Petstagram Web Application
CSE312 Final Project Report

Developers:
- Jackson Kamp (jakamp@buffalo.edu)
- Joyce (joycesom@buffalo.edu)
- John (cantopra@buffalo.edu)
- Dom Sciarrino (dpsciarr@buffalo.edu)

Outline:
- Table of Contents
- Introducing: Petstagram
- Project Requirements
    - Secure Authentication and User Accounts
    - Users can see all users who are currently logged in
    - Direct Messaging (DM) and DM Notifications
    - Storage, Hosting, and Sharing of Multimedia Content
    - Live Peer-to-Peer Interactions using WebSockets
- Back-End Technologies
    - Flask
        - Flask
        - Flask-WTF
        - Flask SQLalchemy (if we don't use MySQL or Mongo)
        - … (other flask-based libraries we end up using)
    - Python Dependencies
        - Include other python dependencies here
    - Docker Compose
        - An overview of Docker Compose
        - Docker Compose in Petstagram
- Front-End Technologies
    - HTML5/CSS
        - If we use bootstrap or other libraries, include them here
    - Javascript
        - If we use bootstrap or other frameworks, include them here
- Appendix
    - (if we decide to put images in the appendix instead of in the report itself)

Table of Contents

Introduction

Our development team is passionate about our pets and animals. We wanted to create a space where people can share their pets with the rest of the world. We set out to create an app that allows users to share photos of their pets, interact with other users and their posts, and make a live artistic environment for friends to create images for their profiles. Petstagram is our passion project and we're excited to share it with the world. In this report, we will discuss the technology behind our program, including what each part does, how it accomplishes their goals, and licenses and frameworks we used to create our app. We'll dive into our front end technologies, backend databases and structures, and the requirements for the project.

Project Requirements

Secure Authentication and User Accounts

As a visitor to the site who is on the homepage and not logged in, there should be an option to sign up. Clicking on this option should redirect me to signup.html, where I should see a form containing a username, email, password and confirm password sections. Properly filling out this form with new login info and pressing the submit button should create the account and store the information in the database. When I go to log in, if I use the same username and password as when I made the account, I should log into my account. I will know that I have successfully logged into my account if I am on the homepage and it says “You are logged in as: “ followed by your username. You will also have the option to click the ‘Profile’ and ‘Logout’ buttons in the header.

Users Can See All Users Who Are Currently Logged In

Direct Messaging (DM) and DM Notifications

Storing, Hosting, and Sharing Multimedia Content

Live Peer-to-Peer Interactions Using WebSockets

Back End Technologies

Flask

Flask is the backend web framework written in Python that is used in this project. This framework handles the routing, rendering of html templates, and sending data back and forth from the end user. It handles the parsing of requests and allows us to easily utilize and manipulate the data. It accomplishes this by utilizing libraries written in python that automatically handles these annoying, granular tasks, making it easier for our team to not think about the tiny technical bits, and instead focus on the design and logic of the website. The following flask libraries/functions were used:

1. Route: Flask opens a TCP socket connection to the client, parses any incoming data, and packages data to send back. The library uses python logic to splice the data from the socket and provides logic to make the task trivial for the programmer to organize the handling of paths by the developer. It also routes back responses from the server, packaging the files as well.
   https://github.com/pallets/flask/blob/master/src/flask/app.py
2. Render Template: This library took care of rendering html templates for the developer, allowing them to feed information into the function, and the renderer automatically replacing the text and returning the rendered file. This is done through python logic, taking another task off the hands of the developer.
   https://github.com/pallets/flask/blob/master/src/flask/templating.py

The license used by Flask is BSD-3-Clause License. No other licenses could be found attached to the project.

Python Dependencies

Docker Engine & Docker Compose

Docker is a set of services used for containerization of software at the operating system level. The purpose of this is to isolate one service from another. It does so by containerizing the source code & its dependencies, and by establishing networking channels that are well-defined between containers.

Two distinct object types are relevant to our project. The first is a Docker container. Containers are the isolated environments that run the applications. The second is the Docker image. Docker images are templates that Docker uses to build the containers. Images are the primary means by which applications are stored, shared, and shipped using Docker.

Using the command-line interface 'docker', one can manually develop and run containerized applications. This is handy when we are dealing with one container, but becomes cumbersome when having to manage multiple containers.

One tool that comes with the Docker engine is Docker Compose. Compose allows us to specify how to build more than one container in a docker-compose.yml file. We can, for example, design a container hosting the main application and design another container responsible for managing a database. The docker-compose.yml file allows us to specify how those containers are to communicate.

Docker licensing depends on which version is being utilized. The Enterprise version is proprietary, whereas the Community Edition is licensed under the Apache License 2.0. This means Docker Community Edition is free for distribution and modification without royalty concerns.

The github repository for Docker has a LICENSE file specifying an Apache License Version 2.0. https://github.com/docker

In our project, we specified a dockerfile and a docker-compose.yml file for building and linking a MongoDB database. See the next page for the file content.

dockerfile:

```
FROM python:3.8
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . .
EXPOSE 5000
ADD
https://github.com/ufoscout/docker-compose-wait/releases/download/2.2
.1/wait /wait
RUN chmod +x /wait
RUN export FLASK_DEBUG=1
CMD /wait
CMD /wait && ["python", "-m", "flask", "run", "--host=0.0.0.0"]
```

docker-compose.yml

```
version: '3.3'
services:
    mongo:
        image: mongo:4.2.5
    server:
        build: .
        environment:
            WAIT_HOSTS: mongo:27017
        ports:
            - '8080:5000'
```

Front End Technologies

HTML5/CSS

HTML Templates

Our site is built off of a set of HTML templates. HTML templates allow us to take information from the user and tailor the site to meet the user's needs. base.html is the page where most other pages extend from. This file contains the Header and the homepage links to other pages. If the user is logged in, it should show links to the profile and logout page. If the user is not logged in, it should show links to the login page. index.html is our main page, and describes if the user is logged in or not. If the user is not logged in, a prompt to login and signup are shown on the homepage. The login page is a form that allows the user to submit a username and password into the form. When the user submits the form, it should access the database to see if the login information is correct. If there is an error, an error message will appear. The signup page prompts the user to enter a username, email, password, and confirm password into the form. Pressing submit on the form should create a new account in the database. If a user is logged into their account and on the homepage (index.html) there should be a link to the user's profile page. Clicking on that link should bring the user to the user.html page, which contains the user's profile information, the option to edit the profile, and previous posts. If the user chooses to edit their profile, they will be brought to the editProfile.html page, which allows the user to change their username and about me information.

HTML Injection

One of the easiest methods for attackers to hack a website is HTML injection. To prevent attackers from posting unwanted HTML (and therefore potentially unwanted javascript) to a website it is necessary to escape HTML ASCII characters to their Unicode equivalent. The easiest way to do this is to convert greater-than and lesser-than characters to > and <, respectively, when reading data input from the client.

JavaScript

Appendix