# Weather Restful APIs Design

This document explains the design choices used in this project.

The challenge requested a Spring Boot, H2 DB backed REST service. That’s a standard Spring use case, and I’ve mostly used the standard Spring design as a result (Starters, Controller-Service-DAO Spring layers). My main difference is defining the service API via OpenAPI 3/Swagger, and using OpenApi to autogenerate Java stubs. This saves developer boilerplate. More importantly, it makes the service more extensible for future API changes. The api-spec.yaml file makes the API definition human-readable and easy to change. The autogenerated code automatically does a lot of the validation - like checking that the required GET query parameters (city, country) are the right format and are present in the request. I’d prefer to use Ebean for the DAO, but I’m not sure how familiar you are with it, so I’m using Spring JPA.

My workplace recently moved to Java 17, Spring Boot 3, but for this challenge I’m using Java 11, Spring Boot 2. Some libraries haven’t caught up with Java 17 (e.g. Jakarta changes), so to prevent getting stuck in those sorts of problems, I’m using the older version.

I ran out of time to implement the API rate limiting. In my work, we would always use something like AWS API Gateway or other cloud services to set up rate limits for our apps. There are complications with doing it in app - if you need to scale to multiple Spring Boot instances how can you limit the API across instances? So I admit I don’t have experience in this. I could have followed a tutorial to add rate limiting to Spring Boot (e.g. using Bucket4j - <https://www.section.io/engineering-education/implement-rate-limiting-in-spring-boot/> ). But I’m not sure you’d be interested in that and how much value it’d provide.

Otherwise, I’ve implemented the other test requirements to some level: Spring service’s API, API calls to OpenWeather, and H2 backing. With the JUnit tests, I hope this exercise gives a representative view of my Spring coding knowledge in a short timeframe. With reference to the coding challenge objectives, the following was achieved **(in bold**):

1. Enforce API Key rate limiting scheme. **Not implemented**
2. Reject requests with invalid input or missing API Keys. **Requests rejected with missing or invalid input (city or country fields), taking advantage of OpenAPI autogenerated code autogenerated validation**
3. Store the data from openweathermap.org into H2 DB. **Implemented**
4. The API will query the data from H2 **Implemented via JPA**
5. Clear Spring Layers are needed.  **Implemented - Controller, Service, Repository**
6. Follow Rest API convention. **Implemented**
7. Demonstrate the skills of writing different types of test cases/TDD Discipline. **Implemented - a small sample of tests demonstrating JUnit unit tests, SpringBootTest tests, and Mockito (I could have gone further and used SpyBeans)**