Jason C. Nucciarone

2/15/2021

M03-A01: Use cases to Code

The first use case:

The first use case that I coded was the Post Rating use case. For this case I assumed that the Actor had already written their rating, the relevant information that entered was already retrieved by the class driving the user interface, and that class handled the necessary data conversion before the review was added to the database. I had my Java code print out the assumptions that I made:

```
Assumptions:

Review #1:


User ID: 1001

User Name: [Kyle, Montenucci]

Business ID: 2002

Business Name: MICHEALS_PIZZA_001

User Safety Rating: 7.25

Tags: [Not clean, Wearing masks wrong!]

User Review: Just bad staff. I usually love this place! Wear masks correctly next time please.


Review #2:


User ID: 1002

User Name: [Sarah, Conner]

Business ID: 2002

Business Name: MICHEALS_PIZZA_001

User Safety Rating: 5.0

Tags: [Not clean, No masks! More the 6 feet]

User Review: None of the kitchen staff were wearing masks, although, to give them credit they did have all
the customers spaced out more than 6 feet!


Review #3:


User ID: 1003
```

User Name: [Kaylie, Arujo]

Business ID: 2003

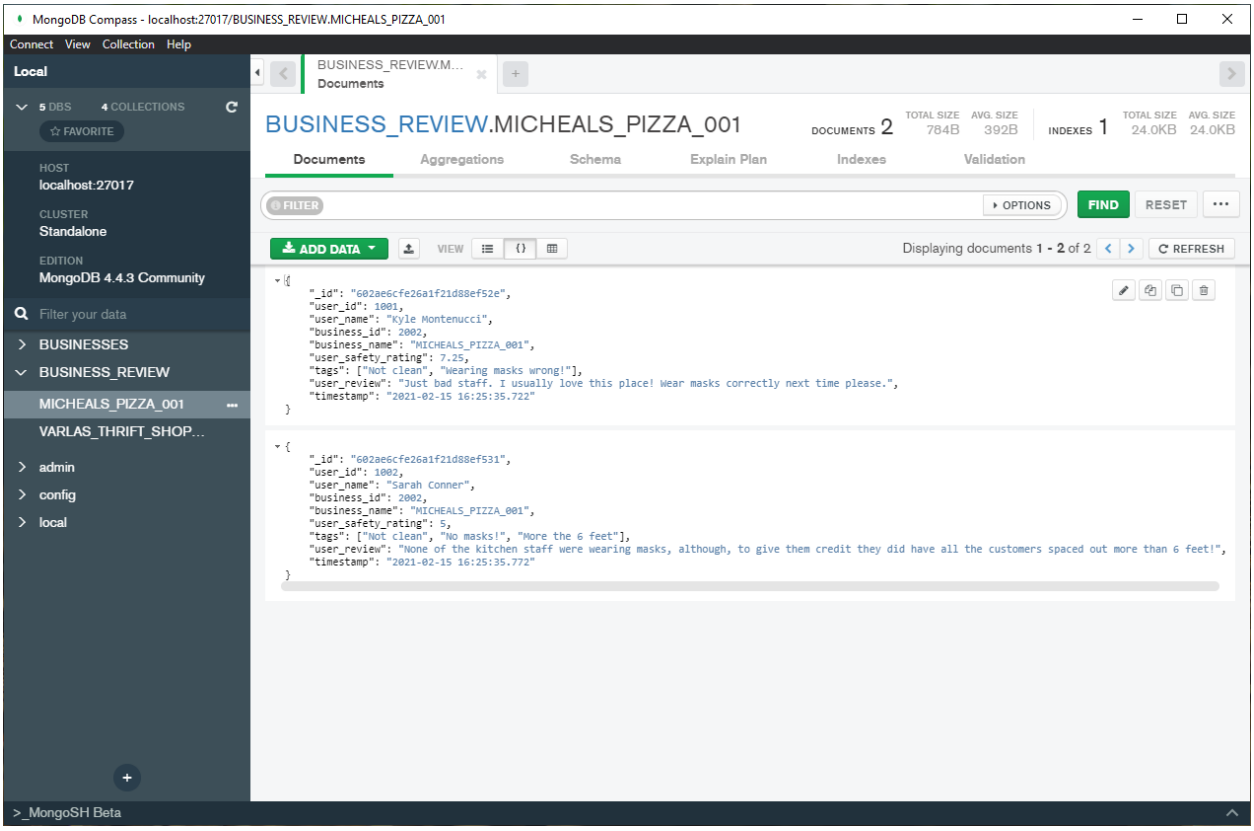
Business Name: VARLAS_THRIFT_SHOP_001

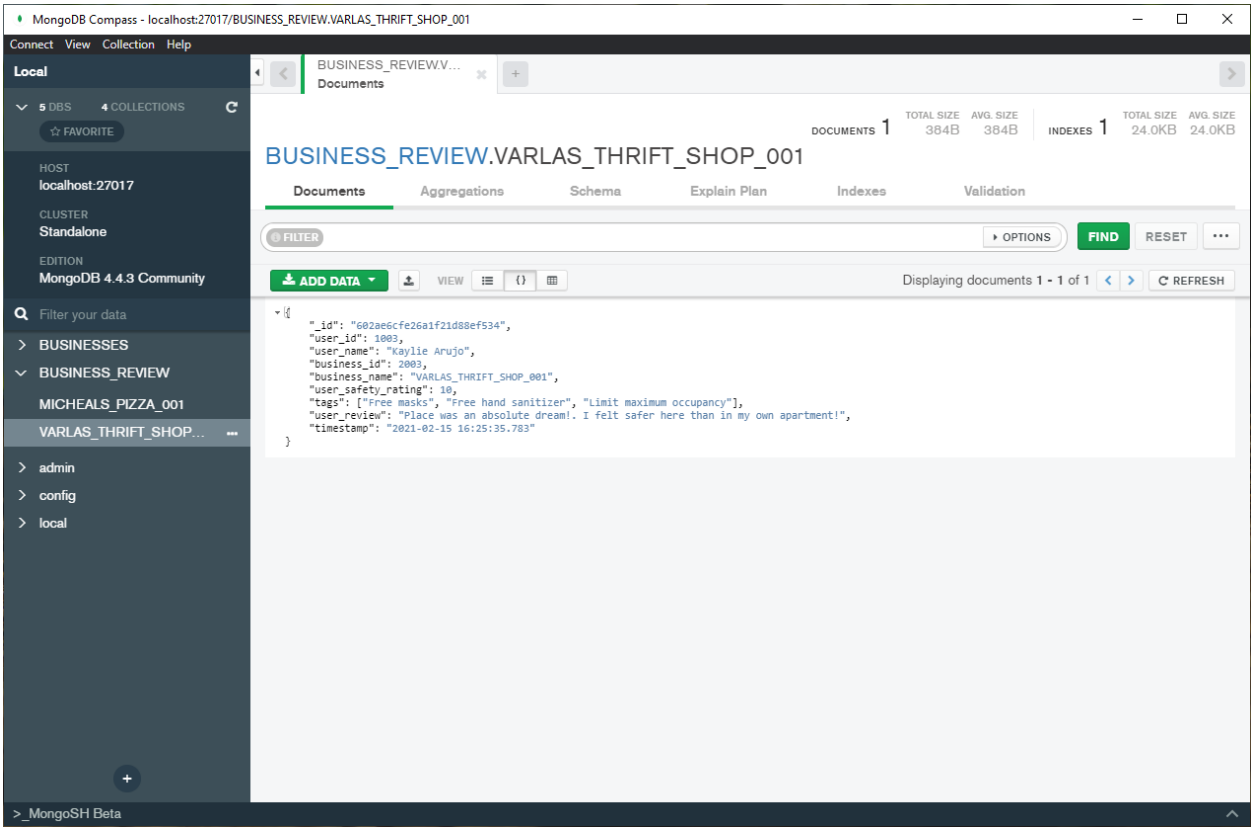
User Safety Rating: 10.0

Tags: [Free masks, Free hand sanitizer, Limit maximum occupancy]

User Review: Place was an absolute dream! I felt safer here than in my own apartment!

You can see that the documents were successfully created by opening MongoDB compass (default interface for MongoDB community edition):





I used a constructor for Post Rating to handle opening the connection to the MongoDB (open source noSQL DBMS) instance, and then I used the function updateCollection to add the rating to the database. After the rating was added to the database, I then used the function refresh to retrieve the updated collection. Here are the results printed out by my Java code below:

```
Results:


Document{{_id=602ae6cfe26a1f21d88ef52e, user_id=1001, user_name=Kyle Montenucci,
business_id=2002, business_name=MICHEALS_PIZZA_001, user_safety_rating=7.25,
tags=[Not clean, Wearing masks wrong!], user_review=Just bad staff. I usually love
this place! Wear masks correctly next time please., timestamp=2021-02-15
16:25:35.722}}

Document{{_id=602ae6cfe26a1f21d88ef52e, user_id=1001, user_name=Kyle Montenucci,
business_id=2002, business_name=MICHEALS_PIZZA_001, user_safety_rating=7.25,
tags=[Not clean, Wearing masks wrong!], user_review=Just bad staff. I usually love
this place! Wear masks correctly next time please., timestamp=2021-02-15
16:25:35.722}}

Document{{_id=602ae6cfe26a1f21d88ef531, user_id=1002, user_name=Sarah Conner,
business_id=2002, business_name=MICHEALS_PIZZA_001, user_safety_rating=5.0, tags=[Not
clean, No masks!, More the 6 feet], user_review=None of the kitchen staff were
```

```
wearing masks, although, to give them credit they did have all the customers spaced
out more than 6 feet!, timestamp=2021-02-15 16:25:35.772}}

Document{{_id=602ae6cfe26a1f21d88ef534, user_id=1003, user_name=Kaylie Arujo,
business_id=2003, business_name=VARLAS_THRIFT_SHOP_001, user_safety_rating=10.0,
tags=[Free masks, Free hand sanitizer, Limit maximum occupancy], user_review=Place
was an absolute dream!. I felt safer here than in my own apartment!, timestamp=2021-
02-15 16:25:35.783}}
```

What is happening here is that the Post Rating use case is returning BSON (Binary JSON) documents that were retrieved from the MongoDB database. Ideally, after writing the code for the user interface, it will pull in these BSON documents and then use the relevant information to update the screen to show that a new review was written.

Here is the code for the Post Rating use case below:

```java
package ist311.usecases;

// Import local packages here
import com.mongodb.*;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import org.bson.types.ObjectId;

import ist311.utils.ConnectDB;

import org.javatuples.Pair;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

/**
 * Insert user review into business's specific MongoDB collection, and then
 * refresh the UI by returning the Document(s) returned by MongoDB.
 * @author Jason C. Nucciarone
 *
 */
public class PostRating {
    private int user_id;
    private Pair<String, String> user_name;
    private int business_id;
    private String business_name;
    private double user_safety_rating;
    private List<String> tags;
    private String user_review;
    private MongoClient conn;
    private Timestamp timestamp;

    // Constructor - assume data is already collected from GUI
```

```
    public PostRating (int user_id, Pair<String, String> user_name, int
business_id, String business_name, double user_safety_rating,
                List<String> tags, String user_review){
        this.user_id = user_id;
        this.user_name = user_name;
        this.business_id = business_id;
        this.business_name = business_name;
        this.user_safety_rating = user_safety_rating;
        this.tags = tags;
        this.user_review = user_review;
        this.conn = ConnectDB.getConnection();

        // Generate timestamp for document
        Calendar calendar = Calendar.getInstance();
        this.timestamp = new Timestamp(calendar.getTime().getTime());
    }

    public void updateCollection(){
        try {
            // Connect to BUSINESS_REVIEW database
            MongoClient mongoClient = getConn();
            MongoDatabase database =
mongoClient.getDatabase("BUSINESS_REVIEW");

            // Grab collection
            MongoCollection<Document> table =
database.getCollection(getBusiness_name());

            // Create Document and create Unique ID
            Document doc = new Document("_id", new ObjectId().toString());

            // Process the data into a BSON readable format
            Pair<String, String> temp = getUser_name();
            String full_name = temp.getValue0() + " " + temp.getValue1();
            List<String> temp_list = getTags();
            temp_list.toArray();
            Timestamp temp_timestamp = getTimestamp();
            String temp_string_2 = temp_timestamp.toString();

            // Add data to Document
            doc.append("user_id", getUser_id());
            doc.append("user_name", full_name);
            doc.append("business_id", getBusiness_id());
            doc.append("business_name", getBusiness_name());
            doc.append("user_safety_rating", getUser_safety_rating());
            doc.append("tags", temp_list);
            doc.append("user_review", getUser_review());
            doc.append("timestamp", temp_string_2);

            // Add document to database
            table.insertOne(doc);

        } catch (Exception e){
            String error = "Failed to connect to MongoDB instance! Check if
MongoDB is running!";
            System.out.println(error);
            throw e;
```

```
        }
    }

    public ArrayList<Document> refresh(){
        try {
            // Refresh the business page by returning Documents from
collection
            MongoClient mongoClient = getConn();
            MongoDatabase database =
mongoClient.getDatabase("BUSINESS_REVIEW");
            MongoCollection<Document> table =
database.getCollection(getBusiness_name());

            // Search the collection by business ID
            ArrayList<Document> docs = new ArrayList<>();
            BasicDBObject query = new BasicDBObject("business_id",
getBusiness_id());
            try (MongoCursor<Document> cursor =
table.find(query).iterator()){
                while (cursor.hasNext()){
                    docs.add(cursor.next());
                }
                return docs;
            }

        } catch (Exception e){
            String error = "Failed to refresh UI! Check MongoDB instance";
            System.out.println(error);
            throw e;
        }
    }

    // Setters and getters
    public int getUser_id() {
        return user_id;
    }

    public void setUser_id(int user_id) {
        this.user_id = user_id;
    }

    public Pair<String, String> getUser_name() {
        return user_name;
    }

    public void setUser_name(Pair<String, String> user_name) {
        this.user_name = user_name;
    }

    public int getBusiness_id() {
        return business_id;
    }

    public void setBusiness_id(int business_id) {
        this.business_id = business_id;
    }
```

```
    public String getBusiness_name() {
        return business_name;
    }

    public void setBusiness_name(String business_name) {
        this.business_name = business_name;
    }

    public double getUser_safety_rating() {
        return user_safety_rating;
    }

    public void setUser_safety_rating(double user_safety_rating) {
        this.user_safety_rating = user_safety_rating;
    }

    public List<String> getTags() {
        return tags;
    }

    public void setTags(List<String> tags) {
        this.tags = tags;
    }

    public String getUser_review() {
        return user_review;
    }

    public void setUser_review(String user_review) {
        this.user_review = user_review;
    }

    public MongoClient getConn() {
        return conn;
    }

    public void setConn(MongoClient conn) {
        this.conn = conn;
    }

    public Timestamp getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(Timestamp timestamp) {
        this.timestamp = timestamp;
    }
}
```

The second use case:

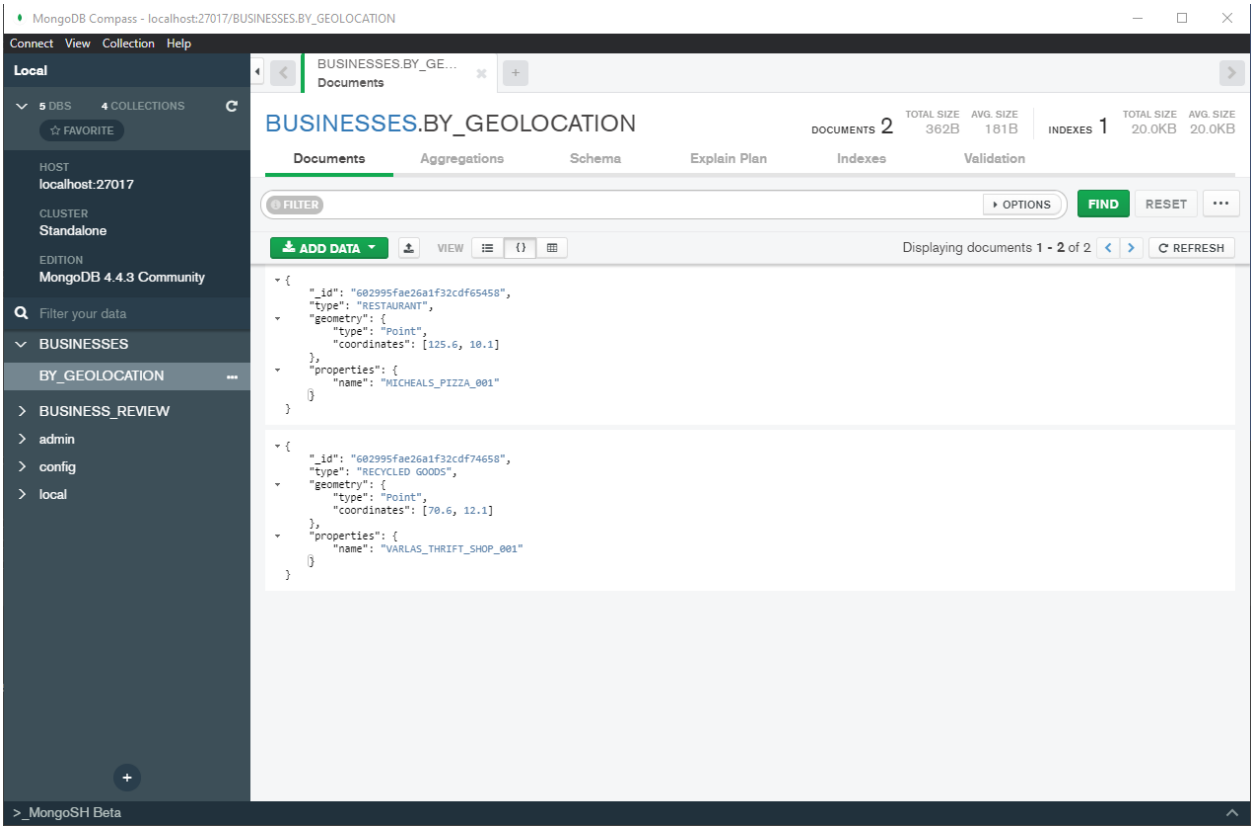
The second use case that I coded was the Search for Business use case. For this use case, I assumed that another use case/earlier step in the process retrieved the Actor's location by utilizing the Actor's device's geolocation API. I also had my Java code print out the location assumptions:

```
Assumptions:

First coordinates: [125.6, 10.1]

Second coordinates: [70.6, 12.1]
```

Using the GeoJSON data format, I queried the MongoDB database looking for a match the coordinates put in by the code driving the user interface. Ideally, as I write out this use case even more, Search for Business will have more constructors that allow the application and Actor to search for businesses in multiple ways. That being said, the function I programmed for this use case is searchByGeolocation. This function queries the BY_GEOLOCATION collection in the MongoDB database:



Once the query finds a match, it returns the name of the business that the user is currently at. I had my Java code print out the result as well:

```
Results:


MICHEALS_PIZZA_001

VARLAS_THRIFT_SHOP_001
```

Another function will then take this information in and will query the database to retrieve another BSON document to update the main screen.

Here is the Java code for the Search for Business use case:

```java
package ist311.usecases;

// Import local packages here
import com.mongodb.*;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;

import ist311.utils.ConnectDB;
import org.javatuples.Pair;


/**
 * Query BUSINESSES database in various ways to find the correct business for
a user.
 * For this assignment, searching by matching GeoJson coordinates.
 * @author Jason C. Nucciarone
 *
 */
public class SearchBusiness {
    private MongoClient conn;
    private Pair<Double, Double> coordinates;

    public SearchBusiness(Pair<Double, Double> coordinates){
        this.conn = ConnectDB.getConnection();
        this.coordinates = coordinates;
    }

    public String searchByGeoLocation(){
        // Connect to BUSINESSES database
        MongoClient mongoClient = getConn();
        MongoDatabase database = mongoClient.getDatabase("BUSINESSES");

        // Grab collection
        MongoCollection<Document> table =
database.getCollection("BY_GEOLOCATION");
```

```
        BasicDBObject query = new BasicDBObject("geometry",
                new BasicDBObject("type", "Point").append("coordinates",
coordinates.toArray()));
        // Locate business by geolocation data
        try (MongoCursor<Document> cursor = table.find(query).iterator()){
            if (cursor.hasNext()){
                // Get specific document
                Document doc = cursor.next();

                // Close connection to database
                mongoClient.close();
                return doc.get("properties",
Document.class).getString("name");

            } else {
                mongoClient.close();
                return null;
            }
        }
    }

    // Setters and getters
    public MongoClient getConn() {
        return conn;
    }

    public void setConn(MongoClient conn) {
        this.conn = conn;
    }

    public Pair<Double, Double> getCoordinates() {
        return coordinates;
    }

    public void setCoordinates(Pair<Double, Double> coordinates) {
        this.coordinates = coordinates;
    }
}
```

Console output when two use case are executed (the entire stacktrace):

```
"C:\Users\Jason Nucciarone\.jdks\openjdk-15.0.2\bin\java.exe" "-javaagent:D:\devtools\IntelliJ IDEA
2020.3.2\lib\idea_rt.jar=56941:D:\devtools\IntelliJ IDEA 2020.3.2\bin" -Dfile.encoding=UTF-8 -classpath
"D:\git_projects_school\jason_M03_A01\out\production\jason_M03_A01;C:\Users\Jason
Nucciarone\.m2\repository\org\mongodb\mongodb-driver\3.0.0\mongodb-driver-3.0.0.jar;C:\Users\Jason
Nucciarone\.m2\repository\org\mongodb\mongodb-driver-core\3.0.0\mongodb-driver-core-
3.0.0.jar;C:\Users\Jason Nucciarone\.m2\repository\org\mongodb\bson\3.0.0\bson-
3.0.0.jar;D:\git_projects_school\jason_M03_A01\lib\mongodb-driver-
3.12.7.jar;D:\git_projects_school\jason_M03_A01\lib\bson-
3.12.7.jar;D:\git_projects_school\jason_M03_A01\lib\mongodb-driver-core-
3.12.7.jar;D:\git_projects_school\jason_M03_A01\lib\javatuples-1.2.jar" ist311.main.Main

Feb 15, 2021 5:22:42 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN,
serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN,
serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:2, serverValue:579}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:1, serverValue:580}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description
ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true,
version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9,
maxDocumentSize=16777216, roundTripTimeNanos=668100}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description
ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true,
version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9,
maxDocumentSize=16777216, roundTripTimeNanos=455600}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:3, serverValue:581}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN,
serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN,
serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: No server chosen by PrimaryServerSelector from cluster description ClusterDescription{type=UNKNOWN,
connectionMode=SINGLE, all=[ServerDescription{address=localhost:27017, type=UNKNOWN, state=CONNECTING}]}.
Waiting for 30000 ms before timing out

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:4, serverValue:582}] to localhost:27017
```

```
Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:5, serverValue:583}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description
ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true,
version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9,
maxDocumentSize=16777216, roundTripTimeNanos=293800}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description
ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true,
version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9,
maxDocumentSize=16777216, roundTripTimeNanos=375300}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:6, serverValue:584}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN,
serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN,
serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: No server chosen by PrimaryServerSelector from cluster description ClusterDescription{type=UNKNOWN,
connectionMode=SINGLE, all=[ServerDescription{address=localhost:27017, type=UNKNOWN, state=CONNECTING}]}.
Waiting for 30000 ms before timing out

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:7, serverValue:585}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description
ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true,
version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9,
maxDocumentSize=16777216, roundTripTimeNanos=360400}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:8, serverValue:586}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description
ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true,
version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9,
maxDocumentSize=16777216, roundTripTimeNanos=281600}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:9, serverValue:587}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN,
serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}
```

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: No server chosen by ReadPreferenceServerSelector{readPreference=primary} from cluster description ClusterDescription{type=UNKNOWN, connectionMode=SINGLE, all=[ServerDescription{address=localhost:27017, type=UNKNOWN, state=CONNECTING}]}. Waiting for 30000 ms before timing out

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:10, serverValue:588}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true, version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9, maxDocumentSize=16777216, roundTripTimeNanos=297100}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:11, serverValue:589}] to localhost:27017

Assumptions:

Review #1:

User ID: 1001

User Name: [Kyle, Montenucci]

Business ID: 2002

Business Name: MICHEALS_PIZZA_001

User Safety Rating: 7.25

Tags: [Not clean, Wearing masks wrong!]

User Review: Just bad staff. I usually love this place! Wear masks correctly next time please.

Review #2:

User ID: 1002

User Name: [Sarah, Conner]

Business ID: 2002

Business Name: MICHEALS_PIZZA_001

User Safety Rating: 5.0

Tags: [Not clean, No masks!, More the 6 feet]

User Review: None of the kitchen staff were wearing masks, although, to give them credit they did have all the customers spaced out more than 6 feet!

Review #3:

User ID: 1003

User Name: [Kaylie, Arujo]

Business ID: 2003

Business Name: VARLAS_THRIFT_SHOP_001

User Safety Rating: 10.0

Tags: [Free masks, Free hand sanitizer, Limit maximum occupancy]

User Review: Place was an absolute dream!. I felt safer here than in my own apartment!

Results:

Document{{_id=602af433e26a1f5644509c0b, user_id=1001, user_name=Kyle Montenucci, business_id=2002, business_name=MICHEALS_PIZZA_001, user_safety_rating=7.25, tags=[Not clean, Wearing masks wrong!], user_review=Just bad staff. I usually love this place! Wear masks correctly next time please., timestamp=2021-02-15 17:22:43.012}}

Document{{_id=602af433e26a1f5644509c0b, user_id=1001, user_name=Kyle Montenucci, business_id=2002, business_name=MICHEALS_PIZZA_001, user_safety_rating=7.25, tags=[Not clean, Wearing masks wrong!], user_review=Just bad staff. I usually love this place! Wear masks correctly next time please., timestamp=2021-02-15 17:22:43.012}}

Document{{_id=602af433e26a1f5644509c0e, user_id=1002, user_name=Sarah Conner, business_id=2002, business_name=MICHEALS_PIZZA_001, user_safety_rating=5.0, tags=[Not clean, No masks!, More the 6 feet], user_review=None of the kitchen staff were wearing masks, although, to give them credit they did have all the customers spaced out more than 6 feet!, timestamp=2021-02-15 17:22:43.062}}

Document{{_id=602af433e26a1f5644509c11, user_id=1003, user_name=Kaylie Arujo, business_id=2003, business_name=VARLAS_THRIFT_SHOP_001, user_safety_rating=10.0, tags=[Free masks, Free hand sanitizer, Limit maximum occupancy], user_review=Place was an absolute dream!. I felt safer here than in my own apartment!, timestamp=2021-02-15 17:22:43.072}}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true, version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9, maxDocumentSize=16777216, roundTripTimeNanos=358600}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:12, serverValue:590}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Closed connection [connectionId{localValue:12, serverValue:590}] to localhost:27017 because the pool has been closed.

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: No server chosen by ReadPreferenceServerSelector{readPreference=primary} from cluster description ClusterDescription{type=UNKNOWN, connectionMode=SINGLE, all=[ServerDescription{address=localhost:27017, type=UNKNOWN, state=CONNECTING}]}. Waiting for 30000 ms before timing out

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:13, serverValue:591}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:14, serverValue:592}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true, version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9, maxDocumentSize=16777216, roundTripTimeNanos=369000}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true, version=ServerVersion{versionList=[4, 4, 3]}, minWireVersion=0, maxWireVersion=9, maxDocumentSize=16777216, roundTripTimeNanos=278200}

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Opened connection [connectionId{localValue:15, serverValue:593}] to localhost:27017

Feb 15, 2021 5:22:43 PM com.mongodb.diagnostics.logging.JULLogger log

INFO: Closed connection [connectionId{localValue:15, serverValue:593}] to localhost:27017 because the pool has been closed.

Assumptions:

First coordinates: [125.6, 10.1]

Second coordinates: [70.6, 12.1]

Results:

MICHEALS_PIZZA_001

VARLAS_THRIFT_SHOP_001

Process finished with exit code 0

The code for Main.java:

In case you were wondering how I ran these tests I used a Main class to function as the controller. Here is the code for Main.java below:

```
package ist311.main;

// Import local packages here
import ist311.usecases.PostRating;
import ist311.usecases.SearchBusiness;

import org.bson.Document;
import org.javatuples.Pair;
import java.util.ArrayList;
import java.util.List;

/**
 * Main method to test that PostRating and SearchBusiness Use Cases work.
 * @author Jason C. Nucciarone
 *
 */
public class Main {
    public static void main(String[] args) {
        /*
          First, to test our ability to update the BUSINESS_REVIEW database.
          Try to add two entries to separate collections.

          Assumptions:
          We will be assuming that the user has already written their review
          and that we have retrieved the information from the user interface.
          We will also be assuming that the database and relevant collections
          are already defined.

          The following fields will have been completed:
            - user_id
            - user_name
            - business_id
            - business_name
            - user_safety_rating
            - tags
            - user_review
         */

        // Define assumptions
        int user_id_1 = 1001;
        Pair<String, String> user_name_1 = Pair.with("Kyle", "Montenucci");
        int business_id_1 = 2002;
        String business_name_1 = "MICHEALS_PIZZA_001";
        double user_safety_rating_1 = 7.25;
        List<String> tags_1 = new ArrayList<>();
        tags_1.add("Not clean");
        tags_1.add("Wearing masks wrong!");
        String user_review_1 = "Just bad staff. I usually love this place!
Wear masks correctly next time please.";

        // Add to database
```

```
        PostRating rating_1 = new PostRating(user_id_1,
                user_name_1,
                business_id_1,
                business_name_1,
                user_safety_rating_1,
                tags_1,
                user_review_1);
        rating_1.updateCollection();
        ArrayList<Document> ratings_for_1 = rating_1.refresh();

        // Create another document
        int user_id_2 = 1002;
        Pair<String, String> user_name_2 = Pair.with("Sarah", "Conner");
        int business_id_2 = 2002;
        String business_name_2 = "MICHEALS_PIZZA_001";
        double user_safety_rating_2 = 5;
        List<String> tags_2 = new ArrayList<>();
        tags_2.add("Not clean");
        tags_2.add("No masks!");
        tags_2.add("More the 6 feet");
        String user_review_2 = "None of the kitchen staff were wearing masks,
although, to give them credit they did " +
                "have all the customers spaced out more than 6 feet!";

        PostRating rating_2 = new PostRating(user_id_2,
                user_name_2,
                business_id_2,
                business_name_2,
                user_safety_rating_2,
                tags_2,
                user_review_2);
        rating_2.updateCollection();
        ArrayList<Document> ratings_for_2 = rating_2.refresh();

        // Create document in a different collection
        int user_id_3 = 1003;
        Pair<String, String> user_name_3 = Pair.with("Kaylie", "Arujo");
        int business_id_3 = 2003;
        String business_name_3 = "VARLAS_THRIFT_SHOP_001";
        double user_safety_rating_3 = 10;
        List<String> tags_3 = new ArrayList<>();
        tags_3.add("Free masks");
        tags_3.add("Free hand sanitizer");
        tags_3.add("Limit maximum occupancy");
        String user_review_3 = "Place was an absolute dream!. I felt safer
here than in my own apartment!";

        PostRating rating_3 = new PostRating(user_id_3,
                user_name_3,
                business_id_3,
                business_name_3,
                user_safety_rating_3,
                tags_3,
                user_review_3);
        rating_3.updateCollection();
        ArrayList<Document> ratings_for_3 = rating_3.refresh();
```

```
        // Print out assumptions and results
        System.out.println("Assumptions:\n");
        System.out.println("Review #1:\n");
        System.out.println("User ID: " + user_id_1);
        System.out.println("User Name: " + user_name_1);
        System.out.println("Business ID: " + business_id_1);
        System.out.println("Business Name: " + business_name_1);
        System.out.println("User Safety Rating: " + user_safety_rating_1);
        System.out.println("Tags: " + tags_1);
        System.out.println("User Review: " + user_review_1);

        System.out.println("\nReview #2:\n");
        System.out.println("User ID: " + user_id_2);
        System.out.println("User Name: " + user_name_2);
        System.out.println("Business ID: " + business_id_2);
        System.out.println("Business Name: " + business_name_2);
        System.out.println("User Safety Rating: " + user_safety_rating_2);
        System.out.println("Tags: " + tags_2);
        System.out.println("User Review: " + user_review_2);

        System.out.println("\nReview #3:\n");
        System.out.println("User ID: " + user_id_3);
        System.out.println("User Name: " + user_name_3);
        System.out.println("Business ID: " + business_id_3);
        System.out.println("Business Name: " + business_name_3);
        System.out.println("User Safety Rating: " + user_safety_rating_3);
        System.out.println("Tags: " + tags_3);
        System.out.println("User Review: " + user_review_3);

        System.out.println("\nResults:\n");

        for (int i = 0; i < ratings_for_1.size(); i++) {
            System.out.println(ratings_for_1.get(i));
        }

        for (int i = 0; i < ratings_for_2.size(); i++) {
            System.out.println(ratings_for_2.get(i));
        }

        for (int i = 0; i < ratings_for_3.size(); i++) {
            System.out.println(ratings_for_3.get(i));
        }

        /*
          Second, we need to check if we are to search the BUSINESSES
          database to find a business as well as return that relevant
          info for the business.

          The assumption that we are making is that we already have the
          geojson data available for conducting the search. We are also
          assuming that the user has their devices geolocation API enabled.
         */

        // Locate first business
        Pair<Double, Double> coordinates_1 = Pair.with(125.6, 10.1);
        SearchBusiness business_1 = new SearchBusiness(coordinates_1);
        String business_location_1 = business_1.searchByGeoLocation();
```

```
        // Locate second business
        Pair<Double, Double> coordinates_2 = Pair.with(70.6, 12.1);
        SearchBusiness business_2 = new SearchBusiness(coordinates_2);
        String business_location_2 = business_2.searchByGeoLocation();

        // Print out assumptions and results
        System.out.println("\nAssumptions:\n");
        System.out.println("First coordinates: " + coordinates_1.toString());
        System.out.println("Second coordinates: " +
coordinates_2.toString());

        System.out.println("\nResults:\n");
        System.out.println(business_location_1);
        System.out.println(business_location_2);
    }
}
```

If you are generally curious to what the overall structure of my code is, then please see the GitHub repository that I set up here: https://github.com/ist-311-crew-and-wist-girl/jason_M03_A01. I recommend viewing the repository because you can see how I structured my code as well as some utility classes that I wrote to remove the tedium from my use cases. If you have any specific questions about the repository, then please contact me at jcn23@psu.edu.