# Project 2 README

## About the Project/Project Title

*This project is the CRUD (Create, Read, Update, and Delete) module for interacting with mongodb. It allows for reading from the mongodb, and rendering data in a table and geolocation map.*

## Motivation

*This project was/is being created to interact with noSQL database mongodb. Mongodb is more effient than SQL in that it provides horizontal sharding, allowing for sub documents within a document. Where SQL is restricted to columns and schemas, noSQL lets the user add column-like attributes within a field.*

*This CRUD module exists to allow a user to add documents through a GUI.*

## Getting Started

*After creating the python module, importing MongoClient and ObjectId are essential. The \_\_init\_\_ method must be created to initialize the MongoDB. Passing the username, password, port number, and AuthSource are essential; as well.*

*After doing so, each of the CRUD functions should include their respective MongoDB shell commands; ie. “self.database.animals.insert(data)” for Create. And so forth for Read, Update, and Delete.*

## Installation

*Requires installation of:*

*Python*

*MongoDB*

*Jupyter Notebook*

*RStudio (to run tests)*

## Usage

### Code Example

*This python module initializes MongoDB, signs in as user (test module uses aacuser), and conducts CRUD methods (Update, and Delete coming soon)*

*from pymongo import MongoClient*

*from bson.objectid import ObjectId*

*class AnimalShelter1(object):*

*""" CRUD operations for Animal collection in MongoDB """*

*def \_\_init\_\_(self,username,password):*

*# Initializing the MongoClient. This helps to*

*# access the MongoDB databases and collections.*

*self.client = MongoClient('mongodb://%s:%s@localhost:37442' % (username, password), authSource="AAC")*

*self.database = self.client['AAC']*

*# Complete theis create method to implement the C in CRUD.*

*def create(self,data):*

*if data is not None:*

*self.database.animals.insert(data) # data should be dictionary*

*data = data*

*else:*

*raise Exception("Nothing to save, because data parameter is empty")*

*data = false*

*return data*

*# Complete theis create method to implement the R in CRUD.*

*def read(self,data):*

*if data is not None:*

*self.database.animals.find(data) # data should be dictionary*

*data = data*

*else:*

*raise Exception("Nothing to save, because data parameter is empty")*

*data = false*

*return data*

*# Complete theis create method to implement the U in CRUD.*

*def update(self,data,dataUpdate):*

*if data is not None:*

*self.database.animals.update(data,dataUpdate) # data should be dictionary*

*data = data*

*else:*

*raise Exception("Nothing to save, because data parameter is empty")*

*data = false*

*return data*

*# Complete theis create method to implement the D in CRUD.*

*def delete(self,data):*

*if data is not None:*

*self.database.animals.remove(data) # data should be dictionary*

*data = data*

*else:*

*raise Exception("Nothing to save, because data parameter is empty")*

*data = false*

*return data*

**Client side Dashboard Code**

*from jupyter\_plotly\_dash import JupyterDash*

*import dash*

*import dash\_leaflet as dl*

*import dash\_core\_components as dcc*

*import dash\_html\_components as html*

*import plotly.express as px*

*import dash\_table as dt*

*from dash.dependencies import Input, Output, State*

*import os*

*import numpy as np*

*import pandas as pd*

*from pymongo import MongoClient*

*from bson.json\_util import dumps*

*import base64*

*# Importing AnimalShelter class from my module aac\_crud*

*from AnimalShelter1 import AnimalShelter1*

*###########################*

*# Data Manipulation / Model*

*###########################*

*username = "aacuser"*

*password = "aacuser"*

*shelter = AnimalShelter1(username, password)*

*# class read method must support return of cursor object*

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

*#########################*

*# Dashboard Layout / View*

*#########################*

*app = JupyterDash('SimpleExample')*

*image\_filename = 'Grazioso Salvare Logo.png' # replace with your own image*

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

*app.layout = html.Div([*

*html.Div(id='hidden-div', style={'display':'none'}),*

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

*html.Center(html.B(html.H1('Charles Merrick Project 2 Dashboard'))),*

*html.Hr(),*

*# html.Img(id='customer-image',src='data:image/png;base64,{}'.format(encoded\_image.decode()),alt='customer image'),*

*html.Div(*

*dcc.RadioItems(*

*id='filter-type',*

*options=[*

*{'label': 'Water Rescue', 'value': 'water'},*

*{'label': 'Mountain/Wilderness Rescue', 'value': 'mount'},*

*{'label': 'Disaster Rescue and Individual Tracking', 'value': 'disaster'},*

*{'label': 'Reset', 'value': 'reset'}*

*],*

*value='reset'*

*)*

*),*

*html.Hr(),*

*dt.DataTable(*

*id='datatable-id',*

*columns=[*

*{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns*

*],*

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

*#FIXME: Set up the features for your interactive data table to make it user-friendly for your client*

*editable=False,*

*sort\_action="native",*

*sort\_mode="multi",*

*column\_selectable=True,*

*row\_selectable=True,*

*row\_deletable=False,*

*selected\_columns=[],*

*selected\_rows=[0],*

*page\_action="native",*

*page\_current= 0,*

*page\_size= 10,*

*),*

*html.Br(),*

*html.Hr(),*

*#This sets up the dashboard so that your chart and your geolocation chart are side-by-side*

*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',*

*)*

*])*

*])*

*#############################################*

*# Interaction Between Components / Controller*

*#############################################*

*# Radio button filters*

*@app.callback([Output('datatable-id','data'),*

*Output('datatable-id','columns')],*

*[Input('filter-type', 'value')])*

*def update\_dashboard(filter\_type):*

*# Water Rescue filter*

*if filter\_type == 'water':*

*df = pd.DataFrame.from\_records(shelter.read({*

*"animal\_type": "Dog",*

*"breed": {"$in": ["Labrador Retriever Mix","Chesapeake Bay Retriever", "Newfoundland"*

*]},*

*"sex\_upon\_outcome": "Intact Female",*

*"age\_upon\_outcome\_in\_weeks": {"$gte":26.0, "$lte":156.0}*

*}))*

*# Mountain or Wilderness Rescue filter*

*elif filter\_type == 'mount':*

*df = pd.DataFrame.from\_records(shelter.read({*

*"animal\_type": "Dog",*

*"breed": {"$in": ["German Shepard","Alaskan Malamute","Old English Sheepdog",*

*"Siberian Husky", "Rottweiler"*

*]},*

*"sex\_upon\_outcome": "Intact Male",*

*"age\_upon\_outcome\_in\_weeks": {"$gte":26.0, "$lte":156.0}*

*}))*

*# Disaster Rescue or Individual Tracking filter*

*elif filter\_type == 'disaster':*

*df = pd.DataFrame.from\_records(shelter.read({*

*"animal\_type": "Dog",*

*"breed": {"$in": ["Doberman Pinscher","German Shepard","Golden Retriever",*

*"Bloodhound","Rottweiler"*

*]},*

*"sex\_upon\_outcome": "Intact Male",*

*"age\_upon\_outcome\_in\_weeks": {"$gte":20.0, "$lte":300.0}*

*}))*

*# Reset to no filter*

*else:*

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

*columns=[{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns]*

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

*return (data,columns)*

*# Change background color of selected columns*

*@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]*

*# Chart*

*@app.callback(*

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

*[Input('datatable-id', "derived\_viewport\_data"),*

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

*])*

*def update\_graphs(viewData):*

*###FIX ME ####*

*# add code for chart of your choice (e.g. pie chart) #*

*dff = pd.DataFrame.from\_dict(viewData)*

*return [*

*dcc.Graph(*

*figure = px.histogram(dff, x='breed')*

*)*

*]*

*# Map*

*@app.callback(*

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

*[Input('datatable-id', "derived\_viewport\_data"),*

*Input('datatable-id', 'derived\_viewport\_selected\_rows')])*

*def update\_map(viewData, row\_ids):*

*dff = pd.DataFrame.from\_dict(viewData)*

*# Sets first num in row\_ids array to the last selected row ID*

*row\_ids[0] = row\_ids[len(row\_ids) - 1]*

*return [*

*# row\_ids[0] gets last selected row from DataTable and parameters 13 and 14 are long and lat of rows*

*dl.Map(style={'width': '1000px', 'height': '500px'}, center=[dff.iloc[row\_ids[0],13],dff.iloc[row\_ids[0],14]], zoom=10, children=[*

*dl.TileLayer(id="base-layer-id"),*

*# Marker with tool tip and popup*

*dl.Marker(position=[dff.iloc[row\_ids[0],13],dff.iloc[row\_ids[0],14]], children=[*

*# parameter num 4 gives the breed of animal*

*dl.Tooltip(dff.iloc[row\_ids[0],4]),*

*dl.Popup([*

*html.H1("Animal Name"),*

*# parameter 9 gives animal's name*

*html.P(dff.iloc[row\_ids[0],9])*

*])*

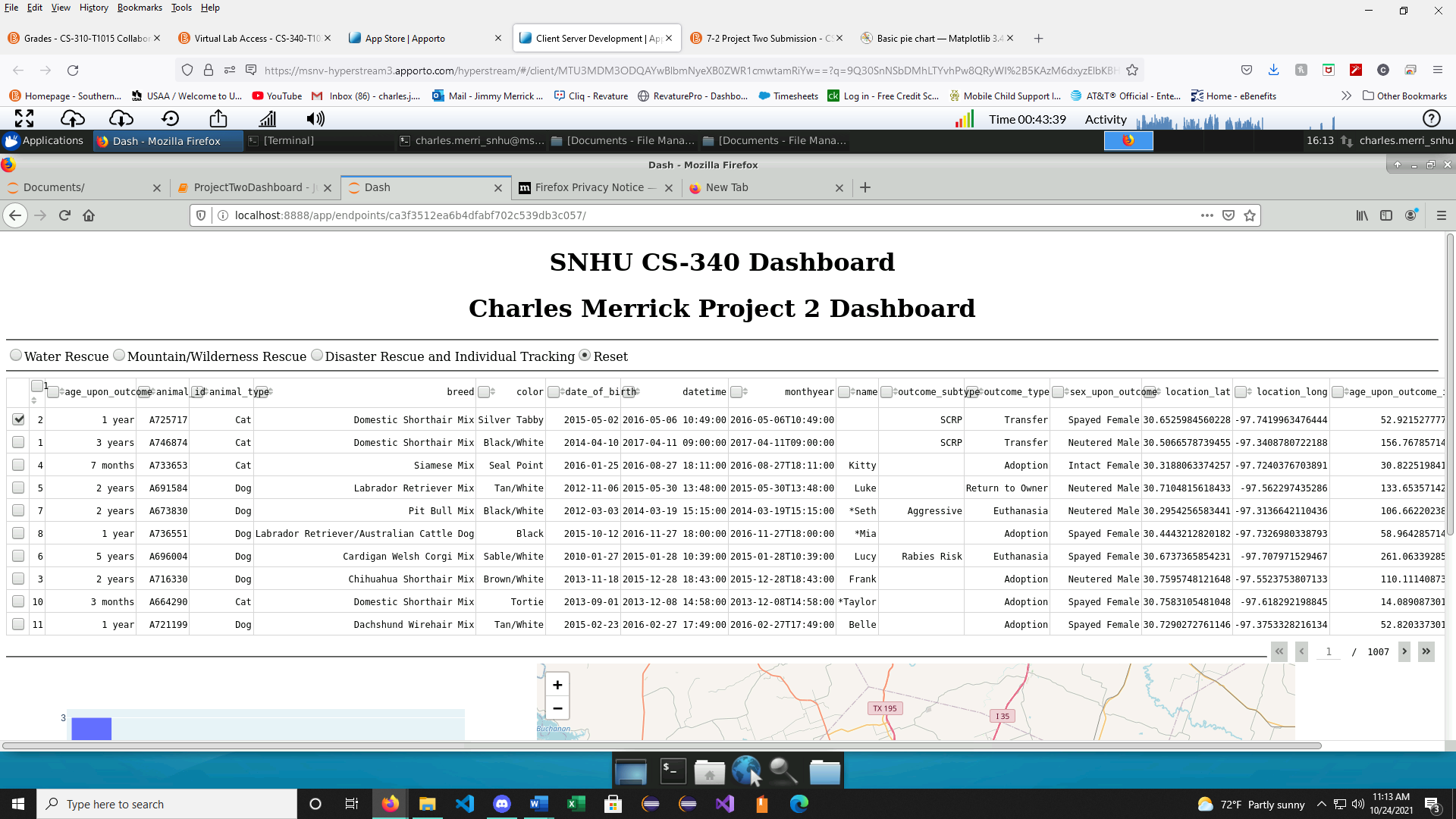
*])*

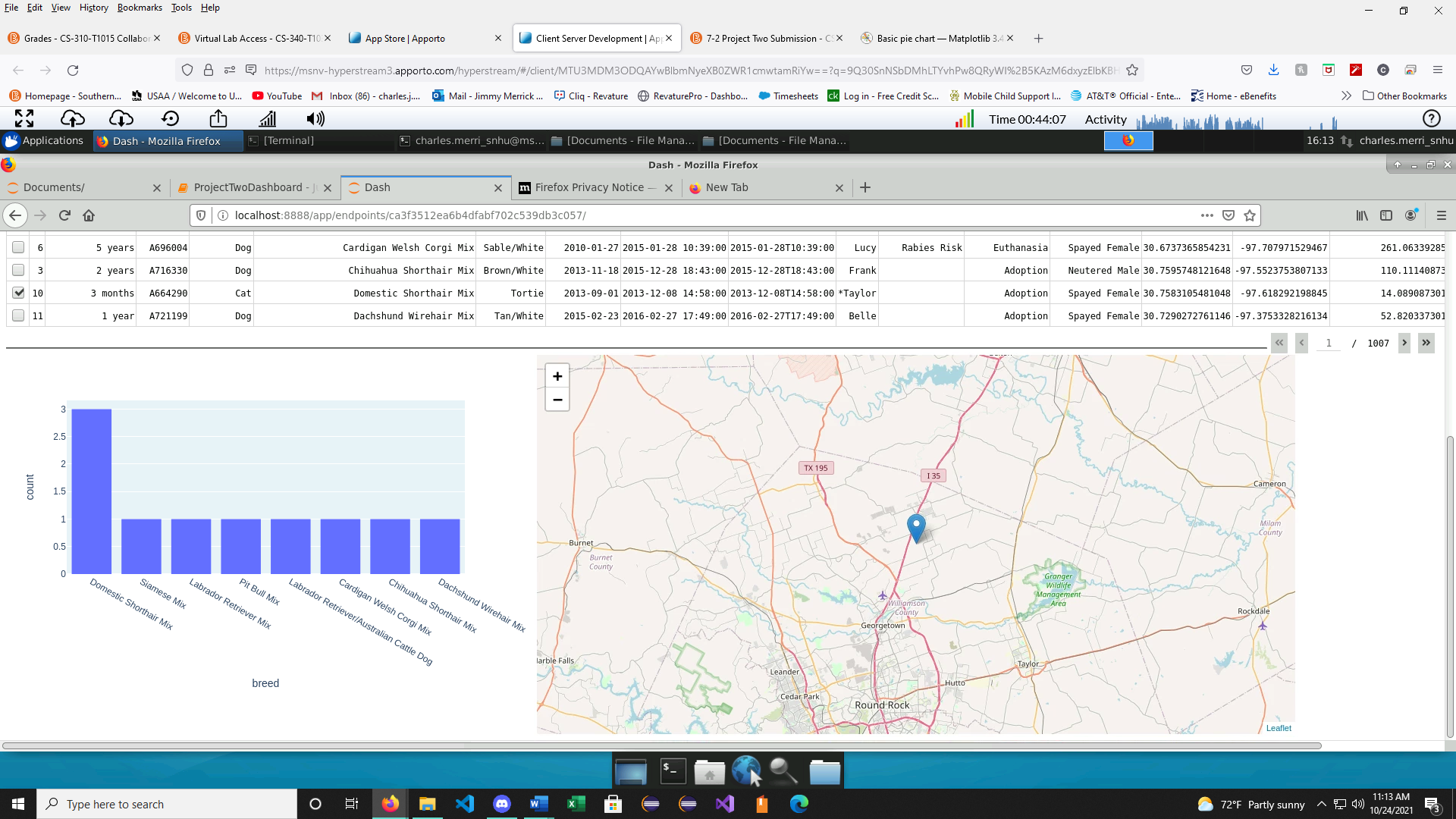
*])*

*]*

*app*

### Screenshots





## Roadmap/Features (Optional)

*The client-side module allows for rendering a data table from MongoDB, a histogram chart of breeds of dogs, and a geolocation map of where the dog is located when selected with a radio button.*

## Contact

Charles “Jimmy” Merrick