**README for CS-340 Dashboard for Grazioso Salvare**

**Project Description**

This project involves developing a dashboard for Grazioso Salvare using the Dash framework. The dashboard visualizes data from the Austin Animal Center Outcomes dataset and includes the following functionalities:

1. Interactive options to filter the dataset.
2. A data table that dynamically responds to the filtering options.
3. A geolocation chart and a second chart (a pie chart) that dynamically respond to the filtering options.
4. The Grazioso Salvare logo and a unique identifier containing my name.

**Required Functionality**

The dashboard includes the following functionalities:

* Interactive filtering options via a dropdown menu.
* A data table that updates based on the selected filter.
* A geolocation chart displaying the locations of the animals.
* A pie chart showing the distribution of animal breeds.
* The Grazioso Salvare logo and my unique identifier.

**Screenshots**

* **Starting State of the Dashboard**:

A screenshot of a map

Description automatically generated

* **Water Rescue Filter Applied**:

A screenshot of a map

Description automatically generated

* **Mountain or Wilderness Rescue Filter Applied**:

A screenshot of a map

Description automatically generated

* **Disaster or Individual Tracking Filter Applied**:

A screenshot of a map

Description automatically generated

* **Reset Filter Applied**:

A screenshot of a map

Description automatically generated

**Tools Used**

* **Dash Framework**: Used for building the web application with interactive components. Dash provides a flexible and powerful way to create interactive web applications with Python.
* **Plotly Express**: Used for creating the pie chart. Plotly Express simplifies the creation of visualizations and integrates well with Dash.
* **Dash Leaflet**: Used for the geolocation chart. Dash Leaflet is an extension of Dash that allows for easy integration of interactive maps.
* **Pandas**: Used for data manipulation. Pandas provides powerful data structures to manipulate numerical tables and time series data.
* **MongoDB**: Used as the database to store and retrieve data. MongoDB's flexibility in handling unstructured data and its powerful querying capabilities make it an excellent choice for this project.

**Rationale for Tools**

* **MongoDB**: MongoDB was chosen for its flexibility in handling diverse data types and its efficient querying capabilities. It allows for seamless integration with Python through the use of libraries like pymongo, making data retrieval and manipulation straightforward.
* **Dash**: Dash was selected because it provides a robust framework for creating interactive web applications using Python. Its component-based architecture makes it easy to build and maintain complex dashboards.

**Explanation of Dash Framework**

The Dash framework provides a structure for building web applications using Python. It consists of three main parts:

1. **Dash Components**: Predefined UI elements (e.g., graphs, tables, dropdowns) that are used to build the layout of the application.
2. **Dash Callbacks**: Functions that are automatically called to update the UI based on user input or other triggers.
3. **Dash Server**: The underlying Flask server that runs the web application.

**Resources and Software**

* Dash Documentation
* Plotly Express Documentation
* Dash Leaflet Documentation
* Pandas Documentation
* [MongoDB Documentation](https://docs.mongodb.com/)

**Steps Taken to Complete the Project**

1. **Setup MongoDB and Dash Environment**: Installed necessary libraries and set up MongoDB for data storage.
2. **Data Retrieval and Manipulation**: Used the AnimalShelter class to interact with MongoDB and retrieve the dataset.
3. **Dashboard Layout Design**: Created the layout of the dashboard using Dash components, including dropdowns, data tables, and charts.
4. **Interactive Callbacks**: Developed callbacks to update the data table and charts based on user input.
5. **Testing and Deployment**: Tested the dashboard to ensure all components work as expected and deployed the application.

**Challenges Encountered and Solutions**

* **Handling Large Datasets**: The initial data retrieval process was slow due to the large size of the dataset. This was resolved by implementing more efficient MongoDB queries and filtering the data at the database level.
* **Dynamic Updates**: Ensuring that all components dynamically update based on user input was challenging. This was addressed by carefully designing the callback functions and testing them thoroughly.
* **UI Layout Issues**: Adjusting the layout to make the legend area longer without removing the scroll section required fine-tuning the update\_layout properties in Plotly.

**CRUD Method Documentation**

The CRUD (Create, Read, Update, Delete) methods were used to interact with MongoDB, providing an interface to perform various operations on the data.

**Create**

The create method is used to insert new documents into the MongoDB collection.

python

Copy code

def create(self, data):

if data is not None:

self.database.animals.insert(data)

else:

raise Exception("Nothing to save, because data parameter is empty")

**Read**

The read method is used to retrieve documents from the MongoDB collection. It supports various query parameters to filter the data.

python

Copy code

def read(self, query):

return list(self.database.animals.find(query, {"\_id": False}))

**Update**

The update method is used to update existing documents in the MongoDB collection based on a specified query.

python

Copy code

def update(self, query, data):

if query is not None:

self.database.animals.update\_many(query, {"$set": data})

else:

raise Exception("Nothing to update, because query parameter is empty")

**Delete**

The delete method is used to delete documents from the MongoDB collection based on a specified query.

python

Copy code

def delete(self, query):

if query is not None:

self.database.animals.delete\_many(query)

else:

raise Exception("Nothing to delete, because query parameter is empty")

**Conclusion**

The project successfully met all the requirements and provided a robust, interactive dashboard for Grazioso Salvare. The use of Dash and MongoDB proved to be effective in building a dynamic web application that is easy to maintain and extend. The comprehensive testing and deployment steps ensured the application is reliable and user-friendly.