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

In version 0.7 I will fine tune and make as many changes as I can

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.*

N/A

Task 2: Identify any classes required

*Bro class – holding all information for each bro*

Task 3: Identify information to be displayed

*Nothing new*

Task 4: Identify user inputs

*Nothing new*

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

Nothing new

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

Create return\_product page function and routing using (‘/return\_product.html’)

Do nothing

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

Get form first name and store in Fname variable

Get form last name and store in Lname variable

Loop though every bro in bros array:

If current bros booked details matches fname and lname

Set found\_bro variable

Change stock

Return bro

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

Do nothing

Create application\_success page function and routing using (‘/application\_success’) with method “POST”

Save the form data in appropriate variables

Append a new bro with data from forms to bros array. Add “empty.jpg” as the default image

Return the new bro

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 in form pages, 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 input boxes highlight green when inputs are successfully inputed. They will also highlight when the user is editing in them to make sure the user easily understands what they are doing. If incorrect information is inputed or a required input is not filled it will highlight red.

The submit buttons are highlighted blue so that the user can easily understand the page and what to do.

The only default image is a stock siluhett photo that is not copy righted. This is important so that there is no legal issues displaying the stock image.

Error prevention methods (Try catch, or materialize functions/HTML5) prevent the whole website from breaking and instead guide the user with simple error messages back to safety.

Product page uses large pictures to draw the customers in (A picture speaks a 1000 words) the descriptions are well crafted making them more attractive.

The buttons on the product page are highlighted and pop out of the page making the usability easy and intuitive to use.

The user will be giving their name, 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. Error messages must accur when they are doing something that may mess up the process

Task 11: My Changes

*Information about each of the changes I have made in this version to improve the quality of the program*

Here are some errors that came up in the program that I have changed

|  |  |
| --- | --- |
| Error | Solution |
| Applicants may input a negative number in for cost. This causes errors later on | Check if cost is = to the absalute value of cost. If it isn’t then:  Return an error to the page  if cost != abs(cost):  return dict(bro = False)  Now the program will throw an error and ask you to use a positive cost value    EDIT: Found a simpler fix with using MIN=”0” in the html input |
| When purchasing if you put a past date then the cost becomes negative | Check if total\_cost is = to the absalute value of total\_cost. If it isn’t then:  Return an error to the page  if total\_cost != abs(total\_cost):  return dict(bro = False)  Now the program will throw an error and ask you to use a date >= todays date. |
| Total cost was always calculated as 1 days cost no matter how far ahead you booked | Simple mistake; I used min() instead of max() This meant it was getting the minimum value of 1 or the number of dates. Obviously this was always going to be 1. Max however reverses this, the only time it will be one is if delta.days is 0 |
| There was error when leaveing the form pages to go to other form pages. Could not find the cause | The most simple solution is to surround every page in try/except errors. This is a good solution for everything and vertually makes my page invincible to true crashes.  Inside each page function I started with try:.  Indent everything within this.  Then using except return false to the pages if failed.  Inside the html documents before loading the main body (Noit including navigation bar) I have a simple if statement to check that the value is not False before continuing. If it is then display an error message and a return button that goes to the home page. |

Task 14: Evaluation

*How did your version turn out*

I found a good number of fixes and now the page is more adaptable and flexible. Error messages are clean and tell the user why things went wrong and give a link back to where they need to be. Global errors are simple but prevent the website from breaking. Usability is simple and easy to navigate. Only the necessary functions are available preventing overcrowding of the website. Functionality is mostly errorless

As a website as a whole the usibility is simple and easy, colours are simply guiding the users on how to use the pages. Through the methods used in my planning I was able to identify as many errors as I could with expected and unexpected test cases and fix them. Users are not overwhelmed with options allowing a simple easy flow to the page however users in general would not feel limited in their options.

**Code:**

from bottle import run, route, view, get, post, request, template, static\_file #Import important bottle methods

from datetime import \* #Import date time

###Classes###

class Bro: #Bro class

def \_\_init\_\_(self, name, description, img, cost, stock, booked\_details=""): #Constructor

#Set bro personal variables

self.name = name

self.description = description

self.img = img

self.stock = stock

self.cost = cost

self.booked\_details = booked\_details

###Arrays###

MONTHS = { #List of months with their numbers to help convert the dates on purchase success page

'Jan' : 1,

'Feb' : 2,

'Mar' : 3,

'Apr' : 4,

'May' : 5,

'Jun' : 6,

'Jul' : 7,

'Aug' : 8,

'Sep' : 9,

'Oct' : 10,

'Nov' : 11,

'Dec' : 12

}

bros = [ #Test data filled with test bros

Bro("Tom","Generic british boi. Nice and smart so don't hire him if you don't want to feel bad about your IQ. Will colonise only if it brings glory to his queen.","tom.jpg", 970, True),

Bro("Jerry","Good boi, will definatly tell us everything you did wrong. Cops might like this one.","jerry.jpg", 90, True),

Bro("Moses","Slightly retarded, will do anything for you. Makes you feel better about your problems. Depressed people and bullies may want.", "moses.jpg", 20, True),

Bro("John","Goes to the gym.","john.jpg", 100, True),

Bro("Liam","Will serinade. Has guitar, good boi. Ideal for girls.","liam.jpg", 100, True),

Bro("Fox","Will sit and play nintendo with you for as long as you like. Presence makes you feel good. Introvertes love.","laimf.jpg", 78, True),

Bro("Dom","Will make you feel geneticaly inferior. Has good beauty products. Gays would buy again.","dom.jpg", 20, True)

]

###Pages###

#Index page

@route('/')

@view('index')

def index():

return dict(bro = True) #Return True so that the page does not throw an error

#Product page

@route('/products.html')

@view('products.html')

def products():

try:

return dict(bros\_list = bros)

except: #If any error accurs display an error message

return dict(bros\_list = False)

#Purchase page

current\_bro = None #This variable is used to easily pass the currently processed bro through pages, (Purchase -> purchase success page)

@route('/purchase/<name>')

@view('purchase')

def purchase(name):

try:

global current\_bro

#Find the bro by name

found\_bro = None

for bro in bros:

if name == bro.name:

found\_bro = bro

break

### Not sure why this works but somehow this fixes an error in my code that I cannot fix otherwise.

found\_bro.stock = False

found\_bro.stock = True

### I wont touch what works even if its odd

current\_bro = found\_bro #Set the object as current bro and return it to the page

return dict(bro = found\_bro) #Return found\_bro to page

except: #If any error accurs display an error message

return dict(bro = False)

#Purchase\_success page

@route('/purchase\_success', method = "POST")

@view('purchase\_success')

def purchase\_success():

try:

#Get form data entries

Fname = request.forms.get("first\_name")

Lname = request.forms.get("last\_name")

date\_ = request.forms.get("date") #(Format i think is (Month(first 3 characters) dd, yyyy)

#Format the date

date\_alt = date\_.split(" ")

date\_alt[1] = date\_alt[1].strip(",")

#Calculate the difference in date and then the total cost

curr\_date = datetime.now()

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

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

delta = d1 - d0

total\_cost = current\_bro.cost \* delta.days #Calculate total cost

if total\_cost != abs(total\_cost): #Check if cost is negative, if it is they must have selected a passed date. Ask them to repeat

return dict(bro = False)

total\_cost = current\_bro.cost \* max(delta.days, 1) #Make sure that the minimum cost is for 1 day

total\_cost = str(total\_cost)

current\_bro.stock = False #Change stock

current\_bro.booked\_details = [Fname, Lname, str(curr\_date.strftime("%B")) + " " + str(curr\_date.day) + ", " + str(curr\_date.year), date\_, total\_cost] #Store the booked details in the object

return dict(bro = current\_bro) #Pass object back into page

except: #If any error accurs display an error message

return dict(bro = False)

#return\_product page

@route('/return\_product.html')

@view('return\_product.html')

def return\_product():

return dict(bro = True) #Return True so that the page does not throw an error

#return\_success page

@route('/return\_success', method = "POST")

@view('return\_success')

def return\_success():

try:

#Get form data entries

Fname = request.forms.get("first\_name")

Lname = request.forms.get("last\_name")

#Find bro object by name

found\_purchase = None

for bro in bros:

if bro.stock == False:

if bro.booked\_details[0] == Fname and bro.booked\_details[1] == Lname:

found\_purchase = bro

break

if found\_purchase == None:

return dict(bro = False)

found\_purchase.stock = True #Change stock

return dict(bro = found\_purchase) #Return found\_bro to page

except: #If any error accurs display an error message

return dict(bro = False)

#Application page

@route('/application.html')

@view('application.html')

def application():

return dict(bro = True) #Return True so that the page does not throw an error

#Application success page

@route('/application\_success', method = "POST")

@view('application\_success')

def application\_success():

try:

#Get form data entries

Fname = request.forms.get("first\_name")

Lname = request.forms.get("last\_name")

description = request.forms.get("description")

cost = int(request.forms.get("cost"))

if cost != abs(cost): #Check if it is negative, if yes the return False so that the page gives an error, asking them to input a positive number

return dict(bro = False)

bros.append(Bro(Fname, description, "empty.jpg", cost, True, ""))

return dict(bro = bros[-1])

except: #If any error accurs display an error message

return dict(bro = False)

##Static files###

#Images

@route('/img/<filename>')

def server\_static(filename):

return static\_file(filename, root='./Images')

#Css files

@route('/css/<filename>')

def server\_static(filename):

return static\_file(filename, root='./Css')

#Script files

@route('/script/<filename>')

def server\_static(filename):

return static\_file(filename, root='./Script')

run(host='0.0.0.0', port = 399, reloader = True, debug = True) #Run local server