Coursework 1 Report
Coffee Makers Web Application

Igors Ahmetovs
40125689@napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09103)

Abstract

This report describes the "Coffee Makers" web application. It summarises the main functionality of the application and gives justification to the design decisions behind it. It also looks at potential improvements that could be made in the future.

Keywords – Python, Flask, Web, App, Coffee, Espresso, Jinja

1 Introduction

Overview The web application contains information about various types of coffee makers that might be of interest to the user. The coffee makers are logically grouped into three categories: automatic, semi-automatic and manual machines. This classification is represented within the web-app by hyperlinks and the navigation bar, which provide the user with a more cleaner presentation and experience than without categorising the items and dividing them into appropriate sections.

By visiting the main page, the user is presented with the choice of the three categories of coffee makers. By selecting a category of interest, the user follows to the page containing the coffee makers of that type.

The last page prints the details of the particular coffee maker, with a link back to the coffee maker selection. This removes the need to hit the "Back" button of the browser to view the details of a different coffee machine.

All the menu options are represented with images and captions to ease the overall experience of navigating the website. However, navigation can also be fulfilled via the navigation bar that is present on each page. This way, the person using the web-app could find the information they are looking for more quickly, without having to follow multiple links until reaching the destination.

The navigation can also be performed by specifying variables in the URL.

2 Design

The most important points during the design phase were ensuring that there is no unnecessary duplication of code present. Pages had to be structured in a way that they could be used with different kinds of data and for different purposes.

An example of this would be the "/category" and "/category/maker" routes, which trigger the (almost) only function in the web-app. However, based on the passed URL variables, the function will have different behaviour and render the correct template.

It was also crucial to ensure that no value would have been hard-coded. The data that the application uses is stored in a single JSON file as an array of dictionaries (that in itself contain another array of dictionaries). The JSON is loaded into the application at web-app start-up and allows the user app data to be verified against it. The data from the JSON is also passed to the HTML templates that render all the pages based on specified parameters.

It is also essential that the application architecture makes sense to the end user as well. For this reason, the data has the same structure within the JSON file, the URL hierarchy and the actual content on the page. The collection of different types of makers are displayed on the main page. Machines of a specific type are listed under the url of that type. And, the details of a particular coffee maker of a single type can be viewed in a URL that is a child of the address of the coffee maker type, thus preserving the overall structure. If the user wishes, they could easily manipulate the URL to get to the page they require with ease.

The application logic is neatly separated from the information displayed in the browser.

3 Enhancements

Despite achieving the required level of functionality, the web application could always be improved in the future and offer more functionality to the end user.

The website could benefit from having some form of user input. New coffee makers could be added to the site's data via a POST request. The data of the request could be retrieved in the application code and serialised into an updated JSON file that is used for populating the pages.

This brings forward another potential improvement - multiple user accounts and methods their authentication. The data stored on the server could also be used to save and retrieve the state of the web-app for each individual user.

The user experience and front-end requires a lot more work, as it is not particularly interesting or visually appealing. However, it still meets the criteria of being usable and functional. The overall user satisfaction would definitely be improved by utilising more front-end technologies such as CSS, JavaScript and their respective libraries (e.g. Bootstrap, jQuery).

The said technologies could for example introduce a search bar in the navigation menu of the page. With jQuery UI, one could implement autocomplete as well, which would make the location and retrieval of required page even quicker. This would make even more sense if the data would become more and more populated with the help of the user population.

4 Critical Evaluation

Even though the web application does not necessarily present a superb user experience, it has strong underlying logic on the application level.

The duplication of data is minimised by using the functionality offered by Flask.

The data is serialised and deserialised to and from a single JSON file, which removes the need to have any information presented in the browser to be stored within the code.

With the help of URL variables, the display of coffee makers and coffee maker types is taken care of by a single function that accepts two path and two arguments, with the second being optional. This, again, minimises the amount of code that has to be rewritten.

The URL variables themselves are always validated against the internal application data. However, if a particular URL for a coffee maker page was not found in the data list, the user is redirected back to the page containing all the coffee makers of that type. If the error was in the part of the URL that corresponds to the coffee maker type, however, the user is issued a custom 404 error with a link to the main page.

As mentioned previously, the information that is presented to the user has been separated from the application logic into the relevant HTML templates. This allows for taking advantage of the functionality offered by Flask, such as template inheritance. This streamlines the development even further, as it means that the same HTML does not have to be rewritten and repeated in every page.

5 Personal Evaluation

Familiarise with Python and proficient and tie previous web development experience. Note the difference in approaches, i.e. developing using Python vs PHP.