# Coding Challenge Solution

## ***Problem Statement:***

Using the provided data (`tickets.json` and `users.json` and `organization.json`) write a simple command line application (or a locally runnable web-app) to search the data and return the results in a human readable format.

When executing a search operation, where data exists, values from any related entities should be included in the results. The user should be able to search on any field and exact value matching is fine (e.g. “mar” won’t return “mary”). The user should also be able to search for empty values, e.g. where description is empty.

## ***My Approach:***

I looked at the 3 JSON files and saw the pattern as well as the number of records in each of the JSON files and it was not more than 1100 items. The unique key is _id in all the 3 JSON files and the other fields may have duplicates. To handle these many entries I decided to use HashMap data structure for storing and retrieval. This is a search application where you want to search on any field with the exact value and I feel the use of HashMap should suffice this.

I have used a JSON parser, simple JSON parser to parse the JSON file and store it in a JSON Array Object . Then read each object from the JSON array, and then store it in 2 hashMaps.

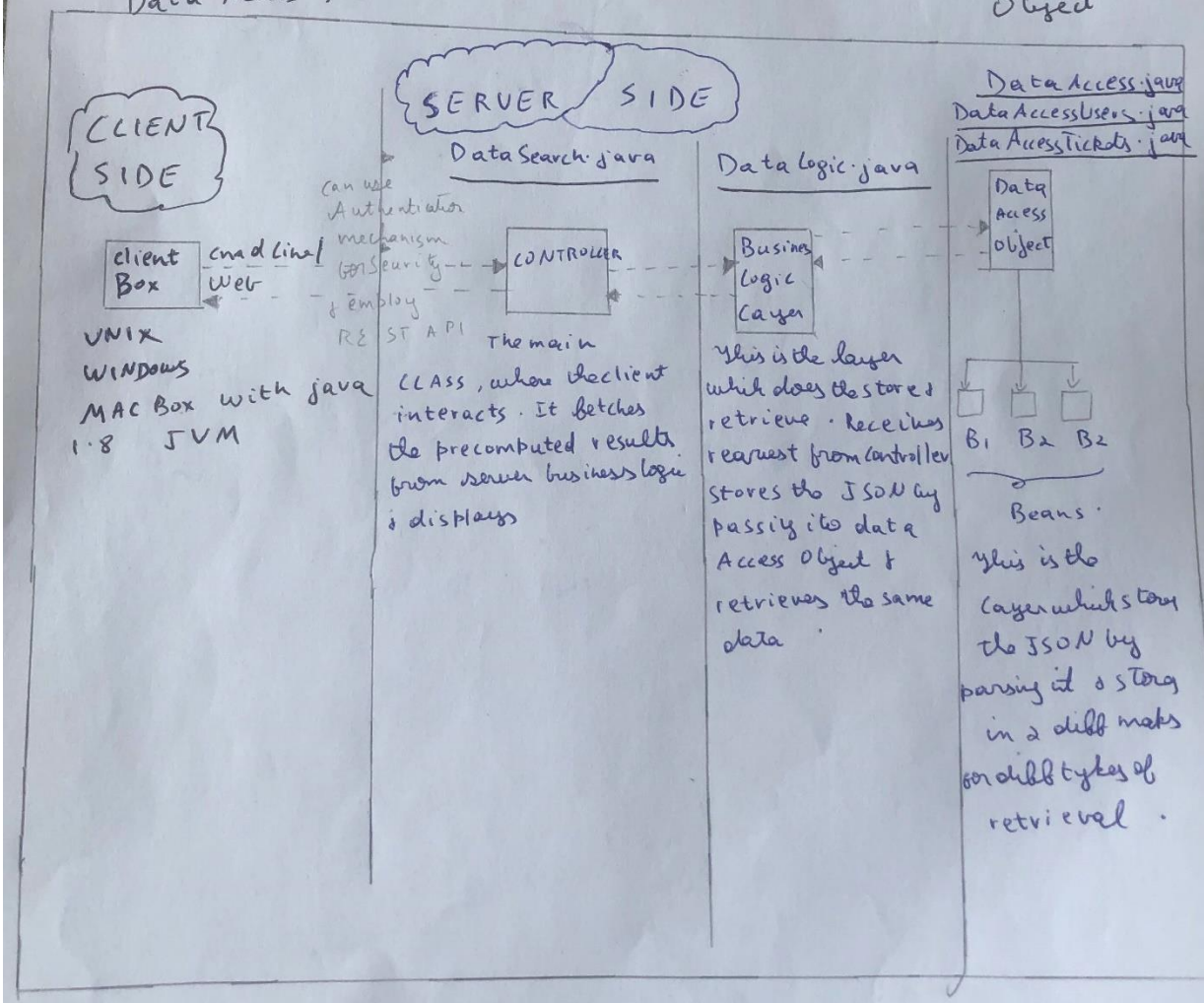
I have employed 2 HashMaps one for searching using _id which is the unique key and for searching using other fields I have used a reference hashMap. In the first HashMap, I have used the _id as the key and the resultant output as the value. In the second HashMap, I have used the combination of JSON key name and its associated value as the HashMap Key and the corresponding _id as the HashMap value.

**Case1:** (Applies to all three types of JSON data) When you are searching the Users and you have entered _id as the search criteria and enter the corresponding value, it goes to the first hashMap and does get the corresponding output value for the search criteria value. Only one value will be returned per id. The key value data type is <Long, String> where the id and the result output

**Case 2:** If you are entering other fields eg. Role and the value is admin, it means we are getting a list of users where all the users are admins. In the second hashMap I am using a <String, ArrayList<Long>> as the type. The string is a combination of JSON key id and the value, so the key in this case is roleadmin. If this is key is already there, I am adding the corresponding user id to the arrayList. When you enter this corresponding search criteria, it gets the arrayList of id’s and then gets the corresponding output from the first hashMap. The second hashmap is a reference map to get the id’s which in turn is fed to the first hashMap to the get the search output.

# Design Layout :-

I have designed the search tool using 5 classes.
DataSearch → Controller, DataLogic → Business Logic
DataAccess, DataAccessUsers, DataAccessTickets → Data Access Object.




Design Architecture.

## ***Data Flow:***

When the client machine runs the Command Line tool and calls the main class, before the client interacts by entering the options, the Controller layer communicates with the dataLogic layer, which handles the data storing logic. The Data Logic layer communicates with the data access object layer and the JSON data is stored in two hash Maps which is already explained above. Even before the client interacts, all the search results are precomputed. This eliminates the redundancy, saving time and space, thereby making it singleton app to some extent. If you have already stored the results, accessing the results for the corresponding search criteria will be done in O(1) time.

When the client starts interacting with the CLI and enters the search criteria, the search criteria is validated and sent to the Data Logic Layer. The Data Logic Layer fetches the result which is passed to controller which in turn is displayed on the command line screen. I have employed a do while loop that the user can do as many searches and finish it. Every time a search request is sent, the results are obtained by querying the HashMap.

## ***Run Time Complexity:***

Data is stored in data access layer by parsing the JSON files. Let N be the number of entries. For users is about (1100 entries) and M be the number of keys for the corresponding entry.

**Storing:** O(N*M) ~ O(N) (as M is small, for tickets the M value is not more than 20, for users M value is also not more than 20, for organization, M value is about 9)

**Retriveing**: O(1) if you are searching using id

**Retrieving:** O(1) if you are searching based on the other field. For eg, via(key) chat (value) while searching tickets, you get 70 entries, it about O(70) which comes as constant time

## ***Space Complexity:***

**HashMap1:** O(N) where N is the number of entries in a JSON file

**HashMap2:** O(N*M) ~ O(N) as M is small, where N is number of entries and M is number of keys in that JSON file

## ***Pros:***

1) Search results are already precomputed in a HashMap. So, it avoids unnecessary loading of JSON data time and again

2) Search results are computed very quickly time and again and proves to be a good solution for this dataset
3) It's a simple command line tool and user interactions are simple

## ***Cons and Future work:***

I have identified the cons, areas of improvement and the corresponding work around. My future work is addressing these cons and working on it to make my search application robust and simple

1) The tool does not scale for a large JSON file with more number of keys which cannot be handled in a single hashMap where the N and M numbers are so high (N total number of entries and M is key associated with that entries). The main drawback is the data structure employed for storing large data . This can be rectified by using a Mongo DB to store the Data and handle a vast volume of data and retrieve data from the Mongo DB. In other words, add a model component to my application design. To further improvise I will implement a Lucene index and it will be even quicker and efficient
2) Data Abstraction can be implemented more efficiently. I am making the design more efficient by employing the Object-Oriented concepts. I have designed the application which handles our primary objective and committed in Git. These optimizations will be added as a branch to the master and make my application more efficient.