Ver3.0 Planning document – School Canteen – Jeremy Roberts

Task 0: Explain what you are doing/ going to accomplish

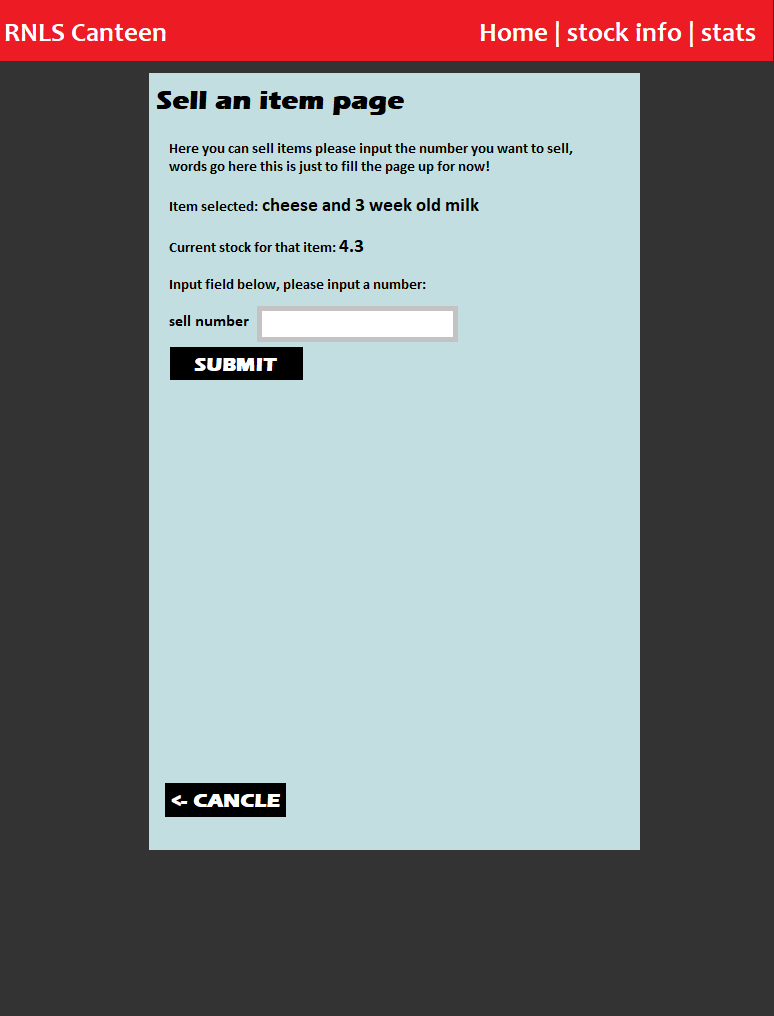
I am going to create a website that manages a school canteen, it will manage the stock of items on the menu and students will be able to order food. I will create a program that stores the students name and what they ordered in an array. The inputs will be processed with a python bottle program and displayed on an html website.

**This version:** This is another Large Version, First I am going add the long-awaited feature of selling food items, you will be able to sell any amount (within its stock level) of that item, it should add to the “food.sold” value for the item. Then I will add a statistics page mainly just for extra because I think it would be very handy for the operator of my system, hopefully this page can display several stats; something along the lines of: total amount of food sold that day, total amount of net profit made that day, item that made the most profit/ sold the most and more if I can think of them. Also fix any bugs that come along with adding these new features

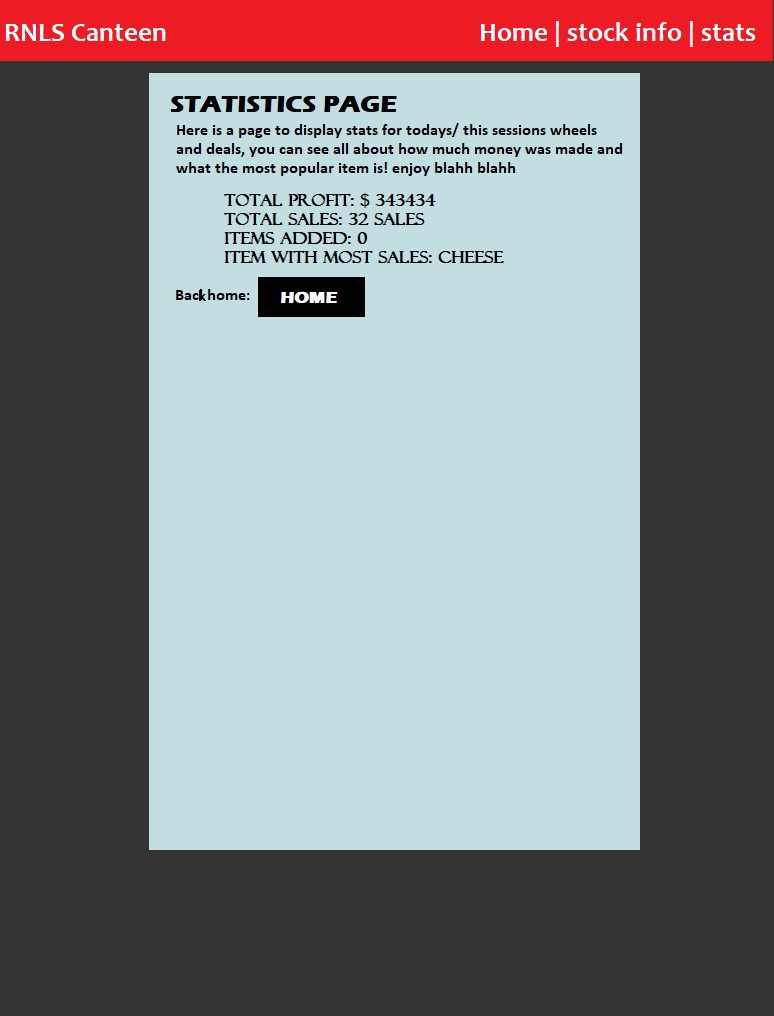
Task 1: Sketch interface design

*Draft a rough design for the interface that allows the user to trigger functionality in task 1*

Sell Item:

 View full sized in planning folder

Stats page:



Task 2: Identify any classes required

*Explain what the class will represent, plus listing what information will be stored in the class and any functions the class will have.*

No new classes added this version

Task 3: Identify information to be displayed

*What information will the interface need to display to the user?*

**On the Statistics page (stats.html):**

A title and a small bio of the pages purpose.

I will create a couple new variables that called something along the lines of total\_sold, total\_profit, popular\_item, after these variables are calculated they will be returned and displayed on the /stats page

**On the Sell item page(sell-item.html):**

There will be a small bio about what the function of the page is, like all pages have, a title. There will be the name of the food that is being sold (food.food\_item) and the current stock level of that food item (food.stock) so the user knows how much they can sell. And finally a input box that only takes numbers

Task 4: Identify user inputs

*What program functions can the user trigger through the interface?*

Will take 1 input in this version, the amount being sold. So, the input must only be a number and not be negative, this is found on the sell item page. Then that is passed to the python code will be called inp\_sell\_item, the python code will request that variable and store the value in a number called “sell\_amount” and the function sell\_item\_success will handle that value

Task 5: Identify any constants or existing data if required

No new constants required

Task 6: Identify indexed data structures

No new index data structures created

Task 7: Determine what calculations are necessary

*Write out the calculations the program will have to compute.*

Multiple new calculations will be added on both pages, they are listed below:

**On the sell items page:**

Calculation to decrease stock and increase sold amount

- food.stock = food.stock – sell\_amount

- food.sold = food.sold + sell\_amount

**On the stats page:**

Calculation to find total items sold, will be in a for loop to cycle through each item

- total\_sold = total\_sold + food.sold

Calculation to find total profit, each item sold amount times it price.

- total\_profit = total\_profit + food.sold \* food.price

Calculation to find the most popular item (by sold amount), an if statement that checks if the next value is larger than the previous

- if food.sold > most\_popular\_sold

Task 8: Develop a modular structure for your program

*Describe any functions that the computer program will have, identifying any sub-functions where required.*

This pseudo-code will only cover the code added in Ver3:

**SELL ITEM PAGES:**

In this route(‘/sell-item/<food\_id>’) view the page named sell-item

Define a function called “sell\_item”(passing the parameter(s): food\_id):

Convert the variable “food\_id” to an integer

Create a variable “found\_food” but give it no value (None)

For each of the items in the dictionary “food” do this:

If the item id is equal to the “food\_id” do this:

Set found\_food to that item

Set variable “data” to a dictionary where the item equals found\_food

Return the variable called “data”

In this route(‘/sell-item-success/<food\_id>, using method “POST”)

View the page “sell-item-success”

Define a function called sell\_item\_success (passing the parameters(s): food\_id)

Convert the varable food\_id to an int

Request input “inp\_sell\_item” give its value to a variable(“sell\_amount”)

Convert the variable to an int

For all the food items in “food” do this:

If that fooditem has the same id and “food\_id” then:

Set the item to the varible “found\_food”

Set found\_foods attribute of stock to itself minus “sell\_amount”

Set found\_foods attribute of sold to itself plus “sell\_amount”

Create dictionary called data, with food\_found in it

Return that data.

**STATS PAGE:**

In this route (‘/stats’) view the page named “stats”

Define a function called “stats” that does this:

For every food object In the dict “food” do this:

Create “total\_profit”, total profit is the food objects price x the sold

Create “total\_sold, total sold is every food objects sold attribute plused

If food objects sold amount is greater than most\_popular sold amount:

Set most\_popular to food object

Create a dict called “data” that holds these numbers

Return the dict “data”

Task 9: Define the functions identified

*Describe the functions for both the main program and any classes in terms of input and/or output where required. You may choose to do this with flow charts or pseudo-code (not Python code!). Add in additional steps or explanations using sequential, conditional, iterative statements where required. Identify global and/or local variables.*

**sell\_item and sell\_item\_success:**

sell\_item is just a page that takes an input, not much needed, the code in the function will assign the food an id so it knows what value it will change in the “food” index data structure, very similar to how re\_stock\_item function works, just a different situation. The sell\_item\_success function requests the input and uses the calculations (mentioned above; decrease stock, increase sold amount) to adjust the values in “food”.

**stats:**

A function that when is run, will calculate the statistics for the current session, will create a couple empty variables and add the collective amounts of sold, then from sold will calculate the profit, and the most popular item.

Task 10: Address any relevant implications such as usability, functionality, legal/ethical requirements.

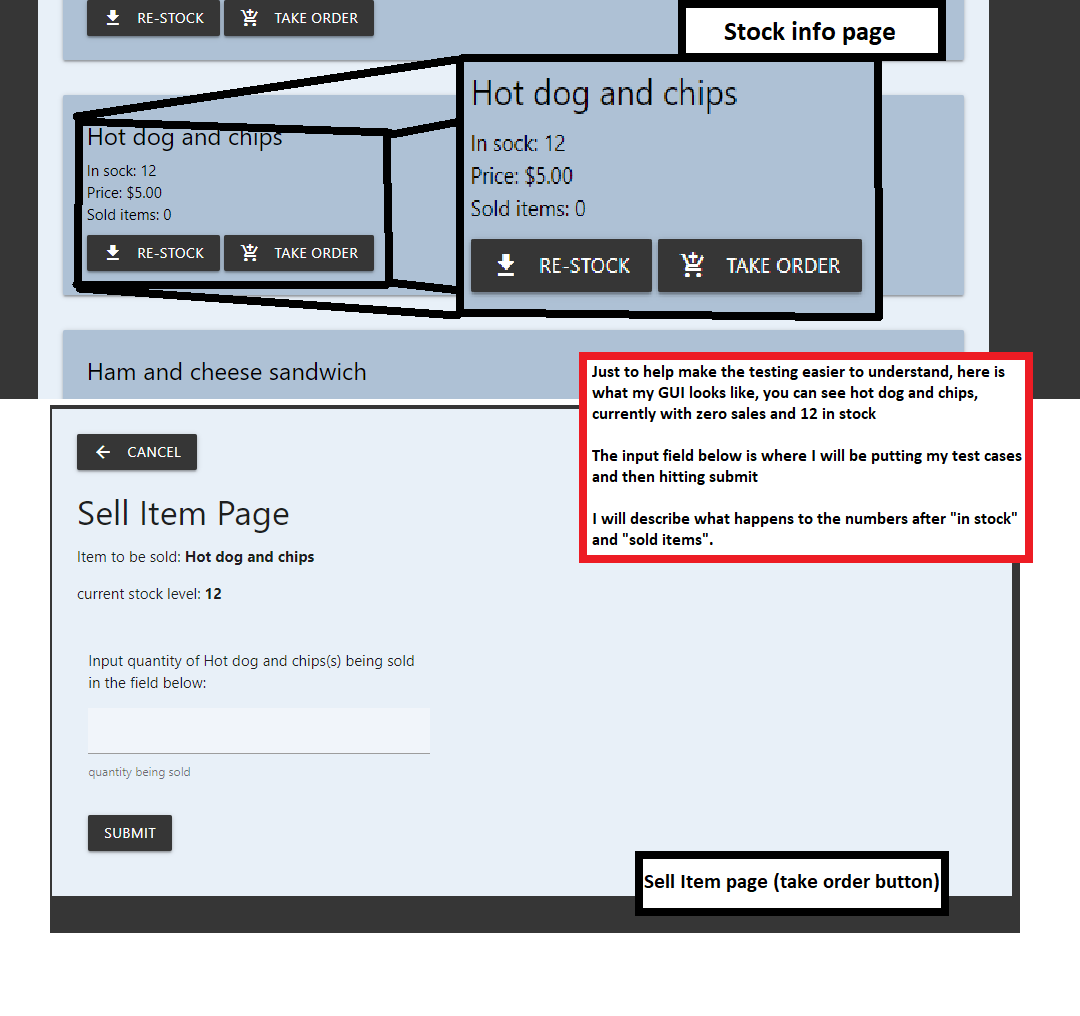
When designing my website, I will take into account the implication of usability and functionality, meaning buttons will be labelled, a colour scheme that is easy to look at, the layout will make sense and be very basic, a school canteen website does not need to be complex, Because I want kids or people who arn’t so good at computers to be able to run my program. The website should not breach and legal and ethical requirement, it will follow copy right laws, won’t be offensive etc..

Task 11: Document test cases for testing the program

*Document any testing that can be used to test your program. If any input is inputted using the keyboard, describe the expected input, plus any exceptional, boundary or invalid cases.*

**Sell Item page testing:**

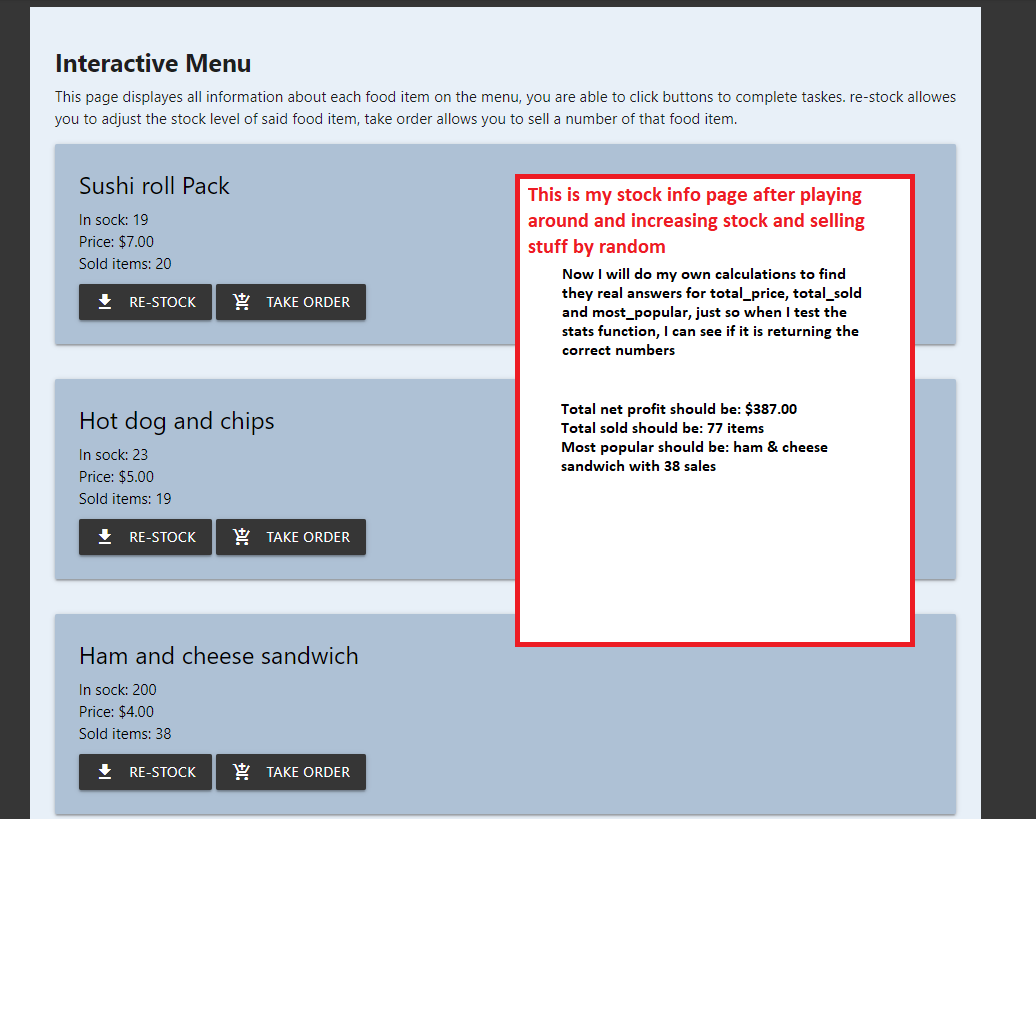
A screenshot below describes what I will be testing on the sell item page:



|  |  |  |  |
| --- | --- | --- | --- |
| Test case | expected outcome | actual outcome | Test case fix |
| **Sell item page**  Input test case 1:  Normal: 5  Input test case 2:  Large number: 100  Input test case 3:  Negative -10  Input test case 4:  Invalid case: four | Expected outcome 1:  To work as normal, decrease stock amount by 5 and increase sold by 5  Expected outcome 2:  Will break my system, input larger than in stock    Expected outcome 3:  Will break my system, negative numbers will mess with the calculations  Expected outcome 4:  Will give a valueError | Actual outcome 1:  Worked fine  Actual outcome 2:  Caused stock to go into negative (-88) and made sold amount go up to 100, this needs a fix  Actual outcome 3:  Made my stock increase by 10, (22) and sold amount go down by 10 (-10)  Actual outcome 4:  ValueError | Fix 1:  No fixes needed  Fix 2:  Explanation to large, look in refine plan for: “sell item page fix 3”  Fix 3  Fix is simple, negative numbers for selling item is completely impossible, to prevent this I added a feature to my html “min=’1’”:    Fix 4:  Set the type to number:    And I added a try/expect ValueError just for extra protection: |

**Stats page testing:**

There is not much I can test here, because it doesn’t involve any input, and the code it solid. But ill try my best:



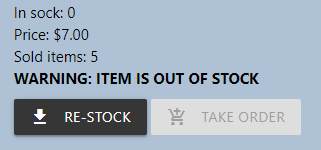
|  |  |  |  |
| --- | --- | --- | --- |
| Test case | Expected outcome | Actual outcome | Test case fix |
| **Stats page**  Test 1:  Expected: I restock a couple things and sold some items at random (shown in above screen shot)  Test 2:  Unexpected: someone clicks on the page without selling any stock? | Expected outcome 1:  My expected outcome is shown in screenshot above  Expected outcome 2:  All values will be zero | Actual outcome 1:  The actual outcome is the same as predicted, proof in document testing  Actual outcome 2:  Error because they don’t exist | Fix 1:  No fix needed  Fix 2:  To fix this I created empty variables at the top of the function, screen shot will be shown in “task 12, stats page fix 2” |

Task 12: Refine the plan

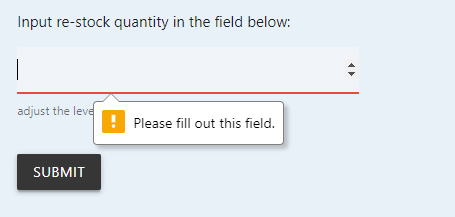
*Note any modifications here when iterating through the development cycles.*

Small things added during the development process in this version:

* The take order button will “disable” if stock for that item is equal to zero



* added information at the top of stock info page
* learned about a feature in html called “required” for input fields, I added that for most input fields as well because it just adds another layer of error prevention on top of the try/expect value errors I already have in place in my python code. Required feature just won’t let you submit until all fields are filled in:

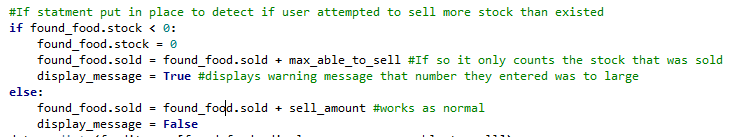


**Sell item page fix 3:**

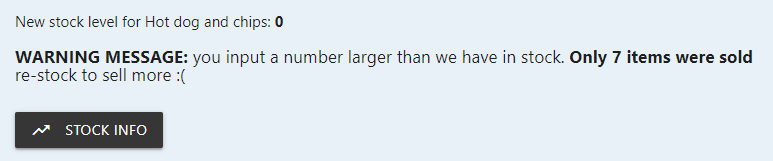
The problem I was having here was to do with a larger number being sold than what was in stock, resulting in stock value going into negatives and sold amount being larger than what is possible for stock levels. **The fix**: before the calculation to decrease stock is made, I create a variable called “max\_able\_to\_sell”:



Then I add an if statement that detects if found\_food.stock has gone below zero, If the stock has gone below zero, then the number they inputed was larger than stock, this if statement is the way my python code detects this problem, now it is able to identify the problem. First thing it does it sets stock back to zero, then this is where the variable “max\_able\_to\_sell” comes in, instead of increasing sold amount by the number they inputted (like it normally would), it will only increase sold items by “max\_able\_to\_sell” number:

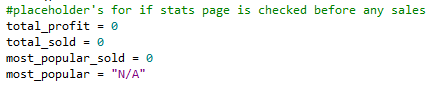


This will always cause the system to balance, more items cant be sold than what has previously been in stock, display\_message is another thing I added witch will tick the boxes for making the user aware of any errors that have occurred, if the message does come up it will just explain that they attempted to sell more than what was there and that only a certain amount was sold:



**Stats page fix 2:**

I had a problem of people checking stats page before it even had any values to work with, html would throw an error when it couldn’t find “most\_popular” because it didn’t create one, so I added this:



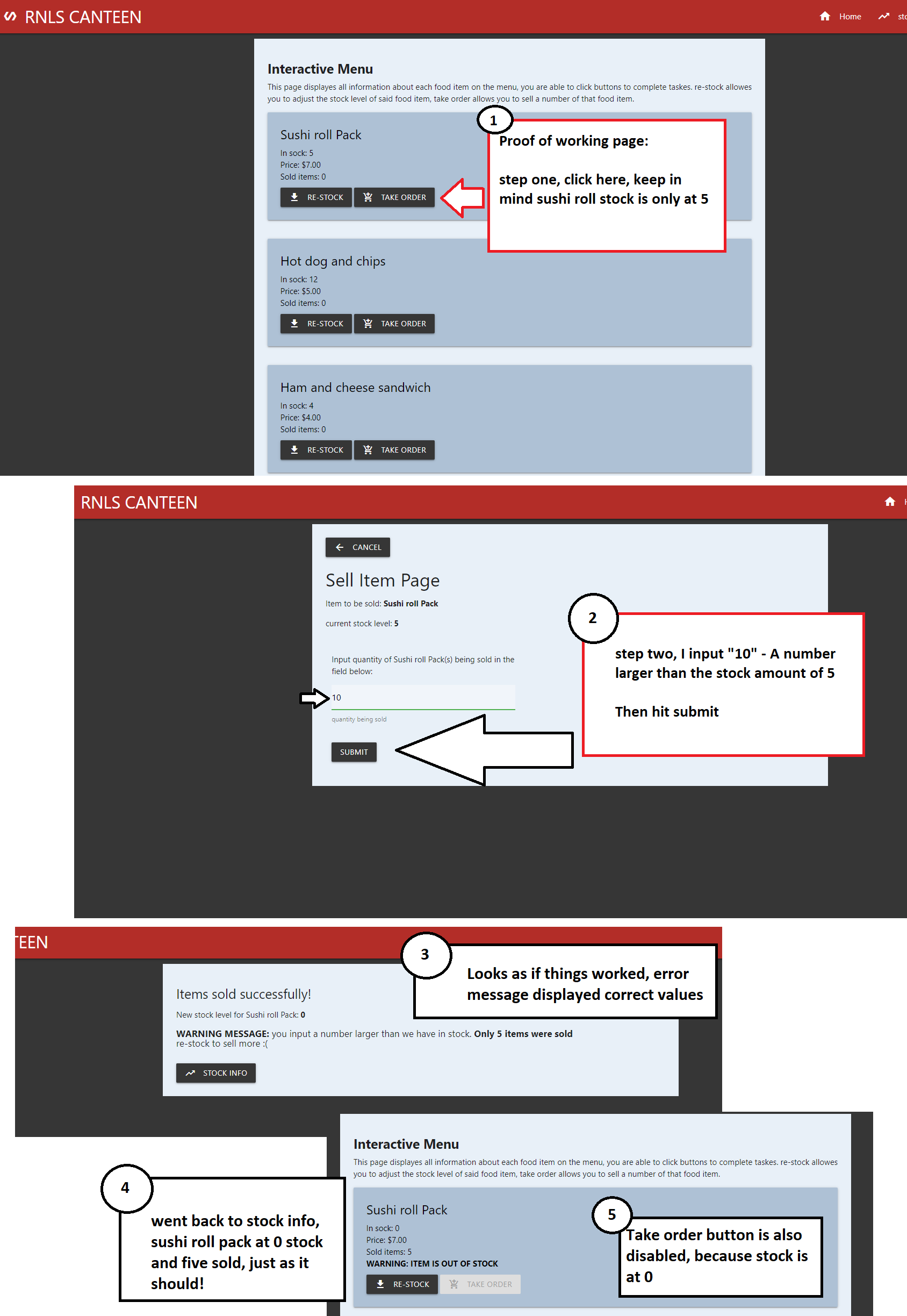
Proof of this working task 13.

Task 13: Document testing

*Show screenshots of your program working with descriptions of each image. These images should test the tests cases listed above.*

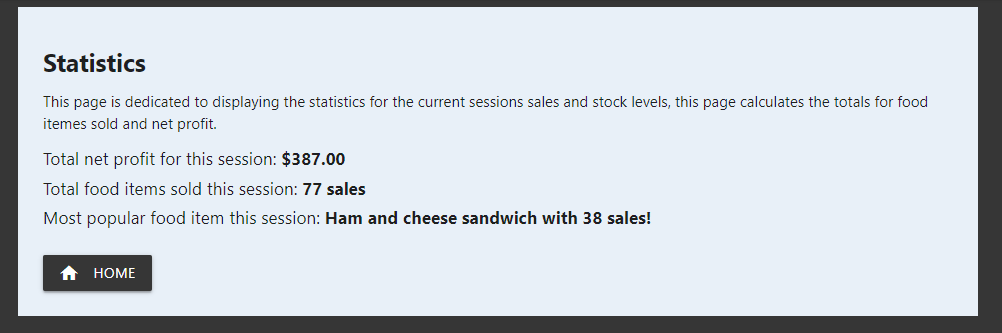
**Sell item page proof:**

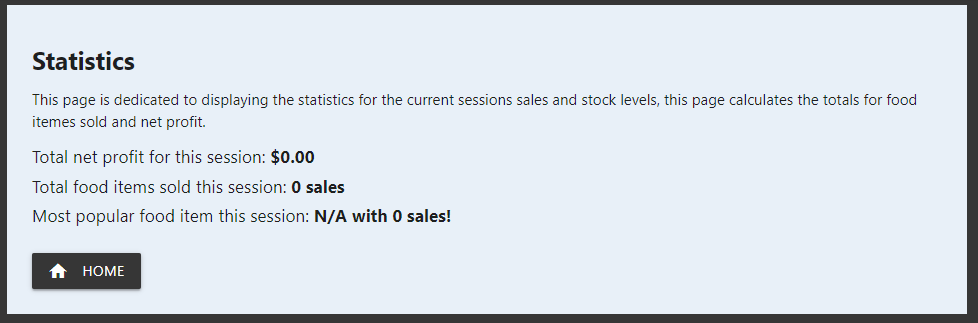
Proof that my fix for test case 2 works:



**Stats page proof:**

**Test case 1** actual outcome, all the numbers were correct:



**Test case 2** after fix, everything works now with no data to collect:  


Task 14: Evaluation

*How did your version turn out?*

I’m very happy with this Turn out, now my website has all the functions needed, I am 95.3% sure that there is no way to break it just by clicking buttons and inputting wrong values. Everything now requires an input before submitting, numbers can now only be input to fields that require numbers, A min max has been set for a few fields to stock negative values, it should be unbreakable by this point. Throughout the design of this website I have had a goal to keep it simple and easy to look at, when picking colours and layout, I kept in mind the 10 heuristics for user interface design, my system is a canteen stock/menu processor, It needs to remain simple so anyone young and old can run it with little to no instructions. (I hope) It fills in all the requirements for this assessment, and it has extra features for example the stats page and new items page.