**Design Document**

Design requirements

1. Code should be reusable, scalable
2. Hardcoding not allowed. Configuration to be managed through XMLs, property files

Design considerations

1. Make use of open source framework like Spring MVC and any JavaScript.
2. Refer to any online Weather API to get source data.

Proposed Solution

Technology stack

1. JDK 1.7
2. Spring MVC
3. AngularJS, Bootstrap.css
4. Apache Http client
5. Build tool - Maven
6. Webserver – Tomcat

Design Approach

* We can leverage the capability of Spring MVC as server side component and AngularJS as the front end component.
* The solution involves two RESTful services built in SpringMVC :

1. GET service that would return a list of cities to be shown on the UI
2. POST service that would take a parameter from the selected city and call the Weather API to get the relevant weather information.
3. Data exchange format to be used is Json.

* The 2 RESTful services would be exposed by a Spring MVC Controller class that would be the entry point for all UI calls.
* The Controller class called *WeatherController.java* would have 2 methods
  + - getCities() that returns a List of cities
    - getCityWeather(cityId) that returns weather information for the selected city.
* The WeatherController calls methods defined in *WeatherService.java* interface which are implemented by *WeatherServiceImpl.java*
* The WeatherAPI used is provided by the website <http://openweathermap.org/api>
* The website recommends the use of *cityId* defined in their city list json file to get accurate results for weather information. See here for city list <http://bulk.openweathermap.org/sample/>
* The cityId and cityName is configured as a Map in the weatherDispatcher-servlet.xml as shown

<util:map id=*"citiesMap"* key-type=*"java.lang.String"* value-type=*"java.lang.String"*>

<entry key=*"2147714"* value=*"Sydney"* />

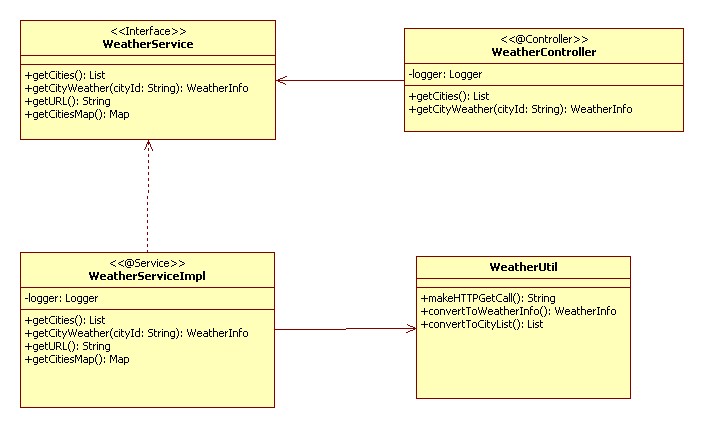
<entry key=*"2158177"* value=*"Melbourne"* />

<entry key=*"7839791"* value=*"Wollongong"* />

</util:map>

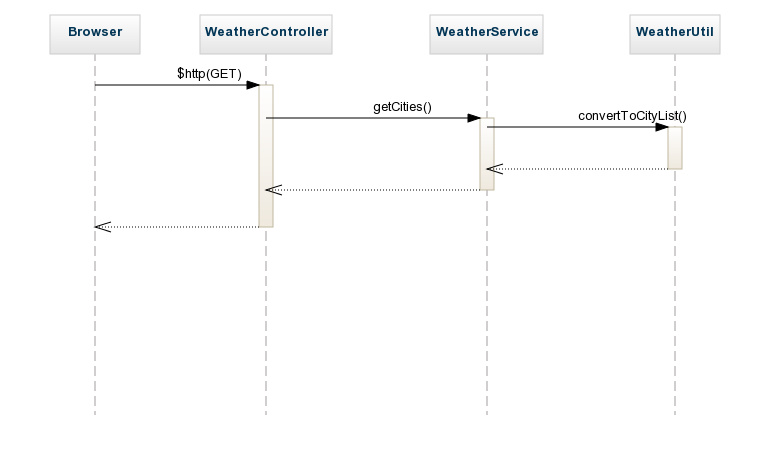
* This gives us the flexibility of adding/removing cities in future without code modifications
* The WeatherServiceImpl.java class reads this Map and builds a List for cities to be returned to the WeatherController class.
* For getting weather information, the WeatherServiceImpl.java calls a utility method makeHTTPGetCall()from *WeatherServiceUtil.java*
* The WeatherServiceUtil.java has a method to make a Http GET call by using Apache HTTP client which invokes the WeatherAPI URL
* The WeatherAPI URL is configured in *weatherservice.properties* file which gives the flexibility of updating the file in future without code modifications.
* The city List data and weather information data is converted to *json* format using Jackson mapper before sending to AngularJS.
* Logging – The logging configuration is mentioned in log4j.xml that is present on classpath
* Exception Handling – All exceptions are caught in the Controller and logged as errors.

Class Diagrams



Sequence Diagrams

Sequence Diagram for getting list of cities



Sequence Diagram for getting weather information

