Coursework Two Report

Andrew Hardie
40162946@live.napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09103)

Abstract

This report will talk you through the web application NASA API Services. This project was set for coursework two of Advanced Web Technologies and here you will find details on the design of the web application, any enhancements made along with the evaluation etc.

Keywords – NASA API Servicesr, Web App, Advanced Web Technologies, SET09103

1 Title

Solar Explorer

2 Introduction

NASA API Services is a web application designed for all types of users for that have an interest in space. Users of this app can use it to search NASA APOD (Astronomy Pictures of The Day) and NEOWS (Near Earth Asteroids).

Both of these utilities use the NASA API services to connect to the NASA database to download the information requested.

3 Design

I decided to write this app so that it followed the flow of a program with button type links which make it easy to use. In the following screenshot from the main page, you can see two of the button type links that I refer to.

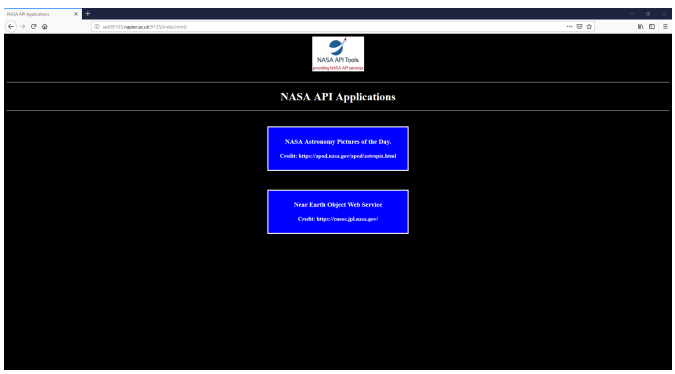

Figure 1: **Main page screenshot**

The following is a screenshot from the current working model of the NASA API Services web app -

3.1 URL Hierarchy

In figure 2 you can see a URL map of the NASA API Services web app.

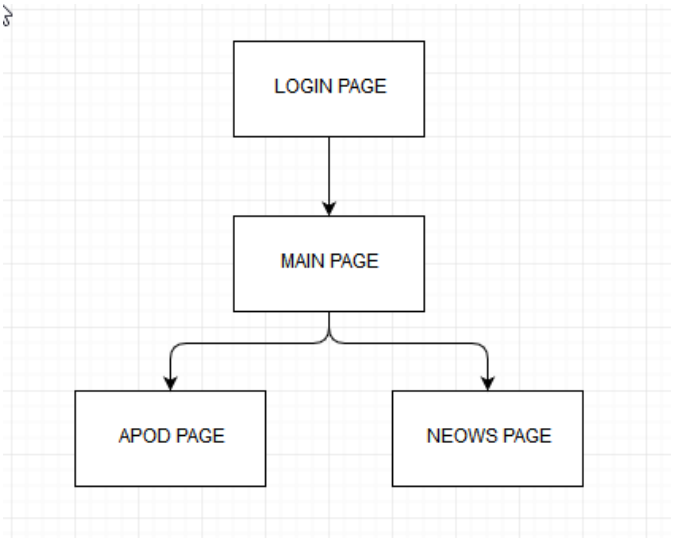

Figure 2: **URL hierarchy map**

As you can see, when heading to http://set09103.napier.ac.uk:9135/, the user is greeted by the Login Page. Here the user can type their user details, if they are registered, and access the web app features. To keep things simple, there is one user account registered which is username - 40162946@napier.ac.uk with the password - password1234.

Once this username and password are entered, the user will be taken to the main page which is the screenshot above. From the main page the user can either navigate to the Apod Page or the Neows Page.

3.2 Routing

Within my app I have setup routing for each page, this consists of code is the following format for example,

```
1 @app.route('/index.html/')
2 @requires_login
3 def index():
4     return render_template('index.html')
```

I have done this for each page of the web app.

3.3 Requests

The following requests within my web app are used within the login function to log a user in and to check if the username and password match the stored version. The following code example does this.

```
@app.route("/", methods=['GET', 'POST'])
def root():
    if request.method == 'POST':
        user = request.form['email']
        pw = request.form['password']

        if check_auth(request.form['email'], request.form['password']):
            session['logged_in'] = True
            return redirect(url_for('.index'))
    return render_template('login.html')
```

3.4 Redirects

If a user tries to go to the following URL's http://set09103.napier.ac.uk:9135/index/ then they will be greated by page 404 error normally, to handle this I have setup a redirect so that the user is sent to http://set09103.napier.ac.uk:9135/index.html instead. The same goes for http://set09103.napier.ac.uk:9135/index/apod and http://set09103.napier.ac.uk:9135/index/neows.

```
@app.route("/index/")
def index_redirect():
    return redirect("http://set09103.napier.ac.uk:9135/index.html/")
```

3.5 Responses

Throughout the code in my web app I have responses and these are used to return web templates mainly.

3.6 Custom Error Code Handling

I have used customer error code handling to give users a more relevant message when running into errors, for example, I have used a customer error message for error 404 as so,

```
@app.errorhandler(404)
def page_not_found(error):
    return "Couldn't find the page you requested or page may not exist, please check the URL in  address bar and try again.", 404
```

I have also done this for the most common error messages such as 403, 504 etc.

3.7 Static Files

I have used static files within my app mainly for images. One of these is to show the logo for the app which i have used on the main page and the login page. Within my static folder I also have my Styles folder which contain all of my .css files for styling each page of my web app.

3.8 Templates

Within my templates folder I have my login.html and index.html pages and then I have to sub-directories called APOD and NEOWS which contain my apod.html and neows.html files.

3.9 Sessions

I believe that I make use of sessions to check weather someone has logged in to the app or not. When a user visits my home url the app will check the session to see if it is True or False. If False they are redirected to the login page and if true then they can visit the web app pages.

```
def requires_login(f):
    @wraps(f)
    def decorated(*args , **kwargs):
        status = session.get('logged_in', False)
        if not status:
            return redirect(url_for('.root'))
        return f(*args , **kwargs)
    return decorated

@app.route('/logout/')
def logout():
    session['logged_in'] = False
    return redirect(url_for('.root'))

@app.route('/index.html/')
@requires_login
def index():
    return render_template('index.html')

@app.route("/", methods=['GET', 'POST'])
def root():
    if request.method == 'POST':
        user = request.form['email']
        pw = request.form['password']

        if check_auth(request.form['email'], request.form['password']):
            session['logged_in'] = True
            return redirect(url_for('.index'))
    return render_template('login.html')
```

3.10 Logging

Following the instructions within the workbook for this module, I have setup logging to work as described.

Imports

```
import ConfigParser
import logging
import bcrypt
from logging.handlers import RotatingFileHandler
from functools import wraps
from flask import Flask, redirect, render_template, request, session, url_for
```

Logging Code

```
def init(app):
    config = ConfigParser.ConfigParser()
    try:
        config_location = "etc/logging.cfg"
        config.read(config_location)

        app.config['DEBUG'] = config.get("config", "debug")
        app.config['ip_address'] = config.get("config", "ip_address")

        app.config['port'] = config.get("config", "port")
        app.config['url'] = config.get("config", "url")

        app.config['log_file'] = config.get("logging", "name")
        app.config['log_location'] = config.get("logging", " location")

        app.config['log_level'] = config.get("logging", "level")
    except:
        print "Could not read configs from: ", config_location

def logs(app):
    log_pathname = app.config['log_location'] + app.config['log_file']
    file_handler = RotatingFileHandler(log_pathname , maxBytes=1024* 1024 * 10 , backupCount = 1024)
    file_handler.setLevel( app.config['log_level'] )
    formatter = logging.Formatter("%(levelname)s | %(asctime)s | %(module)s | %(funcName)s | %(message)s")
    file_handler.setFormatter(formatter)
    app.logger.setLevel( app.config['log_level'] )
    app.logger.addHandler(file_handler)

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=True)
    init(app)
    logs(app)
    app.run(
```

```
34      host=app.config['ip_address'],
35      port=int(app.config['port']))
```

3.11 Testing
3.12 Data Storage
3.13 Encryption

I have used Bcrypt to encrypt the stored password for the user login screen.

```
1 app.secret_key = 'A0Zr98j/3yX R~XHH!jmN]LWX/,?RT'
2 valid_email = '40162946@napier.ac.uk'
3 valid_pwhash = bcrypt.hashpw('password1234', bcrypt.gensalt↩
      ())
4
5 def check_auth(email , password):
6       if(email == valid_email and valid_pwhash == bcrypt.↩
        hashpw(password.encode('utf -8'), valid_pwhash)):
7             return True
8       return False
```

3.14 Data Transports

4 Enhancements

Adding more API services would be a future enhancement goal along with hiring a design artist to make the app look more appealing. When it comes to visual design I wont win any awards for my 90s style look.

Adding a database so that users can sign up instead of usernames and passwords being hard coded would be a priority. I did test this functionality however time was against me with other module outcomes being due.

Adding a logout button to auto delete the login cookie.

5 Critical Evaluation

Working on the visual design is a must for me

6 Personal Evaluation

I would have liked to spend more time getting the user database to work with the app and adding in a registration function. I need to learn more about this.

Using the NASA API's has been a learning curve but has also been fun to use. Learning how to manipulate the data requested in JSON format was difficult but good to learn.

I believe with more time to spend learning this subject would be beneficial however I do believe that this outcome has taught me a lot even if my skills are still weak.

7 References

[1] Learning HTML - https://www.w3schools.com/ [2] HTML and CSS design and build websites by Jon Duckett [3] NASA APIs - https://api.nasa.gov/