# CS 340 README – Justin Dougherty

## About the Project/Project Title

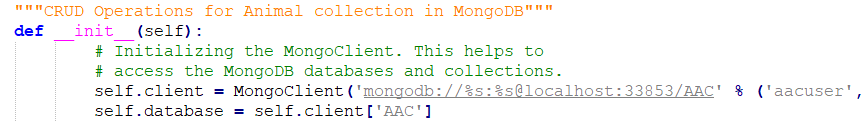
The primary objective of this web application is to furnish users with an intuitive interface for navigating through data housed within a MongoDB-powered database. This interface facilitates the retrieval of preferred combinations of breed, sex, and age, intended for the training of rescue dogs. The project encompasses three essential components: a database infrastructure, an API system, and a dynamically responsive dashboard interface.

**Motivation for Using Mongo DB**

MongoDB was chosen for utilization within this program due to its capacity to swiftly establish a database using a CSV document, coupled with its compatibility with Python. While Python has the capability to interact with database tools such as SQL, the dissimilar syntax between the two can lead to cumbersome transitions between them. In contrast, executing fundamental Create, Read, Update, and Delete (CRUD) operations in a MongoDB database, facilitated by Python, is notably simpler compared to the equivalent procedures in a SQL database.

**Motivation for Using Dash**

Dash was chosen as the preferred framework for constructing the dashboard due to its dynamic characteristics. Dash, which relies on the React JavaScript framework, offers an exceptionally responsive environment. It utilizes HTML Dash tags to govern outputs within segments. Subsequently, any modifications to the designated inputs trigger updates according to the instructions set within the Python module's app callbacks.

1. To replicate this program locally, follow a series of steps. The initial phase involves establishing a Mongo Database, succeeded by crafting a Python module for CRUD operations to interact with said database. The concluding stride entails the development of a Dash web application that incorporates the Python CRUD module. This multi-layered application offers real-time responsiveness to modifications within the table.
2. Mongo Database Setup: Begin by creating a Mongo Database, denoting it as "AAC". Inside this database, shape your required structures.
3. User Authorization Configuration: Generate a user account endowed with read and write permissions exclusively for the AAC database. This user account will be essential for database interaction.
4. Data Import from CSV: Import data from the "aac\_shelter\_outcomes.csv" file into the designated database. Overcoming import-related hitches might necessitate incorporating the user and password of the AAC database during the import process via the Linux terminal. Alternatively, if encountering complications while running Mongo with the "-noauth" flag, the inclusion of user credentials in the import command is recommended. Be vigilant about updating the port number in the file import procedure. This port number, provided when initiating the Mongo service, requires alignment for successful data transfer.
5. Example Import Command: Execute the import process while logged in as "aacuser" or "admin".
6. Update the command below with your specific details:
7. mongoimport --host <hostname> --port <port\_number> --username aacuser --password <password> --authenticationDatabase AAC --db AAC --collection <collection\_name> --type csv --file aac\_shelter\_outcomes.csv --headerline
8. Python Code Configuration: Within the "Animal\_App.py" Python code, make the following adjustments:
9. Update the "localhost" port number to ensure compatibility with the chosen port number during database setup.
10. Replace the "aacuser" and "Password" placeholders with the precise username and password you established for database access.
11. By meticulously following these steps, you'll be equipped to replicate the program on your local environment. This approach amalgamates database setup, user authentication, data importation, code adaptation, and user configuration for a coherent and effective execution
12. Update the “aacuser” and “Password” to the username and password you created.
    1. Initialization (Example): 

**Test Execution:** To perform tests, integrate the testing code into a Jupyter notebook. Ensure that for the *create* function, the test data differs with each iteration or delete the added record between individual tests.

**Dashboard Creation:** Develop a new web application dashboard using Dash. Configure the layout using desired HTML/CSS elements, and assign appropriate identifiers to the data frame, map, and chart components.

**Initial Data Population:** Establish an application callback to populate the initial data frame with the entire dataset.

**Radial Options and Query Programming:** Generate radial options that align with the client's breed preferences. Program database queries based on these preferences. This phase involves crafting intricate queries to amalgamate multiple searches into a consolidated search, producing the intended outcomes. Caution is necessary due to the complexity of the syntax during this process.

**Map Update Mechanism:** Create an application callback to update the map with the first item within a specified category. Modify this mechanism to display the user-selected item on the map. This stage, though complex, necessitates careful handling. The provided approach of utilizing "selected\_virtual\_row\_ids" is advised against; instead, directly utilize "selected\_rows" for smoother integration.

**Pie Chart Generation:** Construct a pie chart using the currently displayed data. Develop an application callback to update the pie chart based on filtered data. Note that there were challenges in the process, such as discrepancies in Plotly's documentation regarding the "values" parameter. The accurate structure involves an index of names under "names" and an array of occurrences under the "values" variable.

**Installation:** To operate this application, you will require Jupyter Notebooks, Python accessible via the command line, and MongoDB. Below are installation instructions for each component:

- \*\*Jupyter Notebooks:\*\*

To install Jupyter, follow the straightforward instructions outlined at https://jupyter.org/install. For more comprehensive insights, including configurations for Proxy servers across diverse operating systems, refer to the guidelines available at https://jupyterlab.readthedocs.io/en/stable/getting\_started/installation.html.

- \*\*Python:\*\*

Detailed installation guidance for Python can be accessed at https://realpython.com/installing-python/. Once Python is successfully installed, you can execute the program either via the Terminal (on Mac/Linux) or the Command Prompt (on Windows).

- \*\*MongoDB:\*\*

MongoDB is offered in both Community and Enterprise editions. For thorough instructions on installation, visit https://docs.mongodb.com/manual/installation/.

By following these instructions meticulously, you'll be equipped to launch and operate the application effectively.

**Required Libraries:**

For the proper generation of charts, ensure you import Plotly, a Python charting tool, into your module. Import Dash Core Components for web application building. Additionally, Pandas is used to create data frames for this web application. Check the installation details for Plotly at <https://www.journaldev.com/19692/python-plotly-tutorial#:~:text=Installation.%20To%20install%20plotly%2C%20open%20a%20terminal%20window,to%20install%20to%20collect%20dependencies%20and%20download%20them%3A>.

**Dashboard Framework:** Dash, a framework for constructing web applications, is instrumental. Import Dash Core Components into your Jupyter notebook, and install Dash using the guidance available at <https://pypi.org/project/dash/>.

**Data Manipulation:** Pandas plays a crucial role in data manipulation within this application. Detailed information about Pandas, including dependencies and installation steps, can be found at <https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html>.

**Functionality Overview:** This application encompasses three primary functions. The first involves utilizing radial buttons to categorize data based on breed, age, and sex characteristics, catering to Grazioso Salvare's desired rescue dog types. Clicking these radial buttons triggers database queries, updating the data frame with the sought-after results. The "Reset" radial button restores the table to its unfiltered state. The second function concerns the interactive map and its updates. The map starts with a marker at the position of the first item in the data frame. Upon user selection (up to five at a time), the map's markers shift to represent the selected row(s). The third function is the dynamic pie chart, which sorts data by breed and generates a pie chart based on the current data frame, specifically focused on animal breeds.

**Functionality Expansion:** This application encompasses four key functions. The initial two functions gather user input for database entry (*create*) or database searches (*read*). The subsequent two functions execute the actual creation and search tasks. The "obtain" functions prompt users for specific inputs (in the *read* function, a key and search value; in the *create* function, only values for provided keys). The *create* function adds a Python dictionary as a new item to MongoDB and returns a Boolean indicating success or failure. The *read* function returns False if no data is available or returns matching data.

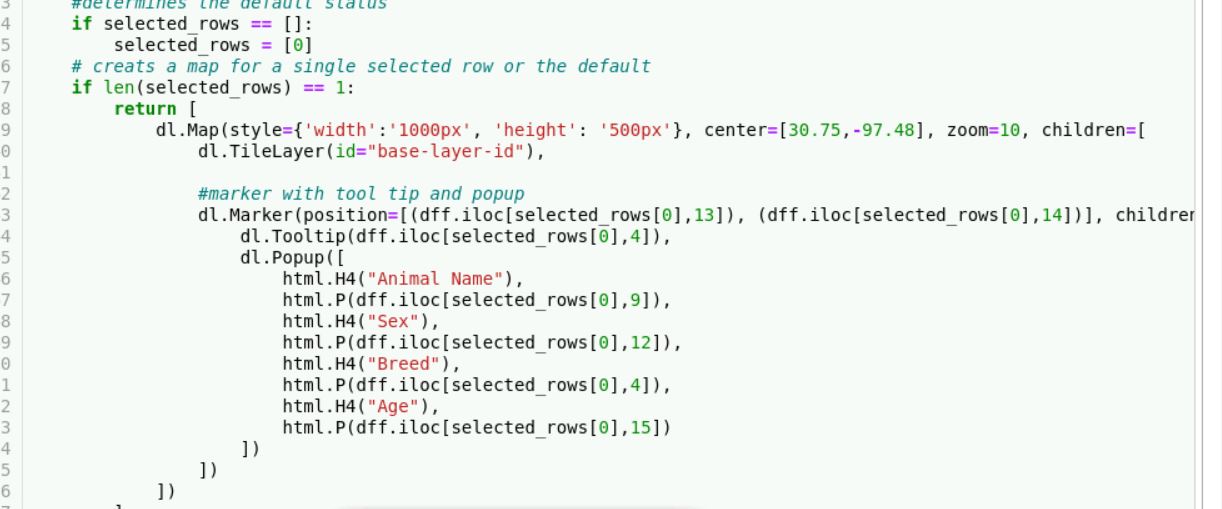
**Data Reset:** To display all data, a "Reset" option is available, which restores the table to its unfiltered state.

**Code Samples from Dashboard**

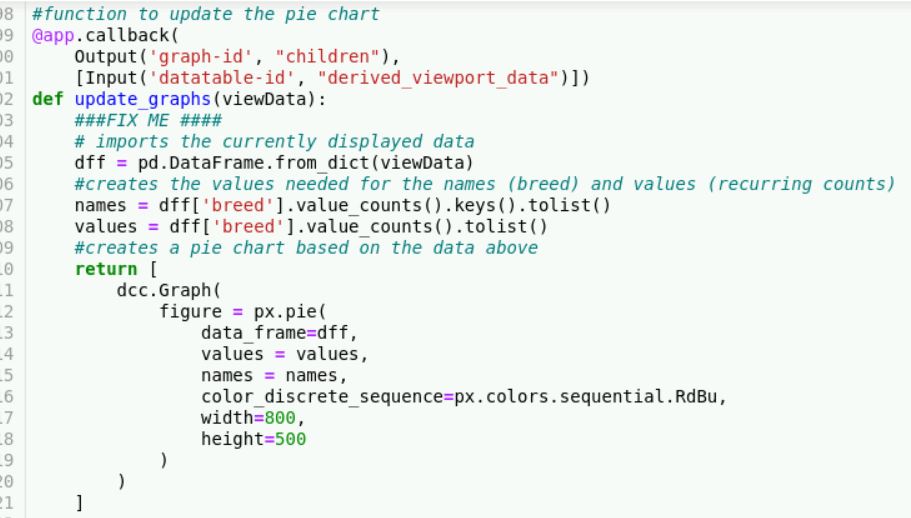
*Complex Query*

**

*Map Markers*

**

*Pie Chart*

**

**Here are the provided code examples refined for clarity and readability:**

**CRUD Operations:**

**Create:**

*def create(data):*

*try:*

*if data is not None:*

*insert\_result = self.database.animals.insert\_one(data)*

*pprint(insert\_result)*

*return True*

*else:*

*raise Exception("Exception content")*

*except:*

*return False*

*\*\*Read:\*\**

*```python*

*def read(target):*

*try:*

*if target is not None:*

*read\_result = list(self.database.animals.find(target))*

*pprint(read\_result)*

*return True*

*else:*

*raise Exception("Exception content")*

*return False*

*except Exception as e:*

*print("Message:", e)*

***Update***

*def update(source, destination, count):*

*try:*

*if source is not None:*

*if count == 1:*

*update\_result = update(self.database.animals.update\_one(source, destination))*

*pprint(update\_result)*

*return True*

*elif count == 2:*

*update\_result = update(self.database.animals.update\_many(source, destination))*

*pprint(update\_result)*

*return True*

*except:*

*raise Exception("Exception content")*

*return False*

***Delete:***

*def delete(target, count):*

*try: if target is not None:*

*if count == 1:*

*delete\_result = delete(self.database.animals.delete\_one(target))*

*pprint(delete\_result)*

*return True*

*elif count == 2:*

*delete\_result = delete(self.database.animals.delete\_many(target))*

*pprint(delete\_result) return True*

*except:*

*raise Exception("Exception content")*

*return False*

***Dashboard:***

***Radial Buttons:***

*# Import view data*

*if radial\_one\_selected:*

*# Run complex query*

*# Update view data*

*return view\_data*

***Map:***

*# Import view data*

*if selected\_rows is None:*

*selected\_rows = 0*

*if len(selected\_rows) == 1:*

*# Return map with 1 map marker and tooltip*

*elif len(selected\_rows) == 2:*

*# Return map with 2 map markers and tooltips # Etc.*

***Pie Chart:***

*# Import view data*

*names = data(breed) # obtain and transfer to list*

*values = data(breed) # obtain occurrence counts and transfer to list*

*# Return graph using the view data, names as the search key, and values as the pie slice values*

### Tests

### The "create" function is subjected to a pair of unit tests, mirroring the structure found in the "read" function. In both scenarios, one test scrutinizes the potential failure scenario, while the other evaluates the outcome in a successful context. The primary objective of the "create" function's tests is to validate the truthfulness of the return value, denoting either success (True) or failure (False). It is important to acknowledge that additional tests have been devised to meticulously explore the multifaceted functionalities of each individual CRUD method.

### A representative code excerpt follows:

### def testData():

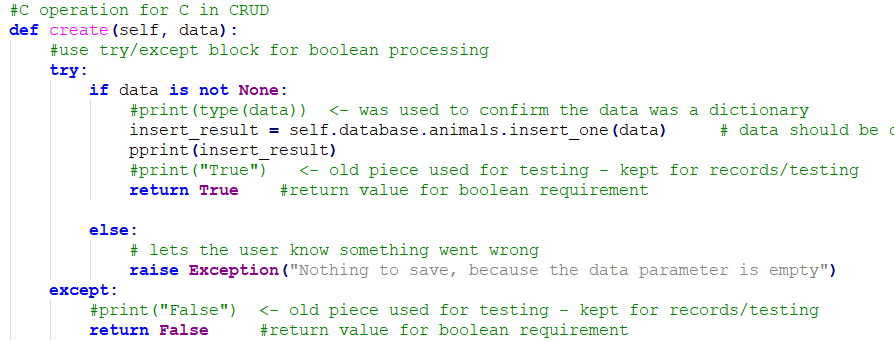
### testShelter = animalShelter()

### assertTrue(testShelter.create(sampleData))

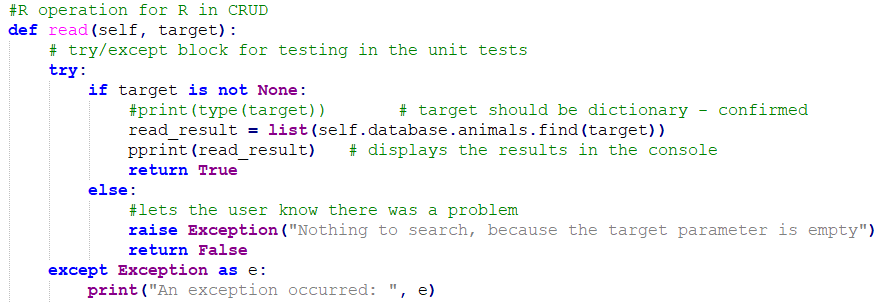
### This code snippet functions as a blueprint for test execution. An instance of the "animalShelter" class is instantiated, and an assertion is invoked to affirm the accomplishment of the "create" operation through the utilization of sample data. This systematic approach to testing guarantees a thorough appraisal of the capabilities encapsulated within these functions..

### Screenshots

Sample Create:

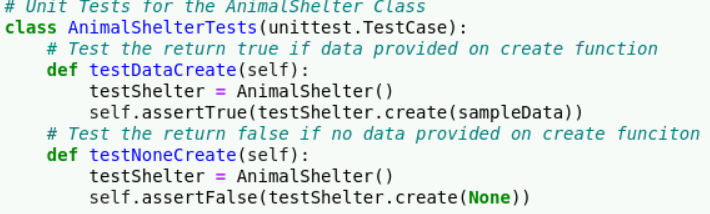


Sample Read:

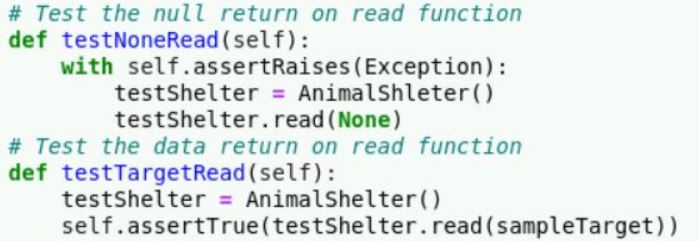


Sample Tests:

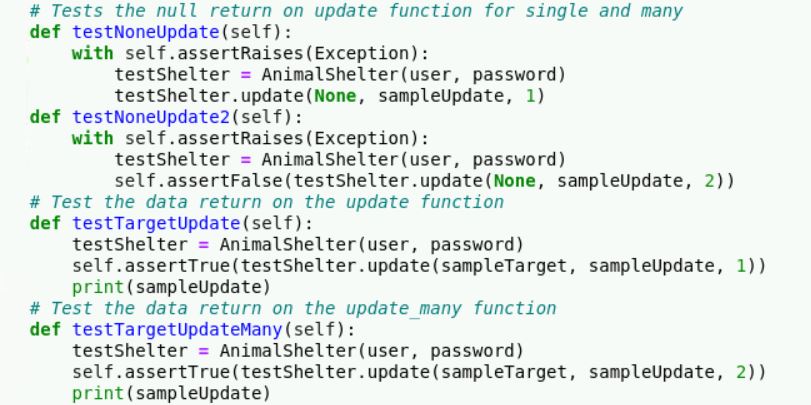
*Create*



*Read*

**

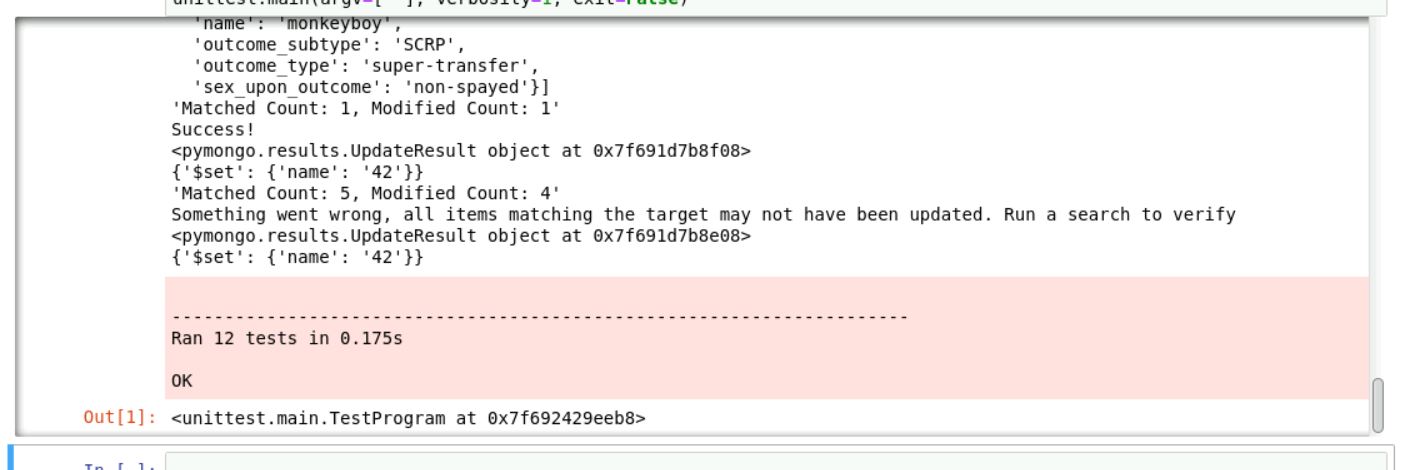
*Update*

**

*Delete*

**

*Test Results:*

**

**Disaster Rescue Dashboard**

A screenshot of a dashboard

Description automatically generated

**Water Rescue**

A screenshot of a computer

Description automatically generated

Multi SelectA close-up of a map

Description automatically generated

Contact

Justin.dougherty@snhu.edu