High Level Design

Problem Statement

To design an algorithm that will determine a list of possible properties with match percentages(40% and above) from a huge set of properties.

Solution

I will maintain two tables : one for property and another for requirement. Initially our DB will be empty. On addition of new property, will keep that in properties table and on each new search requirement, will add that in requirements table.

- **Properties set will have columns**
  - Id,
  - Latitude,
  - Longitude,
  - Price,
  - Number of bedrooms,
  - Number of bathrooms,
  - Type - Sale/Rental
  - Status - Available/Sold

- **SearchRequirements set will have columns**
  - Id,
  - Latitude,
  - Longitude,
  - Min Budget,
  - Max budget,
  - Min Bedrooms required,
  - Max bedroom reqd,
  - Min bathroom reqd,
  - Max bathroom reqd.
  - Type - Buy/Renter
  - Status - active/stale
  - ListofProperties {Comma separated list of eligible propertyIds}

Validity Requirements

- Distance(+/- 10 miles)
- Budget (+/- 25%)
- No. of Bedroom(+/- 2)
- No. of Bathroom(+/- 2)

Functional Requirements

- Validity requirements
- Matching Score >= 40%

Scoring

Distance within 2 miles : 30% weightage
Weightage formula : 30*(10-x)/8 where x is distance

Budget within min and max : 30% weightage
Budget within min and max and within +/-25% : linearly decrease the weightage

Budget without min or max and within +/- 10% : 30% weightage
Budget without min or max and more than +/-10% : 0%

Bedroom/Bathroom within min and max : 20%
Bedroom/Bathroom +/-1 of min and max :13 %
Bedroom/Bathroom +/-2 of min and max :6 %
Bedroom/Bathroom without min or max and exact match : 20%
Bedroom/Bathroom without min and max and +/-1 : 13%
Bedroom/Bathroom without min and max and +/-2 : 6%

Approach 1: Naive

For each new search requirements, I can compare that will each property in properties table. Calculate the score against each property. Will sort the list of properties based on scoring. And will return the properties in sorted manner whose score is greater that 40%.
Pros : Works well for smaller set
Cons : Not scalable

Approach 2: Elastic Search

I will keep the properties which are available from properties table in elastic DB. : **AvailableProperties**

Will do geo distance query on available properties set with search criteria of distance(+/- 10 miles). This will give me a smaller property set : **PropSetWithDistance**
Now, I will filter on PropSetWithDistance with search criteria of budget(+/-25%) : **PropSetWithBudget**
Finally, I will filter properSetWithBudget with search criteria of bedroom(+/-2) and bathroom(+/-2). That will give me **FinalPropSet** compliance with all validity requirements.

Now, I will calculate score for all properties in FinalPropSet and will sort that in ascending order. Will take properties whose score is greater that 40% and this will final result of all properties set with given search requirements.
Pros: Scalable, Faster, Maintainable

Assumption

- Minimum no. of bedroom : 1
- Minimum no. of bathroom : 1

Cases to Consider

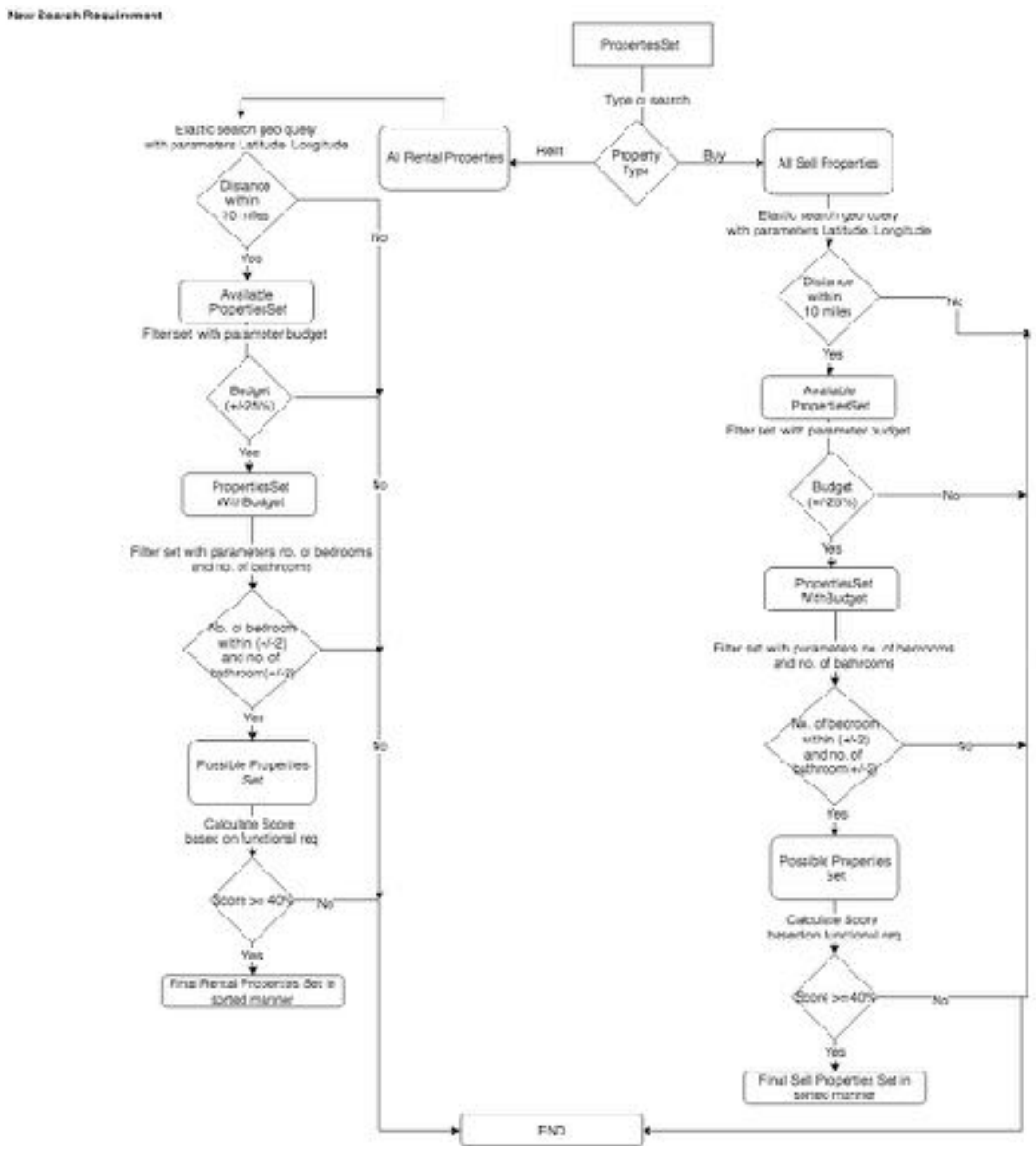
1. New Search Requirement

- TotalProperties : All properties in property table
- AvailablePropertiesSetSale : Available properties from TotalProperties which are of Sale Type. Will store this set in elastic search.
- AvailablePropertiesSetRent : Available properties from TotalProperties which are of Rental Type. Will store this set in elastic search.

For Sale Type properties

1. Will do geo distance query of elastic search on available properties set (AvailablePropertiesSetSale) with search criteria of distance(+/- 10 miles). This will give me a smaller property set : PropSetWithDistance
2. Now, I will do query on PropSetWithDistance with search criteria of budget(+/-25%) : PropSetWithBudget
3. Finally, I will query properSetWithBudget with search criteria of bedroom(+/-2) and bathroom(+/-2) :
4. FinalPropSet : This is the final properties set compliance with all validity requirements.
5. Now, I will calculate score for all properties in FinalPropSet and will sort that in ascending order.
6. Will take properties whose score is greater that 40% and this will final result of all properties set with given search requirements. Will store the list of propertIds with matching criteria in “ListofProperties” column of requirement table.

7. We will also store requirement in elastic search, so that we can quickly find all the requirements which newly added property should satisfies.


New Search Requirement
PropertiesSet
Type of search
Property Type
Rent
Buy
All Rental Properties
All Sell Properties
Elastic search geo query with parameters Latitude Longitude
Distance within 10 miles
Yes
No
Available PropertiesSet
Filterset with parameter budget
Budget (+/-25%)
PropertiesSet With Budget
Filter set with parameters no. of bedrooms and no. of bathrooms
No. of bedroom within (+/-2) and no of bathroom(+/-2)
Possible Properties Set
Calculate Score based on functional req
Score >= 40%
Final Rental Properties Set in sorted manner
Final Sell Properties Set in sorted manner
END

2. Update Search Requirement
We will already have list of matching propertIds in “ListofProperties” column. Will remove that the old list of properties and call the same function for new search requirement. This will update new list of propertIds.

3. Requirement Status Update
When the requirement is fulfilled it’s status will be changed to stale

4. Add new property
Will add new property in properties table. Initially it’s status would be available and it will be added to elastic search. We will find all the requirements it satisfies by looking up in elastic search. (same procedure as looking up properties for a requirement).

5. Update Property Status

When the property is being sold then it’s status will be changed to Sold. Also, will remove it from elastic search collection. Similarly, will remove a requirement from elastic search after it has become stale.

Low Level Design

```
enum PropType
{
        SALE;
        RENTAL;
};

enum ReqType
{
        BUY;
        RENTER;
};


Class Properties{
        int Id,
        float Latitude,
        float Longitude,
        long double Price,
        int Number of bedrooms,
        int Number of bathrooms
        PropType Type
}

Class Requirements{
        int Id,
        float Latitude,
        float Longitude,
        long double Price,
        int Number of bedrooms,
        int Number of bathrooms,
        ReqType  Type,
        List<Properties> properties
}
```

Helper Functions

1. float scoring(Properties Prop, Requirements req)
   This will return the score percentage of a property Prop for a given requirement req.
2. List<Properties> search_requirements(Requirements req)
   This will return list of properties with matching criteria for given requirement req.

NOTE:
This is just the high level design having algorithm logic. To have full fledged application other modules are also required like user login(agent, user) which I hadn’t considered for this document.