Restaurant Management Application

Project Testing and Acceptance Plan

Washington State Unviersity

Team One

Amrit Banga

Tucker Brisbois

Zijian Zhang

[07/05/2024]

TABLE OF CONTENTS

I. Introduction **4**

I.1. Project Overview 4
I.2. Test Objectives and Schedule 4
I.3. Scope 4

II. Testing Strategy **4**

III. Test Plans **4**

III.1. Unit Testing 4
III.2. Integration Testing 4
III.3. System Testing 4
III.3.1. Functional testing: 4
III.3.2. Performance testing: 4
III.3.3. User Acceptance Testing 5

IV. Environment Requirements **5**

V. Glossary **5**

VI. References **5**

Introduction

I.1. Project Overview

The software being tested will be the fully functional Restaurant Management Application. This version of the application will be fully functional minus UI finalization. The functions that will be tested will be the create account, log in system, 2 factor authentication, add business system, inventory management system.

I.2. Test Objectives and Schedule

In order to test these functions, a fully functional build of the application will be required. The testing will be done on the virtual iPhone tester which is provided by XCode when the application is running. Most of the testing will be pass or fail tests, with screenshots providing the evidence. For instance, for the create an account system, the test will enter in new email and password, and if the account is created and pops up in Firebase, then it is a pass. The testing will commence from the beginning of the application flow.

I.3. Scope

The purpose of this document is to outline the plans and steps for testing the Restaurant Management Application.

Testing Strategy

For testing any particular function, the required tests will identified. The required outcome for any given test will also be noted and the compared against what happens during the test. If the outcomes match, then the test will pass. If they differ, then the test will fail. Screenshots will be taken before, during, and after. This process will be repeated with each function.

The approach taken will be continuous delivery (with the delivery location being the main branch of the github repo).

Test Plans

I.1. Unit Testing

For unit testing, we will be testing each specific function of our application by themselves, not paying attention to how it interacts with other objects. This is to ensure the primary, basic functions work before trying to integrate them with others.

I.2. Integration Testing

The majority of our integration testing will see if the functions mentioned above integrate with our database to make sure that the information is saved and can be reused by the user at a later time.

I.3. System Testing

System testing is a type of black box testing that tests all the components together, seen as a single system to identify faults with respect to the scenarios from the overall requirements specifications. Entire system is tested as per the requirements.
During system testing, several activities are performed:

I.3.1. Functional testing:

Tested Aspect: Add Item
Expected Result: When the add item button is clicked, a page will pop up. Once the page is filled out, the user can click “add” and the new item will appear at the bottom of the chart.
Observed Result:
Test Result:
Test Case Requirements: The function must take the input from the user correctly and add the new inventory item to the bottom of the chart

Tested Aspect: Remove Item
Expected Result: When the red ‘X’ on an item is clicked, the entire row item should disappear
Observed Result:
Test Result:
Test Case Requirements: The selected item must disappear from the chart

Tested Aspect: Edit Item
Expected Result: When the blue pencil icon in an items row is clicked, an edit page will pop up. The user should be able to edit any of the information on the page and when the user clicks “Save”, the new, updated information will appear in the row.
Observed Result:
Test Result:
Test Case Requirements: The edit page must pop up when the pencil icon is selected. Then the user must be able to edit any of the previous information and have the new information be reflected in the updated chart once the user clicks the save button.

I.3.2. Performance testing:

Load Testing:

- Simulate multiple users accessing the system simultaneously to ensure it can handle concurrent usage without performance degradation.
- Measure the response time for various actions (logging in, adding inventory items, retrieving data) under normal and peak load conditions.

Stress Testing:

- Stress the system beyond its specified limits to determine its breaking point and observe how it behaves under extreme conditions.
- Identify potential bottlenecks and performance issues that may arise during high usage.

Scalability Testing:

- Test the system's ability to scale up or down based on varying loads.
- Ensure that the system can handle an increasing number of users and data without compromising performance.

I.3.3. User Acceptance Testing:

End-User Validation:

- Involve end-users in testing the system to ensure it aligns with their workflows and requirements.
- Collect feedback from users on the system's usability, functionality, and overall performance.

Requirement Verification:

- Compare the system's functionalities with the initial requirements to ensure all specified features are implemented correctly.
- Verify that the system meets both functional and non-functional requirements.

Operational Readiness:

- Ensure that the system is ready for deployment and can be used in a live environment.
- Conduct installation testing to confirm that the system can be installed and configured correctly on target devices.

Environment Requirements

No special tools will be needed in order to perform the tests. In XCode, when a new build is generated, a virtual iPhone with the application downloaded will be provided to the developer. This will be the location for the testing and will provide the developer a visual landscape to see. All the hardware that is currently in use (computers) will all that will be needed.

Glossary

Define technical terms used in the document.

References

Appendix-A

Example Testing Strategy:

1. Identify the requirements to be tested. All test cases shall be derived using the current Software Requirements Specification.

2. Identify which particular test(s) will be used to test each module.

3. Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.

4. Identify the expected results for each test.

5. Document the test case configuration, test data, and expected results.

6. Perform the test(s).

7. Document the test data, test cases, and test configuration used during the testing process. This information shall be submitted via the revised Test Plan document.

8. Successful unit testing is required before the unit is eligible for component integration/system testing.

9. Unsuccessful testing requires a bug form to be generated. This document shall describe the test case, the problem encountered, its possible cause, and the sequence of events that led to the problem. It shall be used as a basis for later technical analysis.

10. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.