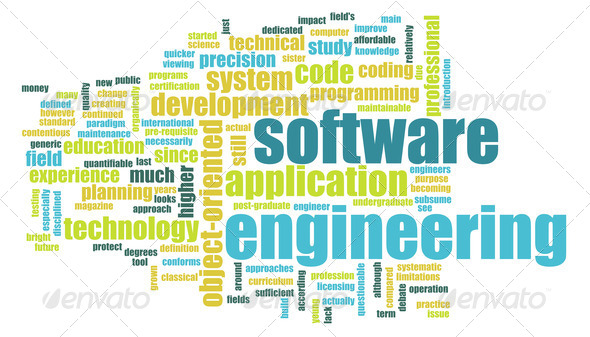
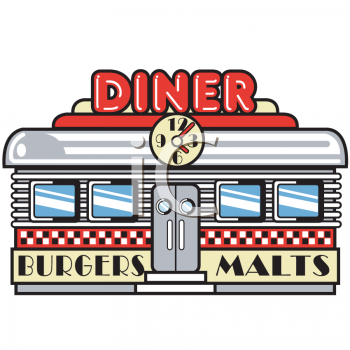
SOFTWARE ENGINEERING THEORY PROJECT  
TEST MANUAL



**Fall Semester 2016-2017**

**TOPIC:**

Restaurant Service System



**Teacher:**

Prof. Alok Chauhan

**STUDENTS:**

* Osho Agyeya(15BCE1326)
* Kashish Miglani(15BCE1003)
* Sanchay Gupta(15BCE1190)
* Sachin Gopal(15BCE1188)

# Introduction:

The project is aimed at developing a web application that provides online order placement facility. The users have a variety of orders to choose from. They get the bill once the order has been placed. The administrator gets to decide which orders will be prepared or not.

The document is aimed at providing an efficient test plan to benchmark the credibility of the system.

# Features to be tested:

The following features are going to be tested:

* login signup activity
* order placement activity
* bill calculation activity
* admin functionalities
* logout activity

# Features not to be tested:

The following features shall not be tested:

* Register activity: This is a onetime activity which shall not be used once the user has registered successfully.
* Individual dishes: All the order options/various dishes on the menu are working perfectly as their source code has been reviewed several times
* About us: This is a single web page denoting the details of the restaurant. Since it is just conveying information, it does not need testing.

# Approach:

The overall approach is as follows:

* Initially relevant test cases are provided.
* Then, the expected outcomes are assumed.
* Test execution
* Comparison of results of the test with assumed/expected results
* Recording the results for statistical evaluation

# Item pass/fail criteria:

The software is supposed to have passed the test criteria if:

* All the inputs are accepted by the software
* The software performs all the steps of the algorithms defined.
* The software produces the desired outputs
* The outputs are provided in the desired time frame

All the other conditions are supposed to come under failure conditions.

# Test deliverables:

After the test has been completed, a summary of the test report is provided which states the following conditions:

* Inputs are accepted by the software
* Outputs generated
* Time slice within with the output is provided
* The improvement that is expected for the next testing

# Testing tasks:

The testing tasks are as follows:

**Test Strategy**

The purpose of testing is to find defects, not to pass easy tests. A test strategy basically tells you which types of testing seem best to do, the order in which to perform them, the proposed sequence of execution, and the optimum amount of effort to put into each test objective to make your testing most effective. A test strategy is based on the prioritized requirements and any other available information about what is important to the customers. Because you will always face time and resource constraints, a test strategy faces up to this reality and tells you how to make the best use of whatever resources you do have to locate most of the worst defects. Without a test strategy, you are apt to waste your time on less fruitful testing and miss using some of your most powerful testing options. You should create the test strategy at about the middle of the design phase as soon as the requirements have settled down.

**Testing Plan**

A testing plan is simply that part of your project plan that deals with the testing tasks. It details who will do which tasks, starting when, ending when, taking how much effort, and depending on which other tasks. It provides a complete list of all the things that need to be done for testing, including all the preparation work during all of the phases before testing. It shows the dependencies among the tasks to clearly create a critical path without surprises. You will be able to start filling in the details of your testing plan as soon as your test strategy is completed. Both your test strategy and testing plan are subject to change as the project evolves. Modify your strategy first, if you need to, and then your testing plan.

**Test Cases**

Your test cases (and automated test scripts if called for by your strategy) are prepared based on the strategy which tells you how much of each type of testing to do. Test cases are developed based on prioritized requirements and acceptance criteria for the software, keeping in mind the customer’s emphasis on quality dimensions and the project’s latest risk assessment of what could go wrong. Except for a small amount of ad hoc testing, all of your test cases should be prepared in advance of the start of testing. There are many different approaches to developing test cases. Test case development is an activity performed in parallel with software development. It is just as difficult to do a good job of coming up with test cases as it is to program the system itself. In addition to figuring out what steps to take to test the system, you need to know the requirements and business rules well enough to predict exactly what the expected results should be. Without expected results to compare to actual results, you will not be able to say whether a test will pass or fail. A good test case checks to make sure requirements are being met and has a good chance of uncovering defects.

**Test Data**

In addition to the steps to perform to execute your test cases, you also need to systematically come up with test data to use. This often equals sets of names, addresses, product orders, or whatever other information the system uses. Since you are probably going to test query functions, change functions and delete functions, you will most likely need a starting database of data in addition to the examples to input. Consider how many times you might need to go back to the starting point of the database to restart the testing and how many new customer names you will need for all the testing in your plan. Test data development is usually done simultaneously with test case development.

**Test Environment**

You will need a place to do the testing and the right equipment to use. Unless the software is very simple, one PC will not suffice. You will need all of the components of the system as close as possible to what it will eventually be. Test environments may be scaled-down versions of the real thing, but all the parts need to be there for the system to actually run. Building a test environment usually involves setting aside separate regions on mainframe computers and/or servers, networks and PCs that can be dedicated to the test effort and that can be reset to restart testing as often as needed. Sometimes lab rooms of equipment are set aside, especially for performance or usability testing. A wish list of components that will be needed is part of the test strategy, which then needs to be reality checked as part of the test planning process. Steps to set up the environment are part of the testing plan and need to be completed before testing begins.

## Test cases

**Machine Configuration:**

Browser: Google Chrome Version 53.0.2785.143 m

Server: MAMP

Database System: SeQueL Pro

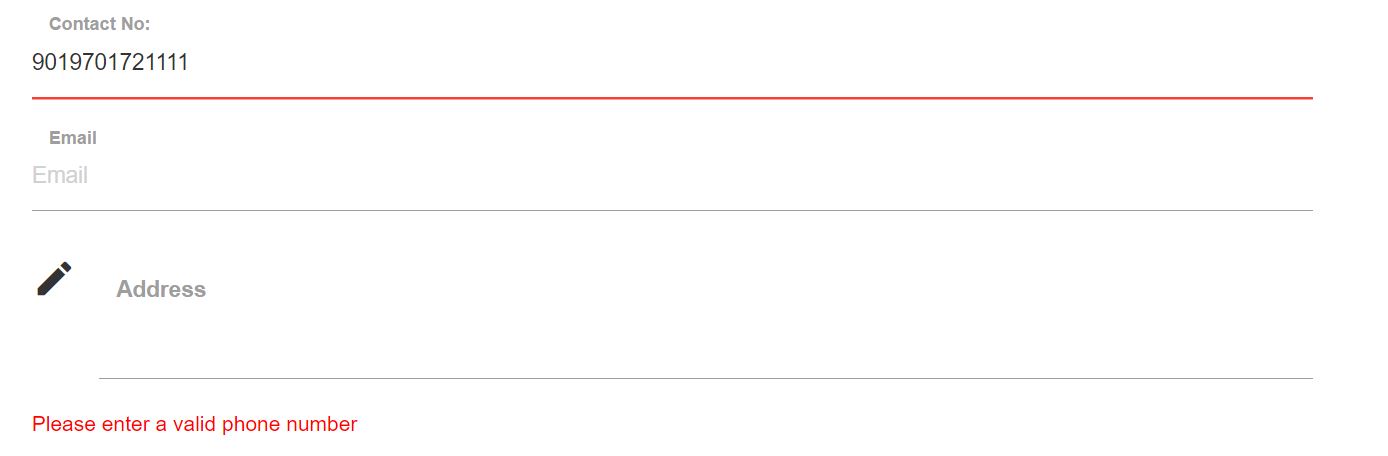
**Requirements being tested:**

The response of the following components of the system to varying stimuli:

* Sign up
* Login
* Order placement
* Order status and administrator overview

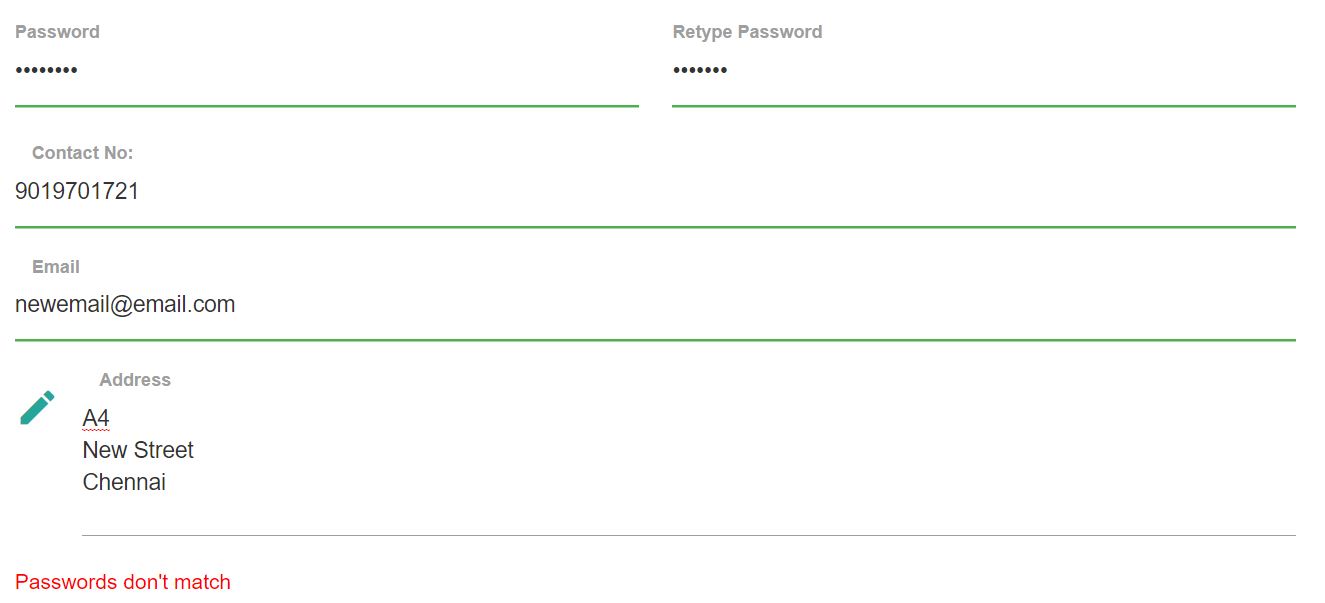
# Sign up Module:

**Test case 1: Functional, Stress and Structural tests**



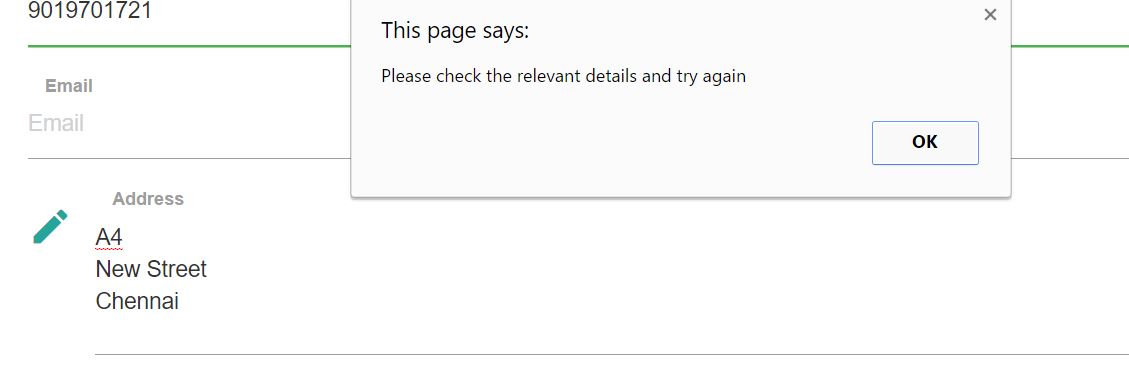
On entering an invalid phone number (that is greater than 10 digits), we receive an error stating the same. The registration fails in the subsequent stage.

**Test case 2: Functional, Stress and Structural tests**



On entering a different password in the retype password we get an error message stating the same. The registration fails in the subsequent stage.

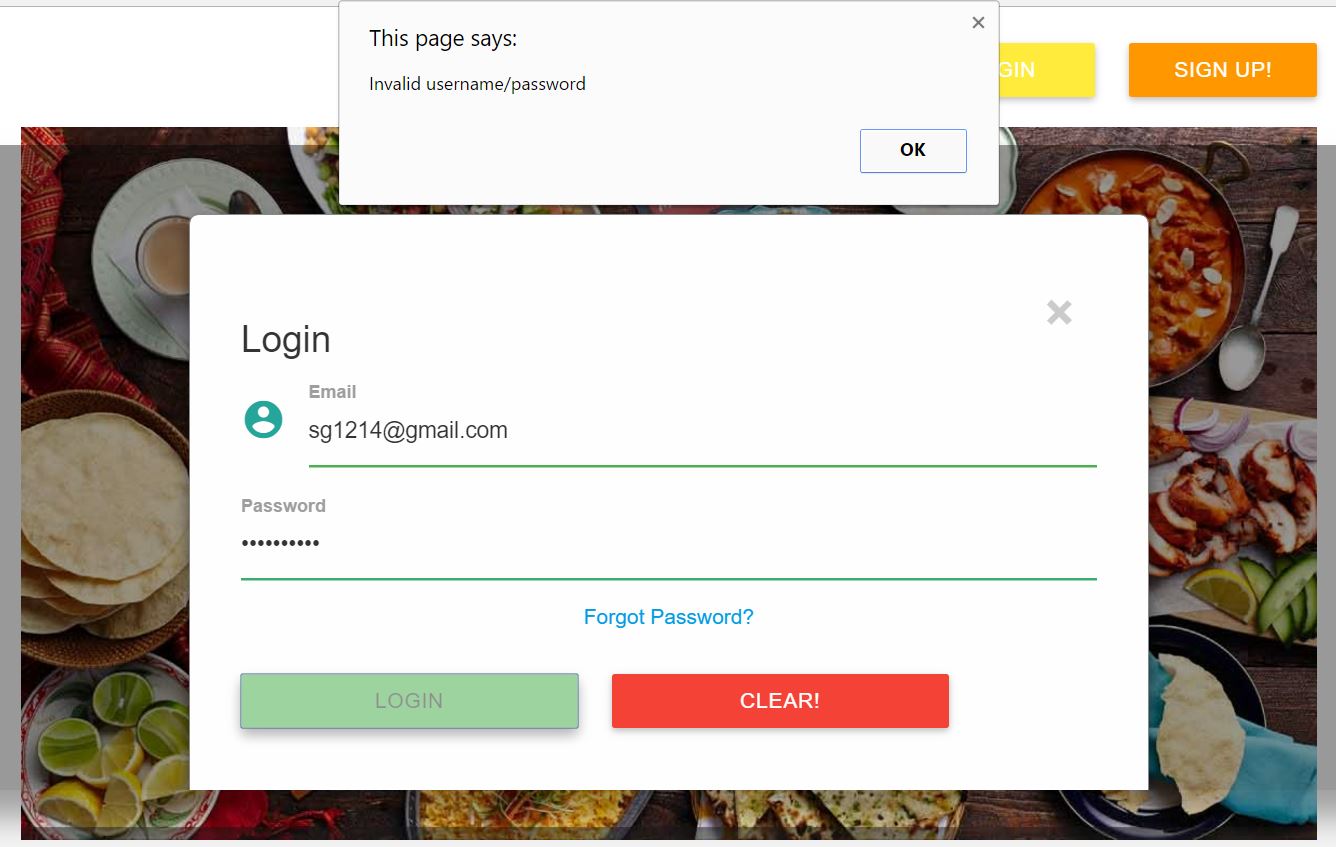
**Test case 3: Functional, Stress and Structural tests**



During the registration process, if a field is left empty, we get an alert to check the relevant details and try again. Database is not updated until valid inputs are provided.

# Sign in Module:

**Test case 4: Functional and Stress test**

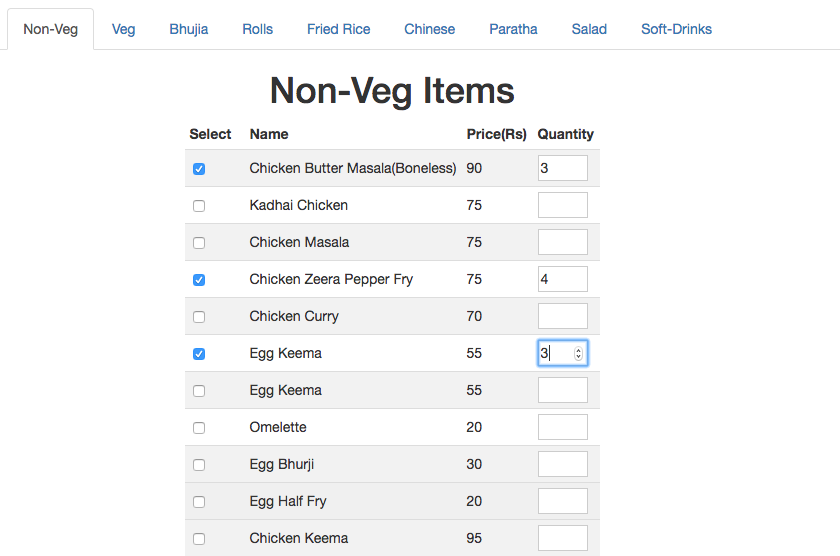


If a user provides an invalid username or password, that is tries to login before registration, an alert message notifies the user of this and the user can try signing in again or register on the website.

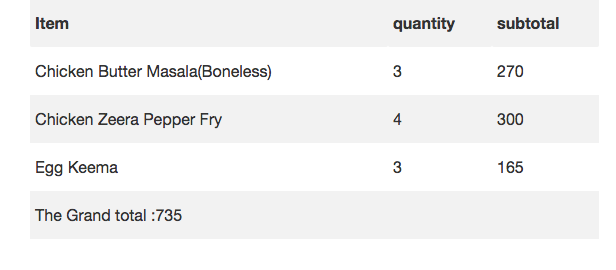
# Order Placement Module:

**Test case 5: Functional test**

**Input:**



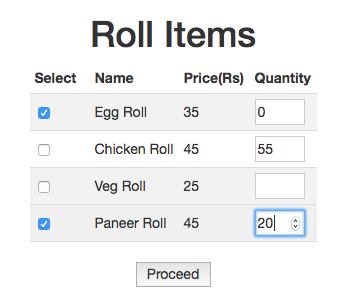
**Output:**



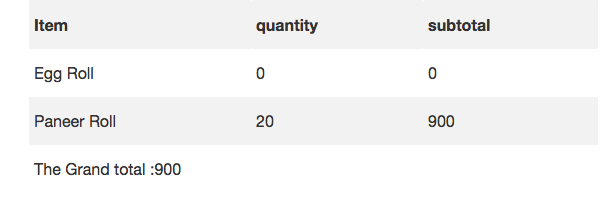
When a user selects the check box and enters a positive quantity value, the order proceeds normally and the bill reflects this. Order calculation also proceeds as expected.

**Test case 6: Functional and Stress test**

**Input:**



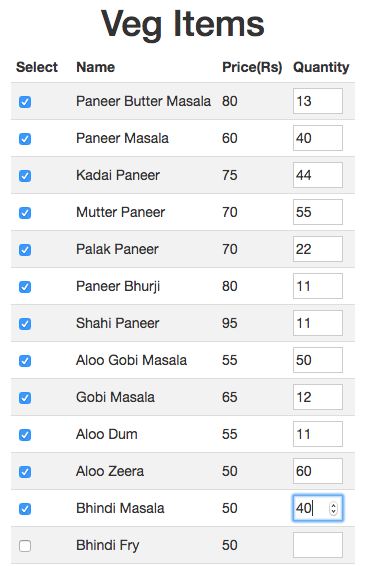
**Output:**



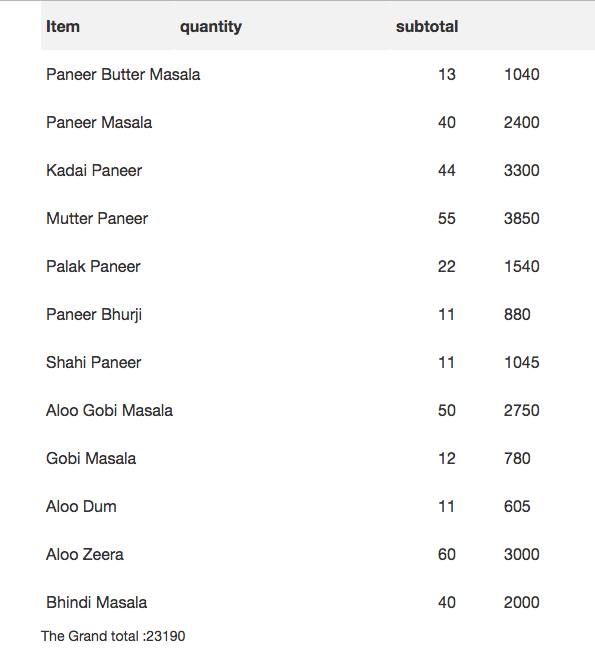
When a user checks a check box without entering the quantity or vice versa, the bill generated shows the quantity as 0 and handles the exception.

**Test case 7: Functional and Stress test**

**Input:**



**Output:**

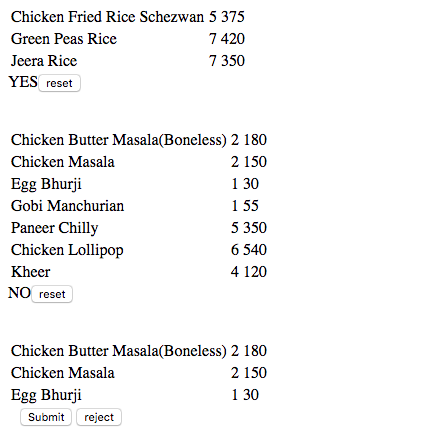


When a user places a bulk order with extremely large quantities, the system computes the bill normally. The page disallows the user to add an item whose quantity is greater than 99. The admin ultimately has the choice to accept or reject the order.

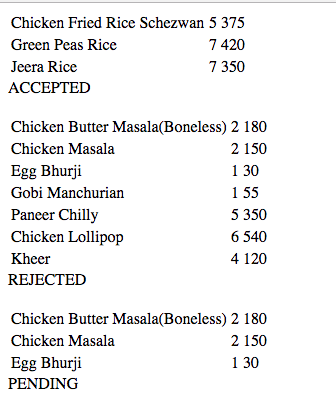
# Order status and administrator overview module

**Test case 8: Functional test**

**Input**



**Output**



The administrator i.e. the restaurant manager has the option to Submit or Reject the order. This is reflected in the order status. This can further be accessed by the restaurant’s employees to start preparing the order for faster deliveries.