iBlock

Person Verification Platform
Final Report
PID 6

Version 1.0

Mentor: Dr. Thanuja Ambegoda

Group Members:

- 190239A - K. G. Akila Induranga
- 190241X - M. I. M. Ishad
- 190242C - Kasun Isuranga

Abstract/ Executive summary

individuals prefer to do numerous tasks online rather than showing up in person. But, some services require you to be in person to provide services to get the identity verified with high confidence. The constraint of using digital platforms is that everyone cannot trust the digital information as it can be altered by the hosted organisation. The development of blockchain technology unlocked the tamper-proofness of the digital world. Each transaction is approved with a consensus algorithm and logged in a blockchain which cannot be altered later. This provides the ability to create a digital platform to verify personal identity with high confidence. The main goal of the project is to provide users complete control over their identification data while allowing service providers to access identity data without storing them in their own databases and the ability to track user information changes.

Table of Content

1. Introduction

1.1 Background of the application domain/ problem

For some sensitive processes of everyday world, the person participating needs to be verified. For example, in a scenario where a person opens a bank account, the person's identity needs to be verified. This process involves visiting a certain physical location, which can be time consuming and usually requires physical presence.

Currently used technologies for remote verifications cannot provide reliable verification of a person's identity. Also storing credentials and other sensitive information on a centralised entity might make data breaches more likely and will not provide an user control over his or her credentials.

1.2 Motivation for the selected system development

Using the technology as of now, an online and remote process can be introduced as an alternative to the traditional process. Smartphones are commonplace now and smartphones can act as placeholders for digital wallets. Blockchains can be used as decentralised storages, solving issues such as low data security and low confidentiality with centralised storage systems.

1.3 Importance and main purpose of the system

The system can be used to simplify and hasten the now used traditional process of person verification, mostly using already available resources and equipment. The usage of the system does not require any physical interactions between the parties (only online interactions). Therefore, the system resolves issues regarding locality and can reduce time requirements of the person verification processes. With the use of blockchain technologies, a transparent mechanism can be provided to analyse the history of interactions.

The system aims to provide a platform and necessary tools to verify people in an online and remote procedure for an entity. Additionally, users have finer control over their credentials.

1.4 Overview/ summary of the system and used approach and outcome

When a user wants to use a service which requires a verification of identity, the user is provided a QR code. After scanning the QR code with the mobile application the user will be prompted to accept or reject the request. After accepting, a request is sent to the blockchain. The library will watch the contract of the user and fetch the required data when possible.

2. Literature Review

Already, there are some types of digital ID verification systems such as Jumio Identity Verification [3]. However, Digital identity systems that exist today are fragmented among service providers. Users need to duplicate their identity information between services, which reduces overall usability and increases the risk of data compromise [4].

With the evolution of web 3.0, centralised databases that hold the credentials of users are fading away from the internet by replacing them with more secure, transparent, controllable decentralised systems having no single point of failure. Use of blockchain technology has made these decentralised systems quite possible, hence the person verification too.

With blockchain technology, identity information is auditable, traceable, and verifiable — in just seconds [5].

uPort is an interoperable identity network for a secure, private, decentralised web. uPort returns ownership of identity to individuals. uPort's open identity network allows users to register their own identity on Ethereum independent from any centralised authority. On uPort, users are always in control of their data and they are free to share it with whomever they choose. The uPort project was later split into two projects, Serto and Veramo [6][7].

Deliverables .

- A Cross-platform application (mobile application/ standalone desktop application) that connects to a digital wallet and verifies the identity of the owner.
- Integration library for third-party service providers.
- A simple demo application to demonstrate the functionality.

3. System Models

3.1 System Requirement

3.1.1 Functional requirements

The functional requirements specified in the SRS document were slightly modified and implemented in the final product. The library has three functional requirements. Generating a QR code embedding the necessary information, loading and deploying (if not already deployed) contracts for a service provider and retrieving data from the blockchain after the user has approved the request are required from the library. The mobile application is required to have the following functionalities. Deploying a contract for an user, scanning generated QR codes, approving or rejecting requests to provide data, recovering an account and viewing transaction history of the account are the functional requirements of the mobile application.

3.1.2 Non-functional requirements

The key non-functional requirements of the system are security, confidentiality, transparency and ease of use.

The credential information is stored in the blockchain, which offers a very high degree of data security. The credential information can only be accessed with the consent of the owner, thus providing substantial data security and confidentiality.

As an ethereum network is used to store data, the users can access the transaction history of an user whose public key is known. As a blockchain, the storage is decentralised and immutable, further making the system more transparent and reliable.

The system is designed to be used by the general public, making the ease of use a key priority. The main interface the users access is the mobile application user interface, which is developed taking user experience into account. The user interface is designed to be simple, improving accessibility to the general public.

3.2 System Design

The system is composed of three main components.

Library – Used to integrate third party service providers web applications with the system

Mobile Application – Provided to users and is used to manage and use credentials

Smart Contracts Application – Acts as the storage of the system, stores data about transactions (Creations and modifications of contracts)

The library and mobile application communicate through the smart contracts application. Service providers access the system from the interfaces in the library and users access the system through the mobile application, thus providing communication links between users and service providers.

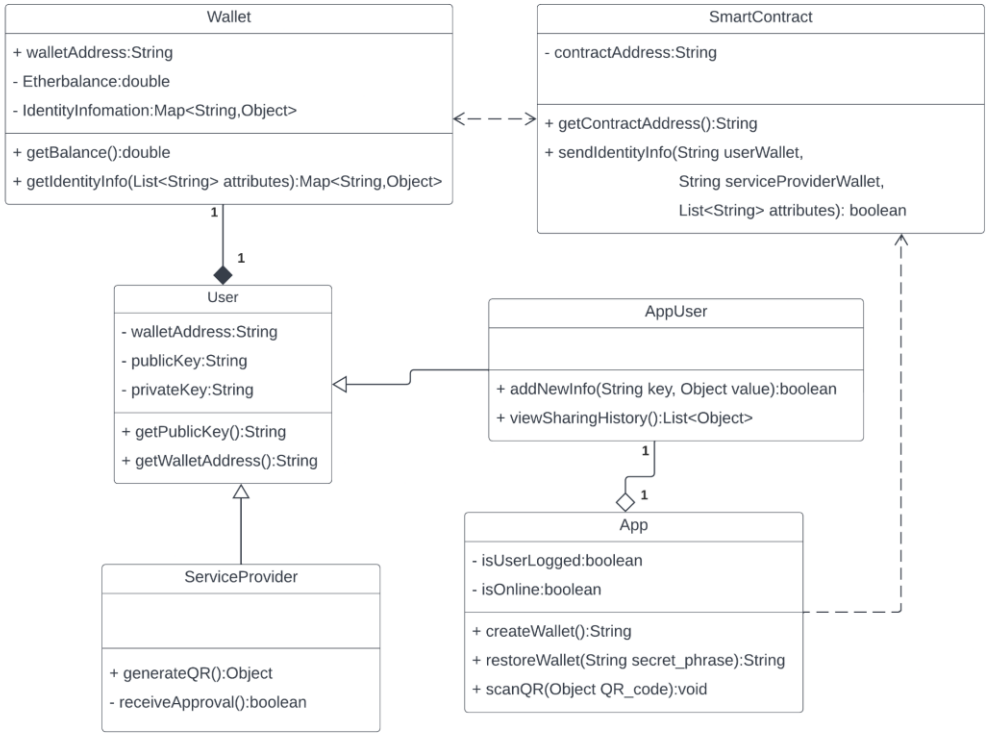


Fig. 1 Class Diagram

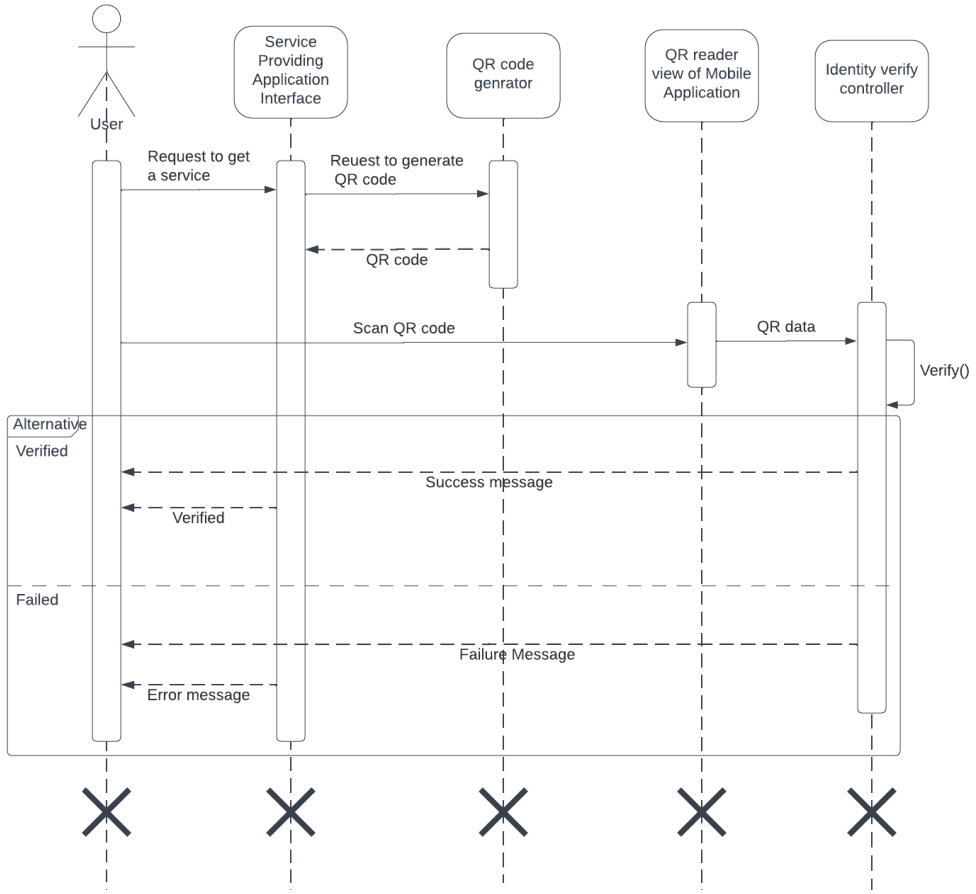


Fig. 2 Sequence Diagram of verifying a person

The library uses a polling mechanism to fetch data from the blockchain, that is it waits for the contract of the user to be modified adding the blockchain address of the verifier which is sent through the QR code to be added to the approved verifiers list. The library will check for changes in the contract of the user from time to time till a specified timeout. If the address is added to the list, the library will fetch the data from the contract of the user.

The QR code is used to send the data from the library to the mobile application. The QR code contains a generated token, the verifier name and the verifier blockchain address.

The mobile application uses the blockchain address of the verifier to modify the contract of the verifier and add the data to a map using the token and user contract address as key and value respectively.

3.3 Database Design

Mobile application uses an embedded database to store the information shared records. It uses only one table(Log table) as all other data is stored in the blockchain.
Log table consist of 3 columns,

- Txhash
- VerifierName
- VerifierContractAddress

4. System Implementation

4.1 Implementation Procedure

Mobile Application – Flutter Development kit

Mobile application is a user friendly interface to interact with their contracts. The development procedure started with developing a prototype with figma [1]. For application development Flutter framework as it can build both android and IOS applications with a single code base. Since, flutter itself doesn't contain all functionalities some packages are used in the development. The flutter_bloc [2] library is used to separate the business logic from the UI rendering logic.

Smart Contracts – Solidity, Remix IDE

Ethereum blockchain [3] was chosen considering the ability to run programs on the Ethereum Virtual Machine (EVM). The smart contracts deployed to the blockchain were implemented using the Solidity programming language [4] on the Remix IDE [5]. Ganache and Truffle

development kit [6] was used as the development environment and for the deployment, it was decided to use the Goerli Ethereum Testnet [7], and Infaura [8] to get access APIs

Integration Library as NPM package

The library was implemented using JavaScript and uses the libraries qrcode [9] [] for generating the QR code and web3js [10][ ] for interfacing with the blockchain. The library is published as a npm package named ‘iblock-verifier’ [11] [].

4.2 Materials

No existing data was used in the implementation of the system.

4.3 The Algorithm

As the system is a person verification platform, data flows are the major components and important algorithms are not present.

4.4 Main Interfaces

4.4.1 Interfaces with users

The interface between users and the system is the user interface of the mobile application.

Fig. 3 iBlock Mobile Application

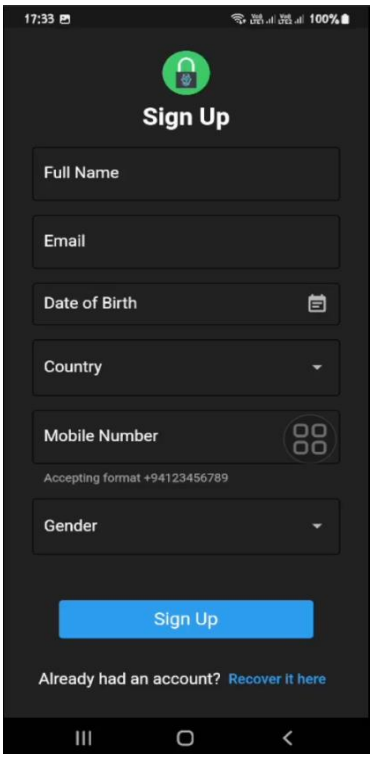


Fig. 4 Signup page

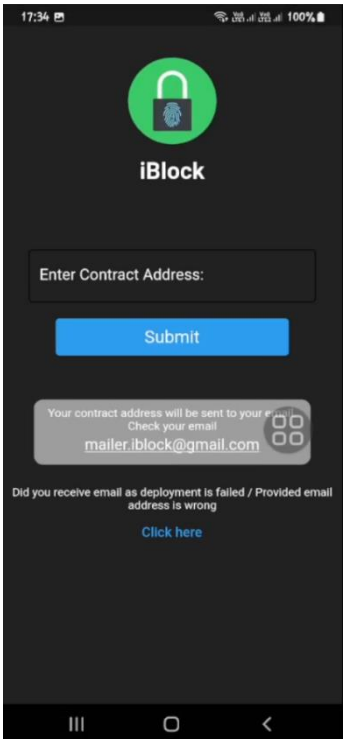


Fig. 5 Page to enter the address sent to the email

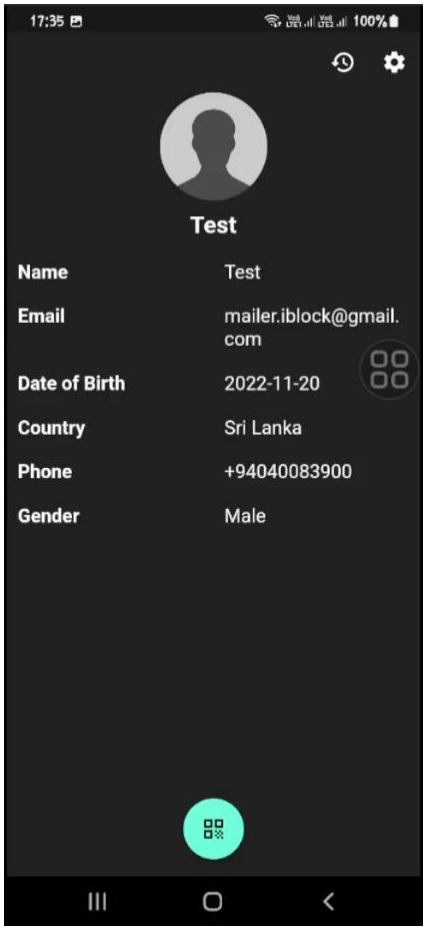


Fig. 6 Details page

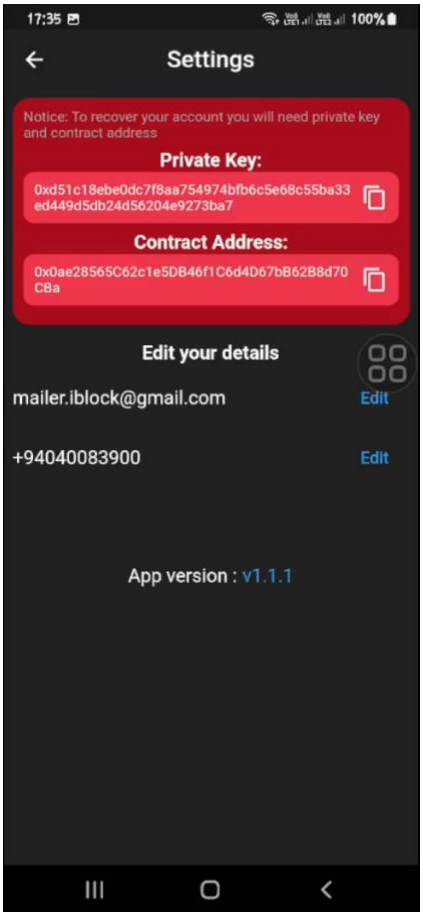


Fig. 7 Edit details page

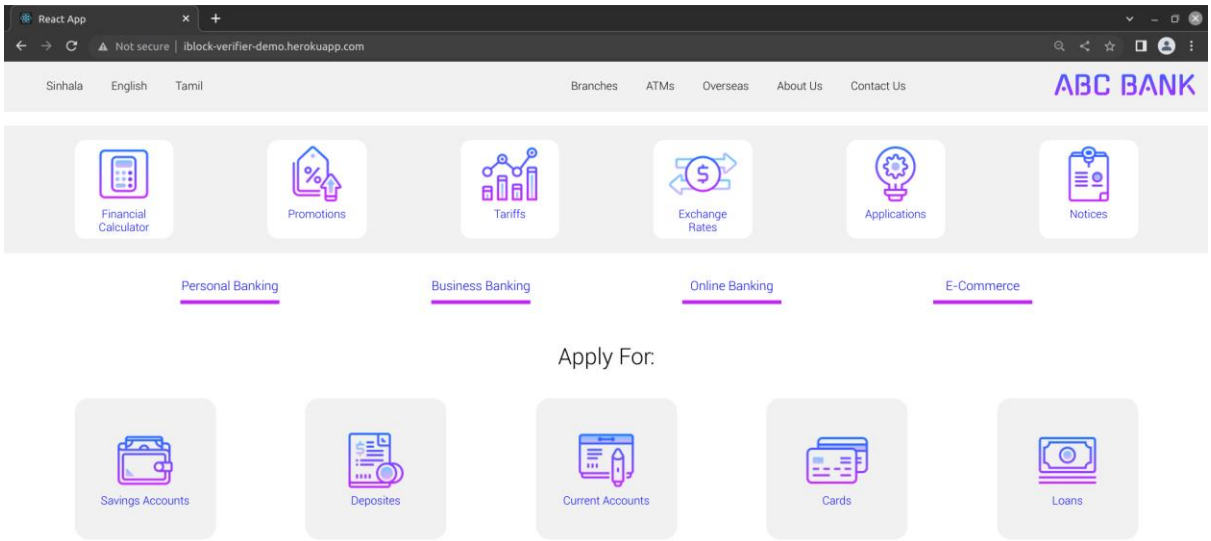


Fig. 8 Demo application home page

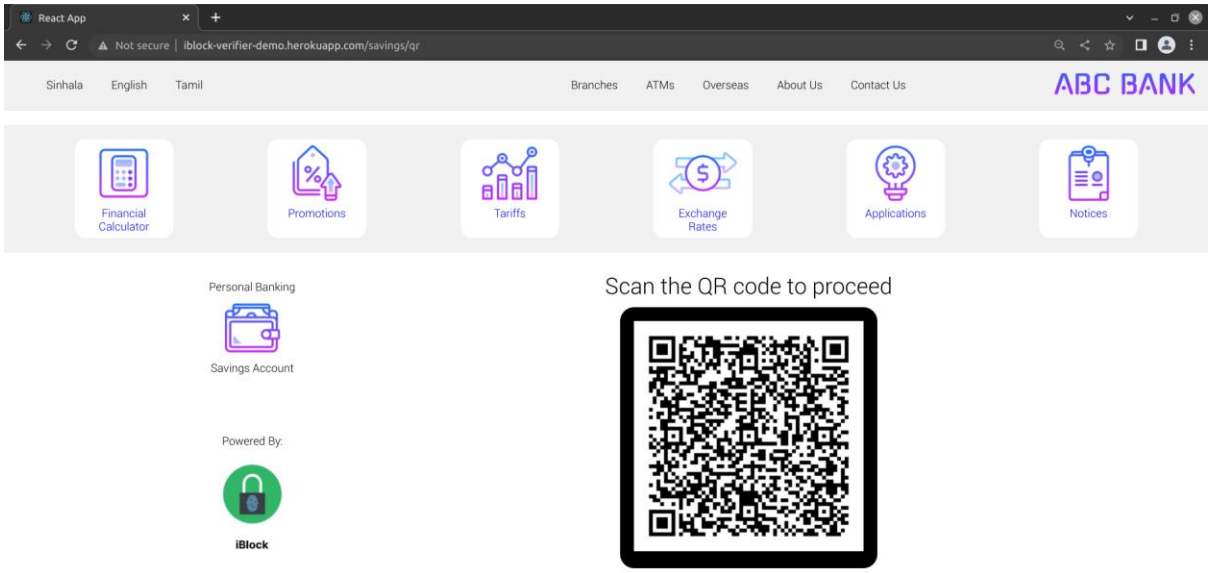


Fig. 9 Page for opening a savings account

4.3.2 Interfaces for service providers

The service provider interacts with the system through the interfaces of the library. getQR, loadContract and getTokenVerified are the interfaces provided by the library.

getQR

```
/**
 * Get the QR code generated in the format accepted by the 'iBlock Mobile Application'.
 *
 * @param {String} _verifierName The name of the verifying institution.
 * Will be prompted to the user who scans the QR code using the 'iBlock Mobile Application'.
 *
 * @returns {JSON} {
 *
```

```
 * qr: {String} - QR code URI that contains the verifier details,
 *
 * token: {String} - unique token to the verification
 *
 * }.
 *
 */
async function getQR(_verifierName)
```

getVerifiedToken

```
/**
 * Get the personal details of a user who scanned and approved a QR code with the given token.
 *
 * If the transaction is not succeeded yet, return "PENDING" as status.
 *
 * If the user has rejected permission, return "REJECTED" as status.
 *
 * @param {String} _token A token previously generated using the 'iblock-verifier package' to identify the verification transaction.
 * @param {String[]} _listOfDataFields A list of data fields of users which need to be fetched from the blockchain
 *
 * currently allows the set of ["name","email","DOB","country","mobile","gender"] only.
```

```
 *

 * @returns {JSON} {

 *

 * status: {String} - status of the verification. "PENDING" or "APPROVED" or
"REJECTED",

 *

 * data(optional): {JSON} - user's data fields fetched from blockchain, only applicable if
status is APPROVED.

 *

 * }

 *

 */

async function getTokenVerified(_token, _listOfDataFields)
```

loadContract

```
/**

 * Load ( or deploy and load) the smart contract tied to the verifier.

 *

 * Read the contract address from the 'deployed-contract' file,

 * or deploy a new contract if the 'deployed-contract' file is not present.

 *

 * @returns {JSON} {

 *

 * private-key:{String} - Private key of the Ethereum Account Address of the verifier,
```

```
*

* verifier-address: {String} - Ethereum Account Address of the verifier,

*

* contract-address: {String} - Smart Contract address of the verifier

*

* }.

*

*/

async function loadContract()
```

5. System Testing and Analysis

5.1 Testing approach

Unit testing is conducted for individual testable components and used the following packages to conduct those unit tests.

- Mobile application - flutter_test package
- Mobile application backend - jest
- Contracts - truffle suite testing

Then integration testing is conducted for the components connected. Since, mobile application is connected with it's backend the mobile application tests include calling the backend APIs.

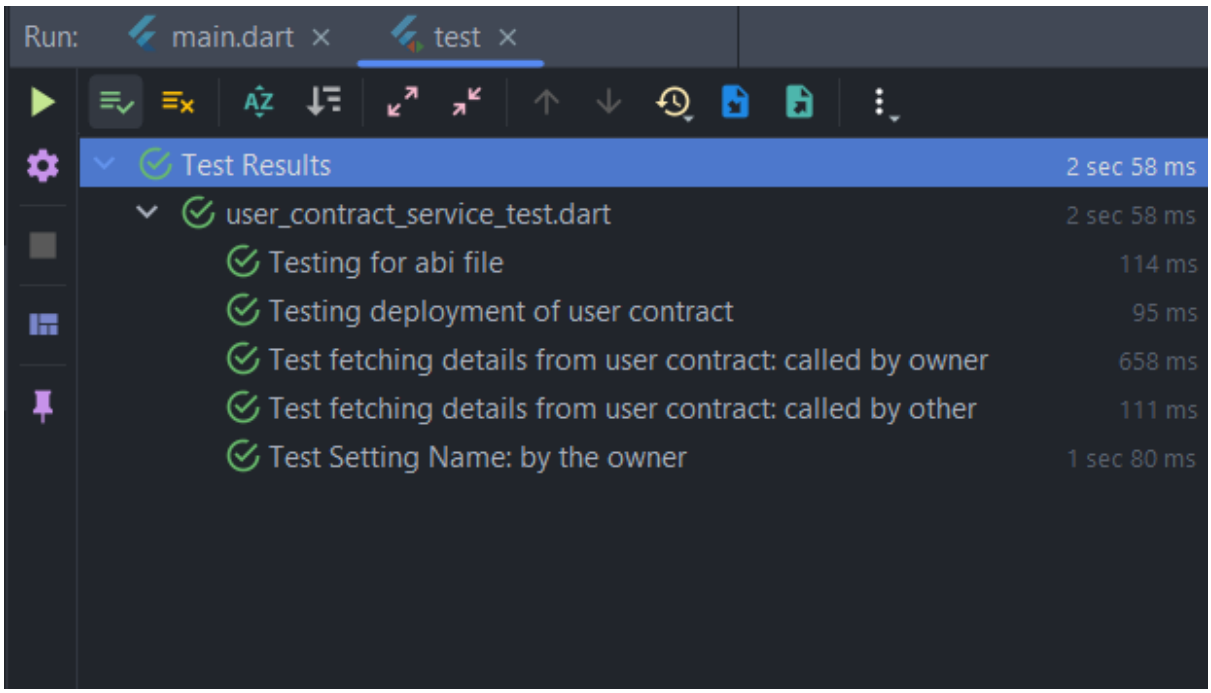
Performance, CPU and memory profiling is done by recording the usage via Flutter DevTools.

5.2 Unit Testing, Results and analysis of testing

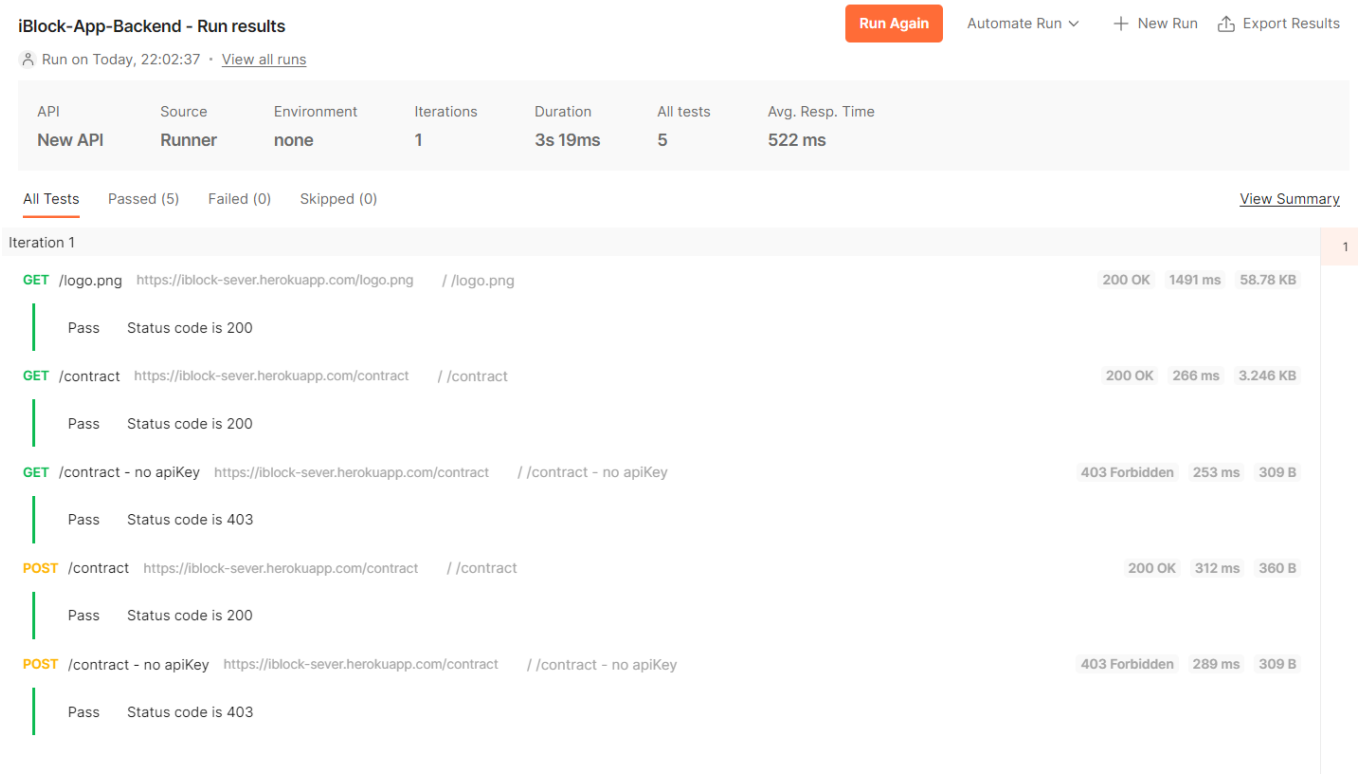
- Unit test results of app backend

```
PS D:\Projects\sem5\iBlock-App-Backend> jest
 PASS  test/app.test.js
 PASS  test/services/mailgun_service.test.js
-------------------|---------|----------|---------|---------|-------------------
File               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------|---------|----------|---------|---------|-------------------
All files          |     100 |      100 |     100 |     100 |
 mailgun_service.js |     100 |      100 |     100 |     100 |
-------------------|---------|----------|---------|---------|-------------------

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        4.934 s, estimated 5 s
Ran all test suites.
```

- Unit test results in mobile application



- API testing with postmon for Mobile Application backend



- Testing the smart contract:

It is recommended to run the testing of the smart contract on a pre-setup environment such as remix IDE as it already has the blockchain and Ethereum accounts set up built in. Load the smart contracts to the remix IDE as well as the test files given in the github repository, and run the test files.

- Unit tests in User Smart Contract

test-user.js | user_contract.sol

```
// Right click on the script name and hit "Run" to execute
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("getName and setName", async function () {
    const User = await ethers.getContractFactory("User");
    const user = await User.deploy("Name","Email","DOB","Country","Mobile","Gender");
```

0 listen on all transactions Search with transaction hash or address

```
RUNS scripts/script.ts....
   ✓ test initial name (62 ms)
user contract deployed at:0xd2a5bC10698FD955D1Fe6cb468a17809A08fd005
user contract deployed at:0xddaAd340b0f1Ef65169Ae5E41A8b10776a75482d
user contract deployed at:0x0fC5025C764cE34df352757e82f7B5c4Df39A836
user contract deployed at:0xb27A31f1b0AF2946B7F582768f03239b1eC07c2c
user contract deployed at:0xcD6a42782d230D7c13A74ddec5dD140e55499Df9
user contract deployed at:0xaE036c65C649172b43ef7156b009c6221B596B8b
user contract deployed at:0x9d83e140330758a8fFD07F8Bd73e86ebcA8a5692
   ✓ test updating and retrieving updated name (3432 ms)
   ✓ test initial email (52 ms)
verifier contract deployed at:0x5FD6eB55D12E759a21C09eF703fe0CBa1DC9d88D
   ✓ test updating and retrieving updated email (182 ms)
   ✓ test initial DOB (37 ms)
   ✓ test updating and retrieving updated DOB (159 ms)
   ✓ test initial county (49 ms)
   ✓ test updating and retrieving updated country (704 ms)
   ✓ test initial mobile (212 ms)
   ✓ test updating and retrieving updated mobile (281 ms)
   ✓ test initial mobile (82 ms)
   ✓ test updating and retrieving updated gender (283 ms)
   ✓ test verify from user contract (249 ms)
Passed: 13
Failed: 0
Time Taken: 5793 ms
>
```

- Unit tests in Verifier Smart Contract

Home | test-verifer.js | verifier_contract.sol

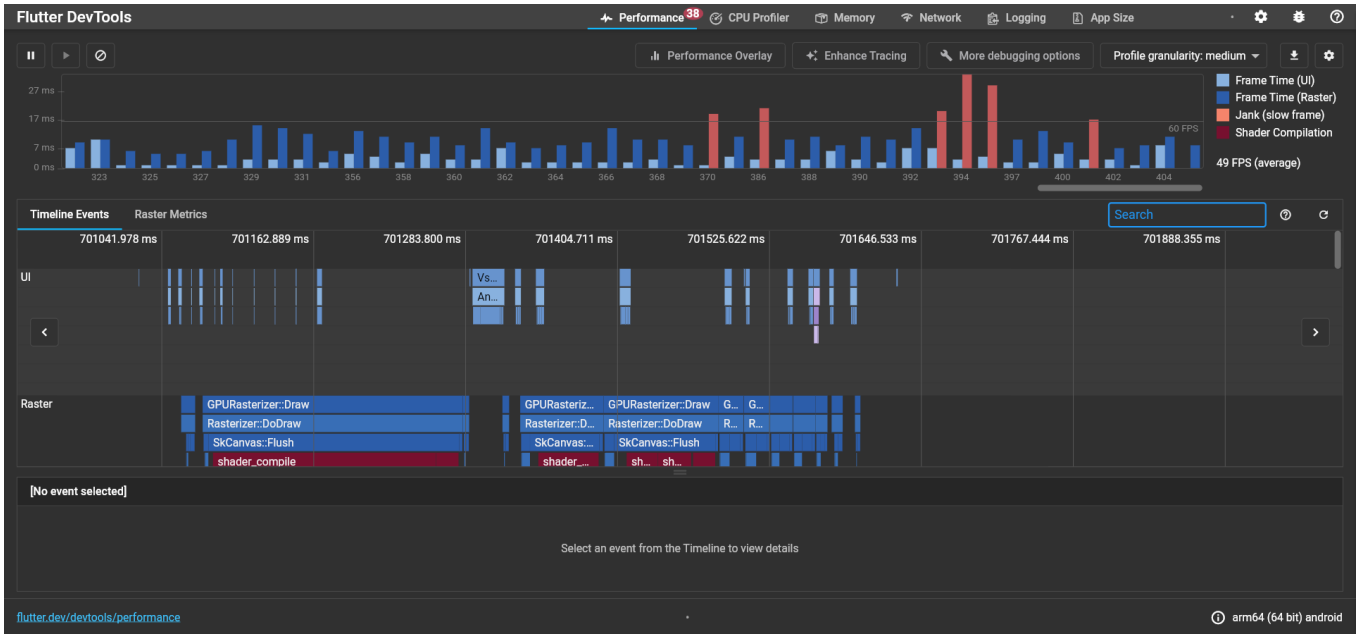
```
// Right click on the script name and hit "Run" to execute
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("Verify Token", async function () {
    const Verifier = await ethers.getContractFactory("Verifier");
    const verifier = await Verifier.deploy();
    await verifier.deployed();
    console.log('verifier contract deployed at:'+ verifier.address)

  it("test verifyToken", async function () {

    const User = await ethers.getContractFactory("User");
    const user = await User.deploy("Name","Email","DOB","Country","Mobile","Gender");
    await user.deployed();
    console.log('user contract deployed at:'+ user.address)

    expect(typeof await verifier.verifyToken("sample Token",user.address)).to.equal("object");
  });

});
```

0 listen on all transactions Search with transaction hash or address

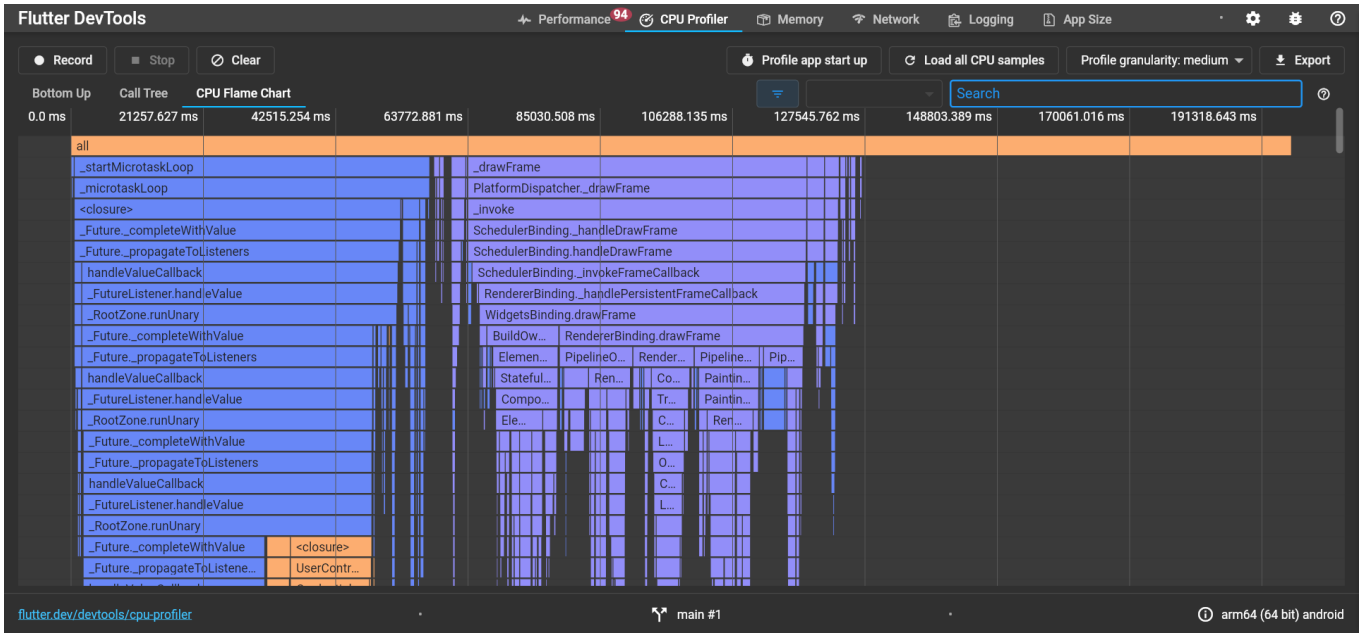
```
running tests/test-verifer.js ...
RUNS scripts/script.ts....
verifier contract deployed at:0x838F9b8228a5C95a7c431bcDAb58E289f5D2A4DC
verifier contract deployed at:0x9a2E12340354d2532b4247da3704D2A5d73Bd189
user contract deployed at:0x3c725134d74D5c45B4E4ABd2e5e2a109b5541288
   ✓ test verifyToken (1154 ms)
user contract deployed at:0x2E9d30761DB97706C536A112B9466433032b28e3
   ✓ test getVerifiedToken (608 ms)
Passed: 2
Failed: 0
Time Taken: 1765 ms
>
```

5.3 Aspects related to performance, security, failures

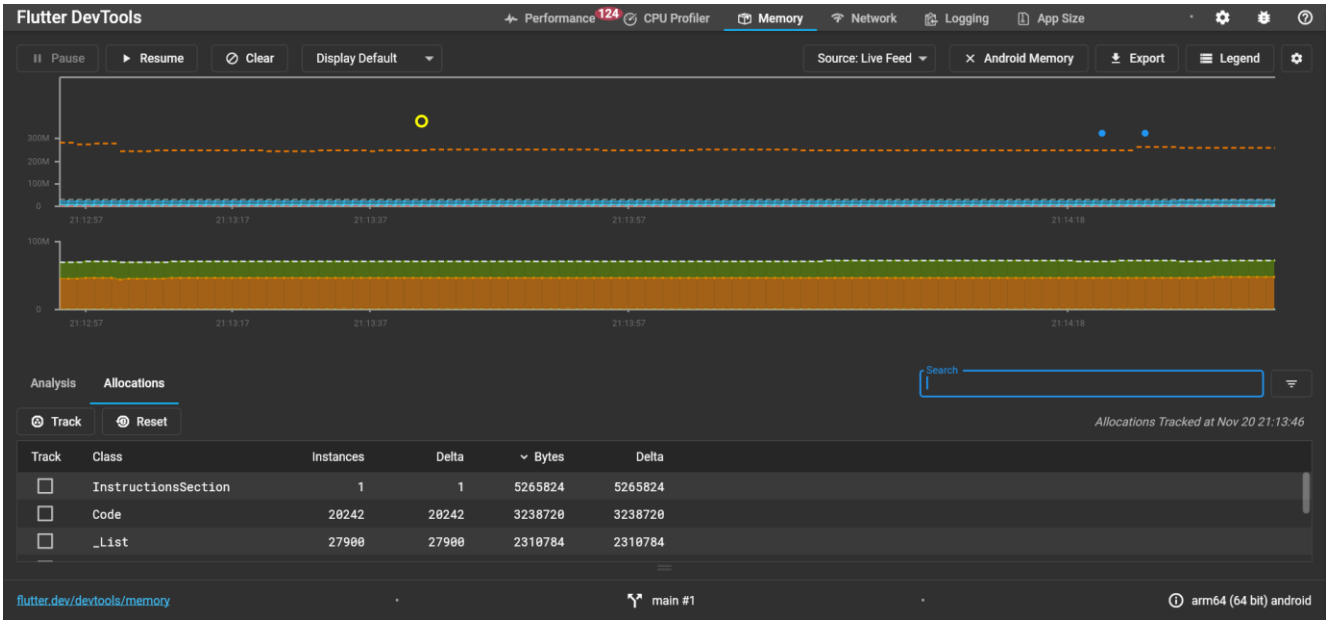
- Performance profiling report of the Mobile Application



- CPU profiling report of the Mobile Application



- Memory profiling for Mobile Application
  Memory usage is consistent. So, No memory leak in the application.



6. Conclusion and Future Work

The finished product can be considered successful as the requirements defined are fulfilled by the product and published to be used by the public.

The main requirement of the system was to provide a platform to verify identities of the users online using blockchain technologies to provide a transparent mechanism. The product provides the ability to verify people's identities remotely, with the use of a mobile application and a library. The credential data and verification processes are recorded in the blockchain and can be viewed by any authenticated user. In a scenario of data loss, corruption or installing the mobile app on a new device, the previous account can be recovered or reused provided that the user has the private key and the smart contract address.

Currently the product only supports a fixed set of data fields to be stored. Adding the ability to add data fields dynamically will be a major improvement to the system.

For usage and testing purposes Ethereum coins are required, which are currently mined by the computational resources of the development team and are not plausible for commercial applications. Therefore, subscriptions might be needed to keep the product operating.

As of now, data fetched from the blockchain might be stored on databases of the third party service providers, which can nullify or weaken data security offered by the product. Changes in the storage methods to decentralised systems would remedy this issue which will be possible in the future.

iBlock Person Verification Platform with these changes would be an excellent solution to the challenges faced by contemporary person verification methodologies and systems.

Reference

[1] https://www.figma.com/

[2] https://pub.dev/packages/flutter_bloc

[3] https://ethereum.org

[4] https://soliditylang.org/

[5] https://remix.ethereum.org/

[6] https://trufflesuite.com/

[7] https://goerli.net/

[8] https://www.infura.io/

[9] https://www.npmjs.com/package/qrcode

[10] https://web3js.org/

[11] https://www.npmjs.com/package/iblock-verifier