheme is the DataBase and the SB software block **Figure 2**. For a more detailed consideration, we divide into two corresponding parts.

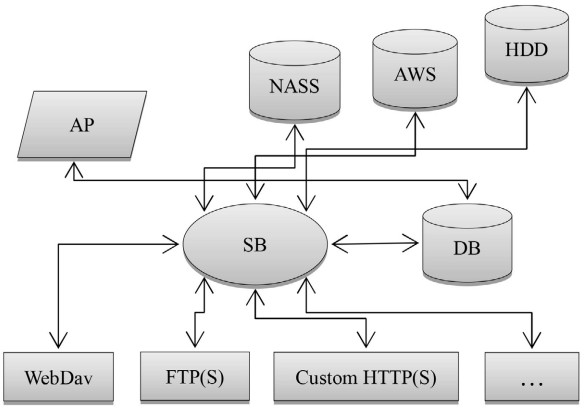

Figure 2. General architecture DFSboRD.

2. Data Base

To better understand the idea it is necessary to consider in detail the main objects of the database. For an example, was used descriptions and snapshots of tables from a working database. For convenience, the database objects was divided into two parts, namely the logical part—the part that stores information about the logical structure of disks, directories, files, access rights, tokens and the physical part—these are Tables 1-4 that store data about physical storages.

2.1. Logical Part

Let's look at the main objects that form the logical part:

Table 1. The disc table.

Column	Description
Id	Bigint PK
Name	Character varying (500)—name of the disc
Size	Bigint—disc size in byte

Table 2. The folder_to_disc linking table.

Column	Description
Disc_id	Bigint PK
Folder_id	Bigint PK

Table 3. The file_to_disc linking table.

Column	Description
Disc_id	Bigint PK
File_id	Bigint PK

Table 4. The folder table.

Column	Description
Id	Bigint PK
Name	Character varying (500)—name of the folder
Parent_id	Bigint—parent directory identifier
Size	Bigint—size
User_del	Bigint—the user who deleted the directory
Date_del	Timestamp—date when was deleted
User_id	Bigint—creator
Date_create	Timestamp—date when was created
Mod_time	Bigint—date when was modified
Perm	Smallint—field indicating existing of access control list

Table 4 has an additional composite index on the fields (Parent_id, Name), this index is used to speed up the search for a child directory if program receive a path as a string, as in the case of WebDAV or FTP protocols. For example, if program have the following string like "John Doe 1/Documents1", need to split it by the "/" character and, having received an array of strings [John Doe 1, Documents1], program will find out the Id of the "John Doe 1" directory using Table 2 link table, then program are looking for Id for "Documents1" and in this case need composite index (Parent_id, Name) for the fastest possible selection. In Table 4 and Table 5, there is a Perm field whose value range is [0, 1] where 0 is the Access control list (ACL) to the object are not set and 1 the ACL to the object are set. Table 1 does not have this field because the disk permissions must be set in advance. And in the case of a directory or a file, the rights can be inherited from the parent object until the Perm field is found in the child element, indicating that this object has its own ACL, which will be inherited by its child elements in the same way. Object File are different then Folder or Disc. The Files object is described using 2 tables: Table 5 and Table 7. Table 5 describes object File and all his attributes and Table 7 stores all file's physical addresses.

Table 3 and Table 6 are al linking tables, this tables realise a one-to-many relationship, because one folder and one disc can contains lot of files.

Table 5. The file table that stores information about a file.

Column	Description
Id	Bigint PK
Name	Character varying (500)—name of the file
Size	Bigint—size
Date_create	Timestamp—date when was created
Date_del	Timestamp—date when was deleted
Mime	Character varying (500)—mime-type
User_id	Bigint—creator
Mod_time	Bigint—date when was modified
User_del	Bigint—the user who deleted the file
Perm	Smallint—field indicating existing of ACL
Load	Smallint—field indicating load stay

Table 6. The file_to_folder linking table.

Column	Description
Folder_id	Bigint PK
File_id	Bigint PK

Table 7 has File_id, Store_id fields and this is the composite PK where File_id is the identifier of file object and Store_id is the physical store id. That is, one file can have several storage locations at the same time (Store_id), and if the physical storage fails, it just need to switch the program to use another storage block, and all other connected programs work as they did. The second important point here is the Crdec field—indicating in what form the file is in the physical storage in decrypted or encrypted, it will be discussed in detail later.

Table 8 is used as a bucket, which stores information from which directory or disk this file was deleted from. There is a similar table for the directory object.

Table 9 contains information about users. Table 10 contains information about user groups. And Table 11 realises one-to-many relationship user to groups.

Table 7. The file_to_way table, which stores information about the physical location of a file.

Column	Description
File_id	Bigint PK
Store_id	Bigint PK
Url	Character varying (500)—path to the storage file
Load	Smallint—field indicating load stay
Crdec	Smallint—field indicating in what form the file is in storage (encrypted or not)

Table 8. The file_delete table, which stores information about deleted file.

Column	Description
File_id	Bigint PK
Disc_id	Bigint
Folder_id	Bigint

Table 9. The user table, which stores information about users.

Column	Description
Id	Bigint PK
Name	Character varying (500)—Full name
User_login	Character varying (40)—User login
Password	Character varying (40)—Password encrypted by sha1

Table 10. The groups table, which stores information about user groups.

Column	Description
Id	Bigint PK
Name	Character varying (500)—name of the group

Table 11. The user_to_group linking table.

Column	Description
User_id	Bigint PK
Group_id	Bigint PK

The final element of the logical part will be the organization of access rights.

It is suggested to use the following types of access rights:

Table 12 contains basic permissions Figure 3 and Table 13 contains basic permissions groups Figure 4. Table 14 form permissions to permissions groups Figure 5. This was made for reasons of database architecture in order to be able to write access rights to an object as a composite PK.

There are similar tables for folder and file objects.

An example query with comments explaining permission checking:

SELECT (*CASE WHEN f.perm = 1 THEN f.id ELSE allowed_permid END*) *INTO allowed_permid FROM folder f*

LEFT JOIN perm_group_folder pgf ON pgf.folder_id = f.id AND pgf.group_id IN (*SELECT group_id FROM user_to_group WHERE user_id = inuser_id*)

LEFT JOIN perm_user_folder puf ON puf.folder_id = f.id AND puf.user_id = inuser_id

LEFT JOIN perm_to_pack ptpg ON ptpg.pack_id = pgf.perm_id AND ptpg.perm_id IN (*1*, *3*)

LEFT JOIN perm_to_pack ptpu ON ptpu.pack_id = puf.perm_id AND ptpu.perm_id IN (*1*, *3*)

WHERE f.id = dirid AND (*f.perm = 0 OR f.perm = 1 AND* (*puf.folder_id IS NOT NULL AND*

ptpu.pack_id IS NOT NULL OR puf.folder_id IS NULL AND ptpg.pack_id IS NOT NULL))*;*

This query checks write permissions ([1, 3] is the id of the permissions table) for user groups ptpg.perm_id IN (1, 3) and user ptpu.perm_id IN (1, 3).

inuser_id—variable storing the user ID.

allowed_permid—variable storing the NULL value from beginning.

This part query "(*CASE WHEN f.perm = 1 THEN f.id ELSE allowed_permid END*) *INTO allowed_permid*"—implements inheritance, that is, if the object has a perm value of 1, then the variable allowed_permid gets the value f.id, otherwise the previous value of allowed_permid is used. The link Table 15 and Table 16 are connected via a composite PK, which speeds up the selection as much as possible. This part query "(*puf.folder_id IS NOT NULL AND ptpu.pack_id IS NOT NULL OR puf.folder_id IS NULL AND ptpg.pack_id IS NOT NULL*)"—first of all, the user's rights are checked, and then the rights for the user's groups.

Table 17 contains the following fields of interest to us, such as Token (SHA1) [12] and what object identifiers it is associated with User_id, Disc_id, Folder_id. Also, one of the main conditions is that the Token value must be unique for any

Table 12. The permissions table storing basic permission codes.

Column	Description
Id	Bigint PK
Name	Character varying (255)
Description	Character varying (255)

Table 13. The perm_pack table represents grouped permissions.

Column	Description
Id	Bigint PK
Name	Character varying (255)
Description	Character varying (255)

Table 14. The perm_to_pack is linking table for perm_pack where perm_id this is Id from permissions and pack_id this is id from perm_pack.

Column	Description
Perm_id	Bigint
Pack_id	Bigint

Table 15. The perm_user_folder table.

Column	Description
Folder_id	Bigint PK
User_id	Bigint PK
Perm_id	Bigint—id of the perm_pack table

Table 16. The perm_group_folder table.

Column	Description
Folder_id	Bigint PK
Group_id	Bigint PK
Perm_id	Bigint—id таблицы perm_pack

Table 17. The token table holds user access tokens.

Column	Description
Id	Bigint PK
Token	Character varying (40)—user token
User_id	Bigint
Disc_id	Bigint
Folder_id	Bigint
Status	Smallint
Creator_id	Bigint

id [PK] bigint	name character varying (255)	description character varying (255)
1	F	Full access
2	R	Read
3	W	Write
4	D	Delete
5	N	Access denied

Figure 3. Main permissions codes list.

id [PK] bigint	description character varying (100)	name character varying (100)
1	Полный доступ	F
2	Закрыт доступ	N
3	Чтение	R
4	Чтение/Запись	R/W
5	Чтение/Удаление	R/D

Figure 4. Grouped permissions list.

perm_id bigint	pack_id bigint
1	1
5	2
2	3
2	4
3	4
2	5
4	5

Figure 5. Permissions grouped into permissionspack.

of the User_id, Disc_id, Folder_id pairs. For example, in Figure 6, the highlighted entry under id = 16 says that token=c80d9ba4852b67046bee487bcd9802c0 identifies the user user_id = 175 to access disc_id = 6. And if is necessary make the directory a network disk, then need specify 0 in the disc_id field and specify in the folder_id directory identifier (id) that must be as a network drive. In the future, Token will be used in SB Figure 2 to identify the client when interacting via the WebDav, FTP/FTPS protocols, this will be discussed in more detail later in the SB section.

2.2. Physical Part

Now let's to consider the implementation of the physical part of the base. But first, let's look at the structure of the file tree for placing files on physical storage, shown in Figure 7. At the root of a physical disk, 1000 directories can be placed

token character varying (32)	user_id bigint	disc_id bigint	folder_id bigint	id [PK] bigint	creator_id bigint	date_create bigint	status smallint
c80d9ba4852b67046bee487bcd9802c0	175	6	0	16	2	[null]	1
7451a7fab07e53d2638f693dd30f120f	175	3	0	109	[null]	[null]	0
fc1f073fe91403f00d2219185fdea79b	176	6	0	90	2	[null]	1

Figure 6. User token table data.

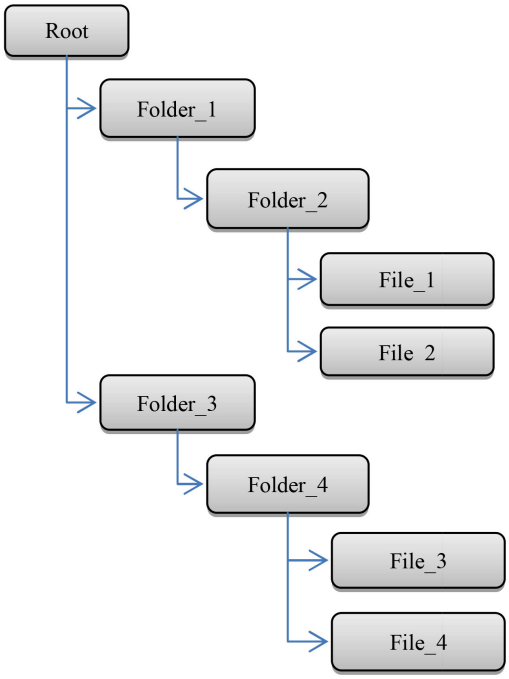

Figure 7. Suggested structure for placing files on storage.

in each directory, there are also 1000 directories and 1000 files in each child directory.

In the database, this structure is implemented by 4 tables.

Table 18 contains fields such as Url—this is the storage identifier within the operating system; in the case of Windows, it can be either the address of the disk itself or the address of the folder on the disk, for example, it can be Y:\ and Y:\storage\, respectively. Also, this table contains such a field as Crdec—indicating in what form files should be stored on this storage in encrypted or not. DFSboRD uses AES [13] encryption implemented by the golang. Also, this field is in Table 7, but in this table it indicates whether the file was encrypted before uploading or, in case of an error, it was not encrypted.

Table 19 provides to implement the structure shown in Figure 7. The Parent_id field indicates whether this is the root directory or not. Counter—counter of nested directories or files if it is a child directory.

Using Table 20 and Table 21, physical storage is grouped into blocks. The Status_replica field of the storage_pack indicates whether this block of physical storage is a replica or the main one that the system works with. Only one block can be the main one with which the system is currently working; the other are

Table 18. The storage_fiz table.

Column	Description
Id	Bigint PK
Name	Character varying (255)—name
Size	Bigint—size in bytes
Allowed_size	Bigint—free space in bytes
Url	Character varying (255)—Storage address
Crdec	Smallint—a field indicating in what form the files should be stored (encrypted/decrypted)

Table 19. The storage_folder table.

Column	Description
Id	Bigint PK
Counter	Integer
Parent_id	Bigint

Table 20. The storage_pack table.

Column	Description
Id	Bigint PK
Name	Character varying (255)
Status	Smallint
Status_replica	Smallint

Table 21. The storage_to_pack_storage linking table.

Column	Description
Storage_id	Bigint PK—Id of storage_fiz table
Pack_id	Bigint PK

used for data replication. Another important point is that in Table 7, within the same file object (File_id), physical stores (Store_id) are selected from different blocks. The program first time uploads the file to the storage located in the main block, and then after the replication operation, the system adds an entry to Table 7 with the storage identifier from other blocks that are replication. And as mentioned earlier, if the storage from the main block fails, the administrator can immediately make one of the replication units the main one, imperceptibly for other programs, and disable the damaged one and deal with the problem.

3. SB

Software block. In this section, was briefly consider the idea of implementing

protocols such as WebDav, FTP/FTPS, HTTP(S) for DFSboRD interaction with the user, taking into account the fact that information such as the logical drive, directories on the disk, files, and their attributes are read directly from the database not from the file system. Also, SB is directly involved in uploading/downloading files to physical storages. The SB configuration files store the key itself, which is used to encrypt/decrypt data. On Figure 8 shows a direct connection to DFSboRD in the form of a network drive using the WebDav protocol.

As shown in Figure 8, the system connects to the storage.domen.com server on port 10111—this port is responsible for the WebDav protocol. What is interesting here is that in the connection string was passing the user token "ert4re3wfgh654rtyhgfr45tre45tre3" which is generated in the AP admin panel in Figure 2 DFSboRD and is unique (Table 17 token) having generated another token, substitute it into the string and already connect to another network drive. Also, this token can be used when connecting to DFSboRD via FTP/FTPS protocols as a login and password.

When connecting to DFSboRD via FTP, the same domain was used, but placed on a different port, namely 2121 ftp://storage.domen.com:2121 further as shown in Figures 9-11. Entered login credentials and accessed data. Also, to work with data, a small web server was made that operates with the HTTP protocol and a small web page was made for working with data that displays the structure of the file tree. Also, the software block is responsible for encrypting and decrypting files before writing or reading to physical storage. Another important point is that when a user connects to DFSboRD, information about the user is checked in Active Directory using the ldap [14] protocol, and it is checked in which groups the user has been added and checking if he has already been blocked.

The implementation of the protocols itself in the sense of network interaction remains the same, only the part that is responsible for getting directory/file list and their information get data from database.

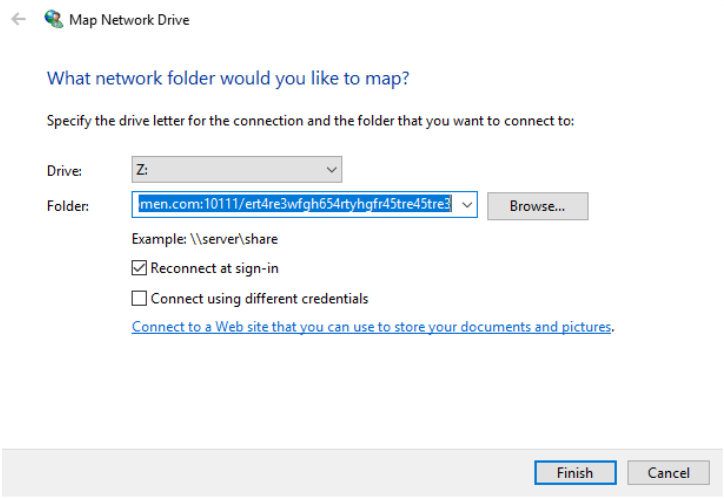

Figure 8. Connections to DFSboRD via WebDav protocol.

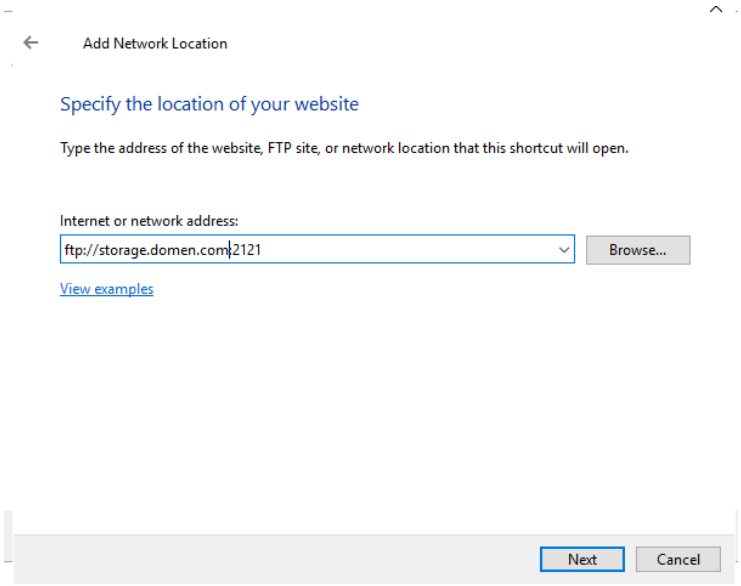

Figure 9. Connecting to DFSboRD via FTP.

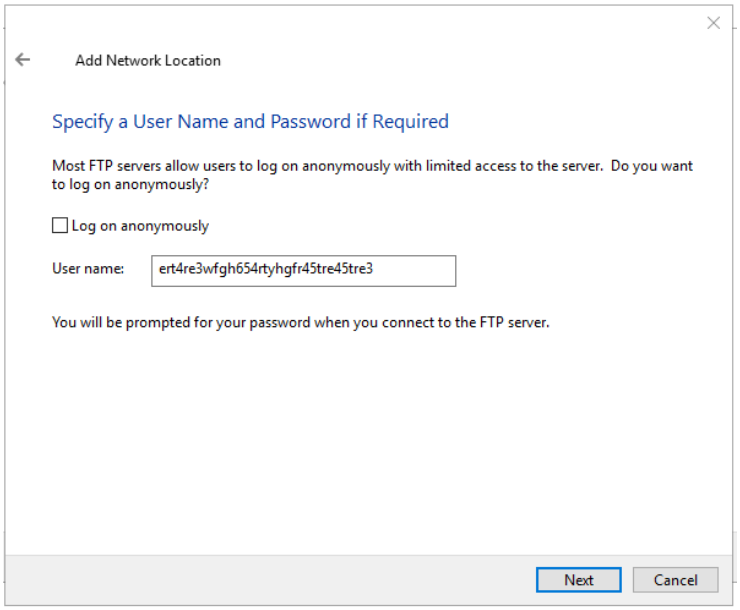

Figure 10. Connecting to DFSboRD via FTP protocol, login request window.

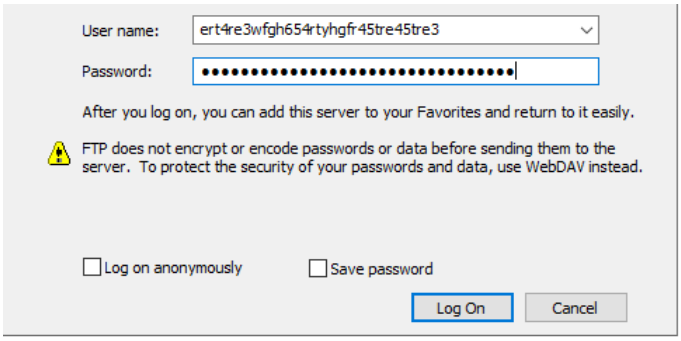

Figure 11. Connecting to DFSboRD via FTP protocol, password prompt window.

4. Discussion

Creating DFSboRD as an abstraction layer is a completely new approach to working with files. The situation in Ukraine serves as irreproachable proof of this. When, after the massive missile shelling of cities, business began to pull his information infrastructure into the clouds in order to save information. But at the same time, the problem arose of the lack of a solution that would allow this to be done quickly and would meet security requirements. The data had to be uploaded to cloud storage, and this is hundreds of terabytes of data, but also power outages. And when copying data while maintaining access rights, it slows down the process even more. Unlike other file systems, DFSboRD stores all data about logical drives, directories, files and their attributes, access rights in the database (which can have several copies, slave servers). Files can also have several copies as on servers in the office and on cloud storages, it is also very important that files are stored in encrypted form on cloud storages, which is very important for the security policy of banks. And in this case, the transfer of the infrastructure takes several minutes, namely, switching the slave server to the master, then you will need to go to the admin panel and make the main block responsible for storing files on cloud storage (Table 20), and its all, DFSboRD is already working on the clouds.

Also, DFSboRD easily integrates with the company's Active Directory, and since the data is stored in the database, it eliminates dependence on the domain.

The problem with the encoding in the file names is also solved, for example when the file is located in windows OS and has a name in Cyrillic, then when transferring to Linux, a situation may arise when the file names become unreadable, then DFSboRD does not have such problems, because the original file names are stored in the database and physically, files may have other names exclusively in Latin.

DFSboRD is very easy to maintain for the administrator, it is enough to sometimes check the replication of databases, other work the system will do itself. For exaple if the space on one of the blocks runs out, the administrator will be informed by letter or message in a messenger convenient for him.

Ease of integration with various CRM systems, to connect you will need a token that can be created in the admin panel and that's it. Further, after loading file, CRM receives its ID, and if CRM need to receive the file, it is enough to transfer its identifier and get need file, and the problem of safety and availability of free disk space is already being solved on the DFSboRD side.

DFSboRD is operating system independent and also supports various data transfer protocols which are reliable and time-tested. Also it is cheap, which is often crucial when choosing software. There are no restrictions on the number of employees, the amount of data, and so on.

5. Conclusions

The advantages of this approach, are the following:

1) Files can be stored in different places at the same time, but logically it will be 1 file, and if the physical storage fails, it is enough to make one of the replication blocks the main one, imperceptibly for other elements of the enterprise information infrastructure.

2) Files can be encrypted using a AES encryption method, that is, the file cannot be physically read without prior software decryption, using SB.

3) Convenience of working with files, the ability to integrate with other systems, since another system will not deal with the safety of the file itself, this is taken over by DFSboRD. Also, for some CRM systems, it is easier to store the Id of the file and receive it via some web server than to bother with storing information, allocating space.

4) More flexible handling. From the admin panel, you can create all the necessary links, grant rights, create a template for setting rights for directories and subdirectories, etc.

5) The system does not depend on storage and operation systems. For example, if the files are hosted on AWS and our connection is organized through the rclone utility, then it doesn't matter to us whether it is possible to issue rights to objects because we have ACLs placed directly in the database.

6) Several orders of magnitude faster search for file directories by name.

7) Portability, extensibility, scalability to work with different data transfer protocols.

The main disadvantage is that without a database it will be impossible to restore the structure of the file tree and their ACL. And this requires appropriate qualifications in the operation and maintenance of the database that will be used. There is also a need a certain amount of RAM, since for AES encryption it is necessary to process the entire file.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] (2022) NTFS: Microsoft. https://learn.microsoft.com/en-gb/windows-server/storage/file-server/ntfs-overview

[2] (2022) FAT: Microsoft. https://learn.microsoft.com/en-us/troubleshoot/windows-client/backup-and-storage/fat-hpfs-and-ntfs-file-systems

[3] (2022) AWS. https://docs.aws.amazon.com/s3/index.html

[4] Andrew, S. (2014) Tanenbaum, Herbert Bos Modern Operating Systems. 4th Edition, Vrije Universiteit Amsterdam, The Netherlands.

[5] (2012) PostgreSQL: Up and Running. 4th Edition, O'Reilly Media, Inc., Newton.

[6] Donovan, A.A.A. and Kernighan, B.W. (2015) The GO Programming Language. Addison-Wesley.

[7] Dusseault, L. (2007) HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). RFC 4918. https://doi.org/10.17487/rfc4918

[8] Postel, J. (1985) File Transfer Protocol (FTP). RFC 959. https://doi.org/10.17487/rfc0959

[9] Ford-Hutchinson, P. (2005) Securing FTP with TLS. RFC 4217. https://doi.org/10.17487/rfc4217

[10] Fielding, R. (1999) Hypertext Transfer Protocol—HTTP/1.1. RFC 2616. https://doi.org/10.17487/rfc2616

[11] Rescorla, E. (2000) HTTP over TLS. RFC 2818. https://doi.org/10.17487/rfc2818

[12] Rescorla, E. (2011) US Secure Hash Algorithms (SHA and SHA-Based HMAC and HKDF). RFC 6234.

[13] Schaad, J. (2003) Use of the Advanced Encryption Standard (AES) Encryption. RFC 3565. https://doi.org/10.17487/rfc3565

[14] Good, G. (2000) LDAP Data Interchange Format. RFC 2849.