**— Project Two —**

**Grazioso Salvare Search & Rescue Candidate Locator Dashboard**

Ryan Rieth

Prof. Kellogg — CS 340 — SNHU

Table of Contents

[**Search and Rescue Candidate Locator for Grazioso Salvare** 3](#_Toc127562314)

[**Tools** 3](#_Toc127562315)

[**MongoDB** 3](#_Toc127562316)

[**PyMongo** 3](#_Toc127562317)

[**Jupyter Notebook** 3](#_Toc127562318)

[**Python IDE** 3](#_Toc127562319)

[**Installation** 4](#_Toc127562320)

[**Running the Dashboard** 4](#_Toc127562321)

[**Usage** 4](#_Toc127562322)

[**Screenshots of Dashboard Results** 5](#_Toc127562323)

[Initial Starting State: 5](#_Toc127562324)

[Filtering via the “Water Rescue” Radio Item: 6](#_Toc127562325)

[Filtering via the “Mountain/Wilderness Rescue” Radio Item: 7](#_Toc127562326)

[Filtering via the “Disaster/Individual Tracking” Radio Item: 8](#_Toc127562327)

[Filtering via the “Reset” Radio Item (Unfiltering): 9](#_Toc127562328)

[**Features** 10](#_Toc127562329)

[**Contact** 10](#_Toc127562330)

## **Search and Rescue Candidate Locator for Grazioso Salvare**

This is dashboard created in *Jupyter Notebook* that utilizes a *Python* module to interface with *MongoDB*. It reads information regarding a collection of animals for the purposes of finding desirable candidates for search and rescue operations in different climate biomes.

The *Python* module it utilizes — *CRUD.py* — implements *Python* script within the *MongoDB* shell.

In **Version 1.0**, it exists with the purpose to *Create* and *Read* data into a collection about animals.

In **Version 2.0**, it exists with the purpose to *Create*, *Read*, *Update*, and *Delete* data within a collection about animals.

In **Version 3.0**, it exists with the same purpose as its previous version but with added structural integrity to the code as well as syntax and logic corrections scattered throughout.

In totality, this project is a roadmap for deeper, more complex implementations of the concepts at play. Scripted modules can be utilized instead of manual, hard-coded *MongoDB* implementation, allowing a modularity not only among software but, in the grand scheme, among projects, companies, and entities.

The benefits? Efficiency, Understanding, and — hopefully — subsequent Enjoyment.

## **Tools**

You will need the *MongoDB* *shell*, the *PyMongo driver* for MongoDB, access to *Jupyter Notebook*, and a *Python* IDE installed on your local device. Please see the “Installation” section for further details.

### **MongoDB**

How else will you run *Python* script within *MongoDB* without *MongoDB* itself?

Click [here](https://www.mongodb.com/) for *MongoDB*.

### **PyMongo**

*PyMongo* allows communication between *Python* and *MongoDB*. In other words, it’s vital\*.

Click [here](https://pypi.org/project/pymongo/) for *PyMongo*.

*\*Note: there are other drivers out there, but* PyMongo *is* MongoDB’s *recommended and maps directly to* MongoDB’s *query language.*

### **Jupyter Notebook**

*Jupyter Notebook* is the driver for the dashboard. Make sure it is installed and set up properly before attempting any step of this project.

Click [here](https://jupyter.org/) for *Jupyter Notebook*.

### **Python IDE**

It’d be easier to personalize the *Python* source code for this *Python* module if you have a *Python* IDE available. This is subjective, but my two personal choices are *Spyder* and *PyCharm*.

Click [here](https://www.spyder-ide.org/) for *Spyder* or click [here](https://www.jetbrains.com/pycharm/download/#section=mac) for *PyCharm*.

## **Installation**

Start up the terminal and connect to the *MongoDB* server within the shell (connection itself does *not* have to be authorized).

Import the default CSV file included to populate data or create your own under the “AAC” database and “animals” collection within *MongoDB*.

Fire up the *Jupyter Notebook*. **Make sure the *CRUD.py* file and the *Jupyter Notebook* dashboard file are stored in the same place**, alongside the Grazioso Salvare Logo PNG file.

## **Running the Dashboard**

The project is as simple as starting up the Jupyter Notebook and running its main kernel. The dashboard will offer access to and interaction with the *MongoDB* data from within.



## **Usage**

The dashboard contains a few filtering buttons, a data table to showcase the collection information, a pie chart to portray the information visually, and a map pinpointing the location of either the first animal in the list or a specified animal in the list.

Dynamic type-to-text boxes under each column in the data table allow customized filtering.

*\*Note that none of the rows, columns, or data in general can either be updated or deleted although the project utilizes these two aspects of the CRUD formula — this was and is for testing and developmental purposes that coincide with ease of use for Grazioso Salvare’s workers.*

## **Screenshots of Dashboard Results**

### Initial Starting State:

Graphical user interface, application

Description automatically generated

### Filtering via the “Water Rescue” Radio Item:

Graphical user interface, application

Description automatically generated with medium confidence

### Filtering via the “Mountain/Wilderness Rescue” Radio Item:

Graphical user interface, application

Description automatically generated

### Filtering via the “Disaster/Individual Tracking” Radio Item:

Application

Description automatically generated

### Filtering via the “Reset” Radio Item (Unfiltering):

Graphical user interface, application

Description automatically generated

## **Features**

*CRUD.py* stands out as a baseline for CRUD development in its basic forms.

It introduces the concept of intercommunication between a Terminal shell such as *MongoDB* and a powerful programming language such as *Python* without demolishing the boundaries of understanding.

In other words, it’s a simple education on how *Python* can be run within *MongoDB* and how *Python* can supplement a Terminal shell’s execution.

Version 1.0

The *CRUD* facets of *Create* and *Read* were implemented first to optimal success.

Version 2.0

The remaining *CRUD* facets of *Update* and *Delete* were implemented to positive results, but full success is yet to be reached — while communication between the *Python* module and *MongoDB* is successful, returning the results of the *Update* and *Delete* tests to the *Jupyter* *Notebook* were unsuccessful\*.

*\*(Hence why the screenshots above for these two* CRUD *components are unique from those of the* Create *and* Read *components)*

Version 3.0

Prior testing did not rectify logistical errors and syntax inaccuracies; the *CRUD* facets — namely, *Update* and *Delete* — were revitalized and re-scripted.

Also understand that the *CRUD.py* module possesses some inherent *MongoDB* tools that allow proper data handling within the module and the shell. For example, updating data within *MongoDB* can be risky as improperly doing so can change a document instead of updating its fields. To alleviate this worry, the “$set” operator is utilized in the *def update()* function to update fields within a document instead of replacing all the fields of the document with whatever fields are specified — whether the same ones are specified or not.

A dashboard was created within a *Jupyter Notebook* to interface the *CRUD* module with *MongoDB* and its animal data collection.

The dashboard is an amalgamation of widgets such as a data table, pie chart, and map to showcase the animal information and allow user connectivity and interaction.

**Required Functionality**

This project required three main components: *Python* interaction with *MongoDB*, *Python* scripting for data widgets, and a dashboard that combines everything into one unit.

*Jupyter Notebook* made both creating the dashboard and containing all necessary files in one place easy.

PyMongo bridges the gap between *MongoDB* and *Python* and allows *Python* statements to “do stuff” within *MongoDB* (so long as a stable server connection is established).

However, the minute details in the coding are what make the project possible. *MongoDB* allows searching for documents via their components — like “filtering” — and this searching is utilized in the *Mongo* shell for reading a document for perusal and updating, deleting, or creating document data. MongoDB’s *find()* and *findOne()* methods are utilized when necessary to return a document itself or a cursor “containing” multiple documents. Python calls to these methods with its own, such as its *find\_one\_and\_update()* method. Beyond this, to search for documents that match specific criteria — such as the filtering done by the dashboard’s radio item buttons — Python utilizes signifier tools such as “$and” or “$or” to eliminate or include documents pertaining to the criteria specified.

## **Challenges Along the Road**

From creating the *CRUD.py* module in its earliest stages to finalizing a map that updates upon a selected animal, this project was wrought with challenges.

Communicating *Python* with *MongoDB* requires an understanding of data types and how they are translated that I did not possess in the beginning of this project. Case in point: it is easy to *Read* a document and print it as it is, but it is something differently entirely to *Read* a document **and** *Update* or *Delete* it unless the proper data type is passed from *Python* to *MongoDB*. One of the more difficult challenges was finding a way to locate an exact document based on its parameters OR the document itself, and then deleting said document. *Python’s find\_one\_and\_delete()* method — and, subsequently, its cousin, *find\_one\_and\_update()* — came in handy here.

Further challenges arose once the dashboard came into existence, but these challenges were more on the minor side of things: how to lay the data table out; how to render features of the data table, such as multi-row selection, abled or disabled; how to update the map based upon a selected row (or, in another mindset, a selected animal). In fact, this last challenge was the literal *last* challenge of the project before I deemed it finalized and ready for submission. Finding success took an understanding of the matrix (the Mathematical concept, not the Oscar-winning film).

A matrix is a grid-like composition of mathematical units, and each unit can be referred to by two numbers: its row number and its column number. Once I remembered this facet about matrices, I was able to figure out how to select a row (animal) and have the map update and showcase specific information about said row (animal) by referring to the row’s individual columns. For example, updating the map with *dff.iloc(selected\_rows[0],[9])* would render the animal at the first row and the information within column number 9, which could be its breed, name, or sex. Overcoming this obstacle overcame the entire project for me.

## **Contribution**

Please, all suggestions on how to improve this project should be shared with me and any other user of this project. Describe any major overhauls prior to attempting. Small changes may be implemented and updated so long as proof of successful testing is submitted.

*Community serves development*.

## **License**

Southern New Hampshire University

## **Contact**

Ryan Rieth