Advanced Web Technologies

# Coursework 1: Technical Report

##### Josh Tait

##### 40176049

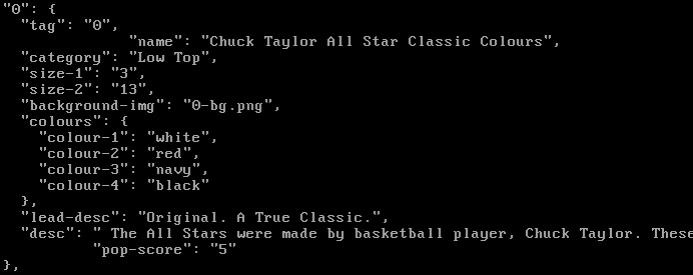
Table of Contents

1. [Coursework 1: Technical Report 0](file:///E:\CW1_Report_40176049.docx#_Toc465281308)
2. [Introduction 2](#_Toc465281309)
3. [Design and Build Architecture 4](#_Toc465281310)
4. [Enhancements & General Improvements 11](#_Toc465281311)
5. [Critical Evaluation 12](#_Toc465281312)
6. [Personal Evaluation 15](#_Toc465281313)
7. [List of References 16](#_Toc465281314)

# II. Introduction

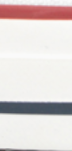
The app that I have designed and developed is an online catalogue for a selection of Converse footwear. Each shoe is categorised (low top, high top, boot etc) and is easily displayed to the user. The user can refine what they are looking for and then view specific information on each shoe.

The app makes use of static image files which are loaded dynamically. All data is stored in a JSON file called shoes.json. Below is an extract to demonstrate how information is stored within the file:



The App is written in Python and uses the Flask framework. The app loads pages dynamically based on the user’s route (see notes on Architecture below).

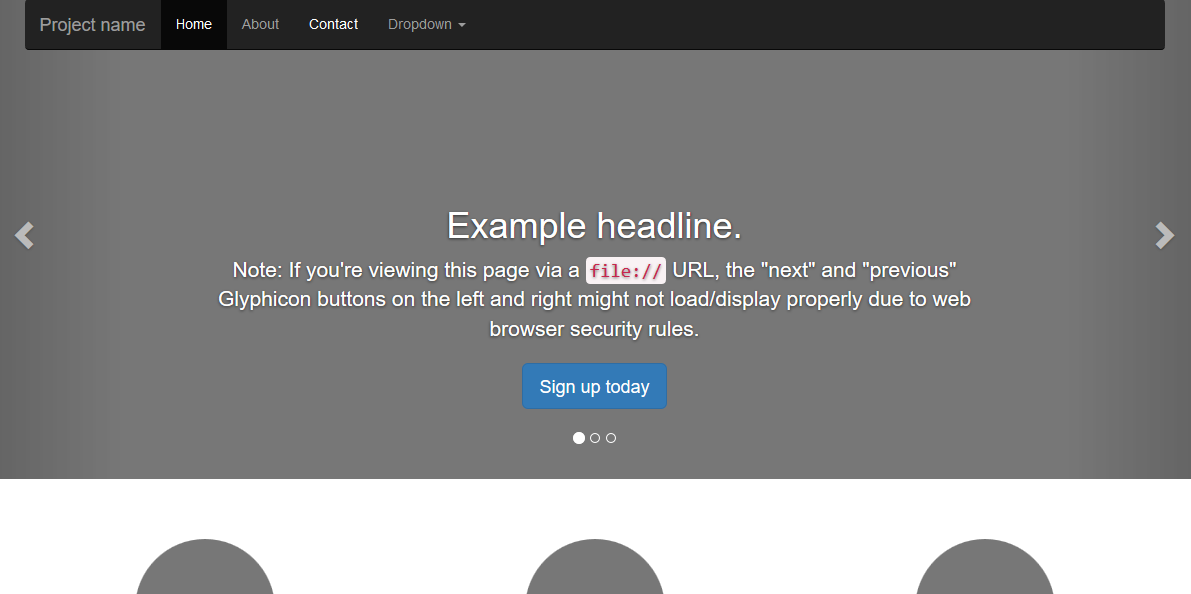
The design makes use of contemporary design theories. For example, the app uses a sans-serif font which is best for body text. The app uses colours which are derived from the original converse shoe:

Example colours from the actual Converse shoe: 

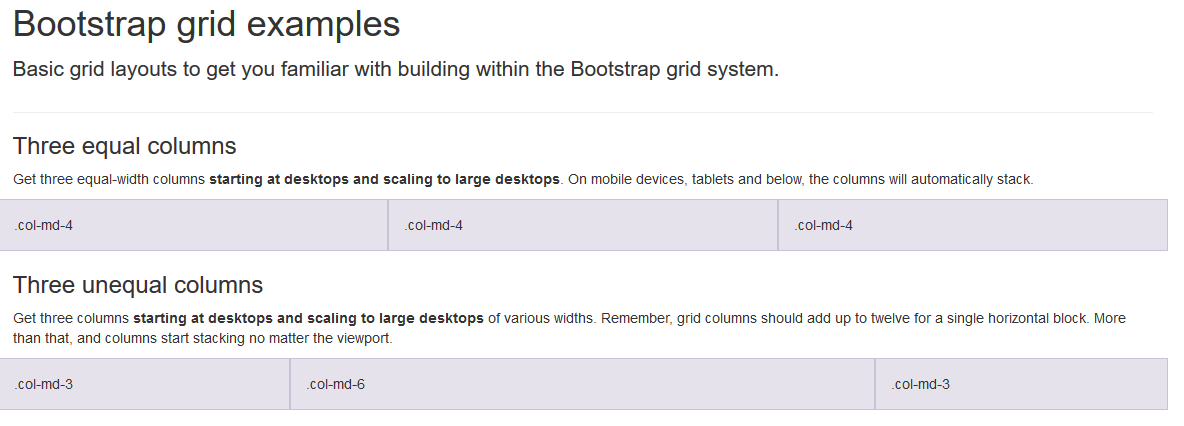
Colours used in the app (in this case, a button): 

The app is designed with modern trends like minimalism and flat-design. This is seen throughout the app with the lack of depth in buttons and overall simplicity of the app.

The app was developed using Bootstrap, an open-source responsive front-end framework, originally created by and for Twitter. Bootstrap allows for rapid prototyping and quick implementation of html elements like the following:



It’s grid system allows for easily placed content like the trendy ‘card’ layout as seen below:



# III. Design and Build Architecture

As previously mentioned, the app was developed in Python, specifically the Flask framework. The app works via python script. Once the script is running, it loads all of the libraries for use via the import function:



The app works by determining the route of the user, for example, the “/” route will display the index page. This is done by rendering a static template which is then displayed to the user.

The user can navigate pages by clicking on buttons and links which will change the user’s route. A different route means a different action. For example, clicking on the ‘Browse’ button on the index page (below), will change the user’s route to “/browse”. In doing this, a new template is loaded, which displays all of the categories of shoes available to the user.

All of the routes can be seen below:

#### Routes and Their Actions

|  |  |
| --- | --- |
| Route (and line number, with reference to above picture) | Action |
| “/” | Root folder. Index page is loaded to user |
| “/browse” | Loads JSON file, extracts data from JSON file and renders template |
| “/browse/<category>” | Makes use of URL Variable (<category>). The category is determined via the data in the URL, and is entered via hyperlink of the previous page.  Template is rendered and category and JSON file are passed to next page. |
| “/shoe/<category>/<shoeID>” | Same logic as above, except shoe ID is taken from the previous page (data is sent via hyperlink) and shoe-detail.html template is returned with the specific shoe data. |
| 404 Error | Not really a route, but included because the function handles a 404 error. User is informed that the page they were looking for cannot be found and allows them to be send to the index page (“/” route) |

#### How Data is Passed between pages

Data has to be passed to different pages in order to generate the correct page, with the correct data from the JSON file. For example, the Shoe ID (what is used to uniquely identify each shoe in the JSON file) is passed to the template *shoe-detail.html*. Python then takes this Shoe ID (which is passed via hyperlink) to the correct page and uses it to show the user correct data. For example, the Shoe ID “0” (the JSON file is zero-indexed) would return the shoe name: “Chuck Taylor All Star Classic Colours”.

This technique is used throughout the app and is used to display lots of information, stored within the JSON file. This design meant that new shoes can be added, as long as they suit the format previously shown and shoes can also be taken away. This means that the app can run and be changed with minimal effort. This would be handy for someone who does not have extensive technical knowledge of Python or JSON. A user (like an admin) could fill out a simple HTML form and easily update the JSON file.

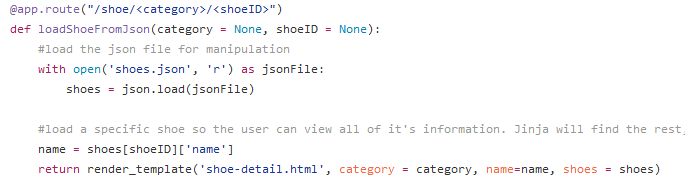
#### 

Example of how data is displayed to the user once the page has rendered.

#### How data is accessed and interpreted

Because all of the data is stored within a JSON file it has to be accessed before it can be used and displayed to the user.

Python handles some of this via the following statements:

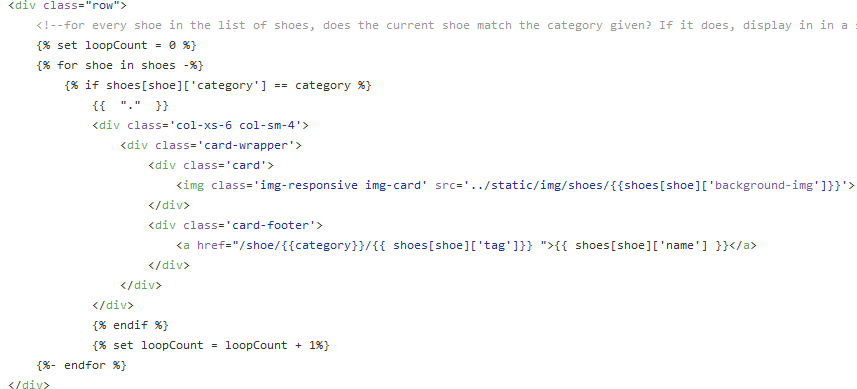


The **with open ()** function loads the JSON file. Python then creates a variable to the value of the JSON file, allowing the data inside accessible, as seen in the following lines.

**Name = shoes[shoeID][‘name’]** allows Python to take the name of the Shoe (referenced via the ShoeID variable), which can then be passed to the template.

Once the page has been rendered, the data can be manipulated and displayed. This is where the language **Jinja2** comes in. It accesses the variables sent from Python which can then manipulate and display to the user. Jinja is a fully-fledged language designed to work with Python. Once the page has rendered, the Jinja instructions are invisible to the user (similar to how inline PHP works in a HTML document), leaving only the pre-formatted data and html behind.

The following is an excerpt from the template **category.html**, which renders all of the shoes in a given category:



Jinja can use a “for” loop to iterate over every shoe (which is available through the variable **shoes**, equal to the value of the JSON file. It is important to note that the **shoes** variable does not directly equal the value of the JSON file, it just stores the data in an accessible format). The category was also sent to this page and is used to filter out the categories available. Essentially, Jinja has been given access to the entire JSON file (**shoes**) and it filters through all of the data and gives the user the correct data. This can be seen in this line of code:

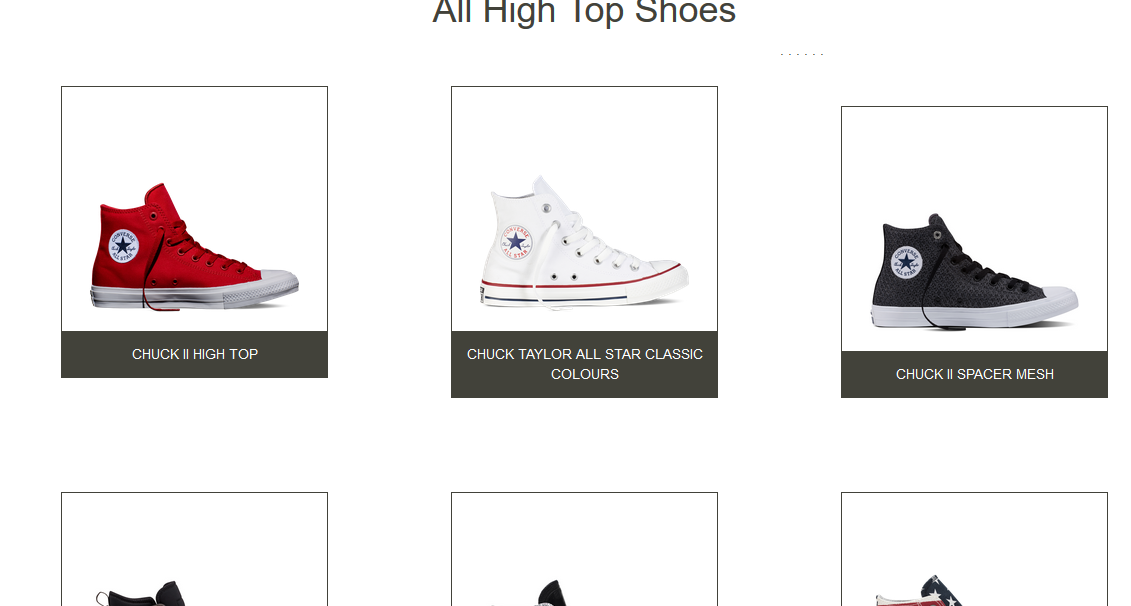


This means:

* For every shoe in the list of shoes
* Check the current shoe’s category matches the category given (the one passed from Python)
* If it does, print out some of it’s data like the image of the shoe
* If it doesn’t, go to the next shoe.

Some of the data above is used to format some of the <div>s shown (this is to prevent a strange bug discovered, see bug notes in Critical Evaluation), and is not used to display any actual data.

The result is the following:

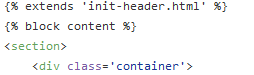


#### Template and Template Inheritance

It has been previously mentioned that templates have been used to display pages to the user. The way that this works is Python has a **render\_template** function, which tells the app to look for the specific template in the **template** folder.

Jinja then uses these templates (and the data passed to them) to ‘build’ a webpage for the user.

Jinja can also inherit templates, and in fact inherits a template which essentially initializes all templates. This works by creating a base template (in this case, **init-header.html**), and the other templates effectively are ‘built’ inside this page.

Below is an example template which ‘extends’ the base template: 

The first two lines are Jinja which basically means ‘take init-header.html’ and place this template within the ‘block content’ tags in the base template.

‘block content’ can be seen in this file, as well as in the base template, as shown below.

**init-header (base template):**



Lines 23 and 24 are Jinja Tags again. These tags basically mean ‘put the other template here’. Notice the closing </body> and </html> tags. This basically means that init-header.html creates a fully rendered html document.

Being able to use templates and inherit other ones makes Jinja very powerful and made it easy to make changes to say, the <title> for example, as only one page had to be modified, the rest would load the modified content (the title of the page). This had a disadvantage however. It meant that the title could not be changed dynamically, i.e. when the user changed between pages.

# IV. Enhancements & General Improvements

There were several factors which made development difficult or otherwise limited in terms of what the app can do.

If time had allowed the following features could be implemented:

* Breadcrumbs. This would allow a visual path for the user to follow.
* A search function. This will improve usability as users will be able to find exactly what they are looking for. And if it cannot be found, then the user will know and can be advised on what to do next.
* A sorting algorithm. Allowing the users to sort through all of the shoes would be handy as it would allow them to quickly find a particular shoe, or a category like most popular shoes.
* A popularity system. This would allow users to see which shoes are more popular, and for whatever reason.
* A contact form. To request new shoes, submit feedback or bugs found in the app, which will improve and aid general maintenance.
* An Admin Panel/Backend system to allow new shoes to be added, as well as the ability to remove some, or edit the JSON file directly, allowing for new data to be displayed to the user.
* A ‘Similar shoes’ section, where users could view other shoes, similar to the ones they are viewing.
* A gallery would have been a good feature, as only one side of each shoe is visible, and even then, only a small portion is visible.

If there had been more time, it would be ideal to improve the JSON file, allowing for larger descriptions of the shoes.

Although not a feature, testing would have been good as it would have highlighted errors and potential flaws in the code, which could be prevented from happening again. Also, there is very little SEO or accessibility features in the templates, meaning it may be hard to find once the website is indexed, as well as potentially difficult for some users with a disability.

# V. Critical Evaluation

The app has some (albeit limited) functionality. It works well and data is passed to pages. The app has been designed to run with minimal effort to maintain it. The app achieves this, but only just. There are no extra features like sort or search. There is no back-end, which would make it hard to non-expert users to update the content on the site.

The website overall feels modern and looks appealing, even though it is very minimal. In some places, the website appears to have the barebones, and could do with some extra content for the user to look at. It is however very easy to use, which should make it accessible for anyone with a large range of technical literacy (unskilled to skilled).

The code is well-written. It makes use of functions, variables, control stuctures and error handling. Comments are available throughout, but more could be recommended, especially for a different user to maintain the app.

The app uses rich photography, although no images have been compressed or optimized for web, which means that load times may suffer.

The app is responsive, but only just. It could use additional CSS media queries to make it more appealing on responsive devices. In today’s market, the majority of devices that people use are smartphones, meaning that it is crucial that websites are displayed well across a range of platforms.

There is no ‘back’ button, which could make it difficult for some users to navigate back words, should they choose to. URL Hierarchy is well thought and is consistent to an extent (browse/<category>). Users should be able to make sense of the URL, and the app should be able to do that. However, the URL paths could be improved. For example, there is a space between characters in the URL. This is considered bad practice as browsers can get confused and return errors. All spaces should be escaped with ‘%20’.

Also, the categorization of shoes could do with improvement. For example, when viewing a shoe in the ‘High Top’ Category, this is the URL path:



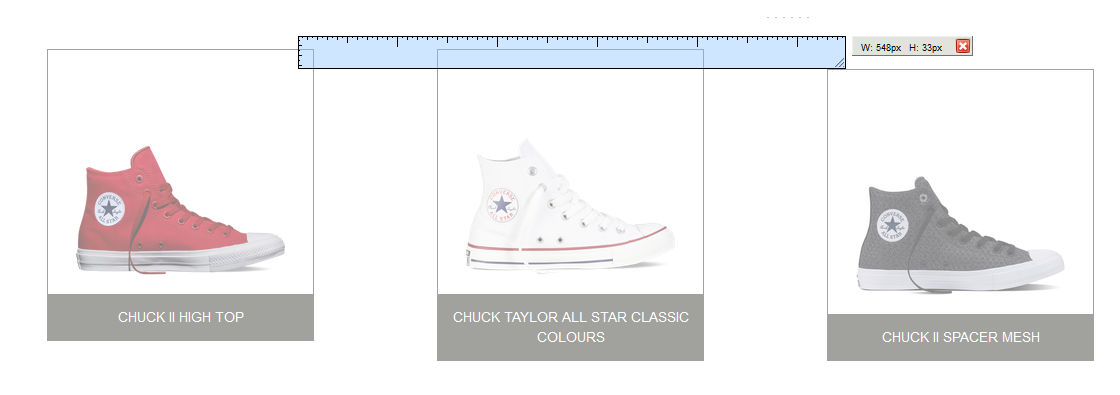
‘0’ does not mean anything to the user, so a name would be better. The ‘/shoe/’ section should be changed to ‘browse’ for consistency. Users would expect to narrow down, in other words go to ‘browse’ then ‘High Top’ then ‘Shoe-Name’.

Going further, any shoe can be accessed by entering a number after the ‘/’ after the category, meaning a user could technically access a shoe which does not match the category in the URL. While this is only a minor issue it is un-professional and should be avoided.

The app seems to be reliable, but offers little in terms of error-handling of the app itself. For example, if a file could not be loaded, there is no safeguard, meaning the user will be thrown an obscure error, which may make it difficult to diagnose and fix.

#### Bug Notes

There are several bugs which are present in the app. The main one is the un-explainable positioning of <div>s which are created dynamically via Jinja loop:

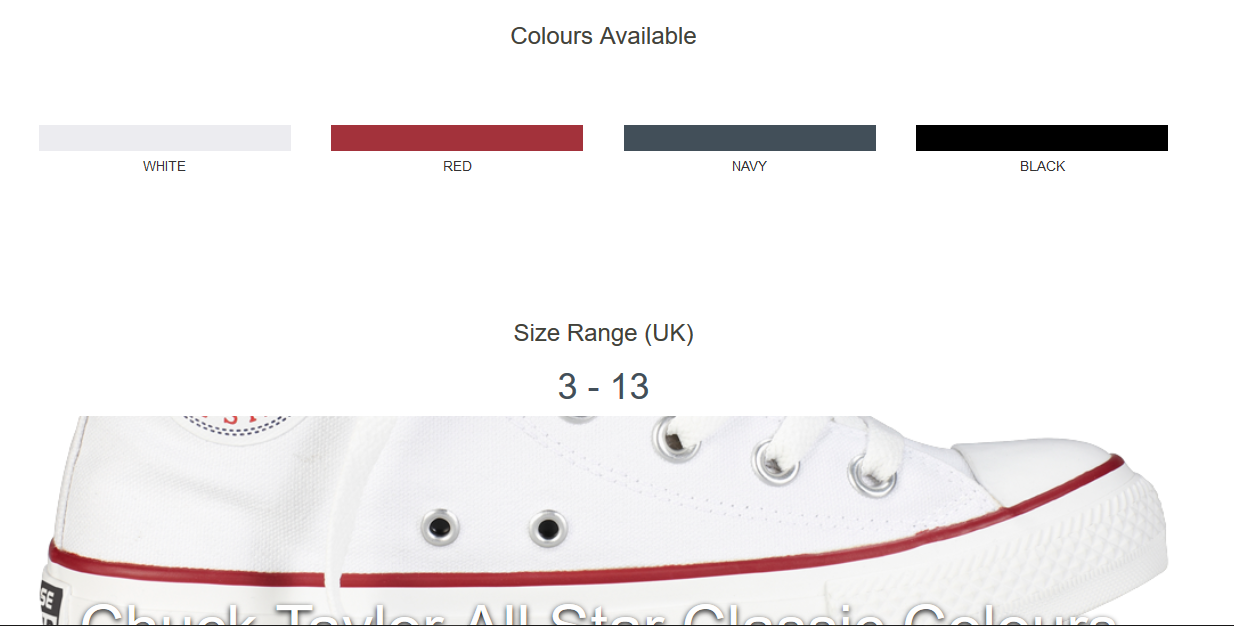


Screen ruler shows that elements do not line up.

This bug could not be fixed, as it was un-clear what was causing it in the first place.

Adding a ‘.’ Between each iteration of the loop seemed to help, as when it was removed, the positioning of the <div>s seemed more random.

Another bug which was successfully fixed was the following:



As you can see, the page starts to repeat itself. This is because of a naming conflct within the JSON file. Two shoes share the same name. When Jinja loops over the data, it pulls both shoes, because both shoes match the criteria (in this case, the name of the shoe). This has now been fixed.

# VII. Personal Evaluation

Overall, I feel that the project went well on the whole. I feel that it could have went a lot better however. I faced many challenges (most of which were cumbersome and difficult to tackle).

I learned a lot during this project. I learned how to use Python and how to set up URL Hierarchies. I learned JSON, and Jinja. I learned how to combine all of the languages and what they’re all used for. I learned how to set up static files and templates.

I also faced a lot of challenges. I feel that the majority of these challenges could have been made a lot smaller had there been clearer documentation. A good example would be with Jinja. I had tried to implement a sort feature, where sort criteria was passed to the page via a form. The page would load and sort the data from the JSON file. However, the page would not correctly load and Python would throw Jinja errors in the Trace. These errors were often obscure (due to lack of experience), which made it difficult to find out what went wrong.

I learned how to determine what was at fault through process of elimination and hardcoding values. I found that instead of building entire algorithms that rely on different languages and constructs, that I should build features piece by piece and test that each piece works first.

When that didn’t work, I would rely on documentation and forums (like Stack Overflow) where users had issues that were either similar in fashion or loosely-related. I learned how to adapt my solution to my problem from someone else’s solution.

# VIII. List of References

The list below contains all of the information I have used and referenced to build my catalogue:

Ronacher, A. (2008). Welcome to Jinja2 — Jinja2 Documentation (2.8-dev). [Online] Jinja.pocoo.org. Available at: http://jinja.pocoo.org/docs/dev/

Tait, J. (2016). Sorting through a json file, using Jinja2 and a JSON Variable. [online] Stackoverflow.com. Available at: <http://stackoverflow.com/questions/40208396/sorting-through-a-json-file-using-jinja2-and-a-json-variable>

Jude, J. (2016). Common Flask Errors And Their Solutions by @jjude. [online] Jjude.com. Available at: http://www.jjude.com/flask-errors/ [Accessed 20 Oct. 2016].

Gundlach, M. (2016). Can comments be used in JSON?. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/244777/can-comments-be-used-in-json?rq=1 [Accessed 21 Oct. 2016].

Crockford, D. and Carter, Z. (2016). JSONLint - The JSON Validator.. [online] Jsonlint.com. Available at: http://jsonlint.com/ [Accessed Oct. 2016].

ECMA International (2013). JSON. [online] Json.org. Available at: http://www.json.org/ [Accessed 14 Oct. 2016].

Converse.com. (2016). Converse - Official UK Store | Converse.com. [online] Available at: http://www.converse.com/uk [Accessed Oct. 2016].

C1.staticflickr.com. (2016). Background header image used in index page. [online] Available at: https://c1.staticflickr.com/5/4095/4790509812\_a11c2c1b09\_b.jpg [Accessed 26 Oct. 2016].

Beckett, D. (2016). How to select all elements whose ID starts and ends with specific strings?. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/16009755/how-to-select-all-elements-whose-id-starts-and-ends-with-specific-strings [Accessed 12 Oct. 2016].

Python. (2016). Python 2x Documentation. [online] Available at: https://docs.python.org/2/library/index.html [Accessed Oct. 2016].

(surname not given), J. and Otto, M. (2016). Bootstrap · The world's most popular mobile-first and responsive front-end framework.. [online] Getbootstrap.com. Available at: http://getbootstrap.com [Accessed Oct. 2016].

Note: where a date is omitted in the reference list above, the respective references were accessed on many occasions in October 2016.