Project Two Readme

Mark Webster

NOTE: The screencast for this project can be found at: https://youtu.be/C1h2VRG9Lyk

The desired functionality of this project was to create a dashboard built with Dash (https://dash.plotly.com/) and Python that would serve to manipulate and display data contained in our MongoDB database. Specifically, the information pertained to an animal shelter in Austin, Texas. We needed to show all data in a datatable, be able to filter it according to Grazio Salvare's needs, and show infographics built using some table information. To that end, radio buttons where created and linked to a callback function which implemented queries using our CRUD functions to filter and display certain records. Specifically, we were able to show specific breeds, sexes, and ages for certain dog training needs. In terms of our infographics, a pie chart was built to show the percentage breakdown of the animals' outcomes at the shelter—adoption, euthanasia, *et cetera*. We also built a geolocation map that used the latitude and longitude found in each animal's data to tag them on an interactive map widget.

Because MongoDB (https://docs.mongodb.com/)is a NoSQL database solution, it has the ability to handle and manipulate many different types of data within its collections. It is also easily used with Python thanks to the PyMongo library (https://pymongo.readthedocs.io/en/stable/), which provides a deep library of functions to manipulate data, collections, and databases in MongoDB through Python commands, as well as authentication. MongoDB was the back-end of choice for our webapp. Python itself is easier to write given its usage of whitespace and indentation to build function structure, instead of more specific vernacular that can found in older OOP languages such as C++ or Java. While Python is interpreted rather than compiled, so it runs a bit slower than the languages previously mentioned, the speed tradeoff during the actual coding process made Python a clear choice for our mid-tier layer. Our CRUD functionality and authentication that actually linked to the MongoDB database was done in Python using the Pymongo functions.

To build our front end, the Dash framework was chosen. The Dash framework is lightweight, and allows users to build quick and elegant interfaces with minimal code. Using some basic HTML, we could build the desired layout that we wanted the dashboard to have, then link our interactive elements to callbacks that defined the desired behavior. Dash allowed us to view our MongoDB database in an organized manner, with both a datatable and infographics, with further interactive elements like our filtering options tapping into the control layer had built into our Python mid-tier.

The first part of our MVC structure that was built was the MongoDB back-end, importing the animal shelter file into a collection and then setting up user authentication to ensure users could only access what they had permission for. Next, we built CRUD modules using PyMongo to manipulate the data without having to actually directly interface with MongoDB, including creating, deleting, reading, and updating. Then over this mid-level, we placed our Dash front-end, which allowed us to view our data both with and without any manipulation from our Python controller. Functionality was implemented using the CRUD functions written earlier to bring the data into organized forms and filtered views.

The trickiest part of the setup was getting user authentication to work. There were numerous problems with the login process while the app was being built using Jupyter Notebooks. Ultimately, this was solved using the terminal the restart the local Jupyter server. Aside from that small hiccup, the CRUD functionality itself was fairly straightforward, and the wealth of documentation that Dash provided (especially Dash Recipes on Github) meant that any small problem was able to be solved quickly. Working with Dash has definitely provided that easiest front-end experience we've had so far.