Ver2.0 Planning document – school canteen – Jeremy Roberts

This document is planning for all version within Ver2

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.

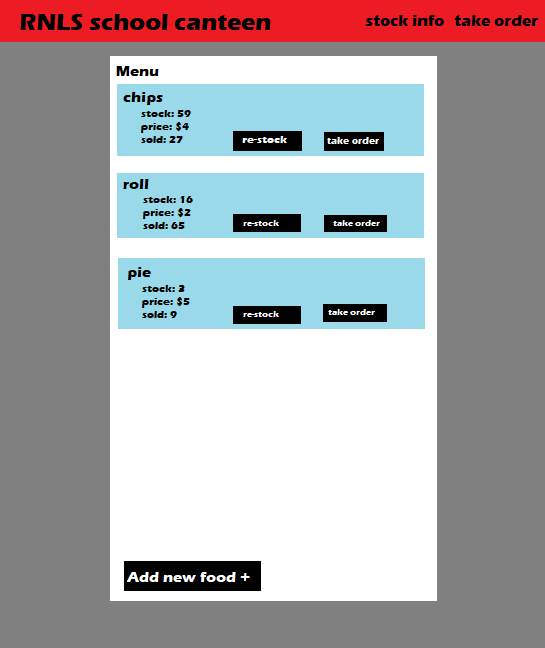
**This version:** This is going to be a large version, in this version I will add the features to manage stock levels and add new items, primary focus around a page that is going to display cards (stockinfo.html), each card will display the info of each food item (name ,price, amount sold, amount in stock). It is pretty much an interactive menu.

Task 1: Sketch interface design

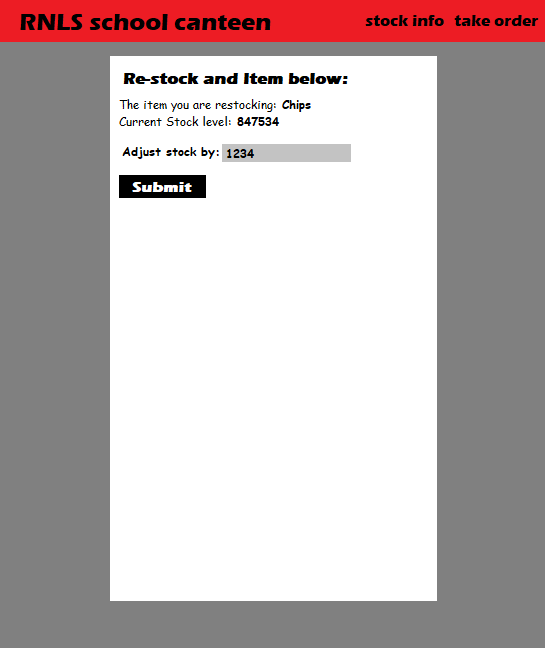
*Draft a* ***rough*** *design for the interface that allows the user to trigger functionality in task 1, while also annotating where the information in task 2 will be displayed. Create another sketch listing the interface widgets used to create the interface.*

To see Full sized images, look at the png files in the Task 1 sketches folder

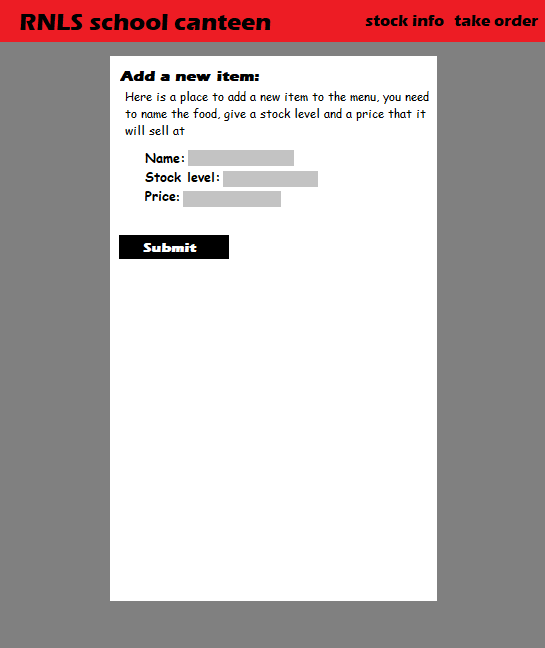
Stock info page:



Re-stock page:



New item 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 needed

Task 3: Identify information to be displayed

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

There will be:

* A page that displays food item, stock level, price, food items sold. (stock info)
* A re-stock page for specific items
* A re-stock success page, confirms to the user that the action was successful
* A add a new item page
* A success page for the new item page, confirms to the user that the action was successful

Task 4: Identify user inputs

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

* The re-stock page will take any number from the user, this will “trigger” the re-stock-success function.
* The new item page takes 3 inputs, first is the name(string), then it takes a stock amount (integer) and price (integer), the function that the user will trigger is the new item success page.

Task 5: Identify any constants or existing data if required

No new constants created

Task 6: Identify indexed data structures

No new index data structures in this version

Task 7: Determine what calculations are necessary

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

There will be calculations for re-stock page.

It will be the user input plus the previous stock level

Something like:

Stock = stock + restock\_value

Calculation added while refining the plan:

This calculation was added to stop stock numbers from going into negative values

If stock < 0:

Then stock = 0

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 to be added in Ver2:

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

Define a function called stock\_info():

Set variable “data” to the dictionary (index data structure) called “food”

Return the variable called “data”

In this route(‘/re-stock-item/<food\_id>) view the page named “re-stock-item”

Define a function called re\_stock\_item(passing 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(‘/re-stock-success/<food\_id> using the method called “post)

view the page named “re-stock-success”

Define a function called re\_stock\_success(passing parameter(s): food\_id):

Convert the variable “food\_id” to an integer

Set variable “restock” to “inp\_re\_stock which is requested from the html

Convert the variable “restock” 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

In the dictionary change “found\_food”s stock to itself plus “restock”

Return the variable called “data”

In this route(‘/new-item)

view the page named “new-item”

Define a function called new\_item():

Pass as no functions are needed

In this route(‘/new-item-success, using the method called “post)

view the page named “re-stock-success”

Define a function called new\_item\_success:

Request from html “food\_item” and set it to “food\_item”

Request from html “stock” and set it to “stock”

Request from html “price” and set it to “price”

Set variable “sold” to zero

new\_item = list containing(food\_item, stock, price, sold)

Append new\_item to dictionary called “food”

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.*

**Stock info:**

Returns data from “food” so that the html page(stockinfo) can display information that is in the dictionary. No other processes in this function

**Re\_stock and re\_stock\_success:**

Pretty much just cycles through the items in the dictionary “food” to see what “food id” matches what item that the user clicks on, so it can later on identify what value it will change within the dictionary, once it found the correct item, it will be called “found\_food”, then with that found\_food, the stock is adjusted accordingly to what the user input.

**new\_item and new\_item\_success:**

No process needed on new\_item page function, nothing happens until the submit button is pressed and redirects you to the new\_item\_success page and function. In the new\_item\_success page it requests variables from the html page and sets them to variables in the python respectively, then appends these variables to a list and adds them to the dictionary “food”.

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.*

|  |  |  |  |
| --- | --- | --- | --- |
| Test case | expected outcome | actual outcome | Test case fix |
| **Re-stock page:**  Input test case 1:  Negative number: “-5”  Input test case 2:  string: “test”  Input test case 3:  normal number: “5”  Input test case 4:  Large number:”1337” | Expected outcome 1:  the Negative will decrease the stock  Expected outcome 2:  the string will give an error  Expected outcome 3:  the normal number should work as normal  Expected outcome 4:  the large number should work as normal | Actual outcome 1:  decreased the number of stock: -5  Actual outcome 2:  gave an error  Actual outcome 3:  worked as normal, stock:5  Actual outcome 4:  worked as normal, stock: 1337 | **->**I will add an if statement that detects if stock has gone below 0 and set it back to 0.  I decided to make it let you input a negative number, This could be a handy feature for when stock is not sold but disappeared another way, like being damaged or stolen.  **->** I changed something in the html code that won’t even allow the user to input anything else than a number: type=”number”  This is built in feature of materialize  ->no fix needed  ->no fix needed |
| **New item page**  *Here I will simplify what inputs are just so it is easier to understand in this table*  *F = fooditem*  *S = stock level*  *P = Price*  *On the new-item page, there are three input fields.*  Input test case 1:  F = “egg sandwich”  S = “13”  P = “5”  Input test case 2:  F = “123!@#abc”  S = “123!@#abc”  P = “123!@#abc”  Input test case 3:  F = (nothing)  S = “-15”  P = “-15” | Expected outcome 1:  Will work as normal for all inputs  Expected outcome 2:  It will give a ValueError because my python code only accepts numbers for stock level and price  Expected outcome 3:  Will work normal, there will be no name (empty string), stock level and price will be negative though | Actual outcome 1:  Worked as expected  Actual outcome 2:  Gave a value error in the python code  Actual outcome 3:  Worked as expected, however the price and stock level shouldn’t be able to go below zero, a fix will be made. | Fix 1:  No fix needed  Fix 2:  Same fix as above ValueError issues, added a “type=number” a built-in feature of materialize. I will also add a try/expect ValueError in my python.  Fix 3:  An If statement, sets stock or price to zero if it is input less than that. |

Task 12: Refine the plan

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

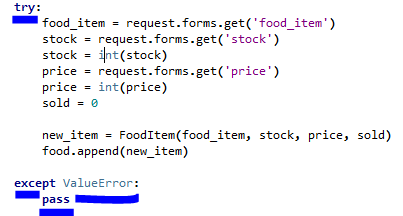
* Added a cancel button to the new item and re-stock pages, for ease of access. This was added because I noticed that with my design, there was no simple way to leave the area without pressing submit.
* I added calculation to stop stock levels from entering negative numbers in both new\_item\_success and re\_stock\_success:



* Using a feature from materialize in my html code, I was able to set input fields to only. allow certain values, for example I only allow number inputs for re-stock, price and stock. Adjustment underlined below:



* Added a try/except Value Error with the inputs in the new\_item\_success function to prevent unwanted inputs:

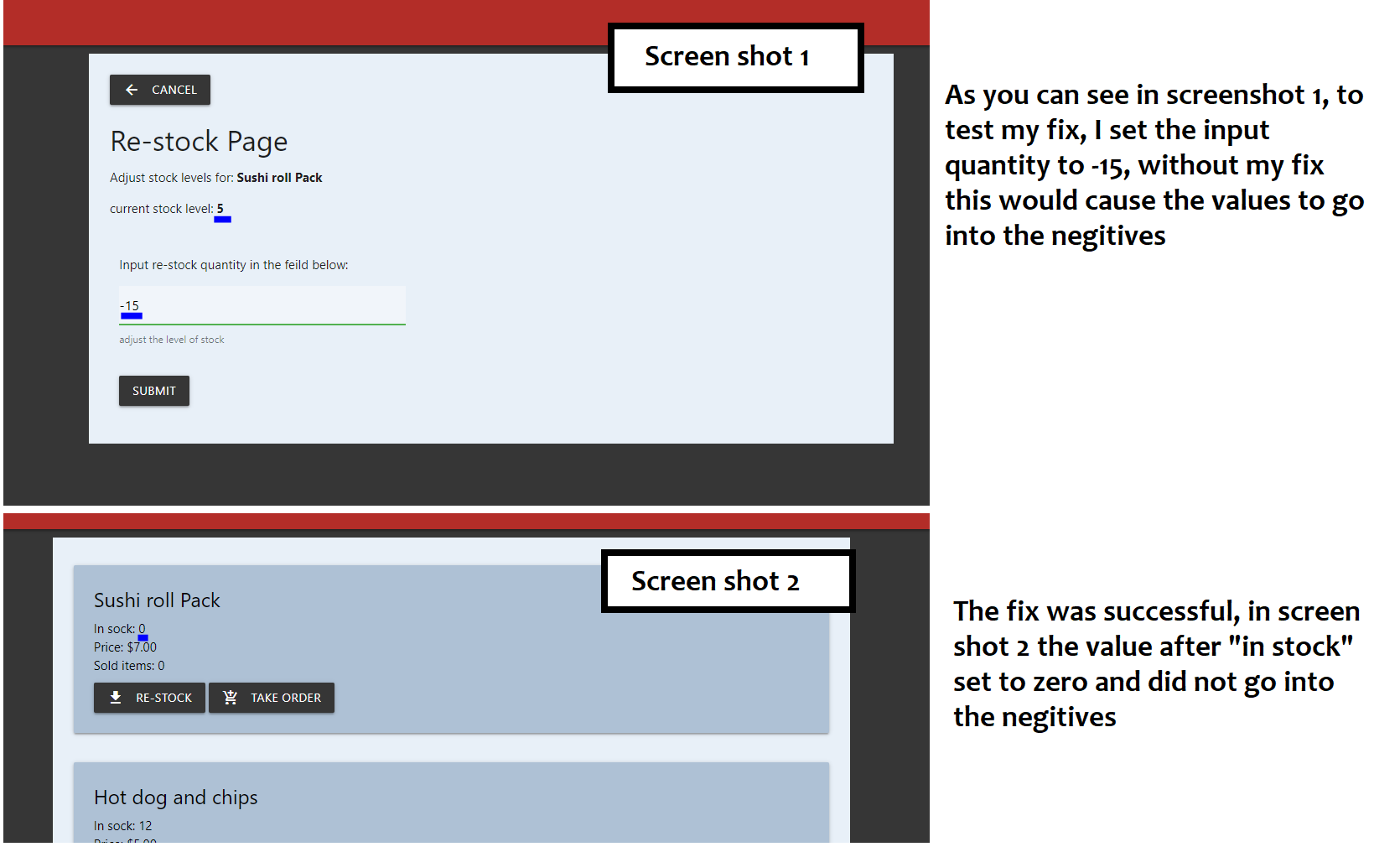


Task 13 on next page

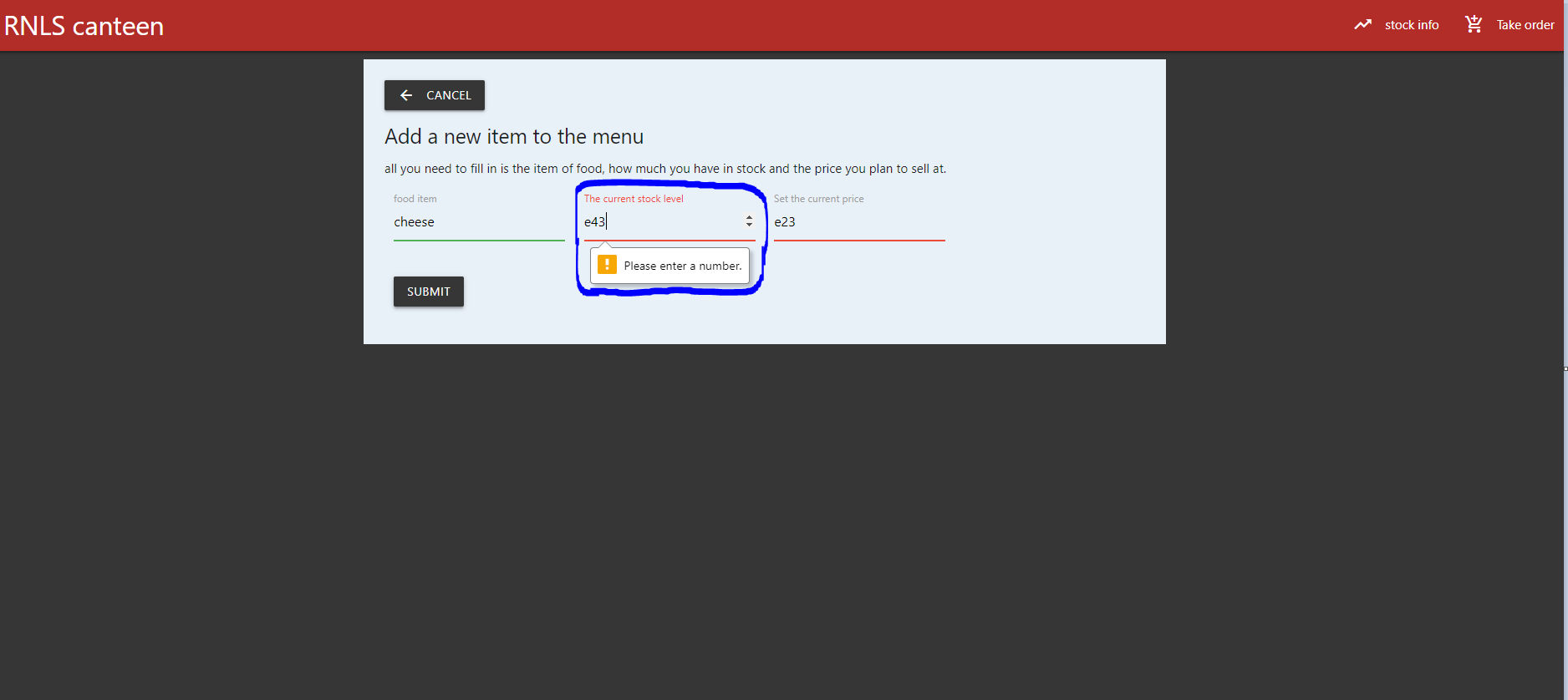
Task 13: Document testing

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

When inputting a negative number that is larger than existing stock:



Fix tested for when anything other than numbers is inputted into the price and stock field:



Task 14: Evaluation

*How did your version turn out?*

One problem I came across was the possibility of the user inputting negative values in the re-stock and new items page, all necessary fixes were made to deal with un-expected inputs, a lot was added to do with displaying and managing the stock levels in this version, I am happy with the turn out.