Southern New Hampshire University

CS-340: Assignment 6-1: Module 6 Assignment

Prepared on: October 21, 2025
Prepared for: Prof. Jeff H. Sanford
Prepared by: Alexander Ahmann

Contents

1 Introduction

Following on the theme of developing a front-end web application to interact with the given backend MongoDB instance, I have used Python's Dash framework to develop a data-driven web application with two key features: a tabular dataset of documents in the `aac.animals` collection, and a geolocation chart that "points" to a specific location of an animal selected from one of the many rows in the aforementioned tabular chart, with a "tool tip" depicting various features of the animal as recorded in the MongoDB instance.

The result is a proof-of-concept that solves the problem given by the assignment's requirements (CS-340, n.d., Prompt)— which has the potential to act as a basis for a mature front-end application for the second project. Specific subsets of the final source code are discussed in §2 of this paper, and (mostly) complete versions of the application's source code is given in the appendix.

2 Problem Set & Solutions

2.1 Task 1: The implementation of a dynamic, interactive data table of MongoDB documents

> "Open the `ModuleSixMilestone.ipynb` file, which contains the starter code for the Grazioso Salvare dashboard. The Notebook file has been preloaded into your Codio environment; open it using the Jupyter Lab application. Be sure to review all the starter code provided. Pay special attention to the import commands and the comments describing what each code section does.
>
> [... snip ...]
>
> Update the code to *create an interactive data table* on the dashboard that shows an unfiltered view of the Austin Animal Center Outcomes data set. To populate the data onto your table, you will *use your previous CRUD Python module*, from Project One, to run a "retrieve all" query and import the data from MongoDB. This data retrieval will access the "model" portion of your MVC pattern: the MongoDB database. Be sure to hardcode in the username and password for the `aacuser` account.
>
> [... snip ...]
>
> Be sure to consider your client when creating the interactive data table. Consider optional features that will make the table easier to use, such as limiting the number of rows displayed, enabling pagination (advanced), enabling sorting, and so on. Review the Module Six resources on data tables to help you select and set up these features. Furthermore, you must enable single-row selection for the data table for the mapping callback to function properly. This property is represented as: `row_selectable = "single"`
> —slightly paraphrased from CS-340 (n.d., Prompt: §1 and §2)

Plate 1.1: Changes made to the "CRUD Class"' `__init__` constructor[a]

```
1.  def __init__(self, USER, PASS, HOST, PORT, DB, COL):
2.      if not isinstance(USER, str):
3.          raise
    Exception("'USER' should be a string type")
4.      elif not isinstance(PASS, str):
5.          raise
    Exception("'PASS' should be a string type")
6.      elif not isinstance(HOST, str):
7.          raise
    Exception("'HOST' should be a string type")
8.      elif not isinstance(PORT, int):
```

[a]Code listing adapted from the most recent version of `CRUD_Python_Module.py`'s `AnimalShelter()` class; also note that I have removed the comments and document string from the original listing for the sake of succinctness.

A worked solution to this involves making changes to the `CRUD_Python_Module.py` library— specifically changing the `AnimalShelter()`'s `__init__` constructor to take in configuration data as parameters from scripts that make use of it. Plate 1.1 depicts the refined constructor for the `AnimalShelter()` class.

In particlar, the `__init__` has been modified to take in six (6) parameters: a username to the instance `USER`, a password `PASS` for authenticating the user, the hostname of the MongoDB instance `HOST` to connect to, the port number `PORT` of the MongoDB instance's daemon, the database `DB` to use, and the specific collection `COL` to use.[1] To ensure that the proper data types are passed to the `__init__` constructor, a series of exceptions are raised if they are not of the proper type.[2]

[1]Plate 1.1, Line 1.

[2]Plate 1.1, Lines 2-13; this is done because of the nature of the Python language: it is a weakly typed language, which is good stuff for flexibility. But such flexibility can lead to problems if the function in question expects certain types when processing data.

Plate 1.1 (cont.)

```
9.          raise
    Exception("'PORT' should be an integer type")
10.     elif not isinstance(DB, str):
11.         raise
    Exception("'DB' should be a string type")
12.     elif not isinstance(COL, str):
13.         raise
    Exception("'COL' should be a string type")

14.     self.client = MongoClient
    ('mongodb://%s:%s@%s:%d' % (USER,PASS,HOST,PORT))
15.     self.database = self.client['%s' % (DB)]
16.     self.collection = self.database['%s' % (COL)]
```

After that, an instance of the `MongoClient` is created, which is used to faciliate a connection between the Dash web application and the MongoDB instance.[3] This modification allows for the Dash application to pass its own credentials as parameters to a new instance of `AnimalShelter()`, in accordance with the requirements described in CS-340 (n.d., Prompt §2). Plate 1.2 depicts the instantiation of an `AnimalShelter()` class in the Python Dash Web Application.

The Dash Web Application starts by importing the needed libraries, including the recently invented and refined Python CRUD module,[4] and setting variables as hardcoded parameters for the `AnimalShelter()` class.[5] Then, an instance of the `AnimalShelter()` is declared under the variable assignment of `shelter`,[6] and finally all the documents in the target collection of `animals` is loaded into a Pandas dataframe.[7]

[3]Plate 1.1, Lines 14–16.
[4]Plate 1.2, Line 1.
[5]Plate 1.2, Lines 2-7.
[6]Plate 1.2, Lines 8–15.
[7]Plate 1.2, Line 16.

Plate 1.2: Initiating a connection to MongoDB[a]

```
from CRUD_Python_Module import *

username = "aacuser"
password = "WingsofRedemption"
hostname = "localhost"
port = 27017
database = "aac"
collections = "animals"

shelter = AnimalShelter(
    username,
    password,
    hostname,
    port,
    database,
    collections
)

df = pd.DataFrame.from_records(shelter.read({}))
```

[a]Code listing adapted from `ModuleSixMilestone.ipynb`

After initiating a connection between the Dash web application and MongoDB instance, I proceeded to make slight refinements to the given front-end stuff defined in the `app.layout`. In particular, the given starter code had a `DashTable` object for rendering an interactvie tabular data set. Plate 1.3 shows a listing of the relevant bit of code that I made changes to.

The stuff that I added is shown in lines 7–10: as per the requirements, I set the `row_selectable` to `= "single"` and the `selected_rows` to `= [0]`. Furthermore, with the assistance of previous versions of this project and my imagination with visual design, I configured the `page_size` to `= 5` so that the `DataTable` will limit its output to only five (5) rows.[8] I also enabled sorting on the columns with the option `sort_action = "native"`, allowing the user to order the tabular data set on the basis of whatever column they choose.

[8]Plate 1.3, Line 9.

Plate 1.3: Rendering an interactive table[a]

```
dash_table.DataTable(
   id='datatable-id',
   columns=[
      {"name": i, "id": i, "deletable": False,
  "selectable": True}
         for i in df.columns],
    data=df.to_dict('records'),

    row_selectable = "single",
    selected_rows = [0],
    page_size = 5,
    sort_action = "native",
),
```

[a]Code listing adapted from `ModuleSixMilestone.ipynb`

This work allows for an interactive table that allows for users to select specific rows that can be used as parameters for another widget. In this case, the "geolocation chart," which is the software engineering problem discussed in the next subsection.

2.2 Task 2: The implementation of a "geolocation chart" charting the geographical distribution of "animals"

> The following subtasks are in regards to the higher task of "[adding] a geolocation chart that displays data from the interactive data table to your existing dashboard,"
>
> - "You are being given the function that sets up accessing the data for the geolocation chart and calls the Leaflet function `update_map`[9]
> - You will need to structure this function into your dashboard code by adding the correct statements to the layout. These statements ensure your layout has a place for the geolocation chart; see the following code listing:

[9]Finalized version shown in plate 2.1, and was derived from CS-340 (n.d., Prompt: §3.1).

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

- You will also need to add the correct callback routines to the geolocation chart. These will look similar to the callback routines used for user authentication and your data table; see the following code listing:

```
@app.callback(
  Output('datatable-id',
    'style_data_conditional'),
  [Input('datatable-id',
    'selected_columns')]
)
```

Furthermore, the Leaflet geolocation chart will show the row selected in the data table. To prevent issues with not selecting an initial row, you can set the selected row as a parameter to the `DataTable` method by adding the parameter: `selected_rows[0],` to the `DataTable` constructor, thereby selecting first row of the data table by default. As you select separate rows in the `DataTable`, the map should update automatically."— slightly paraphrased from CS-340 (n.d., Prompt: §3)

The addition of the `html.Div` and `@app.callback` is fairly trivial, so I will not discuss that here. But, much of the relevant functionality of the geolocation map is implemented in the `update_map()` function, with its finished code listing depicted in plate 2.1.

Plate 2.1: Worked solution to geolocation chart[a]

```
1.   @app.callback(
2.      Output('map-id', "children"),
3.      [Input('datatable-id', "derived_virtual_data"),
4.       Input('datatable-id',
    "derived_virtual_selected_rows")],
5.   )
6.   def update_map(viewData, index):
7.      dff = pd.DataFrame.from_dict(viewData)
8.      if index is None:
9.          row = 0
10.     else:
11.         row = index[0]
12.    return [
13.        dl.Map(style={'width': '1000px',
    'height': '500px'},
14.           center=[30.75, -97.48], zoom=10, children=[
15.           dl.TileLayer(id="base-layer-id"),
16.        dl.Marker(position=[dff.iloc[row,13],
    dff.iloc[row,14]],
17.            children=[
```

[a]Code listing adapted from CS-340 (n.d., Prompt: §3.1), also note that I have removed the comments from the original listing for the sake of succinctness.

The `update_map()` takes in the selected row and stores its column values into the local `dff` variable.[10] Then, values are worked out and returned as a Python list.[11] In particular, I made changes to the `dl.Popup` variable: adding lines to include data for the chosen animal's ID,[12] the chosen animal's name,[13] and their date of birth.[14] This is then returned, and the interactive map is updated in real time as the end-user selected an arbitrary row.

[10]Plate 2.1, Line 7.
[11]Plate 2.1, Lines 12–40.
[12]Plate 2.1, Lines 22–24.
[13]Lines 26–28.
[14]Lines 34–36.

Plate 2.1 (cont.)

```
            dl.Tooltip("{0} (Animal ID: {1})".format(
                dff.iloc[row,4], dff.iloc[row,2])
            ),
            dl.Popup([
                html.B("Animal ID: {0}".format(
                    dff.iloc[row,2])
                ),
                html.Br(),
                html.B("Name: {0}".format(
                    dff.iloc[row,9])
                ),
                html.Br(),
                html.B("Breed: {0}".format(
                    dff.iloc[row,4]
                )),
                html.Br(),
                html.B("DOB: {0}".format(
                    dff.iloc[row,6]
                ))
            ])
        ])
        ])
    ]
```

2.3 Task 3: Basic testing and demonstration of the Python Dash web application

"Finally, run the IPYNB file and take a screenshot of your dashboard as proof of this execution. Your screenshot should include 1) the interactive data table populated with the Austin Animal Center Outcomes data from MongoDB and 2) your geolocation chart showing the location of the first dog in the table. Additionally, your unique identifier[15] should also appear in the screenshot."— CS-340 (n.d., Prompt: §4)

[15]Created in the Module Five assignment (Ahmann, 2025).

After running the Dash web application, I was presented with a web interface with a tabular dataset on the top-half, and an interactive geolocation chart on the lesser-half. Figure 1 depicts the Dash web application. An error is also shown on the top-right corner, which I will shortly elaborate more on.

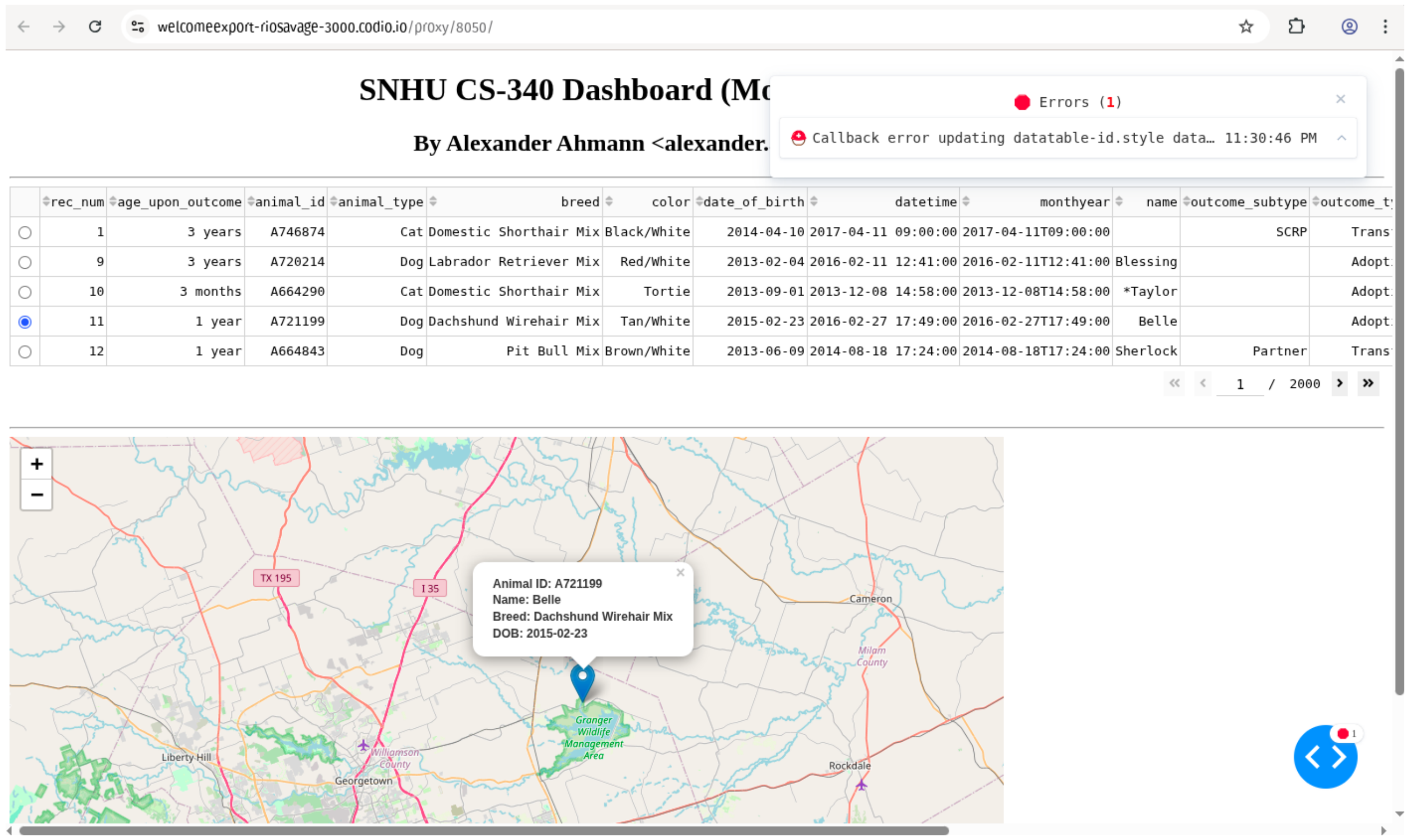
3 Discussion

This Dash web application is the result of a given starter code for a front-end interface in the form of a Jupyter Notebook, and a refined Python CRUD module by which to interface with a MongoDB instance. Much progress has been made with the development of the Python CRUD module, but the current Dash application simply serves as, at the moment, a "proof-of-concept." Further testing for both of them is called for.

Indeed, there is a runtime error that was raised during the execution of the web application,[16] and at the moment I am not quite sure what is causing it. Further progress to the development of a mature Dash front-end component must include the identification of this error's causal agents, and a worked solution to fix the error.

Furthermore, it should be noted that I relied on the work of another *CS-340* student (Udeh, n.d.). I referenced a screenshot of his version of the Dash web application, and used that as an aid to visualize what the resulting front-end output should look like. The specific code snippets that I inserted or changed in the Dash web application was mostly aided by consulting the online documentation for the Dash framework.

[16]See the top-right corner of figure 1.


welcomeexport-riosavage-3000.codio.io/proxy/8050/

SNHU CS-340 Dashboard (Mc

By Alexander Ahmann <alexander.

Errors (1)

Callback error updating datatable-id.style data… 11:30:46 PM

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_t
○	1	3 years	A746874	Cat	Domestic Shorthair Mix	Black/White	2014-04-10	2017-04-11 09:00:00	2017-04-11T09:00:00		SCRP	Trans
○	9	3 years	A720214	Dog	Labrador Retriever Mix	Red/White	2013-02-04	2016-02-11 12:41:00	2016-02-11T12:41:00	Blessing		Adopt
○	10	3 months	A664290	Cat	Domestic Shorthair Mix	Tortie	2013-09-01	2013-12-08 14:58:00	2013-12-08T14:58:00	*Taylor		Adopt
●	11	1 year	A721199	Dog	Dachshund Wirehair Mix	Tan/White	2015-02-23	2016-02-27 17:49:00	2016-02-27T17:49:00	Belle		Adopt
○	12	1 year	A664843	Dog	Pit Bull Mix	Brown/White	2013-06-09	2014-08-18 17:24:00	2014-08-18T17:24:00	Sherlock	Partner	Trans

1 / 2000

Animal ID: A721199
Name: Belle
Breed: Dachshund Wirehair Mix
DOB: 2015-02-23

Figure 1: The Dash web application in action.

4 Summary

- The Python CRUD module has been modified to allow for external scripts that are importing it to supply their own credentials, database and host configuration into an instance of its `AnimalShelter()` class.
- A web application has been written with Python's *Dash* framework, which acts as a front-end for retrieving and outputting data from a MongoDB instance.
- The formats that were used to view "animal shelter" data was a tabular dataset, and a geolocation chart.
- While the new Python CRUD module is showing signs of maturity, the Python Dash web application still has software bugs— namely a runtime error with an antecedent that I have not yet worked out.
- Further software assessment and testing is needed to be done against both the Python CRUD module, and the Dash web application.

4.1 Acknowledgements

Victor Udeh for their version of the *SNHU CS-340 Dash Web Application*, in where I referenced a screenshot of their Dash web application running to give myself a better idea of how to organize the resulting Dash web application. And the Python Dash framework's very nifty online documentation,[17] which I consistently relied on when building upon the starter code of the Python Dash web application.

References

Ahmann, A. (2025). *CS-340: Assignment 5-2: Module 5 Assignment.*

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

Udeh, V. (n.d.). *Snhu-CS340 / module 6.* Retrieved on Oct. 12, 2025 from: https://github.com/vhicktour/Snhu-CS340/tree/main/module%206

[17]https://dash.plotly.com/reference

A Worked codebase for Dash web application

The following are code listings for the Python CRUD Module and the Dash Application that uses it to display data. Note that I stripped most of the comments and document strings for the sake of succinctness.

A.1 CRUD_Python_Module.py

```
 from pymongo import MongoClient
 from bson.objectid import ObjectId

 class AnimalShelter(object):
     def __init__(self, USER, PASS, HOST, PORT, DB, COL):
         if not isinstance(USER, str):
             raise Exception("’USER’ should be
a string type")
         elif not isinstance(PASS, str):
             raise Exception("’PASS’ should be
a string type")
         elif not isinstance(HOST, str):
             raise Exception("’HOST’ should be
a string type")
         elif not isinstance(PORT, int):
             raise Exception("’PORT’ should be
an integer type")
         elif not isinstance(DB, str):
             raise Exception("’DB’ should be
a string type")
         elif not isinstance(COL, str):
             raise Exception("’COL’ should be
a string type")

         self.client = MongoClient
(’mongodb://%s:%s@%s:%d’ % (USER,PASS,HOST,PORT))
         self.database = self.client[’%s’ % (DB)]
         self.collection = self.database[’%s’ % (COL)]

   def create(self, data: dict) -> bool:
       try:
           if data is None:
               raise Exception
```

```
("The data type should not be 'None'")
           elif not isinstance(data, dict):
               raise Exception
("The data type should be a dictionary")
           else:
               obj_id = ObjectId()
               document = {"_id":obj_id}
               document.update(data)
               self.database.animals.insert_one(document)
               return True
       except Exception as e:
           print("Exception raised: {0}".format(e))
           return False

       return False

   def read(self, query: dict) -> list:
       results = []
       try:
           if not isinstance(query, dict):
               raise Exception
("'query' should be a dictionary")
           else:
            query_results = self.collection.find(query)
             results.extend([doc for doc in query_results])
       except Exception as e:
           print("Exception raised: {0}".format(e))

       return results

   def update(self, query: dict, data: dict) -> int:
       total_affected = 0
       try:
           if not isinstance(query, dict):
               raise Exception
("parameter 'query' should be a dictionary.")
           elif not isinstance(data, dict):
               raise Exception
("parameter 'data' should be a dictionary.")
           result = self.collection.update_many(
               query, { "$set": data }
           )
           total_affected += result.modified_count
       except Exception as e:
           print("Exception raised: {0}".format(e))
       return total_affected

   def delete(self, query: dict) -> int:
       total_affected = 0

       try:
           if not isinstance(query, dict):
               raise Exception("parameter ’query’ should be a dictionary.")
           result = self.collection.delete_many(query)
           total_affected += result.deleted_count
       except Exception as e:
           print("Exception raised: {0}".format(e))

       return total_affected
```

A.2 ModuleSixMilestone.ipynb

```
from jupyter_dash import JupyterDash

import dash_leaflet as dl
from dash import dcc, html
import plotly.express as px
from dash import dash_table
from dash.dependencies import Input, Output
JupyterDash.infer_jupyter_proxy_config()

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from CRUD_Python_Module import *

username = "aacuser"
password = "WingsofRedemption"
hostname = "localhost"
port = 27017
```

```
database = "aac"
collections = "animals"

shelter = AnimalShelter(
    username,
    password,
    hostname,
    port,
    database,
    collections
)

df = pd.DataFrame.from_records(shelter.read({}))
df.drop(columns=['_id'], inplace=True)

app = JupyterDash('SimpleExample')

app.layout = html.Div([
   html.Div(id='hidden-div', style={'display':'none'}),
   html.Center(html.B(html.H1('SNHU CS-340 Dashboard (Module 6 Assignment)'))),
   html.Center(html.H2("By Alexander Ahmann <alexander.ahmann@snhu.edu>")),
   html.Hr(),
   dash_table.DataTable(
       id='datatable-id',
       columns=[
           {"name": i, "id": i, "deletable": False, "selectable": True}
           for i in df.columns],
       data=df.to_dict('records'),

       row_selectable = "single",
       selected_rows = [0],
       page_size = 5,
       sort_action = "native",
   ),
   html.Br(),
   html.Hr(),

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

])

@app.callback(
    Output(’datatable-id’, ’style_data_conditional’),
    [Input(’datatable-id’, ’selected_columns’)]
)
def update_styles(selected_columns):
   return [{
       ’if’: { ’column_id’: i },
       ’background_color’: ’#D2F3FF’
   } for i in selected_columns]

@app.callback(
   Output(’map-id’, "children"),
   [Input(’datatable-id’, "derived_virtual_data"),
    Input(’datatable-id’, "derived_virtual_selected_rows")],
)
def update_map(viewData, index):
   dff = pd.DataFrame.from_dict(viewData)

   if index is None:
      row = 0
   else:
       row = index[0]
   return [
       dl.Map(style={’width’: ’1000px’, ’height’: ’500px’},
          center=[30.75, -97.48], zoom=10, children=[
          dl.TileLayer(id="base-layer-id"),
       dl.Marker(position=[dff.iloc[row,13],dff.iloc[row,14]],
           children=[
           dl.Tooltip("{0} (Animal ID: {1})".format(
               dff.iloc[row,4], dff.iloc[row,2])
           ),
           dl.Popup([
               #html.H2(),
               html.B("Animal ID: {0}".format(
                   dff.iloc[row,2])
               ),
               html.Br(),
               html.B("Name: {0}".format(
                   dff.iloc[row,9])
               ),
               html.Br(),
               html.B("Breed: {0}".format(
                   dff.iloc[row,4]
               )),
               html.Br(),
               html.B("DOB: {0}".format(
                   dff.iloc[row,6]
               ))
           ])
        ])
        ])
    ]

app.run_server()
```