Smart Waiter Test Plan

Meraj Patel (patelmu2)
Pavneet Jauhal (jauhalps)
Shan Perera (pererali)

April 18, 2016

[Missing Student Numbers. —DS]

Contents

List of Tables

Revision History

Date	**Comments**
October 30, 2015	first draft.
February 1, 2016	Added usability questions
April 15, 2016	Addressed Dan's comments in regards to spelling and grammar. Modified automated testing
April 16, 2016	Addressed more Dan's comments in regards to spelling and grammar. Fixed tables. Added item selection and customization

1 General Information

The following section provides an overview of the test plan for Smart-Waiter. This section explains the purpose of this document, the scope of the system, and an overview of the following sections

1.1 Purpose

The purpose of this document is to describe various test cases and procedures to evaluate functionality of Smart Waiter. This document is indented to be used as a reference for all future testing.

1.2 Scope

This test plan is used to evaluate Smart Waiter functionality. Various plans are described in detail to test expected functional and non function requirements.

1.3 Overview of Document

The following sections provide more detail about types of test that will be used. Information about the testing process is provided, and the software specifications that were discussed in the SRS document are stated. Testing processes is split up between system test for POC and the final demonstration.

[Acronyms should be defined somewhere. —DS]

2 Plan

This section provides a description of the software that is being tested, the team that will perform the testing, the milestones for the testing phase, and the budget allocated to the testing.

2.1 Software Description

The software being tested is for Smart Waiter Solutions. The project aims to provide a solution that will allow users to order and pay through a mobile application at restaurants.

2.2 Test Team

The team that will execute the test cases, write and review Smart Waiter consists of:

- Pavneet Jauhal
- Shan Perera
- Meraj Patel
- Dr. Rong Zheng

2.3 Milestones

2.3.1 Location

The location where the testing will be performed is Hamilton Ontario. Specifically, we will be performing the testing in McMaster University.

2.3.2 Dates and Deadlines

Test Case Creation: Creation of POC test cases begin October 19, 2015 and is expected to end November 16, 2015. Proceeding, test cases for final demonstration will be created and is expected to be complete by February 1, 2016.

Test Case Implementation: Implementing test cases for POC system test is scheduled to begin November 15, 2015 is expected to be complete in approximately one week.

Implementing test cases for Final Demonstration system test is scheduled to begin February 1, 2016, and will be completed in approximately three weeks.

2.4 Budget

N/A

3 Software Specification

This section provides; the functional requirements the software is expected to complete, and the non-functional requirements the software is expected to exhibit.

3.1 Functional Requirements

- The product shall scan a bar code to access the restaurant menu from a database
- The product shall allow the user to register an account for complete access to payment services
- The product shall allow the user to report bugs
- The product will allow users to place their order
- The product will allow users to pay for their order using a credit card

3.2 Nonfunctional Requirements

Priority nonfunctional requirements are performance, ease of use, reliability, security and maintainability.

4 Evaluation

The following section presents methods and constraints used during the evaluation process, and is followed by extent of testing and test tools to be used.

4.1 Methods and Constraints

4.1.1 Methodology

Testing of Smart Waiter will consist of various types of tests explained in the next section. Manual testing will be conducted for the most part, some automation testing will be in place.

4.1.2 Extent of Testing

The extent of testing will be end to end for the final demonstration. For POC, specific system tests will be conducted.

4.1.3 Test Tools

To avoid regressions and catch bugs quickly two main test tools will be used. Namely the following;

- CodeReviewer —CodeReviewer will be used for code review process mandatory for all developers. This tool helps post code reviews and provides features such as annotations, diff viewers, easy communication and feedback. This tool will be used for static tests to produce high quality code with less defects

4.1.4 Testing Constraints

N/A

4.2 Types of Tests

4.2.1 Functional Testing

Functional testing is in place to uncover errors that occur in implementing requirements or design specifications. Concentration is on results rather then the internals functions of the program. This type of testing is very effective to evaluate functional requirements.

4.2.2 Structural Testing

Structural testing is in place to uncover errors during implementation of the application. Concentration is on evaluating structure of the application. This type of testing focuses on non functional requirements.

4.2.3 Unit Testing

Unit testing refers to providing specific input to a system and verifying it evaluates to expected output.This process can be done manually or automated.

4.2.4 Manual and Automatic Testing

Manual testing is done by people, while automatic is processed through scripts.

4.2.5 Static Testing

Static testing refers to testing techniques that simulate a dynamic environment. This does not involve program execution. Instead, a walk-through will be performed checking pre and post conditions evaluate to requirements specified. As well to make sure proper syntax is used thoroughly.This type of testing is crucial in design stage.
Static testing will be enforced through out the development process. A code review process will be put into place to strictly manage the quality of the code. Specifically, developers will have to sign off on the quality of the code before it gets checked into the production repo.

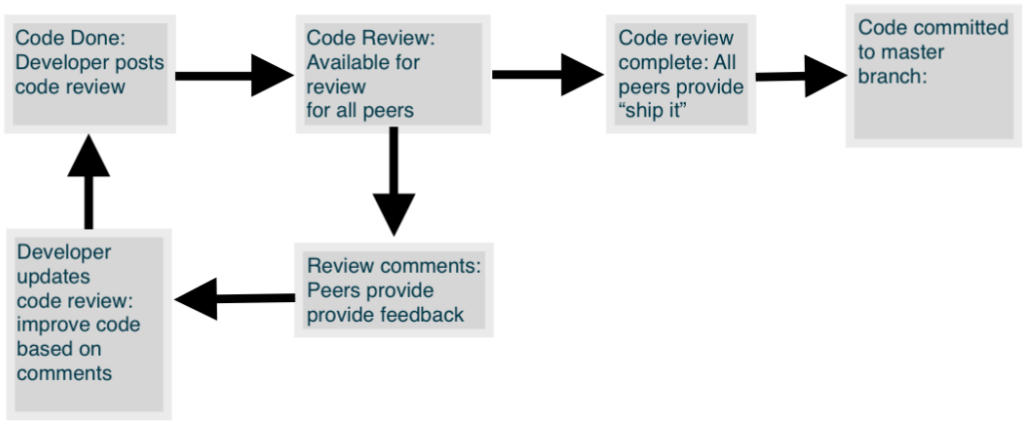

Figure 1: Code Review Process

The reviewers will assess the code quality and critique the code in following criteria;

- General Unit Testing performed. Did it pass?
- Comment and Coding Conventions followed consistently?

- Error Handling done correctly?
- Resource Leaks within the code?
- Is the code thread safe?
- Is the code up to standard with performance?
- Does code provide correct functionality?
- Is the code secure?

The reviewers will assess the code and mark issues found with severity. If any of the high severity issues found, then developer must fix code. For medium severity issues, developer can provide reasoning and try to convince reviewers. Low severity issues can possibly be dropped if they are nits. The reviewers will mark severity as follows;

- Naming Conventions and Coding style considered low severity
- Control flow and Logical issues considered medium or high severity
- Redundant Code considered medium or high severity
- Performance Issues considered high severity
- Security Issues considered high severity
- Scalability Issues considered medium or high severity
- Functional Issues considered high severity
- Error Handling considered high severity
- Reusability considered medium severity

4.3 Dynamic Testing

On the contrary, dynamic testing refers to executing the program while running test cases to view expected behaviour. This is done to find and fix defects in the program. This will be performed after implementation phase. All of the tests performed in the following sections are dynamic. Specifically, different techniques are used to perform this testing such as automated and manual test cases. However, due to the nature of the application all other testing will be dynamic since the values are all dynamic.

5 POC System Test Description

5.1 Database Testing

Database testing is exclusively performed from structural testing point of view. The goal of this component of testing is to verify that the database queries always return valid and correct data, under all circumstances. Specifically, a separate database will be created for testing purposes, because any failed test case should not affect the state of the original database.

5.1.1 Test Factors

For database testing, the following test factors will be considered;

- Correctness
- Performance
- Security
- Reliability

5.1.2 Correctness

In this section of database testing, tester must verify that the application receives complete and correct data from the couchbase server. Automated unit tests will be written using Bash scripts to perform the following tests.

5.1.3 Database Correctness Automated Tests

Test Name	Initial State	Input	Output	Post State
Test 1 Correct retrieval of database	Empty Database	Add menu items to the database	Query menu items to verify that all data matches the input	Clear Database
Test 2 Correct retrieval of updated Database	Empty Database	1) Add menu items to database. 2) Query database 3) Update the database	Query menu items to verify that updated data is retrieved from the server	Clear Database
Test 3 Retrieve Empty Database	Empty Database	None	Retrieve items from database. Verify empty strings	None
Test 4 Support for backwards compatibility	Empty Database	Add menu items to the database, with attributes that the application does not support. (i.e review comments)	Query the database. The application should retrieve complete data matching the input.	Clear Database

Table 1: Database Correctness

5.1.4 Performance

In this section of database testing, tester must verify that the data query is performed in a reasonable amount of time. The application should have a maximum loading period of 5 seconds to complete data queries. Moreover, considering the customer base and size of database our application, none of the following test cases should take more than 5 seconds.

Test Case	Initial State	Input	Output
Test database query with 10 menu items	Empty database	Add 10 items to database	getRestarauntByBarcode() should return all menu in less than 1.5 seconds
Test database query with 50 menu items	Empty database	Add 50 items to database	getRestarauntByBarcode () should return all menu in less than 1.5 seconds
Test database query with 100 menu items	Empty database	Add 100 items to database	getRestarauntByBarcode () should return all menu in less than 1.5 seconds

Table 2: Performance Automated Tests

5.1.5 Database Performance Automated Tests

5.1.6 Security

In this section of database query testing, tester must verify that database allows correct access control. Database penetration testing is not required because we will leverage couchbase security implementation, testing and certifications.

5.1.7 Database Security Automated Tests

Test Name	Initial State	Input	Output	Post state
Test 1 – Verify incorrect access to menu catalog Database is denied.	Empty database	1) Add menu items to Database	Query for menu item with the incorrect client permissions (client key). This request should be denied	Clear database
Test 2 – Verify that correct access to menu catalog database is accepted	Empty database	1) Add menu items to Database	Query for menu item with the correct client permissions (client key). This request should be allowed	Clear database

Table 3: Security Test

5.1.8 Reliability

In this section of database testing, we will test the robustness and reliability of the database. This type of testing is essential the health of the database system. There should be no failures or downtimes on the backend side.

5.1.9 Database Reliability Automated Tests

Test Case	Initial State	Input	Output
Dependability Test	Database consists of 30 menu items	Script for HTTP GET request for menu every 2 minutes	JSON structure of menu
Verify database does not malfunction with stress	Database consists of 30 menu items	Query one menu item concurrently 30 times	JSON structure of menu

Table 4: Reliability Automated Tests

6 Final Demonstration System Test Description

6.1 Barcode Scanning

Smart-Waiter needs to ensure users are able to scan a barcode with minimal attempts. The number of expected attempts will be presented in the final SRS document.

6.1.1 Test Type

- Manual
- Functional test
- Unit test

6.1.2 Nonfunctional Test Factors

- Performance
- Correctness
- Ease of use

6.1.3 Methods of testing

Dynamic testing is used to ensure correctness and data integrity, and to observe the application behaviour when given incorrect information.

6.1.4 Test Cases

Functional Unit Test

Summary
Manual black box tests will be performed to assess barcode scanning. This test will be in the form of unit testing. Various test cases described below will be conducted. This is effective as it replicates real world usage and will provide a census of expected number of successful attempts to evaluate non-functional test factors.

Test Case	Initial State	Input	Output
Scan working barcode	Barcode scanning page	Eligible barcode	Restaurant menu
Scan corrupt barcode	Barcode scanning page	Corrupt barcode	Barcode scanning page Message - "Invalid barcode. Please try again"
Scan random picture	Barcode scanning page	Random picture	Barcode scanning page Message - "Invalid barcode. Please try again"
Scan corrupt barcode – third attempt	Barcode scanning page	Corrupt barcode	Barcode scanning page Message - "Invalid barcode. Please contact host"
Scan deprecated barcode	Barcode scanning page	Deprecated barcode	Barcode scanning page Message - "Invalid barcode. Please try again"
Scan deprecated barcode – third attempt	Barcode scanning page	Deprecated barcode	Barcode scanning page Message - "Invalid barcode. Please contact host"

Table 5: Barcode Scanning Test

6.2 Account Login

Smart-Waiter must use accounts to keep track of a user's personal information. The account module has to provide a secure login service.

6.2.1 Test Type

- Manual
- functional dynamic test

6.2.2 Nonfunctional Test Factors

- Correctness
- data integrity

6.2.3 Methods of testing

Dynamic testing is used to ensure correctness and data integrity, and to observe the application behaviour when given incorrect information.

6.2.4 Test Cases

Dynamic Testing

Summary
Manual dynamic tests will be performed to evaluate the account creation module. This test will be in the form of structural testing. The tests being performed on the module are presented below.

Test Case	Initial State	Input	Output	Result
6.1.1	Create account menu, empty	All fields empty / left blank	Message reading: "Password too short"	PASS
6.1.2	Create account menu, empty	Password, first name, last name, home address, postal code, phone number	Barcode scanner menu	PASS
6.1.3	Create account menu, empty	Password, first name, last name, home address, phone number, incorrect postal code	Message reading: "Postal Code Invalid"	PASS
6.1.4	Create account menu, empty	Phone number consisting of 6 digits	Message reading: "Phone Number Invalid"	PASS
6.1.5	Create account menu, empty	Empty First Name Field	Message reading: "First Name Invalid"	PASS
6.1.6	Login menu, empty	Correct password	Barcode Scanner Menu	PASS
6.1.7	Login menu, empty	Empty password field	Message reading: "Invalid Password"	PASS
6.1.8	Login menu, empty	Incorrect password	Message reading: "Invalid Password"	PASS

[Test 1 should state it uses a username that is already in use. Why are there no tests for invalid Google/Facebook accounts? —DS]

6.3 Item Selection and Customization

Smart-Waiter needs to ensure a user is able to select, customize and add item to cart.

6.3.1 Test Type

- Manual
- functional dynamic test

6.3.2 Functional Test Factors

- The product shall allow the user to customize and add items to cart

6.3.3 Methods of testing

Item selection and customization is performed using dynamic unit testing. The goal of this component is to verify if a user can select, customize and add multiple menu items to cart. Also, testing is performed to check if a user is able to re-modify an already selected item. Testing is performed from a functional view to evaluate functional requirements stated in SRS.

6.3.4 Test Cases

Dynamic Unit Test

Summary
Manual black box tests will be performed to assess item customization and for the ability to add to cart. This test will be in the form of unit testing. Various test cases described below will be conducted. This is effective as it replicates real world usage and will provide a census of expected number of successful attempts to evaluate functional test factors.

Test Case	Initial State	Input	Output
Select menu item	Menu items page	User clicks item	Item customization page
Add item to cart	Special instructions page	User clicks checkmark	Message - "Added item to cart" Item visible on cart page
Add toppings	Toppings page	User clicks on toppings checkboxes. Adds item to cart.	Item name along with item toppings displayed on cart page
Add side order	Side order page	User clicks on side order name. Adds item to cart	Item name along with side order displayed on cart page
Modify item toppings	Cart page	User clicks customize icon. Proceed to click on toppings checkboxes	Item name along with modified item toppings displayed on cart page
Modify side order	Cart page	User clicks customize icon. Proceed to click on side order name.	Item name along with modified side order displayed on cart page
Delete item	Cart page	User clicks delete icon	Item is removed off cart page

6.4 Order Transaction

Smart-Waiter needs to ensure that a user can send in their order, and pay for their order easily and securely. Order transaction will be vigorously tested to ensure complete customer satisfaction.

6.4.1 Test Type

- Manual
- functional dynamic test

6.4.2 Nonfunctional Test Factors

- Correctness
- Reliability
- Data integrity
- Data security
- Ease of use

6.4.3 Methods of testing

Dynamic testing is used to ensure validity, record the number of successful tests given a sample.

6.4.4 Test Cases

Dynamic Testing

Summary
Manual dynamic tests will be performed to evaluate the order transaction module. This test will be in the form of structural testing. It will test vigorously for all possible inputs of credit card information and account details.

The tests being performed on the module are presented below.

Test Case	Initial State	Input	Output
Proceed to payment with items in cart	Cart page. Items in cart	Click checkmark	Advances to confirm order page
Proceed to payment with no items in cart	Cart page. No items in cart	Click checkmark	Message – "Cart is empty. Please add items before checkout"
Proceed to payment page with items in cart	Confirm order page	Click checkmark	Advances to credit card information page
Input valid credit information	Credit card information page	Input valid credit card number, name, expiration date, CVC	Message – "order sent" Return to account sign in page
Input invalid credit card information	Credit card information page	Input invalid credit number or name or expiration date or CVC	Message – "Invalid card details"

6.5 Usability Testing

The goal of usability testing is to verify if a user can successfully use all functionalities of the application from start to finish in a timely manner. This type of testing aids in evaluating if functional requirements are met. As well it provides scope for evaluating non functional requirements.

6.5.1 Nonfunctional Test Factors

- Reliability
- Performance
- Correctness
- Ease of use

6.5.2 Methods of testing

Manual black box tests will be performed to assess usability of the application. This test will be in the form of unit testing.

Six participants will be selected to evaluate this test. At least three will have some experience with using android applications. The remaining will have little to no experience. Each participant will be asked to conduct a walk through of the application and complete set tasks. Afterwards a short questionnaire will be given to assess the impressions of the application.

6.5.3 Tasks conducted

Participants were given a list of tasks to complete including:

- Task 1: Create and login to account
- Task 2: Scan barcode to retrieve menu
- Task 3: Customize and add items to cart
- Task 4: View cart
- Task 5: Delete item
- Task 6: Modify item
- Task 7: Confirm and pay for order

6.5.4 Survey Questions

The survey consists of seven questions. Participants are asked to give each question a score between 1 to 5. 1 being strongly disagree and 5 being strongly agree.

1. I was able to complete the task quickly using the system
2. It was easy to learn how to use the system
3. I prefer using Smart-Waiter over traditional sense
4. The system has all functions and capabilities i expect it to have
5. The interface of the system was pleasant
6. Whenever I made a mistake in the system, I could recover easily and quickly
7. Overall I was happy with the system

7 Testing Schedule

Active testing is necessary to build Smart-Waiter into a successful application. Therefore, testing will be performed on numerous occasions throughout the development of this application. [When specifically? This should be in your schedule. —DS] Individual testing will be performed before any code is committed to the repository. [That is fine, however, there should be milestones where testing will be performed on a predetermined schedule. —DS] Furthermore, as newer features and functions are being added and the development of Smart-Waiter progresses, the testing team and developers will be changing the test cases accordingly. [This last bit is unnecessary, as your test plan will be updated as you update your test cases. —DS]
[Who will be testing what and when? —DS]

Test Document	Objective	Delivery Date
Test Plan – Revision 0	Set test cases, guidelines and objectives for testing	October 30th, 2015
Test Report – Revision 0	Report the progress of the application testing, and the results of the tests	March 24th, 2016
Test Plan – Final	Provide test cases for testing before the final release of Smart-Waiter	April 1st, 2016
Test Report – Final	Report the progress of the application testing, and the results of the tests	April 1st, 2016

[I did not mark all of your spelling and grammar mistakes. You should go through the document and find them on your own. There are also many more possibilities for test cases that you have not yet considered. —DS]