SLIIT

COMPUTING | BUSINESS | ENGINEERING

VAULT

A shared distributed and redundant storage solution

Project ID: 19-002

IT 16158528 – K.V.A Sachintha

Structure of the SRS
(Software Requirements Specification)

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

DECLARATION

I, K.V.A Sachintha declare that this is my 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.

…….

K.V. A Sachintha

Table of Contents

Table of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of this SRS document is to describe the requirements and the process related to the Blockchain API module and the Messenger API module. The document will explain the purpose, features, functional and non-functional requirements, design constraint, project approach, constraints under which above mentioned modules must operate and how the modules will interact with the other modules and the external applications. 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

The components which will be discussed in this document are called “Blockchain and Messenger APIs”. The components mainly contain two separate processes, read/write to a blockchain and relay metadata via websockets (Messenger). The main goal of the Blockchain API is to store and retrieve data to and from the low level blockchain where metadata of files and fragments are stored. The Blockchain API works together with the Messenger API to provide storage and communication. The Messenger API’s main goal is to relay file metadata and other control information to all other nodes (hosts) in the peer-to-peer swarm. The Messenger API works by establishing http connections and reliably upgrading the connection to a websocket thus ensuring continuous connectivity and minimizing system resources.

1.3 Definitions, Acronyms, and Abbreviations

Table 1: Definitions and Abbreviations

OS	Operating System
API	Application Programming Interface

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 Description

Blockchain base metadata storage and Messenger API

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.

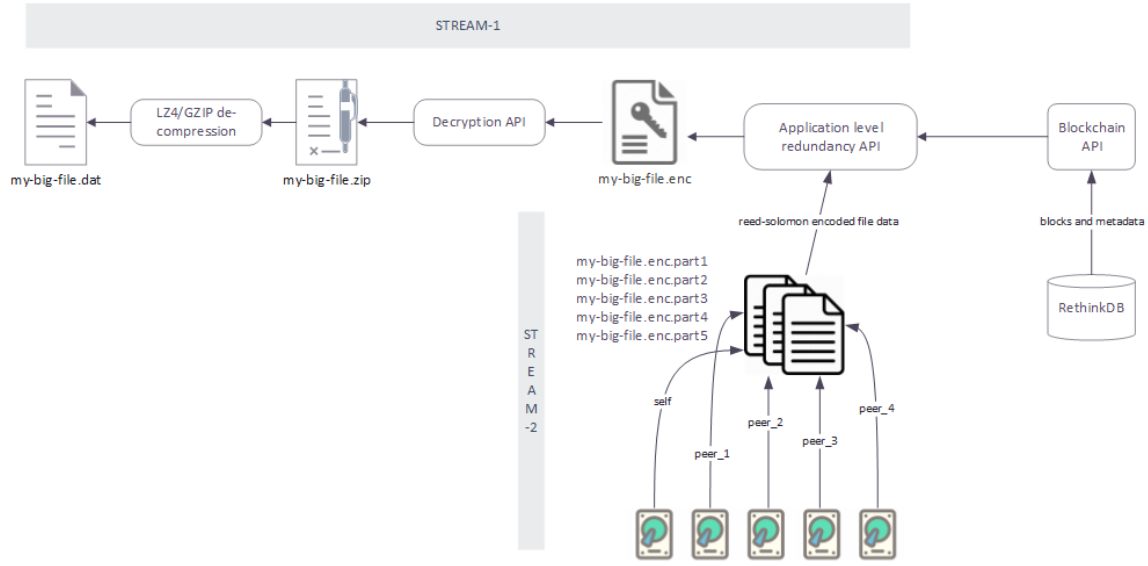


Figure 1: Process Overview

2.1 Product perspective

The table below describes the similarities and the differences of the final product, not only the components mentioned in this document, with the existing products in the market.

Table 2: Feature Comparison

Features	MooseFS	IBM Spectrum Scale	OCFS 2	OrangeFS	BWFS	Minio	Ceph	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 and the component mentioned in this document both acts as standalone solutions final product, and the component mentioned in this document 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. The blockchain will be stored in a NoSQL database, namely RethinkDB and therefore will act as a dependency to the program.

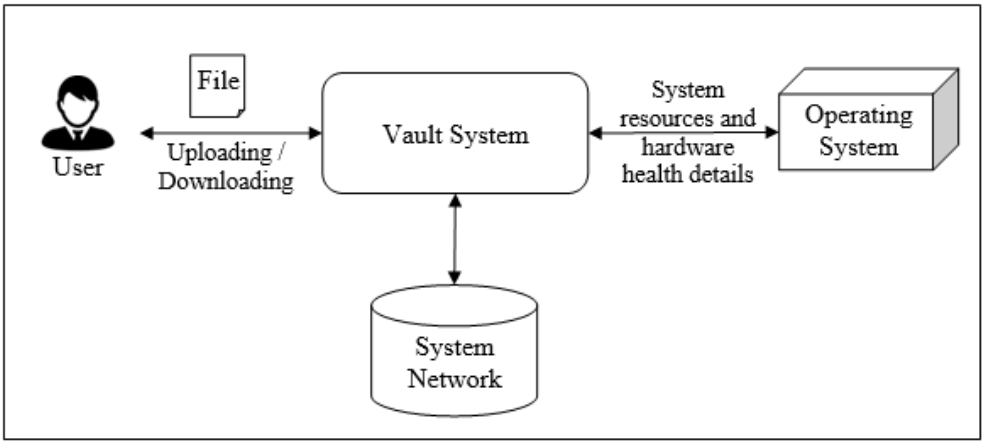


Figure 2:System Overview

2.1.2 User interfaces

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.

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

The Blockchain API and the Messenger API will only bind to the base operating system via the NodeJS middlewares. Both APIs mentioned above will not employ a RPC based software interface because both modules work in the same application thread and the runtime environment is shared.

2.1.5 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*** socketservers.

2.1.6 Memory constraints

Since the application is engineered to use disk space more than Memory, a machine with at least 2GB 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 web browsers. Administrators can connect/ disconnect workstations form 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.

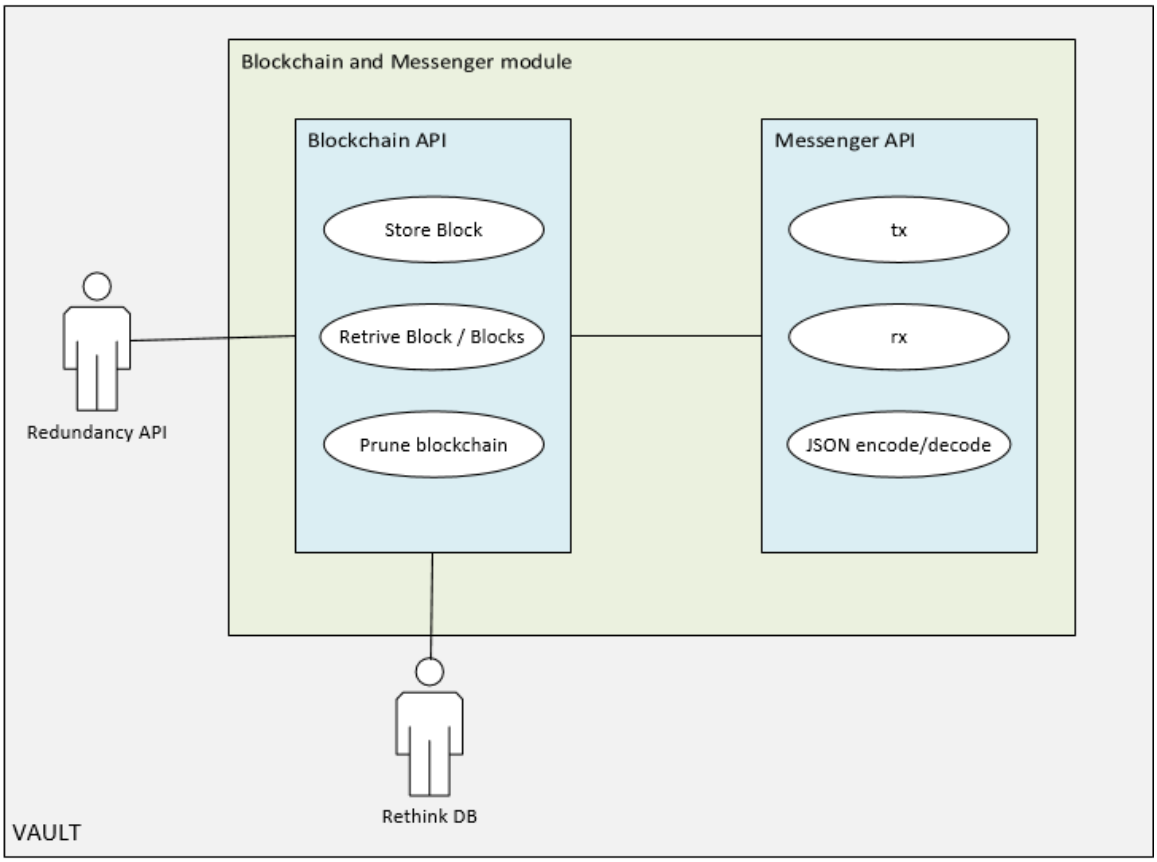
When a normal user accesses the web application from the web browser a login page will be displayed to the user, and users can use the logging credentials provided by the administrators to log in to the application. When a normal user can log in they can upload, download files to the system which will 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. CS^2

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



Store Block

Table 3: Store Block

Use Case No	01
Use Case	Store Block in Blockchain
Actors	Redundancy API, Rethink-DB
Pre-Conditions	Block Data validated, and transactions verified
Flow of Event	1. File and fragment data are received from the Redundancy API 2. Create a JavaScript Object containing Block data. 3. Create Transaction objects from Block data. 4. Calculate hashes of the Transaction Objects. 5. Get the Merkle Root of the transactions. 6. Store the Merkle Root in Block 7. Get previous Block Hash and set in in the Current block field. 8. Calculate Block hash and store it in Block. 9. Save Block in the RethinkDB database.
Post Conditions	Messenger API awaits block data.

Alternatives	Trigger a callback and update UI when an error occurs that isn't handled in the application.

Retrieve Block from chain

Table 4: Retrieve Block from chain

Use Case No	02
Use Case	Retrieve Block from chain
Actors	Redundancy API, Rethink-DB
Pre-Conditions	User must be logged in
Flow of Event	1. Redundancy API requests Block data. 2. The Blockchain API issues a request to the Rethink DB using the block id. 3. Blockchain API awaits database response 4. Data arrives as JSON object. 5. Create Block object from JSON object 6. Pass the newly created Block object to Redundancy API
Post Conditions	Redundancy API awaits Block
Alternatives	Display an error message to inform user that the block was not found.

Prune Blockchain

Table 5: Prune Blockchain

Use Case No	03
Use Case	Prune Blockchain
Actors	Rethink DB
Pre-Conditions	
Flow of Event	1. The Blockchain API starts going through the blockchain from the final block following the hash chain verifying each block. 2. If a block fails validation, the entire blockchain will be replaced
Post Conditions	Entire blockchain will be replaced
Alternatives	Alert the user when the blockchain is replaced and make the entire UI unresponsive in the duration it takes to replace the blockchain

Tx (Transmit Message)

Table 6: TX (transmit Message)

Use Case No	04
Use Case	TX (Transmit Message)
Actors	Blockchain API
Pre-Conditions	A properly formatted (JSON) Block object must exist.
Flow of Event	1. After the Blockchain API stores a block, the same details are relayed to the Messenger API. 2. The Messenger API validates the data and creates a websocket response. 3. The response is broadcasted on a separate websocket channel.
Post Conditions	The Messenger API awaits acknowledgement from the swarm's nodes.
Alternatives	If the input JSON object is invalid, the block data is requested from the Blockchain API to be fetched from the base blockchain itself.

Rx (Receive Message)

Table 7: RX (Receive Message)

Use Case No	05
Use Case	RX (Receive Message)
Actors	Remote Node
Pre-Conditions	
Flow of Event	1. Listen for messages actively 2. Parse Message and validate Authenticity of the Message. 3. Using parsed Message data, identify message type. 4. Relay the message to a specific function within the Messenger API to be processed.
Post Conditions	Message is dispatched to the relevant function for further processing.
Alternatives	The Message is dropped if the authenticity checks failed.

JSON encode/decode

Table 8: JSON encode/decode

Use Case No	06
Use Case	Encode/Decode JSON
Actors	Redundancy API
Pre-Conditions	
Flow of Event	1. Listen for messages actively 2. Parse Message and validate Authenticity of the Message. 3. Using parsed Message data, identify message type. 4. Relay the message to a specific function within the Messenger API to be processed.
Post Conditions	Message is dispatched to the relevant function for further processing.
Alternatives	The Message is dropped if the authenticity checks failed.

2.3 Constraints

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.

2.4 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.

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

3.1 External interface requirements

3.1.1 User interfaces

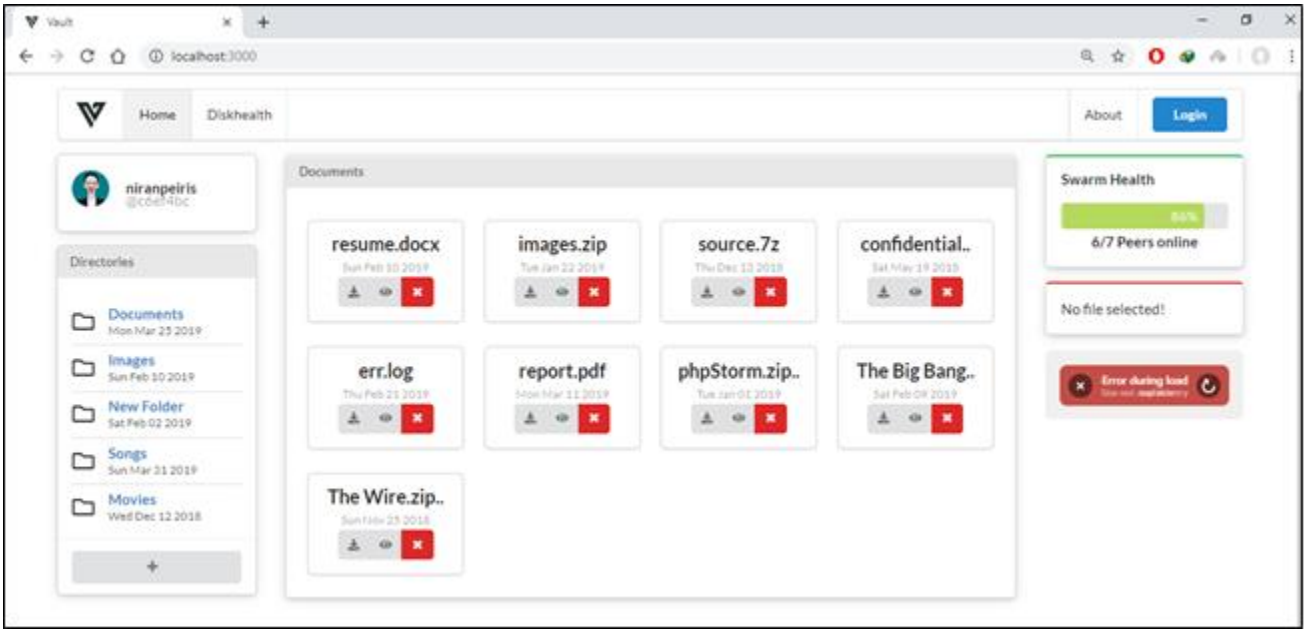


Figure 3: File Uploading/Downloading interface

Table 9: File uploading/downloading interface description

Name of item	File uploading/downloading interface (Redundancy handling)
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

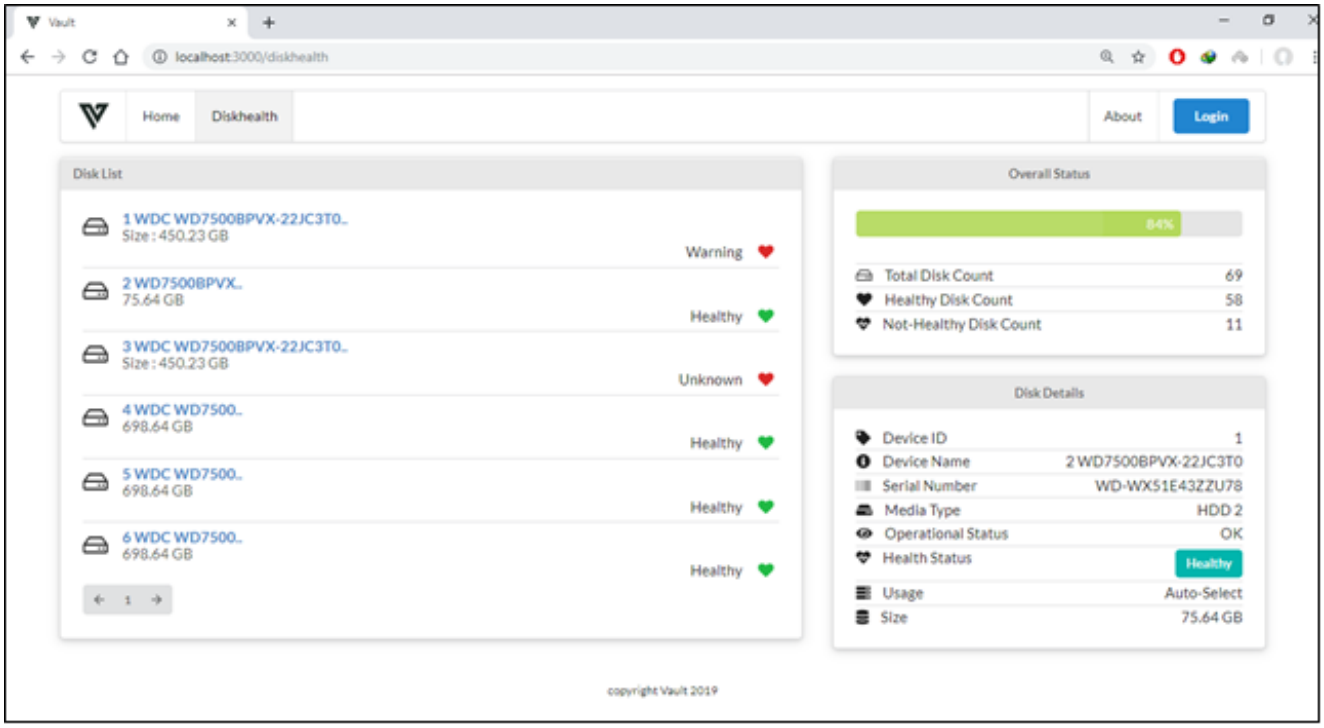


Figure 4: Hardware monitoring interface

Table 10: Hardware status 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

3.1.2 Software interfaces

The Blockchain API and the Messenger API will only bind to the base operating system via the NodeJS middlewares. 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.

3.1.3 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 socketservers.

Table 11: Blockchain API module communication interfaces

Interface	External Module	Connection details
Interface 1	Rethink DB	This interface is used to store and retrieve data from the base blockchain through the "thinky" ORM library. The communication type is set to JSON for consistency.
Interface 2	Redundancy API	This interface is used to establish a communication pathway between the Redundancy API and the Blockchain API. File and Fragment metadata is sent to the Blockchain API for storage through this interface. Data retrieved from the base chain is sent to the Redundancy API through this interface as well.

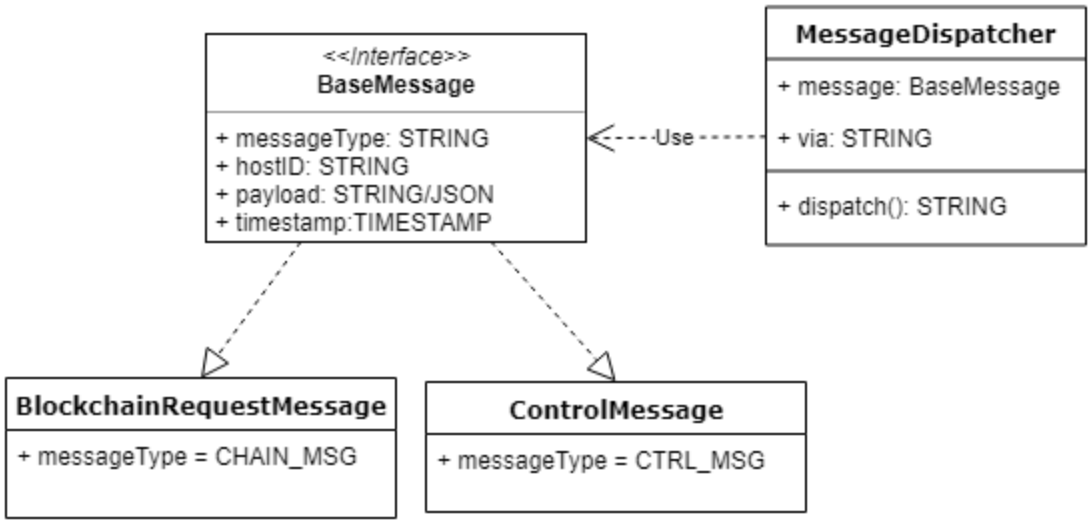
Table 12: Messenger API module communication interfaces

Interface	External Module/ Feature	Connection details
Interface 1	Blockchain API	Gather disk health and availability information from all over the network through distribution function.
Interface 2	socketserver >socketclient	Broadcast and targeted requests are sent to remote nodes in the swarm through this connection interface. The node that acts as the socketserver sends the message/request while the recipient node (socketclient) awaits.

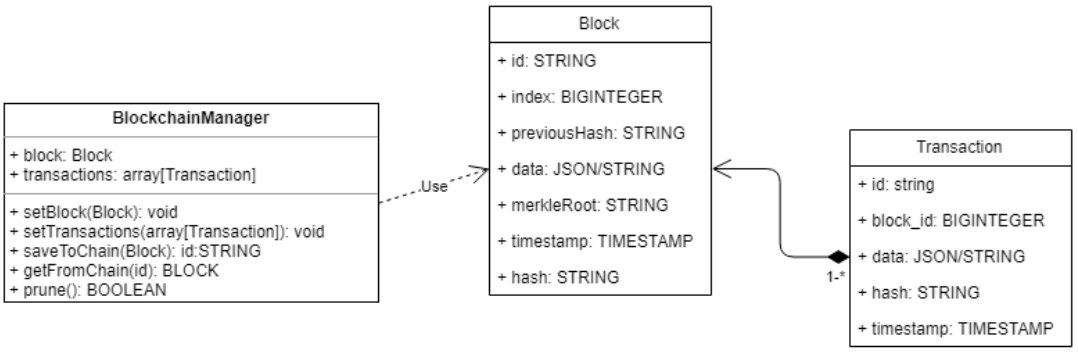
Interface 3	socketclient > socketserver	Targeted responses are sent by multiple socketclients to the node that acts as the socketserver.

3.2 Classes/Objects

3.2.1 Messenger API Class Diagram



3.2.2 Blockchain API 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 reliability of the Blockchain is inherent in its design. The addition of the Merkle root in each and every transaction makes the design modification resistant. The hashes computed from each block links the blocks to form a hash chain. This fact added to the Merkle Root of transactions in each Block constitutes to an irreversible, decentralized data storage model that is highly reliable in many contexts.

3.5.2 Availability

The availability of the communication model in the Messenger API is derived from the properties of websockets. The protocol used when communicating with hosts is a real-time, bi-directional protocol that has the ability to switch the protocol stack from simple http to Websockets with pooling. This inherently increases the availability of the application if and when network issues arise.

3.5.3 Security

The communication protocol used in the Messenger API implements HTTPS. This secures the message in transit as well as ensuring origin authentication. The Blockchain is stored in password-protected Rethink-DB shards in the user’s local storage.

3.5.4 Maintainability

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 References

[1] socket.io, "Socket.io Documentation," [Online]. Available: https://socket.io/docs/. [Accessed May 2019].

[2] https://rethinkdb.com/docs, "Rethink DB - Documentation," [Online]. Available: https://rethinkdb.com/docs/. [Accessed May 2019].