Development report for Python Flask

iChoons music catalog application

Tom Brown

Edinburgh Napier University SOC

Module: Advanced Web Tech

Tutor: Simon Wells

Date: October 2016

Summary

This report will illustrate the design and development architecture and processes during creation of a Python Flask web application. Flask is a micro web framework written in Python, an object-oriented, programming language.

Furthermore, the document will evaluate the application as well as gauge any personal achievements and hurdles overcome during the operation. Findings will be noted and any potential enhancements to the application will be highlighted.

Table of Contents

1. Introduction

The project aim was to demonstrate the skills learned so far with Python Flask. To realize this goal, the context would be by way of a digital catalogue application. This form of implementation would evidence any accomplishments of using the Flask framework. The subject matter was down to the individual's choosing. In this case, the context would be a music catalogue. The application is called iChoons and allows the user to search for a specific genre of music from some common categories. Also built in to the application is a user authenticated admin area where the user can upload album artwork to the static folder in readiness for new additions to the catalogue database.

2. Web architecture

2.1 Python

Python is an open source, highly readable, cross-platform, object-oriented programming language. The syntax is approximate to English, making it easier to learn than some other languages [1]. Python has a broad support base and is very well documented online. The Flask micro web framework is written in Python.

2.2 Flask

The Flask framework is a scaffolding, affording creation of web applications in minimal time. The architecture it utilizes is the MVC paradigm, which stands for Model View Controller. Separation of concerns means web application is broken down into three separate tiers. One layer represents the data store or database; this is the Model (M). One for the presentation layer or user interface, which is called the View (V), and the logic is isolated from the other tiers in the Controller (C) layer [2]. This is evidenced in the image below.

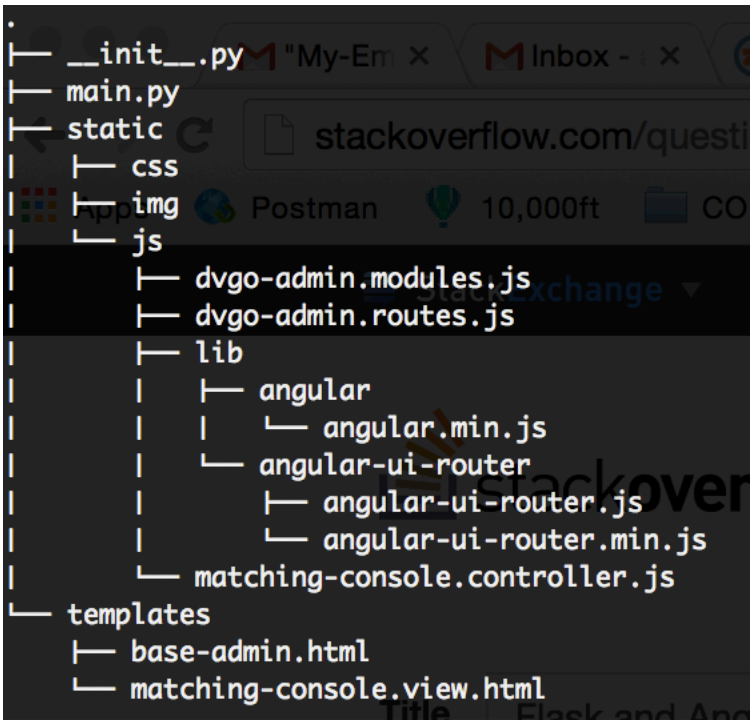

Figure a.1: Folder structure of a typical Flask application showing separation of concerns

The model layer is the data store where data pertinent to the running of the application is held. Models can be in the form of a database like MySQL or MongoDB. However, in the context of the iChoons app, the data is stored in a JSON (JavaScript Object Notation) file. The controller can grab data from the Model and use it to respond with a specific view.

The view is the user interface that illustrates data gathered from the model. In Flask, views are quickly generated via the Jinja templating language. An advantage of templating is that multiple responses can be administered by a single controller route. [3]

The controller is the nerve center of the application. The connection between the user and the system. All request handling occurs here, and appropriate responses or views are given in reply to requests. The controller executes the logic that can update the model. [4]

Consequences of using an architecture like the one described above are many, including a faster development process. Separation of concerns allows for multiple developers to work on a single project simultaneously. Compartmentalising the 3 facets of an application allows for any one tier to be changed without drastically affecting another, thus reducing time and resources spent in updating systems [5]

3. Enhancements

Functionality-wise, the application does everything that was in the brief. However, there could be some improvements made. With this in mind, a few points are outlined where enhancement is possible.

3.1 Search Functionality

Basic search functionality was built into the application by way of an HTML form entry. The search terms were limited to three different genres. Time permitting, more extensive functionality could be implemented, permitting a more refined search, facilitating an improved user experience.

Search functionality was not explicitly defined in the brief as a functional requirement. However, this feature, even at a simple level affords easier use over typing a path into the URL.

3.2 User Interface (UI) Design

Focus on design was not a prerequisite for the project. Nonetheless, some time would be devoted to the design aspect of the application. That said, additional time could be spent on styling and creating more custom graphics. Implementation of some more sophisticated CSS animations on buttons [6] and perhaps an off-screen or sidebar navigation [7] would also make for a better overall look and feel. A more refined view of the search results could also improve aesthetics.

3.3 User authentication

Users can log into an admin area where images can be uploaded. Elementary authentication is provided for development which, going forward, is not ideal for production. Authentication could definitely be upgraded, should the app go into production [8].

3.4 Pagination

For the purpose of demoing the application, a limited number of albums were stored in the accompanying JSON documents. Despite this, the application skill exhibits the necessary functionality requested in the project brief. More albums/bands to display would yield the opportunity for a pagination component to be added to the application [9].

4. Critical Evaluation

In the project brief, it was specified that the web-app should ‘demonstrate your mastery of aspects of Flask covered so far. You should make appropriate use of routing (and URL hierarchy design), static files request, redirects, responses, and templates.'

This section will break down the requirements and evaluate them to establish if the application is fit for purpose. For this section, when reference is made to the ‘code base’ it pertains to https://github.com/Tommy-B-Git/src where all code can be examined.

General Functionality

An important question to consider when evaluating a piece of software is ‘Does the program do what is expected of it?’

Routes are another name for URLs. Overall, there is a proper use of routing as well as the other requirements throughout application, as is evidenced in the code base. The home page or root of the program is routed to (‘/’). All other routes in the application are built on top of the root route. The routes for the genre sections could have worked better. For example, when browsing the rock section, the URL would be more meaningful if it read as /browse/rock. However, this is not the case, and further work will need to be required in app.py to remedy this. See figure a.2. below.

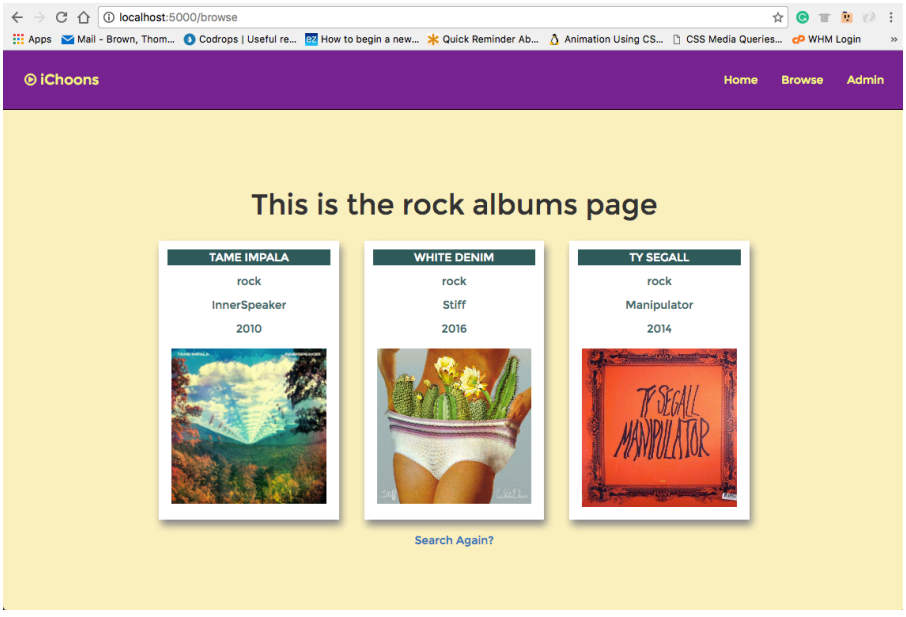

Figure a.2: Note the URL. It should read ‘browse/rock’

```
@app.route('/browse')
def browse():
  return render_template('browse.html'), 200

@app.route('/browse', methods=['POST', 'GET'])
def genre():
  #Always set an error object to None
  error = None
  #grab the user input from form field and make lower case
  my_genre = request.form['my_genre'].lower()
  term1 = 'rock'
  term2 = 'indie'
  term3 = 'popular'
  # I know there must be a better way of error handling below.
  if my_genre == "":
    error = 'Empty search box - Click here to try again'
    return render_template('/browse.html', error=error)
  #validate against gibberish search term
  elif my_genre not in (term1, term2, term3):
    error = 'You must enter a search term - Click here to try again'
    return render_template('/browse.html', error=error)
  else:
    json_data = open('static/json/' + my_genre + '.json').read()
    result = json.loads(json_data)
    return render_template('results.html', my_genre=my_genre, result=result)
```

Figure a.3: Code for the browse route does not point to the browse/'genre' URL

The application comprises of a digital catalogue of items that were to be displayed when a request was made to a data storage mechanism, in this case, a JSON file. Displayed results are illustrated firstly by figure a.2, and then by the code in figure a.3, above. The code for the application does work and has been subject to ongoing rigorous testing through the development cycle. The functionality of this part of the application could have been improved by adding more content to the JSON file. Nonetheless, the app does display the catalogue in the manner it is supposed to. More content could be included in the JSON file, time permitting.

Admin Login

To further meet the requirements of the project brief, the inclusion of an admin login section is provided. The log in is the portal to the image upload section of the application. Written into the code is some basic validation is written in the app.py code. The password and username fields will accept a very simple duplicated of name and password which is not great for security. Despite that, it does help to illustrate the implementation of routing, responses and templates very well. With more time this could be bettered by adding code to increase security. See figure a.4 and a.5 below.

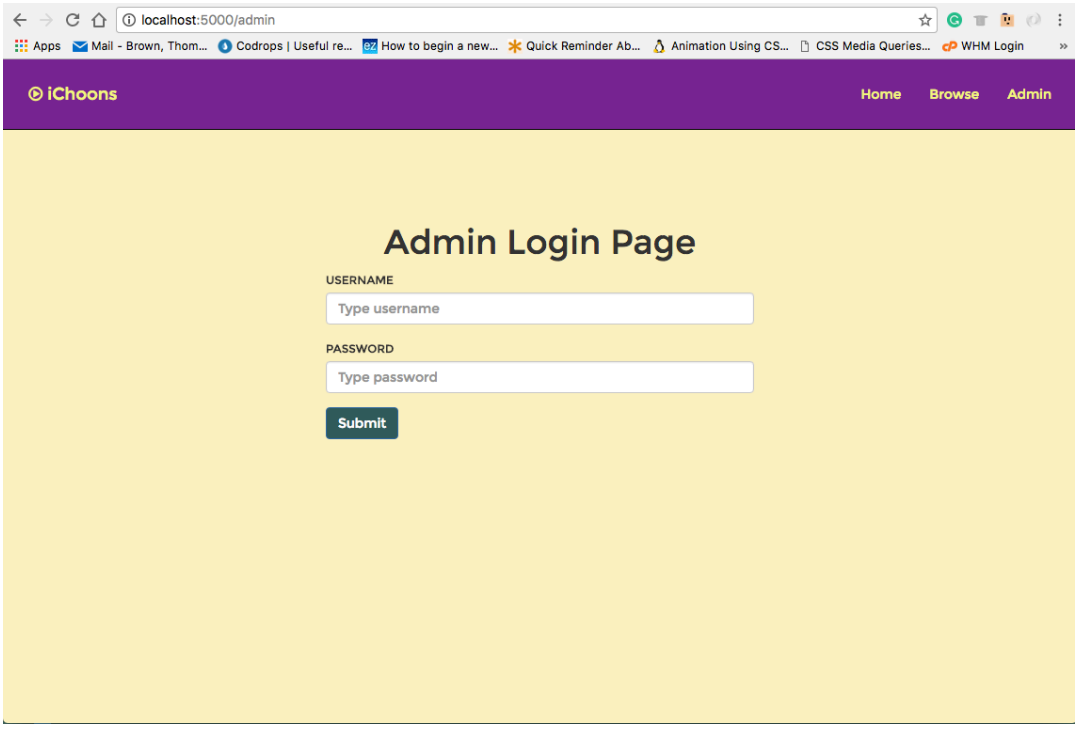

Figure a.4: Admin login page /admin

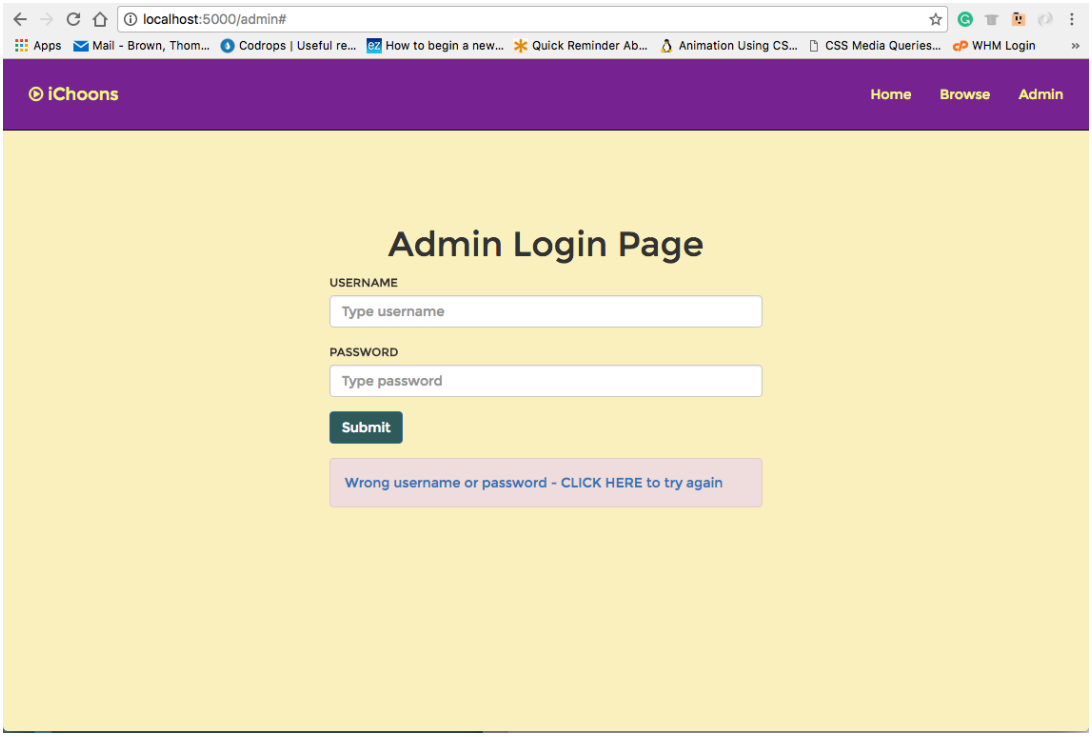

Figure a.5: Admin login - validation

Image Upload functionality

To add images to the application, an upload facility was included. This works well and delivers appropriate responses and feedback to the user if the upload is successful or unsuccessful. Had more time been available, it would have been improved by adding further fields to the form; to enable the JSON document also to be updated as and when a new entry was added or delete from the catalogue. Indicated below, in figure a.6, the routes for admin and upload are evidenced. I feel this works well enough for the purpose of the project but could

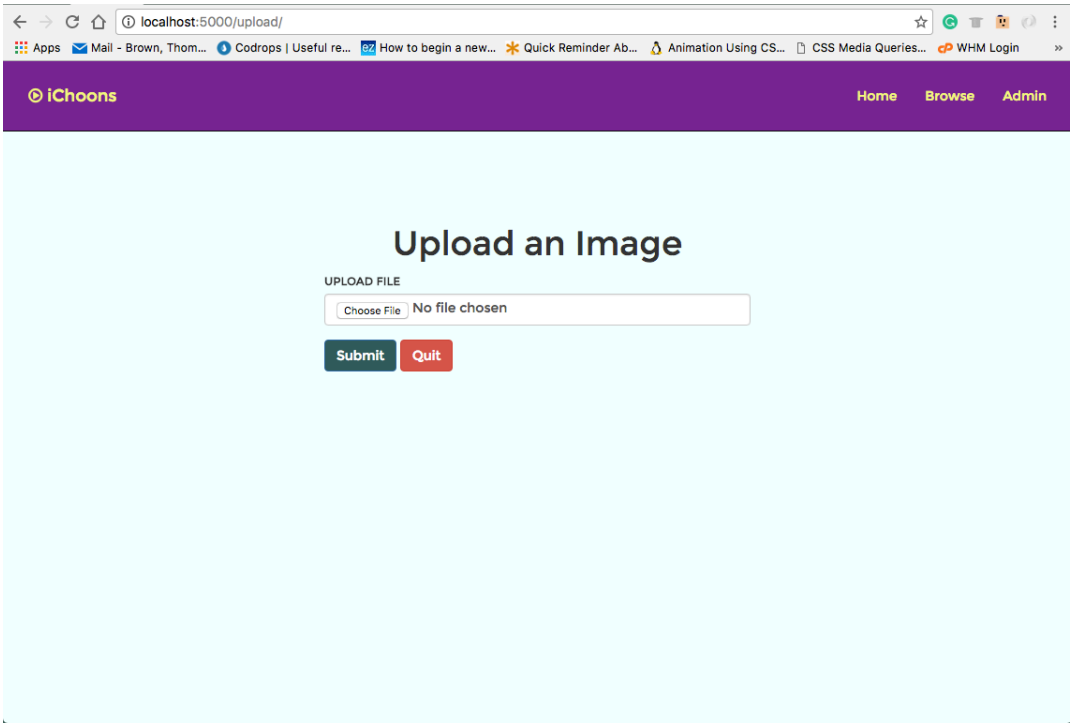

Figure a.6: Route for /upload shown above

5. Personal Evaluation

What I have learned

Being a complete beginner with Python, I feel that I have already learned a great deal from this project. Not only the skills in Python and Flask, but I have gained a much better knowledge of using Vim and GitHub.

I have learned to build a relatively substantial web app using these tools in a relatively short space of time, resulting in a steep learning curve.

That said, I sense my time management could improve as time was tight toward the end of the project meaning things got somewhat rushed.

Challenges

There were several problems throughout the course of the project, one of which was trying to get the file upload to save the file into the static folder as the matching file name to that of the JSON file. I struggled for a while but was able to get past this after searching online for a solution [10]. In the end, it was a simple solution, but it can be tricky to get a straight answer from Stack Overflow.

It was also challenging but useful learning how to integrate Bootstrap into Flask. This makes the look and feel of the application much cleaner. When I also added my custom style sheet, the final version came out looking good.

I the search function I had to come up wit a way of only allowing the three permitted genres to be input to the form. I found that 'not in' was a way of getting the result I was seeking. This time I was able to find a simple explanation on Stack Overflow to solve the problem [11].

```
  #validate against gibberish search term
  elif my_genre not in (term1, term2, term3):
    error = 'You must enter a search term - Click here to try again'
    return render_template('/browse.html', error=error)
  else:
    json_data = open('static/json/' + my_genre + '.json').read()
    result = json.loads(json_data)
    return render_template('results.html', my_genre=my_genre, result=result)

# Validate admin logon func
def auth_logon(username, password):
  if username == password:
    return True
  else:
    return False
```

Figure a.6: Route for /upload shown above

6. Summary of Resources

A number of resources were utilized in the development of the iChoons application. The main ones being the Flask micro framework and the Python programming language. Bootstrap was implemented in the creation of the program [12]. Bootstrap is an open-source, front-end framework that allows for easy styling of web applications enabling rapid prototyping of responsive web applications [13]. Google fonts [14] API is a way of adding custom typefaces to your application out with the system fonts on your server [15].

7. Bibliography

1. https://en.wikiversity.org/wiki/Python/Why_learn_Python

2. https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture

3. http://flask-diamond.readthedocs.io/en/latest/developer/writing_views_with_jinja_and_blueprints/

4. https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/Controllers

5. https://www.brainvire.com/six-benefits-of-using-mvc-model-for-effective-web-application-development/

6. https://daneden.github.io/animate.css/

7. http://bootsnipp.com/snippets/featured/fancy-sidebar-navigation

8. https://flask-login.readthedocs.io/en/latest/

9. http://flask.pocoo.org/snippets/44/

10. http://getbootstrap.com/

11. http://tech.queryhome.com/51419/about-bootstrap-framework

12. https://developers.google.com/fonts/docs/getting_started

13. https://designshack.net/articles/css/a-beginners-guide-to-using-google-web-fonts/

14. https://www.tutorialspoint.com/flask/flask_file_uploading.htm

15. http://stackoverflow.com/questions/10406130/check-if-something-is-not-in-a-list-in-python