Grazioso Salvare Animal Shelter Dashboard

# Project Overview

The Grazioso Salvare Animal Shelter Dashboard is a web application designed to manage and visualize data from an animal shelter database. The dashboard provides functionalities such as data filtering, visualization, and interaction with the data in various forms like tables, graphs, and maps.

# Purpose

# The Grazioso Salvare Animal Shelter Dashboard was developed with the goal of providing a comprehensive and user-friendly interface for managing and visualizing animal shelter data. The motivation behind this project stems from the need to streamline the process of interacting with animal records, enhancing the efficiency of shelter operations, and facilitating data-driven decisions. The result is a dynamic and modular system that can be tailored to fit a wide array of use cases, ultimately contributing to the well-being of animals by improving the operational capabilities of shelters and rescue organizations.

# **Functionality**

Data Interaction: Users can interact with the animal data through sorting, selecting, and filtering mechanisms.

Visualization: The dashboard visualizes data distributions and geographical information using graphs and maps.

Responsive Design: The application is responsive and can adapt to different screen sizes for optimal viewing.

A screenshot of a computer

Description automatically generated

# Tools and Technologies

MongoDB:

MongoDB is used as the model component of the development. It provides a flexible schema and efficient storage for JSON-like documents, making it ideal for handling diverse data types found in animal shelter records. Its dynamic query language allows for powerful data retrieval and is well-suited for Python’s data manipulation capabilities.

Python and PyMongo:

Python serves as the backbone of the project, offering robust libraries for data analysis and web development. PyMongo is utilized as the MongoDB driver for Python, enabling seamless communication between the Python application and the MongoDB database.

Dash Framework:

The Dash framework provides the view and controller structure for the web application. It allows for the creation of interactive, web-based data visualizations that are rendered in the browser. Dash is built on top of Flask, Plotly, and React.js, offering a comprehensive solution for building analytical web applications.

Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. In this project, Jupyter Notebook serves as the development environment where the Dash application is written and tested. Its interactive nature allows for real-time code execution, making it ideal for prototyping and iterative development. It’s particularly useful for this dashboard application because it supports the JupyterDash library, which enables the integration of Dash apps within Jupyter environments.

AnimalShelter Module:

The AnimalShelter module is a cornerstone of this project, facilitating seamless CRUD operations with MongoDB. It encapsulates a suite of methods that empower applications to create, read, update, and delete documents within a MongoDB collection with ease and precision. Designed to be highly modular, the AnimalShelter class can be effortlessly integrated into other applications, providing a robust and reusable interface for database interactions. This design choice underscores our commitment to modularity and reusability, ensuring that our solution can serve as a foundational building block for a myriad of applications.

For more details on the AnimalShelter module, please refer to the [AnimalShelter README](AnimalShelter_README.docx).

# Installation and Setup

Prerequisites:

Before you begin, ensure you have Python and pip installed on your system. Python is a powerful programming language that lets you work quickly and integrate systems more effectively. If you don’t have Python installed, you can download it from the official [Python website](https://www.python.org/downloads/). Pip is a package manager that will help you install other Python packages. If you haven’t already installed pip, you can do so by following the instructions on the [pip installation page](https://pip.pypa.io/en/stable/installation/).

## Step-by-Step Installation:

1. Open your command prompt (CMD) or terminal.
2. To install Jupyter Notebook, which is essential for running the JupyterDash app, use the following command:

pip install notebook

After installation, you can launch Jupyter Notebook by running jupyter notebook in your terminal or command prompt, which will open the application in your web browser. More information on Jupyter Notebook can be found on the [Project Jupyter website](https://jupyter.org/).

1. This project requires specific Python packages, which can be installed using the following command:

pip install dash pandas plotly dash-leaflet jupyter-dash

This command will install:

* dash: A Python framework for building reactive web apps.
* pandas: A library providing high-performance, easy-to-use data structures, and data analysis tools.
* plotly: A graphing library that makes interactive, publication-quality graphs online.
* dash-leaflet: A Dash component library for creating interactive and customizable maps.
* jupyter-dash: An extension for running Dash apps in Jupyter environments.

Connecting to your MongoDB:

To connect to a MongoDB instance, the user credentials, host, port, database, and collection must be provided in the \_\_init\_\_ method within the AnimalShelter class. Example:

A computer screen with colorful text

Description automatically generated

Run the application:

Navigate to the project directory. The application can be started by entering this command into a CMD or Terminal:

python app.py

This will start the Dash server, the web application can be accessed through your browser with the following URL:

[http://127.0.0.1:<PORT>/](http://127.0.0.1:%3cPORT%3e/)

Replace <PORT> with the port number your application is using.

# Usage

Base Dashboard:

Upon entering the webpage via the URL, you will be greeted with the following page. This page is the base, unfiltered, dataset containing all the information in the database collection. Clicking the “Reset Dashboard” button, will return the dashboard to this unfiltered state.

A screenshot of a computer

Description automatically generated

Filtering and Sorting:

The application offers various methods for users to filter and sort information. Each column can be sorted via ascending/descending order. The 3 filter dropdown buttons allow for selective filtering. The first filter button (on the left side of the page) contains the rescue methods. These are hardcoded values which can be specified within the project’s .ipynb file. The second filter button is dynamic (not hard coded) it will update based on the provided keys / headers for the columns. Example:

A screenshot of a map

Description automatically generated

The Third filter button is an optional filter, which is also dynamic, it will populate options based on the other two selected filter options. In this instance, after selecting breed. I can specify to which breed I would like to see (Siamese Mix) on the table. Example:

A screenshot of a computer screen

Description automatically generated

Graph and Map:

The graph is a histogram that will show the distribution of the selected key (column) filter. Selecting a row will update the map to match the selected row. Both are dynamic and update alongside filtered options.  
A screenshot of a map

Description automatically generated

Additional Information:  
The data on the page is not asynchronous or updated upon callbacks. rather, the instance for the data is loaded upon start / refresh of the application. In this example it reduces computation / update time of the graphs / map and table by storing and passing the (pre-loaded) data. This method can increase memory usage. However, this can be easily adjusted within the update\_dashboard function in the .ipynb file by changing the variable: “df” to “fetch\_data()”.

Example:

…rest of code

def update\_dashboard(parameters)

dff = df

Change to…

dff = fetch\_data()

…rest of code