# CS-340 Project Two

## Grazioso Salvare Dashboard

This project provides a dashboard with interactive widgets for the company Grazioso Salvare. Using the interactive widgets allows us to filter results in a datatable along with a geolocation map and pie chart that update in real time to help visualize the data.

## Motivation

Grazioso Salvare is a company that identifies dogs that will be suitable for search-and-rescue operations. Different characteristics of dogs (breed, sex, age, etc.) are suitable to different types of operations, so we have provided a way to quickly filter the results of available dogs depending on those criteria and show the results to the user in a clean, informative dashboard format with interactive widgets.

## Getting Started

To get started setting up this project locally, you will need the Python CRUD file, the Jupyter Notebook file. These will be downloadable from GitHub and can be run in Jupyter inside of a web browser. Make sure both files are in the same directory, then run the Jupyter Notebook file within the Jupyter browser application, and this will produce the dashboard locally within the browser that can then be interacted with.

## Installation

For this project, I used Python to create the CRUD file that access the database, and Jupyter Notebook to run the code and create the actual dashboard. Within those files, however, are several dependencies that needed to be imported. I used PyMongo within Python to interface with a MongoDB database, and I used Dash within the Jupyter file to create the dashboard.

MongoDB is an excellent choice for creating our database for the model component of our project for several reasons. MongoDB offers flexibility, a flexible query model, native aggregation, and a schema-less model. (Giamas, 2022) The feature we are likely most concerned with is its flexible query model. This allows us to perform queries for specific combinations of desired characteristics using Python.

Dash was used for the view and controller components because it is a powerful framework used frequently for data visualization. Using Dash Core Components, we have several interactive data visualization components we can “plug and play” into our code to convey information in an efficient and visually appealing way.

Within the Python CRUD file, I imported the following:

from pymongo import MongoClient

from pymongo.errors import PyMongoError

from bson.objectid import ObjectId

from typing import Dict, Any

Within the Jupyter dashboard file, I imported the following:

from jupyter\_dash import JupyterDash

import dash\_leaflet as dl

from dash import dcc

from dash import html

import plotly.express as px

from dash import dash\_table

from dash.dependencies import Input, Output, State

import base64

import os

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from AnimalShelterCRUD import AnimalShelter

This may sound like a lot to worry about, but they are mostly the different components that Dash uses to represent all the data in a clean way, and these lines are all included in the files already to be imported.

## Usage

### Code Example

The AnimalShelterCRUD Python file is a basic CRUD application, allowing us to Create, Read, Update, and Delete records as needed. A quick example of the Read method is below, as that will be the method we will be using the most for the purposes of this dashboard.

def read(self, query:Dict[Any, Any]):

"""

Method to find a document in the database that accepts a dictionary as an argument.

Returns the document as a list if successful, otherwise returns an empty list.

"""

if isinstance(query, Dict) and query is not None:

try:

return list(self.database.animals.find(query))

except PyMongoError as error:

print('Find error: ', error)

return []

else:

raise Exception('Error: Query must be a dictionary.')

Within the Jupyter Notebook file, there are several methods included that exemplify the Model View Controller (MVC) design pattern. This acts as the Controller that links the Model (Python file) to the View (user-interactive dashboard). Below is the method that shows the different queries that filter the results based on the dog characteristics we are looking for.

def update\_dashboard(filter\_type):

if filter\_type == 'Water Rescue ':

df = pd.DataFrame.from\_records(db.read({'animal\_type':'Dog',

'breed':{'$in':

['Labrador Retriever','Chesapeake Bay Retriever','Newfoundland']},

'sex\_upon\_outcome':'Intact Female',

'age\_upon\_outcome\_in\_weeks':{'$gte':26},

'age\_upon\_outcome\_in\_weeks':{'$lte':156}}))

elif filter\_type == 'Mountain and Wilderness Rescue ':

df = pd.DataFrame.from\_records(db.read({'animal\_type':'Dog',

'breed':{'$in':

['German Shepherd','Alaskan Malamute','Old English Sheepdog','Siberian Husky','Rottweiler']},

'sex\_upon\_outcome':'Intact Male',

'age\_upon\_outcome\_in\_weeks':{'$gte':26},

'age\_upon\_outcome\_in\_weeks':{'$lte':156}}))

elif filter\_type == 'Disaster Rescue or Individual Tracking ':

df = pd.DataFrame.from\_records(db.read({'animal\_type':'Dog',

'breed':{'$in':

['Doberman Pinscher','German Shepherd','Golden Retriever','Bloodhound','Rottweiler']},

'sex\_upon\_outcome':'Intact Male',

'age\_upon\_outcome\_in\_weeks':{'$gte':20},

'age\_upon\_outcome\_in\_weeks':{'$lte':300}}))

elif filter\_type == 'None (Reset Filters) ':

df = pd.DataFrame.from\_records(db.read({}))

df.drop(columns=['\_id'],inplace=True)

return df.to\_dict('records')

### Tests

Using Jupyter Notebook, you can run some simple tests to make sure the CRUD module is working. These are simple lines of code that prove each of the CRUD methods are functioning as intended.

from ProjectOne import AnimalShelter

shelter = AnimalShelter('aacuser', 'aacuserpwd')

dog1 = shelter.create({'name':'SampleDogName'})

print(dog1)

This returns True, showing that the dog object was created after the class was instantiated.

shelter.read({'name':'SampleDogName'})

This returns the dog object that we created.

shelter.update({'name':'SampleDogName'},{'$set': {'name':'NewSampleDogName'}})

This returns 1, the number of records updated.

shelter.read({'name':'SampleDogName'})

This returns an empty list, showing the original name does not exist in the database any longer, since it was updated.

shelter.read({'name':'NewSampleDogName'})

This returns the updated dog object with the new name.

shelter.delete({'name':'NewSampleDogName'})

This returns 1, the number of records deleted.

Shelter.read({‘name’:’NewSampleDogName’})

This returns an empty list, proving the record has been deleted.

### Screenshots

### This screenshot shows the starting state of the dashboard. This is an unfiltered view of all animals returned in the default database query.

A screenshot of a computer

AI-generated content may be incorrect.

This screenshot shows the updated widgets after the “Water Rescue” filter radio button has been selected.

A screenshot of a computer

AI-generated content may be incorrect.

This screenshot shows the updated widgets after the “Mountain and Wilderness Rescue” filter radio button has been selected.

A screenshot of a computer

AI-generated content may be incorrect.

This screenshot shows the updated widgets after the “Disaster Rescue or Individual Tracking” filter radio button has been selected.

A screenshot of a computer

AI-generated content may be incorrect.

This shows the dashboard return to its default unfiltered state after the “None (Reset Filters)” radio button has been selected. This is also the default option in the radio buttons and thus is the default state the dashboard loads into.

A screenshot of a computer

AI-generated content may be incorrect.

## Contact

Robert DiMaio

**Reference**

ALEX GIAMAS. **Mastering MongoDB 6.x : Expert Techniques to Run High-volume and Fault-tolerant Database Solutions Using MongoDB 6.x**. Birmingham, UK: Packt Publishing, 2022. Disponível em: https://research.ebsco.com/linkprocessor/plink?id=a5bcc20e-3306-36b5-ad4f-0d0bd1f1567e. Acesso em: 16 ago. 2025.