



A shared distributed and redundant storage solution

Project ID: 19-002

IT 16106420 – T.R.N.R. Peiris

IT 16158528 – K.V.A. Sachintha

IT 16016026 – B.A. Ganegoda

IT16091276 – W.M.U.K.M.T. Bandara

Structure of the SRS
(Software Requirements Specification)

B.Sc. Special (Honors) Degree in Information
Technology

DECLARATION

We declare that this is our own work and this software requirement specification document does not incorporate with acknowledge any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

.....

T.R.N.R. Peiris

.....

B.A. Ganegoda

.....

K.V.A. Sachintha

.....

W.M.U.K.M.T. Bandara

Table of Contents

1	Introduction	7
1.1	Purpose	7
1.2	Scope	7
1.3	Definitions, Acronyms, and Abbreviations	7
1.4	Overview	8
2	Overall Descriptions	8
2.1	Product perspective	11
2.1.1	System interfaces	11
2.1.2	User interfaces	12
2.1.3	Hardware interfaces	14
2.1.4	Software interfaces.....	14
2.1.5	Communication interfaces.....	14
2.1.6	Memory constraints.....	15
2.1.7	Operations.....	15
2.1.8	Site adaptation requirements.....	16
2.2	Product functions	16
2.3	User characteristics	23
2.4	Constraints	23
2.5	Assumptions and dependencies	24
3	Specific requirements ⁽¹⁾ (for “Object Oriented” products)	25
3.1	External interface requirements	25
3.1.1	User interfaces	25
3.1.2	Hardware interfaces	29
3.1.3	Software interfaces.....	29
3.1.4	Communication interfaces.....	29
3.2	Classes/Objects	31
	34
3.3	Performance requirements	34
3.4	Design constraints	34
3.5	Software system attributes.....	34

3.5.1	Reliability	35
3.5.2	Availability	35
3.5.3	Security.....	35
3.5.4	Maintainability.....	35
4	Supporting information.....	36
4.1	Appendices	36
5	Reference	38

Table of Figures

Figure 1: Redundancy handling and hardware monitoring component overview	8
Figure 2: Blockchain base metadata storage and messenger module overview	9
Figure 3: Identity management, key derivation for file sharing and encryption overview	10
Figure 4 : Virtual storage pool overview	10
Figure 5: System Overview	12
Figure 6: Disk health monitoring interface	12
Figure 7: Login interface	13
Figure 8: Integrity check.....	13
Figure 9: Disk availability interface.....	14
Figure 10: Use case diagram.....	16
Figure 11: Redundancy module use case diagram.....	16
Figure 12: File Uploading/Downloading interface	25
Figure 13: Hardware monitoring interface	26
Figure 14: File sharing interface	27
Figure 15: Disk status interface	28
Figure 16: Hardware monitoring API class diagram	31
Figure 17: Redundancy handling API class diagram.....	31
Figure 18: Blockchain API class diagram	32
Figure 19: Messenger API class diagram	32
Figure 20 : File sharing module class diagram	33
Figure 21: Identity management module class diagram.....	33
Figure 22 : Disk availability module class diagram.....	34
Figure 23: Disk health API use case	36
Figure 24: Redundancy API use case.....	36
Figure 25: Identity management use case.....	37
Figure 26: Blockchain and messenger module use case	37

List of Tables

Table 1:Definitions and Abbreviations	7
Table 2:Feature Comparison.....	11
Table 3:Login use case scenario	17
Table 4:File uploading use case scenario	17
Table 5:File downloading use case scenario.....	17
Table 6:View disk health use case scenario	18
Table 7:View disk availability use case scenario.....	18
Table 8: Share a file use case scenario	18
Table 9: View files use case scenario.....	19
Table 10: Delete files use case scenario	19
Table 11: View the health use case scenario	19
Table 12: Create users use case scenario	20
Table 13: Delete user use case scenario	20
Table 14: Disable user account use case scenario.....	21
Table 15: Activate user account use case scenario	21
Table 16: Add devices use case scenario.....	21
Table 17: Remove device use case scenario	22
Table 18: Merge files use case scenario	22
Table 19: Assign storage space to the storage pool use case scenario	22
Table 20: Release storage space from the storage space use case scenario	23
Table 21:File uploading/downloading interface description	25
Table 22:Hardware monitoring interface description	26
Table 23: File sharing interface description	27
Table 24: Disk status interface description.....	28
Table 25: Redundancy handling API communication interfaces	29
Table 26:Hardware monitoring API communication interfaces	30
Table 27: Identity management module communication interfaces	30
Table 28: Key derivation and file sharing module communication interfaces	30

1 Introduction

1.1 Purpose

The purpose of this SRS document is to describe the requirements and the process related to developing VAULT, a shared distributed and redundant storage solution. The document will explain the purpose, features, functional and non-functional requirements, design constraint, project approach, constraints under which above mentioned application must operate and how the application will interact with the user. This document is designed not only with the intention of proposing the solution to a customer, in order to get the approval but also to give an idea to the developers and the other stakeholders about what are the functions available, in what order they needed to be done and the boundaries within which they need to work.

1.2 Scope

An ideal distributed storage solution must have the ability to provide the users with redundant, reliable, shared and secure access to the user's data and nevertheless must have the ability to scale and descend according to the storage space available while maintaining a higher performance level.

The security of the data that is stored on the cloud is uncertain with few exceptions. VAULT plans to address this problem in a decentralized methodology. It eliminates the authoritative figure of a cloud service provider such as Google or Amazon thus completely assuring privacy of stored data. VAULT will encrypt, segment and store data in multiple redundant locations (throughout the VAULT swarm / network) in the pooled storage space and use a custom blockchain to map the data hash to the actual location of the file segments. The blockchain will act as a DHT (Distributed Hash Table) in this scenario. The application will fragment and store file blocks throughout the peer to peer network in a redundant manner. The redundancy of data fragments is achieved by implementing the Erasure coding method which can be used to fragment the file into data and parity blocks. These blocks can then be used to recover the original file to a maximum of $N/2$ peer failures (N – Number of peers in the Vault network) depending on the Reed-Solomon configuration.

1.3 Definitions, Acronyms, and Abbreviations

Table 1:Definitions and Abbreviations

OS	Operating System
SMART	Self-Monitoring, Analysis, and Reporting Technology
API	Application Programming Interface
LAN	Local Area Network

1.4 Overview

Remainder of this document mainly can be divided into two sections plus appendix. First section is called overall description and it provides readers an understanding about the overall functionalities of the component, and the interactions of the component with the other components. Future more this section describes functional and nonfunctional requirements, design constraints which includes user interfaces, system interfaces, hardware interfaces and system constraints.

Second section which called specific requirements provides requirements specifications in a detailed manner. Specify requirements clearly for different audiences to understand by using various kinds of specification techniques. Future more software system attributes and performance requirements will be discussed in this section.

2 Overall Descriptions

The application mainly consists of four integral modules, which will be discussed in this document. Brief overview of them are as follows

Redundancy handling and hardware monitoring module

This module mainly can be divided into two separate functions, Redundancy handling and Hardware monitoring. Redundancy handling function allows users to store their data in a redundant and distributed manner, where each and every file will be divided in to, N number of data segments plus M number of parity segments, which will be generated automatically using erasure coding concept, where $M=N/2$. Altogether K ($K = N + N/2$) number of files will be output from the system and they will be scattered among K number of nodes available in the network. Advantage of the above-mentioned process is that even though $N/2$ number of nodes are down from the K number of nodes at the downloading time, the original file can be regenerated. Hardware monitoring function assist in the process of regenerating the original file to check whether a certain device is online or not and also give an overview about hard disk health to the administrative users, which will help them in the process of maintain consistency.

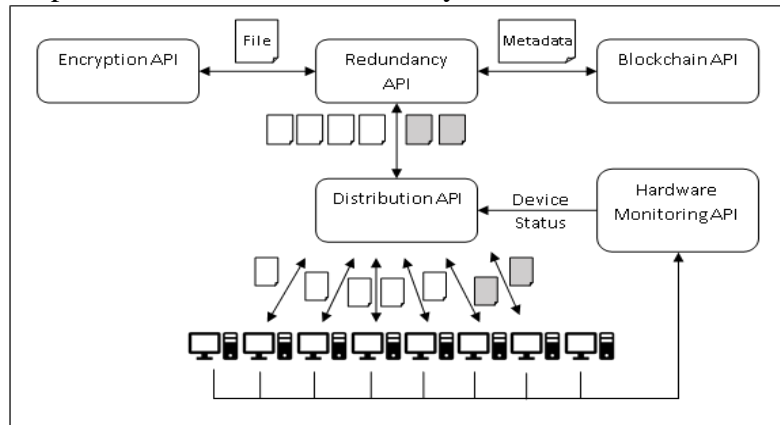


Figure 1: Redundancy handling and hardware monitoring component overview

Blockchain based metadata storage and Messenger module

The functionality of the Blockchain API depends on the details provided by the Redundancy API such as file metadata, fragment metadata, file hashes etc. The Blockchain API will then process the data received from the Redundancy module into a “Block”. All details pertaining to a single file uploaded by a user will be saved into a single block. The block will contain transactions representing each fragment. A file with 5 fragments for instance would contain 5 “transactions” within the “Block”. The Messenger API will then relay the block metadata to all other nodes in a JSON encoded payload via the WebSocket connection.

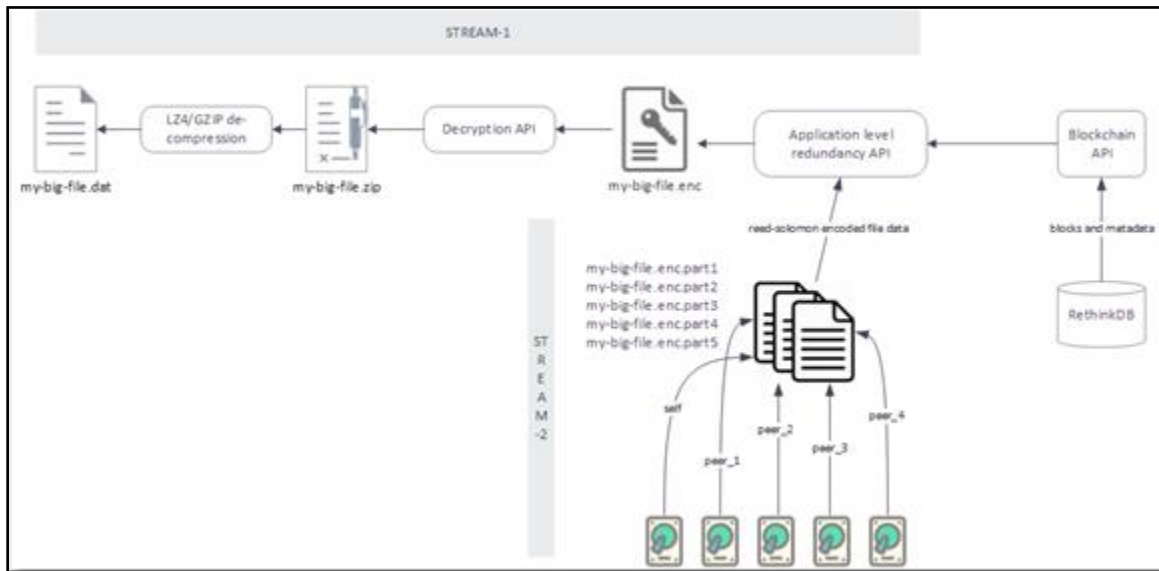


Figure 2: Blockchain base metadata storage and messenger module overview

Identity management, key derivation for file sharing and encryption module

Node authentication by public key cryptography can be used to identify the trusted nodes in the cluster. Every user has a list of public keys of every node stored in the custom blockchain. A Master key(M) will be sent to validate the nodes that is connected in the network. By comparing the master key, the nodes will be able to verify the secure nodes. The encryption will be done using AES 256 for maximum speed and security. The hash values that are stored in the blockchain will provide a validation of the files or file parts that are stored in the cluster and ensure the integrity of the files. When considering the sharing scenario, a link will be shared with the intended user that has the locations of the file parts that are stored in the network. With that link, the user can access the files by downloading the file into the storage. A separate temporary key will be used to decrypt the file. That key will be shared with the trusted node using Public-key cryptography.

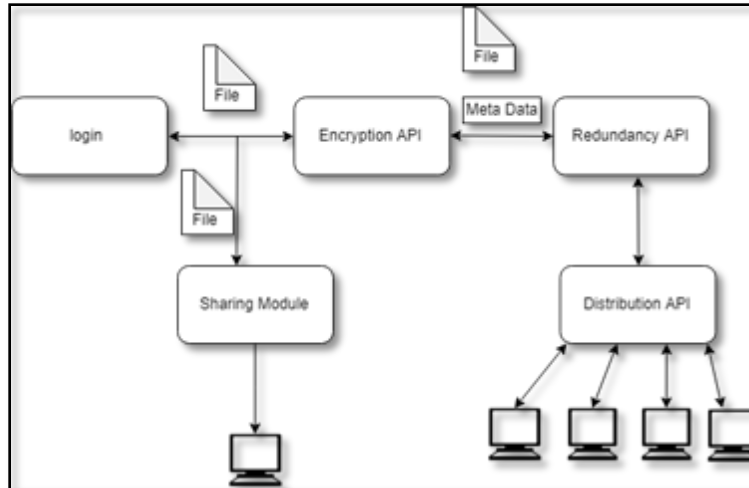


Figure 3: Identity management, key derivation for file sharing and encryption overview

Virtual Storage Allocation and Deallocation

Virtual storage pool management is a part of the storage management and it provides the facility to create virtual storage pool for the user to store data. The storage space will be created from the virtual storage pool which will collect physical storage disks from the network. There are two option for creating a disk image: fixed-size or dynamically allocated. If create a fixed-size storage from virtual pool an image file will be created roughly the same size as the virtual disk's capacity. (when create a 10 GB disk space, file with a size of 10 GB will be created). Dynamically allocated storage will initially be very small and not occupy any space for unused virtual disk sectors but will grow every time a disk sector is written to for the first time, until the drive reaches the maximum capacity chosen when the drive was created.

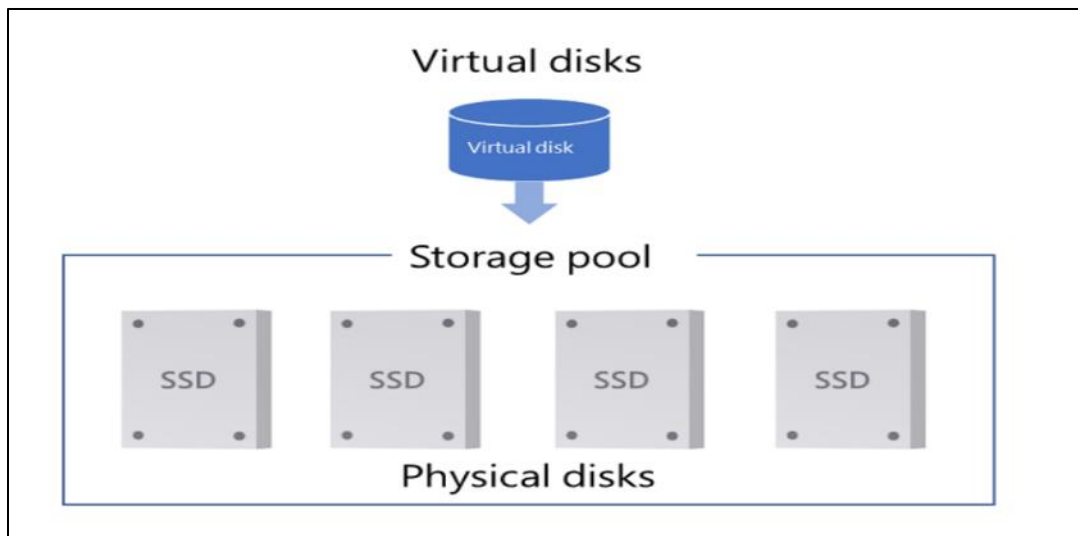


Figure 4 : Virtual storage pool overview

2.1 Product perspective

Below diagram will describe the similarities and the differences of the final product with the existing similar products.

Table 2:Feature Comparison

Features	MooseFS [1]	IBM Spectrum Scale [2]	OCFS 2 [3]	OrangeFS [4]	BWFS [5]	Minio [6]	Ceph [7]	VAULT
High Availability	✓	✓	✓	✓		✓	✓	✓
Scalability	✓	✓	✓	✓	✓	✓	✓	✓
Minimal Investment	✓			✓		✓	✓	✓
Big Data Support	✓	✓	✓	✓		✓	✓	✓
Data encryption		✓				✓	✓	✓
Data Recovery	✓	✓		✓	✓	✓	✓	✓
Platform independent	✓	✓						✓
Security		✓	✓	✓		✓	✓	✓
Data Redundancy					✓	✓		✓
Minimum additional storage for backup								✓
Blockchain integration								✓
Easy to setup and run								✓
File Sharing								✓

2.1.1 System interfaces

Since the system will be designed and developed in line with the modular approach in mind, each research component will be designed, developed and tested out individually as modules which then can be imported and assembled in the final software product. Since JavaScript is used as the developing language, the application can be run inside any Linux, MacOS and Windows environment (Platform independent). Since the final product acts as standalone solutions it won't be interacting with the other existing applications other than the web browser which will be needed to display the user interfaces of the web-application. Although hardware monitoring function will be interacting with the OS build in features

such as SMART to gather hardware health related data. The blockchain will be stored in a NoSQL database, namely Rethink DB and therefore will act as a dependency to the program.

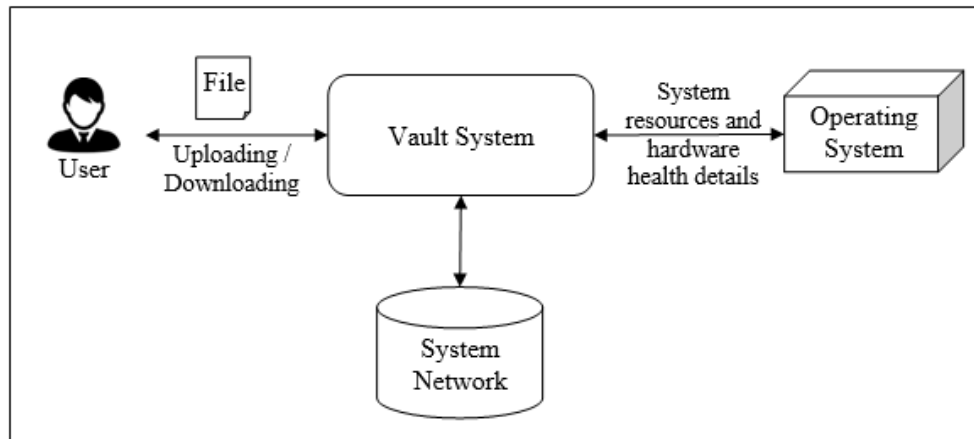


Figure 5: System Overview

2.1.2 User interfaces

Since the redundancy function is a core module which acts in the middle of the file uploading and downloading process it doesn't require any specific interface, it takes/gives data from/to encryption module and then send/receive them to/from distribution module, hence the interface which is responsible for triggering the redundancy function, interface which will be used by the users to upload and download files will be included as the redundancy function interface and will be mentioned in section 3.

Hardware monitoring function gathers data from the Operating Systems through the network and displays them in an interactive manner to the administrative users.

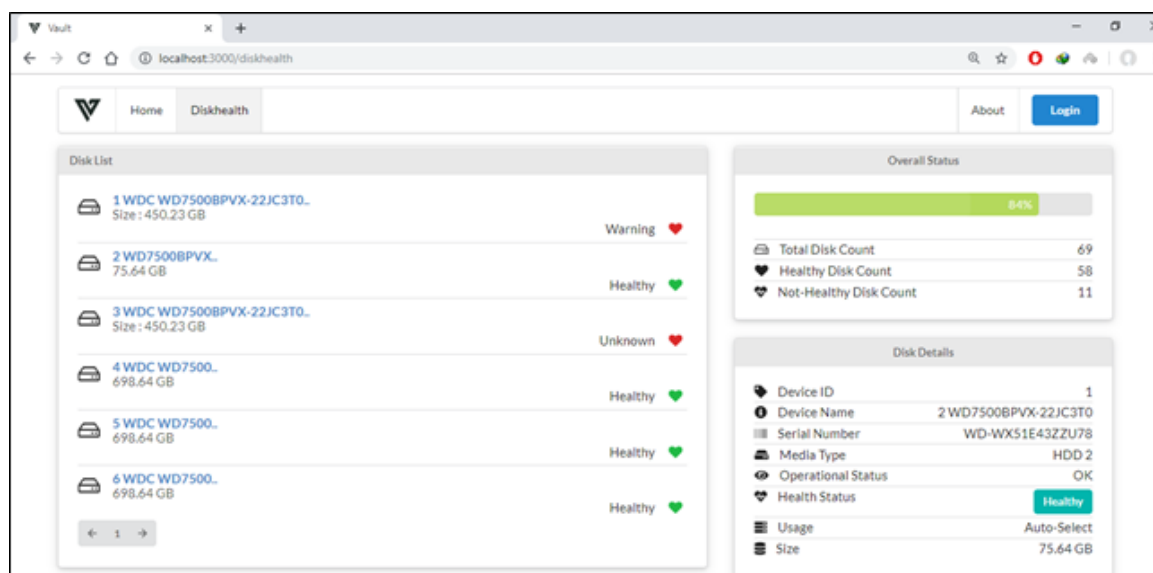
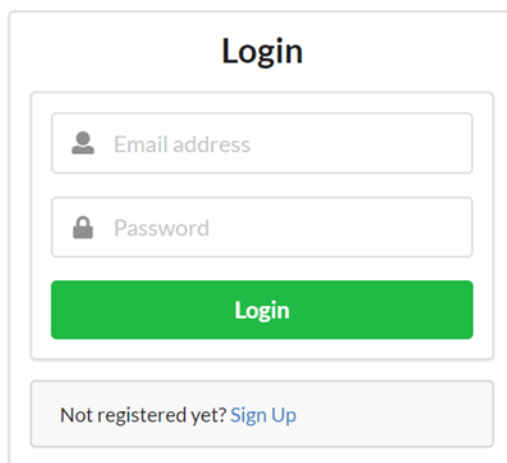


Figure 6: Disk health monitoring interface

Since the Blockchain API and the Messenger API are low level core modules, the necessity of a user interface is redundant. However, callbacks can be defined such that status updates in the swarm are portrayed in the default user interface. An example of this could be a new node being added to the swarm. The addition of the node will be identified by the Messenger API which then executes a UI callback to update the existing node list in the user interface.

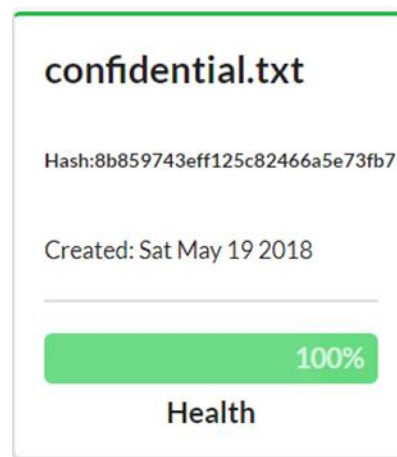
Since identity management consists of the user authentication process, node authentication and the process of file integrity check, these modules excluding the user authentication, it doesn't require any specific interface. This module takes data at beginning and then encrypts it. And then validate the nodes in the process. Interface which will be used to share the files, encrypt the files and file integrity using hashing will be included in the section 3.

User authenticate will enable to validate the users who are valid and give access to those users only. A user can log on to any computer and can sign-in to their account at any given moment.



The login interface is a white rectangular box with a light gray border. At the top, the word "Login" is centered in bold black text. Below it, there are two input fields: the first is labeled "Email address" with a person icon, and the second is labeled "Password" with a lock icon. A green "Login" button is positioned below the password field. At the bottom, there is a light gray box containing the text "Not registered yet? [Sign Up](#)".

Figure 7: Login interface



The integrity check interface is a white rectangular box with a light gray border. At the top, the filename "confidential.txt" is displayed in bold black text. Below it, the hash "Hash:8b859743eff125c82466a5e73fb7" is shown. The creation date "Created: Sat May 19 2018" is listed below the hash. A green progress bar is shown with "100%" written inside it. At the bottom, the word "Health" is centered in bold black text.

Figure 8: Integrity check

The disk health interface indicates all connected physical drives. Here, administrator can view what are the workstation currently online/offline. Also, can view workstation storage status. When workstation storage capacity low it will indicate as a percentage.

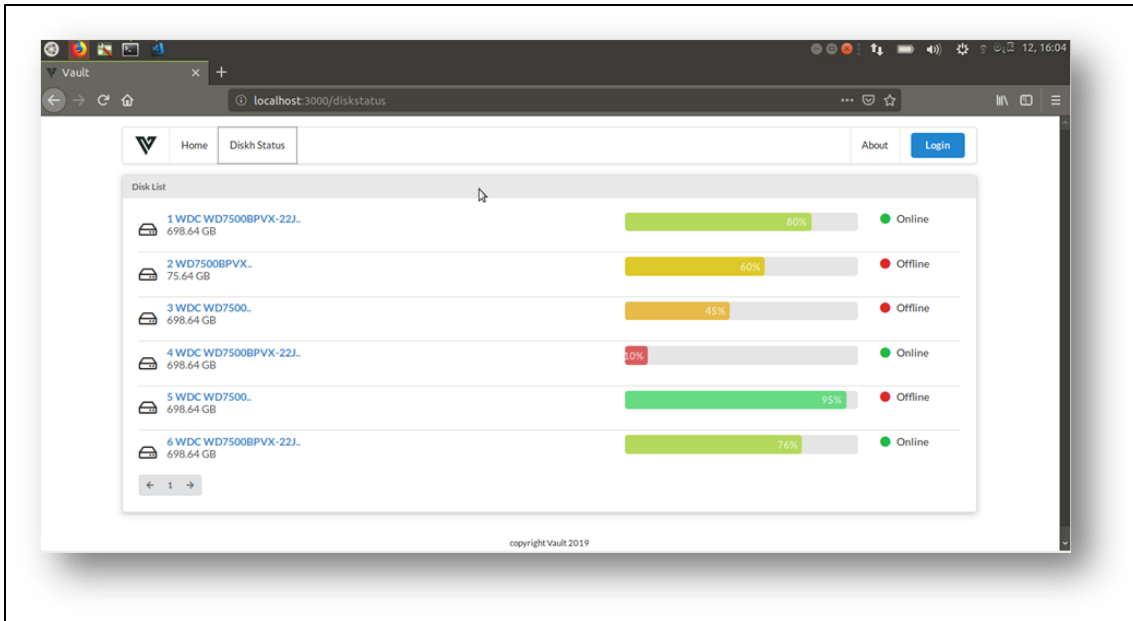


Figure 9: Disk availability interface

2.1.3 Hardware interfaces

Since the software does not directly interact with Hardware level components and doesn't rely on how storage devices handle data, there is no requirement to document hardware interfaces.

2.1.4 Software interfaces

Most of the modules included in the application won't be connecting with any external applications other than the OS built-in features such as SMART to gather disk health related information. Only requirement needed to run the component successfully is a physical machine with NodeJS installed on them (Platform Independent).

The Blockchain API and the Messenger API will only bind to the base operating system via the NodeJS middleware and both of them will not employ an RPC based software interface because both modules work in the same application thread and the runtime environment is shared.

2.1.5 Communication interfaces

Most of the APIs act as middle layer processes, hence they won't be needing any external communication interfaces other than the interfaces needed to communicate with the other APIs within the application.

Redundancy handling module will need two specific internal interfaces to connect with the encryption module and the distribution module. Hardware monitoring module will be

mainly needing two communication interfaces as well to communicate with the internal module called distribution module and another interface to communicate with the OS.

Encryption handling module will need two specific internal interfaces to connect with the Redundancy module and the distribution module.

The Blockchain API will establish 2-way communication with the backend low-level document storage database (rethink-db.) using the rethink-db. client library. The library will issue HTTP requests to the database server which will then return a JSON object, again via HTTP. The Messenger API consists of a “socketclient” and a “socketserver” since the node works in a peer-to-peer basis. The socketclient is an array list of socket objects that maps with a 1:1 relationship to other peers in the swarm. The socketserver handles socketclient requests that are made via websockets. A typical node in a swarm of n nodes will contain an array of $(n-1)$ sockets. However, the entire swarm (n) will only contain n socketserver.

The communication among the virtual storage management will be established over the local area network. To identify the workstation this required local area connection. It will load the user storage data from the physical hard drives to the web application. However, the storage module will be communicating with the operating system.

2.1.6 Memory constraints

Since the application is engineered to use disk space more than Memory, a machine with at least 2GB of RAM is sufficient.

2.1.7 Operations

Prior to use, Docker application will be deployed among the users by using the internal network or else physically. Application basically have two types of user accounts called administrator accounts and normal accounts.

All the user account related activities will be carried out by the administrative users using the web application through the web browsers. Administrators can connect/ disconnect workstations from the system, can view available workstations at a certain time and can view the disk health information related to all the workstations connected. Administrators most important task is to assign storage space from the workstations to create the storage pool where all the data will be stored.

When a normal user can finally can access the web application from the web browser a logging page will be displayed to the user, and users can use the logging credentials provided by the administrators to logged in to the application. When a normal user can log in they can upload, download files to the system which will be stored among the network in a secure and redundant manner.

2.1.8 Site adaptation requirements

In order to run the application users must complete few tasks in a certain order which will be presented below in the order that users must carry them out. When users accessing the application, they will be guided using English language.

1. NodeJS must be installed
2. Docker application which contains the system must be deployed
3. The Docker image must contain Rethink-DB
4. All the workstations must connect to the local network.
5. At least there must be 6-10 users that can contribute 10 GB storage space by each
6. Web browser must be installed
7. Users must have login credentials to access the web application

2.2 Product functions

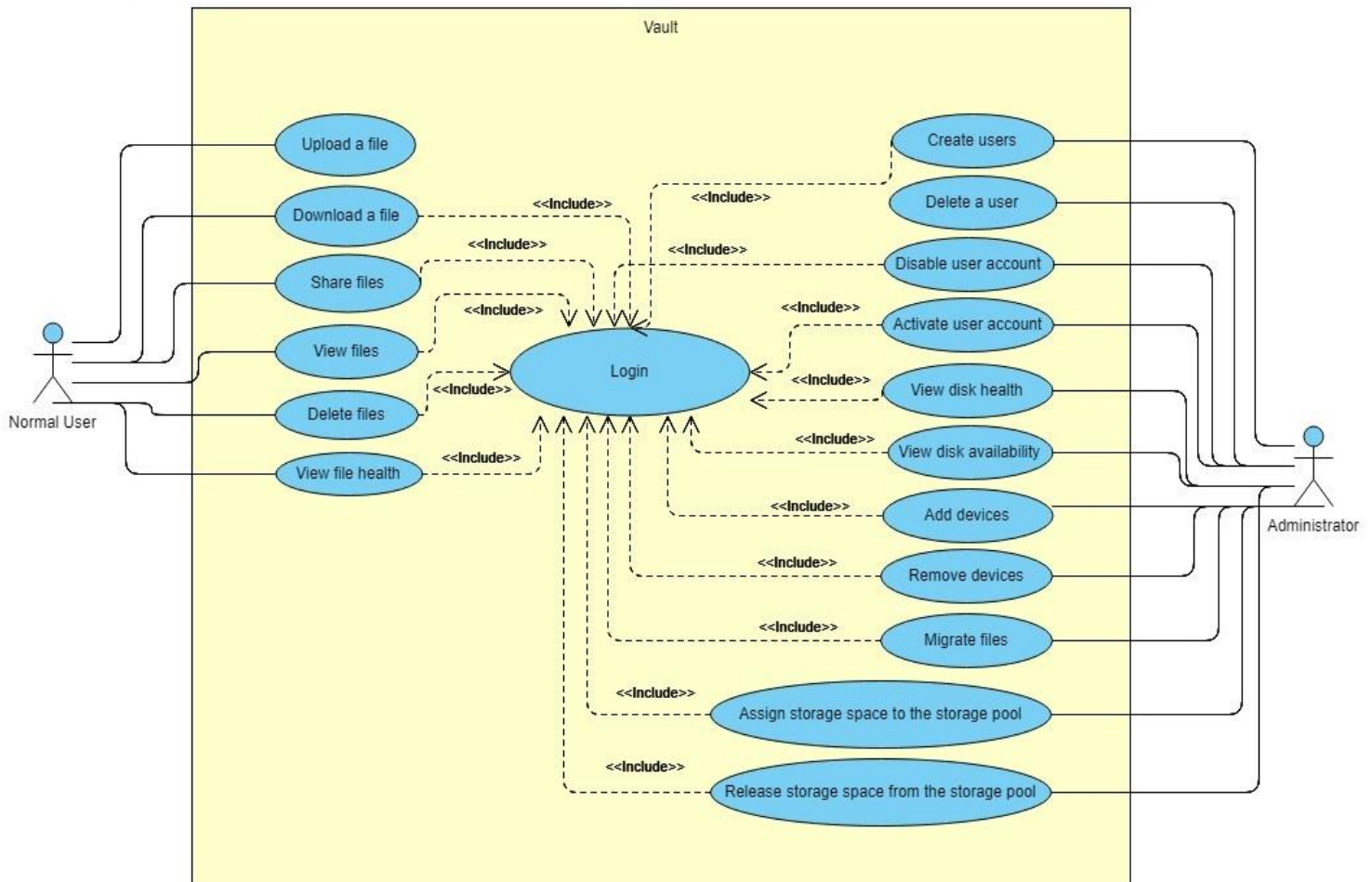


Figure 10: Use case diagram

Login

Table 3:Login use case scenario

Use Case No	01
Use Case	Login
Actors	Normal User, Administrator
Pre-Conditions	User must be a registered user
Flow of Event	1. Enter the username and password 2. Click Login
Post Conditions	Allow user to the web application
Alternatives	Display an error message to inform that the user credentials are invalid.

Upload a file

Table 4:File uploading use case scenario

Use Case No	02
Use Case	Upload a file
Actors	Normal User
Pre-Conditions	User must be logged in
Flow of Event	1. Press on “upload a file” button 2. Select a file 3. Press on “Submit” button 4. Web application will display message “Successful Uploaded”
Post Conditions	Display “Upload Successful” message
Alternatives	Display an error message to inform user that the file won’t be uploaded due to some reason.

Download a file

Table 5:File downloading use case scenario

Use Case No	03
Use Case	Download a file
Actors	Normal User
Pre-Conditions	User must be logged in
Flow of Event	1. Select the file that needs to download 2. Click on “Download” button 3. File will be downloaded
Post Conditions	Download the file
Alternatives	Display an error message to inform user that the file won’t be downloaded due to some reason.

View disk health

Table 6:View disk health use case scenario

Use Case No	04
Use Case	View disk health
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	1. Select “Disk Health” tab. 2. Disk health will be displayed.
Post Conditions	Disk health of all the devices should be displayed.
Alternatives	Display an error message if there are any complications in connecting to the network.

View disk availability

Table 7:View disk availability use case scenario

Use Case No	05
Use Case	View disk availability
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	3. Select “Disk Status” tab. 4. Disk availability will be displayed.
Post Conditions	Disk availability of all the devices should be displayed.
Alternatives	Display an error message if there are any complications in connecting to the network.

Share a file

Table 8: Share a file use case scenario

Use Case No	06
Use Case	Share a file
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	1. Select a file from the file directory. 2. Press on “Share” button 3. Type in the temporary password to encrypt the file. 4. Press on “Select recipient” button. 5. Select the intendent recipient 6. Press “Ok” button to send

	7. Web application will display message “Link shared with the recipient”
Post Conditions	Web application will display message “Link shared with the recipient” and a link will be shared with the intendent recipient
Alternatives	Display an error message if there are any complications in sharing the link or selecting the user.

View files

Table 9: View files use case scenario

Use Case No	07
Use Case	View file s
Actors	Normal User
Pre-Conditions	User must be logged in
Flow of Event	1. Web application will automatically display the files when user logged in
Post Conditions	
Alternatives	

Delete files

Table 10: Delete files use case scenario

Use Case No	08
Use Case	Delete files
Actors	Normal User
Pre-Conditions	User must be logged in
Flow of Event	1. Select a file from the directory 2. Press on “delete” button 3. Press on “Ok” button in the conformation window. 4. Web application will display message “deleted Successfully”
Post Conditions	Display “Deleted Successfully” message
Alternatives	Display an error message to inform user that the file cannot be deleted.

View the health

Table 11: View the health use case scenario

Use Case No	09
Use Case	View the health
Actors	Normal User
Pre-Conditions	User must be logged in

Flow of Event	<ol style="list-style-type: none"> 1. Select a file from the directory 2. Press on “Health” button 3. Web application will display the health of the files.
Post Conditions	Display the health of the file in a window.
Alternatives	-

Create users

Table 12: Create users use case scenario

Use Case No	10
Use Case	Create users
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	<ol style="list-style-type: none"> 1. Input user details file from the directory 2. Press on “Create user” button 3. Type in the user details. 4. Press on “Add user” button 5. Press on “Ok” button. 6. Web application will display message “User created Successfully”
Post Conditions	Display “User created Successfully” message
Alternatives	Display an error message to inform user that the user cannot be created.

Delete user

Table 13: Delete user use case scenario

Use Case No	11
Use Case	Delete user
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	<ol style="list-style-type: none"> 1. Select a user from the user list 2. Press on “delete user” button 3. Press on “confirm” button. 4. Web application will display message “User deleted Successfully”
Post Conditions	Display “User deleted Successfully” message
Alternatives	Display an error message to inform user that the user cannot be deleted.

Disable user account

Table 14: Disable user account use case scenario

Use Case No	12
Use Case	Disable user account
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	1. Select a user from the user list 2. Press on “disable user” button 3. Press on “confirm” button. 4. Web application will display message “User disabled Successfully”
Post Conditions	Display “User disabled Successfully” message
Alternatives	-

Activate User account

Table 15: Activate user account use case scenario

Use Case No	13
Use Case	Activate user account
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	1. Select a user from the user list 2. Press on “Active user” button 3. Press on “confirm” button. 4. Web application will display message “User activated Successfully”
Post Conditions	Display “User activated Successfully” message
Alternatives	-

Add devices

Table 16: Add devices use case scenario

Use Case No	14
Use Case	Active user account
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	1. Press on “Add device” button 2. Input device details 3. Press on “confirm” button. 4. Web application will display message “Device Added Successfully”

Post Conditions	Display “Device Added Successfully” message
Alternatives	-

Remove Device

Table 17: Remove device use case scenario

Use Case No	15
Use Case	Remove device
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	1. Press on “Remove device” button 2. Select the device 3. Press on “confirm removal” button. 4. Web application will display message “Device removed Successfully”
Post Conditions	Display “Device removed Successfully” message
Alternatives	-

Migrate Files

Table 18: Merge files use case scenario

Use Case No	16
Use Case	Migrate Files
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	1. Select a file 2. Press on “Migrate file” button 3. Press on “confirm Migrate” button. 4. Web application will display message “File migrated Successfully”
Post Conditions	Display “File migrated Successfully” message
Alternatives	-

Assign storage space to the storage pool

Table 19: Assign storage space to the storage pool use case scenario

Use Case No	17
Use Case	Assign storage space to the storage pool
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	1. Select the “allocate storage” tab directory

	2. Select storage 3. Input the storage size that wants to be allocated. 4. Press on “allocate” button 5. Web application will display message “Allocated Successfully”
Post Conditions	Display “Allocated Successfully” message
Alternatives	Display an error message to inform user that the storage space cannot be allocated.

Release Storage space from the storage pool

Table 20: Release storage space from the storage space use case scenario

Use Case No	18
Use Case	Release Storage space from the storage pool
Actors	Administrator
Pre-Conditions	User must be logged in
Flow of Event	5. Select the “deallocate storage” tab directory 6. Select storage 7. Press on “deallocate” button 8. Web application will display message “Deallocated Successfully”
Post Conditions	Display “Deallocated Successfully” message
Alternatives	Display an error message to inform user that the storage space cannot be deallocated.

2.3 User characteristics

There can be mainly two types of users in the system administrators and normal users.

- Administrator- Software/hardware professional who will be configuring the system and maintain the consistency of the system.
- Normal Users- Anyone with the basic knowledge of computing who will be using the system to store their files and download them when needed.

2.4 Constraints

Redundancy handling module constraints

- Module to be run properly at least there must be 6 workstations
- NodeJS must be installed

Hardware monitoring module constraint

- NodeJS must be installed

Blockchain API module constraints

- The module can be tested when Rethink DB is installed.
- Module input and output is strictly set to JSON
- NodeJS must be installed

Messenger API module constraints

- Uses 2 channels to synchronize blockchain responses and control information respectively.

Identity management module constraints

- NodeJS must be installed

key derivation for file sharing and encryption module constraint

- NodeJS must be installed

Virtual storage management module constraints

- NodeJS must be installed
- At least 10 GB Hard disk free space
- User login and the user authentication is must
- User should have the Local Network connection to deal with the application.

2.5 Assumptions and dependencies

The Blockchain API module relies on the “thinky” ORM tool to communicate with Rethink-db. The Messenger API relies on socket.io for WebSocket implementation, other than that there was no assumption made due to the reason most modules are platform independent and require no additional dependency from third party applications application or any additional hardware other than the assist get from the OS.

3 Specific requirements⁽¹⁾ (for “Object Oriented” products)

3.1 External interface requirements

3.1.1 User interfaces

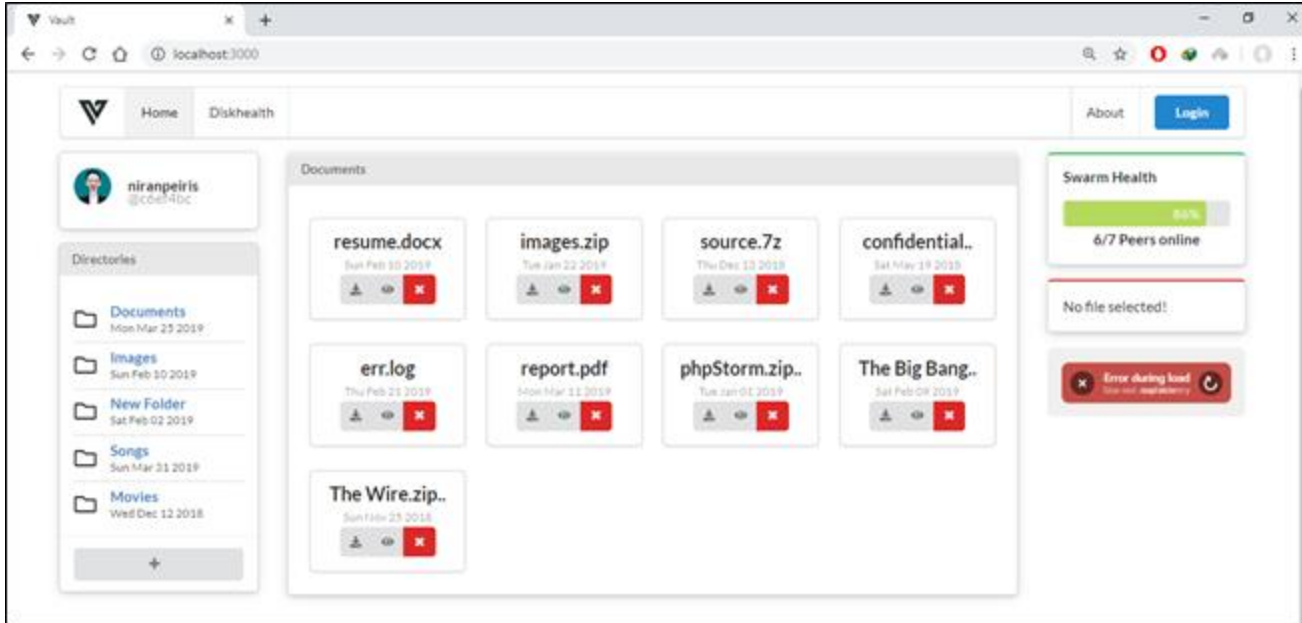


Figure 12: File Uploading/Downloading interface

Table 21:File uploading/downloading interface description

Name of item	File uploading/downloading interface
Description of purpose	Input files needed to the encryption and redundancy modules which will be segmented by the redundancy module will be input to the system via this interface.
Source of input or destination of output	Any type of files
Valid range, accuracy and/or tolerance	100%
Units of measure	Bytes
Timing	-
Relationships to other inputs/outputs	Original file information such as name, size, extension, locations of the segments will be outputted to the blockchain API.
Screen formats/organization	Screen is organized in a monitor view
Window formats/organization	-
Data formats	Bytes

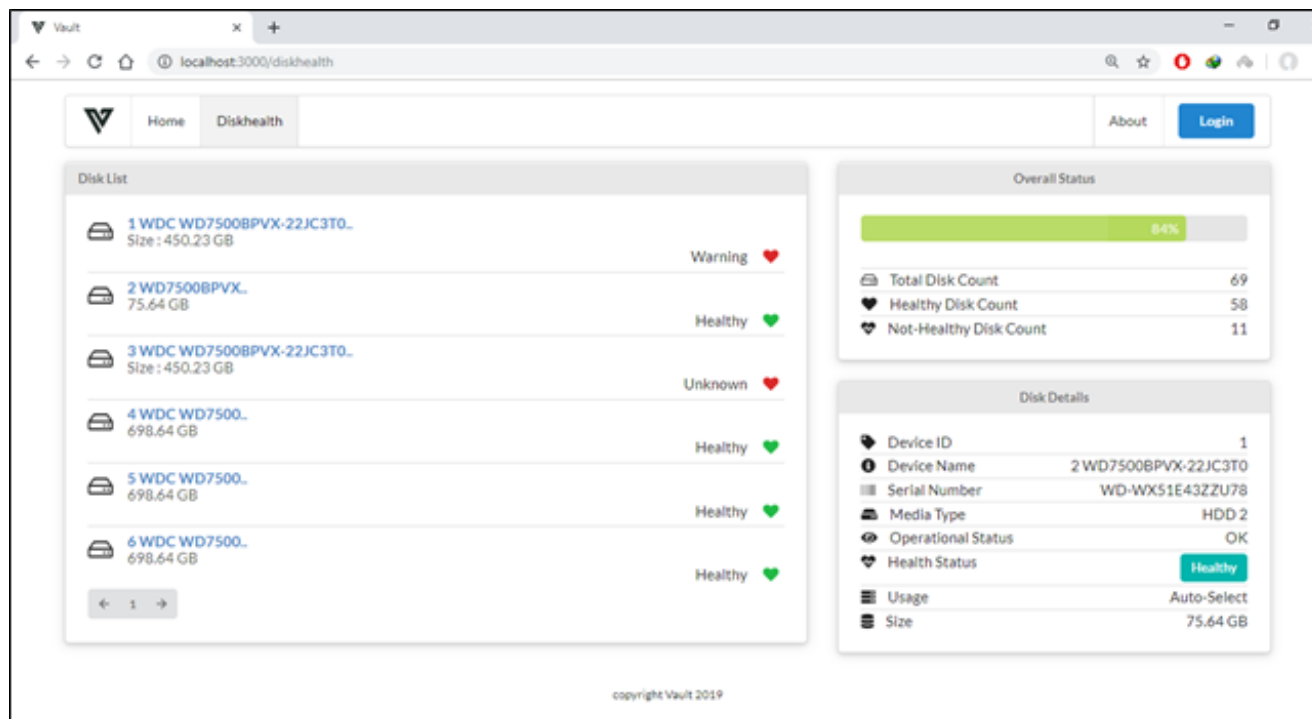


Figure 13:Hardware monitoring interface

Table 22:Hardware monitoring interface description

Name of item	Hardware monitoring interface
Description of purpose	Health and the availability of the each and every hard disk will be presented to the user by this interface.
Source of input or destination of output	Health related information from Operating system
Valid range, accuracy and/or tolerance	80%
Units of measure	-
Timing	-
Relationships to other inputs/outputs	Availability information will be outputted to the distribution API when segments of a file is recollecting.
Screen formats/organization	Screen is organized in a monitor view
Window formats/organization	-
Data formats	Alphanumeric

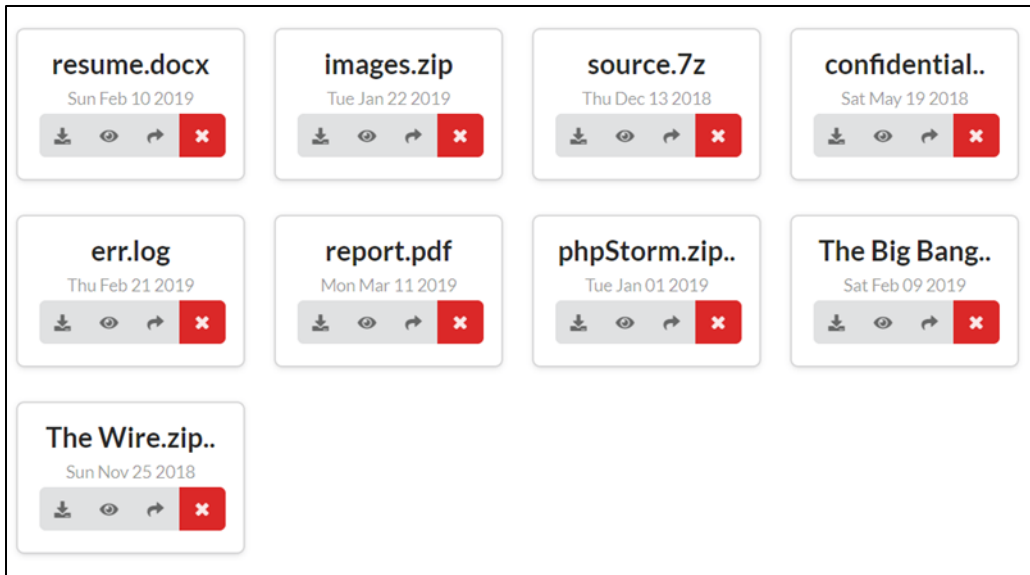


Figure 14: File sharing interface

Table 23: File sharing interface description

Name of item	File Sharing Interface
Description of purpose	User will be able to share the files of his or her that are stored with intended parties with this interface.
Source of input or destination of output	Intended node address from the blockchain
Valid range, accuracy and/or tolerance	100%
Units of measure	-
Timing	Depends on the file size
Relationships to other inputs/outputs	Availability information will be outputted to the distribution API when segments of a file is recollecting.
Screen formats/organization	Screen is organized in a monitor view
Window formats/organization	-
Data formats	-

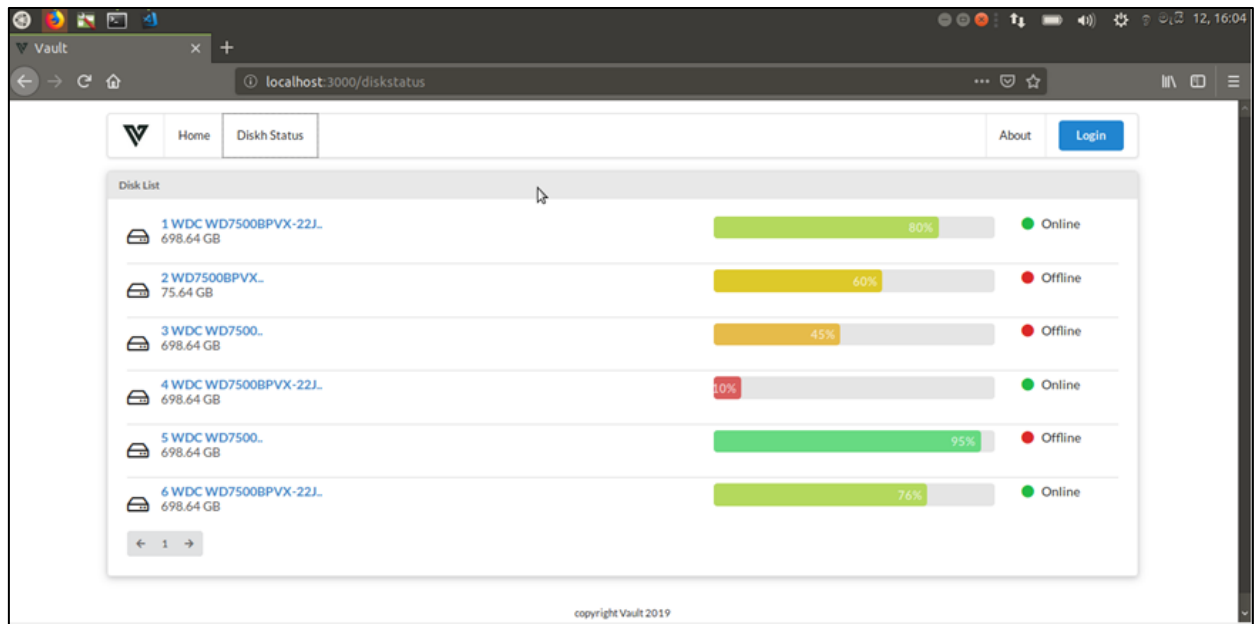


Figure 15: Disk status interface

Table 24: Disk status interface description

Name of item	Disk status
Description of purpose	Workstation availability and storage status of the each and every hard disk will be presented to the user by this interface.
Source of input or destination of output	Storage related information from Operating system
Valid range, accuracy and/or tolerance	100%
Units of measure	-
Timing	-
Relationships to other inputs/outputs	Storage status information will be outputted to the distribution API when segments of a file is recollecting.
Screen formats/organization	Screen is organized in a monitor view
Window formats/organization	-
Data formats	Alphanumeric

3.1.2 Hardware interfaces

Since the software does not directly interact with Hardware level components and doesn't rely on how storage devices handle data, there is no requirement to document hardware interfaces.

3.1.3 Software interfaces

The Blockchain API and the Messenger API will only bind to the base operating system via the NodeJS middleware. Both APIs mentioned above will not employ an RPC based software interface because both modules work in the same application thread and the runtime environment is shared. Hence other than NodeJS and OS the product mentioned in this document won't be interacting with any other software application.

3.1.4 Communication interfaces

The Blockchain API will establish 2-way communication with the backend low-level document storage database (rethink-db.) using the rethink-db. client library. The library will issue HTTP requests to the database server which will then return a JSON object, again via HTTP. The Messenger API consists of a "socketclient" and a "socketserver" since the node works in a peer-to-peer basis. The socketclient is an array list of socket objects that maps with a 1:1 relationship to other peers in the swarm. The socketserver handles socketclient requests that are made via websockets. A typical node in a swarm of n nodes will contain an array of $(n-1)$ sockets. However, the entire swarm (n) will only contain n socketserver.

Both redundancy handling and hardware monitoring APIs act as middle layer processes, hence they won't be needing any external communication interfaces other than the interfaces needed to communicate with the other APIs within the application and the OS.

Table 25: Redundancy handling API communication interfaces

Interface	External Module	Connection details
Interface 1	Encryption Module	When user uploads a file, it will be encrypted by the encryption module and then encrypted file will be send to the redundancy API to be segmented. When user downloads a file, segments will be used to regenerate the original file and then it will be sent to encryption module to be decrypted.
Interface 2	Distribution Module	When a file is segmented, segments will be sent to the distribution function to scatter them across the network. When redundancy API needs segments to regenerate an original file a request will be made to the distribution function and it will collect segments from the network.

Table 26:Hardware monitoring API communication interfaces

Interface	External Module/ Feature	Connection details
Interface 1	Distribution Module	Gather disk health and availability information from all over the network through distribution function.
Interface 2	OS	Gather disk health related information from OS.

The local network connection will be needed by the users as well as for the storage management purpose. Since the storage management deal with physical hard drives, there should be an LAN network connection. The LAN network connection will be used to communicate in between application and all other physical hard drives in the network

Table 27: Identity management module communication interfaces

Interface	External Module	Connection details
Interface 1	Login module	When user tries to log into the system the user is verified using a simple username and password basis authentication and this module can enable creation of users as well. Multiple failed login attempts will disable a user access from the system.
Interface 2	Common Module	When a file is segmented and ready to send to nodes for storing, this module will be used to verify the trusted nodes in the cluster

Table 28: Key derivation and file sharing module communication interfaces

Interface	External Module/ Feature	Connection details
Interface 1	Redundancy Module	Gather disk health and availability information from all over the network through distribution function.

3.2 Classes/Objects

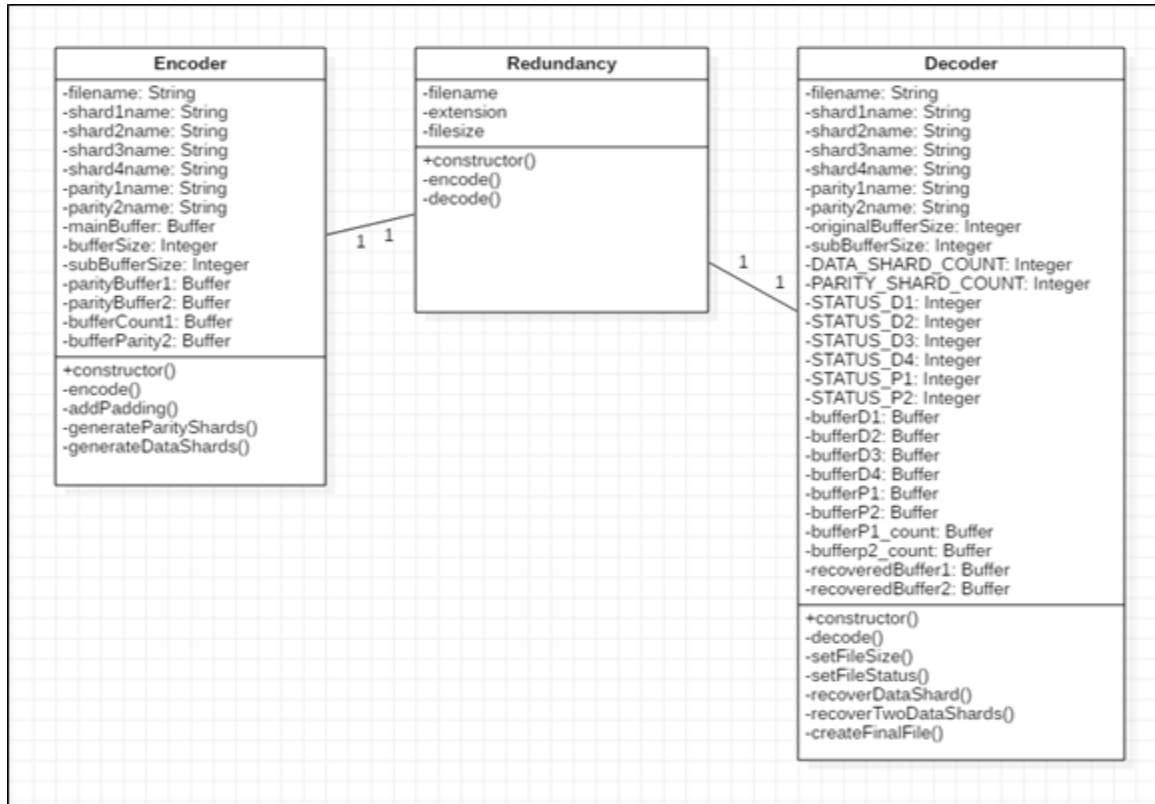


Figure 17: Redundancy handling API class diagram

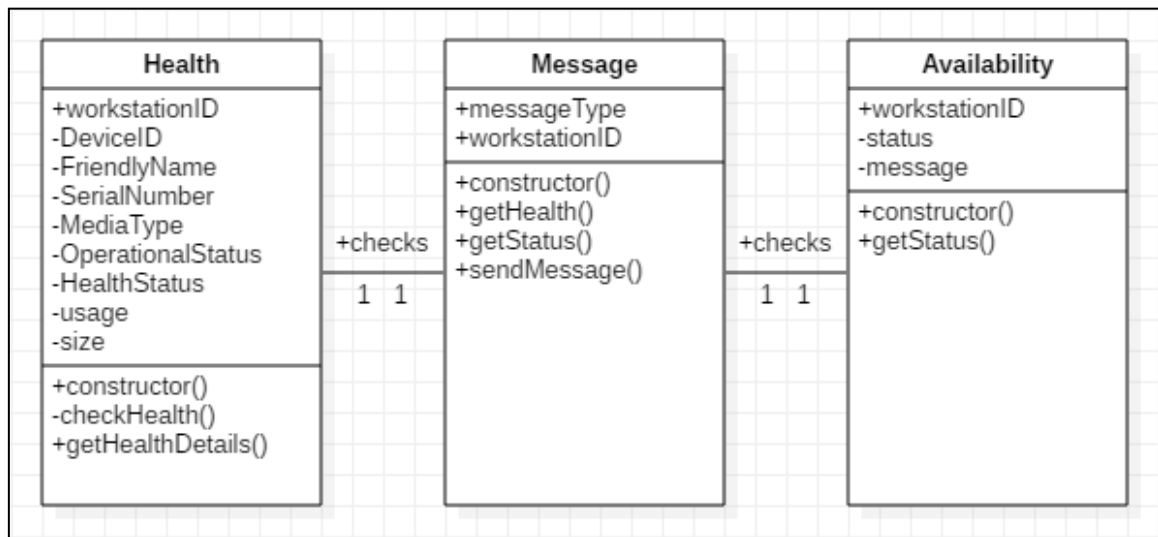


Figure 16: Hardware monitoring API class diagram

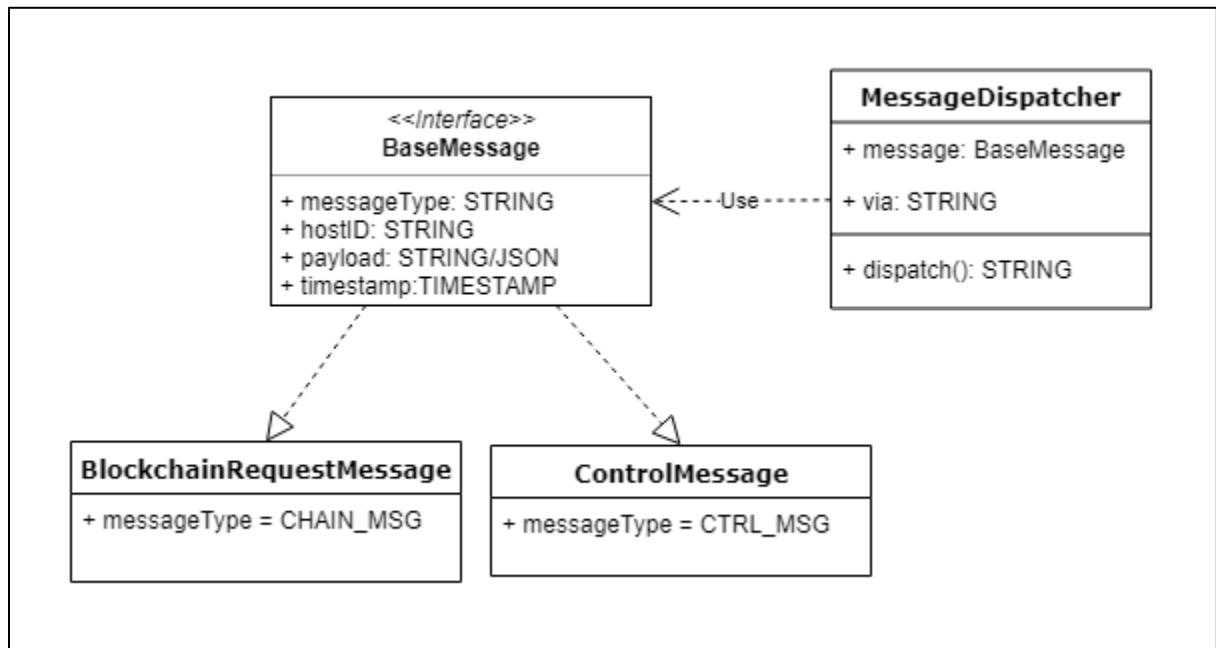


Figure 19: Messenger API class diagram

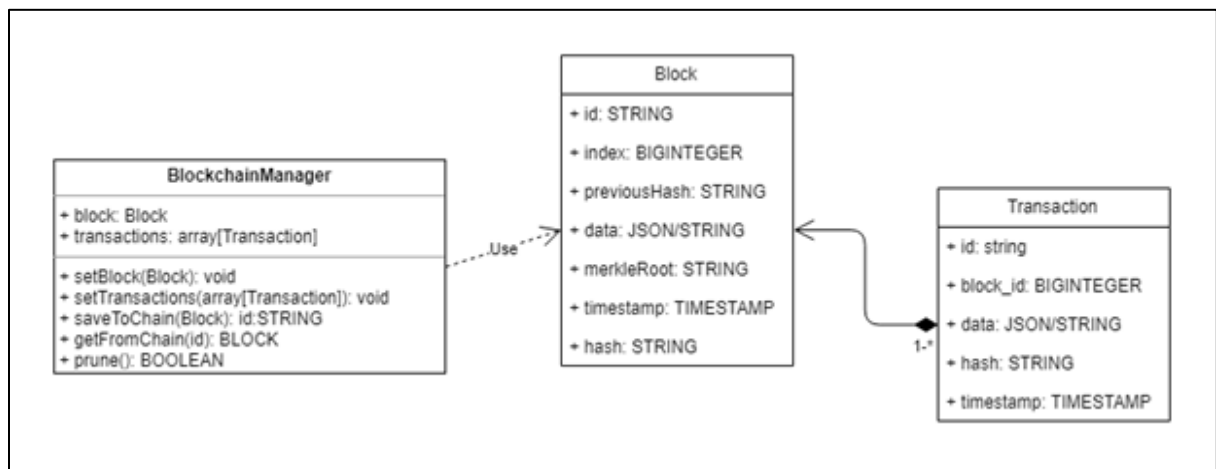


Figure 18: Blockchain API class diagram

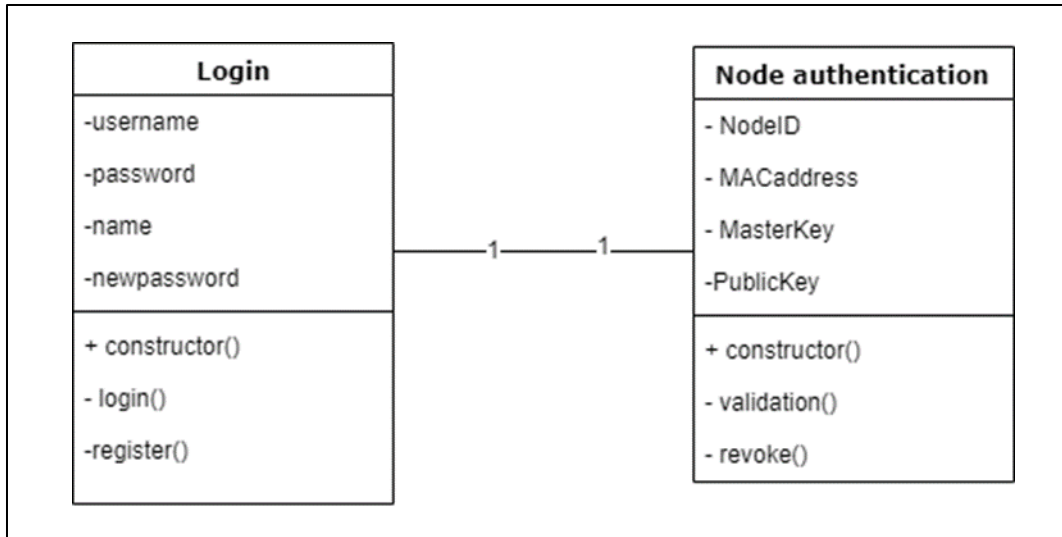


Figure 21: Identity management module class diagram

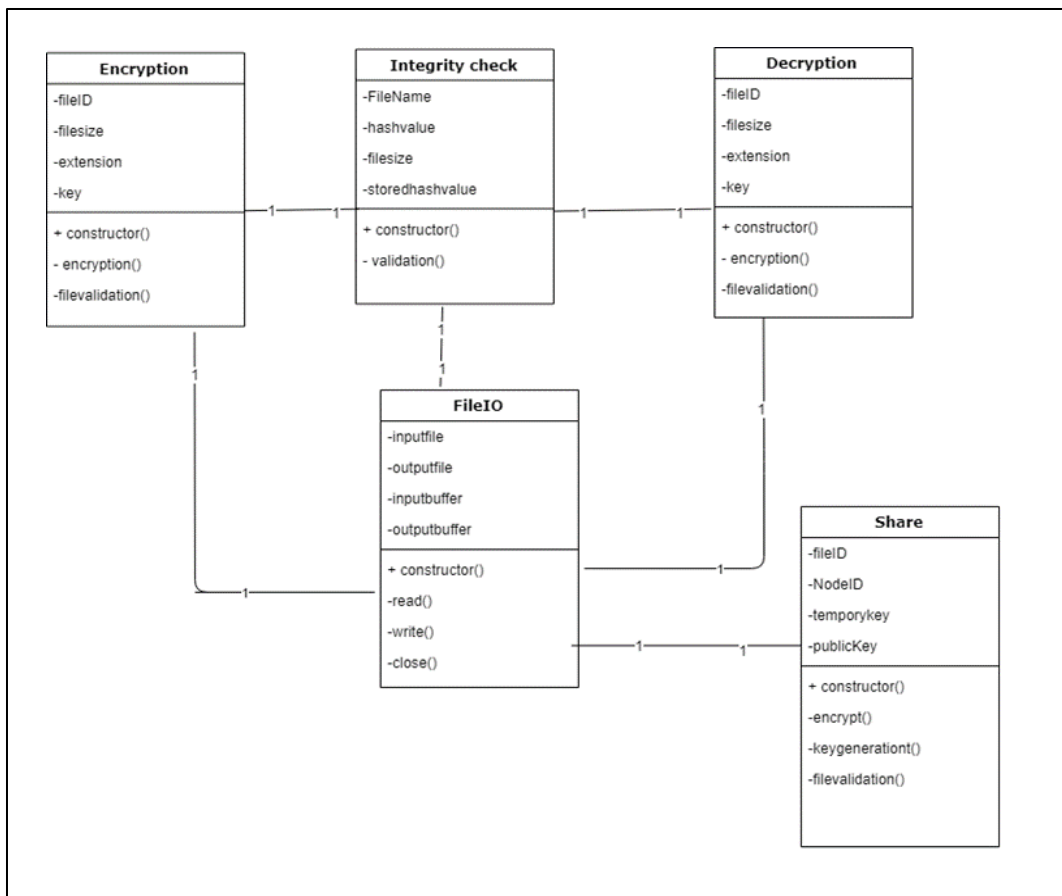


Figure 20 : File sharing module class diagram

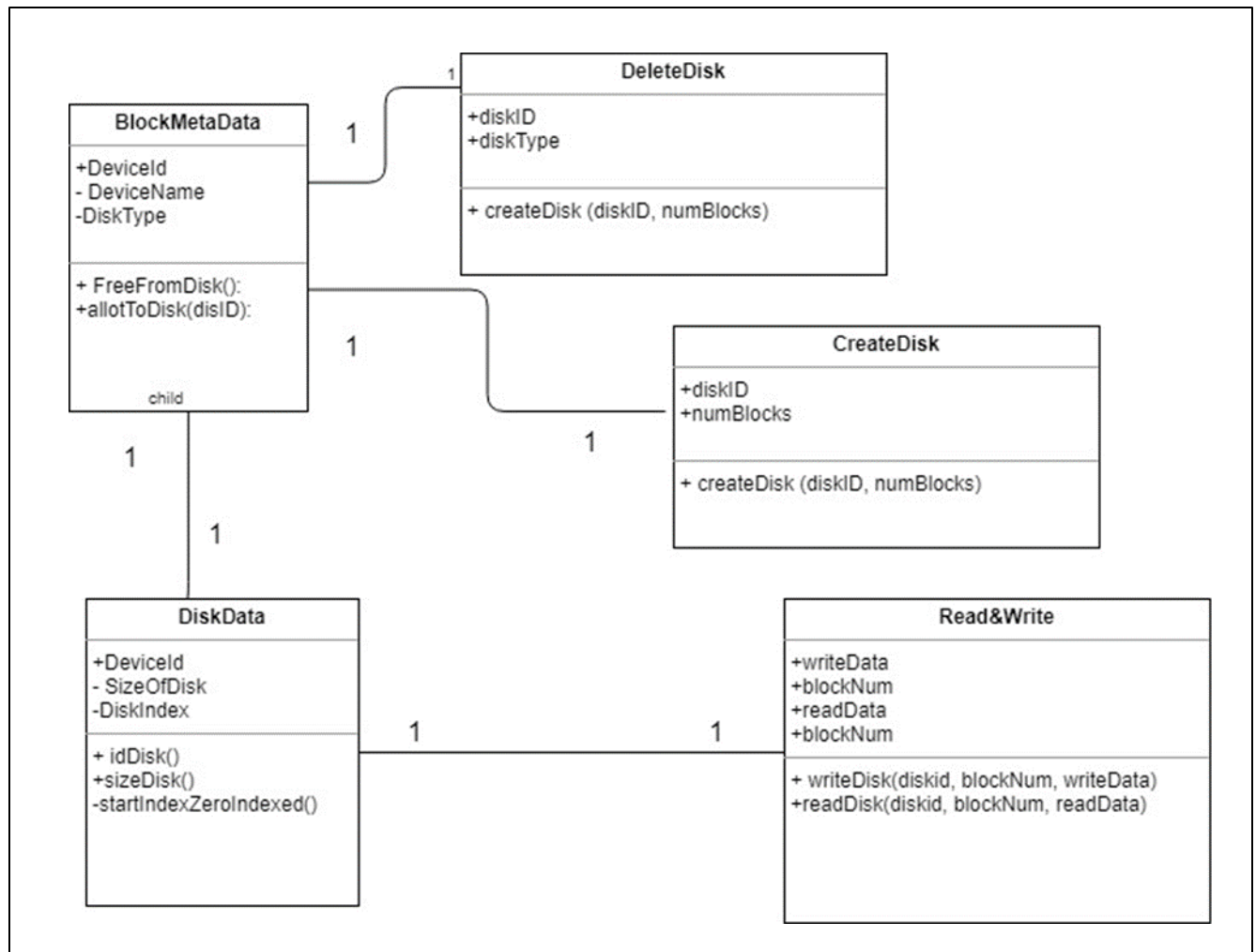


Figure 22 : Disk availability module class diagram

3.3 Performance requirements

Storage and network requirements can be identified for the blockchain and Messenger API facilitation. The host machine should have the necessary capacity to host storage for the Rethink DB server. The network should be reliable. Network speed/bandwidth is not a main concern since the amount of data throughput required by the module is very low.

3.4 Design constraints

There are no specific design constraints involves with the component which is mentioned above in this document, as long as the interfaces are easy to understand and use.

3.5 Software system attributes

In this section the features which will be offered to the customers will be described.

3.5.1 Reliability

Ability of a system to maintain its ordinary operation within given time in a given environment with minimum failures is called reliability.

The main objective of Vault is to ensure reliability of stored data at any given time. The decentralized architecture of Vault is what facilitates reliability when deployed as a swarm. A single node in the swarm can rebuild and re-index their base blockchain where file data is held when and if a failure occurs. The ability to regenerate files when only partial fragments exists contributes equally to the reliability of the software.

3.5.2 Availability

Probability a system is functioning when its services are required by the users of the system is called availability of a system.

The advantage of using Vault is that it is highly available. This is again due to the fact that the design architecture is distributed. For instance, when a user requests a file from their own interface, multiple requests are made throughout the swarm where fragments of the file exists. Once these files are downloaded asynchronously in parallel the rebuilding happens. If one or more host is down/unresponsive, the process can still work uninterrupted because of the Redundancy functionality built into it.

3.5.3 Security

Security of a system is the function which allows system to provide its services to its legitimate users, while resisting the other unauthored users from gaining access to the system or its data and resisting authorized users from performing unauthorized actions.

Vault uses AES-256 when encrypting files that are placed on storage. Public key cryptographic functions are used when nodes are authenticated and authorized to access the swarm. The usage of a blockchain as the primary object storage model brings in features like disintermediation where trust on a single authority is redundant. These features alone make Vault a highly secure application.

3.5.4 Maintainability

Maintainability is the ability to change the systems functionalities and increase the performance by applying system repairs and updates while maintaining systems availability, security and reliability. The modules are designed in the Object-Oriented paradigm where modularity is of importance. This constitutes into a highly maintainable design. The use of JavaScript JSON objects when inter-modular communication occurs ensures a consistent data transfer model between modules that can be tweaked as and when required.

4 Supporting information

4.1 Appendices

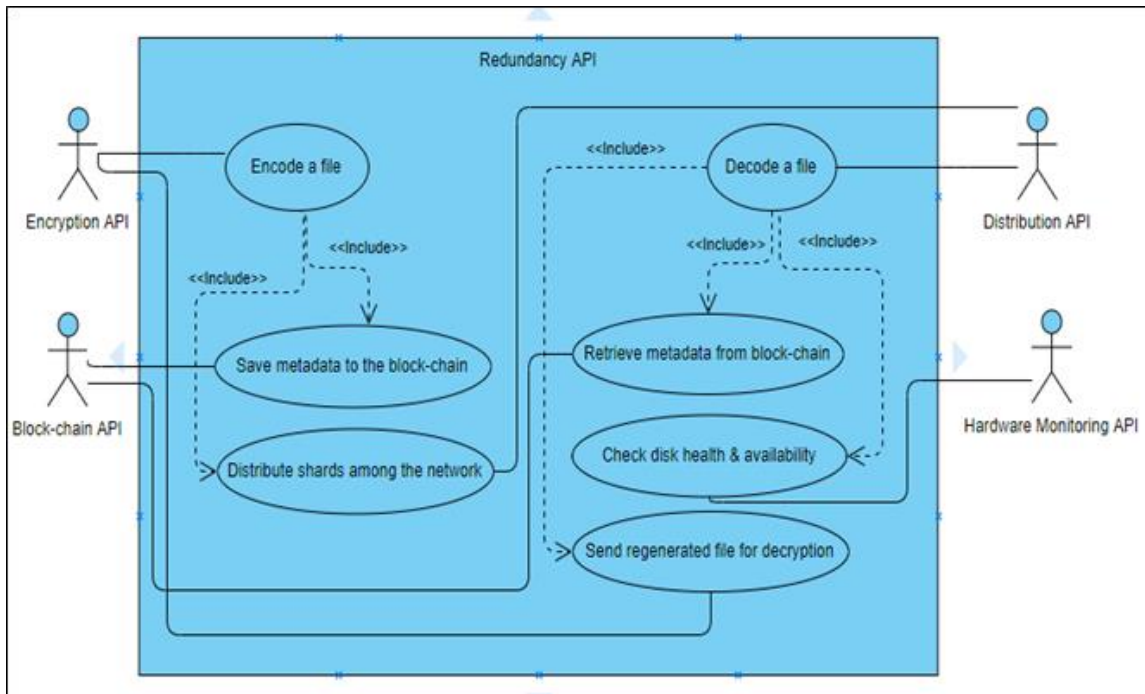


Figure 24: Redundancy API use case

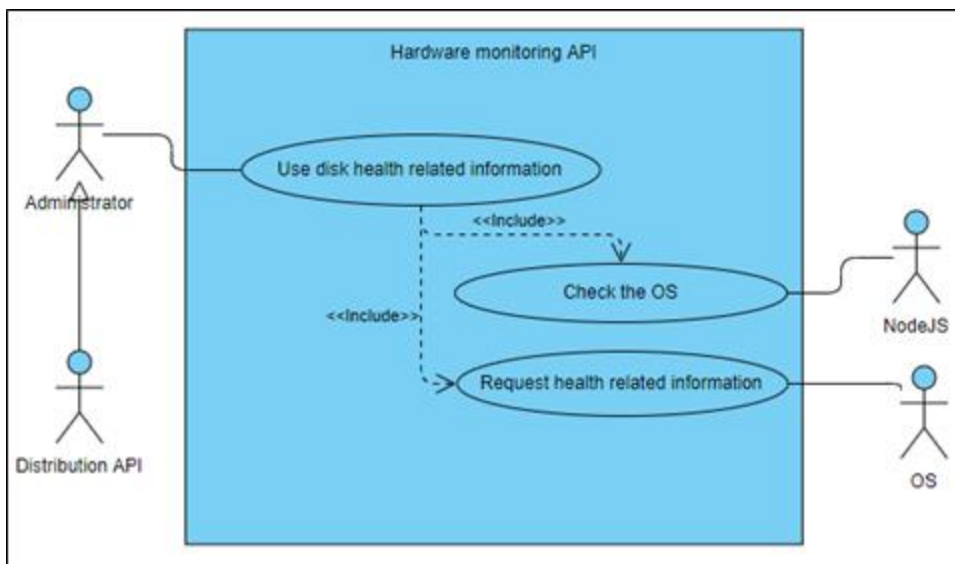


Figure 23: Disk health API use case

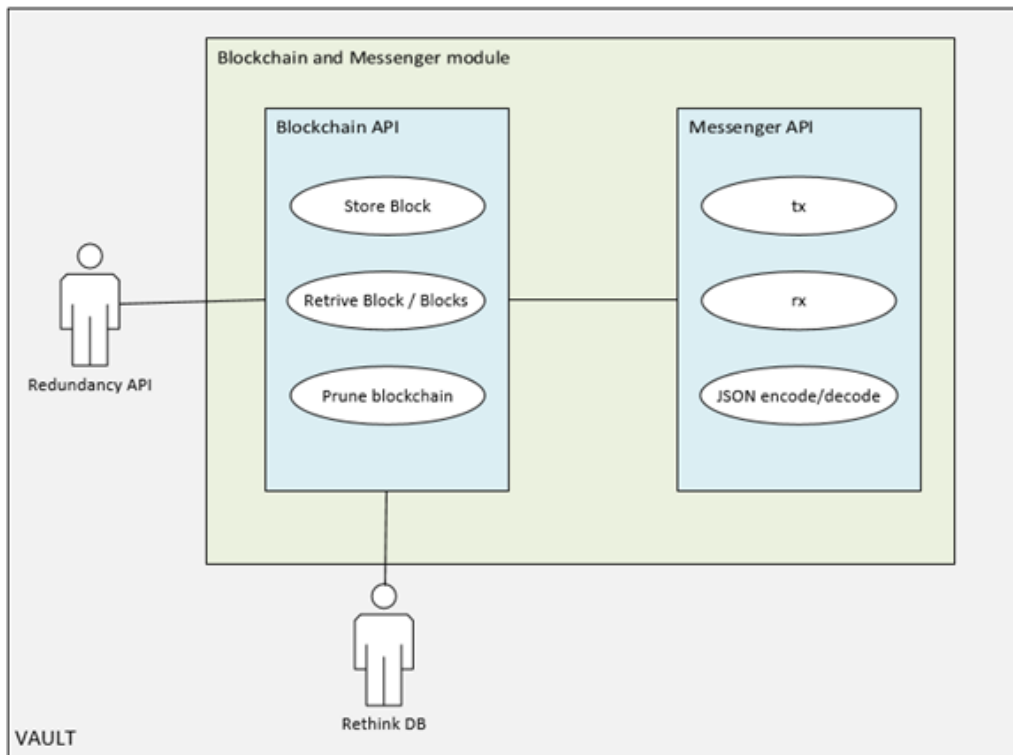


Figure 26: Blockchain and messenger module use case

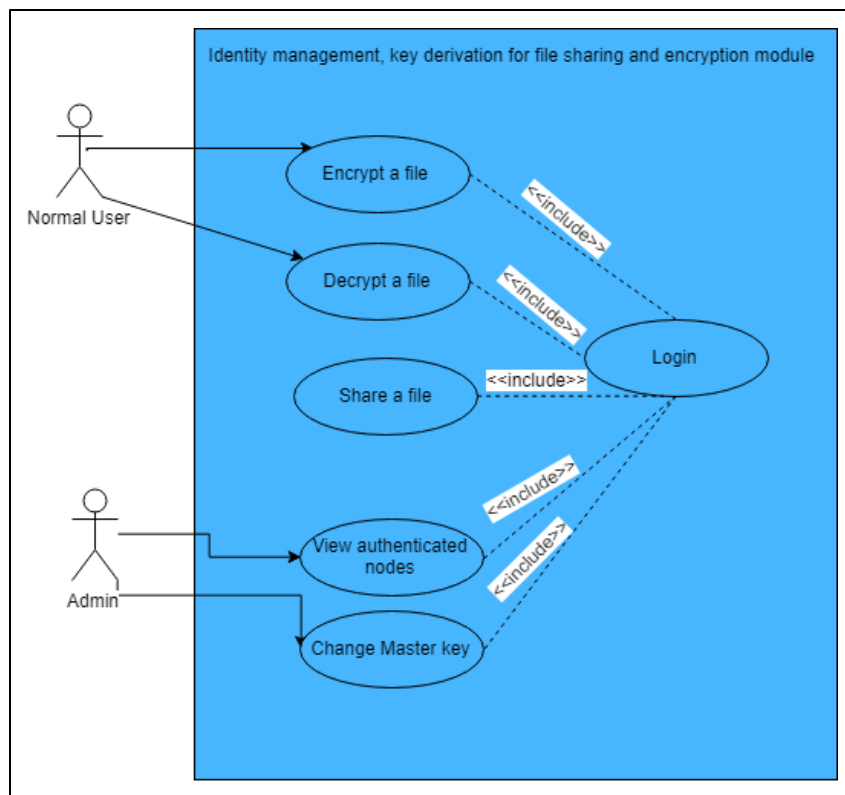


Figure 25: Identity management use case

5 Reference

- [1] "moosefs.com," moosefs, [Online]. Available: <https://moosefs.com/>.
- [2] IBM, "www.ibm.com," IBM, 03 2019. [Online]. Available: <https://www.ibm.com/us-en/marketplace/scale-out-file-and-object-storage>. [Accessed 03 2019].
- [3] Oracle, "oracle.com," Oracle, 03 2019. [Online]. Available: <https://oss.oracle.com/projects/ocfs2/>. [Accessed 03 2019].
- [4] The OrangeFS Project, "orangefs," The OrangeFS Project, 03 2019. [Online]. Available: <http://www.orangefs.org/>. [Accessed 03 2019].
- [5] BlueWhale, "bwstor.com," BlueWhale, 03 2019. [Online]. Available: www.bwstor.com.cn/templates/T_product_EN/index.aspx?nodeid=150&page=ContentPage&contentid=402. [Accessed 02 03 2019].
- [6] Minio, "minio," Minio, 02 2019. [Online]. Available: <https://www.minio.io/>. [Accessed 03 2019].
- [7] Ceph, "ceph.com," Ceph, 03 2019. [Online]. Available: <https://ceph.com/ceph-storage/file-system/>. [Accessed 01 03 2019].
- [8] socket.io, "Socket.io Documentation," [Online]. Available: <https://socket.io/docs/>. [Accessed May 2019].
- [9] <https://rethinkdb.com/docs>, "Rethink DB - Documentation," [Online]. Available: <https://rethinkdb.com/docs>. [Accessed May 2019].