Problem Statement

Develop a simple suite of interfaces for a **“contact book app”**

Functional Requirements

- The System needs to support adding/editing/deleting contacts.
- Allow search by name and email address. Search should support pagination and return 10 contacts by default per invocation.
- Each contact should have unique email address.
- Add basic authentication for the system

Non Functional Requirements

- The system needs to “**scale out”** for millions of Contacts per Contact Book.
- Also the system needs to be extensible in terms of adding new attributes for each contact of a Contact Book.

Problem Analysis and Deriving additional NFR’S

From the Problem statement I understood that Contact Book System allows End Users/Customers to manage their Contacts. Each End User/Owner will be able to create/update/delete and search for the contacts which he/she created. They should not be able to access/view the other end-user’s contacts.

Based on this, one more additional NFR which can be derived is

- “System needs to be Multi Tenant”.

Also as we need to scale out for millions of Contacts per Contact Book of Customer, which actually means

- we need to make our system more distributed and scale horizontally

Also addition of new fields for each Contact should be done easily without downtime if possible.

- System needs to be extensible

Data Modelling

This section explains how to Model Contact.

In the system each Contact Book has a Owner(ContactBook Owner) who can do operations like

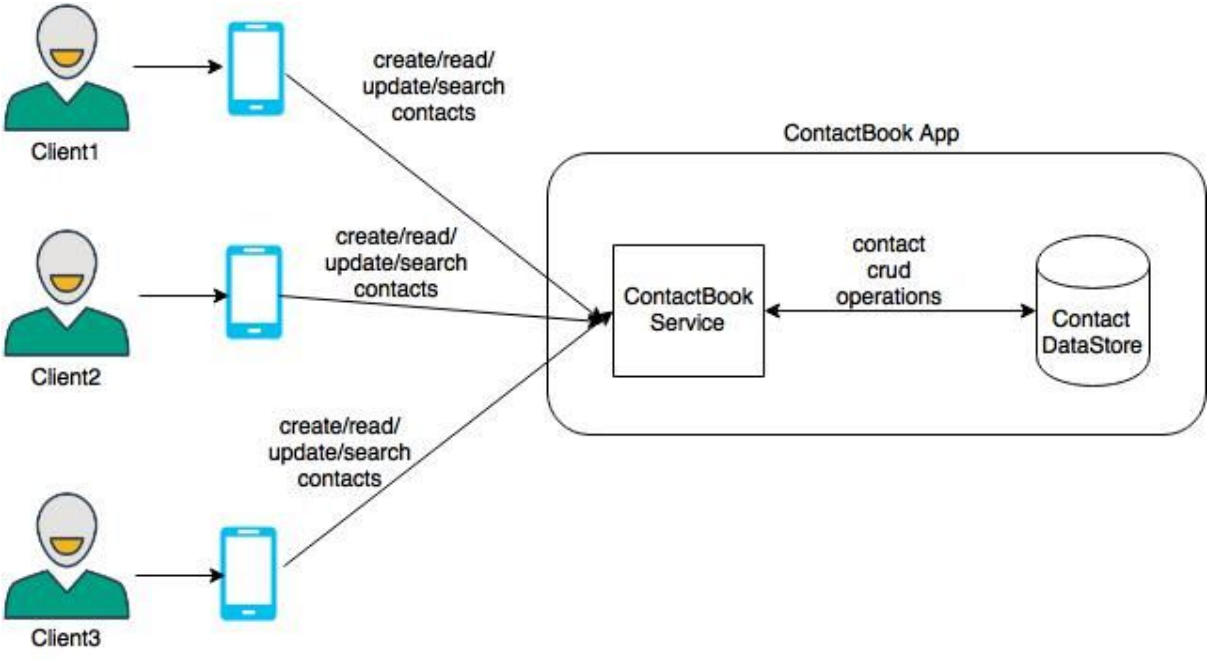
- Create New Contact
- Update Existing Contact
- Delete Existing Contact
- Search For Contact

Also every contact which the contact owner creates will have two attributes like

- ContactName
- ContactEmail

This is how we can model the Contact Class

High Level System Design



ContactBook Service

This service will allow to do lifecycle operations on contact creation/updation/deletion and search operations.

Contact DataStore

This datastore masters the contact info for all the contacts created by multiple Customers via their ContactBook App.

In order to make the contact datastore multi tenant, and restrict access of contacts only to the ones the contact Owner created.

Here is the approach I have taken from multi tenancy perspective.

Every contact is also tagged with the owner info as well.

This way whenever we create/update/delete/search for contacts, we will do based on the realm the Contact Owner is belonging to.

Also as Contact Owner searches for his contacts in his contactBook based on contactName and contactEmail, we can create unique constraint on combination of contact Book Owner and contactEmail.

This way each owner of a contactBook can have many contacts. But can have one email assigned to only of his contacts. Contact Email can't be duplicated across contacts as per business requirements.

Here we need to make our system distributed and also use distributed data store for storing/managing contacts.

The reason why relational databases are not right candidates for this kind of problems are

1) Relational Databases can scale up only. We can't scale out. Also when it comes to scale, they can't scale to millions of Users.
2) Also if we go with Relational databases schema also will be a problem as adding of new columns will be a problem if we have for change millions of contacts without having downtime.

Hence the need for distributed data Store like MongoDB

Why MongoDB ?

MongoDB is a distributed Document Store where we can store each contact related info as a separate document.

Also as its nosql and schemaless, adding of new fields for each contact would be easy.

And coming to scale perspective, it can handle huge amounts of scale and also its horizontally scalable by creating proper replica set cluster.

Implementation Details

Here I used Spring Boot Framework in Java.

Using Spring Boot we can quickly develop our micro services and also spring boot+mongodb support is so good for supporting operations like pagination etc with very minimal code changes.

Conclusion

System in its current state is supporting the above asked requirements(both functional and nonfunctional requirements). We can make it more smart in the future by adding additional attributes like contact's social details etc.