# Word counter application

Implementation:

* Created an interface exposing two methods enterWords(words) and getNumberOfOccurrencesForWord(word)
* Created a implementation class that implements the before mentioned interface and its methods
* The enterWords method takes care of any whitespaces on the ends of the word entered, converts multiple spaces between words into one space and splits the string into words in case the entered string is a multi word string (“like this”) will be injected into the map with the keys “like” and “this”.
* Created a Translator Singleton class since I’m assuming I’ll be using its translate method to translate the word from a certain foreign word to english. (Mocked the functionality of it to return the actual word provided and to “translate” one word from Serbian to English in the implementation itself)
* Each instance of counter has its own ConcurrentHashMap<String, Integer> which is mainly used because it’s thread safe and has locking operations when updating the keys and their values.
* Upon analyzing . destructuring the words into terms and translating the words I inject the translated words into the before mentioned map as a key using parallelStream() which creates a stream which runs on multiple cores (threads) which makes the operation much faster for a larger entry of words
* If the key isn’t present in the map I default the number of its occurences to one since it’s the first showing of that word in the map, otherwise I retrieve the number of occurrences and add 1 to it and put it back in the map under the same key value so the value gets overriden.
* When invoking the getNumberOfOccurrences method the user provides a word which he wants to get the number of occurrences for and the method returns a 0 if the word is not found and the actual number of occurrences if the word is found

In this implementation of the library, because it wasn’t stated otherwise, each user accesses the same map, so multiple users will modify the same map and read from it. If there is a need for a user to have his own map, I’d probably create a map of concurrent maps which key is a unique ID and the value is the concurrent map assigned to a specific user. Each user would then need to, not knowingly send the unique id of the map assigned to them to access his own word counter map to then test against the number of occurrences or add new words.

Local setup:

I’ve created a git repository where I’ve published both the microservice and the library itself. (link to this is: ) . To start it locally you’d need to go to the library’s root folder and do a mvn clean install, that will create the library artifact in your local .m2 folder and you will be able to reference it from the microservice. All you need to do in the microservice is to refresh the dependencies for it to find the artifact locally and inject it in the libraries the spring boot app is using.

Cloud setup:

In a cloud like setup, I’d create repositories for both the microservice and the library since the library is maintained by us. I’d create an adequate CI/CD pipeline to run tests, create an artifact out of the library and push the artifact onto a artifact repository (I’ve done something similar in an Azure environment with Azure Artifacts) . The code changes to the library would inflict a semantical version change and a manual change of version on the dependent microservice would need to be done.

The microservice itself would need to know where the artifact is stored (it can be on the maven central repo or our own artifact repo like Azure Artifacts through the <repositories> section of pom.xml) and it should use the version annotated in the <dependencies> section of the pom.xml file.

Architecture solution:

If the microservice is in a multiple microservice environment, there will probably be a need to have a API Gateway that is used as a sort of middleware. The logic in the API Gateway is for the developers discretion and usually mechanisms for authentication and authorization are implemented here. Here we can implement the authentication to see if the user trying to use the services of our library is authorized to do so (whether it being through a JWT or an API key + secret combination).

There will possibly be a problem if you’re trying to horizontally scale the microservice. With the current implementation, because it wasn’t stated otherwise, every microservice has its own central concurrent map. If we instantiate the microservice multiple times to assure availability, then each microservice will have it’s own map which will then provide non-consistent results to the user since the user could be entering words for one microservice through the first request, and reading the map from another microservice he didn’t enter words for.