**Grazioso Salvare Dashboard README**

David Smith

Department of Computer Science, Southern New Hampshire University

CS-340: Client/Server Development

Professor J. Sanford

October 17, 2021

**About the Project/Project Title**

Grazioso Salvare Dashboard is a Dash web dashboard built on a PyMongo CRUD interface called MongoCRUD and MongoDB. This dashboard allows for ease of use for information about animals in the selected database.

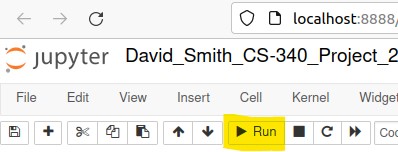
**Motivation**

Grazioso Salvare was interested in an easy to use interface that allows them to search for dogs from animal shelters that would be good matches to the company’s requirements for different types of rescue situations. Dash was chosen for its ease of use, and MongoCRUD was developed specifically for this project, but has its own documentation.

MongoDB was chosen as it is a NoSQL database. It also has a driver available in PyMongo that allows for an easy to use interface with Python. The Dash framework is an intuitive interface to make visually pleasing dashboards with minimal effort.

**Getting Started**

To use the Grazioso Salvare Dashboard, load the ipynb file into Jupyter Notebook and execute. A link at the bottom of the running cell will open the dashboard in a new window. This is strongly encouraged, as Jupyter Notebook limits the available space for display.



**Installation**

Ensure that the environment is set up as explained in the MongoCRUD README.

The Grazioso Salvare Dashboard requires the following Python modules:

Text

Description automatically generated

MongoDB can be obtained from mongodb.com. Python3 can be found through your operating system’s application store or package repository. PyMongo, dash, JupyterDash, plotly express, pandas, numpy, matplotlib, and base64 can be installed through the pip package installer. MongoCRUD can be obtained from my MongoCRUD repository.

The login information for the MongoDB database will need to be changed to match the local configuration when instantiating the connection to MongoCRUD.

Graphical user interface, text, application

Description automatically generated

The database information is then loaded into a pandas dataframe:



The columns of the dataframe can be rearranged to move more pertinent information to the left side:

Text

Description automatically generated

**Usage**

Using the environment starts by defining the app and creating a layout with IDs for the Dash dashboard. This is a group of nested html properties that define how objects will be displayed on the dashboard. The types of properties used in this dashboard include:

* html.Div – Used to separate sections of the dashboard.
* html.A – Used to create an anchor link, also known as a hyperlink.
* dcc.Graph – Used to create the pie chart.
* html.Button – Used to create a button that can be used to change states.

Create a style for changing row highlighting of a table based on currently clicked cell. This is done with a style\_data\_conditional variable. Also, we loaded our image for the hyperlink as a base64 encoded image.

Next, create a DataTable. Load the dataframe into the DataTable. The important part of this step is (df is our dataframe):

For this particular project, page size was limited to twenty entries.

Callbacks must then be created to handle the interaction between the different objects on our dashboard. We defined three separate callbacks.

Update\_selected\_row\_color takes the DataTable’s active\_cell as input and outputs to the DataTable’s style\_data\_conditional. This allows modification of the style to change the color of the row of the currently active cell.

Update\_display takes the datatable’s derived\_viewport\_data, active\_cell, and children properties and returns the map-id’s children and pie-chart-id’s figure properties. It also has access to the DataTable’s state property. The function then copies the data for the current page in the DataTable into a dataframe. If the dataframe is empty, we return the state of the DataTable to avoid any changes to the dashboard objects.

If there is no active cell chosen, the active row is set to 0. This is for initializing the datatable and also used when filtering to avoid index out of bounds errors. If an active cell is chosen, its row is used as the active row.

The next step is to use the data in the dataframe to determine the latitude and longitude of the currently selected dog. This location is then copied into a string to format for display. The pie chart is created by using the data from the dataframe, which only has the currently selected page’s data in it. The title for the pie chart is centered by using the command: 

The data for the pie chart is stored in a variable to be returned at the end of the callback function.

Next, the map is created from a Dash Leaflet. The properties for this object include a marker, which is set to the location already retrieved from the dataframe for the current animal. The animal\_id field is used for the mouseover tooltip. The name of the dog, or Unnamed if that value is empty, is used for the first piece of information on the marker. This is followed by the breed and then the geolocation coordinates. When the active\_cell changes, the marker updates to the newly selected dog and the map recenters on the marker.

Finally, the update\_display callback ends by returning the updated map information and the pie chart.

The last callback, on\_click, takes four buttons’ n\_clicks properties as input, as well as the current state property of the DataTable. The n\_clicks for the buttons ensures that this callback is run any time a button is clicked. The outputs are all to the DataTable and include the active\_cell, the selected\_cells, and the data.

This callback allows for filtering of the data in the DataTable to match preferred dog breeds, genders, and ages for specific types of rescue. The function starts by defaulting to all dogs in the database and returns that data accordingly, along with setting the active cell to the top left corner of the Datatable.

Next, the function checks to see if each of the buttons was clicked, setting a variable based on which one evaluates to true. A query is then run on the MongoDB database through the MongoCRUD module to return the data matching the currently selected filter. Returning this data updates the DataTable, which triggers the callback to update the map and pie chart.

To determine which button was clicked last, the dash.callback\_context.triggered property was used:



and then compared with the button IDs:



**Code Examples**

Defining the app:



Creating a default style:

Text

Description automatically generated with medium confidence

Loading a base64 image to embed in the html:



Defining the layout for the dashboard:Text

Description automatically generated

Callback to highlight selected row in DataTable:

Text

Description automatically generated

Callback to update map and pie chart based on current active\_cell in DataTable:

Text

Description automatically generated

Filtering data in DataTable based on most recent button clicked:

Text

Description automatically generated with low confidence

**Testing and Proof**

The following are a few screenshots of the finished product. A video is also available at <https://youtu.be/GU0zfVHd1b4> and included with this submission which demonstrates the interface and its use. Graphical user interface, application

Description automatically generated

A picture containing table

Description automatically generated

The last few screenshots cut off the right side of the dashboard but show the changes to the DataTable and pie chart based on selections in the DataTable or clicking one of the filtering buttons. These changes are explained better in the description video.

A picture containing graphical user interface

Description automatically generated

A picture containing chart

Description automatically generated

A picture containing graphical user interface

Description automatically generated

**Roadmap / Features**

Currently, the next step is to evaluate the ability to use themes to stylize the dashboard. I have not yet been able to get this functionality to work from a Jupyter Notebook with JupyterDash.

Another modification that needs to be made is determining how to keep the graph from displaying before the pie chart has been populated with data. This is minor, but it is an annoyance.

## Credits

The data used in the examples is from the Austin Animal Center Outcomes dataset, which is available at <https://doi.org/10.26000/025.000001>

**Contact**

David Smith