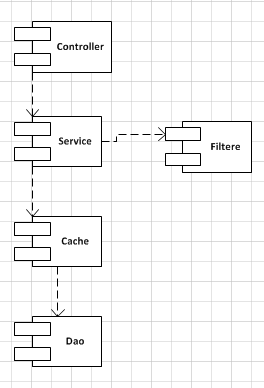
# Task Description and Design

By: Reza Ghafari - 03/Feb/17

# High Level Component View



## Controller

The component that receives the initial request from clients, applies some validation to it and calls the backend service.

## Service

This backend have been isolated from controller specific technologies or REST concepts. The Service obtains the full list of airports from the Cache and applies filtering on them based on the request.

## Cache

A simple cache using static variable has been implemented. A caching library could be used in order to have better control on airport data such as TTL in case data gets refreshed, etc.

## Dao

This component is responsible for reading the downstream system and gets the list of airports. It is not aware of any caching mechanism as that is done in a higher level. More rigorous validation could be done around validation of airport data coming back from downstream. Currently I assumed data is syntactically and semantically correct and sane.

# API Design

Normally query parameters are best for filtering data. This is the approach that has been chosen for this task. There are other options available which have their merits and problems as follows below.

## GET with message body

We could send a GET request with the request parameters and JSON or XML in the body of the request. This is not a normal way of using GET requests.

## POST with body

POST could be used with request parameters in the body. As per definition POST doesn’t really suggest that a change in the backend should be applied but again this is not a widespread way of implementing a filter.

## URL search

This could also be the second good option after query parameters. Example requests could have been like:

/search/country/<country>

/search/country/<country>/code/<code>

/search/country/<country>/code/<code>/intl/<true|false>

Again I believe for search and filtering the query parameters make more sense and easier to follow.

## HATEOAS

Possibly it’s not a useful feature for a search. It could have its own use cases if there is a requirement for it and clients are intelligent enough.

# API Documentation

http://<host:port>/search

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Query Param** | **Description** | **Default Value** | **Mandatory** | **Values** |
| country | The county code or display name | Empty | No | String |
| code | The airport code | Empty | No | String |
| Intl | Whether it’s an international airport or not | Empty | No | True/False or 1/0 |
| domestic | Whether it’s a domestic airport or not | Empty | No | True/False or 1/0 |

Example:

|  |  |
| --- | --- |
| /search?country=au&code=xyz&intl=false | List of airports that their country code or country display name is “au” AND airport code is “xyz” AND is not an international airport |
| /search | Return all the airports |

## Design Notes

* The country is used query parameter check both country code and display name.
* Search is case in-sensitive
* International/domestic could have been grouped in a “type” field instead but this also depends on the requirements
* If no query param specified all airports will be returned
* Query params perform as logical AND

# Error Handling

The API has been designed to be quite forgiving for simplicity. I think this goes back to the actual requirements and type of the client. We could throw different types of exceptions from backend and translation them into HTTP error codes.

# Technology Choice

* Springboot has been used as it’s very easy to prototype Java applications.
* Maven 3.3.9 has been used for building.
* Developed and tested with java 8 on Eclipse IDE.

# How to Execute

* To generate binary run mvn clean install

You would require internet connection, maven 3 and java 8 installed

* Run the binary using java -jar airport-filter-0.1.0.jar --spring.profiles.active=dev

It runs on port 9090. It can be changed in the property files.