Advanced Web Technologies

# Coursework 2: Technical Report

##### Josh Tait

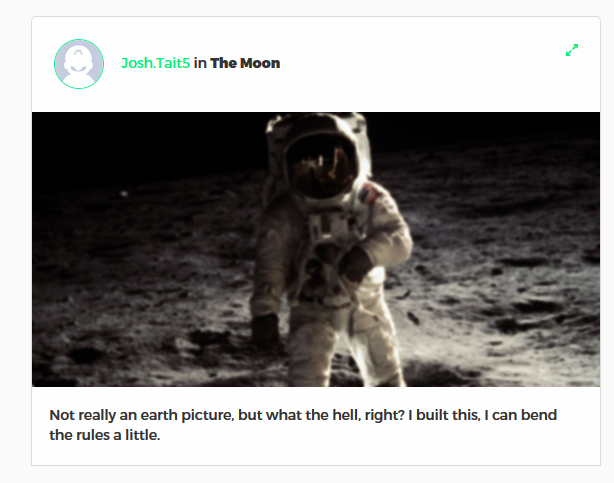
##### 40176049

Table of Contents

1. [Coursework 2: Technical Report](file:///C:\Users\Josh\Desktop\CW2_Report_40176049.docx" \l "_Toc468309642) 0
2. [Introduction 2](#_Toc468309643)
3. [Design and Build Architecture 4](#_Toc468309644)
4. [Key Features 7](#_Toc468309645)
5. [Enhancements & General Improvements 10](#_Toc468309646)
6. [Critical Evaluation 11](#_Toc468309647)
7. [Personal Evaluation 12](#_Toc468309648)
8. [List of References 12](#_Toc468309649)

# II. Introduction

The app that I have designed and developed is an Instagram/Twitter clone called Globe. It’s designed like any other social network and it’s main feature is the Feed, where people can view each other’s pictures of our planet. These could be holiday snaps or satellite imagery (though the app doesn’t have any of this at the moment) The app is driven on user-generated content called posts. Below is an example post:



As you can see, the user can post an image, state where the image was taken and an optional message below the picture. The user can also view the picture in more detail by clicking on the ‘Enlarge’ button (top right).

The App is written in Python and uses the Flask framework. The app loads pages dynamically based on the user’s route and some other requirements like whether or not the user is logged into an account or not.

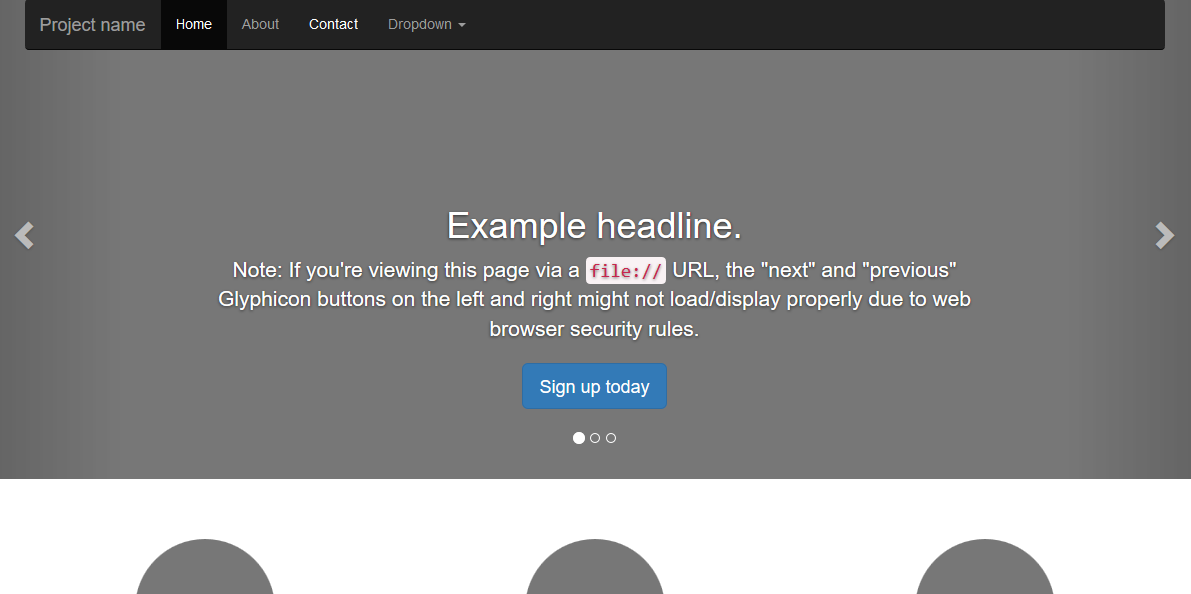
The design makes use of contemporary design theories. For example, the app uses a sans-serif font which is best for body text. The app uses bright colours which draw the reader’s attention to certain elements. For example, see the button below:



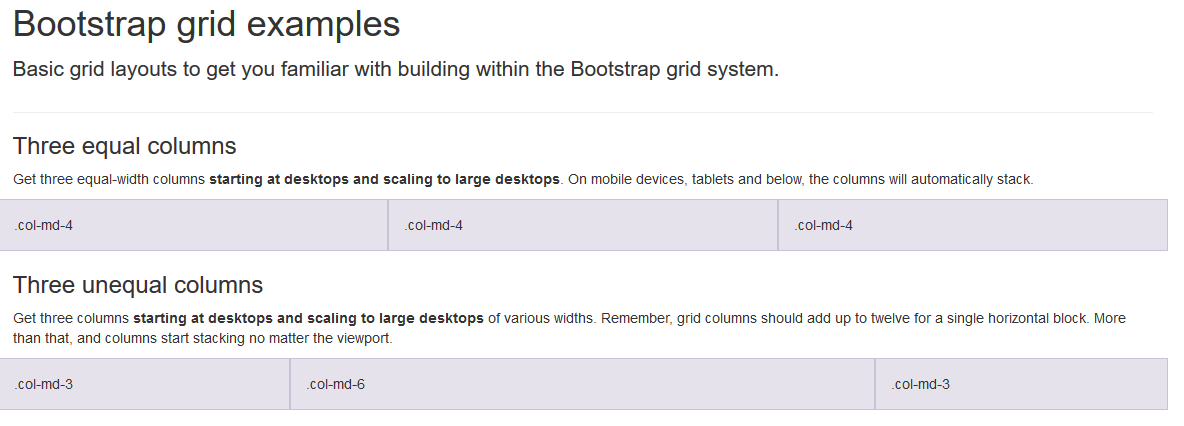
The reason for using green is because it is associated with the colour of life which is in abundance across Earth.

The app is designed with modern trends like minimalism and material-design. This is seen throughout the app with the depth in buttons (see the Floating Action Button above) 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:

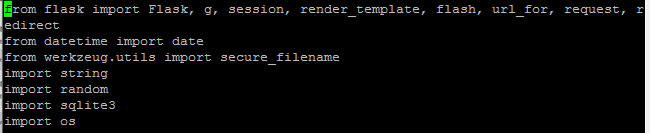


It’s grid system allows for easily placed content like the trendy ‘card’ layout as seen below:



# 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:



These modules are essentially collections of pre-written code, which can be easily integrated into any Flask project.

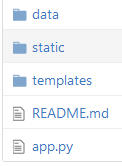
Below is a table of what each imported module does:

|  |  |
| --- | --- |
| Module | Purpose |
| Session | Used for user accounts and general user authorisation |
| Render\_template | Loads a template from the ‘templates’ folder |
| Flash | Used to send the user feedback on the fly when a template is loaded |
| url\_for | Used to return URL Routes |
| Request | Used to retrieve form data from the user (like post content, images etc) |
| Redirect | Similar to url\_for, redirects the user to different URL Routes (“/”, “/feed/”, “/profie” etc) |
| Datetime / date | Used to note when the user posted something |
| Secure\_filename | Used to secure the filenames of images uploaded by the user (for security purposes, see notes on security for more details) |
| String, random | Used to create a secret key by generating a string of random numbers and letters. Also used to create User IDs and Post IDs |
| Sqlite3 | Used to access the database, where all of the user’s data is held. Note: images are not stored here just identifiers like Post ID, username, password etc. |
| Os | Used to access the file system of the ‘server’. This module is used to save files to the server, as well as renaming, deleting and creating user files and directories. |

It was anticipated that a table of each URL Route and function would be listed and described, but due to its enormity, it has been omitted from this document.

##### App Structure

Below is a screen shot of the app’s hierarchy:



Note: README.md is just a markdown file for the Git repository and doesn’t affect how the app runs.

As you can see in the diagram, there is a python file, ‘app.py’ that essentially runs the website. ‘Data’ stores the database. ‘Static’ stores all of the resources like the user’s content (images for posts, profile pictures and cover photos) as well as the resources needed to render the web pages properly (CSS, JavaScript etc)

Here is an example work flow demonstrating how the app works when the user accesses the feed.

* The user loads the feed by typing in /feed/ in their URL bar
* App.py loads the function get\_feed()
* App.py runs all the code in the function
  + App.py accesses the data on the database (in the /data folder seen above) via SQL Query
  + App.py returns feed.html, the template used to display the feed to the user
  + App.py also sends the data returned from the Query ran in the function to this page, which Jinja2 then parses and displays to the user

This work flow above describes how every action the user could make is interpreted in the website.

# 

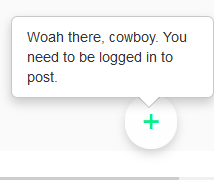
# Key Features

Below is a comprehensive list of what the user can do with the App:

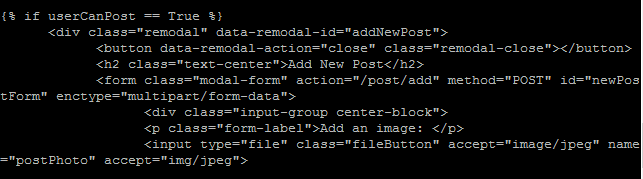
* Profile: create an account and use the app. Users must be logged in to post, but anyone can view posts
* Edit Profile: A user can update their profile (location, biography, cover and profile pictures)
* Add Posts: Each logged in user can add posts to the website. There is no functionality to edit or delete posts at this time.
* View Profiles: If a logged in user navigates to /profile/ their profile is loaded. The user can also navigate to someone else’s profile by clicking on their name in the feed.
* Search: anyone can search for users registered to Globe via search bar
* A relaxed, informal register. When users interact with the website, they are greeted with informal language, which makes the experience more personal and less ‘robot-like’. This is seen in all of the error messages seen in the website.

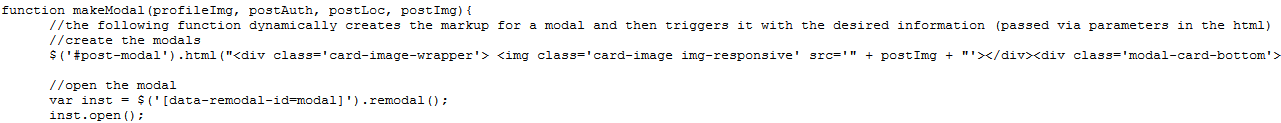
Below is a list of what the app is capable of:

* Dynamic Content: as previously stated, the app effectively runs on ‘auto-pilot’ and can display new content to the user
* Intelligent App routes: the website can respond to what the user types in their URL bar to an extent. For example, if a user types /profile/ and they’re logged in, it will display their profile. The app has been designed to feature an intuitive URL Hierarchy (where possible) by allowing multiple ways for a user to navigate.
* Message Flashing: The app can also provide limited user feedback, like when a user enters an incorrect username/password combination
* Writing to the server: Users can add posts to the App. A user can add an image (which is stored in their own personal folder in /static/user-uploads/). The app can check if the image has a suitable and safe file extension and can save, rename and delete files where necessary
* User Authentication: Users can access certain pages and carry out certain actions when not logged in. For example, a guest user cannot post anything to the website. This is achieved by checking if there is a session set for the user. If there is no session set, the button is locked and a message is displayed to the user:



* There are other features of user authentication, like determining whether or not a user can edit a profile. Only logged in users can edit their own profile, and this logic in Jinja determines what will be rendered to the user in the template. From the diagram below you can see that if the UserHasControl = 1 (if the user has the power to post something, i.e. if they’re logged in), render the following HTML to allow the user a means of posting content:

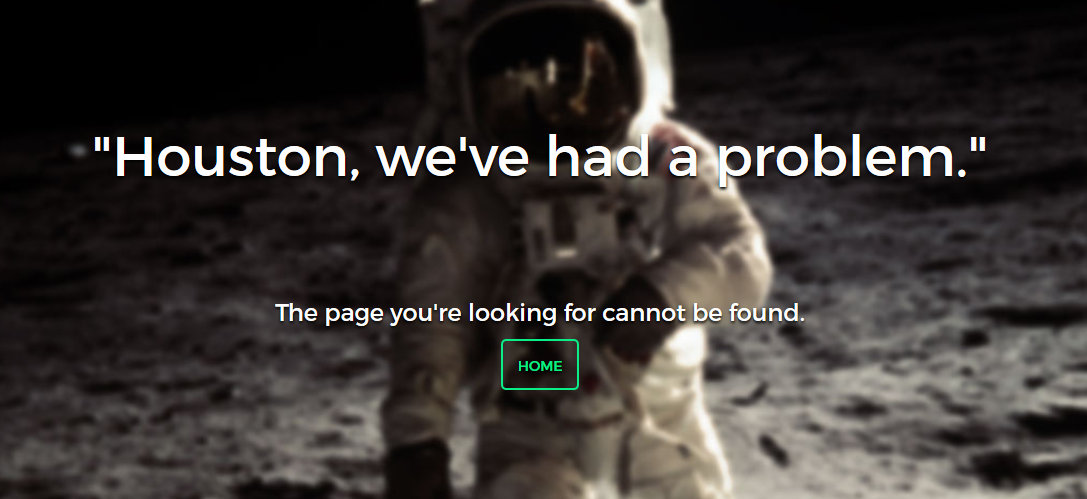


* Creating Dynamic Modals pre-loaded with content: When a user clicks on the ‘enlarge’ button on a post, this triggers a function in the page which creates a modal in the Document Object Model (DOM), and fills it out with content via JInja2 Tags. Below is some sample code illustrating how a modal is created on a button click:
* 

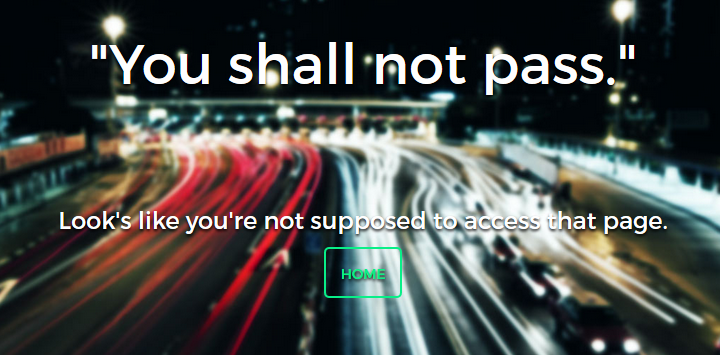
The modal is created via jQuery and then initialized and triggered. All with the click of a button from the user.

##### Responding to user input via error messages

The website has several functions which provides feedback to the user. This can be seen in the error messages:



404 – page not found



403 – forbidden access (user is not logged in and is trying to access their profile or post something)

The ‘sense of humor’ is popular among websites today and is seen being used by Google (for their elobarate April Fool’s jokes), by GitHub (for their Star Wars reference in their 404 page) amongst others. This is intended to sub-consciously make the user think that Globe is popular and professional to the standard set by today’s most-popular websites.

# IV. Enhancements & General Improvements

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

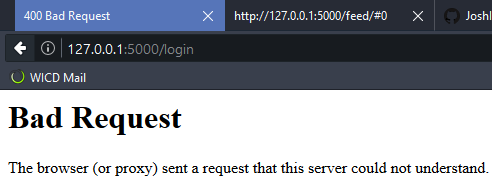
If time had allowed the following features could be implemented:

* Password encryption/Hashing. Passwords are stored in plain text which can be potentially very dangerous if the website is compromised, as the database stores user’s personal data like an email address.
* Logging. It would be very beneficial to log certain events that occur when the app is deployed and being used by a live audience. For example, noting when a user has more than 3 failed log in attempts: this could highlight a ‘brute-force’ attack on a user’s account.
* Input Sanitisation: it is a well-known phrase to ‘never trust user input’ in the web development field. If input fields are sanitized it means that the website is protected against SQL Injection: a method used by attackers to run arbitrary code by putting raw SQL into an input field. This is especially dangerous for the app as it runs via SQLIte3, which makes this type of vulnerability a very real and dangerous possibility.
* A back-end/moderation system for administrators. This could help censor inappropriate content which will most likely feature on any website that allows user input. This feature would relate to a ‘Report Post’ feature where users could flag inappropriate or offensive content. The reason for flagging inappropriate content would be to increase the potential user demographic. A larger age range means that potentially more users can access the website, increasing traffic and, in an ideal world, making the website more money.
* Monetization (Through an ‘Upgraded Account’): It would be a good idea to offer users ‘extra features’ for a fee. This ideally could cover running costs of the website and make the company running the website money. This money could then be invested on better hardware for the website which could improve website performance.

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

Overall, the website works well. It is robust enough to work effectively and efficiently. However, there are still error messages being returned, when the user does something un-expected:



This effectively breaks the app for the user until they visit another page. This isn’t very user-friendly, as users are unable to diagnose the problem and recover from it.

Occasionally, the app will hang for several seconds when the user quickly jumps between pages (although this has only been experienced on a local server, which may not be produced when the app is deployed on a live server with better hardware performance).

# 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).

The main challenge I faced was dealing with user’s images and folders. It was very difficult to come up with an algorithm which would handle of the issues that could occur: the user doesn’t have folders to store files in, the user has tried to upload an .exe file, etc. After creating my own algorithm (which was messy, large and time-consuming to debug), I found a library did this for me. This was perfect until it no longer ran in Levinux. After spending hours trying to get the library to achieve this, I had to create a new, better algorithm, one which was mostly provided by Flask anyway. I learned that it is important to research problems before ‘diving head first’ into a problem which was very time-consuming and prevented key features from being implemented, as well as a poorly written report.

# VIII. List of References

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

## References

CodePen. (2016). *Material Design Box Shadows*. [online] Available at: https://codepen.io/sdthornton/pen/wBZdXq [Accessed 30 Nov. 2016].

Docs.python.org. (2016). *8.1. datetime — Basic date and time types — Python 2.7.12 documentation*. [online] Available at: https://docs.python.org/2/library/datetime.html [Accessed 30 Nov. 2016].

Dreamlovewallpapers.com. (2016). *Header Image*. [online] Available at: http://dreamlovewallpapers.com/wp-content/uploads/2015/02/Golden-Gate-Bridge-View-San-Francisco-Wallpaper.jpg [Accessed 2 Nov. 2016].

Fillmurray.com. (2016). *Fill Murray: Placeholder images*. [online] Available at: http://www.fillmurray.com/ [Accessed 30 Nov. 2016].

GitHub. (2016). *VodkaBears/Remodal*. [online] Available at: https://github.com/VodkaBears/Remodal/releases/tag/1.1.0 [Accessed 30 Nov. 2016].

Instagram. (2016). *Instagram (design inspiration)*. [online] Available at: https://www.instagram.com/ [Accessed 30 Nov. 2016].

Julienmelissas.com. (2016). *No Gutter Column Trick for Bootstrap*. [online] Available at: http://julienmelissas.com/no-gutter-column-trick-for-bootstrap/ [Accessed 30 Nov. 2016].

Python, R. (2016). *Random string generation with upper case letters and digits in Python*. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/2257441/random-string-generation-with-upper-case-letters-and-digits-in-python/23728630#23728630 [Accessed 30 Nov. 2016].

reddit. (2016). *Help with [ImmutableMultiDict' object is not callable] when getting data from a form • /r/flask*. [online] Available at: https://www.reddit.com/r/flask/comments/4be1vg/help\_with\_immutablemultidict\_object\_is\_not/ [Accessed 30 Nov. 2016].

Stack Abuse. (2016). *Python: Check if a File or Directory Exists*. [online] Available at: http://stackabuse.com/python-check-if-a-file-or-directory-exists/ [Accessed 30 Nov. 2016].

Note: where a date is omitted in the reference list above, the respective references were accessed on many occasions in November 2016. Some images do not have a reference (I forgot to take not of them). I do not own any images on the website, and they are property of their respective owners.