Submit a PDF file with a detailed test plan. For every use case you had, you must have at least one test. Each test needs to contain:

1. Requirement it is testing
2. Use case it is testing (there may be several use cases for one requirement)
3. Inputs to the test
4. Expected outputs
5. Expected backend changes (change to DB state for example)
6. Test procedure (which screen to load, what action to take etc.)

Main Use Cases

- ### Test for Search Engine

  Requirement:

  easy maneuvering for user over the menu

  Use case:

  search engine

  Input:

  cuisine or category or dish name

  Output:

  relevant dishes

  Backend Changes:

  no backend changes

  Test Procedure:

  homepage to load

  Negative tests:

  Since we're providing drop-down for choosing cuisine/category and dish name, there is no chance that users can provide wrong inputs and hence no negative tests are needed here.

- ## Test for Place an Order

  Requirement:

  It is a basic necessity that the user is able to choose the dishes he/she wants to eat and place an order of the list of items he has chosen for his/her meal. This use case is designed to support this need of the users. After placing an order, the orders relation will be updated.

  Use case:

  The Place and Order Use Case (Use Case No. 2)

  Input:

  The primary input is the list of dishes and the customer ID that drives the further actions. Once, this is provided, this input is passed into the system using the “order” button. This required the dish This will open up a form with a dropdown to choose the required dishes. This will kind of put the customer’s order into a buffer so it can possibly be sent to chefs. However, users can modify this until he/she does not click on “confirm”.

  Output:

  The order gets placed in the owner’s view.

  Backend Changes:

  Firstly, the ordered dishes will be added into the orders relation (relationship between the non_waiting customer and dishes). The corresponding attributes of this relation (Cost, Date, Time, Day, Quantity, ID) will be set as of now. The review though will be set later. All the additions may possibly be shown by adding the corresponding instances and orders in the frontend grid view of the corresponding customers.

  Apart from this the order frequency attribute of the customers database will also be updated accordingly.

  Test procedure:

  The Order Food page is selected from the main menu/navigation bar of the website. The customer should be able to place his/her order first by clicking on order and then on action. The dish ordered by the customer will be shown in front of his/her id as a sanity check to ensure that he/she is given precisely those dishes that he/she has ordered.

- **Test for Manage Employee Information**

  Requirement:

  The need to update/manage employees after sufficiently long intervals of time. Only the owner can update this relation. Many details of employees may change (eg: contact numbers, roles, address etc.). Moreover employees may leave or new employees may join. It is important to have such a use case.

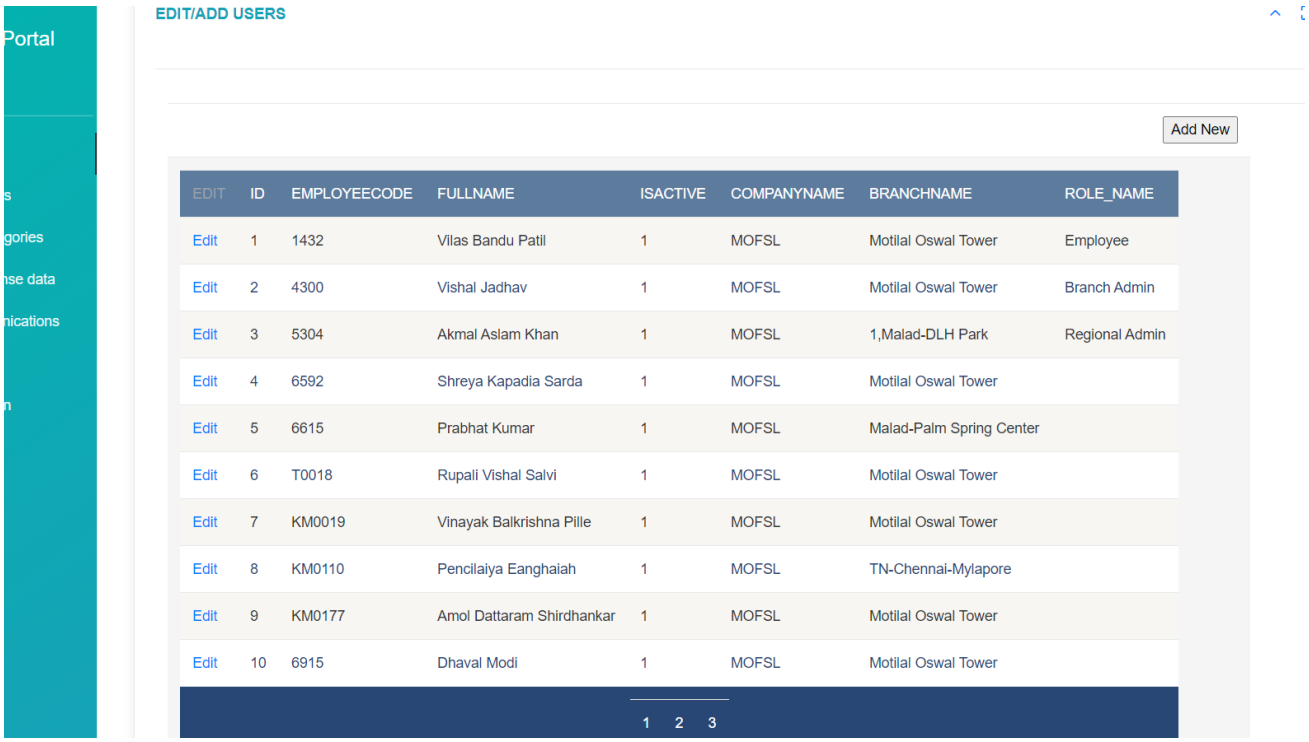
  Use case:

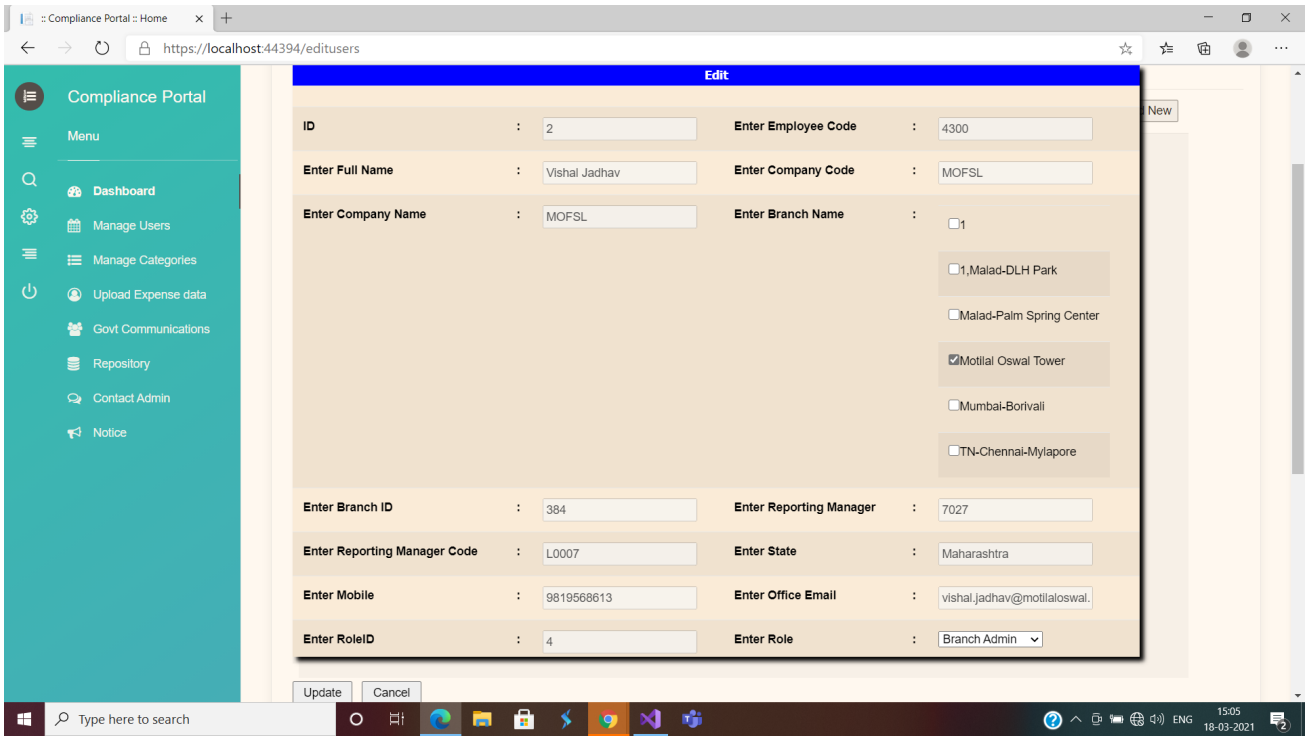
  The Manage Employee Information Use Case (Use Case No. 3)

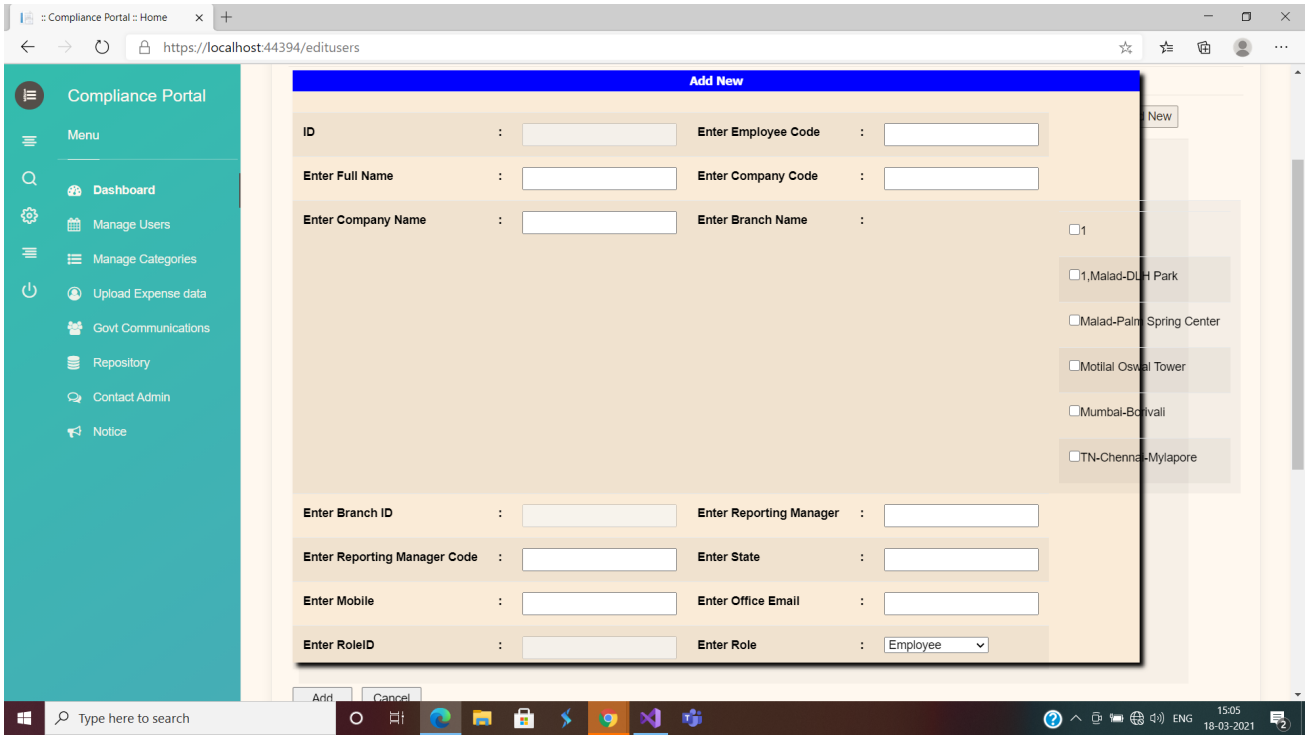
  Input:

  Delete/Update/Create (of CRUD) command. In case of delete the input is just the delete button press. In case of create/update after the button press a form opens up with old values/blank values for update and create respectively. Then the new values are typed in (as shown in screen design in Deliverable 3) and then the confirm button is pressed.

  Note: Sample screens for input and output given below- they were also presented in Deliverable 3







Output:

Output is the updated grid view in the page (automatic display of the updated relation without need to reload the page). It will be part of the controller and backend logic.

Backend Changes: (change to DB state for example)

The change in the corresponding relation in the database (staff contact/staff/cooks tables). A particular row is updated/deleted or a new row may be added.

Test procedure: (which screen to load, what action to take etc.)

The Manage Employee Information page is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner else access is denied.

Then all 3 options will be tested. (on all the employee tables)

- Create: Add an employee with the asked attribute values in the form.
- Update: Update existing employee information with the new attribute values in the form.
- Delete: Delete a tuple from a relation shown in the grid view.

For the backend programmer one additional test can be to go to the database and check if the updates have taken place properly. This is in some ways redundant since grid view has a one-one correspondence with the backend relations.

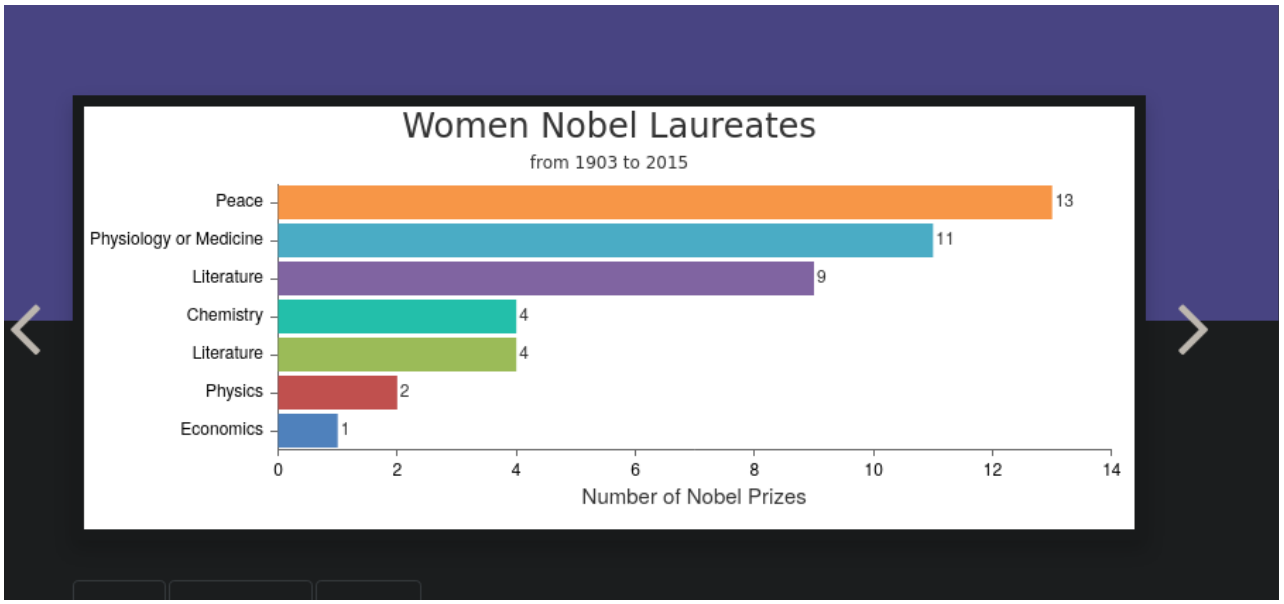
- **Test for Analytics**

  Requirement:

  This test helps us do an efficient analysis of the resources being used, the consumption and food wastage being done, the profit being earned, the order frequency and the amount spent by different customers, the reviews of the dishes, the Total cost of dishes made by the chefs and the total servings done by the waiters.

  Use Case:

  The Analytics Use Case (Use Case No. 4)



  Input:

  the quantities and amount of the required parameters to be analysed vis-a-vis the day, the amount of money spent on the day, the profit earned for the day, the food wasted for the day, the different orders placed by the different customers that arrived and ordered in the restaurant for the day, the different dishes cooked by different chefs and the different dishes served by different waiters.

  Output:

  The output consists of plots, graphs, visual and textual analysis for the attributes of different relations mentioned in the requirements.

  Backend Changes:

  The backend changes will now include multitude of dimensions.

  Some involve change of the tracks relation between the owner and accounts. A tuple involving the current day, the day's expenditure, the day's profit and the total food wasted on that day will be added to the tracks relation at the end of the day.

  Further, the order frequency of the customers will be updated considering

different customers that came on that day. When the customer gives the review for the dish, this attribute will also be updated.

Apart from this, the TCDM attribute for chefs and total servings done by the waiters will be updated accordingly

Test Procedure:

The Analysis page is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner else access is denied.

Once the login is successful, the owner has the option to view the analysis in the form of bar graphs, pie charts, scatter plots, histograms, etc. This will depend on the attribute to be analysed and the available analysis options.

Also, this use case will also test whether the TCDM attribute of staff relation, Quantity, Review of the orders relation, the discount offered in the payment relation are modified for the day accordingly.

- ## Test for Allot Order to Chef

  Requirement:

  After the user is done with the ordering of the corresponding dishes for his/her meal, we need to allot and assign the ordered dishes to the appropriate chefs using the mapping order -> chef by the cooks relation. This use case is designed for it.

  Use case:

  The Allot Order to Chef Use Case (Use Case No. 5)

  Input:

  The confirm button will confirm placing of the order and the order will be sent to the chefs for further processing.

  Output:

  After the confirm button is pressed, the dishes in the order will be allocated to different chefs. The output will also logically include the preparation of dishes by the chefs

  Backend Changes:

  Once the chefs are assigned the dishes, the order chef mapping values will also be added to the cooks relation. All the additions may possibly be shown by adding the instances in the frontend grid view of the corresponding chefs.

  Apart from this, the statistics will also be updated. This will include an update of the TCDM (Total cost of dishes made) attribute of the chefs in the staff relation. Also, the count of different dishes ordered till now will also be updated.

  Test procedure:

  Once the orders are placed, the orders can be assigned to chefs and we can test if the corresponding tuples are getting added into the grid view in the frontend and (the backend). We can move to the analytics page immediately and test if the orders cooks count of the concerned chef has been incremented or not. If these 2 conditions are satisfied - Test is successful.

  For the backend programmer one additional test can be to go to the database and check if the updates have taken place properly. This is in some ways redundant since grid view has a one-one correspondence with the backend relations.

- ## Test for Order History

  Requirement:

  the owner needs to have an account of the successful/unsuccessful delivery of orders to the customers

  Use case:

  The order history use case

  Input:

  date range and then click on order history

  Output:

  orders placed in that date range sorted from newest to oldest

  Backend Changes:

  No changes

  Test procedure:

  We will add orders for different dates in the database and see if only the orders in the specified date range are being outputted and also check whether all orders in the daterange are being outputted.

  Negative Test:

  In the “to” and “from” of date range, put the “to” date which comes after “from” date and it should give no output.

- ## Test for Update Inventory

  Requirement:

  The need to update inventory stock after sufficiently long intervals of time. Only the owner can update this relation. Inventory here includes the need to update the stock in ingredients relation and the dishes relation.

  Use case:

  The Update Inventory Use Case (Use Case No. 7)

  Input:

  Delete/Update/Create (of CRUD) command. In case of delete the input is just the delete button press. In case of create/update after the button press a form opens up with old values/blank values for update and create respectively. Then the new values are typed in (as shown in screen design in Deliverable 3) and then the confirm button is pressed.

  Note: Sample screens for input and output given here in manage employee information - they were also presented in Deliverable 3

  Output:

  Output is the updated grid view in the page (automatic display of the updated relation without need to reload the page). It will be part of the controller and backend logic.

  Backend Changes: (change to DB state for example)

  The change in the corresponding relation in the database (dishes/ingredients table). A particular row is updated/deleted or a new row may be added.

  Test procedure: (which screen to load, what action to take etc.)

  The Update Inventory page is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner else access is denied.

  Then all 3 options will be tested. (on all the inventory tables)

  - Create: Add an ingredient/dish with the asked attribute values in the form.
  - Update: Update existing ingredient/dish with the new attribute values in the form.
  - Delete: Delete a tuple from a relation shown in the grid view.

  For the backend programmer one additional test can be to go to the database and check if the updates have taken place properly. This is in some ways redundant since grid view has a one-one correspondence with the backend relations.

- ## Test for Serving Food

  Requirement:

  After the chef completes making the dishes in the order and marks it we have to serve it to the customers. So a mapping for order->waiter has to be made which is done through this use case.

  Use case:

  The Serving Food Use Case (Use Case No. 8)

  Input:

  Input firstly is the serve button of a particular row in the (pending) orders relation's grid view shown. which will open up a form with a dropdown to choose a waiter. Then the confirm button.

  Output:

  After the confirm button is pressed the corresponding order is no longer pending and so it is deleted from the orders relation. (removed from grid view in the frontend) - Automatic display of the updated relation without need to reload the page). It will be part of the controller and backend logic.

  Not only an output on the same page but the statistics in the analytics page is also updated. Here the concerned statistic is the number of orders served by the particular waiter.

  Backend Changes: (change to DB state for example)

  Firstly the served order must be deleted from the backend database and thus removed from grid view in the frontend.

  Then the above mentioned changes about employee performance is made to the corresponding relation in the backend database.

  Test procedure: (which screen to load, what action to take etc.)

  The Serve Food page is selected from the main menu/navigation bar of the website. Screen of this use case will then be loaded only if the login is of the owner or manager else access is denied.

  Then orders can be assigned to waiters and we can test if the corresponding tuples are getting deleted from the grid view in the frontend and (the backend). We can move to the analytics page immediately and test if the orders served count of the concerned waiter has been incremented or not. If these 2 conditions are satisfied - Test is successful.

  For the backend programmer one additional test can be to go to the database and check if the updates have taken place properly. This is in some ways redundant since grid view has a one-one correspondence with the backend relations.

- ## Automation

  Requirement:

  The dish recommendation part is quite common in today's E-Commerce/food delivery apps which helps users to decide faster and have a better experience by ordering the best quality & most loved products. There are two recommendations - most loved, your personal favorite. The other part we're automating is quantity flag (flagging if quantity of any particular ingredient is below a certain threshold)so that the owner is always aware of what needs to be ordered without having to constantly keep checking the inventory.

  Use case:

  The automation use case

  Input:

  Since this is automated there's no input from the user side. There will be sections on the user and admin interface where based on the entries in the database the recommendations will be added.

  Output:

  Most loved dishes will include the most ordered dishes (in the advanced version, we can set the metric of most loved to include - ratings as well as number of times people ordered it (loyalty)) sorted to include top k dishes. And user favorite dishes will include the ones he/she has ordered most frequently. The output for quantity automation will be: "These ingredients need to be ordered asap" and will be followed by a list of ingredients. If the list is empty the "display" of the entire text will be "hidden".

  Backend Changes:

  None

  Test procedure:

  Place a few orders from different users' accounts and verify if they're being recommended the right dishes. Print the output of the queries for most loved and personal favorite dishes to see if the counting is being done correctly. Based on the orders we'll check if the quantity of ingredients is being updated properly. We'll create multiple orders for a particular dish so that it's ingredients' quantity in the inventory falls below the threshold and we can verify that it does indeed flag an error! We will also place orders corresponding to different dishes containing a common ingredient to recheck the error flagging.

Negative Tests

- ### Invalid Order

  Requirement:

  The system must be able to handle invalid inputs and searches without crashing, and should direct the user appropriately to perform corrective action.

  Use case:

  The Place and Order Use Case (Use Case No. 2)

  Input:

  The input is either in the form of selection of dishes/categories from a drop down list or plaintext input from the user in a text box. The former ensures no input error, as we have valid options. However the latter is completely up to the user, so must be handled.

  Output:

  If the specific combination that the selection of filters from the drop-down list yields is non-existent, then the system must indicate empty and provide the user with easy access to resubmit the dish selection options. If the user enters something altogether invalid, then the system must indicate as such without going to the back end, and request the user to retry.

  Backend Changes:

  No backend changes. However, we may capture the error log for analytics (How often an invalid order is placed,etc, time wasted etc).

  Test procedure:

  Same as the test for place an order (test 2)

- **The login credentials of the owner or manager or the concerned user is incorrect**

  Requirement:

  The system must be able to handle invalid inputs without crashing, and should direct the user (owner/manager/customer etc) appropriately to perform corrective action.

  Use cases: (for various users)

  The Place and Order Use Case (Use Case No. 2),
  The Manage Employee Information Use Case (Use Case No. 3),
  The Update Inventory Use Case (Use Case No. 7)

  Input:

  The input is either in the form of a username and a password, or userID and password.

  Output:

  The concerned user should be unable to progress in the case where something invalid is entered. The system should inform the user of his/her error and prompt the user to retry.

  Backend Changes:

  No backend changes. However, we may capture the error log for analytics (How often an invalid login attempt is made, time wasted etc).

  Test procedure:

  This will be tested on the screens where a login is required (i.e, before placing an order for a user, or when the owner logs in (the screen before he/she gets the options to manage employees, update inventory etc.

- ## Payment Failure

  Requirement:

  Payments for one order mustn't be made twice. A failed payment should be indicated and the customer shouldn't see that the order has been successfully placed. The order should not be allotted to the chef yet.

  Use case:

  The Place and Order Use Case (Use Case No. 2)

  Input:

  The customer chooses a method of payment. Online methods may fail (some percent chance, drawn from a probability distribution. Invalid credit/debit card numbers must result in a failure.

  Output:

  An error message must be output, and payment requested again.

  Backend Changes:

  No backend changes. However, we may capture the error log for analytics (How often a payment fails, time wasted etc).

  Test procedure:

  On the payment screen, a non-16 digit card number or a non-3 digit CVV may be entered. Or, even if all is proper, there is a non-zero chance that the payment fails on the bank side. These are the cases where the same screen should display an error message, without redirecting away.