Software Requirements Specification

APXIEA: Blockchain-Based Storage System

Version <1.0>

Prepared by

Group Name:

|  |  |  |
| --- | --- | --- |
| Nasibullah Qarizada  Walid Abdul Hakim  Sulaiman Aqaie  Abdurrahman Ekin  Mehmet Tuğrul Arasan  Hasibullah Samadi  Oussame El Kharchy | 1900004691  1900000480  1900004901  1800002970  1800004088  1800005761  1900000577 | [1900004691@stu.iku.edu.tr](mailto:1900004691@stu.iku.edu.tr)  [1900000480@stu.iku.edu.tr](mailto:1900000480@stu.iku.edu.tr)  [1900004901@stu.iku.edu.tr](mailto:1900004901@stu.iku.edu.tr)  [1800002790@stu.iku.edu.tr](mailto:1800002790@stu.iku.edu.tr)  [1800004088@stu.iku.edu.tr](mailto:1800004088@stu.iku.edu.tr)  [1800005761@stu.iku.edu.tr](mailto:1800005761@stu.iku.edu.tr)  [1900000577@stu.iku.edu.tr](mailto:1900000577@stu.iku.edu.tr) |

|  |  |
| --- | --- |
| Instructor: | Akhan Akbulut |
| Course: | Software Engineering |
| Lab Section: | *CSE6064-2* |
| Teaching Assistant: | *Büşra Kocaçınar* |
| Date: | 08/05/2022 |

Contents

Contents ii

Revisions iii

1 Introduction 1

1.1 Project Purpose and Scope, and Objectives 1

1.2 Roles and responsibilities 1

1.3 Technical Assumptions and Constraints 1

1.4 Naming Conventions 1

2 Requirements 2

2.1 Functional Requirements 2

2.2 Non Functional Requirements 2

3 Other Requirements 3

4 System Architecture and Architectural Design 4

4.1 Logical View 4

4.2 Deployment View 4

4.3 Use Case View 4

5 Design and Implementation 5

6 Other Supporting Information 6

7 References 7

Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| 1.0 | Nasibullah Qarizada | Introduction, purpose and scoop of the Project, and the role of the group members on developing it, other supporting information and technical assumptions has been shared. | 08/05/22 |
| 1.0 | Sulaiman Aqaie | Design and Implementation and Deployment View has explained. | 15/05/22 |
| 1.0 | Walid Abdul Hakim | Overall editing Documents and Logical view. | 20/05/22 |
| 1.0 | Abdurrahman Ekin | Functional Requirement and Use-case Diagram has been explained. | 10/05/22 |
| 1.0 | Oussama El Kharchy | Non-functional Requirements and introduction to system architecture. | 17/05/22 |

# 

# Introduction

## Project Purpose and Scope, and Objectives

### Introduction

**APXEIA: Decentralized Storage App** is a blockchain-based decentralized storage which helps users to store their files and share public. Decentralized storage system is the concept of storing files online by splitting them into encrypted objects and delegating them on multiple nodes on distributed networks. Thus, it is among most preferred ways on storing files due to its’ security.

This Software Requirements Specification (SRS) document provides an overview of the entire Project (APXIEA) with purpose, scope, definitions, requirements, user interface designs, used components, references. The aim of this document is to gather and analyze and give an in-depth insight of the complete **APXEIA: Decentralized Storage App** by defining the properties statements in detail. Nevertheless, it also concentrates on the capabilities required by stakeholders and their needs while defining high-level product features.

### Purpose and Objectives

With the rising number of internet users daily, so too the risk level of keeping data safe is getting higher. From solving our challenges of safely storing data to not to having all our personal information on 3rd parties. Therefore, Blockchain-based Decentralized Storage apps like APXIEA is most preferable way to keep data safe. In decentralized storage apps, you can request it then receive the file. Requesting your files works similarly to BitTorrent and other P2P clients where you download fragments of that file from participants in the network until you have the full file. Which will gathered back and decrypted only with the encryption key or hash shared with you. Furthermore, you can share these files in both ways: public or private. Compared to centralized storage, decentralized storage has also advantages like low storage costs, Faster speed, Great Security, and minimal file loss.

### Scope

Decentralized data storage marketplaces enable consumers to improve autonomous control over and access to their data. The demand for decentralized cloud storage is rapidly growing as a generation of new platforms increase the flexibility and security of cloud storage, while lowering costs for end users.

Primarily, APXIEA is targeting to take storage away from centralized entities to reduce the risk level of data leaking/changing to more stable network and secure way. It also focuses on the company, the stakeholders, and applications to create a good background and good services which allow for storing data, sharing data public, and marketing.

## Roles and responsibilities

|  |  |  |
| --- | --- | --- |
| Name: | Role: | Explanation: |
| Nasibullah Qarizada | Analyst | Briefly scoops the project and runs for finding errors and improvements. (Smart Contracts, Backend, Front-end) |
| Walid Abdul Hakim | Manager | Developed and implemented overall aspects. (Blockchain and Back-end, Front-end) |
| Suleiman Aqaiee | UI/UX Designer | Creating UI/UX models and developing them. (Infographics and UI sections) |
| Abdurrahman Ekin | Back-End Developer | Developing structure in database and file uploading cases. (IPFS implementation) |
| Oussama El Kharchy | UI/UX Designer | Creating UI/UX models and developing them. (Infographics and UI sections) |

## Technical Assumptions and Constraints

### User Classes and Characteristics

The system users are divided into two categories: Customers, and Customer Support Team. Customer Support Team should be trained and have some knowledge about using the application. On the other hand, customers do not need a training or a background knowledge. Customer Supports Team consist of specialists which have good analytical and problem-solving skills, up-to-date technical knowledge, good interpersonal and customer care skills.

### Operating Environments

The designed system is thought to be a website and will be available via any web-browser application. It will not be dependent on the technical capabilities or operating system of user’s device

### Design and Implementation Constraints

Gas prices and fees should be displayed according to the size of file. Additionally, information about any changes that are made in the system should be displayed with no delay.

### Assumptions and Dependencies

It is assumed that the user has an internet access and web3.0 wallet can do online payments. The performance of APXIEA depends on the quality and speed of the internet connection and file size.

## 1.4 Naming Conventions

In this project CamelCase format is going to be used.

# Requirements

## Functional Requirements

1. **Authentication**: Permits a User to use the web and access its functions, which will be possible only if the user has a Ethereum wallet.
2. **Blockchain**: A blockchain is required so the web application can be connected to the specific blockchain, which in this case is the Ethereum Blockchain.
3. **Special Database**: We need a special data base that holds the data in a decentralized manner.
4. **Smart Contracts**: Smart contracts are digital contracts stored on a blockchain that are automatically executed when predetermined terms and conditions are met. Which we will use to store the file’s encrypted location in the blockchain.

## Non-Functional Requirements

### Performance Requirements

1. **High performance**: Fast Response time which should be less than 2 seconds and should be able to handle at least 50 transaction per second.
2. **Reachability**: Platform shall be available and reachable 24/7.
3. **Fast upload:** The upload time should be minimized and should be between 1sec to max of 5 min for larger files.
4. **Fast Connection:** The connection between the web3.0 and the system should be less than 2 sec after confirmation.
5. **Bug Free:** The System should have near to no bugs so that it makes it easy to use.

### Safety and Security Requirements

1. **Authentication and password management:**

* The logins are done by the users’ web 3.0 wallets which secures their private information.
* The system will verify user each time by sending a verification to the wallet.
* There is not any information stored anywhere about the users.

### Data Transfer

* The system shall use secure sockets in all transactions that include any confidential customer information.
* The system shall confirm all payment and data-reaching process with the customer’s web 3.0 wallet.

### Data Storage

* The data will be stored in decentralized network type database.
* System is 100% privacy focused. Therefore, we don’t store any kind of information about user.

### Software Quality Attributes

1. **Reliability**: It should be an application that is reliable and has near to no bugs and problems.
2. **User friendly UI**: The UI should be built so that it is easy to understand and is appealing to the eyes with color scheme that doesn’t include very bright colors which strains the eye.

## Reachability

* The product shall be based on web and must be run from a web server.
* The product shall take initial load time depending on internet connection strength which also depends on the media from which the product is run.
* The performance shall depend upon hardware components of the client.
* The maintenance and fixing bugs of the program shall be solved at the ending of each 6 month.

# System Architecture and Architectural Design

The project will be Decentralized, it is built using Sol and react languages. Sol is used to create Smart Contracts, connect web3.0 wallets and to connect the blockchain to the web application. Smart Contracts are compiled and migrated using Truffle, Ganache is used as a testing blockchain, Meta Mask is the main web3.0 agent, IPFS the network database and react the User Interface. It also consists of Kommunicate and EmailJS to help us connect to the User to find any bugs or suggestions in improving our project.

## Logical View

## Timeline Description automatically generated

1. (0-0.2) Connection of Contracts to Blockchain.

0 - 0.2 shows how the system compiles the Smart contracts to the Blockchain.

1. (1-1.4) Connection of Web 3.0 to Blockchain.

1 - 1.4 shows how the System connects the blockchain and the User using a web3.0 wallet (Ethereum based).

1. (2-2.2) Connection of React.

2 – 2.2 shows how React is connected between the User and blockchain.

1. (3-7) Upload files.

3-4 is how react sends a file to IPFS for storage and gets a hash in return

5-7 is how the File location is then stored to the blockchain using the wallet for verification from the user which without the confirmation of the user the file isn’t uploaded.

1. (8) View and Download File.

Lastly this the connection on how the user can view and download their files.

## Deployment View

The User can Access the web app using a computer with an internet connection while having a web3.0 wallet as it is the main Authenticator as without it, he/she cant access the web app.

The user Interface is created using React which is connected to the Blockchain using the Web3.0 wallet and is also connected to the IPFS servers Using the API’s provided by the company itself.

When the User wants to upload a file, React sends the file to IPFS and gets a hash and then the hash is sent to the blockchain using smart contracts and a confirmation is requested from the user through the Authenticator Web3.0 wallet.

When the file is uploaded the location is then stored in the blockchain where the user can access it using their wallet and React to View and Download it.

## Use Case View

### Use Case Scenarios

* User opens meta mask account.
* User logs into the system.
* User request uploads any file to the system.
* IPFS generates a hash for the file received from the user and presents it to the user.
* The user can see the uploaded file and download it again.
* The user can get support information from the chatbot about the system.
* User can provide feedback

# Design and Implementation

**Navbar:**



**Upload Page:**

Graphical user interface, application

Description automatically generated

**Contact Us:**

Graphical user interface

Description automatically generated

Some of main codes:

**Contact us:**

Text

Description automatically generated

**Upload Page:**

Text

Description automatically generated

# Other Supporting Information

Implementing components is a preferred way which provides advantages to developers and software owners as saving time and money. In case to not to spend most of our time and spend energy on developing system as chatbot and feedback form, which might be causing to mislead the team from original goals. Therefore, we’ve decided to implement and define pre-designed models.

**Kommunicate.io:**

The way customer conversations and support are happening is broken, redundant, and time and cost inefficient. And Kommunicate aims to correct it. Kommunicate is a human + bot hybrid customer support software for proactive, real-time, and personalized support.

The amount of customer service queries is ever increasing. That is why AI becomes extremely important. With chatbots, businesses can serve their customers effectively without spending vast amounts of time, resources, and workforce.

Although, chatbots are not fail-proof and can only be as smart as you can make them. They are bound to have limitations. The best bet is a human + chatbot hybrid model, where chatbots can answer the basic queries and leave the complex ones for humans.

Kommunicate aims to empower your business to build long-lasting customer relationships that drive growth. With Kommunicate, you can manage customer conversations, create, and integrate chatbot, website chat, support agents, team conversations, customer delight aspects all at the same place.

**EmailJS:**

EmailJS is a service that allows us to send emails directly from your client-side JavaScript code. As you can see, if we use EmailJs we don’t need a server to do that.

EmailJS is a mail sending platform that directly sends an email without the requirement of code. This software supports both personal email and transactional email services. Moreover, users can create their own templates and choose a code editor or text editor according to their requirements. EmailJS is a very helpful tool to send email using only client-side technologies. Users have to connect EmailJS to one of their supported email services, then create an email template and use their JavaScript library to transact an email. This tool offers features like attachments that can be included as streams, file parts, or strings. You can send emails with multiple attachments using EmailJS with personal, business, professional pricing strategies.

# References

* Elad E. (2019), The Blockchain developer: Guide for Designing Blockchain-based Projects
* <https://tr.wikipedia.org/wiki/Blok_zinciri\>
* <https://www.asynclabs.co/blog/blockchain-development/how-is-blockchain-used-in-the-software-development-industry/>
* <https://www.youtube.com/channel/UCnxrdFPXJMeHru_b4Q_vTPQ>
* <https://ethereum.org/en/>
* <https://www.file.io/developers>
* <https://zeemly.com/p/ipfs/alternatives>
* <https://spideroak.com/one/>
* <https://alternativeto.net/software/ipfs/>