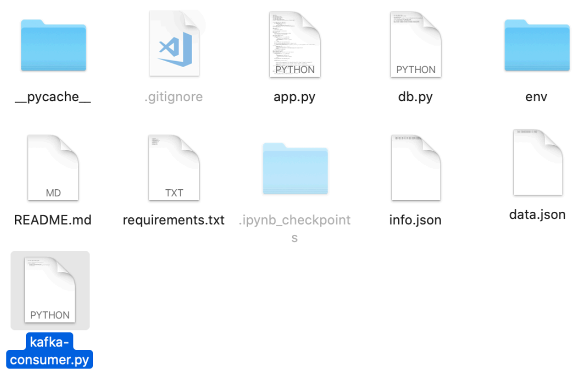
I worked on Database Management throughout the quarter. This is a module between data manipulation and model training. The idea of the module is to take the pre-processed data and store them in mongoDB with appropriate schema and to provide support for the CRUD operations. There are two main parts for the data, user data and actual model data. One of the important parts is to create a mapping between user and the model data he created.

To the job interviewer:

The project involves an about-10 people team developing a time-series data model training and predicting application. I took on the database management part. The task was to manage the user information and data each user trained. I decided to create a Flask Rest API by using mongoDB with Flask. The API contains four function calls which create entries for user objects and data objects and support queries to find the objects. More specifically for each user, there is a field “dataid” which creates a mapping between the user and the data one owns. The API is further enhanced by importing Apache Kafka. There is one topic created called ‘msg_collection’. When the users and their data are registered, you can subscribe to this topic and see messages regarding relevant information. In all, this project experience not only allows me to familiarize with one of the most popular NoSQL database(mongoDB) and its CRUD operations, but also lends me a unique perspective regarding micro web framework and Apache Kafka.

How to use:

First this is the layout of my project:



Here db.py contains the connection string to connect to database and points to the databases and collections accordingly; app.py is where the rest api gets implemented; kafka-consumer.py consumes the data and basically dumps out the messages;

Before we start on the project:
1 make sure we have kafka installed on our end (do ‘brew install kafka’ and ‘brew install zookeeper’ if necessary)

2 make sure we have flask installed on our end (do ‘pip3 install flask-pymongo’ if necessary)

3 make sure we have the mongodb on our end (‘brew tap mongodb/brew’ and ‘brew install mongodb-community@4.2’

4 make sure you add your current IP to the whitelist on mongoDB Atlas ‘Network Access’ Page

5 make sure you have postman application installed

Here are the steps to kick off the project:

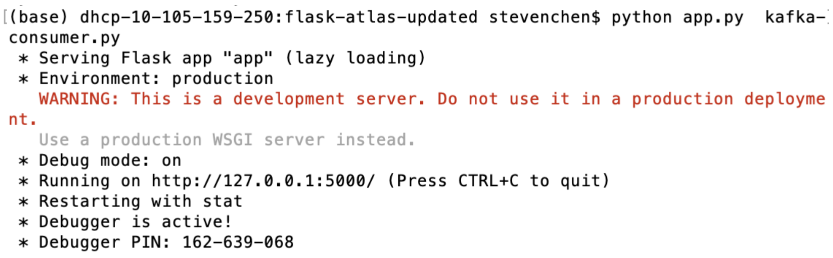
Step1: fire up zookeeper; the command is:
zookeeper-server-start /usr/local/etc/kafka/zookeeper.properties

Step2: In a new terminal window, fire up kafka; the command is:
kafka-server-start /usr/local/etc/kafka/server.properties

Step3: In a new terminal window, kick off our flask application and kafka consumer; the command is:
python app.py kafka-consumer.py

Then we take a detour and connect to mongoDB Atlas and prepare for monitoring changes in databases; here I do everything in a database called "flask-atlas-updated"(I put my personal information in CONNECTION_STRING and that part should be substituted for whichever your account is).

Then in terminal you should see this if you flask is running alright:

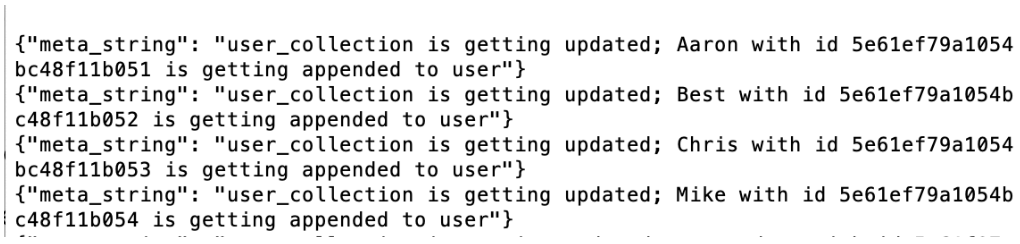


```
(base) dhcp-10-105-159-250:flask-atlas-updated stevenchen$ python app.py  kafka-
consumer.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 162-639-068
```

Step4: In a new terminal window, subscribe to the kafka topic and be ready to check on the messages; the command is:
kafka-console-consumer --bootstrap-server localhost:9092 --topic msg_collection --from-beginning

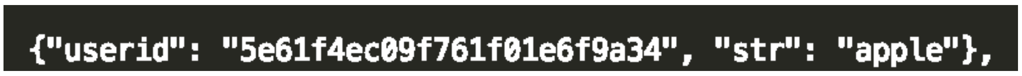
Step5: open up the browser and enter:
127.0.0.1:5000/register_user

This would allow the execution of the register_user() function call and take in info.json and create a new collection 'user_collection' to store the user objects. A new field 'dataid' which is an empty array will be created at the same time. Upon successful completion, the page would display 'users are registered' and we could see messages in the kafka-console-consumer window.



```
{"meta_string": "user_collection is getting updated; Aaron with id 5e61ef79a1054
bc48f11b051 is getting appended to user"}
{"meta_string": "user_collection is getting updated; Best with id 5e61ef79a1054b
c48f11b052 is getting appended to user"}
{"meta_string": "user_collection is getting updated; Chris with id 5e61ef79a1054
bc48f11b053 is getting appended to user"}
{"meta_string": "user_collection is getting updated; Mike with id 5e61ef79a1054b
c48f11b054 is getting appended to user"}
```

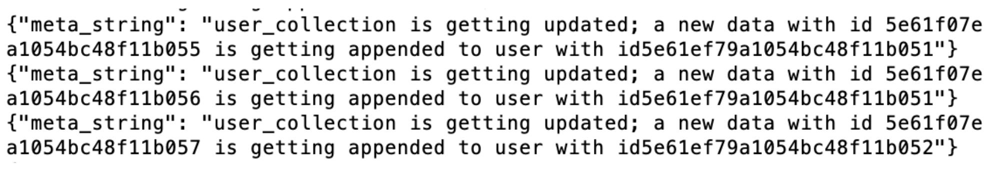
Step 6: go to data.json and create a 'userid' field for each entry; make sure you put the id string of each user to the according data



```
{"userid": "5e61f4ec09f761f01e6f9a34", "str": "apple"},
```

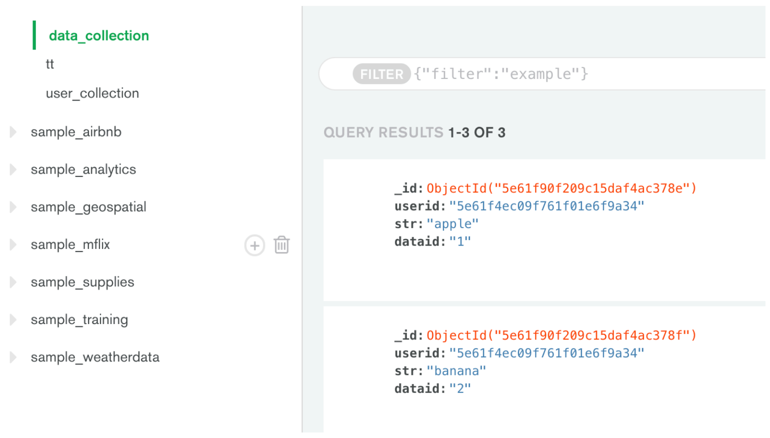
Step 7: clear up the browser and enter:
127.0.0.1:5000/register_data

This would allow the execution of the register_data() function call and take in data.json and create a new collection ‘data_collection’ to store the data objects. Here I set up an auto-incrementing data id for each data object so the ‘dataid’ array for user object can be more direct for view. Upon successful completion the page would display ‘data are saved’ and we could again see messages in kafka-console-cunsumer window:



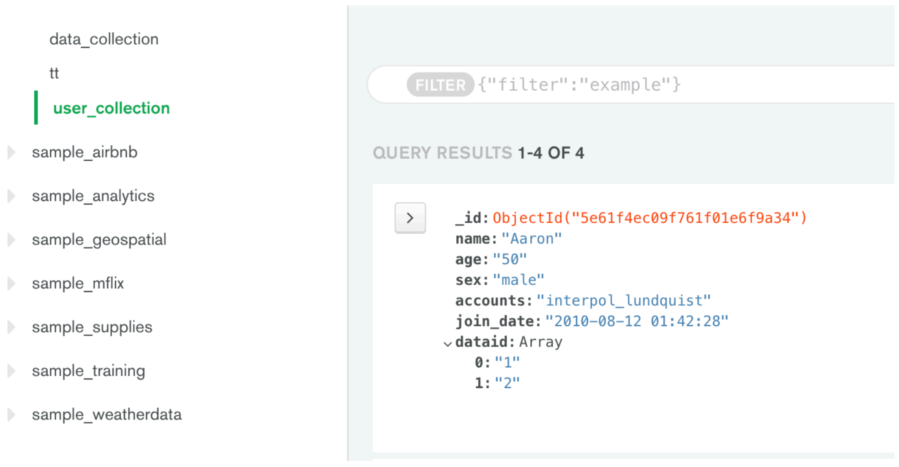
```
{"meta_string": "user_collection is getting updated; a new data with id 5e61f07e
a1054bc48f11b055 is getting appended to user with id5e61ef79a1054bc48f11b051"}
{"meta_string": "user_collection is getting updated; a new data with id 5e61f07e
a1054bc48f11b056 is getting appended to user with id5e61ef79a1054bc48f11b051"}
{"meta_string": "user_collection is getting updated; a new data with id 5e61f07e
a1054bc48f11b057 is getting appended to user with id5e61ef79a1054bc48f11b052"}
```

Now in MongoDB Atlas we can see data_collection and user_collection respectively like this:



```
{"filter":"example"}
```

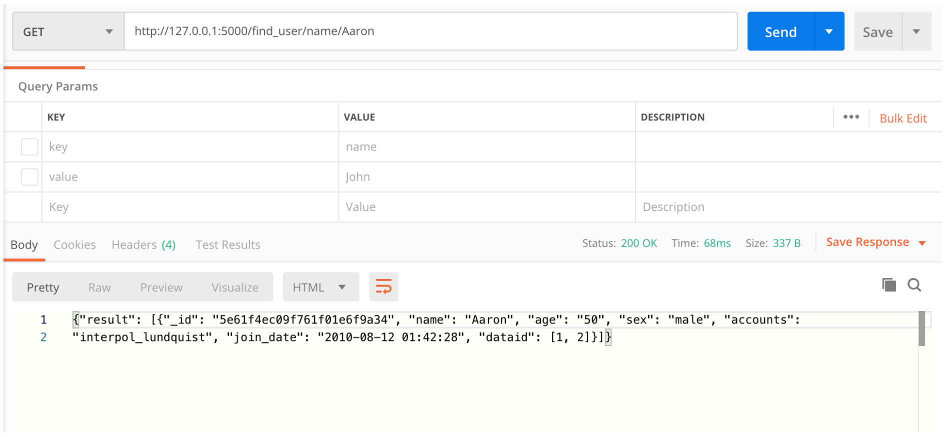
```
_id: ObjectId("5e61f90f209c15daf4ac378e")
userid: "5e61f4ec09f761f01e6f9a34"
str: "apple"
dataid: "1"

_id: ObjectId("5e61f90f209c15daf4ac378f")
userid: "5e61f4ec09f761f01e6f9a34"
str: "banana"
dataid: "2"
```



```
{"filter":"example"}
```

```
_id: ObjectId("5e61f4ec09f761f01e6f9a34")
name: "Aaron"
age: "50"
sex: "male"
accounts: "interpol_lundquist"
join_date: "2010-08-12 01:42:28"
dataid: Array
   0: "1"
   1: "2"
```

8 To retrieve the objects with their attributes, we can go to postman and hit our request to find user and data like this:

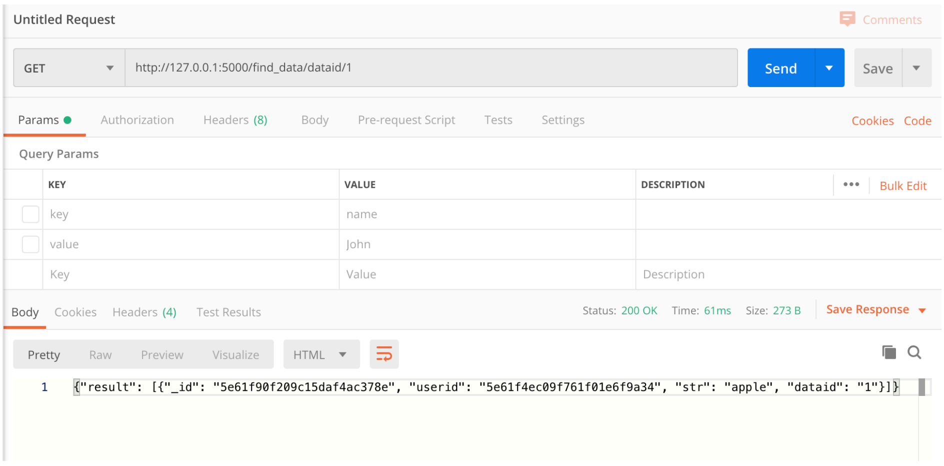

GET http://127.0.0.1:5000/find_user/name/Aaron

Query Params

KEY	VALUE	DESCRIPTION
key	name	
value	John	
Key	Value	Description

Status: 200 OK Time: 68ms Size: 337 B

```
{"result": [{"_id": "5e61f4ec09f761f01e6f9a34", "name": "Aaron", "age": "50", "sex": "male", "accounts":
"interpol_lundquist", "join_date": "2010-08-12 01:42:28", "dataid": [1, 2]}]}
```


Untitled Request

GET http://127.0.0.1:5000/find_data/dataid/1

Query Params

KEY	VALUE	DESCRIPTION
key	name	
value	John	
Key	Value	Description

Status: 200 OK Time: 61ms Size: 273 B

```
{"result": [{"_id": "5e61f90f209c15daf4ac378e", "userid": "5e61f4ec09f761f01e6f9a34", "str": "apple", "dataid": "1"}]}
```

Here we can see our queries are running successfully and that concludes our database management process.

links to tutorial I have used:

1 official documentation provided by mongoDB: https://docs.mongodb.com/manual/

2 kafka-mongo git repository: https://github.com/dpauk/kafka-mongo/blob/master/kafka-producer.py

3 running kafka on mac: https://medium.com/@Ankitthakur/apache-kafka-installation-on-mac-using-homebrew-a367cdefd273

4: python mongodb tutorial on w3school: https://www.w3schools.com/python/python_mongodb_getstarted.asp

5: flask example: https://www.freecodecamp.org/news/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492/

These are links for the code specifically; for other links of educational purposes I have included them in the presentation decks.

Talking about the whole project:
The project starts with data acquisition: we get raw data of interest from different sources and that comes in different formats. Then we use data lake as a container for the natural/raw data. The data from the lake is then pre-processed to deal with missing values/null inputs and make sure it is ready to be stored in databases. The data from database is then retrieved for model training and parameter tuning, the result of which is generated as report to present to user with data visualization techniques. The user only needs to put in the raw data and output should be the well-formatted report. All the intermediate steps should be invisible to the user and run on backend.

How my module can be enhanced:
the kafka messages can be put into S3 bucket storage and we can use the Amazon MSK which is AWS version of kafka instead. The find functions can be more diverse (support searching by multiple fields etc.)