INTERNATION INSTITUTE OF PROFESSIONAL STUDIES, DAVV

SYSTEM ANALYSIS AD DESIGN

PROJECT REPORT-

**System Testing Plan [STP]**

**Online Food Ordering System**

**SUBMITTED BY-**

DRAKSHI CHOPRA

**IT-2K19-16**

MTECH V Sem

**SUBMITTED TO-**

Dr. SHALIGRAM PRAJAPAT

**TABLE OF CONTENTS**

1. TESTING DESIGN
   1. TESTING
   2. PHASES
2. REQUIREMENT TRACEBLITY
3. TESTING SCHEDULE
4. RECORDING PROCEDURES
5. HARDWARE AND SOFTWARE REQUIREMENTS
6. GENERAL STEPS FOR TESTING

**PHASES**

The structure of the system can be divided into three main logical components, plus the database, which is invisible to the end user. Each of these components must be tested individually, and the approaches which will be used for each component are described in the following sections.

Testing of the database component is very straightforward, and has actually already been mostly completed. The database was the first component designed and before beginning work on any of the applications, I wrote all of the SQL statements I expected to need and executed them directly, essentially isolating the database, using the psql client. By doing this I was able to reveal, and promptly fix a large percentage of the errors within the database itself.

#### Web Ordering System

Testing of the web ordering system will be the most strenuous, as it is the component that will see the highest frequency of use and will be exposed to the most users, which leads to a higher potential of failure. Testing here will be divided into two phases. During normal use case testing I will execute all of the functions available through the web interface using a broad spectrum of reasonable values that a user would be expected to input. In addition to simply observing the in-application effects, I will also be monitoring and inspecting the JSON requests and responses (using Firebug) to make sure that everything is sent and received correctly.

In phase two I will perform exceptional use case testing, where I will artificially generate cases that shouldn’t arise, but possibly could, and monitor how the system handles these cases. These cases fall into one of two categories – when the mistake happens in the browser and the server has to deal with it, or the other way around. I have tried to place appropriate checks on all values being sent back and forth so the system realizes something is wrong before going to the database and potentially changing the state of the system, but it will very important to see if there is anything I have not accounted for.

#### Menu Management System

Testing of the menu management system will be very similar to that of the web ordering system, as I will first run test cases where the user supplies acceptable values, and afterwards test how the system responds to unexpected input.

#### Order Retrieval System

Of all the components, testing of the order retrieval system will be the simplest. Since it is simply an interface to display the results of database queries and has no potential to change the state of the system, the only thing that really needs to be tested is how the system responds when a result set is not in the form it is expecting. This will be done by intentionally corrupting the database and analyzing the response of the order retrieval system.

**REQUIREMENT TRACIBLITY**

In the requirements document, I specified the following functional requirements:

For the Web Ordering System:

* Create an account.
* Manage their account.
* Log in to the system.
* Navigate the restaurant’s menu.
* Select an item from the menu.
* Customize options for a selected item.
* Add an item to their current order.
* Review their current order.
* Remove an item/remove all items from their current order.
* Provide delivery and payment details.
* Place an order.
* Receive confirmation in the form of an order number.

For the Menu Management System:

* Add a new/update/delete vendor to/from the menu.
* Add a new/update/delete food category to/from the menu.
* Add a new/update/delete food item to/from the menu.
* Add a new/update/delete option for a given food item.
* Update price for a given food item.
* Update default options for a given food item.
* Update additional information (description, photo, etc.) for a given food item.

For the Order Retrieval System:

* Retrieve new orders from the database.
* Display the orders in an easily readable, graphical way.
* Mark an order as having been processed and remove it from the list of active orders.

In order to assure thorough testing, I will have to generate cases in which each of these functions is performed, not just with a single input value, but an example of each possible class of input. This may seem tedious, but is absolutely necessary since the system is so heavily dependent on user input and must respond appropriately to anything the user may do. The good news is that because the system was design to contain only the absolutely necessary functionality, the testing of the different functions should flow smoothly into one another.

**TESTING SCHEDULE**

Throughout all of the design and development phases, I have been performing unit tests on each component, assuring that it works properly before introducing it into the rest of the system, but I plan on beginning rigorous testing of the system starting in the first week of December. I will begin by putting the system through its normal paces as a normal user would be expected to. I will test the entire functionality of the system, but will do so following the normal logical flow, and only providing reasonable values for user input.

Once I am happy with how the system performs for normal use cases (which hopefully will be completed by December 8th), I will move on to testing the exceptional use cases. These are the use cases that would never be encountered by the average user, but, whether through confusion or malice, may come up. Two examples would be adding hundreds of items to an order or attempting to supply an SQL statement as a login credential.

In the final phase of testing, which I will begin on December 15th, I will carry out stress and performance testing. In this phase, I will bombard the system with an increasing number of HTTP requests and then measure response times and see at what point things begin to break down.

**RECORDING PROCEDURES**

The recording procedures I will use can be divided into two categories. For the use case testing, the tests will be designed in a “Pass/Fail” manner, making recording the results very simple. Each time a series of tests is run, the results will be recorded in a spreadsheet, where each outcome can be identified by the name of the test case along with the execution date and time.

For the stress and performance testing, the procedure will be similar. However, since these tests involve numerical results rather than simply “Pass/Fail”, I will be able to not only record the results, but to also perform analysis (both numerically and graphically) on them to get a better idea exactly what the numbers mean.

**HARDWARE AND SOFTWARE REQUIREMENTS**

The testing to be performed will require no special hardware, but some specially designed software may be used, particularly when performing stress testing. I plan on writing and utilizing a multi-threaded Java program to bombard the system with an increasing number of HTTP requests and then measure response times and see at what point things begin to break down. I also will be using the Firefox plug-in Firebug to monitor the JSON requests and responses.

**GENERAL STEPS OF TESTING**

Integration of all the modules/forms in the system.

Preparation of the test cases.

Preparation of the possible test data with all the validation checks.

Actual testing done manually. Recording of all the reproduced errors.

Modifications done for the errors found during testing.

Prepare the test result scripts after rectification of the errors.

Following the conclusion of system testing, Acceptance Testing is the

next step. Clients accomplish this on their end and review the software

to suggest any more modifications that are to be made.