iBlock

Person Verification Platform
Master Test Plan
PID 6

Version 1.0

Mentor, Group members and contributions

Mentor: Dr. Thanuja Ambegoda

Index Number	Name	Contribution (Topics covered by each student)
190239A	K. G. Akila Induranga	● Evaluation Mission and Test Motivation ● Data Integrity Testing ● User Interface Testing ● Security and Access Control Testing ● Deliverables
190241X	M. I. M. Ishad	● Target Test Items ● Performance Profiling ● Failover and Recovery Testing ● Deliverables
190242C	Kasun Isuranga	● Function Testing ● Load Testing ● Risks, Dependencies, Assumptions and Constraints ● References

Revision History

Date	Version	Description	Author
06/11/2022	1.0	First Revision	K. G. Akila Induranga M. I. M. Ishad Kasun Isuranga

Table of Contents

Master Test Plan

1. Evaluation Mission and Test Motivation

The following master test plan report of the project 'iBlock' – the person verification digital platform solution is focused on what will be tested, why they have to be tested, and how each identified item will be tested according to the well-defined standards. 'iBlock' project was started with the mission of providing an easy-to-use, transparent, as well as a secured solution for both the people who want to verify their identity on certain occasions and for the institutions who want to get people verified before serving them. As the scope of the problem includes privacy and security concerns, it is needed for the system to be tested covering all such aspects other than the traditional UI/UX testing and performance testing.

Project 'iBlock' consists of a mobile application, a JavaScript library with well-defined APIs, and a sample demo web application. All those components will be tested along the way under the specifications mentioned in the report. The primary objectives of conducting such a testing procedure can be listed as follows.

- To identify as many bugs as possible that the users may encounter while using the product, and eliminate them. Since the product is supposed to be used not only by a single institute or a community, it is essential to eliminate the potential bugs as much as possible to reduce the number of users leaving the product for an alternative.
- To find any kind of a problem that can be considered as important in the aspects of quality of the product, and the predefined user requirements which were elaborated in extent in the previously presented Software Requirements Specifications.
- To identify the potential risks that the product may have to deal with after the deployment phase. As this product is identified as a high privacy and security-related product, the potential risks have to be dealt with without causing any kind of damage to the system, the procedure of usage, or the users at all possible times.
- To verify that the product meets the user requirements throughout the development phase, and the deployment phase. The personal details of the users are validated at each unit test and verified at the integrated tests as well. Both white-box testing and black-box testing are proposed to be carried out to ensure a smooth user experience while meeting user expectations at all possible times.
- To ensure the fulfillment of process mandates of development and deployment. Again, as the product is associated with privacy and security concerns, the legal status of the system needs to be thoroughly examined and any legal conflict has to be resolved before bringing the product to the user.
- To have the users advised about the testing phase and the procedure in order to take maximum out of user acceptance testing, when the stakeholders are provided with a 'beta' version of the product to test.

2. Target Test Items

Since there are many components in the platform each component will be tested independently and then integration tests will be conducted. Following are the identified testable components.

- iBlock mobile application.
- Mobile application backend
- Integration library
- Smart contracts
- Demo application frontend
- Demo application backend

3. Test Approach

The test approach presents the recommended strategy for designing and implementing the required tests. Here is the approach for testing for the project 'iBlock' – person verification digital platform.

- Data and Database Integrity Testing
    - Integrity is guaranteed by the blockchain technology

- Function Testing
    - Function Testing includes the testing done to the controller section of the mobile application, and the functions of the smart contracts.
- User Interface Testing
    - User Interface Testing includes the testing done for the views of the system which includes both the mobile application as well as the demo web application. This testing is done after the function testing because the functionalities must give the right business logic for the views to implement them.
- Performance Profiling
    - Performance Testing is very crucial since a lot of users may create accounts and get themselves verified using the blockchain at many different institutions. Handling of that many concurrent transactions is tested in this section.
- Load Testing
    - Load Testing is the testing done for the system under different data loads, the APIs of the library are tested for different loads in this section.
- Security and Access Control Testing
    - Security and Access Control Testing is done manually by using different blockchain addresses.
- Failover and Recovery Testing
    - This is also done manually to test how to recover from packet losses and power interruptions.
- Configuration Testing
    - The system needs to run on different platforms and different servers. So the system must be tested and configured to run on different platforms like Operating Systems and should be able to run on many servers depending on the server the project is going to be deployed on.

3.1. Testing Techniques and Types

3.1.1 Data and Database Integrity Testing

Project ‘iBlock’ completely relies on the blockchain technology on data creation, reading, updating, and deleting (CRUD). Because of the verification process of adding a new transaction to the blockchain (which can include any CRUD operation), the data integrity and availability is guaranteed almost 100% with zero downtime. Hence it does not need to be tested explicitly.

1.1.1 Function Testing

The product contains three applications, Smart Contracts, Integration Library and iBlock mobile app. For demonstrating the functionalities, a demo web application is also developed. Each of these needs to be tested to check whether the expected functionalities are provided. The applications of the product iBlock will be given priority when testing as they are the expected product. The testing will be conducted according to the following criteria.

Technique Objective:	The following will be tested by the testing procedures. Integration Library ● The data passed to the library by the interface is validated. ● A QR code with required data is generated and passed back through the interface. ● After the user has approved the request, required data is sent through the interface. ● Deploying a contract in the smart contracts by a third-party service provider. Demo Application ● The intended navigational structure of the website. ● A QR code is shown in required pages. ● After scanning the QR code and approving the request to fetch data, the fetched data is displayed. Smart Contracts ● Contracts are deployed properly. ● Setting lastVerifierID of a contract of an user. ● Data of a contract is fetched. Mobile Application ● A contract is deployed by filling a form and submitting. ● Valid data is extracted out of a QR code. ● Requests are sent to Smart Contracts after approving a data fetch request.
Technique:	Integration Library ● Each use case in objectives mentioned above will be tested separately using Postman API testing. Demo Application ● Mocha will be used for backend testing and Jest will be used for front end testing. Some manual tests will also be conducted. Smart Contracts ● Truffle will be used to test the Smart Contracts. Mobile Application ● Flutter dev tools provided in Flutter IDE will be used for testing.
Oracles:	Testing will ensure that for any given input, the necessary output will be given by each component. Automated testing will be used in scenarios applicable.

Required Tools:	• IDEs for code editing and debugging • Postman for Integration Library testing. • Mocha for backend testing of the demo application. • Jest for front end testing of the demo application. • Flutter devtools for testing of the mobile application. • Truffle for Smart Contracts testing.
Success Criteria:	All test cases for all the use cases are passed.
Special Considerations:	Some tests might need manual interference.

1.1.1 User Interface Testing

Project 'iBlock' consists of two types of user interfaces and both of them are supposed to be tested under the below-explained criteria to ensure the user interaction with the product goes smoothly, and conveniently. However, the mobile application will be subjected to testing under strict conditions while the demo-web application needs no such strict testing as it would not be a marketing product of the project. The goal of UI testing is to ensure that each object / component in the user interface does function as it is expected to be, according to the industry standards.

Technique Objective:	Exercise the following to observe and log standards conformance and target behavior: • Render the UI elements with minimum memory usage • Render the UI elements and the whole UI screen responsively for different screen resolutions and screen sizes. • All elements should be matched with the theme of choice (light / dark) • Color schemes used in the UI should match with the industry standards • The UI should render with no bugs within all the environments it is designed for (Android version 6 and up, iOS) • Buttons in the UI should load the exact required page at all times. • Virtual keyboard should pop up automatically for relevant form inputs. • The UI elements and their functionality should be easily understood by the user in a short period of time.
Technique:	• Design test cases to cover all the aspects mentioned above, using the built in Flutter testing library and carry out running test cases using different mobile emulators. • Use IntelliJ IDEA to emulate different versions of android environments. • User experience to be assessed through a phase of user acceptance testing with a beta version of the product, while the UI development is done 100%
Oracles:	Oracles for most of the test cases will be developers and testers themselves, as there are only a few test cases that can be automated such as form submissions, misspelled routes and other router related errors. All other test cases such as memory usage testing, user experience testing, and responsiveness testing will take place manually.

Required Tools:	● JetBrains IntelliJ IDEA ● Physical android device for initial testing ● VirtualBox to emulate iOS environment ● Flutter devtools ● Flutter testing library ● Flutter debugger ● VS code to debug errors in code and routing in the demo-web application ● Mozilla Firefox developer tools to test the memory usage and performance
Success Criteria:	● All the routes behave correctly and load the exact page required. ● No misspelled or a broken route crashes the user interface. ● Theme of choice applies for all the components in all pages persistently. ● User acceptance of UI receives success under the conditions of smoothness, responsiveness, understandability, and learnability.
Special Considerations:	Most of the user interfaces can only be tested using mocked data when they are tested alone, but in the user acceptance testing phase, the data can be loaded runtime and the interactive experience can be guaranteed.

3.1.2 Performance Profiling

Performance profiling is a performance test in which response times, transaction rates, and other time-sensitive requirements are measured and evaluated. The goal of Performance Profiling is to verify performance requirements have been achieved. Performance profiling is implemented and executed to profile and tune a target-of-test's performance behaviors as a function of conditions such as workload or hardware configurations.

Technique Objective:	Performance profiling will be conducted to ensure a better user experience and unexpected crashes in the application under the following circumstances. ● Under normal and heavy workload on the phone while running the mobile application. Since it will greatly affect the user experience. ● Under normal and heavy workload on the backend
Technique:	For performance profiling in the mobile application following technique will be used ● Run the flutter application in profiling mode. ● Connect to the flutter dev tools and start recording the events. ● Start using each and every functionality in the app to ensure they are recorded in the profiling. ● Condition changes from normal to heavy will be done by running a heavy CPU-intensive application along with the iBlock application Performance profiling in the backend will be done using recording the number of API calls that will be served within a limited period.
Oracles:	When using the functionalities to assess performance, the results of automated tests are taken into account. However, the results vary by platform and are also reliant on the user's internet connection.
Required Tools:	● Flutter devtools (To performance profiling in the mobile application). ● Apache JMeter (To test the performance of the backend)

Success Criteria:	**Testing with Flutter DevTools** If the time taken for the actions performed in the app is within the responsive range (< 10s) it will be considered a success. **Testing with JMeter** For testing each use case, the testing tools are supported for both single and multiple user transactions (using either one or multiple threads, according to JMeter). Additionally, there are looping options that allow running the same request repeatedly while utilizing the testing tools to obtain average results. If average results is in the responsive range (< 10s) it will be considered as a success
Special Considerations:	**Testing with Flutter Devtools** If frames of the application keep rendering repeatedly without performing any action will be considered as poor utilization of resources. (Because it keeps the CPU busy for the application for no reason) If the amount of memory used by the application keeps increasing without freeing up the memory it's an indication of a memory leak in the application.

3.1.3 Load Testing

Load testing will be primarily conducted in the demo application. The front end and back end servers will be subjected to thorough load testing to evaluate if the maximum load that will give reasonable performance will be adequate for the given uses.

Technique Objective:	Testing to evaluate the maximum loads that can be handled with reasonable performance by the demo application front end and back end.
Technique:	LoadRunner will be used to simulate user load on a server. Time sensitive data collected by the LoadRunner will be analyzed.
Oracles:	Different system configurations will have different results. Testing on a moderately powerful computer is required. Tests need to be automated as it is impossible to do load testing manually.
Required Tools:	● LoadRunner ● Resource usage monitors (Task Manager)
Success Criteria:	Response times under 5s are expected under moderate loads. (500 users)
Special Considerations:	Network performance also needs to be factored in.

3.1.4 Security and Access Control Testing

System-level security ensures that only those users granted access to the system are capable of accessing the applications and only through the appropriate gateways.]

The security and access control aspect of the product is considered in two key areas and the testing is also carried out under the same two areas namely;

- Application-level security – which will be focused on accessing the user data and other business logic functions
- System-level security – which will be focused on registering, logging and remote accessing the system

'iBlock' having both registering, logging features in the mobile application, and accessing personal information

of users on the blockchain using the smart contracts, will consider implementing and testing the security features of both above-mentioned key areas.

Technique Objective:	Exercise the following to observe and log standards conformance and target behavior: ● Application-level Security: a user of the 'iBlock' mobile application must be able to access his personal information that is stored on a smart contract under his Ethereum blockchain address at each login, and the same personal information access must be restricted to any other party which has a different Ethereum blockchain address, other than the verified address which the user will choose to approve at the scanning a QR code of a verifier. ● System-level Security: Backend system of the 'iBlock' product is pretty much almost hosted on the Ethereum blockchain, both user's side and the verifier's side. Therefore the system and its components are visible to any other party if they have the smart contract address. But the actual data and their edit permissions are always restricted to the authorized parties only.
Technique:	● Application-level Security: o Create tests for actual user address, random addresses and verify each permission by creating transactions specific to each user Ethereum address. o Modify user address and re-run tests for the same addresses. In each case, verify that additional functions or data are correctly available or denied. o Create tests for verifier addresses that have produced an actual QR code, a random address and verify each permission of accessing user information by creating transactions specific to each user Ethereum address. ● System-level Access: o Since the system backend part is deployed on Ethereum blockchain, the accessing and modified system components are restricted by default and hence this type of testing is not required for this product.
Oracles:	● For the mobile application which is built using Flutter, the security aspect is tested using the built in features of Flutter testing library and hence they can be automated. ● For the blockchain and smart contracts, the tests are written and automated using the built in features of Truffle test suite.
Required Tools:	● VS code text editor ● JetBrains IntelliJ IDEA ● Truffle test suite ● Flutter testing library
Success Criteria:	The technique only allows registered users to access only their personal information and allows the approved verifiers to access the personal information of the user who approved the verifier.
Special Considerations:	No special considerations.

3.1.5 Failover and Recovery Testing

Failover and recovery testing ensures that the target-of-test can successfully failover and recover from a variety of hardware, software or network malfunctions with undue loss of data or data integrity.

For those systems that must be kept running, failover testing ensures that, when a failover condition occurs, the alternate or backup systems properly “take over” for the failed system without any loss of data or transactions.

Recovery testing is an antagonistic test process in which the application or system is exposed to extreme conditions, or simulated conditions, to cause a failure, such as device Input/Output (I/O) failures, or invalid database pointers and keys. Recovery processes are invoked, and the application or system is monitored and inspected to verify proper application, or system, and data recovery has been achieved.

Technique Objective:	Simulate the following conditions of failure and observe the process of recovery to restore the mobile application, smart contracts, and the demo-web application: ● Power interruption to the server of demo web application ● Communication interruption through the network servers to both mobile application user as well as the demo web application server ● Missing, or renamed files that hold private keys in local storage ● Passing corrupted, mismatched data as arguments to the system functions ● Incomplete data flow cycles (from verifier to user and back to verifier) ● Lost connections to the blockchain.
Technique:	Some test cases are simulated in hardware conditions such as power interruptions while the test cases such as bad arguments are created using the test suite and can be automated. ● Power interruption to the demo web application server is simulated by initiating power down procedures for the deployed server. ● Communication interruptions are simulated by physically disconnect the devices from the networks, demo web application server is simulated this test case using configurations ● Missing or renamed files on local storage are simulated manually. ● Corrupted, or mismatched argument passing is simulated by creating separate test cases and running them. ● Incomplete data flow cycles are simulated by simply not scanning the QR code or do neither approve, nor reject after scanning the QR code. ● Lost connection to the blockchain is simulated by simply turning off the local blockchain servers, as the production Ethereum testnet is much less likely to go down because of the number of nodes.
Oracles:	This Failure and recovery testing needs some functions to be performed manually. Powering down the PC or the server manually in the middle of a transaction by switching them off or unplugging to test the results can be categorized into those conditions. But testing for a huge workload can be automated but as mentioned previously, the code must be modified accordingly to test it.
Required Tools:	● External power supplies such as UPS devices. ● External hard disks for backing up data ● Installation monitory tools such as Task Manager ● Flutter testing library ● Jest, Mocha testing framework

Success Criteria:	● Successful recovery of data on a Client or a Server power down situation. ● Successful rollback of the operation when an interruption occurs in the middle of a transaction
Special Considerations:	● Recovery testing is highly intrusive. Procedures to disconnect cabling may not be desirable or feasible. Alternative methods, such as diagnostic software tools may be required. ● Resources from the Systems (or Computer Operations), Blockchain, and Networking groups are required. ● These tests should be run after hours or on an isolated machine.]

3.1.6 Configuration Testing

Configuration testing verifies the operation of the target-of-test on different software and hardware configurations. In most production environments, the particular hardware specifications for the client workstations, network connections, and database servers vary. Client workstations may have different software loaded and, at any one time, many different combinations may be active using different resources.

Technique Objective:	Exercise the following to observe and log standards conformance and target behavior under different configurations to verify the operation of the target-of-test on different software and hardware configurations: ● To observe the behavior of application on different platforms (android and iOS) ● To determine the extent of the configurability of the application as a requirement. ● To analyze the performance of the application under different configurations and hence determine the optimal. ● To determine the rate of production of bugs under different configurations and reduce the number to a minimum.
Technique:	● Mobile Application – emulates the build of applications of both Android and iOS platforms. Different versions of the OS will also be picked to test the build (Android version 6.0+) ● Smart Contracts – deploy the compiled smart contracts on different Ethereum networks such as Rinkeby, Kovan, and Gorli. ● Demo Application – run the frontend on different web browsers and different browser versions to observe the compatibility. run the backend on different servers to observe the performance.
Oracles:	All the test cases will be carried out on different versions and variations of browsers, as well as different operating systems. The smart contract tests will also be carried out at the deployment on all possible Ethereum testnet blockchains. The results of the test cases and the user experience will be observed.
Required Tools:	● Web Browsers – Microsoft Edge, Mozilla Firefox, Google Chrome, Brave, Opera, Safari ● Operating Systems – Microsoft Windows 10 and 11, Ubuntu 22.04, Linux Mint 21, Mac OS X ● Ethereum Testnets – Rinkeby, Kovan, Gorli ● Mobile Operating Systems – Android 6.0+ , iOS 13+

Success Criteria:	Mobile application should run without failures in all configurations. Smart contracts should successfully get deployed and should be interactive on all Etherum testnet. The demo application should work without failures in all server and browser configurations.
Special Considerations:	• Deploying a smart contract into a Ethereum testnet is a process with a considerable cost, hence they are deployed to the active testnets only after testing well on the local testnets and re-deployments will be avoided at all possible times. • If the changes of a configuration does not create a reasonable impact on the system, those configurations are neglected.

4. Deliverables

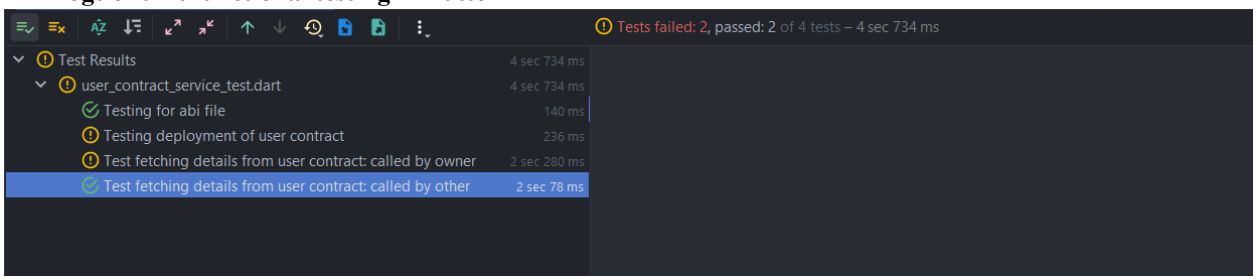
4.1 Test Evaluation Summaries

4.1.1. Test Logs

Test logs are comprehensive logs that can be utilized to discover in-depth details about the results of the tests.

When a test file fails, error messages are typically displayed, and the entire testing sequence is stopped, even if there are several test files in the program. This is typical for any testing tool. The tests can be resumed when the errors have been fixed, and if the tests are successful, a success message with more details will be presented. Test logs will always be visible in the console, success or failure. Here is an illustration of a test log.

Logs of unit functional testing in flutter

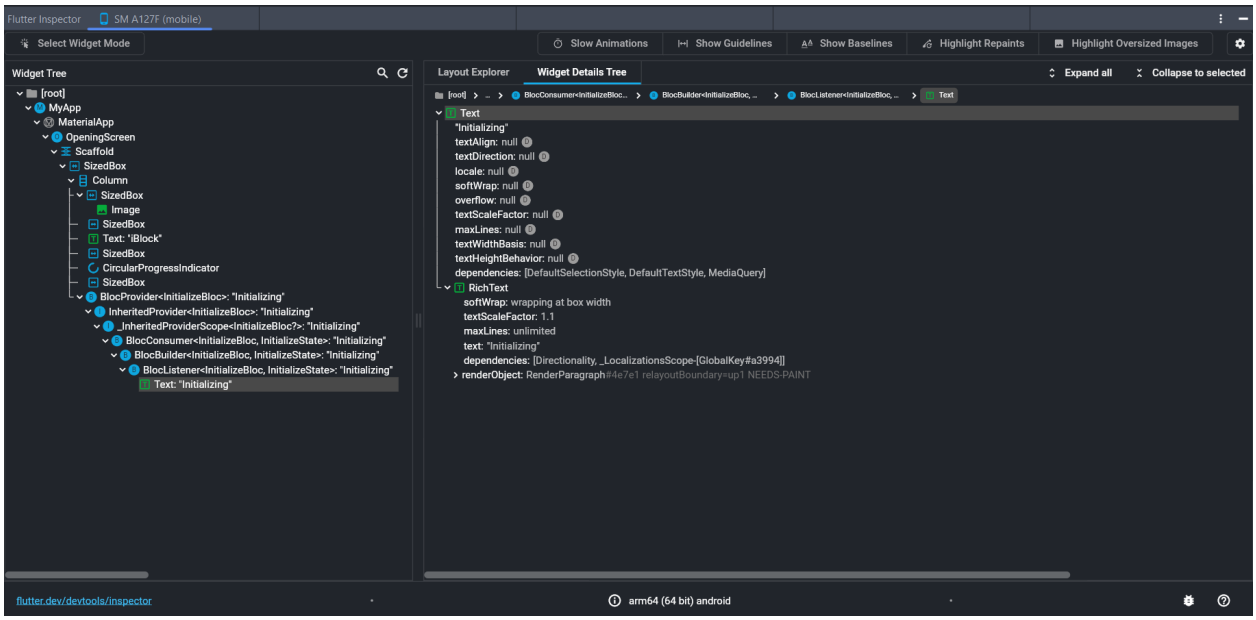


Logs of functional testing of smart contracts

```
Contracts/test on  main [!?] via  v16.17.0 took 2s
❯ truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Compiling ./contracts/user_contract.sol
> Compiling ./contracts/verifier_contract.sol
> Artifacts written to /tmp/test--55196-suHePnQcin6w
> Compiled successfully using:
   - solc: 0.8.17+commit.8df45f5f.Emscripten.clang


  Contract: User
    ✔ Testing the constructor arguments funtion of user_contract (225ms)


  1 passing (418ms)

Contracts/test on  main [!?] via  v16.17.0 took 9s
❯
```

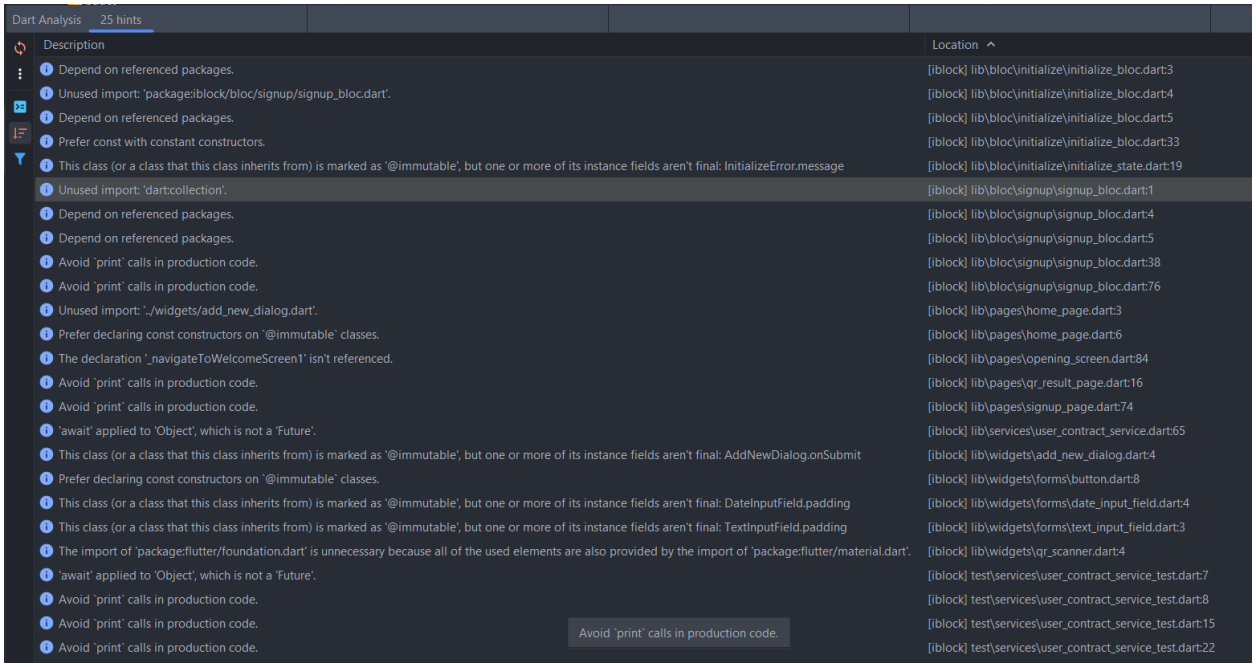
4.1.2. User Interface

For testing the UI in mobile application Flutter Inspector is used. Following is snapshot in UI testing using the Flutter Inspector tool.



4.1.3. Code Inspection

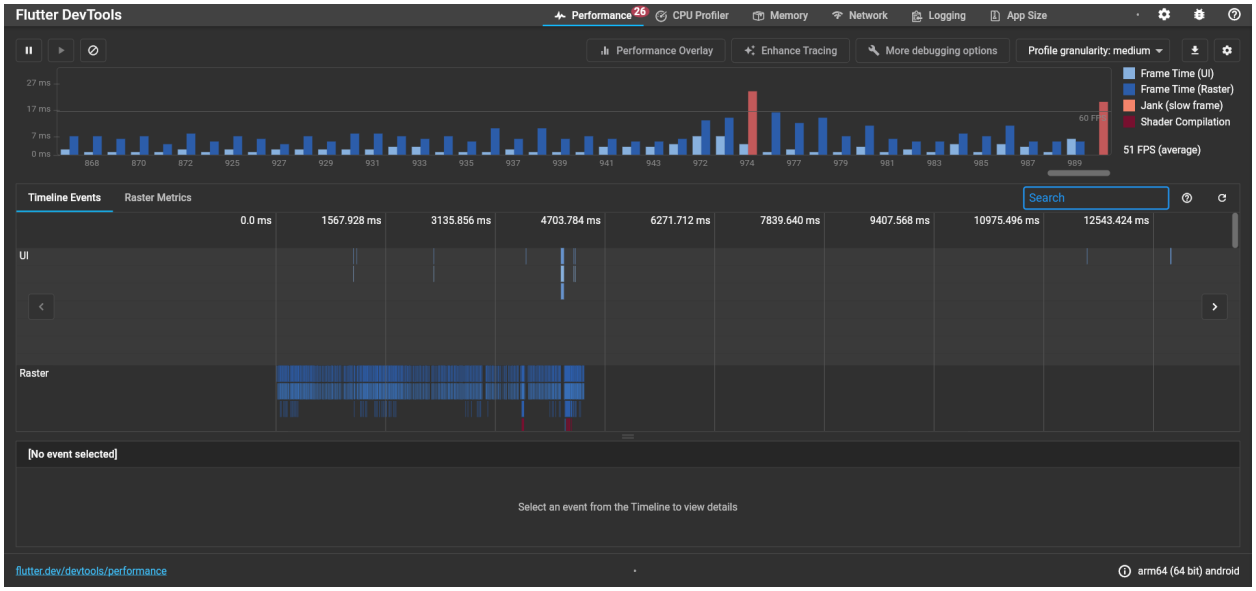
Inspections of the code can be used to increase its quality. Usually, modem IDEs offer this feature as a standard service and, with the use of development libraries, it will alert users to empty tags, misspellings, and other syntax corrections in the code base. Along with that, because of this if a web application, the front end code may be examined using the DevTools feature of the web browser (DOM inspections)



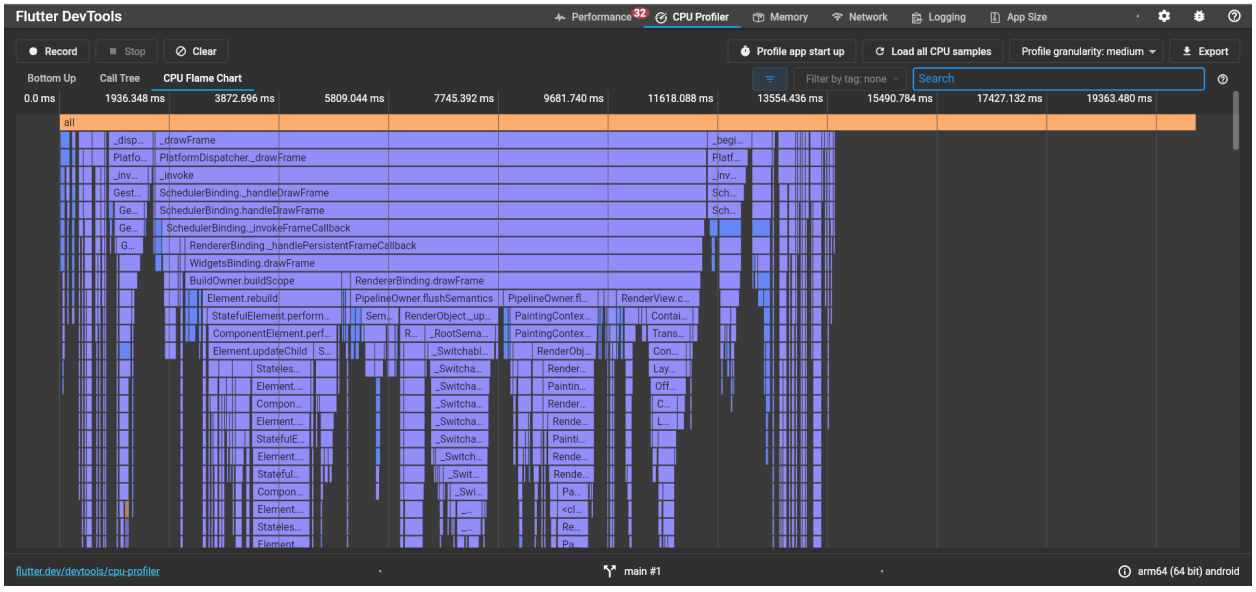
4.1.4. Mobile application Profiling

The profiling result under 3.1.2 Performance profiling will include following test results.

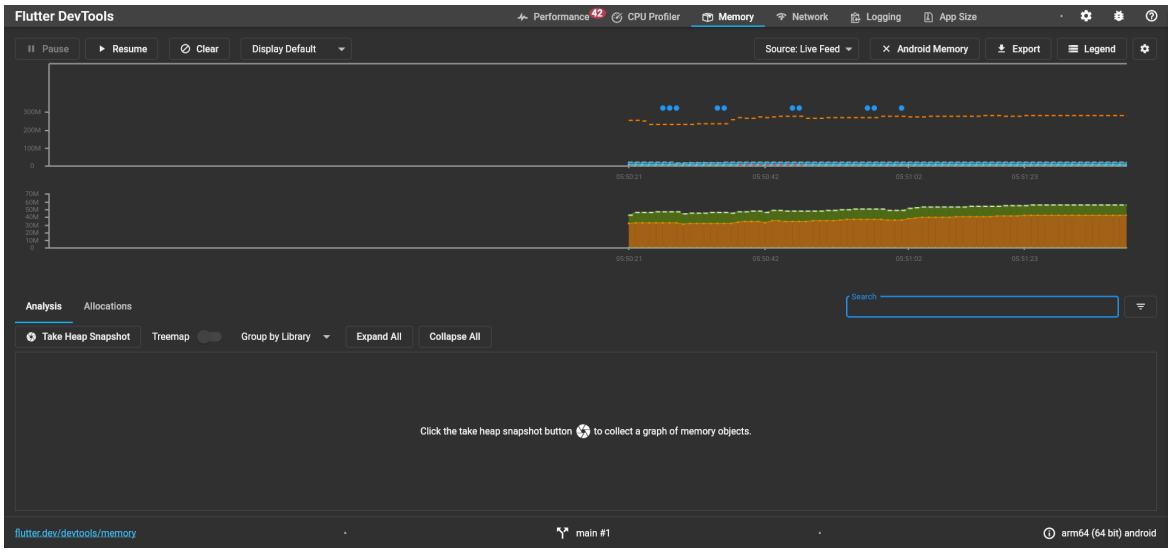
Performance Test



CPU Usage profiling



Memory Usage



5. Risks, Dependencies, Assumptions, and Constraints

Risk	Mitigation Strategy	Contingency (Risk is realized)
Test cases not covering possible exception scenarios.	Tester will thoroughly examine the code and the test cases and improve the test cases.	• Define new test cases • Identify exception scenarios
Corrupted configuration file of third party users.	Third party users will reinitialize a contract thus rewriting the configuration file.	• Reinitialize the contract
Smart Contracts failed to compile because of missing dependencies.	Tester will install the dependencies missing.	• Check for required dependencies and install
Prerequisites are not met.	Project 'iBlock' developer will define the prerequisites that must be met before Load Testing can start. Mobile application user and Web3 integration library developer of the verifier's institute will endeavor to meet prerequisites indicated by the project 'iBlock'.	• Meet outstanding prerequisites • Consider Load Test Failure

6. References

Postman : https://www.postman.com

Mocha : https://mochajs.org

Jest : https://jestjs.io

Truffle Suite : https://trufflesuite.com

Flutter : https://flutter.dev

LoadRunner : https://resultspositive.com/products/hp-loadrunner