Sam Ralston OpenWeather API Consumer Test

***Overview***

My application contains the elements of the typical Spring Boot architecture. That is, I have a Controller-Service-Repository pattern with those distinctive layers. In addition, I also have a model layer that specifies the entities.

***To Use***
Start the Spring Boot application. The build tool is Maven. Once online, the endpoint can be reached at http://localhost:8080/{city}/{apikey}. Where api key is either eddykey, tommykey, anotherkey, howmanymore, lastkey.

***The Controller Layer***
The controller layer is very simple. It contains only one controller, the WeatherController, which handles a single HTTP request at the endpoint '/{city}/{apikey}'. This controller is annotated with @RestController to add web requests handling functionality. The controller also uses the @Autowired annotation to inject the service dependency that is used in the '/{city}/{apikey}' endpoint to getWeatherByCity().

***The Service Layer***
The service layer contains two services, the WeatherService and the ApiKeyService.

The WeatherService is responsible for retrieving the data required by the WeatherController. In this case, I currently only have one method, getWeatherByCity, that performs a couple of actions. These are:

1. Validates the API Key used. If not valid, returns a failure result.
2. Checks to see if the request is cached by the H2 Database. If so, returns a success result
3. If the request is not cached by H2, WebClient will make a GET request to retrieve the needed data.
4. Cache new data in H2 Database
5. Returns a success result.

This method makes use of the apiKeyService service and Weather repository to function.

***The Repository and Model Layers***
The WeatherRepo repository is an interface that extends JpaRepository and receives the type of entity of Weather (located in Models). JpaRepository has been configured with Hibernate and H2 libraries, making it easy to add new data using the Weather schema/entity an easy process. I have extended this interface to simplify data retrieval from the database by adding a findByCity
method that JpaRepository automatically uses to create a query that retrieves the row based on the city field.

***Dependency Injection***
The following dependencies have been used for this project:

- Spring Boot
- Spring Data JPA
- H2 Database (includes the runtime)
- Webflux

***Caveats***

Due to time constraints, there are a few caveats that need to be addressed.

- While API control is implemented, I was not able to add in the ability for ApiKeyService to observe time. Therefore, the keys currently will not reset after an hour and will be permanently disabled (or until the JVM is restarted).
- I have hardcoded the API key for OpenWeather for convenience of running the application. This should normally be be stored in an environment variable for security.
- The city name must be correct otherwise the API will fail. Some basic error handling should be implemented.
- The application returns the ID for the weather location rather than the description. This is due to a matching issue I had with Jackson and JpaRepository mapping.