**Milestone Two Narrative – Software Design and Engineering**

**Artifact Description**

The artifact is a **Python Dash dashboard** project created for **CS 340: Client/Server Development**. It connects a **Dash client dashboard** to a **MongoDB server** via the AnimalShelter CRUD module. The dash ## Setup the Jupyter version of Dash

from jupyter\_dash import JupyterDash

from dash import dcc, html, dash\_table

from dash.dependencies import Input, Output

import dash\_leaflet as dl

import plotly.express as px

import pandas as pd

import base64

import requests

# Flask server for RESTful API

from flask import Flask, jsonify, request

from AnimalShelter import AnimalShelter

# -----------------------------

# Database Connection

# -----------------------------

username = "aacuser"

password = "SNHU1234"

db = AnimalShelter(username, password)

# -----------------------------

# Flask API Setup

# -----------------------------

server = Flask(\_\_name\_\_)

# Dummy user session for RBAC demonstration

current\_user = {"username": "lilly", "role": "admin"} # Change "admin" -> "user" to test RBAC

@server.route('/api/animals', methods=['GET'])

def get\_animals():

query = request.args.to\_dict()

if "age\_upon\_outcome\_in\_weeks" in query:

query["age\_upon\_outcome\_in\_weeks"] = {"$lt": int(query["age\_upon\_outcome\_in\_weeks"])}

results = db.read(query)

return jsonify(results)

@server.route('/api/animals', methods=['POST'])

def add\_animal():

if current\_user["role"] != "admin":

return jsonify({"error": "Unauthorized"}), 403

data = request.json

db.create(data)

return jsonify({"status": "success"})

# Additional endpoints for PUT/DELETE can be added similarly

# -----------------------------

# Dash App Setup

# -----------------------------

app = JupyterDash(\_\_name\_\_, server=server)

# Load initial data from MongoDB

df = pd.DataFrame.from\_records(db.read({}))

if "\_id" in df.columns:

df.drop(columns=["\_id"], inplace=True)

# Encode logo image

image\_filename = 'grazioso.png'

encoded\_image = base64.b64encode(open(image\_filename, 'rb').read()).decode()

# -----------------------------

# Helper Functions

# -----------------------------

def fetch\_animals(filter\_type="all"):

# Define rescue filters

rescue\_filters = {

"Water Rescue": {"breed": ["Labrador Retriever Mix", "Chesapeake Bay Retriever", "Newfoundland"],

"sex\_upon\_outcome": {"$regex": "Intact Female"},

"age\_upon\_outcome\_in\_weeks": {"$gte": 26, "$lte": 156}},

"Mountain or Wilderness Rescue": {"breed": ["German Shepherd", "Alaskan Malamute", "Old English Sheepdog", "Siberian Husky", "Rottweiler"],

"sex\_upon\_outcome": {"$regex": "Intact Male"},

"age\_upon\_outcome\_in\_weeks": {"$gte": 26, "$lte": 156}},

"Disaster or Individual Tracking": {"breed": ["Doberman Pinscher", "German Shepherd", "Golden Retriever", "Bloodhound", "Rottweiler"],

"sex\_upon\_outcome": {"$regex": "Intact Male"},

"age\_upon\_outcome\_in\_weeks": {"$gte": 20, "$lte": 300}}

}

if filter\_type == 'all':

query = {"age\_upon\_outcome\_in\_weeks": {"$lt": 104}}

else:

query = {

"breed": {"$in": rescue\_filters[filter\_type]["breed"]},

"sex\_upon\_outcome": rescue\_filters[filter\_type]["sex\_upon\_outcome"],

"age\_upon\_outcome\_in\_weeks": rescue\_filters[filter\_type]["age\_upon\_outcome\_in\_weeks"]

}

response = requests.get("http://localhost:8051/api/animals", params={"age\_upon\_outcome\_in\_weeks": 104 if filter\_type=="all" else None})

data = response.json()

df\_filtered = pd.DataFrame.from\_records(data)

if "\_id" in df\_filtered.columns:

df\_filtered.drop(columns=["\_id"], inplace=True)

return df\_filtered

def generate\_pie\_chart(df):

breed\_counts = df['breed'].value\_counts().reset\_index()

breed\_counts.columns = ['Breed', 'Count']

threshold = 1

total\_count = breed\_counts['Count'].sum()

breed\_counts['Percentage'] = (breed\_counts['Count'] / total\_count) \* 100

small\_breeds = breed\_counts[breed\_counts['Percentage'] < threshold]

other\_count = small\_breeds['Count'].sum()

top\_n = 10

breed\_counts = breed\_counts.sort\_values('Count', ascending=False)

if len(breed\_counts) > top\_n:

breed\_counts = breed\_counts.iloc[:top\_n]

breed\_counts = breed\_counts.append({'Breed': 'Other', 'Count': other\_count}, ignore\_index=True)

fig = px.pie(breed\_counts, names='Breed', values='Count', title="Breed Distribution (Top 10 + Other)", color\_discrete\_sequence=px.colors.qualitative.Set3)

fig.update\_traces(textposition='inside', textinfo='percent+label')

return fig

def generate\_map(df):

markers = [

dl.Marker(

position=[row.get('latitude', 30.75), row.get('longitude', -97.48)],

children=[dl.Tooltip(row.get('breed', 'Unknown')),

dl.Popup([html.H1(row.get('name', 'Unknown')),

html.P(f"Age: {row.get('age\_upon\_outcome', 'Unknown')}")])]

)

for \_, row in df.iterrows()

]

return dl.Map(style={'width': '1000px', 'height': '500px'}, center=[30.75, -97.48], zoom=10,

children=[dl.TileLayer()] + markers)

# -----------------------------

# Layout

# -----------------------------

app.layout = html.Div([

html.Center(html.B(html.H1('CS-340 Dashboard'))),

html.Img(src=f"data:image/png;base64,{encoded\_image}", style={'width': '200px'}),

html.P("Dashboard by: Lilly"),

html.Hr(),

html.Label("Filter by Rescue Type"),

dcc.Dropdown(

id='filter-type',

options=[{'label': t, 'value': t} for t in ["All", "Water Rescue", "Mountain or Wilderness Rescue", "Disaster or Individual Tracking"]],

value='All',

multi=False

),

html.Br(),

dash\_table.DataTable(

id='datatable-id',

columns=[{"name": i, "id": i} for i in df.columns],

data=df.to\_dict('records'),

page\_size=10,

style\_table={'overflowX': 'auto'},

row\_selectable='single',

selected\_rows=[0]

),

html.Br(),

html.Hr(),

html.Div(className='row', style={'display': 'flex'}, children=[

html.Div(id='graph-id', className='col s12 m6'),

html.Div(id='map-id', className='col s12 m6')

])

])

# -----------------------------

# Callbacks

# -----------------------------

@app.callback(

Output('datatable-id', 'data'),

Input('filter-type', 'value')

)

def update\_table(filter\_type):

df\_filtered = fetch\_animals(filter\_type)

return df\_filtered.to\_dict('records')

@app.callback(

Output('graph-id', 'children'),

Input('datatable-id', 'derived\_virtual\_data')

)

def update\_graph(viewData):

if not viewData:

return "No data available"

df\_view = pd.DataFrame.from\_dict(viewData)

if 'breed' not in df\_view.columns or df\_view['breed'].isnull().all():

return "No breed data available"

fig = generate\_pie\_chart(df\_view)

return dcc.Graph(figure=fig)

@app.callback(

Output('map-id', 'children'),

Input('datatable-id', 'derived\_virtual\_data')

)

def update\_map\_callback(viewData):

if not viewData:

return "No data available"

df\_view = pd.DataFrame.from\_dict(viewData)

return generate\_map(df\_view)

# -----------------------------

# Run the Dash App

# -----------------------------

app.run\_server(mode='inline', debug=True, port=8051)board currently provides:

* Interactive filtering of animals by rescue type
* A data table displaying animal records
* Breed distribution visualizations using Plotly
* Geolocation mapping with Dash Leaflet

The artifact was created as part of a class project to demonstrate client/server integration, real-time data visualization, and basic CRUD operations.

**Justification for Inclusion**

I selected this artifact for my ePortfolio because it demonstrates:

* **Software design skills:** Integration of client and server components, modular callbacks, and data filtering.
* **Data visualization skills:** Pie charts for breed distributions and interactive maps.
* **Database interaction:** Use of MongoDB for storage and retrieval of structured data.

Enhancing this artifact allows me to demonstrate **advanced software engineering skills**, including modular system design, API implementation, and security improvements. These enhancements make the project more professional and industry-relevant.

**Enhancement Plan**

The enhancements I implemented for this milestone are:

1. **Role-Based Authentication (RBAC)**
   * Added **admin and user roles** to control access to certain operations (like adding or deleting animal records).
   * Integrated current\_user.role checks to restrict unauthorized operations in CRUD methods.
2. **RESTful API Layer (Flask)**
   * Replaced direct database access from the dashboard with **API endpoints**.
   * All CRUD operations now go through the API layer for better modularity, logging, and future security enhancements.

**Example of RESTful API integration:**

@app.route('/animals', methods=['GET'])

def get\_animals():

query = request.args

results = db.read(query)

return jsonify(results)

@app.route('/animals', methods=['POST'])

def add\_animal():

data = request.json

if current\_user.role == 'admin':

db.create(data)

return jsonify({"status": "success"})

else:

return jsonify({"error": "Unauthorized"}), 403

1. **Improved Modularity**
   * Refactored callbacks into separate functions to **decouple dashboard logic from database logic**.
   * Encapsulated filtering, charting, and mapping into **service modules** for cleaner architecture.
2. **Future Enhancements (Optional)**
   * Add logging/auditing of CRUD operations for security monitoring.
   * Implement field-level encryption for sensitive data if needed.

**Skills Demonstrated and Alignment with Course Outcomes**

**Skills Demonstrated:**

* Designing **modular client/server architectures** for maintainability.
* Implementing **RESTful APIs** for secure, scalable data operations.
* Applying **role-based access control** to enforce security.
* Enhancing **interactive data visualizations** (Dash/Plotly) and mapping features.

**Course Outcomes Addressed:**

* **Outcome 3:** Design and evaluate computing solutions using algorithmic principles and standards.
* **Outcome 4:** Use innovative techniques and tools (Dash, MongoDB, REST APIs) to implement solutions.
* **Outcome 5:** Develop a security mindset by adding authentication, RBAC, and secure endpoints.

**Reflection on the Enhancement Process**

Enhancing this project allowed me to improve my **software engineering design skills** while learning **best practices for modular and secure application development**. I faced challenges in:

* Refactoring tightly coupled code while maintaining functionality.
* Integrating a REST API layer without breaking dashboard callbacks.

Through these enhancements, I learned how to **structure code for scalability**, apply **role-based security**, and improve the **readability and maintainability** of a client/server application. This artifact now better represents my capabilities in **software design, engineering, and security** for my ePortfolio.