Coursework Report

40402355@napier.ac.uk

Edinburgh Napier University – set09103

Tricktionary – An Online Skateboard Directory

The main purpose of the web-app that I have built, is to look up professional skateboarders and find out some information about them and their skateboarding sponsorships. There is a search feature that will allow people to be able to search by keyword and return results that are relevant to what has been searched. As well as this, all the details will have clickable links that will also show other data that is relevant to what has been selected. There is a contact form in the app that also allows users to enter details of new entries that will be added to a JSON file. This will help to build a bigger, fuller and interactive website.

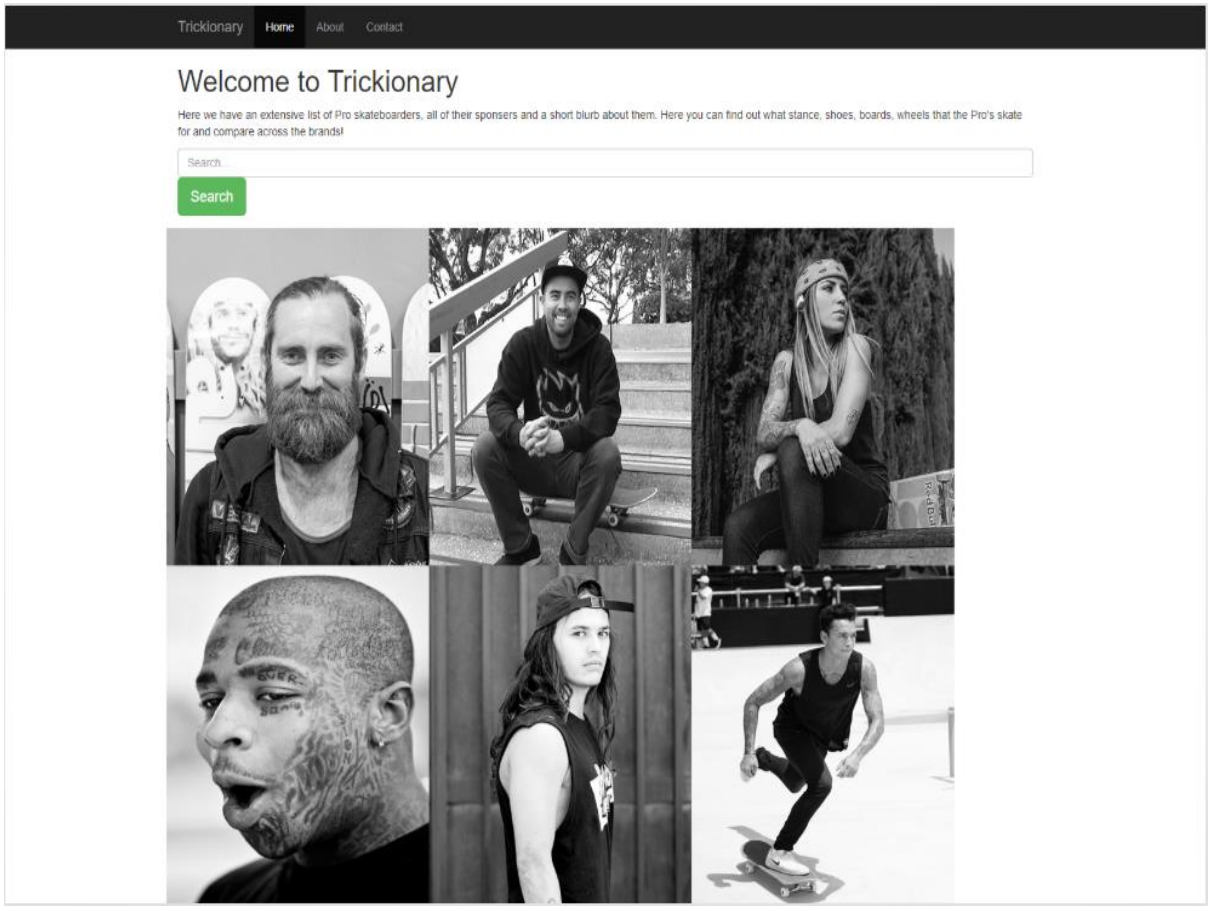

Figure 1: Main Page – This is the landing page of the website with a search bar and some information about the site and six images of professional skateboarders. These six images are all links to their profiles.

Design

My web-app is made up of one python file (Bootflask.py), two JSON files (skaters.json, newSkaters.json), and eight HTML files (about.html, contact.html, home.html, Search.html, results.html, uploads.html, brand.html and 404.html). These names are appropriate to their functionality and fully describe the content of each page. For example, the 'about' page has a short piece of text that describes the website's purpose and the search page shows what users have entered in the search bar. As well as this, the results page shows the specific character profile a user has selected and the contact page allows users to enter their own entry onto the website. These entries are stored in the newSkater.json file. This is so that the entries in the other json file are not overwritten with the new information.

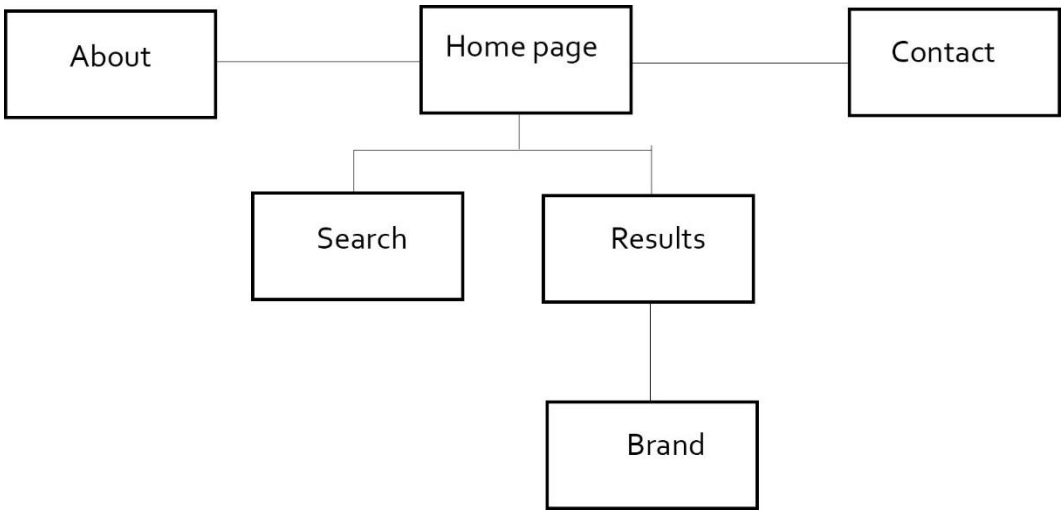

Figure 2: Navigation map for the website. The homepage connects all the pages together and the files from the static folder are fed into the search, results and brand pages.

The Bootflask.py file is the heart of the web-app and pumps data around the website, allowing the other pages to interact with each other. Here I have used a few different methods to create the functionality that powers the site. I have imported Flask, redirect, render_template, request, url_for and json into this python document. Redirect gives the user a message if the URL they have requested is not correct. Render_template allows me to pull up HTML templates when specific requirements are met by the user. Request allows the transfer of data between pages and url_for allows me to create links between these pages.

I have also used a json file to store information about the Skateboarders sponsorships and stances. I have unloaded this json file into a global variable called 'data'. This variable is called upon a few times throughout the different functions that are written within the Bootflask.py file. For example, there are six images of professional skaters on the home page, each of these lead to the same HTML file ('results.html'). However, there is a variable passed in the URL that signals to the function what individuals stats should show up. For example, if Chris Haslam's picture was selected, the URL would read "/results/Chris/". From here, Chris is assigned to a variable called 'stat' that is passed through the function. Using python, I can use this variable's value to access the static folder which will load text files and jpg images that start with the word "Chris". This gives a dynamic web-app that changes depending on which skater is selected from the homepage.

If the user puts in a URL that does not exist, I have implemented a HTML file called 404.html. This page is aesthetically continuous with the design of the website so that users do not feel like they have been thrown off the web-app.

```
@app.route('/results/<string:stat>')
def results(stat):

        txt_url=open('static/' + stat + '.txt')
        content = txt_url.read()
        txt_url.close()
        img_url = url_for('static', filename= stat+'.jpg')
        for skaters in data:
                print(skaters)
                if stat in skaters:
                        skaters=skaters[stat]
                        print(skaters)
                return render_template("results.html", skaters=skaters, img_url=img_url, txt_url=content,  stat=stat)
```

Figure 3: Code from the results function.

From here, there is a ‘for loop’ I found on pybit.es[1] that I have incorporated into my code. This would present the data that is within the python dictionary that I created from the json file. This loops through the dict, looking for the entry that starts with the same data the variable ‘stat’ has currently stored within it. Once this is found, the function returns the correct page template as well as the correct variables assigned to the right data. This is rendered within the HTML doc using jinja2. Jinja2 uses curly braces and percentage signs ({% %}) as a way to slice into HTML code and display the data that was created within the python file. In the results.html page, we can see that I have created a ‘for loop’ and an ‘if’ statement. These both allow the page to become more dynamic as the information within the curly braces will be different for every individual that is selected.

```
<div class="container">
        <div class="row">
                <div class="col-lg-12">
                        {% if stat %}
                        <p><h2>You have selected {{ stat }} these are your results</h2></p>
                </div>
                <div class="col-lg-4">
                        <table class="centered thick-border">
                                {% for s, facts in skaters.items() %}
                                        <tr>
                                                <td><h2>{{ s }}<h2></td>
                                                <td><h2><a href={{ url_for ('brand', stat=facts) }}>{{ facts }}</a></h2></td></tr>
                                {% endfor %}
                                        </tr>
                        </table>
                </div>
        <div class="row">
                <div class="col-lg-6">
                        <p>{{ txt_url}} </p>
                        {% endif %}
                </div>
                <div class="col-lg-6">
                        <img src="{{img_url}}">
                </div>
        </div>
</div>
```

Figure 4: Jinja2 uses {% if stat %} to initiate the if statement.

Whenever {{stat}} is written after this, the variable stat will be inserted into this section of the HTML document. Below, the ‘for loop’ is started, with the variable ‘s’ holding the constants (skateboard sponsor, wheel sponsor, etc.) and the variable ‘facts’ holding the individual’s information. Next, I have used a piece of concatenation to allow the value of the variable ‘stat’ to equal the value of ‘facts’ and then add this value to the end of the URL for the page ‘brand’. The brand HTML file works similarly to the results page, with text documents and jpg images coming from the static folder where the text of the URL after ‘brand’, is the file name.

Enhancements

The search function that I have incorporated works to an extent. This function only works if the first name of the professional skater is entered. This is due to the way the json file is structured; there are six entries that have five sub entries containing unique data. The search function does not look that deep into the json file and will only allow the user to search the first name. It does, however, return all the information stored about that professional skater to be displayed in a table using a 'for loop'.

There is also a contact section of this web-app that allows users to fill in a form that will allow them to upload information about a professional skater who should be on the site. This could be improved as it does store the data, but then nothing happens with it afterwards. It was my intention to allow this data to be seen on the homepage, as it would let the site grow exponentially. Should the site be developed further, I would integrate a system that required the data to be checked for validity before being posted onto the homepage. I would also seek to incorporate an 'image uploader' into this contact form, to give more depth to the character profiles.

Finally, I would have liked to add an embedded YouTube video player that would have one video of the skaters. I feel this would be a good edition to the site as it helps contain users to this site, rather than having to go offsite to find footage of the person that they are reading about.

Critical Evaluation

I feel like the way the data is transferred from the homepage, into the results page, to finally reach the brand page works well. I am very happy that each entry has its own page that is rendered with text files and jpg images instead of creating upwards of thirty HTML pages that would just be repeating the same code over and over again.

The search functionality of the website is rather poor as it only allows six keywords to be searched. If these words are incorrectly entered, it will just give the user a message saying their search was unsuccessful. This is due to me not spending enough time working on this particular section of the web-app as I was focusing my attention more on the results/brand HTML pages.

Personal Evaluation

I feel like this has been a great project to work on. I allowed a good amount of time to research different methods of data exchange and read through the workbook a few times to give myself a good foothold onto where I should keep my reading going. From having never worked with PuTTY, python, flask, vim or jinja2, I feel like I have a good, base knowledge that I can now use and develop in future projects. The biggest challenge I had to deal with during this project was working with Python Dictionaries and JSON files. At first, the for loop would only show the very last entry in the JSON file, which lead me to believe that it was just rewriting the table with new data after each loop. I could see this is what was happening in the terminal as I put "print" functions through my code so I could trace what data was moving through the functions. I realised here it was how the JSON data was being loaded into the "data" variable that was causing this, so I added the suffix "items()" to the variable and it worked.

However,this did not solve all my problems, as whilst it did show me all the information that was in the JSON file, each entry was just one long string of nonsensical data. It was hard work to figure out why all my information was coming through as one long string rather than individual strings. After reading through a few other people's problems with similar situations on different web developing boards, I realised my issue was with the structure of my JSON file and the way the data was being called. My 'for loop' was originally just using one variable instead of two, so it was holding the whole entry within itself. I would keep seeing the error *"Too many values to unpack".* I realised that I would need a second variable in my 'for loop' to hold the second section of data that was being unpacked. I could then turn this second variable into the 'stat' variable that would be at the end of the URL. This solved the issues I was having and has led to the website that currently stands.

I feel like I have performed well in this task, if I had more time to work on this project I think I would be able to fix and add all the issues and enhancements I have mentioned in this report so far.

References

[1] Julian, Flask for Loops - Printing Dict Data, pybit.es, <https://pybit.es/flask-for-loop.html>, 06 April 2017