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

In version 0.4 I will make each product be able to be purchased. This will mean inputting name and email and out dates into the system. It should then calculate the cost and display the correct information after a successful purchase.

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.*



Task 2: Identify any classes required

*Bro class – holding all information for each bro*

Task 3: Identify information to be displayed

*Product page:*

*A card for each bro showing their name, photo, description and stock*

*Purchase page:*

*Just the forms that the user fills out with their own information*

Task 4: Identify user inputs

*Name, last name, email and return date*

Task 5: Identify any constants or existing data if required

*Test Data will be created as well as a list of month words with corresponding numbers*

Task 6: Identify indexed data structures

*bros – Holds all Bro objects*

*months – Holds all month words with number keys*

Task 7: Determine what calculations are necessary

Cost:

\*Return date – Current date) \* bro.cost

Task 8: Develop a modular structure for your program

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

The only functions being created are the page functions containing page instructions and routing, and Static File import functions giving file directions.

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.*

Import bottle functions

Import datetime functions

Create Bro class:

Create constructor method passing name, description, image link, cost, stock and booked details(set to “” by default):

*Set all self.variables to passed varaibles*

*Self.name = name*

*Self.description = description*

…

Create months dictionary with month names as the key and the month number as the data

*‘Jan”:1,*

*“Feb”:2,*

…

Create bros class holding all bro objects:

Fill with test data

*Bro("Tom","…”,” tom.jpg", 970, True),*

…

Create index page function and routing using (‘/’)

Create product page function and routing using (‘/products.html’)

Return bros array to page to be displayed

Create purchase page function and routing using (‘/purchase.html’) with “name” as a parameter passed by the link

Create variable found\_bro

Loop through every bro in bros array

Check if parameter name is the same as bro.name

Set found\_bro to this bro

Set current\_bro to found\_bro

Return current\_bro to page to be displayed

Create purchase\_success page function and routing using (‘/purchase\_success.html’) with method “post”

Collect all form data and store in apporopriate variables: Fname, LName and date\_

Set variable Curr\_date to current date using dateTime function

Set variable difference to the result of (date\_ - curr\_date)

Set variable total\_cost to the resilt of (difference \* current\_bro.cost)

Set found\_bro stock to False

Set current\_bro booked details to the Fname, Lname, current date, date\_ and total\_cost

Return current\_bro to page to be displayed

Route images from folder “./Images” using route “/img/<filename>”

Route Css files from folder “./Css” using route “/css/<filename>”

Route Script from folder “./Script” using route “/script/<filename>”

Call ‘run’ function passing port 399

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

The user will be giving their name and email, so it needs to not be displayed to other users for privacy purposes. Controls must be obvious as page won’t have instructions. Buttons must be clearly labelled.

The form pages are separate allowing the text boxes to be large and easy to click on. The form boxes highlight green when you are editing them clearly identifying to the user what is going on. If they do not fill out a required box or input wrong values the box whill highlight red asking the user to fix the issue. The purchase button is large and obviously centered in the page while being highlighted blue so that the user has no issue finding the button.

The date selector opens a calender that is intuitive and designed after the common style of callenders. The current date is highlighted making it easier for the user to find the date they need. Once the user clicks a date a large circle goes around it and the date on the side changes clearly displaying the date selected.

The success page is simple and displays only the key information in a simple design keeping to the simplistic theme of the page.

If any error occurs the page will display an error message taking the user back to the last safe page (Home page is gurranteed not to crash as no python is running)

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.*

Try inputting numbers for names and email

Try selecting passed dates

Try selecting current date

Try different months

Task 12: Refine the plan

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

Success\_page did not work until I put “found\_bro.stock = False” into purchase page function

(No idea why this works)

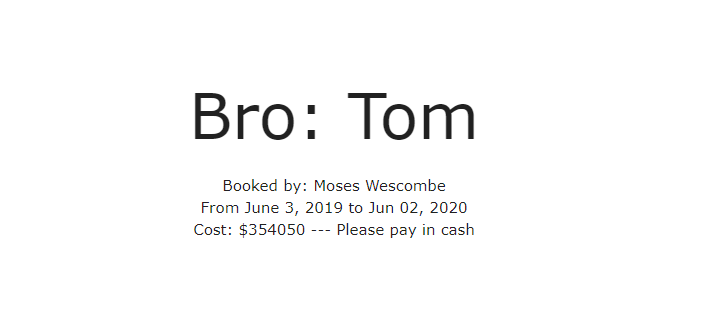
To fix the negative cost issue was hard. Because javascript isn’t working with Bottle there was no in page limiting that I could do. So the best I could think of was to simply make any previous days one year ahead.

if delta \* -1 == abs(delta):

d0 = date(curr\_date.year, curr\_date.month, curr\_date.day)

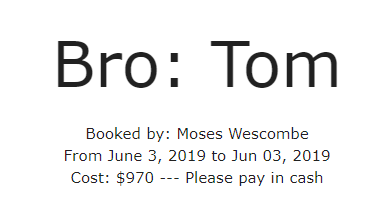
d1 = date(int(date\_alt[2]) + 1, int(MONTHS[date\_alt[0]]), int(date\_alt[1]))

delta = d1 - d0



To fix the current date selection issue I simply changed:

total\_cost = current\_bro.cost \* delta.days ---> total\_cost = current\_bro.cost \* min(delta.days, 1)



Task 13: Document testing

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

|  |  |  |  |
| --- | --- | --- | --- |
| Test case | Expected outcome | Actual outcome | Solution |
| Numbers for name  1  8  4 | Accept because people can be named numbers. Ever seen Divergent? | Accepts the input | None |
| Numbers for email  234  54  0 | Accepts as long as there is an @ | Accepts as long as there is an @ | None |
| Selecting passed dates  (At least 3 different dates) | Not allowed to select passed days | You can select passed days | If it is a passed date add +1 year to the cost heheheeheh |
| Select current date | Should display one days cost | Cost is $0 | Get the min(total\_cost, 1) when multiplying cost |
| Selecting different months  (1 of every month) | Should function perfectly | Only May works | Change months list to the 3 character versions because that’s the form that materialize uses |
|  |  |  |  |
|  |  |  |  |

Task 14: Evaluation

*How did your version turn out*

Turned out good, worked as I wanted it to with a few bug fixings.

Purchase is easy and smooth, cost is caluculated and comfirmation is displayed as required for this version. Ethical implications are met and the page is simple to navigate and intuitive to use.

Using this method I have indentified multiple errors that would have made my page