Coursework Report

Lea Whitelaw
40279124@live.napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09103)

Abstract

This coursework report is following the first coursework assignment in Advanced Web Technologies, explaining the process and decision making in developing this web-app. The coursework brief states that the objective of the coursework is to demonstrate understanding of the Python Flask microframework by creating a prototype web application for an online directory about a given subject. The subject that was chosen for my web-app was Boxing Champions. It was instructed that the nature of the problem domain should be carefully considered, and an appropriate URL hierarchy should be designed, for finding and retrieving information about the collection.

Keywords – web, tech, advanced, technologies, json, python, flask, boxing, directory

1 Introduction

Introduction For the choice of categorical subjects for my web-app, I decided to choose Boxing Champions. This decision seemed to be an easy one to me, I thought that it would not be a commonly made choice, like music or footballers. It seemed as though it would be a intelligent choice to make in terms of showing off the specified coursework skills due to the fact that they have lots of comparable options and categories between boxers.

Decision Making The first step was to do some research and decide which information I would put onto the page for each boxer. After a few minutes of research I came across a web page called BoxRec [1], the self defined "Boxing's Official Record Keeper." This website proved to be very useful for finding information that I needed for the first step in collecting my information. It has information on every professional boxer including: height, percentage of knock-outs, weight category, stance, Nationality, status, alias, reach, and so on. BoxRec[1] proved very useful for my decision making. After finding my way through the website for hours, trying to decide how to do it, I came to the decision that there was too much information for me to be able to make a prototype web-app, in the time frame that I had. This is why I decided to pick one weight category and stick with it, because more than one weight category would require so much data that would go into needs for a database, which was not in the scope of this coursework. At least with one weight class, all though I'd be removing a major variable between the boxers (weight class), at least I would be able to focus on boxers which had fought each other, giving me the opportunity to link those to each other using Jinja2 templates and Flask.
I decided to choose heavyweight boxers due to the fact they are the most well known, so would be of most interest to other people, as well as the fact that it would be easiest to find information and good photographs of them. If I was not factoring in those points, I possibly would've dedicated the site to female boxers to raise awareness. I only chose fourteen boxers, as the web-app is only a prototype, to show the skills that needed to be shown. If it were a different scenario with less time constraints I would've used a database and chosen many more, but as there are over one thousand professional heavyweight boxers alone, that is how I came to the decision of doing only fourteen. It was also made aware to us that we couldn't use up too much RAM on the shared server, so I had to bare that in mind. The considerations of writing a python scraper, something I have covered in a previous module, to scrape the information that I needed, but that wasn't in the scope of this coursework and seemed unnecessary for the skills that I wanted to show off.

2 Software Design

In this section I will explain the software design and my approach and planning of the wed application. The approach I decided to take was to store my data in a JSON file. I went between the ideas of having a JSON file for each boxer, or having one JSON file with the fourteen boxers in it, having the same Key:Value pairs. This seemed like a more logical approach because then I could call all the boxers alias', for example, at the same time. One of the key points that I wanted to have in my design was the option to have multiple different lists of the boxers, separating them for example, by nationality or stance. Initially, it was a part of the plan to have the boxers separated into weight categories, but now that they were all "heavyweights" that part was removed.
With the JSON file holding all the data, and the ability to sort the data on the client side, to however they wanted to view it, the next part of the design plan was to have an individual page for each boxer, with a gif or photo showing their personality, and some further information about the boxers' background in professional fighting. Once directing onto the boxers individual page, there would then be another option to direct onto a page with this boxers' last 5 fights, consisting of the opponent(which would have a link to the opponents page, if applicable), the date, the result and the result of the final 6 rounds. This was all also information that I could find from BoxRec[1].
At this point in the planning process, I considered adding a login feature, but after careful consideration I decided that it would not be a very appropriate feature for the page. For

this type of data that I was displaying, it did not seem necessary to have a LogIn function purely because there would not be any benefit to a user logging in. The only person who would maybe need a login could be the admin to update the information about the boxers, in which case I could have a login that redirected to a form that would update the data. Although, for this sort of interaction it seemed to be more appropriate to have a database, which as mentioned before, was not really in this scope of this particular coursework and could be addressed in the next coursework.
A feature, however, that did seem like an appropriate function to have for my web-app would be a search function, so this was added to the software design plan, a search function that was available on every page which would link to the appropriate boxer when called.

3 Implementation

This section will address the implementation of my code and the issues and their solutions that occurred throughout the process.

3.1 Loading JSON

The first step that I took was writing out the data for the fourteen boxers in a JSON file and writing a basic flask app that would just call the first names of each boxer and print them on the page, just to make sure that everything was working. This part of the process took a lot longer than expected. I had multiple errors when trying to load the JSON file. One of the first problems is that I was researching solutions for 'open Json in Flask' instead of just looking at how to open Json in Python. Another problem I found was that my JSON file was incorrectly formatted, after using a online JSON validator[2], but that helped me fix the issue promptly. I found a commonly used way using "with - open - as" on stack-overflow[3] to open JSON files, but it still would not work with my code. Eventually after a few tweaks in directing it to the correct file-path after hours of research, I finally managed to construct code that worked for me:

Listing 1: Open JSON in Python

```
1
2 def getJSON(fileAndPathName):
3     with open(fileAndPathName, 'r') as fp:
4         return json.load(fp)
5
6 boxers = getJSON(.static/js/boxer.json')
```

After constructing this code, I managed to print a list of names of boxers. It was also helpful further on when I had to load multiple more JSON files.

3.2 Templates with Jinja2

The next step of the process was using Jinja2 templates to construct the HTML pages and layout. This was where the bulk of work was done. My first thought was to construct a template for one of the boxers, and then copy it for the other boxers as well. The way I attempted to do this was to use the boxersâĂŹ last name to then call and print out the rest of the information. After halfway through writing this list, I realised that it would be a lot easier and less time consuming to just have one template for all of the boxers, as long as I could somehow feed the template the information of the last name, or any key:value from the JSON file. I managed to do this through the URL, by making the conditional that if this boxer is clicked on from the home page, the URL will now become : '/(boxer last name)' and the template will pull the information from the URL and if the URL is equal to the last name (boxer.lname) then to print the information in this format. So the code in python would look like this:

Listing 2: get ¡name¿ from URL

```
1
2 @app.route('/<name>')
3 def boxer_details(name=None):
4     return render_template('boxer_layout.html', boxers=boxers, ↩
        name=name)
```

And the code in my Jinja2 template would correspond like this:

Listing 3: Display boxer template in Jinja2

```
1 {% if boxer.lname == name %}
2     <img src="/static/media/{{boxer.lname}}.jpg">
3     <h3>Last name: {{boxer.lname}}
4     <br>First name: {{boxer.fname}}
5     <br><a href="/">Alias:</a> {{boxer.alias}}
6     <a href="/{{boxer.Nationality}}">Position in own country↩
        :</a> {{boxer.Pos_in_c}}
7     <br>Nationality:<a href="/{{boxer.Nationality}}">{{boxer.↩
        Nationality}}</a>
8     <br><a href="/world">Position in world:</a> {{boxer.↩
        Pos_in_w}}
9     <br><a href="/rating">BoxRec Rating:</a> {{boxer.↩
        rating}}
10    <br><a href="/reach">Reach:</a> {{boxer.reach}}
11    <br>Stance: <a href="/{{boxer.stance}}">{{boxer.stance↩
        }}</a>
12    <br><a href="/age">Age:</a> {{boxer.age}}
13    <br><a href="/height">Height:</a> {{boxer.height}}
14    <br>Wins/Losses/Draws: {{boxer.wins}}/{{boxer.loses}}/{{↩
        boxer.draws}}
15    <br><a href="/byko">Percentage Knock−outs:</a> {{↩
        boxer.kos}}</h3>
16 {% endif %}
```

This also meant that I could keep an image in my media folder called "(boxer.lname).jpg" and it would call that different picture depending on which boxers information was being displayed. The links in this code are there so that, for example, if you clicked on "Stance:Orthodox" it would take you to a list of all the boxers whos stance is also 'Orthodox.'

Figure 1: **Home Page**

These bits of code would take the user from the home page(figure 1) to the boxer description page (figure 2 and figure 3).

Figure 2: **Boxer Description Page**- Anthony Joshua

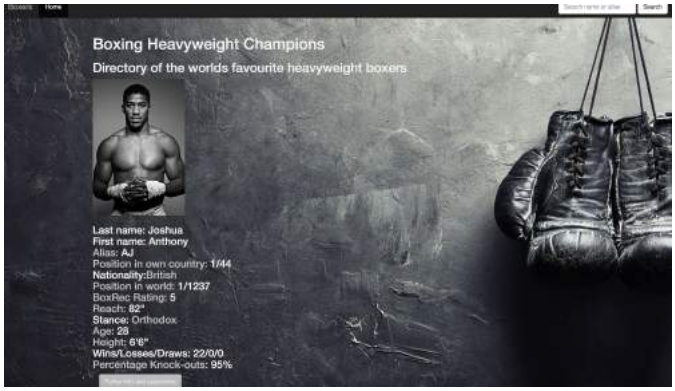

Figure 3: **Boxer description page example 2** - Carlos Takam

Doing my templates this way saved a lot of time and space for displaying the boxers' information, but I soon realised that there wasn't a very easy way to put the further information for these boxers and their GIFs or pictures which showed their character on the same page as their basic stats, so I decided to have a button at the bottom of the boxers layout template which takes the user to another page named "/(boxer.lname)/info". For this second information page I had to create a template for each of the fourteen boxers with some background information and a GIF. I got most of this information from Wikipedia[4], and the rest from my own knowledge. This was a largely time consuming part of the project, trying to find information for each boxer. The layout of the boxer further information page is shown in figure 4.

As shown at the bottom of figure 4, there is a button which says "Recent Fights" which takes the user to a fights page personalised to each boxer, with their five most recent fights(Figure 5). All the opponents have links to their information page if clicked on, where applicable. All of these pages use one template between them all, and call "(boxer.lname).json" to get the information for the individual boxers.

Figure 4: **Boxer further information page example** - Tyson Fury

Figure 5: **Boxer opponents page** - most recent 5 fights

3.3 Working with Python to 'sort' JSON

To get the json file to display the boxers in the order that I wanted, I had to do some work with Python in the Flask app. This required some research online, and I found a thread on Reddit[5] which showed me a neat way of sorting the json by the value in key:value pairs, where reverse=True is only used if the value is required to be sorted from descending going down the list instead of ascending, as shown:

Listing 4: Sort JSON by rating (high to low)

```
1
2 @app.route('/rating')
3 def rating():
4     sorted_rating=sorted(boxers, key=lambda d:d["rating"], ↩
          reverse=True)
5     return render_template('loop.html', boxers=sorted_rating)
```

I used this piece of code for all of the different ways I wanted to sort the data:
Height(High to low)
Age(High to low)
Stance (Orthodox, Southpaw)
Nationality(French, British, American(these were the only reoccurring nationalities out of the selection of boxers, so were just used as an example.) When the nationality was display the boxers were sorted in position in own country(1st-last). (Shown in figure 6)
Position in world (1st-last)
Reach(High to low)
Percentage of knock-outs(high to low)

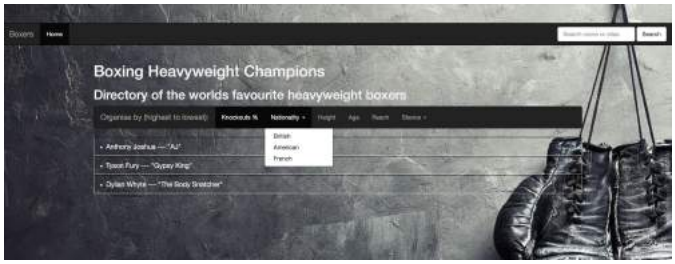

Figure 6: **Nationality Page** - British boxers displayed in order of who is first in country

3.4 Implementing search bar

Implementing the search bar was not something that was covered in the workbook although we were shown how to use forms briefly, so I used my knowledge of that and my knowledge of calling JSON files to get it to work. Up until this point my code had worked fine without the "methods=('GET','POST')" written in the flask route, but after the search bar was implemented, because it uses get and post to get the information and post the relevant results. Since all of the information in my JSON files had the first letter of each work capitlaised for formality, I had to use two python functions: .title() and .capitalize(). I used various methods to join and capitalize the the words that were searched so that the result was that any first name, last name or alias provided could be matched to a boxer. For some reason, when using an else statement for the results not found, the code would skip straight to the else part of the function without iterating through the entire list of boxers first. For this reason I had to use two "for" statements in my search function, as shown below:

Listing 5: Search bar function

```
1
2 def search_bar():
3     for boxer in boxers:
4         s=" "
5         name=(boxer['fname'], boxer['lname'])
6         search = request.form['search']
7         if boxer['lname'] == search.capitalize():
8             return redirect("/"+ boxer['lname'])
9         elif boxer['alias'] == search.title() or boxer['alias'] == ↩
          search:
10            return redirect("/"+ boxer['lname'])
11        elif (s.join(name)) == search.title():
12            return redirect("/"+ boxer['lname'])
13        elif boxer['fname'] == search.capitalize():
14            return redirect("/"+ boxer['lname'])
15
16    for boxer in boxers:
17        search = request.form['search']
18        if boxer['lname'] != search.capitalize() or boxer['alias'] != ↩
          search.title():
19            return render_template('alert.html')
```

An example shown in figures 7, 8, 9 and 10. If something similar to what is typed in figure 7 is searched, the corresponding boxer page (figure 8) will be returned. However, if something that does not correlate to a boxer page is searched (figure 9), and error message with a link to the home page will be returned (figure 10).

Figure 7: **Search** - Deontay Wilder alias

Figure 8: **Search** - Deontay Wilder descriptor page

3.5 Implementing Bootstrap

Every web project or coursework I have done up until now I have always created my own CSS style templates. This was my first time implementing bootstrap or any sort of template for design. I found it slightly confusing to start with but after some time of research how it works on w3Schools[6] I managed to quickly adapt to get the design as shown in all the figures. I found the background image from a Google search for boxing gloves and that image, paired with the bootstrap theme made a nice black and white aesthetic. I had some design problems with the font color for a while as the default bootstrap was grey which clashed quite badly with the background, however, after adding some "style" to the templates, I managed to override the font so that it was white. I know that normally it is messy to keep style-sheets inside the HTML, but because it was only small things and I needed them in the right places otherwise the Jinja2 template would just inherit the style from the base layout template, I made an exception on this occasion. I also thought it would look aesthetically pleasant to make all the 'profile pictures' for the boxers black and white to fit with the theme, but in their further info pages, I left the colours in the GIFs and JPGs the way they were to add some colour and personality to those individual pages.//In the opponents/fights template I made the font a dark shade of red because all black, white and grey were clashing quite badly with the table layout and the background. The reason I chose red was because red is a colour that is typically associated with boxing due to the common colour of boxing gloves, it also mixes quite well with the black and white theme as being the only popping colour.

Figure 9: **Search** - word or words not relevant to boxers

Figure 10: **Search** - error response page

4 Evaluation

4.1 Critical Evaluation

From what I visualised at the start of this project to how far it has come now, I am quite happy with my work. After going through the demonstration of my work and asking classmates for verbal feedback, I see now that there are a few flaws in terms of ease of use, for example there are a couple of places where things could be laid out better, and a few places where links could have been been put in for an easier way to direct through the app. Overall though, I think that the functionality is quite well done. Looking back over it now, I think that for added functionality I could have added something like a subscription list where people could subscribe their emailed and get notified any time that a new fight is coming up. I also think that the website could have used a better layout in general with more inclusion of 'back' buttons. I think that the aesthetic of the website is well done, and I think that there is a lot of potential for it to be coming a well done, fully functioning directory of boxers, if I had used a database. I will continue to work on this project so that maybe one day it will become the directory that I visualised with every different weight category and more variables, but that would take a lot of time.

4.2 Personal Evaluation

I am quite pleased with my progress in learning web technologies from this assignment. I could easily say that I understand perfectly how every part of code in my scripts that I have written work, and I feel that I have learned a lot from this coursework, making me excited and ready to move on to the second one. I think I possibly underestimated the time constraints on this assignment which meant that I was pushed for time in the last week coming up to the deadline, but I managed to come out with a web-app that I am happy to present as my work and further knowledge of how web technologies work. Moving forward from this I think I will start writing my data much earlier on, as it is a very time consuming task which I didn't take into full consideration when I started this project. I will also continue to get to learn bootstrap better and leave more time at the end of my projects for design so that I can keep all my CSS in one place and keep the website looking as aesthetically pleasing as possible.

5 Conclusion

To conclude this report, I will reflect on how my coursework met the specifications of the coursework brief. I believe that I have demonstrated my understanding of the Python Flask micro-framework to the best of my abilities. I think that I have gotten to know Python, Flask and Jinja2 quite well throughout this project. I would say that the URL hierarchy that I designed is appropriate, well done and useful for finding and retrieving information about the collection of boxers. I also think that the pages link together appropriately and I showed my abilities and knowledge that extend the workbook. Overall I am pleased to be handing in this work, and I am looking forward to move further into my learning of these web application tools.

References

[1] "BoxRec boxing's official record keeper." http://boxrec.com/en/. Accessed: 2018-10-02.

[2] "JSONLint the json validator." https://jsonlint.com.

[3] "StackOverflow python- parsing values from a json file?." https://stackoverflow.com/questions/2835559/parsing-values-from-a-json-file.

[4] "Wikipedia the free encyclopedia." https://en.wikipedia.org/wiki/Main_Page. Accessed: 2018-10-15, 2018-10-16.

[5] "Reddit sort json response by value: Python." https://www.reddit.com/r/Python/comments/3vi2n1/sort_json_reponse_by_value/. Accessed: 2018-10-15, 2018-10-19.

[6] "Bootstrap bootstrap get started." https://www.w3schools.com/bootstrap/bootstrap_get_started.asp. Accessed: 2018-10-20, 2018-10-21.