**Legend: Yellow highlight – Section is OK**

**Chapter Four**

# **METHODOLOGY**

Diagram

Description automatically generatedThis chapter provides an overview of the strategies used to attain the study's goals. It describes the study's respondents as well as the research instruments that were used. It then goes on to explain the data collection strategies that contributed in the completion of the study endeavor.

**Figure 4.0.1: Prototype Model Phases and Process**

Figure 4.0.1 illustrates Prototype Model used by the researcher in developing the proposed study entitled “A Blockchain Implementation for Secured Vaccine Certificates” which is under the family of System Development Life Cycle (SDLC). Prototyping was used to ensure faster turnaround time on each phase while addressing client’s requirements and feedbacks. This model also enables the researcher and client to have discussions in between development cycles.

The next sections of this chapter will discuss the phases of the used model.

## **4.1 Requirements Modeling**

The Prototype Model starts with outlining the requirements. The researcher will conduct an initial investigation to determine the purpose and utilization of the application coupled with the nature and scope of the study. It is also in this stage that the researcher requested permission from medical unit authorities and other parties to conduct the study and all relevant data and information were examined.

Fact-finding was used via interviews and probing of processes to build a logical model of the application. With these interviews, the researcher was able to piece out a picture of transactions involved and analyzed them against the proposed solution. This information will also enable the researcher to identify critical decisions geared toward implementing the application.

For vaccination records, citizens are encouraged to register online via the web portal. This will ensure a scheduled slot on a specified date. On the day of vaccination, patient will be checked up by a physician to ensure he is fit for vaccination. The physician’s findings are logged on the system. Upon issuing a go signal, patient can now be vaccinated. After vaccination, vaccination site will sign a vaccination card while tagging the patient in their system as fully vaccinated.

The study will be focused on Vaccine Certificates. Mocked test data will be used and will only be for the purpose of this research. This is due to various privacy regulation such as Health Insurance Portability and Accountability Act (HIPAA). This is a United States created health law adopted by medical facilities in the Philippines.

**Qr code

Description automatically generated**

Sample Generated Vaccine Certificate File

Below are requirements grouped by specific role:

Patient

* Register and Login – register to gain access to the system
  + Upon registration, system will create private and public keys to be used for data encryption
* Download Vaccine Certificate
* View QR Code for Vaccine Record Summary
* View Record Summary Details

Verifying Third Party

* Publicly Available
* Validate Vaccine Certificate if existing in system

Physician/Medical Unit

* Register and Login – register to gain access to the system
* Create vaccine record for patient

**4.1.1 Data Storage Scheme**

Since the study is primarily concerned on how medical records will be stored, this section will discuss the different schemes that will be used in the application. This will involve simulation and detailed discussions.

**4.1.1.1 Blockchain Components**

The main purpose of using blockchain is to log and validate user transactions. Transactions we want to monitor and be authenticated are summary details and vaccine certificate creation. Apart from file and summary hashes, the app will include supporting details such as userId and Content Identifier from IPFS.

Assuming we have obtained hashes and Content Identifiers, we will create a blockchain of transactions given the files were already uploaded to IPFS and CIDs are generated. JSON Objects will be used as format of the payload. Summary details of vaccine records will also be stored in the blockchain

Both file and summary hash lookup have the same structure:

<hash>: <userId>

Sample:

[QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX](https://cid.ipfs.io/#QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX): 1001

Below is the Solidity code to manage hash lookups for vaccine certificate files and summary details:

// SPDX-License-Identifier: GPL-3.0

**pragma** solidity >=0.5.0 <0.9.0;

**contract** Certificate {

**mapping** (**string** => **uint256**) fileHashUserId;

**mapping** (**string** => **uint256**) summaryHashUserId;

**function** isFileHashUserIdExists(**string** **memory** \_fileHash, **uint256** \_userId)

**public** **view** **returns**(**bool**)

{

**if** (fileHashUserId[\_fileHash] == \_userId) {

**return** **true**;

}

**return** **false**;

}

**function** saveUserIdHashes(**string** **memory** \_fileHash, **string** **memory** \_summaryHash, **uint256** \_userId)

**public**

{

summaryHashUserId[\_summaryHash] = \_userId;

fileHashUserId[\_fileHash] = \_userId;

}

**function** isSummaryHashUserIdExists(**string** **memory** \_summaryHash, **uint256** \_userId)

**public** **view** **returns**(**bool**)

{

**if** (summaryHashUserId[\_summaryHash] == \_userId) {

**return** **true**;

}

**return** **false**;

}

}

**4.1.1.1.1 Keccak-256**

The hashing algorithm used by Ethereum (implemented in Clique POA) is Keccak-256. Below is a simulation of the algorithm using a simple input string:

Keccak256 presets:

bitrate\_bits = 1088

capacity\_bits = 512

output\_bits = 256

bitrate\_bytes = 136 -- convert bitrate\_bits to bytes

multirate\_padding(used\_bytes, align\_bytes)

padlength = align\_bytes - used\_bytes

zero\_elements = [0] \* padlength - 2

padding = [1] + zero\_elements + [128]

return padding

#example

#if used\_bytes = 130, align\_bytes = 136

#padlength = 136 - 130 = 6

#zero\_elements = [0, 0, 0, 0]

#padding = [1, 0, 0, 0, 0 128]

bytesToLane(input\_bytes)

accumulator = 0

for b in reversed(input\_bytes)

accumulator = ( accumulator << 8 ) | b

#apply 8 bitwise left shit to accumulator then XOR with b

return accumulator

#example

#input\_bytes = [104, 101, 108, 108, 111, 32, 119, 111]

each iteration will result to (consecutively)

0

28416

7304960

1870077952

478739984128

122557435964416

31374703606918144

8031924123371070720

8031924123371070824

#final value will be 8031924123371070824

-----------------------------------------------------------

input\_text = "hello world"

1. Get byte array (input\_byte\_array) equivalent of input\_text

input\_byte\_array = [104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100]

2. Pad input\_byte\_array using multirate\_padding

used\_bytes = input\_byte\_array.length = 11 (count number of elements inside array)

align\_bytes = presets.bitrate\_bytes = 136

padded\_bytes = [104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128]

3. Append another batch of zero elements to padded\_bytes

zero\_count = convertToBytes ( (presets.bitrate\_bits + presets.capacity\_bits) - presets.bitrate\_bits )

= convertToBytes((1088 + 512) - 1088)

= 64

zero\_elements = [0] \* 64

padded\_bytes += zero\_elements

= [104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

4. Convert padded\_bytes to array of lanes (lane\_array) and put then in a 5x5 2D array.

- get a batch of 8 elements from padded\_bytes

- convertedBatch1ToLane = bytesToLane(batch)

- result will be:

[[8031924123371070824L, 0, 0, 0, 0], [23358578, 0, 0, 9223372036854775808L, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]

5. Process lane\_array to permutation\_rounds (greek alphabet methods - theta, rho and phi, chi, iota)

lanew

= (presets.bitrate\_bits + presets.capacity\_bits) // 25 #the floor division // rounds the result down to the nearest whole number

= 64

l

= int(log(lanew, 2))

= 6

number\_of\_rounds = 12 + 2 \* l

= 24

TODO – mention Greek alpha permutation

**4.1.1.1.2 Clique Proof-of-Authority**

Proposed application will use Clique Proof of Authority consensus. Below is a simulation of various test cases:

*// block represents a single block signed by a parcitular account, where*

*// the account may or may not have cast a Clique vote.*

**type** block **struct** {

signer **string** *// Account that signed this particular block*

voted **string** *// Optional value if the signer voted on adding/removing someone*

auth **bool** *// Whether the vote was to authorize (or deauthorize)*

checkpoint []**string** *// List of authorized signers if this is an epoch block*

}

*// Define the various voting scenarios to test*

tests **:=** []**struct** {

epoch **uint64** *// Number of blocks in an epoch (unset = 30000)*

signers []**string** *// Initial list of authorized signers in the genesis*

blocks []block *// Chain of signed blocks, potentially influencing auths*

results []**string** *// Final list of authorized signers after all blocks*

failure **error** *// Failure if some block is invalid according to the rules*

}{

{

*// Single signer, no votes cast*

signers**:** []**string**{"A"},

blocks**:** []block{

{signer**:** "A"}

},

results**:** []**string**{"A"},

}, {

*// Single signer, voting to add two others (only accept first, second needs 2 votes)*

signers**:** []**string**{"A"},

blocks**:** []block{

{signer**:** "A", voted**:** "B", auth**:** true},

{signer**:** "B"},

{signer**:** "A", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// Two signers, voting to add three others (only accept first two, third needs 3 votes already)*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B", voted**:** "C", auth**:** true},

{signer**:** "A", voted**:** "D", auth**:** true},

{signer**:** "B", voted**:** "D", auth**:** true},

{signer**:** "C"},

{signer**:** "A", voted**:** "E", auth**:** true},

{signer**:** "B", voted**:** "E", auth**:** true},

},

results**:** []**string**{"A", "B", "C", "D"},

}, {

*// Single signer, dropping itself (weird, but one less cornercase by explicitly allowing this)*

signers**:** []**string**{"A"},

blocks**:** []block{

{signer**:** "A", voted**:** "A", auth**:** false},

},

results**:** []**string**{},

}, {

*// Two signers, actually needing mutual consent to drop either of them (not fulfilled)*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "B", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Two signers, actually needing mutual consent to drop either of them (fulfilled)*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "B", auth**:** false},

{signer**:** "B", voted**:** "B", auth**:** false},

},

results**:** []**string**{"A"},

}, {

*// Three signers, two of them deciding to drop the third*

signers**:** []**string**{"A", "B", "C"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Four signers, consensus of two not being enough to drop anyone*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

},

results**:** []**string**{"A", "B", "C", "D"},

}, {

*// Four signers, consensus of three already being enough to drop someone*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

},

results**:** []**string**{"A", "B", "C"},

}, {

*// Authorizations are counted once per signer per target*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B"},

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B"},

{signer**:** "A", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// Authorizing multiple accounts concurrently is permitted*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B"},

{signer**:** "A", voted**:** "D", auth**:** true},

{signer**:** "B"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** true},

{signer**:** "A"},

{signer**:** "B", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B", "C", "D"},

}, {

*// Deauthorizations are counted once per signer per target*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "B", auth**:** false},

{signer**:** "B"},

{signer**:** "A", voted**:** "B", auth**:** false},

{signer**:** "B"},

{signer**:** "A", voted**:** "B", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Deauthorizing multiple accounts concurrently is permitted*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

{signer**:** "A"},

{signer**:** "B", voted**:** "C", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Votes from deauthorized signers are discarded immediately (deauth votes)*

signers**:** []**string**{"A", "B", "C"},

blocks**:** []block{

{signer**:** "C", voted**:** "B", auth**:** false},

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "A", voted**:** "B", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Votes from deauthorized signers are discarded immediately (auth votes)*

signers**:** []**string**{"A", "B", "C"},

blocks**:** []block{

{signer**:** "C", voted**:** "D", auth**:** true},

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "A", voted**:** "D", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// Cascading changes are not allowed, only the account being voted on may change*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "C"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

},

results**:** []**string**{"A", "B", "C"},

}, {

*// Changes reaching consensus out of bounds (via a deauth) execute on touch*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "C"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

{signer**:** "A"},

{signer**:** "C", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// Changes reaching consensus out of bounds (via a deauth) may go out of consensus on first touch*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "C"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

{signer**:** "A"},

{signer**:** "B", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B", "C"},

}, {

*// Ensure that pending votes don't survive authorization status changes. This*

*// corner case can only appear if a signer is quickly added, removed and then*

*// readded (or the inverse), while one of the original voters dropped. If a*

*// past vote is left cached in the system somewhere, this will interfere with*

*// the final signer outcome.*

signers**:** []**string**{"A", "B", "C", "D", "E"},

blocks**:** []block{

{signer**:** "A", voted**:** "F", auth**:** true}, *// Authorize F, 3 votes needed*

{signer**:** "B", voted**:** "F", auth**:** true},

{signer**:** "C", voted**:** "F", auth**:** true},

{signer**:** "D", voted**:** "F", auth**:** false}, *// Deauthorize F, 4 votes needed (leave A's previous vote "unchanged")*

{signer**:** "E", voted**:** "F", auth**:** false},

{signer**:** "B", voted**:** "F", auth**:** false},

{signer**:** "C", voted**:** "F", auth**:** false},

{signer**:** "D", voted**:** "F", auth**:** true}, *// Almost authorize F, 2/3 votes needed*

{signer**:** "E", voted**:** "F", auth**:** true},

{signer**:** "B", voted**:** "A", auth**:** false}, *// Deauthorize A, 3 votes needed*

{signer**:** "C", voted**:** "A", auth**:** false},

{signer**:** "D", voted**:** "A", auth**:** false},

{signer**:** "B", voted**:** "F", auth**:** true}, *// Finish authorizing F, 3/3 votes needed*

},

results**:** []**string**{"B", "C", "D", "E", "F"},

}, {

*// Epoch transitions reset all votes to allow chain checkpointing*

epoch**:** 3,

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B"},

{signer**:** "A", checkpoint**:** []**string**{"A", "B"}},

{signer**:** "B", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// An unauthorized signer should not be able to sign blocks*

signers**:** []**string**{"A"},

blocks**:** []block{

{signer**:** "B"},

},

failure**:** errUnauthorizedSigner,

}, {

*// An authorized signer that signed recenty should not be able to sign again*

signers**:** []**string**{"A", "B"},

blocks []block{

{signer**:** "A"},

{signer**:** "A"},

},

failure**:** errRecentlySigned,

}, {

*// Recent signatures should not reset on checkpoint blocks imported in a batch*

epoch**:** 3,

signers**:** []**string**{"A", "B", "C"},

blocks**:** []block{

{signer**:** "A"},

{signer**:** "B"},

{signer**:** "A", checkpoint**:** []**string**{"A", "B", "C"}},

{signer**:** "A"},

},

failure**:** errRecentlySigned,

},,

}

**4.1.1.1 IPFS – Merkle DAG**

The algorithm used in IPFS to manage content and assets is Merkle DAG. Suppose we want to upload 2 vaccine certificates. For brevity, we will use a small size text file to better illustrate the process. The default chunk size of IPFS is 256Kb but in this example we will reduce it to 32Kb to have appropriate representation using small sample files.

File 1

Name: cert\_allen\_smith.txt

Size: 86 bytes

Content:

A picture containing text

Description automatically generated

File 2

Name: cert\_john\_doe.txt.txt

Size: 83 bytes

Content:

A picture containing text

Description automatically generated

Generated Details for cert\_allen\_smith.txt:

**Table 4.0.1: Generated Hash Value for Sample Record #1**

|  |  |  |
| --- | --- | --- |
| Node Type | Size (Bytes) | Hash |
| Root | 0 | [QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX](https://cid.ipfs.io/#QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX) |
| Links | 32 | [QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq](https://cid.ipfs.io/#QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq) |
| Links | 32 | [QmPsFk9hcP4WmN96r8mXYjV5rKCNNb94c95jfqLBNZvigT](https://cid.ipfs.io/#QmPsFk9hcP4WmN96r8mXYjV5rKCNNb94c95jfqLBNZvigT) |
| Links | 22 | [QmVVrfBPAnF5DC1DXDZH2yftW6MEoCSKXEQEbY5LKfFzAt](https://cid.ipfs.io/#QmVVrfBPAnF5DC1DXDZH2yftW6MEoCSKXEQEbY5LKfFzAt) |

Generated Details for cert\_john\_doe.txt:

**Table 4.0.2: Generated Hash Value for Sample Record #2**

|  |  |  |
| --- | --- | --- |
| Node Type | Size (Bytes) | Hash |
| Root | 0 | [QmanmTVLostTHeeLiz8vr99QDWmVbmbd53rSA2iFoDcmXu](https://cid.ipfs.io/#QmanmTVLostTHeeLiz8vr99QDWmVbmbd53rSA2iFoDcmXu) |
| Links | 32 | [QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq](https://cid.ipfs.io/#QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq) |
| Links | 32 | [QmTfDsTDe3nVu7b3hij43R3mBzyhJZgVm9eFBewVb5FfKV](https://cid.ipfs.io/#QmTfDsTDe3nVu7b3hij43R3mBzyhJZgVm9eFBewVb5FfKV) |
| Links | 32 | [QmRF3DNTkA43a7AG26uva4n7pgR22ctz6PjZW4KMuN5Cvu](https://cid.ipfs.io/#QmRF3DNTkA43a7AG26uva4n7pgR22ctz6PjZW4KMuN5Cvu) |

We can now map out the links with their respective roots. Notice that link “Qmdsy” is referenced by both root objects.

Diagram

Description automatically generated

**Figure 4.0.5: Merkle DAG representing sample records**

## **4.2 Quick Design**

After identifying the requirements, a design of the proposed application is created. This is not a detailed design with complete technical specifications but a simplified one with critical aspects of the solution. This phase will give a bird’s eye view to the client of the application.

**4.2.1 Context Diagram**

**Diagram

Description automatically generated**

The context diagram shown in Figure 4.0.6 summarizes the application on inputs and outputs of the system and targeted users. On general, users of the application will be required to provide public/private keys and vaccine details. Application will generate vaccine certificate and summary details based off these details.

Then it will trigger and execute various processes to upload, encrypt/decrypt, or release files. Note that this is a general illustration of inputs and outputs. Next sections of this chapter will discuss the mentioned processes on this diagram.

**4.2.2 Data Flow Diagram**

**Diagram

Description automatically generated**

Figure 4.0.7 illustrates how various types of users receives and provides information to the application and how the application provides and receives data from users. This also mentions the executing process to generate the data.

It is important to note that authorized medical personnel are the only ones allowed to upload files. Patients will have to generate private and public keys for their files to be uploaded or requested. These keys are crucial for a patient file to be encrypted or decrypted. Third parties can request for patient files and will be granted access to view decrypted files.

**Diagram

Description automatically generated4.2.3 Use Case Diagram**

**Figure 4.0.8: Proposed Use Case Diagram**

The suggested application's development is not solely dependent on the system's functionality. It also depends on the workflow procedure that needs to be identified, implemented, and followed. The components of the proposed application “A Blockchain Implementation for Secured Vaccine Certificates”, is demonstrated in Figure 4.0.8 and utilized a Use Case Diagram. The patient, being the central user of this system will provide appropriate keys with reference to the executing process. These in turn can trigger uploading or granting of view access to either medical unit or a third party.

**4.2.4 Transactional Operation Diagram**

Table

Description automatically generated with low confidence

Figure 4.0.9 illustrates the operations that exist in the proposed application. It is divided according to the users triggering the process. The crucial process of generating the private and public keys will be prompted by the patient. Without these keys, medical personnel cannot upload files which in turn, the third parties will not be able to request any file validation.

**4.2.5 System Flowchart of the Proposed Application**

Patient Registration

Figure 1

Diagram

Description automatically generated

Create Vaccine Record

Figure 2

Diagram

Description automatically generated

Store Files

Figure 3

Diagram

Description automatically generated

File Validation

Figure 4

Diagram

Description automatically generated

Retrieve Files (patient only)

Figure 5

Diagram

Description automatically generated

Figure 1 shows patient registration process. Once the patient successfully register, public and private keys will be generated based off patient’s details. For confidentiality, private key will be masked by user’s password. User’s password will then be hashed before saved to the database.

Figure 2 details the process on creating vaccine record. Users will be required to scan a QR code with valid patient details. If this is not met, user is not allowed to enter vaccine details. If met, user will be required to input details such as Dose Number, Vaccine Brand etc. Once submitted, app will generate vaccine file based from these details and be stored with the app’s storage mechanism.

Figure 3 illustrates how data will be stored. First, the patient public key will be checked and if exists, will proceed to storage process. App will first generate a file hash from the raw file then proceeds on encrypting the file using the public key. Next, encrypted file will be sent to IPFS and IPFS will return a Content Identifier. Both filehash and Content Identifier will be sent to the blockchain.

Figure 4 describes the method for file validation. User will be allowed to upload the raw file. App will generate a hash from the file and said hash will be checked against existing blockchain logs. User will receive either a successful or error message for the transaction.

Figure 5 demonstrates file retrieval process. Patient’s metadata is automatically generated once logged in. Once metadata is validated from the blockchain, app will request encrypted file from IPFS and decrypt it using patient’s private key then will be available to the patient via browser download.