# Freshworks Duck-Feeding App Assignment Writeup

**Application URL:** https://dahv-freshworks-assignment.herokuapp.com/
**Github Repo URL:** https://github.com/dahvreinhart/duck-feeding-app-assignment
**Total time spent on development:** 10 hours

## My Approach to The Problem

To the requirement of taking user input, I chose to go with a single page form that is displayed on initial page load. This is a standard way of taking user input which has the bonus of being quite easy to implement in a short period of time. It also has the added benefit of being very clear UX which is advantageous when dealing is less computer savvy users. Since the problem description hints that 'little old ladies' might be using this application, I wanted to keep things simple.

To the requirement of making user input data persistent, I chose to use a mongo database (which is hosted in the cloud on mLab). The rational for this decision will be explained in the Technologies section below. When the user input form is submitted, it gets saved into a database model and stored in the aforementioned database. Once saved it is queried on for both the reporting functionality and the recurring submission functionality of the app.

To the requirement of reporting on the data collected, I implemented a simple CSV download feature of a dump of the database. This was primarily out of time concerns and ease of development. It would have been interesting to build an on-platform reporting system using a graph making package but that would have almost certainly pushed development time outside the allowed window. The CSV download feature allows people to download a full dump of the database to do with what they will. Coupled with a program such as Microsoft Excel, the possibilities for data manipulation and presentation from that point on are endless. Additionally, I made the decision to think of the duck feeding project as open source and so I didn't lock the CSV download feature behind a permissions wall.

To the optional requirement of enabling automated submissions, I chose to allow this but in a defined way. I implemented a feature to put one's submission on a daily recurring submission schedule for one week. This option is selected on the user input form and is optional to the user. I chose a one-week time period because of practicality. An indefinite recurring schedule has the disadvantages that one can never edit their submission again and if they ever want to input new data they would be doubling up their responses thus making incorrect data in the database. People can generally forecast their behavior accurately for a one-week period. Thus, users who wish to use this feature simply must use the application once per week which cuts down on the number of submissions they need to make substantially. An important note is that the asynchronous job that handles these recurring submissions is set to run early in the morning (3:30PST). This is supposed to mimic a real deployment where processes that have the potential to increase latency via database load are run during low-traffic hours. However, this requires the server hosting the app to be running at that time. Since Heroku's free servers have auto-shutdown, and since I didn't want to pay real money to host this example assignment, this will most likely not happen (unless a user happens to be on the app at the given time). In a real deployment scenario with a persistent server, however, this would not be a problem.

**The Technologies I Used**

Below I will list the major technologies used to create my app and a brief rational for choosing each one:

- ***Flask with Flask-PyMongo***: I chose Flask due to how easy it is to quickly get a simple web app off the ground. Flask also has a lot of really helpful documentation online which I expected would help when solving any potential issues I would run into. Additionally, Flask plays nicely with some of the other tech that I wanted to use to complete this assignment, namely MongoDB and Heroku. Case-in-point, I was able to simply download the Flask-PyMongo package, throw in a DBURI and a DBNAME and I had a persistent database ready for further development.

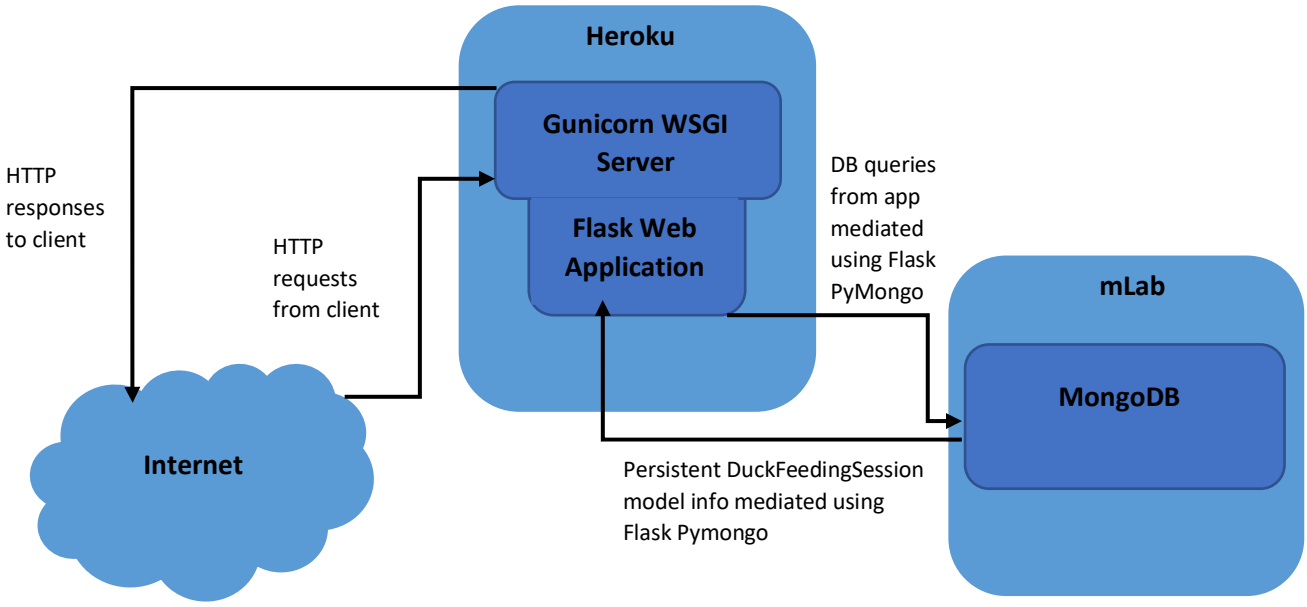
- ***MongoDB hosted on mLab***: I chose MongoDB as my database primarily because it is what I have the most experience with***.*** Mongo is what I use at my current job and I didn't want to stray away from that knowledge when there was a time limit in play. I chose to host my DB on mLab because it is simple to get going and is very nice for development as you have a really easy and fast way of visually checking the state of your DB.

- ***HTML/CSS with Bootstrap***: For my front-end, I chose to rely on Flask's templating system in concert with Bootstrap. I didn't really need much more to complete the task. One of the main things I would want to do if I had more time, or if I was coding this project as a legitimate side project, would be to go deeper into the front-end design and make the site look more polished and modern.

- ***APScheduler***: For the automated submission feature on the site, I used the Advanced Python Scheduler, or APScheduler, due to the fact that it is very simple to set up, has great documentation and provides asynchronous functionality.

- ***Heroku running Gunicorn***: For my deployment, I am using Heroku which is running Gunicorn as the WSGI web server. Gunicorn, in turn, runs my Flask application. I chose Heroku because it offers free hosting and is quite easy to setup. Additionally, on the free tier of Heroku you are allowed more than one process or thread to be running concurrently which is something I needed for my scheduling functionality to work correctly. The use of Gunicorn was mainly because of the limited nature of Flask's built in web server which only allows one user to make requests at a time. This probably would have been sufficient for this assignment but Gunicorn is so easy to use that it seemed like I should just go ahead and enable it so that more than one user could use the site's functionality concurrently.

**Component Diagram**




**Database Model Diagram**