Coursework 2 Report

Marc Inglis

40402355@live.napier.ac.uk

Edinburgh Napier University – Advance Web Technology (Set09103)

Instaslam

Introduction

Instaslam is a Python-Flask powered Instagram clone with modifications. It makes use of the Jinja2 engine to display data that is being stored within a SQLite3 database. The requirements for running this web project can be found in the appendices at the end of the report.

This web product allows users to create their own personalised profiles, attach profile photos to their accounts to make themselves identifiable and follow other users, to keep up to date with media that appeals to each individual. It also utilises 'Uploadcare'; a widget that allows users to upload photographs, edit them and add effects to these images. They are then sent to the online storage, which displays them on the website I have created with the upload users name attached.

The images (posted only by the relevant accounts that you follow) appear with the most recently uploaded image at the top of the page, descending the page in chronological order. This ensures the user has the most efficient and relative experience using the website as they stay in touch with the most recent posted material and avoid seeing duplicated posts. The usernames above the images also act as links to the profiles of the uploader, providing the user with an individual experience that allows them to navigate profile's they personally find interesting.

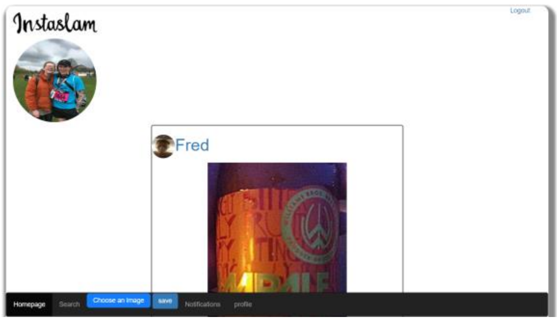


Figure 1. Homepage – Once the user has logged in, they will find a photo feed in chronological order.

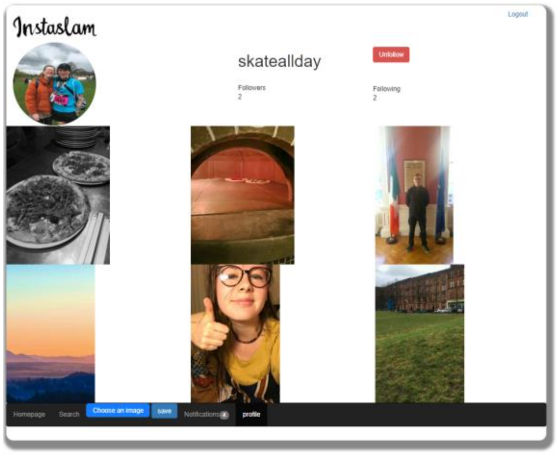


Figure 2. Profile page – This is all the photographs that the profile owner has uploaded. There is also the option to follow or unfollow this user here. This provides an important overview of each user.

Creating a website must start with the target audience at heart. For businesses, artists or promotional accounts, this provides a quick visual overview of their style, product range and ethos of the company. Even for individuals, this page is a summary of the content they release.

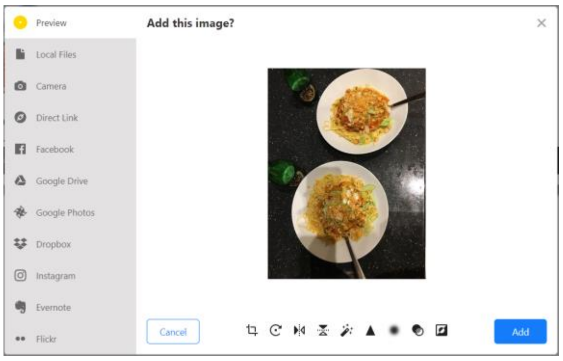
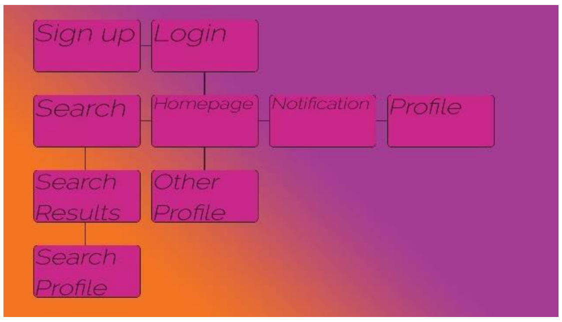


Figure 3. Uploadcare's image upload widget. This allows users to easy upload and edit images, to create a high quality, personalised aesthetic style

Design

The design for this project is largely inspired by its predecessor: Instagram. Using bootstrap and a few custom CSS tags I have managed to replicate the simple, legible layout that encourages the user to scroll down the page and the eye-catching colours for the logo. I feel it is important to create an easily navigable website, I have maintained the website name present in the top left-hand corner throughout the website so it is always easy to navigate back. Through photoshop, I designed my own logo, echoing the same font use and colour scheme that makes the brand so iconic.

On the profile page of each user, I have set the images out in rows of three, instead of single file, also in chronological order. I noticed some users on Instagram like to play around with the 'three image per row' arrangement to their advantage, perhaps giving each row a similar theme or colour scheme, it allows the user to apply their own style to their Instagram feed and allows other accounts to quickly review whether their material is interesting for them to follow. The navigation bar is always constant at the bottom of the screen, allowing users easy access to the main sections of the site. The link to the image uploader is also imbedded into the navigation bar so that users can upload photographs from any point in the website.



The site starts off with a login page, that also links to a sign-up page for people that have no account or require a new one. A user needs to be logged in to the Flask Sessions to continue through to the main section of the web product: the homepage. From here, the user can access their own profile, scroll through the homepage, go to the search page or access their notifications.

From the homepage, it is possible to click on a user's name or uploaded picture to travel to their profile. This can also be achieved by clicking on the search tab, typing in their name and then clicking on their profile photo that appears on the results page. This ensures the website is slick and easy to use.

Enhancements

I aspire to eventually complete my website, by adding popular features such as liking images, commenting and tagging other users. The so called 'social experience' of these websites such as Facebook and Instagram, are what make the sites so popular and widespread.

Whilst these features are beneficial to users, I believe that with the huge variety and number of users generally produces an excess of poor-quality posts on these websites, an element of 'quality' is being lost. I believe a richer social experience would emerge from introducing a restricted number of 'likes' to be used each day. Therefore, the value of a 'like' is increased and it would encourage users to produce higher quality material to their profiles. It would also cause users to think more carefully about how many people they follow and really find high quality and relatable accounts.

Another enhancement that could bring this version of the project into the future would be allowing users to upload video files as well as photographs. Having multiple types of media available would help appeal to a wider market.

Critical Evaluation

I created a web app that allows users to create a profile that includes a profile picture. Once they sign in their login will be kept in a cookie session until they decide to logout. This is done through Flask sessions, the code for this is in the appendices of this documents. If anyone should find themselves on the site without logging in, they will be returned to the login page. This allows control over what content the users will be allowed to see which a key feature of social media, especially in the light of recent highly published court cases. The new General Data Protection Regulation in May 2018 helps to ensure the users safety. In a world where so much information is available online, it's very important to offer privacy and for the website to safeguard the users.

A user needs to register to be allowed onto the website. They submit a username, a password and a profile picture of their choice. Security leaks are serious issues that could lead a lot of website participants losing their value data. To combat this, I have learnt to use and applied Flask-Bcyrpt: a flask extension that allowed me to hash passwords going into the database. This means if the database was compromised, the users would know their data was protected.

From here, there is a homepage with a photo stream. Each photo is stored offsite with its URL stored in a database alongside the username of the uploader. This is done through a form which is stored on the navigation bar. When the image is being uploaded to the uploadcare website a request method grabs the URL and stores it in a database table called 'photos', along with the username. This works well as it does not put any strain on the server or database I am working with as all I am saving is profile pictures and strings of text.

There is a search function that will allow users to search for each other's profile pages. The search function works by requesting data from a Flask WTForm. This string is then stored in another Flask session. The app then passes this information into the SQLite database through a 'SELECT FROM WHERE' query. It checks against all the usernames that are stored in the USER table of the database and then uses a For Loop to cycle through the results. If no results come up, the user simply gets a message asking them to try the search again. If successful, the user will have access to see what is on their pages and can choice to follow or unfollow them.

The profile will also have the number of followers and following displayed underneath their name. To ensure that the correct version of the profile page is loaded, there is another database table called relationships. This stores the unique identification number that is assigned to each user when they create an account. There is a piece of logical programming that checks for an instance when the user that has been search for appears next to the user that is logged in through the session's cookie. If this occurs, then the profile will appear with an "Unfollow" button. If this is not the case and there is no case of these two identification numbers aligning in this manner, the "Follow" button will appear on the profile page. This stops users from following each other multiple times which would break the followers/following numbers that are displayed.

The navigation bar that is available on everyday after the login page always easy movement between all the sections of the website. As well as this it has a form in it that allows users to upload photographs and edit them.

I have created a notification database that stores two user identifications numbers and a value that indicates what notification message should be displayed. If a user finds another's page that they wish to follow they can click the "Follow" buttons on their profiles which will trigger two functions: one that stores this interaction in the relationships table and another in the notification table of the database. However, this only worked if the exact pre-set requirements are made before selecting the notification tab. It works poorly so I have removed it from the final edition of the code.

Personal Evaluation

I feel like I have learned a lot about how Flask works with SQLite3. I feel quite confident now in creating larger scale Flask apps and can say I have a good grasp on how to program Python comparison to before this module where I had never encountered the language before.

There are many Django Instagram clones out there since this is the language that it is built in. However, Instagram started off as a Flask app, so I knew it would be possible to recreate. A lot of the challenges I had were trying to see how parts of the Django app was created and replicate it in Flask. Uploadcare has a lot of good documentation on how its Content Delivery Service works, but it is almost all written for Django. After reading through a document that was written for PHP however, I manage to figure what would be needed for the widget to work within a Flask App.

If I were to develop the project, I would complete the page using the suggestions above. With so many different aspects to the website, it was fascinating to learn how to display the different interactions between users and functions. I have thoroughly enjoyed comprehending and creating backend social media platforms and being able to apply functions from a purely visual stance on the website, through to mirroring this in it's functionality and programming. Learning through mimicking a website allows you to challenge yourself to recreate functions so that you fully understand the process of each function. The important thing to reflect upon when recreating a website is also to identify it's flaws or possible ways to create alternative methods or enhancements to the website.

References

http://www.sqlitetutorial.net

-SQLite3 database walkthrough

http://jinja.pocoo.org/

-Jijna2 Documentation

https://flask-bcrypt.readthedocs.io/en/latest/

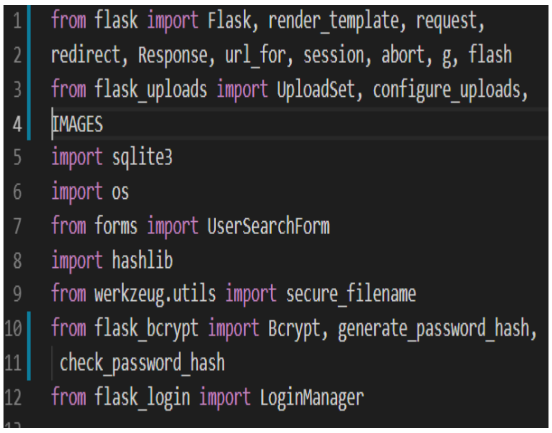
-Flask Documentation

https://flask-wtf.readthedocs.io/en/stable/

-Flask forms documentattion

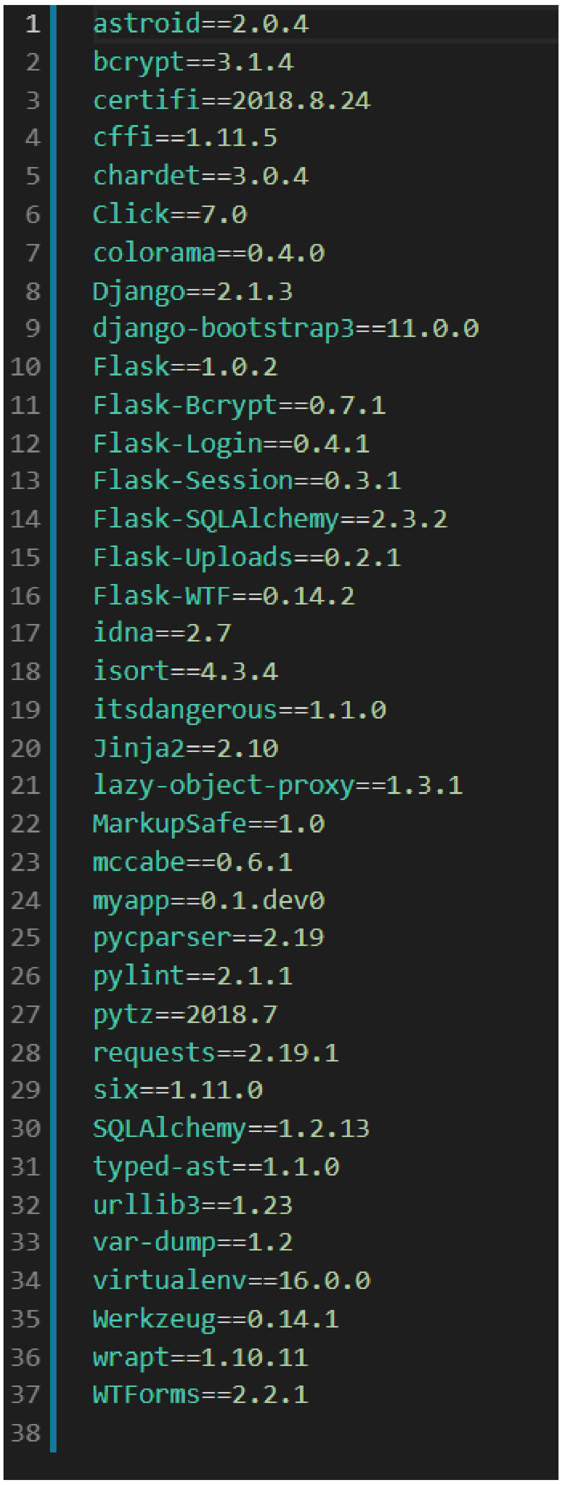
Appendices

List of Imports



```
from flask import Flask, render_template, request,
redirect, Response, url_for, session, abort, g, flash
from flask_uploads import UploadSet, configure_uploads,
IMAGES
import sqlite3
import os
from forms import UserSearchForm
import hashlib
from werkzeug.utils import secure_filename
from flask_bcrypt import Bcrypt, generate_password_hash,
 check_password_hash
from flask_login import LoginManager
```

Requirements for project



```
astroid==2.0.4
bcrypt==3.1.4
certifi==2018.8.24
cffi==1.11.5
chardet==3.0.4
Click==7.0
colorama==0.4.0
Django==2.1.3
django-bootstrap3==11.0.0
Flask==1.0.2
Flask-Bcrypt==0.7.1
Flask-Login==0.4.1
Flask-Session==0.3.1
Flask-SQLAlchemy==2.3.2
Flask-Uploads==0.2.1
Flask-WTF==0.14.2
idna==2.7
isort==4.3.4
itsdangerous==1.1.0
Jinja2==2.10
lazy-object-proxy==1.3.1
MarkupSafe==1.0
mccabe==0.6.1
myapp==0.1.dev0
pycparser==2.19
pylint==2.1.1
pytz==2018.7
requests==2.19.1
six==1.11.0
SQLAlchemy==1.2.13
typed-ast==1.1.0
urllib3==1.23
var-dump==1.2
virtualenv==16.0.0
Werkzeug==0.14.1
wrapt==1.10.11
WTForms==2.2.1

```