Smart Waiter Test Plan

Meraj Patel #1137491
Pavneet Jauhal #1149311
Shan Perera #1150394

October 30, 2015

Contents

List of Tables

1 General Information

The following section provides an overview of the test plan for Smart Waiter Solutions. 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 are met.

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.

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 is McMaster University.

2.3.2 Dates and Deadlines

Test Case Creation: The creation of test cases is scheduled to begin October 19th, 2015. The deadline for creation of test cases for POC November 16, 2015. The deadline for final demonstration is February 1st, 2016.

Test Case Implementation: Implementing test cases for POC system test is scheduled to begin November 15, 2015 The implementation period is expected to last approximately one week.

Implementing test cases for Final Demonstration system test is scheduled to begin February 1, 2016 The implementation period is expected to last 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's menu from a database
- The product shall allow the user to register an account for complete access to the products 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 first presents methods and constraints used during the evaluation process. This 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;

- JUnit —Junit will be used to build an automation test suite for the application. Specifically, JUnit is a testing framework used to implement unit testing in Java. This tool will easily integrate into the android studio build environment used for application. In addition, it will help developers leverage their Java knowledge used during application development.
- 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.

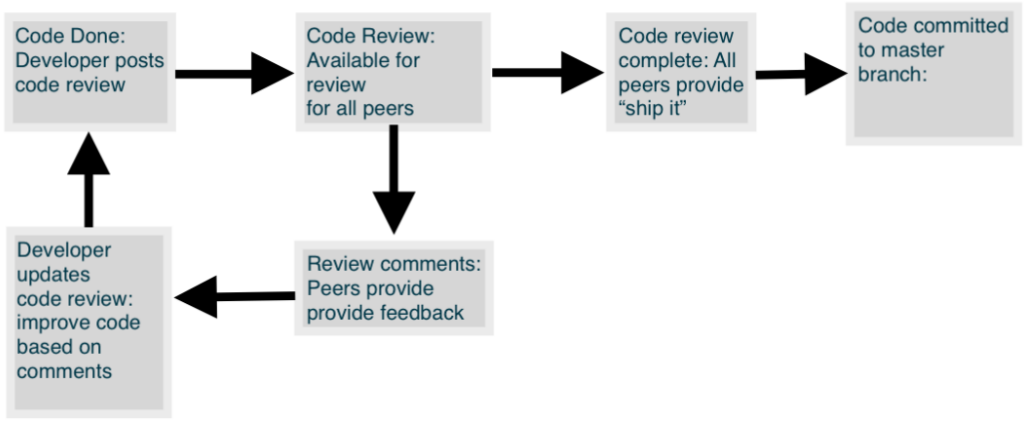
4.2.4 Manual and Automatic Testing

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-though 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. The tool used for code reviews be Codereviewer.

Below we have highlighted the 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 JUnit 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

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.

5.1.5 Database Performance Automated Tests

Test Name	Initial State	Input	Output	Post state
Test 1 – Test data base query with 10 menu items	Empty database	Add 10 menu items to Database	Query for 1 menu item should not take longer than 5 seconds	Clear database
Test 2 – Test data base query with 50 menu items	Empty database	Add 50 menu items to Database	Query for 1 menu item should not take longer than 5 seconds	Clear database
Test 3 – Test data base query with 100 menu items	Empty database	Add 100 menu items to Database	Query for 1 menu item should not take longer than 5 seconds	Clear database

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

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 Name	Initial State	Input	Output	Post state
Test 1 – Create a server program to test dependability	This test will run on the original database	A server program will run 24/7 and query the database periodically (every 10 minutes)	The Query should always return a menu item form the database	Publish report if query failed else do nothing.
Test 2 – Verify lockout during Menu Catalog updates	Empty database	1) Add menu items to Database 2) Create a write/block on server side to simulate update.	Query for menu item should be denied	Clear the database and remove block
Test 3 – Verify database does not malfunction with stress	Empty database	1) Add items to database 2) Launch multiple amounts of queries to retrieve items from database (consider multi threading) **This test will run on 3 different machines concurrently to simulate 100's of users	Query for the menu items should always return the requested data	Clear the database

6 Final Demonstration System Test Description

6.1 Barcode Scanning

Smart-Waiter needs to insure 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 performance.

Test Case	Initial State	Input	Output
5.1.1	Barcode Scanning Page – Empty	Eligible Barcode	Restaurant Menu
5.1.2	Barcode Scanning Page – Empty	Corrupted Barcode	Message prompt reading, "Please try again". After third attempt, prompt will read, "Please contact waiter" and will indicate an option to report bug
5.1.3	Barcode Scanning Page – Empty	Picture without barcode	Message prompt reading, "Please try again". After third attempt, prompt will read, "Please contact waiter" and will indicate an option to report bug

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
6.1.1	Create account menu, empty	Username, first name, last name, email, date of birth, address, credit card information	Message prompt reading: "The username you entered is already in use, please choose a different username."
6.1.2	Create account menu, empty	Username, first name, last name, email, date of birth, address, credit card information	My account menu
6.1.3	Create account menu, empty	Username, first_name, last name, email, date of birth, address, fake credit card	Message prompt reading "The credit card information you have provided is invalid, please enter a different credit card."
6.1.4	Create account menu, empty	Google account information	Add credit card menu
6.1.5	Create account menu, empty	Facebook account information	Add credit card menu

6.3 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.

6.3.1 Test Type

- Manual
- functional dynamic test

6.3.2 Nonfunctional Test Factors

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

6.3.3 Methods of testing

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

6.3.4 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
6.2.1	Restaurant menu module	n_3 from set of orders n_i	Message prompt reading: “This order does not follow the restaurant's menu policy, a waiter has been called to the table to assist with the ordering process”
6.2.2	Restaurant menu module	n_7 from set of orders n_i	Order summary menu
6.2.3	Payment confirmation menu	Valid credit card	Payment receipt menu
6.2.4	Payment confirmation menu	Expired credit card	Message prompt, reading: “This credit card is expired, please enter valid credit card information”
6.2.5	Payment confirmation menu	Fake credit card	Message prompt reading: “The information provided is invalid, please provide another credit card”
6.2.6	Payment confirmation menu	VISA debit card	Payment receipt menu
6.2.7	Payment confirmation menu	VISA gift card	Message prompt reading “VISA gift cards are not accepted as a valid form of payment with Smart-Waiter, sorry for any inconvenience”

6.4 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.

6.4.1 Nonfunctional Test Factors

- Reliability
- Performance

- Correctness
- Ease of use

6.4.2 Methods of testing

Dynamic testing is used to evaluate end to end system testing. This effectively evaluates functional non functional requirements entirely of the system.

6.4.3 Test Cases

Functional Unit Test

Summary
Manual black box tests will be performed to assess usability of the application. This test will be in the form of unit testing. Ten participants will be selected to evaluate this test. At least five 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. Number of errors made will be counted and recorded to further evaluate functional requirements and improve test cases. As well, qualitative feedback from participants will be taken in the end. For example, participants will be asked to comment on things they liked and things they didn't like. Also, they will be asked to provide a rating out of 10 in terms of experience; 1 being a poor experience and 10 being a wonderful experience. This feedback will help evaluate non function requirements; user interface appearance, learning requirements and accessibility.

7 Testing Schedule

Active testing is necessary to build Smart-Waiter into a successful app. Therefore, testing will be performed on numerous occasions throughout the development of this application. Individual testing will be performed before any code is committed to the repository. 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.

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