CS-340: Assignment 5-2: Module 5 Assignment

Prepared on: October 6, 2025
Prepared for: Prof. Jeff H. Sanford
Prepared by: Alexander Ahmann

Contents

1 Introduction

This writeup outlines the procedure that I took to develop a simple web application in Python's *Dash* framework.[1] In particular, I present a dynamic web interface as a solution to the engineering problem given in this assignment's guidelines (CS-340, n.d.). This web interface is to be implemented in the *Jupyter Notebook*, and it simply interfaces with the given *MongoDB* instance, retrieves data based on an arbitrary filter, and prints out the query results onto the web page.

2 Problem Set and their Respective Solutions

A Jupyter Notebook called `ModuleFiveAssignment.ipynb` has been given as starter code for the *Python Dash* application, and my task is to make changes

[1]See the Dash Documentation: https://dash.plotly.com/

to it in accordance with the assignment guidelines.[2]. A semi-abridged listing of the final Python code is shown in the appendix.

2.1 Task 1: Adding a simple HTML component

> "The code provides you with a very basic dashboard. Add an HTML component, such as a header, to create a unique identifier for your dashboard. This unique identifier could be your name or a specific handle or image that is unique to you."—CS-340 (n.d., Prompt, §2)

This part was fairly straightforward: The Python Dash application renders HTML tags through the `app.layout` object and various HTML tags offered by Dash's `html` class. Specifically, `app.layout` is set to `html.Div([])`, and the contents of the list is what the Dash application's layout is set to render. I added the following line in that list:[3]

```
html.H2("By Alexander Ahmann <alexander.ahmann@snhu.edu>"),
```

This will act as a "unique identifier" stating that I have made changes to this code.

2.2 Task 2: Simple interface to the MongoDB instance

> "Customize the starter code given to you by developing connections between the dashboard username/password interface and your CRUD Python module. The dashboard should prompt the user for their username and password, and then return the output of the test query. Be sure to complete each of the following steps:
>
> - Import the CRUD Python module that you created for Project One.
> - Add the functionality in the callback routine for instantiation of your CRUD object. Remember to apply the user authentication when creating your CRUD object.
> - Finally, add functionality to test your dashboard connection to MongoDB. Add functionality by writing code that returns the following read query: `{"animal_type":"Dog", "name":"Lucy"}`." —CS-340 (n.d., Prompt, §3.1–3.3)

[2]Specifically, concerning prompts §2 and §3

[3]Line 22 in Appendix A.

The solution to this task involves first using the recently invented *Python CRUD module* to perform basic data modelling with the arbitrary filter {"animal_type":"Dog", "name":"Lucy"}, and then rendering the results in the `app.layout`. In regards to rendering, I intend to print out an integer showing the number of documents returned, and an unordered list of each returned document's contents. Plate 1 depicts my approach to the data modelling part.[4]

Plate 1: Worked solution to data modelling requirement

```
1.  from CRUD_Python_Module import *
2.  mongo_instance = AnimalShelter()
3.  lucy_query = {
4.      "animal_type" :"Dog",
5.      "name" :"Lucy"
6. }
7.  execute_lucy_query = mongo_instance.read(lucy_query)
8.  query_results = []
9.  for results in execute_lucy_query:
10.     resultant = ""
11.     for k, v in results.items():
12.         resultant += "key: {0},
    value: {1} ".format(k, v)
13.     query_results.append(resultant)
```

I imported the custom Python CRUD module,[5] instantiated the `AnimalShelter()` class,[6] performed a simple `find()` transaction through the CRUD module's `read()` function,[7] and finally concatinated each key, and its respective value, into a string, and where each string is to be appended into the `query_results` list to later be rendered.[8]

Next, the Python Dash application actually had to print the results onto the web page. Plate 2 depicts the code listing that I used to accomplish this.[9]

[4]Excerpt from Appendix A, specifically lines 5, and 7–18.

[5]Plate 1, Line 1.

[6]Plate 1, Line 2.

[7]Plate 1, Lines 3–7.

[8]Plate 1, Lines 8–13.

[9]Excerpt from the appendix, specifically lines 34–46.

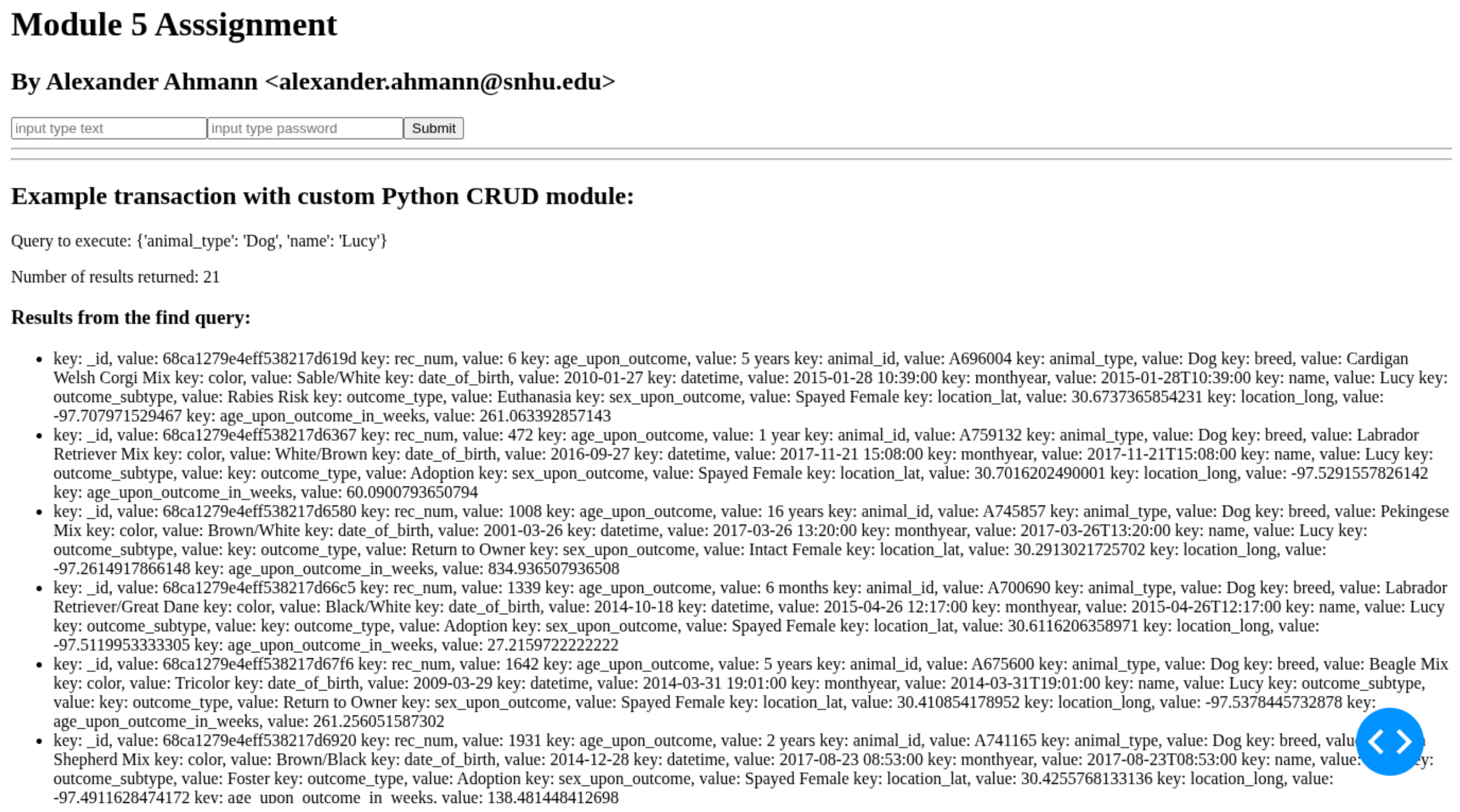
Plate 2: Worked solution results printout

```
1. html.Hr(),
2. html.H2("Example transaction with
      custom Python CRUD module:"),
3. html.P("Query to execute: {0}".format(
4.    lucy_query
5. )),
6. html.P("Number of results returned: {0}".format(
7.     len(execute_lucy_query)
8. )),
9.  html.H3("Results from the find query:"),
10.   html.Ul(
11.     [html.Li(result) for result in query_results]
12.   )
13. ])
```

The important parts are lines 3–5, 6–8 and 9–12:

- Lines 3–5 renders out the query that is to be executed.
- Lines 6–8 renders out an integer of the number of BSON documents returned.
- Lines 9–12 renders out an *unordered list* of strings showing each of the `key/value` pairs from the results.

I then launched the Python Dash web application from the Jupyter Notebook. Figure 1 depicts the resulting web page after I have made the changes to the starter code.



Module 5 Asssignment

By Alexander Ahmann <alexander.ahmann@snhu.edu>

Example transaction with custom Python CRUD module:

Query to execute: {'animal_type': 'Dog', 'name': 'Lucy'}

Number of results returned: 21

Results from the find query:

- key: _id, value: 68ca1279e4eff538217d619d key: rec_num, value: 6 key: age_upon_outcome, value: 5 years key: animal_id, value: A696004 key: animal_type, value: Dog key: breed, value: Cardigan Welsh Corgi Mix key: color, value: Sable/White key: date_of_birth, value: 2010-01-27 key: datetime, value: 2015-01-28 10:39:00 key: monthyear, value: 2015-01-28T10:39:00 key: name, value: Lucy key: outcome_subtype, value: Rabies Risk key: outcome_type, value: Euthanasia key: sex_upon_outcome, value: Spayed Female key: location_lat, value: 30.6737365854231 key: location_long, value: -97.707971529467 key: age_upon_outcome_in_weeks, value: 261.063392857143
- key: _id, value: 68ca1279e4eff538217d6367 key: rec_num, value: 472 key: age_upon_outcome, value: 1 year key: animal_id, value: A759132 key: animal_type, value: Dog key: breed, value: Labrador Retriever Mix key: color, value: White/Brown key: date_of_birth, value: 2016-09-27 key: datetime, value: 2017-11-21 15:08:00 key: monthyear, value: 2017-11-21T15:08:00 key: name, value: Lucy key: outcome_subtype, value: key: outcome_type, value: Adoption key: sex_upon_outcome, value: Spayed Female key: location_lat, value: 30.7016202490001 key: location_long, value: -97.5291557826142 key: age_upon_outcome_in_weeks, value: 60.0900793650794
- key: _id, value: 68ca1279e4eff538217d6580 key: rec_num, value: 1008 key: age_upon_outcome, value: 16 years key: animal_id, value: A745857 key: animal_type, value: Dog key: breed, value: Pekingese Mix key: color, value: Brown/White key: date_of_birth, value: 2001-03-26 key: datetime, value: 2017-03-26 13:20:00 key: monthyear, value: 2017-03-26T13:20:00 key: name, value: Lucy key: outcome_subtype, value: key: outcome_type, value: Return to Owner key: sex_upon_outcome, value: Intact Female key: location_lat, value: 30.2913021725702 key: location_long, value: -97.2614917866148 key: age_upon_outcome_in_weeks, value: 834.936507936508
- key: _id, value: 68ca1279e4eff538217d66c5 key: rec_num, value: 1339 key: age_upon_outcome, value: 6 months key: animal_id, value: A700690 key: animal_type, value: Dog key: breed, value: Labrador Retriever/Great Dane key: color, value: Black/White key: date_of_birth, value: 2014-10-18 key: datetime, value: 2015-04-26 12:17:00 key: monthyear, value: 2015-04-26T12:17:00 key: name, value: Lucy key: outcome_subtype, value: key: outcome_type, value: Adoption key: sex_upon_outcome, value: Spayed Female key: location_lat, value: 30.6116206358971 key: location_long, value: -97.5119953333305 key: age_upon_outcome_in_weeks, value: 27.2159722222222
- key: _id, value: 68ca1279e4eff538217d67f6 key: rec_num, value: 1642 key: age_upon_outcome, value: 5 years key: animal_id, value: A675600 key: animal_type, value: Dog key: breed, value: Beagle Mix key: color, value: Tricolor key: date_of_birth, value: 2009-03-29 key: datetime, value: 2014-03-31 19:01:00 key: monthyear, value: 2014-03-31T19:01:00 key: name, value: Lucy key: outcome_subtype, value: key: outcome_type, value: Return to Owner key: sex_upon_outcome, value: Spayed Female key: location_lat, value: 30.410854178952 key: location_long, value: -97.5378445732878 key: age_upon_outcome_in_weeks, value: 261.256051587302
- key: _id, value: 68ca1279e4eff538217d6920 key: rec_num, value: 1931 key: age_upon_outcome, value: 2 years key: animal_id, value: A741165 key: animal_type, value: Dog key: breed, valu [illegible] Shepherd Mix key: color, value: Brown/Black key: date_of_birth, value: 2014-12-28 key: datetime, value: 2017-08-23 08:53:00 key: monthyear, value: 2017-08-23T08:53:00 key: name, value: [illegible] ey: outcome_subtype, value: Foster key: outcome_type, value: Adoption key: sex_upon_outcome, value: Spayed Female key: location_lat, value: 30.4255768133136 key: location_long, value: -97.4911628474172 key: age_upon_outcome_in_weeks, value: 138.481448412698

Figure 1:

3 Summary

This assignment may not directly contribute to the second phase of the data application to be developed for the hypothetical *Grazioso Salvare*, but it does nonetheless serve as a proof-of-concept that can demonstrate the potential usefulness of our home-grown data modelling and visualization tool. Such stuff is typical in the commercial world, rapid prototyping and PoCs are used to demonstrate a somewhat crude solution to a given problem, and such a solution is to later be refined into a more elegant one.

References

CS-340 (n.d.). *Module Five Assignment Guidelines and Rubric.*

A Python Dash Application's Source Code

Note that I have removed many comments from the full source code, for the sake of succinctness. The full code listing is shown in the `ModuleFiveAssignment.ipynb` Jupyter Notebook.

```
from jupyter_dash import JupyterDash
from dash import dcc, html
from dash.dependencies import Input, Output
JupyterDash.infer_jupyter_proxy_config()
from CRUD_Python_Module import *
import urllib.parse

mongo_instance = AnimalShelter()
lucy_query = {
    "animal_type" :"Dog",
    "name" :"Lucy"
}

execute_lucy_query = mongo_instance.read(lucy_query)
query_results = []
for results in execute_lucy_query:
    resultant = ""
    for k, v in results.items():
        resultant += "key: {0}, value: {1} ".format(k, v)
```

```
  query_results.append(resultant)

app = JupyterDash(__name__)
app.layout = html.Div([
  html.H1("Module 5 Asssignment"),
  html.H2("By Alexander Ahmann <alexander.ahmann@snhu.edu>"),
  dcc.Input(
      id="input_user".format("text"),
      type="text",
      placeholder="input type {}".format("text")),
  dcc.Input(
      id="input_passwd".format("password"),
      type="password",
      placeholder="input type {}".format("password")),
  html.Button('Submit', id='submit-val', n_clicks=0),
  html.Hr(),
  html.Div(id="query-out", style={'whiteSpace': 'pre-line'}),

  html.Hr(),
  html.H2("Example transaction with custom Python CRUD module:"),

  html.P("Query to execute: {0}".format(
    lucy_query
  )),

  html.P("Number of results returned: {0}".format(
    len(execute_lucy_query)
  )),

  html.H3("Results from the find query:"),
  html.Ul(
    [html.Li(result) for result in query_results]
  )
])

@app.callback(
  Output('query-out', 'children'),
  [Input('input_user', 'value'),
  Input('input_passwd', 'value'),
    Input(component_id='submit-val', component_property='n_clicks')]
)

def update_figure(inputUser,inputPass,n_clicks):
    if n_clicks > 0:
        username = urllib.parse.quote_plus(inputUser)
        password = urllib.parse.quote_plus(inputPass)

app.run_server()
```