MODULE 5-2 MILESTONE FOUR:

ENHANCEMENT THREE: DATABASES

Arturo Santiago-Rivera

Prof. Brooke Goggin, M.S., M.S., M.S., Ed.D (ABD)

CS-499 Computer Science Capstone 22EW4

Southern New Hampshire University

April 3, 2022

Module 5-2 Milestone Four: Enhancement Three: Databases

This paper is a narrative that accompanies the artifact enhancements for databases. It explains why the selected artifact was included in this category of our ePortfolio and reflects on the process used to create the artifact. The narrative focuses on the learning that happened through the artifact's creation (Southern New Hampshire University, 2022).

Prompt

The artifact selected for the databases category is the Salvare Search for Rescue Web App. The web application aims to develop a web interface that works with an existing database from animal shelters to identify and categorize available dogs to train for different rescues. The web app was planned, designed, and developed for the CS340 Client/Server Development computer science course. The application is developed in Python with the Dash framework and the non-relational database MongoDB. The integration of Python and MongoDB is through the python driver PyMongo. The application can run in Jupyter Notebook as a test tool and the computer terminal with an internet browser. The application's functionality involves a CSV data file of existing dogs in shelters to import into MongoDB, the import of dependencies such as Python PyMongo driver, Python libraries, Dash framework, and a Python source code and CRUD module to manipulate the data imported into MongoDB.

This artifact was selected because it involved a multi-tier application with a Model View Controller (MVC) and RESTful protocol design to extend the HTTP protocol to provide an application programming interface (API). The most attractive concept of the MVC pattern is a separation of concerns. The model's job is to manage the data, MongoDB, and Python data structures. The view's job is to decide what the user sees and how on their screen, Dash framework. The controller's responsibility is to pull, modify, and provide data to the user, PyMongo driver.

Each record in the MongoDB database is a document described in BSON, a binary representation of the data retrieved by the web app in a JSON format. When the main app file is executed in the computer terminal, the user opens an internet browser new tab pointing to the corresponding app address (ex. http://127.0.0.1:8050/). The browser starts loading and generates the client-facing web application dashboard like the following screen:

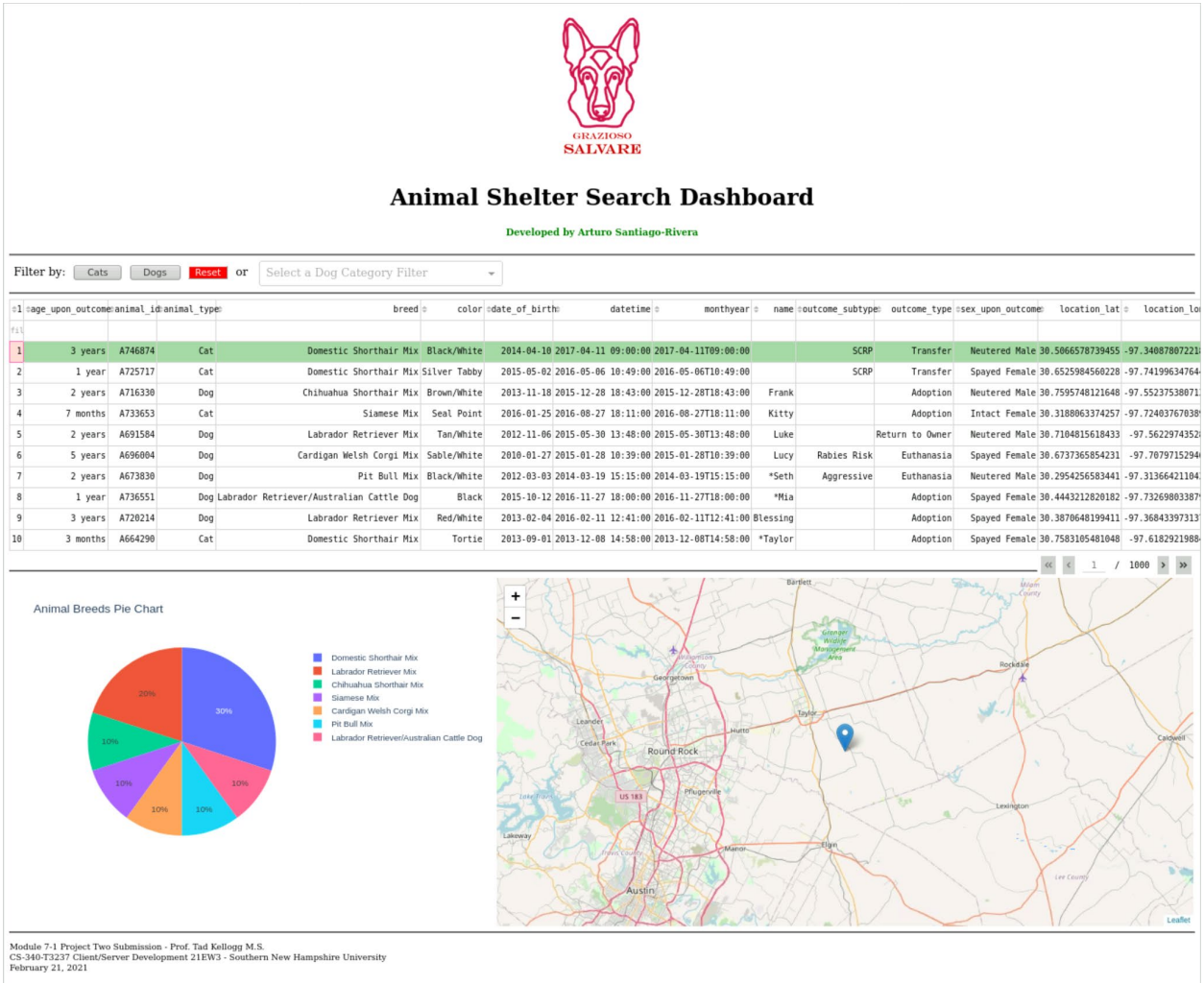


Figure 1 Web App Client/Database Dashboard

The artifact involves the engineering of practices of validating input data and architect and design with default denial when accessing databases records. This skill in me a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential

vulnerabilities, mitigate design flaws, and ensure privacy and enhanced data security and resource. The source code is clearly and adequately documented with an easy-to-maintain commenting style and consistent with Python standards and supports clean code through descriptive functions names and variable names. The CRUD module that handles the database records supports code reusability by importing it as a module by other Python scripts.

```python

class AnimalShelter(object):
    """ CRUD operations for Animal collection in MongoDB """

    def __init__(self, username, password, dbname):
        # Please change the port to the localhost.
        port = 27017
        # Initializing the MongoClient. This helps to
        # access the MongoDB databases and collections.
        self.client = MongoClient(
            'mongodb://%s:%s@localhost:%d/?authMechanism=DEFAULT&authSource=%s' % (username, password, port, dbname))

        # Set database and list collection in database
        self.database = self.client[dbname]

        # Set collection to use
        collname = 'animals'
        self.collection = self.database[collname]

# Create method to implement the C in CRUD.
    def create(self, data):
        if data is not None:
            # data should be a list of one or more dictionary
            return self.collection.insert_many(data)
        else:
            raise Exception(
                "[-] ERROR: Nothing to save, because data parameter is empty.")

# Read method to implement the R in CRUD.
    def read(self, data):
        if data is not None:
            # data should be dictionary
            return self.collection.find(data, {"_id": False})
        else:
            raise Exception(
                "[-] ERROR: Nothing to read, because data parameter is empty.")

# Update method to implement the U in CRUD.
    def update(self, data):
        if data is not None:
            # data should be dictionary
            return self.collection.update_many(data)
        else:
            raise Exception(
                "[-] ERROR: Nothing to update, because data parameter is empty.")

# Delete method to implement the D in CRUD.
    def delete(self, data):
        if data is not None:
            return self.collection.delete_many(data)  # data should be dictionary
        else:
            raise Exception(
                "[-] ERROR: Nothing to delete, because data parameter is empty.")

```

Figure 2 AnimalShelter Class CRUD Module

The implemented data structure is programmatic, where the stored variable values can be used efficiently in different functions and callbacks through the web application. This approach

improves the design and evaluation of computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to it while managing the trade-offs involved in design choices.

```
217
218   # This callback will highlight a column or row on the data table when the user, at first, selects it on the currently visible page
219   @app.callback(
220       Output('datatable-id', 'style_data_conditional'),
221       [Input('datatable-id', 'selected_columns'),
222        Input('datatable-id', "derived_viewport_selected_rows"),
223        Input('datatable-id', 'active_cell')]
224   )
225   def update_styles(selected_columns, selected_rows, active_cell):
226       if active_cell is not None:
227           style = [{
228                       'if': { 'row_index': active_cell['row'] },
229                       'background_color':'#a5d6a7'
230                   }]
231       else:
232           style = [{
233                       'if': { 'row_index': i },
234                       'background_color':'#a5d6a7'
235                   } for i in selected_rows]
236
237       return (style +
238                   [{
239                       'if': { 'column_id': i },
240                       'background_color': '#80deea'
241                   } for i in selected_columns]
242               )
243
244   # This callback add a pie chart that displays breed percentage from the interactive data table
245   @app.callback(
246       Output('graph-id', "children"),
247       [Input('datatable-id', "derived_viewport_data")]
248   )
249   def update_graphs(viewData):
250       ### DONE: ####
251       dff = pd.DataFrame.from_dict(viewData)
252
253       # code for pie chart
254       fig = px.pie(
255           dff,
256           names='breed',
257           title='Animal Breeds Pie Chart'
258       )
259
260       return [dcc.Graph(figure=fig)]
```

Figure 3 App Source Code Example Screenshot

The artifact enhancements focused on recreating the web app in a Windows OS environment since the original web app was developed in the Apporto Virtual Lab, a university remote desktop Linux platform. Using the web app initial documentation, I follow the guidance on how to recreate the web app environment and update/revise the documentation according to the actions executed to replicate and run the web app in a Windows terminal. The process was challenging because others set the initial Linux environment where the web app was developed. In our case, we have to go through the setup Python and MongoDB process before starting to follow the guidance in the web application documentation. The setup process as well the update

of the source code functionality because of changes in the PyMongo driver new version and the use of an upgraded MongoDB platform, improved my ability to use well-founded and innovative techniques, skills, and tools in computing practices to implement computer solutions that deliver value and accomplish industry-specific goals.

```
28
29 # Create method to implement the C in CRUD.
30     def create(self, data):
31         if data is not None:
32             docs = self.collection.insert(data)  # data should be a list of one or more dictionary
33             print("[+] Document Created Succesfully %s" % len(docs))
34             print("-------------------------------------")
35             for doc in docs:  # iterate docs to list the ObjectId of the documents created
36                 print("ObjectId => %s" % doc)
37         else:
38             raise Exception(
39                 "[-] ERROR: Nothing to save, because data parameter is empty.")
40
```

Figure 4 Original code of Create Method using PMongo v3.0

```
31
32 # Create method to implement the C in CRUD.
33     def create(self, data):
34         if data is not None:
35             docs = self.collection.insert_many(data)  # data should be a list of one or more dictionary
36             print("[+] Document Created Succesfully %s" % len(data))
37             print("-------------------------------------")
38             for doc in docs.inserted_ids:  # iterate docs to list the ObjectId of the documents created
39                 print("ObjectId => %s" % doc)
40             return print("-------------------------------------\n*** End of List ***")
41         else:
42             raise Exception("[-] ERROR: Nothing to save, because data parameter is empty.")
```

Figure 5 Modified Code of Create Method Using PyMongo v4.0

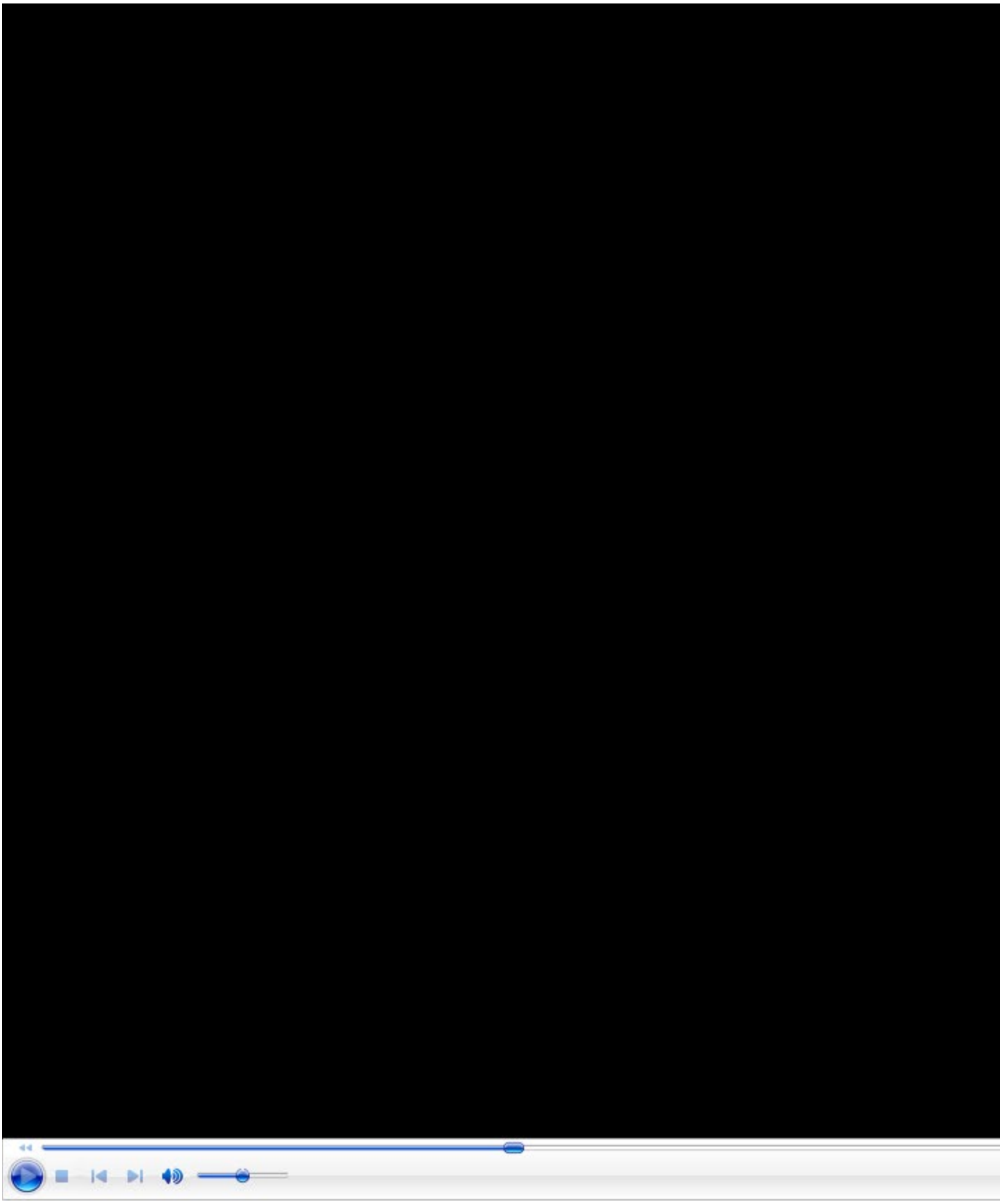
In this artifact, I employ industry-standard Python code best practices and techniques such as in-line comments, appropriate naming conventions, and formatting in conformance with proper coding standards, making the code easy to read and enhancing the application code organization. The program code is easy to read and follows formatting best practices defined by the industry, such as indentation in conformance with appropriate coding standards. The source code is well-structured, consistent in style, and consistently properly formatted, including line breaks. We utilize appropriate syntax and conventions in terms of their best practice and use in programming. The implemented data structures are programmatic, where the stored variable values can be used efficiently in other methods. Method names are verbs as they represent actions being performed on something. All cases are covered in an IF-ELIF block, including ELSE or DEFAULT clauses.

```python

    ############################################
    # Interaction Between Components / Controller
    ############################################

    # DONE: This callback add interactive dropdown filter option to the dashboard to find dogs per category
    # or interactive button filter option to the dashboard to find all cats or all dogs
@app.callback(
        Output('datatable-id', 'data'),
        [Input('filter-type', 'value'),
         Input('submit-button-one', 'n_clicks'),
         Input('submit-button-two', 'n_clicks')]
    )
def update_dshboard(selected_filter, btn1, btn2):
    if (selected_filter == 'drit'):
        df = pd.DataFrame(list(shelter.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":{"$lte":300}
                }
            )
        ))
    elif (selected_filter == 'mwr'):
        df = pd.DataFrame(list(shelter.read(
                {
                    "animal_type":"Dog",
                    "breed":{"$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 (selected_filter == 'wr'):
        df = pd.DataFrame(list(shelter.read(
                {
                    "animal_type":"Dog",
                    "breed":{"$in":["Labrador Retriever Mix","Chesapeake Bay Retriever","Newfoundland"]},
                    "sex_upon_outcome":"Intact Female",
                    "age_upon_outcome_in_weeks":{"$gte":26},
                    "age_upon_outcome_in_weeks":{"$lte":156}
                }
            )
        ))
    # higher number of button clicks to determine filter type
    elif (int(btn1) > int(btn2)):
        df = pd.DataFrame(list(shelter.read({"animal_type":"Cat"})))
    elif (int(btn2) > int(btn1)):
        df = pd.DataFrame(list(shelter.read({"animal_type":"Dog"})))
    else:
        df = pd.DataFrame.from_records(shelter.read({}))

    data = df.to_dict('records')

    return data
```

Figure 6 App Source Code IF-ELIF Example

As a document database, MongoDB brings a facility to manage a significant amount of data and makes it easy to store structured or unstructured data. It uses a JSON-like format to store documents. This format directly maps to native objects in modern programming languages, making it a natural choice as I don’t need to think about normalizing data. The web application's CRUD is transparent and straightforward. However, the dashboard development using the Dash framework was more time-consuming. Understanding and exploring how the dash core, HTML components, and callbacks work to produce an efficient and straightforward coding structure.

There is a lot behind the framework, and their libraries are under active development, so installation and upgrade frequently are necessary. With the reproduction of the web application in a different environment, I update and actualize the web app documentation to guide others in recreating the web app in a Linux or Windows environment. The clear and well-documented guide improves my skill in designing, developing, and delivering professional written and visual communication documentation that is coherent, technically sound, and appropriately adapted to specific audiences and contexts.



As shown in the short video, I accomplished the process of recreating the web app in a Windows environment. I produced a working program beyond the initial Linux environment, which required researching things I have done in other languages to produce an artifact on multiple operating systems like Windows and macOS.

References

Southern New Hampshire University. (2022, March 4). *CS499 Final Project Guidelines and Rubric.* Retrieved from Module 1-3 Final Project Review: https://learn.snhu.edu/content/enforced/1014915-CS-499-T4547-OL-TRAD-UG.22EW4/Course%20Documents/CS%20499%20Final%20Project%20Guidelines%20and%20Rubric.pdf?_&d2lSessionVal=sZSvR1M8MBrfiTVb6OZOu3IQB&ou=1014915

Southern New Hampshire University. (2022, March 31). *Milestone Four Guidelines and Rubric Enhancement Three: Databases*. Retrieved from Module 5-2 Milestone Four: Enhancement Three: Databases: https://learn.snhu.edu/d2l/common/dialogs/quickLink/quickLink.d2l?ou=1014915&type=coursefile&fileId=Course+Documents%2fCS+499+Milestone+Four+Guidelines+and+Rubric.pdf