README

**Required Functionality**

As a developer for Global Rain, I have been tasked with developing a full-stack software application for Grazioso Salvare that can use data from existing Austin animal shelters to identify and categorize available dogs that have the potential for search-and-rescue work. Grazioso Salvare has issued the following requirements for the application:

1. Branding Requirements
   1. The Grazioso Salvare logo must be shown somewhere on the application and link to their homepage ([www.snhu.edu](http://www.snhu.edu))
   2. There must be a unique identifier that gives credit to the developer
2. Dashboard Interactive Elements
   1. A data table that can be dynamically updated based on an applied filter
   2. Four filter options
      1. Water Rescue – Must have sex ‘Intact Female’, be anywhere from 26 weeks to 156 weeks old, and be one of the following breeds: Labrador Retriever Mix, Chesapeake Bay Retriever, Newfoundland
      2. Mountain or Wilderness Rescue - Must have sex ‘Intact Male’, be anywhere from 26 weeks to 156 weeks old, and be one of the following breeds: German Shepherd, Alaskan Malamute, Old English Sheepdog, Siberian Husky, Rottweiler
      3. Disaster Rescue or Individual Tracking - Must have sex ‘Intact Male’, be anywhere from 20 weeks to 300 weeks old, and be one of the following breeds: Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, Rottweiler
      4. Reset – Clear any previously selected filters
   3. A Geolocation chart that shows where the selected animal was rescued in Austin
   4. Another chart giving additional information about the current filtered selection

The following screenshots were taken from using the final, finished application and depict meeting all of the requirements given by Grasioso Salvare:

A screenshot of a computer

AI-generated content may be incorrect.Upon starting the Application

A screenshot of a computer

AI-generated content may be incorrect.After selecting the Water Rescue filter

A screenshot of a computer

AI-generated content may be incorrect.After selecting the Mountain or Wilderness filter

A screenshot of a computer

AI-generated content may be incorrect.After selecting the Disaster Rescue or Individual Tracking filter

A screenshot of a computer

AI-generated content may be incorrect.After selecting the Reset filter, follow another filter option.

**Tools and Rationale for Each**

**MongoDB –** Due to its flexible NoSQL nature, along with its ability to horizontally scale and organize vast quantities of data, I decided upon this as the right solution to model the data from the Austin animal shelters for Grazioso Salvare. In addition, integrating MongoDB into any Python project using the PyMongo API makes this database more accessible. It serves as the Model in the Model View Controller (MVC) architecture used for the application.

**PyMongo –** I specifically chose the PyMongo library to implement a Python module that allows me to create, read, update, and delete documents from MongoDB. It is the officially supported Python client library for MongoDB by default, and it provides all the necessary functionality out of the box to start querying the database from a Python project. PyMongo is the middleware for interacting between MongoDB and Dash, and is part of the controller in the MVC application architecture.

**Plotly Dash Python –** Dash is a low-code framework for developing data-focused web applications and uses Python as its primary programming language. Dash serves as part of the controller and the majority of the view in the MVC application architecture. It allows for simple function calls to generate the HTML that is ultimately rendered in a client's browser.

**Dash Leaflet –** Dash leaflet is an extremely niche library that is used to generate the geolocation chart that can be found in the bottom right corner of the dashboard.

**Jupyter Notebooks / Jupyter Dash –** Ultimately, the Dash application is written in a Jupyter Notebook environment and, as such, requires Jupyter Dash to be rendered as the output of a notebook cell. These tools were primarily implemented to make the time between each prototype iteration shorter.

**Pandas –** Pandas is another middleware layer between PyMongo and Dash. It simply converts the data retrieved by PyMongo from MongoDB into a form that Dash can use in its core web app components.

**Plotly Express** – Express is a library used for generating graphs from certain forms of data, such as a pandas dataframe. It was used to create the age distribution graph seen in the bottom left of the dashboard.

**From Prototype to Application**

1. The first task in developing Grazioso’s application was to aggregate and import all of the Austin Animal Shelter data into MongoDB and test simple queries on it to ensure it was imported successfully. This also involved setting up an account for interacting with the database that had limited permissions for security reasons.
2. After confirming all the data had been imported successfully and was in an accessible format, I developed the CRUD\_Python\_Module.py. This Python module contained the connection information for creating a client and connecting to MongoDB using the account set up in step 1. After the creation of the client, PyMongo could be used to interact with the database through CRUD (create, read, update, delete) operations. I specifically made a function for each operation that takes a MongoDB-style query and can be used after instantiating an instance of the AnimalShelter class contained in the module. By importing this Python module and instantiating an AnimalShelter object, you can call any of the AnimalShelter methods create(), read(), update(), or delete() to make changes to the Austin shelter data contained in MongoDB.
3. With the completion of this middleware Python CRUD module, I started development of the web application dashboard by using a Jupyter Notebook alongside Jupyter Dash to create a simple Plotly Dash Python application. This environment enabled me to add a Dash component to the cell that the Dash application was contained in and just re-run the cell to see an updated version of the application in my web browser. This made iterating on my original design seamless.
4. I added the logo as a clickable link, the dropdown with each filter option, an appropriate title, alongside my custom signature.
5. I then added a Dash Data Table that initially gets populated with all the data found in the Austin animal shelter databases. Once this was functioning properly, the addition of nice-to-haves such as pagination, ascending and descending filters, and even text filtering on each column was added. I also made it possible to select a single row, indicating a rescued animal for use with the geolocation map.
6. Here, I added the geolocation map using Dash Leaflet and made it update every time a new row is selected in the datatable.
7. Next, I created a callback that ensures the datatable is updated every time one of the filter options from the dropdown menu is selected, and created the relevant queries for each dropdown option according to the requirements.
8. Finally, I added a bar graph that displays the distribution of ages amongst all the rescue animals currently in the data table. It updates dynamically based on whether the data table has been filtered or not.

**Setbacks and Overcoming Challenges**

1. As I was developing the application, I found that styling each of the Dash components was a bit difficult. I am not sure if this is because some components are generated as nested HTML elements or some other side effect. An example of this was trying to align the filter dropdown, title, and my identifier as a flexbox with space-between each item. Even though I added what I believed to be the correct CSS styling, for some reason, my title was never centered correctly relative to the logo above it. I had to hack my way to a centered title by adjusting the amount of text in my identifier. It turned out okay, but this solution is definitely not recommended or portable.
2. I also had a lot of trouble trying to create the graph that I wanted in the bottom left corner. I must have tried three different ways to use the date and age data that were given in the Austin animal shelter dataset to generate this graph, but it always ended up wrong in some way. For instance, I tried to use the age\_upon\_outcome field, but it ended up messing up the x-axis labels, as there was no consistent formatting for this field. You could have 5 years as a value and 8 months as another, and they could not be sorted easily because of this. The solution I ended up creating was a temporary column called ‘age in years’ that was the age\_outcome\_in\_weeks divided by 52, and I used this in a histogram to generate the graph.