# High Level Design for Match-Filter app

## Problem Statement:

To design an application which allows users to filter on a prebuilt set of filters (using server side filtering).

More details on the GitHub page -<https://github.com/sparknetworks/coding_exercises_options/tree/master/filtering_matches>

## Design Assumptions:

1. User Login functionality and associated security is not in scope of this application
2. Persistence of data in a data store outside the application scope is not a requirement
3. Two separate modules – in the lines of Microservices concepts are being designed for Back-end and Front-end features

## Design Propositions/Justifications:

1. For back-end server,
   1. a spring-boot-JAVA application with
      * + spring-boot provides good integration with embedded application servers and databases, alongside injectable rest-full services and dependency management and an extensive test-bed, alongside all the goodness of Spring.
   2. Embedded Tomcat
      * + Out of box web application server container
   3. Embedded -MondoDB
      * + Easily configurable data-store available in application lifecycle
        + Mongo was chosen for its inherent support of Geo-spatial queries
2. For front-end,
   1. A single-page-Angular application running on Node Server has been designed.
      * + The choice of Angular has been due to some pre-existing exposure of this developer
        + Angular supports a large component library community, of which PrimeNG Component Library has been leveraged.
        + Front-end MV testability is key driving reason.

# Detail Level Design for Match-Filter app

## Back-end Server app:

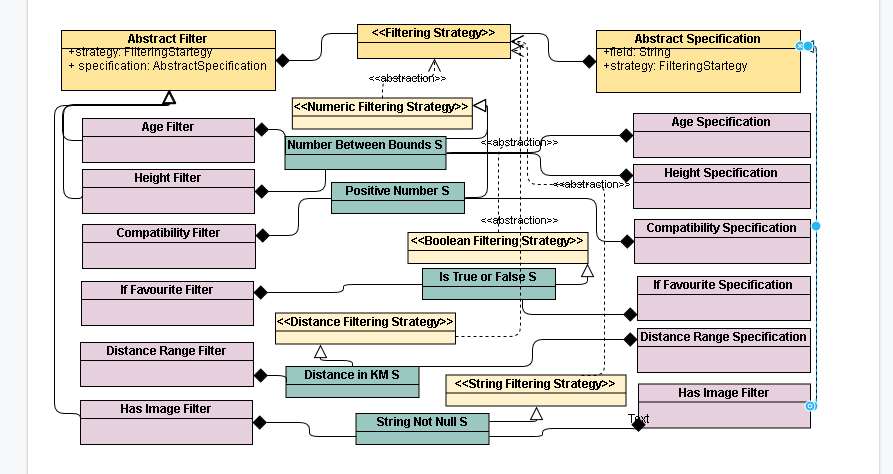
The core logic is built around the concepts of extendibility guided by:

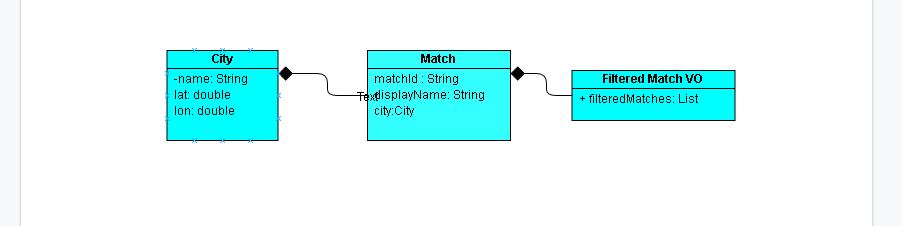
* + - * SOLID principles of software design
      * Strategy design pattern
      * Modularity and reduced coupling between layers (UI, Service, DataStore)

The principle Classes performing the core task has been represented in a class diagram below.

The members of the diagram are –

* Filters ( Classes defining the contract between UI and Server )
* Strategies ( Classes defining different strategies for filtering, e.g. – *NumberIsGreaterThanStartegy*, *IsTrueOrFalseStrategy, etc.*)
* Specifications ( Classes churning out Criteria to be applied while querying DB. They have a *Strategy* to work with )



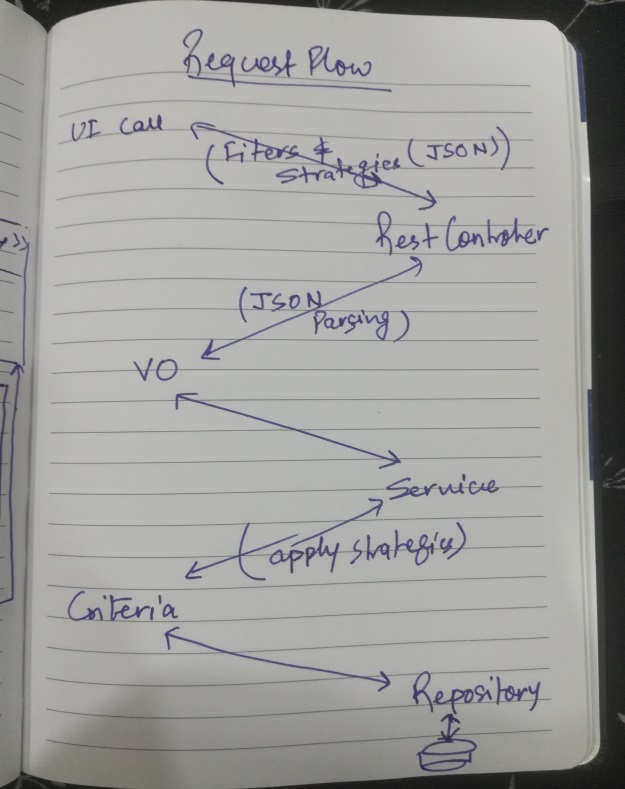
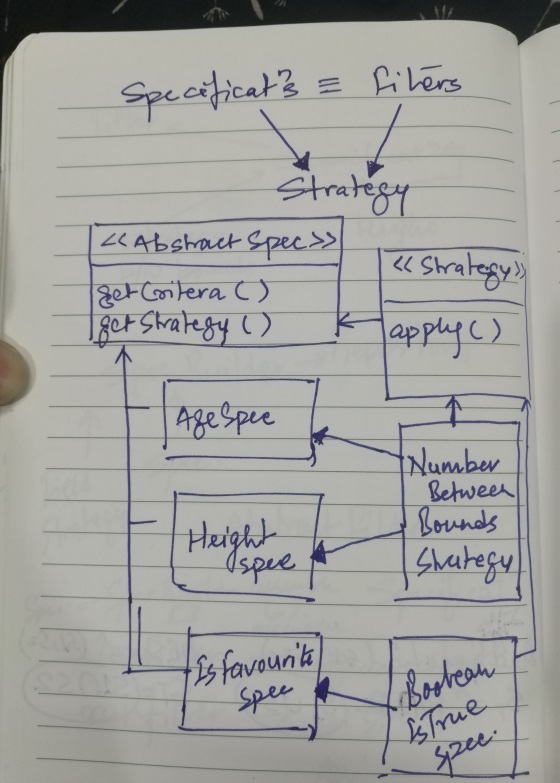
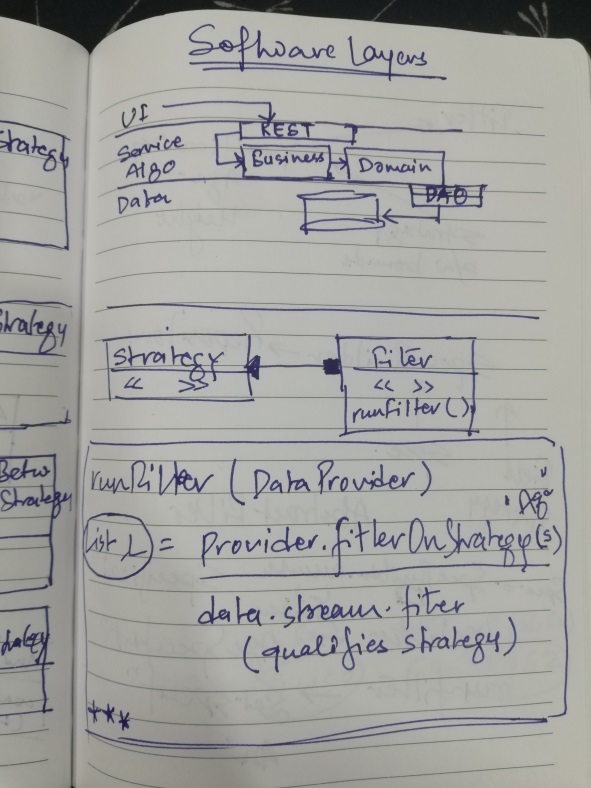
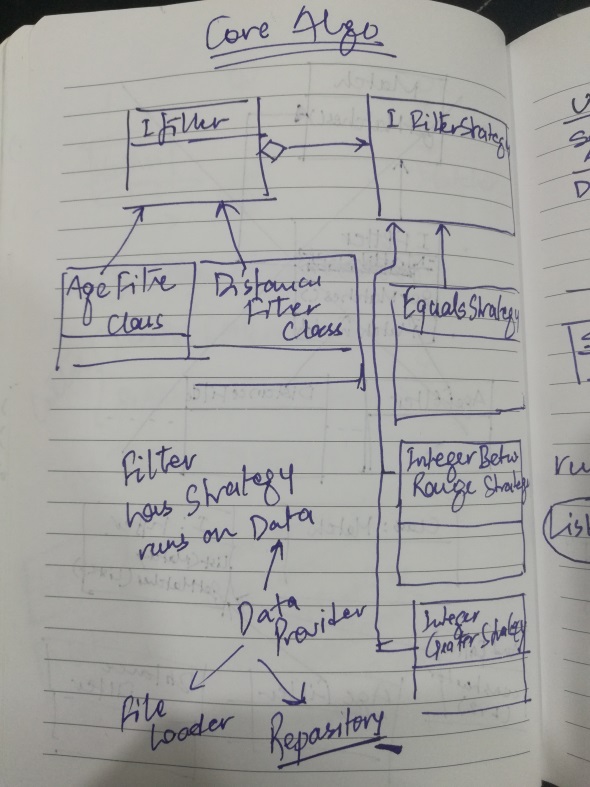


The backend server exposes rest-full web service to be consumed by the client.

The services and repositories are spring managed beans with the repositories extending spring-data capabilities and thus provide an abstraction from the underlying DB.

The DB is an embedded mongo instance starting up mongod process and which is populated with initial data using a Jackson Repository Populator wired to the Spring container. The database is capable of running Geo-spatial queries once the 2D-Speherical indexing is applied.

## Initial Design Scribbles:



## Limitations and extension Scope:

1. The Filter and Specification layers are apperantly parallel in nature and should be merged, if possible
2. Theoretically a dataprovider bean may be injected into a Filter or a Specification calling a common apply method and then runnign a polymorphic execute method to render filtered data. This dataprovider can be anactual list of data or a proxy on repository in itself.
3. Distacne Strategy behaves differently than all other strategies as Spring-Mongo implementations or Query and NearQuery are different.
4. The Strategy and Filter classes are created on the fly by Jackson Object Mapper, hence here seems to be no use of making them Spring-managed, even on prootype or request scope. This couldbe further looked into.
5. Ui is basic. Componatents can be enhanced, styled further.
6. UI test-bed can be expanded.