**Weather Service**

**All code challenges should be hosted on a publicly accessible git repository (i.e. GitHub, BitBucket, etc).**

* Uploaded to github, https://github.com/alewi13/WeatherServiceDemo

**The business wants a web service developed that returns current wind conditions based on a user’s current zipcode.**

**Constraints**

**·         Framework:**[**Spring Boot**](https://projects.spring.io/spring-boot/)

* Implemented in Spring Boot using a starter project on spring.io

**·         Dependency Manager:**[**Gradle**](https://spring.io/guides/gs/gradle/)**or**[**Maven**](https://maven.apache.org/what-is-maven.html)

* I went with Maven. I built the project from the sts eclipse IDE but you should be able to run it with stock maven although I haven’t tested this

**·         Test Framework:**[**JUnit**](https://junit.org/junit5/)**and**[**Mockito**](http://site.mockito.org/)**/[Powermock](http://powermock.github.io/" \t "_blank)**

* Unfortunately this is the only part I didn’t get fully done. I was called into work on Sunday but was still able to get some base unit testing on the controller including the returned JSON params

**·         Coding Style Guide:**[**Google Java Style Guide**](https://google.github.io/styleguide/javaguide.html)

**Design Goals**

**·         Bind your weather client to an interface, and not to the concrete implementation of the OpenWeatherMap API.**

* Completed, there is a WindService interface for the WindServiceOpenWeatherMap implementation

**·         Implement the cache handler as a decorator for the weather client.**

* Completed, there is a WindCache interface with a simple WindCacheSimple class. I then created a WindCacheDecorator class and WindCacheHashMap using the decorator design pattern

**·         Bind services to an interface (not an implementation) in the service container.**

* Completed, there is a WindService interface for the WindServiceOpenWeatherMap implementation

**Functional Requirements**

**·         Consume weather data from**[**https://openweathermap.org/**](https://openweathermap.org/)**.**

* Completed, this is implemented in WindServiceOpenWeatherMap

**·         Provide an HTTP GET /wind/{zipCode} method that takes a zipcode as a required path parameter and returns a wind resource.**

* Implemented in WeatherController and wired automatically with Spring Boot

**·         Validates input data.**

* Validation is inside the wind model. Validation is done using a regex, please see my comments for other possible ways I could have done this.

**·         Response format should be JSON.**

* Response is JSON, example: {"zip":"89110","speed":4.6,"degrees":130.0,"gust":0.0,"refreshed":"2018-10-01T17:38:54.550+0000"}

**·         Cache the resource for 15 minutes to avoid expensive calls to the OpenWeatherMap API.**

* Caching is handled by the WindCacheHapmap class which is utilized by the service. If a record exists and is <15 min old it is used. Otherwise it is pulled via the weather API and the cache is updated with a new timestamp

**·         Provide a CLI command that will bust the cache if needed.**

* I created another service call for this, the cache can be cleared with “curl http://localhost:8080/api/v1/wind/clearcache”

**·         Ensure that the cache is thread safe.**

* Please see my full explanation in WindCacheHashMap but I believe this is thread safe by design using Spring’s single DispatherServlet and spring beans being Singleton by default.

**·         Response fields should include:**

**o    Wind Speed**

**o    Wind Direction**

* My response includes that and more, ex: {"zip":"89110","speed":4.6,"degrees":130.0,"gust":0.0,"refreshed":"2018-10-01T17:38:54.550+0000"}

**Unit Testing Requirements**

**·         Use mock responses for the OpenWeatherMap API.**

**·         Use mocks when interacting with the cache layer.**

* I was a bit crunched for time due to current work responsibilities but did include some basic JUnit testing on my controller including testing on my JSON responses.

**How To Run**

**1.       Clone the repository.**

**2.       Build project and generate the JAR.**

**3.       Execute project JAR.**

**4.       The wind resource should now be accessible by running a curl command:**

|  |
| --- |
| **$ curl -x**[**http://localhost:8080/api/v1/wind/89101**](http://localhost:8080/api/v1/wind/89101) |
|  |

* I wasn’t familiar with the “x” switch but I looked it up and that’s for specifying a proxy. I don’t run through a proxy but “curl  <http://localhost:8080/api/v1/wind/89101>” without the -x works fine for me.
* This all works as asked, there is also a <http://localhost:8080/api/v1/wind/clearcache> url you can navigate to and will get a simple response or simply use curl.