SE 3XA3: Test Report
Synergy Inventory Management System (SIMS)

Team #33, 'Sick Ideas'
Nathan Coit – 400022342
Lucas Shanks – 400029943
Cameron Van Ravens – 400020215

December 4, 2017

Contents

List of Tables

List of Figures

1 Introduction

This document provides the results of testing for the Synergy Inventory Management System. Functional requirement testing, Nonfunctional requirements testing, unit tests and system testing evaluations are all covered in this document. Traceability between testing and Requirements and Modules is given in the final sections of this document. For a Gantt Chart outlining the testing schedule please reference this ganttproject file.

2 Nonfunctional Requirements Evaluation

2.1 Usability

For assessing usability, a simple usability survey was used among a random set of 7 various users. The users were asked to visit the URL provided and complete a simple set of tasks unassisted: Create an account, add some products, categories, brands, warehouses, and to add another user to their company. The users then had to fill out a short questionnaire with some quantitative and qualitative response. The quantitive responses we received are shown in Figure 1:

As we can see from the test results, the overall usability was satisfactory. This resulted in considering the usability testing a pass, although from the lower scoring test subjects we received some valuable suggestions on improvements that could be made to the user interface in order to improve the usability.

2.2 Performance

To evaluate performance of the site, we used stress testing under a load of concurrent users. To provide a varying load of concurrent users, we used a tool called LoadImpact, which runs varying amounts of connections from

Table 1: **Revision History**

Date	**Version**	**Notes**
12/02/2017	1.0	Initial Revision
12/04/2017	1.1	Final Revision

User	**Appeal**	**Ease Of Use**	**Responsiveness**
1	7	8	10
2	8	8	9
3	8	7	10
4	10	8	7
5	9	10	9
6	8	9	8
7	8	7	9
Overall	**8.3**	**8.1**	**8.9**

Figure 1: Usability Survey Results

around the world to the website and measures the response time. The specifications of the server the site is hosted on is given in Figure 2.

The overall response time of the LoadImpact tests averaged under 5 seconds for up to 25 concurrent connections, as shown in the graph Figure 3. This met our performance requirement of having a response time of no greater than 5 seconds, and is therefore this test is considered a pass.

Location	**Hardware**	**Server OS**
Toronto, CAN.	1 CPU Core 512MB RAM 20GB SSD	CentOS 7.4

Figure 2: Server used for Hosting

2.3 Robustness

The robustness of the application with regards to malformed or malicious input were tested in the automatic unit tests of the backend, as well as manually during the system tests. The requests made to the backend were given inputs that were either not expected, of the wrong type, or formed in a way set to break the code or determine a security flaw. However, upon providing these inputs the system responded correctly by rejecting the inputs and returning the proper exceptions back to the user/system. This same

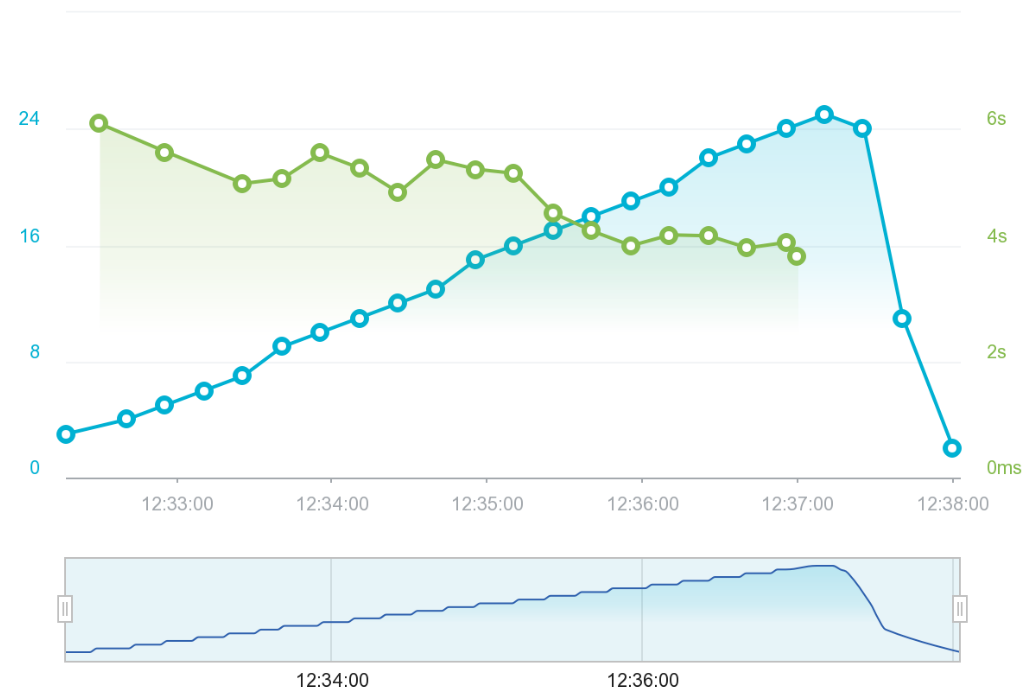

Figure 3: Results of the LoadImpact Tests

behaviour was testing and observed across all of the endpoints, therefore resulting in these tests being considered a pass.

The robustness of the application in regards to performance and handling the stress of concurrent loads was evaluated in the Performance section.

3 Comparison to Existing Implementation

In the existing implementation of this project there are no test cases or unit tests provided as it was not setup to include any automated testing, and therefore any testing was done manually. In this implementation, there are test cases provided as well as some support for automated unit testing.

4 Unit Testing

The unit testing done on this project was done through automatic blackbox testing done on the backend. Each endpoint was tested in situations ranging from normal, expected use, to malformed input data, to malicious input (i.e. SQL injection). This was done for each endpoint, covering usage of required and optional parameters. As of Rev1 of the project all of the unit test cases pass.

5 System Testing

The system testing done for the Synergy Inventory Management System was done manually to ensure that the site behaved and reacted to user input as expected. Some of the system tests that were executed are as follows:

Test ID: SYS1
Initial State: User is not logged into the site.
Input: A username and password that have been previously registered.
Expected Output: The user is logged in and brought to the welcome page.
Result: PASS

Test ID: SYS2
Initial State: User is logged in and at the welcome page.
Input: Visits the inventory page and creates a new product with valid fields.
Expected Output: New product is added to the list of displayed products persistently.
Result: PASS

Test ID: SYS3
Initial State: User is logged in and at the users page.
Input: Creates a new user with the same email theirs.
Expected Output: System should give an exception as the email is already in use.
Result: PASS

Test ID: SYS4
Initial State: User is logged out.

Input: Attempts to access the categories page by URL.
Expected Output: User is redirected to the login page.
Result: PASS

Test ID: SYS5
Initial State: User is logged in as company owner.
Input: Delete the company from the company page.
Expected Output: After several warnings, the company and it's data is deleted and user is redirected to login page.
Result: PASS

Test ID: SYS6
Initial State: User is logged in as company owner, on the account page.
Input: Requests to change account type to user.
Expected Output: System should not change the owner's account type.
Result: PASS

Test ID: SYS7
Initial State: User is logged in and at the inventory page.
Input: Uploads a properly-formatted CSV using the upload routine.
Expected Output: The products should be added to the inventory and appear on page.
Result: PASS

6 Changes Due to Testing

A summary of the changes made due to testing is given in Table 4.

7 Automated Testing

As the Synergy Inventory Management System platform is web-based, Automated Testing in general was not feasible for the entire project. This is due to the fact that user interface elements are very difficult and not practical to test using automated testing methods. Therefore, the system tests and integration tests had to be carried out manually by the testing team.

Figure 4: Changes made due to Testing

Changes
Improper data validation implementation on creating a user fixed.
Improperly formatted CSV causing server to crash was fixed.
Frontend not receiving certain data due to miscommunication fixed.
Certain endpoints URI paths being interpreted incorrectly fixed.
Registration failing on non-unique company name fixed.
Changing password sets password to old password.

However, for unit testing done on the RESTful API backend, automated testing was achieved using the Postman REST test runner. This allowed for all of the endpoints of the backend platform to be tested against various blackbox test cases, with the output being compared to the expected output and reported to the Postman test runner. A description of the Unit tests carried out by this automated testing method are described in the section on Unit Testing.

8 Trace to Requirements

A trace between testing and the Requirements is given in Table 5.

Figure 5: Trace between Testing and Requirements

Req't	**Test Case ID**
REQ1	00 - 05
REQ2	10 - 11
REQ3	20 - 22
REQ4	30 - 31
REQ5	40 - 41
REQ6	50
REQ7	60 - 62
REQ8	70 - 71
REQ9	80 - 84

9 Trace to Modules

A trace between testing and the Modules is given in Table 6.

Figure 6: Trace between Testing and Modules

Test Case	**Module ID**
SYS1	mH2, mH4, mH5, mH6
SYS2	mH2, mH4, mH5, mH7, mH8
SYS3	mH2, mH4, mH5
SYS4	mH6
SYS5	mH2, mH4, mH5
SYS6	mH2, mH4
SYS7	mH1, mH3, mH4, mH5, mH7, mH8

10 Code Coverage Metrics

For a quantified code coverage of the tests that were run, it would be about 80% to 90% coverage. The code coverage in this case must be estimated, as absolutely calculating the code coverage using a tool is not feasible for this project. This is because the majority of testing was done via manual testing methods, since the system is controlled by a user interface. Although the backend had automated Unit testing, these tests were done blackbox using an external tool, which cannot perform or judge the analysis of code coverage of the tests.