IOTA Visualizer
— Report Two —

Ao Shen, Yuxiang Wu, Ruikun Cao, Jiangbo Yu, Xiao Luo, Dongxiao Huang
{as5017, yw1117, rc2917, jy3016, xl1716, dh4317}@doc.ic.ac.uk

Supervisor: Prof. William J Knottenbelt, Dominik L Harz
Course: CO530, Imperial College London

5th March, 2018

1 Progress Summary

We progressed consistently throughout the past month with minimal slippage. Many design approaches were discussed and tested. The details explaining our choices will form part one of our report.

Many challenges surfaced during development and testing. Discussions of our problems and solutions form the second part of our report.

A criticism of our previous report was the lack of task-delegation to each individual. While the role of each individual remains fluid, we have developed a task log (Appendix A), to detail each member's contribution to the team.

2 Prototype Description

We have completed a prototype, meeting the minimum requirements agreed to by the team and our supervisor. While feedback from our project supervisor regarding the UI and computed statistics may result in revisions to our schedule going forward (Appendix B and C), our specification remains largely unchanged.

2.1 Website

The basic web page from report one has been extended with a tree and spherical map of the IOTA network. Basic statistics are now also being updated dynamically on the website. Note that since the goal of this project is to provide a visualization tool, there exists minimal user input requirements. Thus, GET requests will constitute the majority of requests to the server.

2.2 Web Server

The web server is responsible for receiving client requests, querying the relevant data from the database and then sending it back to the client. In other words, it is an intermediary between the database and client. To customize the experience for each user, we indexed several routers. Each routed user will receive an initial data batch (approximately 100 data points for tree graphs and 500 for sphere graphs) from our IOTA collection in the database. A graph will then be drawn based on these data. We use cookies to save user information, and determine if updates are necessary. Cookies are removed from the client browsers and our database when the session ends.

2.3 Database

When instructed by the web server, the database will query transaction data from the IOTA API, update each node and calculate statistics for different transactions. To update transactions, we call the official API to discover "confirmed" nodes. For other nodes, we will check if a particular node has approvers. If a node has one or more approvers, it will be categorized as "unconfirmed". Otherwise, it will remain as "tips" (no approvers at all). Hashes from each node are then used to query their tips, trunk and branch nodes. Each of these data sets are then stored in a separate collection. To update the client's dataset, the database will check if the "confirmed status" of its own collection has changed.

Collection	Schema	Description
IOTA-related	hash(str), trunkTransaction(str), branchTransaction(str), type(str), value(str)	IOTA node details
Cookie Tracker	oldValue(array), updateValue(array)	Customize user experience
Statistics	Totaltips(number), MeanconfirTime(number), ValuePerSecond(number)	Compute statistics

Table 1: Database Structure Summary

3 Design and Solution Choices

3.1 vis.js vs SigmaJS

Initially, we implemented the SigmaJS [6] graph drawing library as was discussed in the first report. [3] However, as we learned more about the IOTA data structure, we thought vis.js [1] would be a better fit.

1. Edges and Vertices: Although SigmaJS is great at drawing charts and diagrams, vis.js was specifically designed to draw node graphs. This is important, because IOTA is literally a acyclic node graph. vis allows us to cull verticies and edges from sets directly to clearly convey concepts critical to the IOTA network. Extensions of this library also allow the user to dynamically interact with each node (or block), which might be a useful feature for potential extensions.

2. Dynamic Redraws: Since the aim of each IOTA transaction is to update its neighboring nodes, the status of each node is changing constantly. Additionally, there are new transaction broadcasts onto the network every second. We need to find a way to dynamically reflect these changes. SigmaJS was unable to do this. vis.js allowed us to dynamically add, delete and change the information of each individual nodes.

3.2 Runtime Environment - NodeJS vs PHP

We researched and experimented with both, but progressed with Node for the following reasons.

1. Front-back Integration: Unlike PHP, Node allows the same JavaScript files to be used for both front-end and back-end development. An integrated runtime environment is critical for team cohesion and integrated testing. Using different frameworks for front and back-end development will introduce unnecessary code reconciliation overhead and bugs.

2. Flexibility and Testing: Unlike PHP, NodeJS comes with fewer hard dependencies and rules that has allowed the community to develop many feature-rich libraries. The NPM package manager comes with a wide variety of frameworks and test packages like Mocha. The integrated nature of Node would also make our system testing more efficient.

There are drawbacks to Node such as single-threading and nested callbacks that are not present in PHP. Nonetheless, since we are not building high performance applications, single-threading will actually let us avoid many concurrency bugs.

3.3 Database Structure - MongoDB vs MySQL

Many of our team members have had previous experience working with SQL and MySQL databases. However we decided on MongoDB because SQL databases have restrictive rules. This required a lot of upfront investment in structuring the database. Since the IOTA data structure and API were still unfamiliar to us, flexibility and optionality were critical for us. This gave us more room to make mistakes, thus spending less time on fixing database problems and more time on other important aspects of our project.

4 Problems and Challenges

4.1 Browser Computing Capacity

Initially, in order to dynamically update the graph, the web browser had to sift out nodes that required updating and query the server for information. Once new data was fed from the server, the graph was redrawn. Because we had approximately 500 transaction data points, this overloaded the browsers and slowed them to a crawl. The first problem was that the sifting drained a lot of the browser's computing power. The second, was that after each redraw, the canvas was wiped clean which made it hard to track the dynamic growth of nodes. For the first problem, we moved the sifting to the server alleviating some pressure from the browser. For the second problem, we found that the data array could be updated by passing it by reference. This allowed us to avoid many iterations of schema copying. These problems were key motivators for us to create a back-end database.

4.2 Slow API

In many of our server-side processes, we needed to query data from the official IOTA API. However, due to limited throughput and slow status change times, the IOTA API updated very slowly. This was not anticipated, and forced our graph to appear less dynamic than it was. The limitations of the official API was beyond our control. We are currently looking at ways to increase the dynamism by loading older nodes first and use current ones as "updates." However, this is only a temporary solution that will result in the same problem eventually as current nodes are used up. In the meanwhile, we are also exploring other solutions to ameliorate the problem like folding our datasets so we can display them in rotation.

4.3 API Errors

While we were processing data from the official IOTA API, the 'getTips' method retrieved seemingly odd hashed blocks like "9999...." Nonetheless, when we used the findTransactionObjects method to retrieve information about that block, it reported a "hash invalid" error. To reconcile this problem, We had to write custom functions to filter out these invalid hashes manually. We subsequently reported this bug to the IOTA foundation.

5 Unit Testing

Unit testing is the first stage of our testing process. It will be followed by system testing and a discussion about regression testing.

5.1 Testing Software

1. Sinon.JS [7] was used to generates stubs for unit tests. We used this to full-effect by sending mock XMLHttpRequests to our back-end server so we can retrieve objects from IOTA API to test our written functions.
2. Our JavaScript unit tests were written using Mocha [5].
3. Our front-end was tested using the Selenium Webdriver and ChromeDriver
4. Our coverage report was generated using Istanbul which is integrated with Mocha.

5.2 Important Test cases

Target	Description	Expected	Actual
Connection	Authentication and timeout	Status Code 200	”
Database Ops	Add/Delete/Modify/Find	New query reflects changes	”
Schema	Validate entry length, characters, duplicity	filter correct	”
Statistics	Compare with hand-computed results	Results Match	”
Cookie Tracker	Created new collection for one user	Cookies Changed	”
getTips	Partition tested tip hash values	Stub matches output	”
getObjects	Validate object schema	Stub matches output	”
addAlltips	Validate tip schema	Stub matches output	”
queryTip	Correct objects are returned	Stub matches output	”
getLatestInclusion	Correct objects are returned	Stub matches output	Time out

Table 2: Test case summary of functions and operations critical to our project

5.3 Coverage

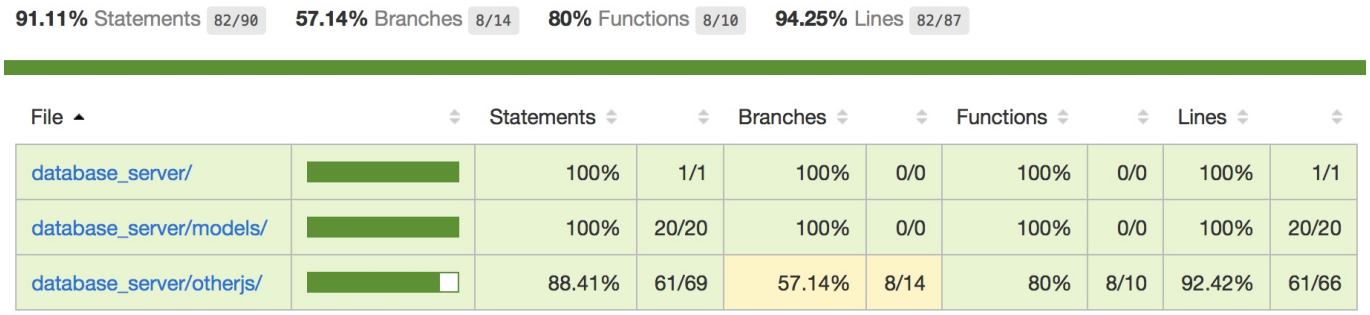

Figure 1: Unit Test Coverage Summary generated by Mocha and Istanbul libraries

The above is the summary of test coverage by branch, functions, lines and statements. It also illustrates how many tests we ran for each script. Since nearly all of our written functions are in models and otherjs, these test results are exhaustive when not considering code from third party libraries. It has to be noted that branch coverage is particularly lacking in otherjs. We have investigated the source and have narrowed it down to IOTA API's getlatestinclusion method. [2] It appears that this function calls getNodeInfo to find out the latest solid milestone hash, and then calls getInclusionStates. This chained API call takes very long and thus times out in our test. (Appendix D) While extending the timeout duration would allow us to pass this test, we believe it is more important to leave it in as the discovery of this bug is a great contributer to our project. Going forward, we have identified this API call as a negative to the performance of our algorithm and will call getNodeInfo and write our own InclusionStates function instead. Detailed coverage reports can be found in (Appendix E, F).

6 System Testing

6.1 Azure Performance Test with 1000 Concurrent Users

Our database server was deployed on Azure. To gauge the performance of our database under stress, we simulated a scenario where 1000 users would be visiting our website simultaneously. (Appendix G) We chose 1000, because usage at this level would exhaust our current Azure credits immediately. Average Response Time of 9.3 seconds, average requests of 280 per second and successful request rates of 97 percent all seemed normal according to performance benchmarks. [4]

6.2 Azure Performance Test with 2000 Concurrent Users

However, when we increased the load to 2000, our successful request rate dropped to 88.78 percent, with response times scaling down predictively. (Appendix H) After investigating the successful request issue, it seemed like 502 bad-gateway was the main culprit. (Appendix I) Most of the errors occurred when requesting for JavaScript files with IOTA APIs calls in them. We believe that this is a result of time-outs that occurred when the IOTA API is repeated queried. To tackle this problem, we will likely build a cache to pre-load blocks from the IOTA API, instead of querying it every time a new client navigates to our website.

6.3 Automated UI Testing

We utilized Katalon, a software test suite that records the tester's activities (for example, clicks, hovers, and navigation) to generate a testing script (Appendix J). These tests utilize selenium webdrivers that simulates human actions and HTTP requests. The record allows us to see the interaction between the front-end UI and our back-end servers, giving us great insight into the integration of our project components. Due to the static nature of these tests, they are generally version and implementation agnostic, meaning that the test results should not change unless new features are added to the UI. A successful test is generally defined as yielding a desired action. These can include:

1. A navigation click leading to the correct URL
2. A cursor hovering over a block chain, yielding the correct and relevant block information
3. Dragged blocks will bounce back to their original position
4. Correct statistics being displayed to the user

6.4 User Stories

We also ran several user story tests based on scenarios generated by humans. Some are from ourselves, while others are from our supervisors and peers. The goal of these tests is to ensure that our outputs are not only functionally correct, but also suits the requirements and specification of our supervisor and users. These manual actions on our UI are then scripted and tested. While the automated tests are more exhaustive, the user stories allow us to focus on frequently used paths, that are more relevant to the end user. (Appendix K)

7 Regression Testing

Since our current test suite is tested against the latest version of our prototype, regression testing is not necessary. However, we do envision significant expansion of our code base in the coming months. Our plan is to migrate our Mocha test suite to Vue.js [8], to take advantage of GitLab's Continuous Integration environment. In essence, GitLab CI will run our designated tests during every commit. Since the scope of our project is quite small, we believe the cost of re-testing all of our functions during every commit would not be prohibitively high, and would be a great way to alleviate revision risk.

Appendix

A

Task Log

A.1 Ao Shen: Back-end, Tester, Document Editor

01/12/2017 (60min): Designated as Document Editor, will initiate report one and build logbook
01/15/2017 (60min): Report One initiated, logbook built, took notes in group meeting
01/22/2017 (120min): Completed Report One
01/29/2017 (60min): Learned back-end JS and Mongo from Cao, will write test code for UI
02/05/2017 (60min): Learned front-end JS from Bo, completed testing the UI
02/21/2017 (120min): Will complete 50pct of Report Two, write unit tests
02/29/2017 (120min): Completed 50pct of Report Two, wrote 8 unit tests
03/05/2017 (240min): Complete Report Two

A.2 Yuxiang: Front-end, Report Contributor

01/17/2017 (60min) Complete the layer design with Jiangbo
01/25/2017 (60min) Learned the vis.js library online
02/05/2017 (120min) Drew the first version of sphere graph node
02/19/2017 (120min) Complete the redraw function of the tree graph
02/26/2017 (120min) Complete the dynamic effects in displaying all the indexes
02/28/2017 (120min) Complete the switch between sphere graph and tree graph node
03/02/2017 (120min) Report two and start to optimize the whole front-end next week

A.3 Ruikun: Front-end, Back-end, Report Contributor

01/17/2017 (60min) Made first version of front-end webpage with AO and Dongxiao. Next will add some dynamic interactions and send some requests to server

1/25/2017(120min) Completed first version of our main server and it's already can respond to user's requests. Next will think of how to deal with requests more efficiently

2/10(120min) Deployed the MongodDb in Azure and learned how to do CRUD manipulation. Next will deploy MongoDb in our Express framework

2/15(60min) Used mongoose driver store IOTA transaction data into MongoDb and update the data. Next will try to calculate statistics for IOTA

2/25(120min) tried to calculate statistics for IOTA. Next will think of using graph database instead of document database

A.4 Jiangbo: Front-end, Report Contributor

01/17/2017 (60min) Complete the layer design with Yuxiang
01/23/2017 (60min) Learned the vis.js library online
02/12/2017 (120min) Drew the first version of tree graph node
02/19/2017 (120min) Complete the redraw function of both the sphere graph node
02/26/2017 (120min) Found the bugs that causes the failure of the toggle switch
03/02/2017 (120min) Report two and start to complete art design of the website next week

A.5 Xiao: Back-end, Report Contributor

12/25/2017 (60min) Learned the fundamentals of IOTA online
01/16/2018 (180min) Completed Requirement Document version1.0
01/22/2018 (60min) Report One
03/05/2018 (10min) Prepare for implementation of an intermediate neo4j database this week

A.6 Dongxiao: Back-end, Tester, Report Contributor

01/13/2018 (60min): Learned web development skills such as markup, attributes and url of HTML and learned the other part next week

01/14/2018 (60min): Learned web development skills such as HTTP protocol, request and response and started to develop front-end next week

01/17/2017 (60min): Participated in the first version of front-end page and learned server and looked for suitable database next week

01/23/2017(120min): Learned server and MongoDB development and improved the performance and fixed bugs next week

02/10(120min): Tested the server and browser and tested the other parts next week

02/15(120min): Tested MongpDB connection and try to create coverage report

02/25(120min): Report Two and change the started to learn Neo4j Database next week

B

Old Schedule

Deadline	Description	Progress
Jan 15	Gathering Requirements	Completed
Jan 22	Website (See Appendix Figure 2)	Completed
Jan 29	Report One	Completed
Feb 5	Queuing the API using JavaScript/JQuery	In Progress
Feb 12	Choosing and Implementing a Visualization Library	Not Started
Feb 19	Finalizing the Visualization Library	Not Started
Feb 26	Building the back-end database	Not Started
Mar 5	Report Two	Not Started
Mar 12	Finalizing the database	Not Started
May 16	Extensions, Final Report, Presentation Preparation	Not Started

Table 3: Adopted concrete and timed deliverables to combat slippage

C

New Schedule

Deadline	Description	Progress
Jan 15	Gathering Requirements	Completed
Jan 22	Website (See Appendix Figure 2)	Completed
Jan 29	Report One	Completed
Feb 5	Queuing the API using JavaScript/JQuery	Completed
Feb 12	Choosing and Implementing a Visualization Library	Completed
Feb 19	Implementing the back-end database	Completed
Feb 26	Prototype Testing	Completed
Mar 5	Report Two	Completed
Mar 26	IOTA Block Extraction	In Progress
April 9	Statistics Computation	In Progress
May 16	Final Report	Not Started
May 21	Presentation	Not Started

Table 4: Revised timetable

D

Unit Test Error

```
1×  function create_father(Collection, tips, results, callback) {
3×    let hashes = [];
3×    for (let i = 0; i < tips.length; i++) {
7×      hashes.push(tips[i]["hash"]);
      }
3×    iota.api.getLatestInclusion(hashes, function(error, values) {
3×      I if (error) console.log(error);
3×      E if (values) {
3×        for (let i = 0; i < values.length; i++) {
6×          if (values[i] === true) {
1×            results = tools.deleteDuplicates(results);
1×            for (let j = 0; j < results.length; j++) {}
1×            Collection.remove({})
                .then(() => {
                  setTimeout(function() {
                    Collection.create(results, function(error, docs) {
                      if (error) console.log(error);
                      else {
                        console.log("update finish");
                        callback();
                      }
                    });
                  }, 1000);

                });

1×            return;
            }
          }
```

Figure 2: Unit Test Error

E Models and Computation

all files database_server/models/

100% Statements `20/20` **100%** Branches `0/0` **100%** Functions `0/0` **100%** Lines `20/20`

File		Statements		Branches		Functions		Lines	
Stat.js		100%	5/5	100%	0/0	100%	0/0	100%	5/5
iotaes.js		100%	5/5	100%	0/0	100%	0/0	100%	5/5
statistics.js		100%	5/5	100%	0/0	100%	0/0	100%	5/5
track.js		100%	5/5	100%	0/0	100%	0/0	100%	5/5

Figure 3: Unit Test Coverage Summary for models and computations

F Functions

all files database_server/otherjs/

88.41% Statements `61/69` **57.14%** Branches `8/14` **80%** Functions `8/10` **92.42%** Lines `61/66`

File		Statements		Branches		Functions		Lines	
get_tips.js		100%	3/3	100%	0/0	100%	0/0	100%	3/3
query_interval.js		83.67%	41/49	50%	6/12	75%	6/8	89.13%	41/46
tools.js		100%	17/17	100%	2/2	100%	2/2	100%	17/17

Figure 4: Unit Test Coverage Summary for functions

G

Azure 1000

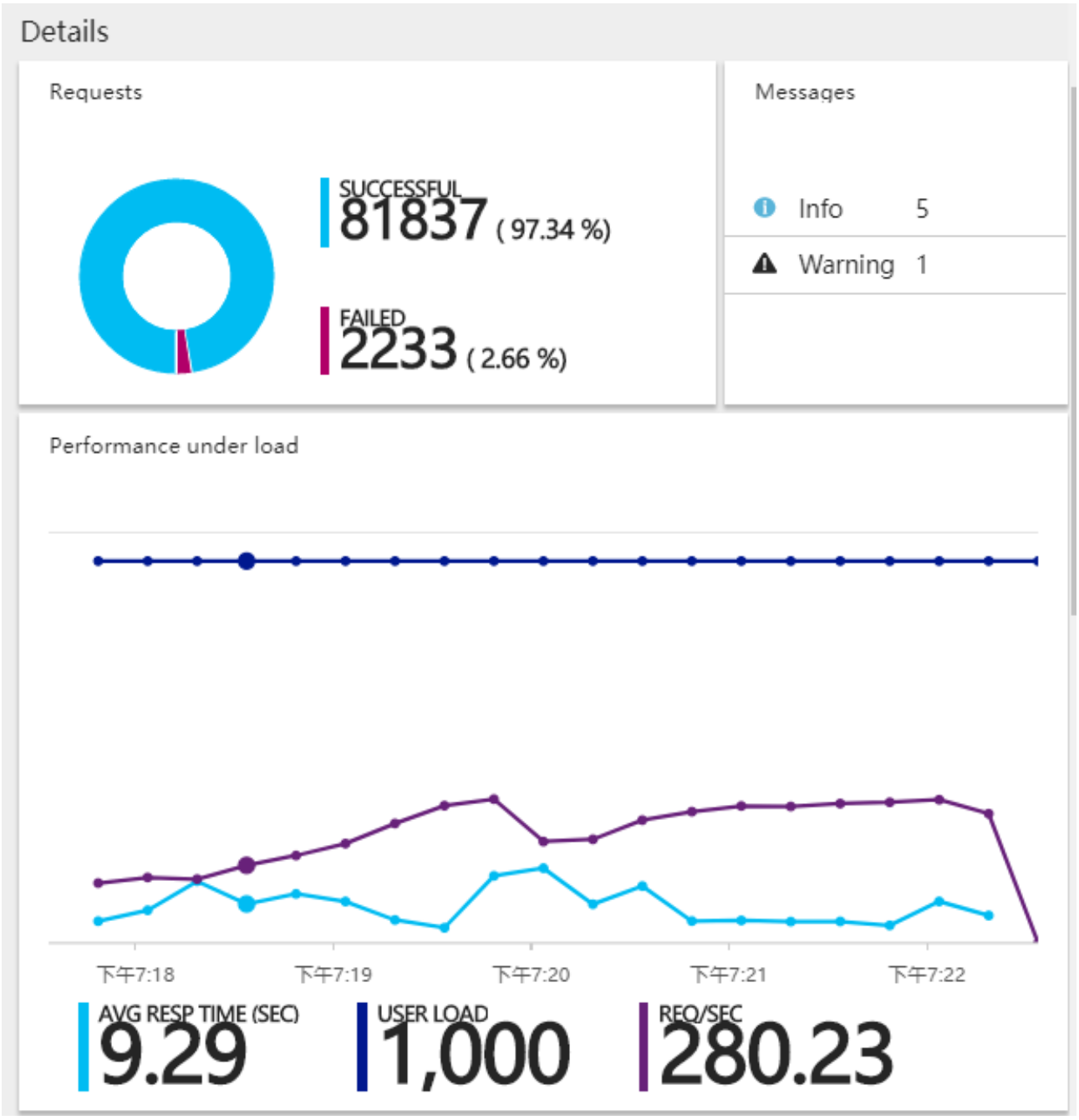

Figure 5: Azure Performance Test Results with 1000 users

H

Azure 2000

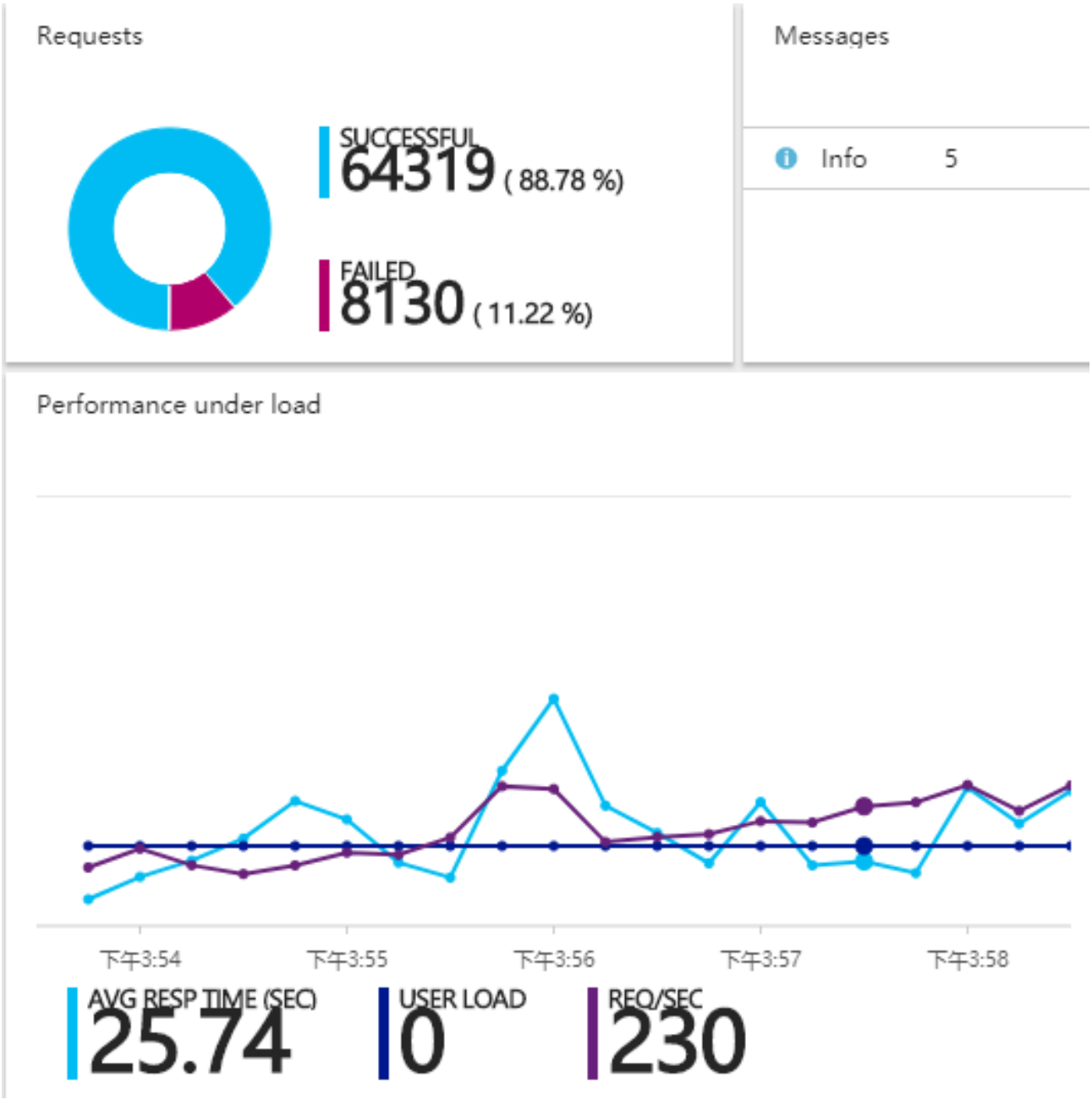

Figure 6: Azure Performance Test Results with 2000 users

I
Azure Error

Request Failures

REQUEST	TYPE	SUBTYPE	COUNT	LAST MESSAGE
-	Exception	LoadTestErrorL...	2	More than 100...
http://iotaserv....	HttpError	502 - BadGate...	151	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	141	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	137	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	7	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	150	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	7	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	147	502 - BadGate...
Request1	HttpError	502 - BadGate...	81	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	156	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	5	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	8	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	5	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	2	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	1	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	1	502 - BadGate...
http://iotaserv....	HttpError	502 - BadGate...	1	502 - BadGate...

Figure 7: Azure Performance Test Results with 2000 users: Failures Details

J

Test Script

05/03/2018 Script1519756363940.groovy.txt

```
import static com.kms.katalon.core.checkpoint.CheckpointFactory.findCheckpoint
import static com.kms.katalon.core.testcase.TestCaseFactory.findTestCase
import static com.kms.katalon.core.testdata.TestDataFactory.findTestData
import static com.kms.katalon.core.testobject.ObjectRepository.findTestObject
import com.kms.katalon.core.checkpoint.Checkpoint as Checkpoint
import com.kms.katalon.core.checkpoint.CheckpointFactory as CheckpointFactory
import com.kms.katalon.core.mobile.keyword.MobileBuiltInKeywords as MobileBuiltInKeywords
import com.kms.katalon.core.mobile.keyword.MobileBuiltInKeywords as Mobile
import com.kms.katalon.core.model.FailureHandling as FailureHandling
import com.kms.katalon.core.testcase.TestCase as TestCase
import com.kms.katalon.core.testcase.TestCaseFactory as TestCaseFactory
import com.kms.katalon.core.testdata.TestData as TestData
import com.kms.katalon.core.testdata.TestDataFactory as TestDataFactory
import com.kms.katalon.core.testobject.ObjectRepository as ObjectRepository
import com.kms.katalon.core.testobject.TestObject as TestObject
import com.kms.katalon.core.webservice.keyword.WSBuiltInKeywords as WSBuiltInKeywords
import com.kms.katalon.core.webservice.keyword.WSBuiltInKeywords as WS
import com.kms.katalon.core.webui.keyword.WebUiBuiltInKeywords as WebUiBuiltInKeywords
import com.kms.katalon.core.webui.keyword.WebUiBuiltInKeywords as WebUI
import internal.GlobalVariable as GlobalVariable
import org.openqa.selenium.Keys as Keys

WebUI.openBrowser('')

WebUI.navigateToUrl('http://51.140.113.215:3000/')
WebUI.click(findTestObject('Page_IOTA Node Visualizer/img'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/a_Map'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/a_About'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/a_Contact Us'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/a_Map'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/span_handle'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/div_limit to 4k'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/button__simple-switch-track'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/button__simple-switch-track_1'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/html_IOTA Node Visualizer'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/html_IOTA Node Visualizer'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/th_Hash'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/td_dongxiao huang'))
WebUI.doubleClick(findTestObject('Page_IOTA Node Visualizer/canvas'))
WebUI.doubleClick(findTestObject('Page_IOTA Node Visualizer/div_Total Tips      10027'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/html_IOTA Node Visualizer'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/body_Map'))
WebUI.click(findTestObject('Page_IOTA Node Visualizer/html_IOTA Node Visualizer'))
WebUI.closeBrowser()
```

file:///homes/as5017/Desktop/Script1519756363940.groovy.txt 1/1

Figure 8: Automated Test Script

K

User Story

1

Information

ID	Test Suite/1		
Description			
Start	2018-02-27 18:34:27	**End**	2018-02-27 18:34:35
Elapsed	8.044s		
Status	PASSED		

Details

#	Description	Elapsed	Status
1	openBrowser Browser is opened with url: ''	3.535s	PASSED
2	navigateToUrl Navigate to 'http://51.140.113.215:3000/' successfully	1.642s	PASSED
3	click Object: 'Object Repository/Page_IOTA Node Visualizer/img'	0.221s	PASSED
4	hover Object: 'Object Repository/Page_IOTA Node Visualizer/node' displays tip	0.711s	PASSED
5	drag Object: 'Object Repository/Page_IOTA Node Visualizer/block' is dragged	0.294s	PASSED
6	highlight Object: 'Object Repository/Page_IOTA Node Visualizer/table' increases	0.317s	PASSED
7	display Object: 'Object Repository/Page_IOTA Node Visualizer/stat' displays +2	0.258s	WARNING

Figure 9: User Story test results

References

[1] Almende B.V. *Node Graph Drawing Library*. 2018. URL: http://visjs.org/.

[2] IOTA Foundation. *IOTA API*. 2018. URL: https://iota.readme.io/v1.2.0/reference.

[3] Team IOTA. *Report One*. 2018. URL: https://www.overleaf.com/read/fpqkmjzfnpkd.

[4] Microsoft. *Performance Test*. 2018. URL: https://docs.microsoft.com/en-us/vsts/load-test/performance-reports.

[5] MochaJS. *JavaScript Tester*. 2018. URL: https://mochajs.org/.

[6] SigmaJS. *Graph Drawing Library*. 2018. URL: http://sigmajs.org/.

[7] SinonJS. *Stub Generator*. 2018. URL: http://sinonjs.org/.

[8] Evan You. *Regression Testing*. 2018. URL: https://vuejs.org/.