# PROJECT TWO README

## About the Project/Project Title

This project is the second and final phase of a bigger project that includes creating a full stack development application involving a database and a client-facing web application dashboard. For the first phase of the project, the focus was on developing a Python module to interact with a MongoDB database that enables the user to insert new data, read and update existing data, and delete data in a specific database with provided authentication. In this second phase of the project, the focus was on developing a client-side application dashboard that interfaces with the MongoDB database, while using Python code as the middle-ware application.

## Motivation

The project is for an international rescue-animal training company, Grazioso Salvare, that needs a software application that can work with existing data from the animal shelters to identify and categorize unique dog breeds for various rescue operations. The first phase focused on creating a CRUD module to query and interact with data in MongoDB with Python language. The second phase focused on building a web application dashboard that uses the Python language to access those queries and create data tables, pie charts, and a geolocation chart, along with other filtering options.

## Getting Started

Note: To see how to initially load the CSV file and data into the MongoDB database, as well as how to create the CRUD module in python for initialization and authentication, please refer to the “Project One README” file.

## Required Functionality

* Dashboard Branding - To meet Grazioso Salvare’s branding requirements, the company has requested the inclusion of the following components somewhere on the dashboard:
  + The Grazioso Salvare logo. The company has requested that this logo include a URL anchor tag to the client's home page: www.snhu.edu.
  + A unique identifier (text or image) containing your name. Grazioso Salvare would like to credit you as the creator of the dashboard.
  + Graphical user interface, text, application

    Description automatically generated
  + The logo is on the top left, the unique identifier, “Dashboard by Andreas Galatis” is in the top center, and the URL anchor tag can be seen on the bottom left corner, which is connected to the logo image and appears when the mouse hovers over it.
* Required Dashboard Widgets - Grazioso Salvare is requiring the following widgets for the dashboard interface as displayed below:

Graphical user interface

Description automatically generated with medium confidence

* + In the above screenshot you will find:
    - Interactive filter options (buttons) to filter the Austin Animal Center Outcomes data set by: Water Rescue, Mountain Rescue, Disaster Rescue, and Reset (return all widgets to their original, unfiltered state}
    - A data table which dynamically responds to the filtering options
    - A geolocation chart and a second pie chart that dynamically respond to the filtering options

**Testing**

* Below are screenshots of the test results when you click each filtering options
  + - Water Rescue

Graphical user interface, application, table

Description automatically generated

* + - Mountain Rescue

Graphical user interface, application

Description automatically generated

* + - Disaster Rescue

Graphical user interface, application

Description automatically generated

* + - Reset (return all widgets to their original, unfiltered state

Graphical user interface, application, table

Description automatically generated

## Tools

There were three main components used for the completion and successful operation of this application, which is based on a Model-View-Controller (MVC) architectural pattern. The Model component is where all the data and data related logic resides. The View component is the user interface logic of the application, or the client-side view of the app. And the Controller component acts an intermediary between the Model and the View components to process all the incoming requests and render data manipulation by interacting with both the Model and the View components.

* [MongoDB](https://www.mongodb.com/) was used as the Model component, as it has a very flexible schema and compatibility with modern programming languages such as Python, as well as having an efficient data storage and retrieval process, with its document data model. Some of its other benefits include an easy to start process for developers, a fully scalable architecture that can support huge volumes of data, and a large community of developers that can support each other.
* For the Controller component, the [Python](https://www.python.org/) language was used, as it offers an easy scripting language that can be implemented rapidly and efficiently, while also providing the ability to create object-oriented code to produce modules and classes that can be reused for several operations without having to rewrite redundant code. Additionally, Python is a great middleware application that can interface with the backend MongoDB database while also providing interaction with the client-side dashboard and web interface.
* Lastly, for the View component, the use of [DASH](https://plotly.com/dash/), which is an opensource framework that gives developers the ability to easily create data apps with a point-and-click interface model, makes navigating a database much faster and user-friendly. DASH uses Python to build web applications with a framework that provides several interactive callback functions that are user friendly, such as dropdown menus, radio items, graphs and charts, and several other input and display options.

### Code Example

* To create the DASH web interface for this application, begin by importing all the necessary dependencies into a Python script, such as all the DASH components to enable HTML compatibility and enable the use of graphs and callback functions.

Text

Description automatically generated

* Then connect to the Model component, which is MongoDB, by importing the CRUD module from project one, and instantiating an object using the proper authentication, and using that object to load the data.

Graphical user interface, text, application, email

Description automatically generated

* The dashboard and layout view have several parts that display the required functions. Begin with the “app” command and then add the image logo as well as a personal identifier.

Text

Description automatically generated

* The next segment of code details the interactive filtering options by way of clickable buttons and data filtering option on the data table.

Text

Description automatically generated

* The following HTML commands enable the pie chart and the geolocation chart to be rendered next to each other

Graphical user interface, text, application

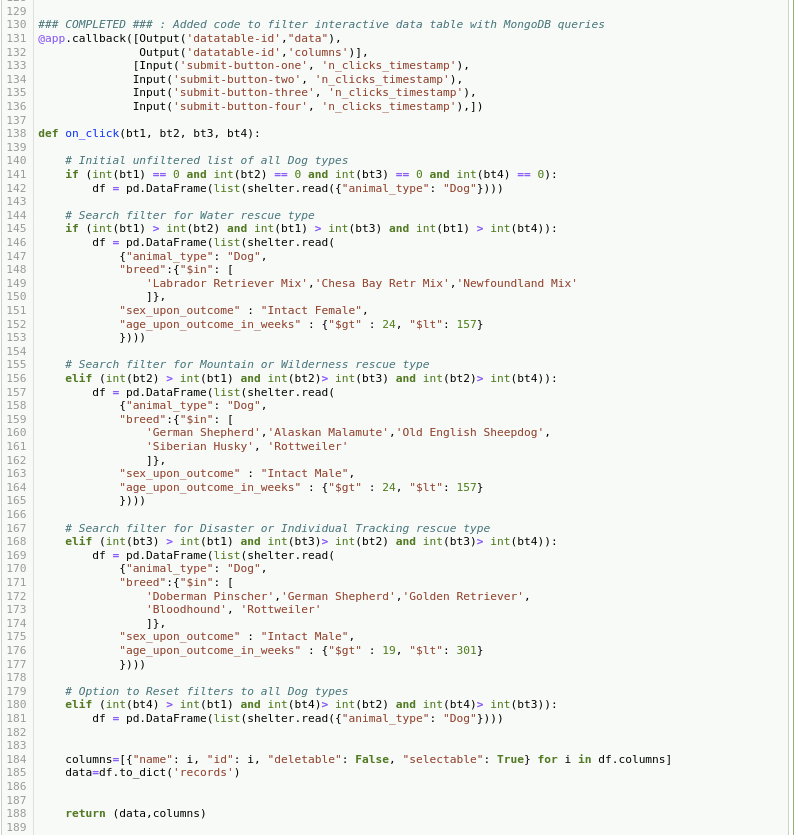
Description automatically generated

* The remaining segment of code involves several callback functions as well as methods that enable the data table and charts to be interactive with the use of MongoDB queries that are called using the CRUD module.
  + The following callback function enables the user to highlight a row on the data table when it is clicked on.

Text

Description automatically generated

* + The most important callback function and querying method occurs next, allowing the user to click one of the filtering buttons and have the data table and charts display the appropriate results. For the inputs on this callback function, “n\_clicks\_timestamp” was used. This function records the last time a button is clicked and when used with some “if” and “elif” conditionals, it can be manipulated to switch between button clicks. Inside each conditional, a CRUD function was called with the required querying parameters.



* Lastly, two more callback functions were called that also take in as input the results from the queries and output a pie chart and a geolocation chart respectively. And then end with another “app” command.

Text

Description automatically generated

**Challenges**

Some of the challenges faced in writing this code were mostly in determining the proper syntax for the MongoDB querying commands, as well as the input and output callback functions. Determining the proper code to get the pie chart was also a bit of a struggle, since at first none of the data was populating, and only an empty graph was displayed. Most of the of the challenges were remediated by looking up the related documentation that show how to write the various commands, and they can found on the MongoDB website resource page (<https://www.mongodb.com> ), as well as the Dash website resource page (<https://dash.plotly.com>).

## Contact

Andreas Galatis