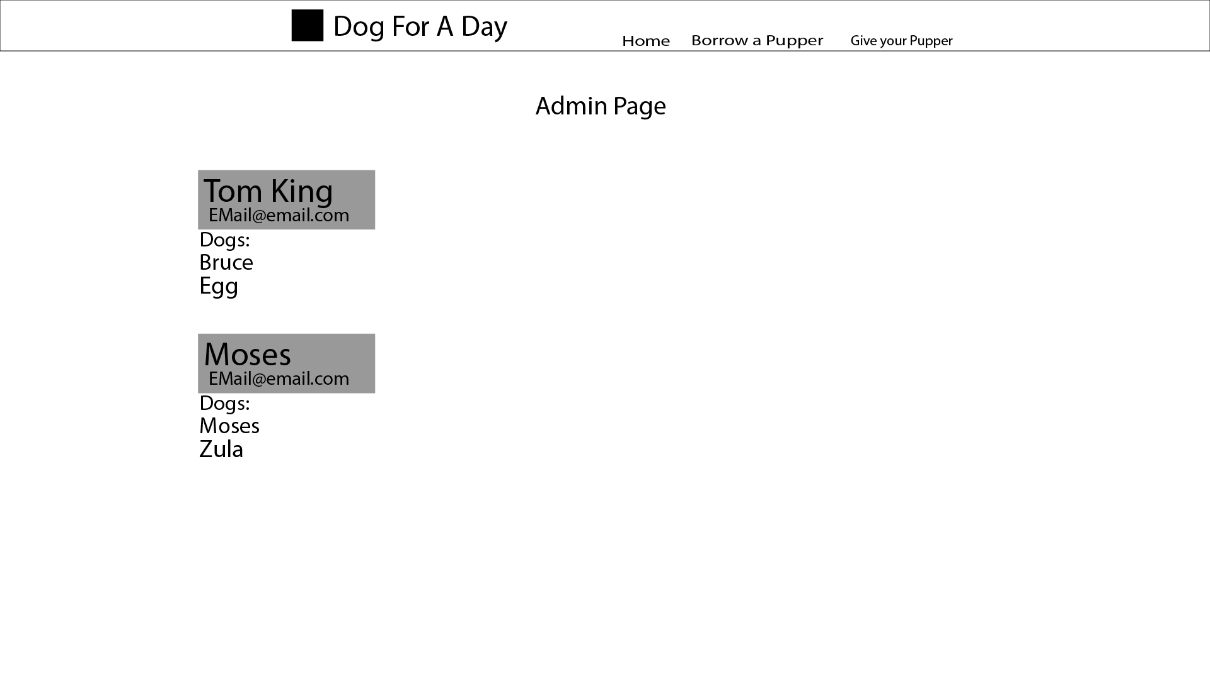
Version 8.0

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

I will create an “admin” page where the users and what dog they have rented out can be shown.

Task 1: Sketch interface design



This will be a simple page made for the admin. Thus, it requires no bells and whistles. It will follow the same design as the other pages

Task 2: Identify any classes required

Person class

Dog Class

Task 3: Identify information to be displayed

Persons name/email + dog name

Task 4: Identify user inputs

Name, email address

Task 5: Identify any constants or existing data if required

No constants for this version

Task 6: Identify indexed data structures

List called Persons made up of people objects with the variables name, email and dog

Task 7: Determine what calculations are necessary

None

Task 8: Develop a modular structure for your program

Set route to “/admin-page”

Set view to “admin-page”

Define admin\_page as:

Set a dictionary of humans form the person list

Return that dicitonary

Task 9: Define the functions identified

Admin page, which define the webpage

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

Within this version, I will need to create a website interface that is easy to read and simple to understand as many of the users may be older/unfamiliar with technology. I also need to follow the general rules of design when it comes to websites with colours layout etc. Buttons need to be clear and laid out, everything should make sense.

It need to be functional, it should first fulfil its purpose and secondly look aesthetically pleasing,

No copyrighted images. No illegal or explicit images etc.

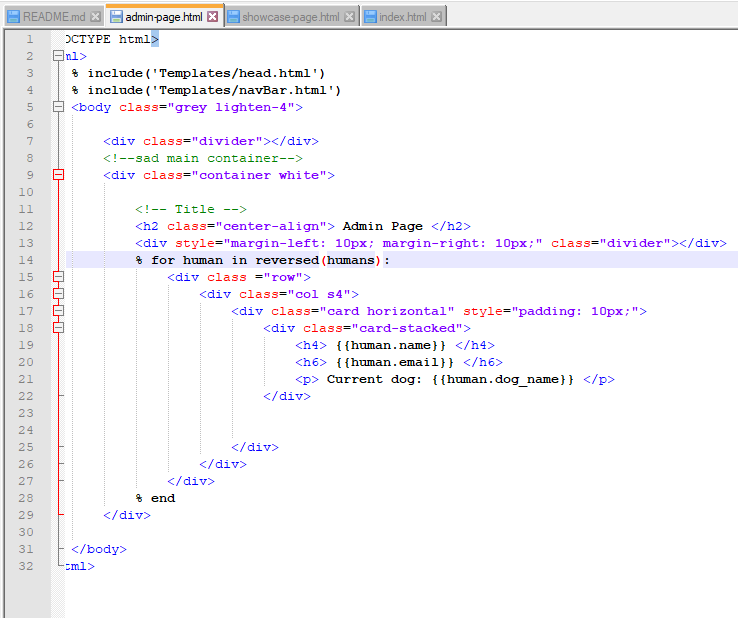
Task 11: Document test cases for testing the program

To test this program, first I will load localhost:8080/admin-page, to check the page shows up correctly. And then I will try taking out a new dog. And then seeing if the new renter turns up on the list or not. Then I will return that dog and check to see the dog doesn’t still appear on the list.

Task 12: Refine the plan

Everything worked first time surprisingly.

However, after consideration, I realised that over time, this list of past users would build up and up and you will have to scroll very far down to find the users that have a dog out currently. To do that I needed to reverse the list, so that the newest additions would be shown on the list first on the webpage.

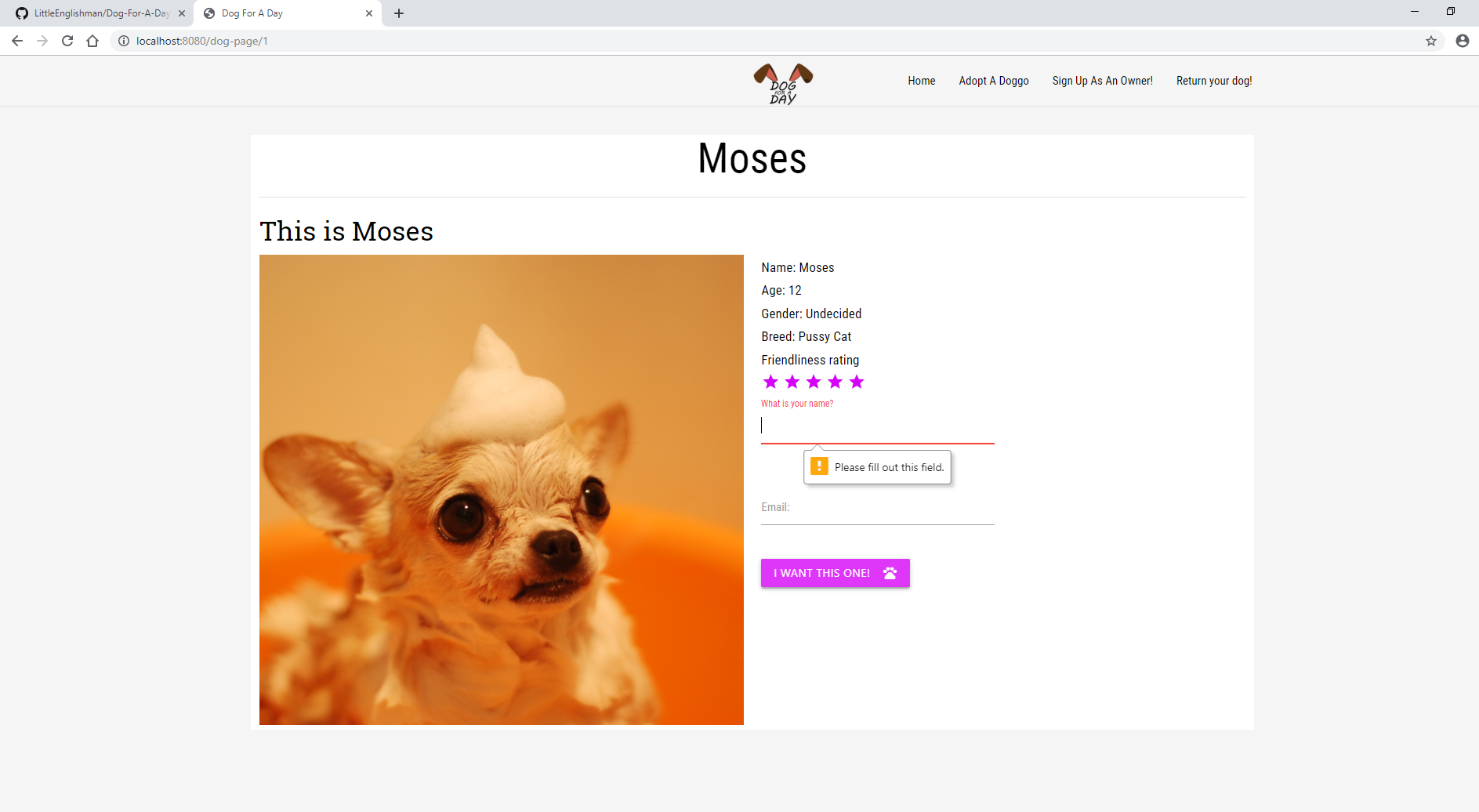


I did this by adding in the “reverse() code” that is selected above.

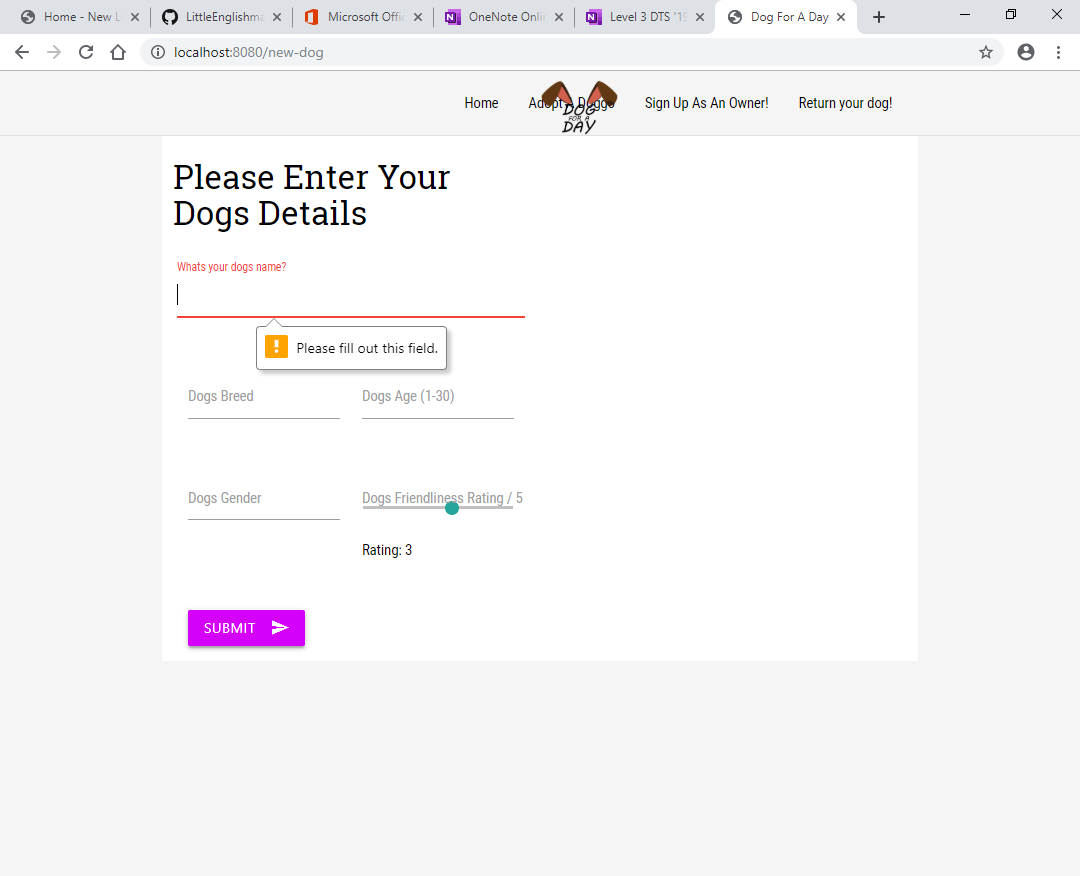
(<https://thispointer.com/python-different-ways-to-iterate-over-a-list-in-reverse-order/>) I found it on this website

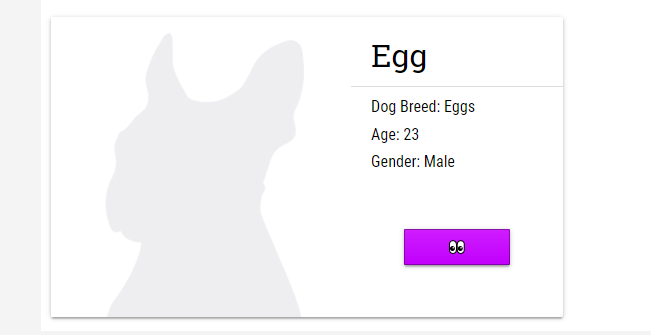
That fixed the problem as it know shows the list in “reverse order”.

Throughout my testing, I also found out that I could input an empty form, leading to an empty buyer taking the dog. To combat this, I added in the word “required” to each input in the form to make them necessary for the form to be submittable. This fixed the bug as forms are now not able to be submitted if empty



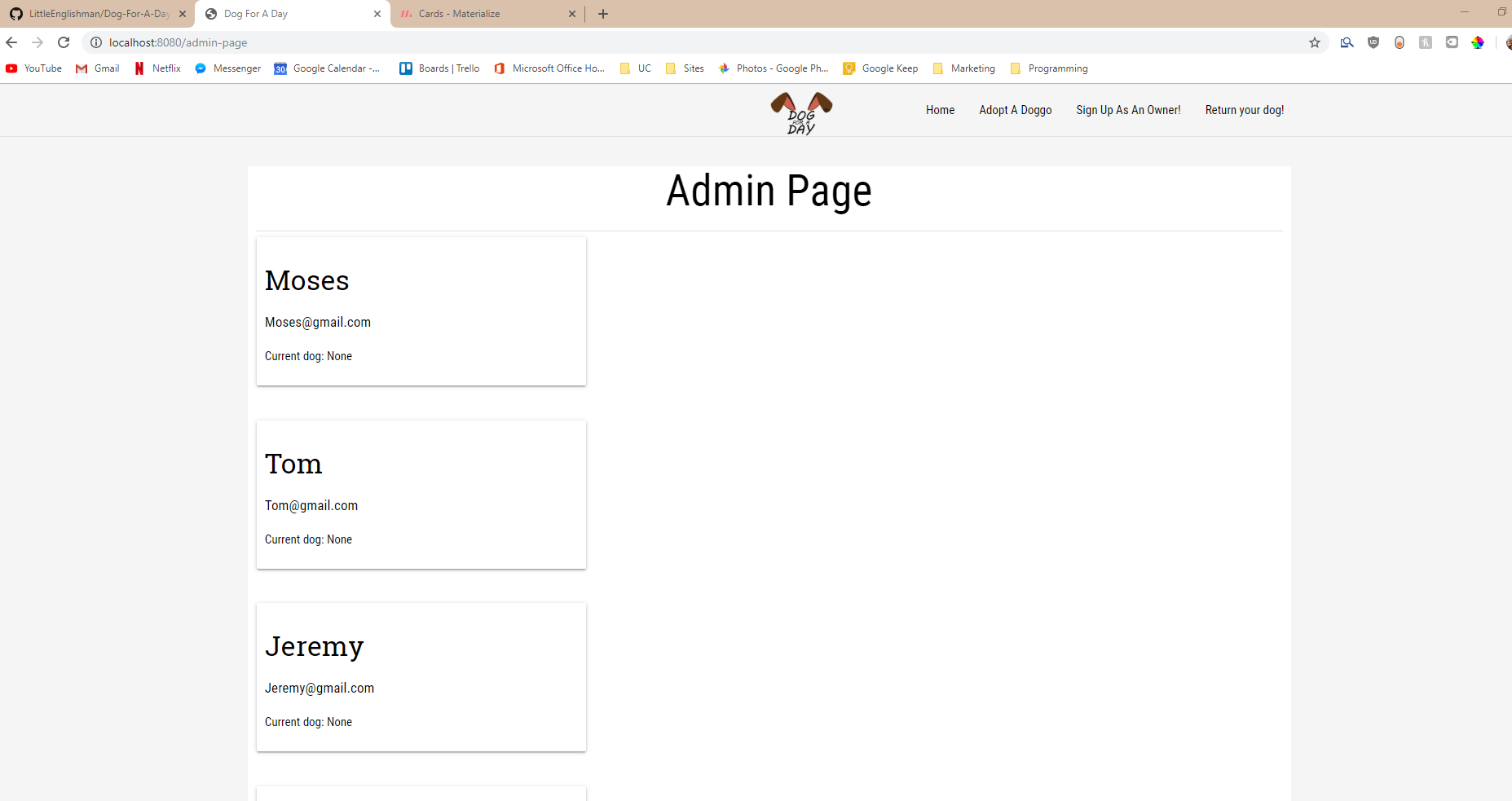
Here you can see the forms do not submit if empty as they show a relevant error message. Luckily most of this is built into materialize forms so I do not have to format it or code it in separately.



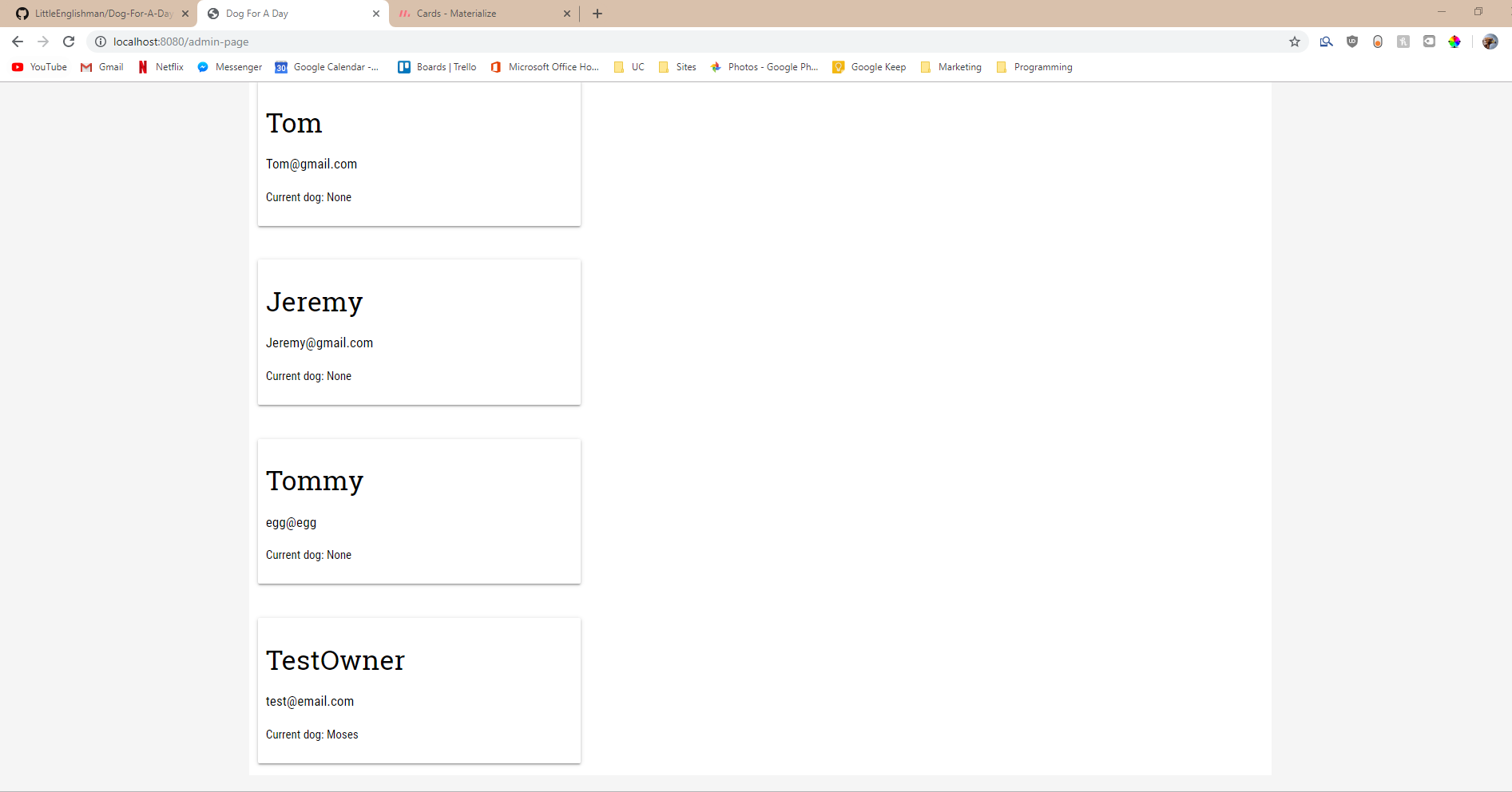
Another change I made to the program was to add in a placeholder image for new dogs. Due to lack of ability to add custom images. I had originally being using the image of the dog Bruce, for every new dog that I added. I realised that while this was functional, it was not true to the dog so a better image would be a place holder silhouette of a dog.

I decided the picture above was a good easily recognisable silhouette as a dog. Which is why I ended up choosing it as my placeholder image.

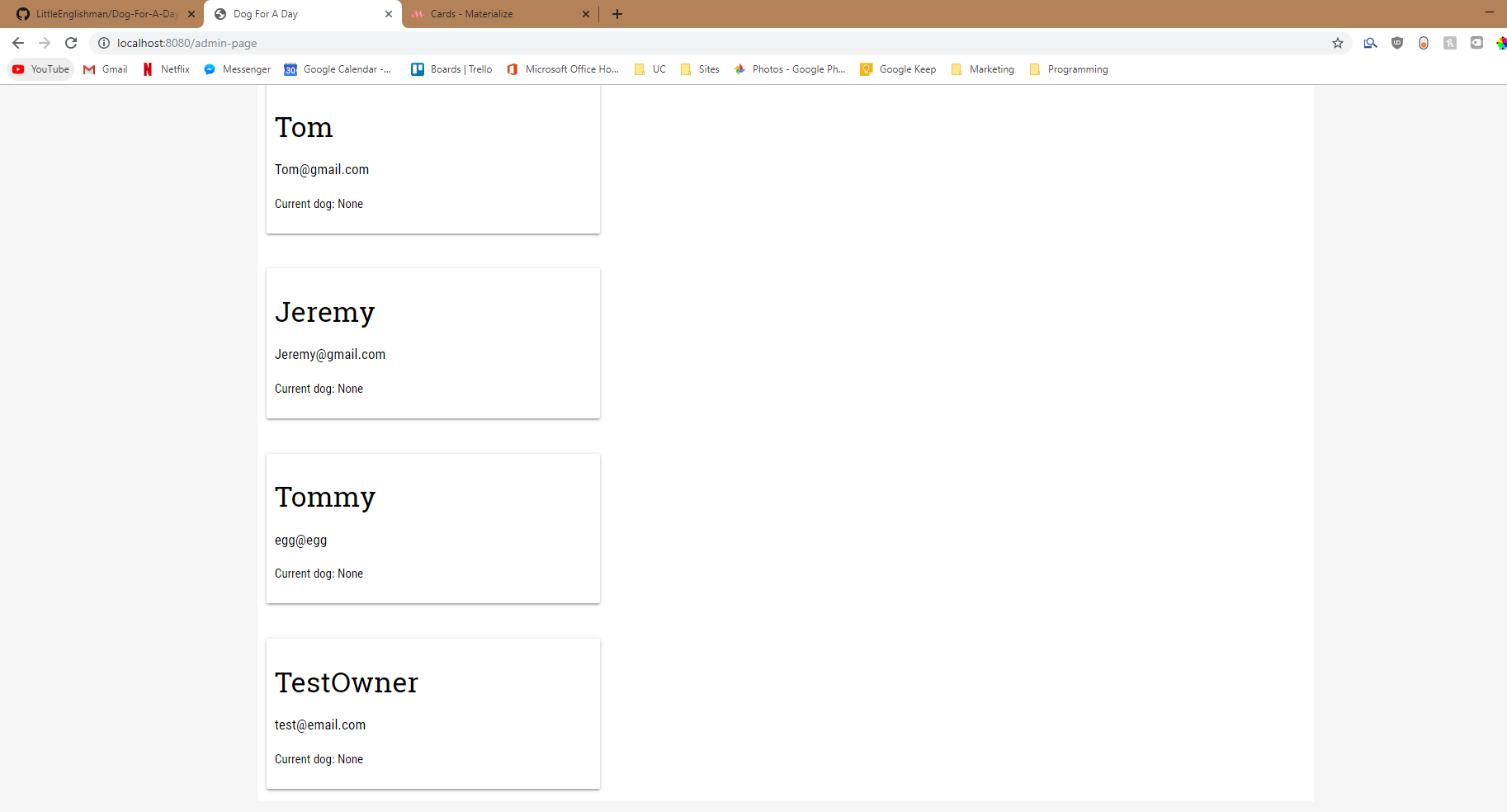
Task 13: Document testing



Here you can see the page loads with the test cases placed in. Dog shows up as none as they currently are just empty values.



Adding in a test owner works. Also shows the dog they have

`

When the dog is returned, the Current dog goes to none.

Final Testing As Well.

To test the webpage as a final version I got my father and my brother to play with the website.

Their feedback. Sam – “Looks good, I wish it actually sent an email though”

My Response – “Setting up an email system is above what I need to do within this assessment and is not possible within bottle”

Dad – “Why are there not two columns? Also needs custom pictures.”

My Response – “Custom images for each new dog are not possible within bottle at our level, I tried but there were no easy ways around it. Having two in each column is doable, however I thought it looked cleaner and easier to read just one at a time”

Neither found any major bugs in my website. However the degree to which they tested it may have been lower than needed to find any bugs.

Task 14: Evaluation

This version was very successful and everything shows up as needed. This version allows an admin to see who has hired out which dog and contact them. It fulfils the need of an Admin page and works well but does not need to look’s amazing. The change to make all forms required also fixes the problem of having empty forms which is a great fix and ups the functionality of the page.

After version 8.0 Here is my Python Code.

# Import all neccessary libraries

from bottle import run, route, get, post, request, view, static\_file

from itertools import count

from datetime import datetime, timedelta

"""

# BUILD LOG

# Version 1.0, created the main python framework with class and test dicitonary.

# Version 1.1, added in code for custom CSS and for Images

# Version 2.0, added in ability to use images/css files. Created the showcase page

# Version 3.0, made the personal dogs pages

# Version 3.1, added in rent functionality

# Version 4.0 , made the add a new dog function/the success page.

# Version 5.0, bug fixing

# Version 6.0, added the ability to track owners names/emails

# Version 7.0, added the functionality to return the dog

# Version 8.0, made the new function for the admin page

"""

# Ver1.0 Class Dog creation

class Dog:

# Give each Dog object a key number

\_ids = count(0)

# Set up initialised variable

def \_\_init\_\_(self, name, age, gender, breed, friendliness, available, image):

#Add new ID to dog

self.id = next(self.\_ids)

# Date to be added later

self.date = None

# Set self variables to given parameters

self.name = name

self.age = age

self.gender = gender

self.breed = breed

self.friendliness = friendliness

self.available = available

self.image = image

# Ver1.0 data dictionary

dog\_list = [

# Test Data

Dog("Bruce", 5, "Male", "Bulldog", 3, 1, "/dog\_image/Bruce.jpg"),

Dog("Moses", 12, "Undecided", "Pussy Cat", 5, 1, "/dog\_image/Moses.jpg"),

Dog("Rex", 3, "Female", "Alpaca", 1, 0, "/dog\_image/Rex.jpg"),

Dog("Max", 4, "Male", "Bulldog", 3, 0, "/dog\_image/Max.jpg"),

Dog("Zula", 6, "Male", "Border Collie", 5, 1, "/dog\_image/Zula.png")

]

# ver 6.0

class Person:

# Give each Person object a key number

\_ids = count(0)

def \_\_init\_\_(self, name, email):

# Add new ID to person

self.id = next(self.\_ids)

self.name = name

self.email = email

# These will be added later

self.dog = None

self.dog\_name = None

self.return\_date = "0/0/0"

# Ver 6.0 data dictionary of persons

person\_list = [

# Test Data

Person("Moses", "Moses@gmail.com"),

Person("Tom", "Tom@gmail.com"),

Person("Jeremy", "Jeremy@gmail.com")

]

# Images Ver1.1

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

def server\_static(filename):

# Return static file from the images folder

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

# Dog Images Ver2.0

@route('/dog\_image/<filename>')

def server\_static(filename):

# Return file from the Dogs folder in images

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

# Code to be able to link custom css Ver1.1

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

def stylesheets(filename):

# Return CSS file from assets folder

return static\_file('{}.css'.format(filename), root='./Assets')

# Ver1.0 Index page setup

@route('/')

@view('index')

def index():

# Pass as no information needed for page

pass

# Ver 2.0 Showcase-page

@route('/showcase-page')

@view('showcase-page')

def showcase\_page():

# Set dog\_list to the data variable and return that to the page

data = dict(dogs = dog\_list)

return data

# Ver 3.0 Personal Dog Pages

@route('/dog-page/<dog\_id>')

@view('dog-page')

def dog\_page(dog\_id):

# Set dog\_id to integer

dog\_id = int(dog\_id)

found\_dog = None

# Loop through dog list to find the target dog

for dog in dog\_list:

if dog.id == dog\_id:

found\_dog = dog

break

#Return dogs data to page in form of a dictionary

data = dict(dog = found\_dog)

return data

# Ver3.1 Rent a dog success

@route('/dog-rent-success/<dog\_id>', method = "POST")

@view('dog-rent-success')

def dog\_rent\_success(dog\_id):

# Ver 6.0 Code added for human form

# Set the variables

name = request.forms.get("person-name")

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

new\_person = Person(name, email)

new\_person.dog = dog\_id

# Find the dog being rented

dog\_id = int(dog\_id)

found\_dog = None

for dog in dog\_list:

if dog.id == dog\_id:

found\_dog = dog

break

# Set dogs availablity to 0 (unavailable)

found\_dog.available = 0

# Add dogs name to the person

new\_person.dog\_name = found\_dog.name

#Set new available date for the dog to be rented out + 1 day from today

date = datetime.now() + timedelta(days=1)

#Set the date on both the dog and person object

dog.date = date.strftime("%d/%m/%Y")

new\_person.return\_date = date.strftime("%d/%m/%Y") #datetime.now() + timedelta(days=1)

# Set human\_data as a dictionary of the humans data we collect

human\_data = dict(human = new\_person)

person\_list.append(new\_person)

# Return the created data to the page

return human\_data

# Ver 4.0 Creating the new Dog page

@route('/new-dog')

@view('new-dog')

def new\_dog():

# Set dog\_list to the data variable and return that to the page

data = dict(dogs = dog\_list)

return data

# Ver 4.0 New dog page action

@route('/new-dog-action', method="POST")

@view('new-dog-action')

def new\_dog\_action():

# Get the variables form the form

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

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

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

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

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

# Create new Dog object (using placeholder image for now)

new\_dog = Dog(name,age,gender,breed,friendliness, 1, "/dog\_image/placeholder.jpg")

# Add new dog to the list oof

dog\_list.append(new\_dog)

# Return data in the form of a dictionary

data = dict(dog = new\_dog)

return data

# Ver 7.0 return page

@route('/return-page')

@view('return-page')

def return\_page():

# Set dog\_list to the data variable and return that to the page

data = dict(dogs = dog\_list)

return data

# Ver 7.0

@route('/return-success/<dog\_id>')

@view('return-success')

def return\_success(dog\_id):

# Find the dog within the list

dog\_id = int(dog\_id)

found\_dog = None

# Find dog within the dog\_list

for dog in dog\_list:

if dog.id == dog\_id:

found\_dog = dog

break

# Set availablity to 1

data = dict(dog = found\_dog)

found\_dog.available = 1

# Find the owner of the current dog

found\_person = None

for person in person\_list:

if person.dog\_name == dog.name:

found\_person = person

break

# Remove dog name from person class only if it doesn't have one to begin with

if found\_person.dog\_name != None:

found\_person.dog\_name = None

return data

# Ver 8.0 Admin Page

@route("/admin-page")

@view("admin-page")

def admin\_page():

# Set person\_list to the data variable and return that to the page

data = dict(humans = person\_list)

return data

# Bottle run

run(host ='localhost', port = 8080, debug = True)