Overview

We'd like you to create a RESTful web service in Java. Please read the instruction below carefully. The service you develop should be deployable by a standard WAR file.

Requirements

* The service has two endpoints.
* One is to take a Unicode string as a parameter and register it in the service
* The other is to take a string id and return all registered strings that match with the string id
* String id is a number calculated as follows:
* String id is a sum of each character's id A character id is a sum of the current and previous character's Unicode code point
* The character id of the first character in the string is the character's Unicode code point If the current and previous characters are the same, the character id is the current character's Unicode code point String id must be calculated without using a loop Example: "abc" => 97 + (97 + 98) + (98 + 99) "abbc" => 97 + (97 + 98) + (98) + (98 + 99)
* The registered strings must be persisted so that the strings are still available after a reboot Use a file for the persistent medium For easier troubleshooting, the file must be a text file The REST APIs accept and produce JSON
* The service doesn't have to be designed to work reliably in the clustered environment, but it must be reliable as a single node service.
* The code must be thoroughly tested by unit tests Deliverables Source code Build and deployment instruction REST API documentation If you have a github account, you can create a repo there. Otherwise send us your deliverables in a zip file We expect a production grade code, not a throw-away test code

**Design for Registration and search**

API Name – RegisterAndSearchString

**urI:-**

/RegisterAndSearchString/rest/RegisterAndSearchService

**Service End Points: -**

/RegisterAndSearchString/rest/RegisterAndSearchService/register

/RegisterAndSearchString/rest/RegisterAndSearchService/search

**Method**

POST

**String Registration: -**

The client makes a POST call to service endpoint. It sends JSON data with the string to be registered to service endpoint.

The control transfers to the web service method **registerString(String).** This method is designed to consume and produce JSON data.

**String Search: -**

The client makes a POST call to service endpoint. It sends JSON data with the string Id to be searched to service endpoint.

The control transfers to the web service method **getStringByStringId(String).** This method is designed to consume and produce JSON data.

**Business Logic: -**

A static Hashmap<Integer, HashSet<String>> is introduced to keep track of StringId (Integer which will be used in search) and unique set of strings corresponding to this stringId in HashSet.

It serves 2 purposes: -

1. During registration, if we pass on the same string to register, we can compute the string id and get the Hashset with list of strings in O(1) time complexity. If any of the strings in Hashset contains the string, we can return immediately saying, the string is already registered. This is again in O(1) time.
2. During Search, based on the string Id passed in input, we can check in HashMap in O(1) time to return the strings corresponding to the string id.
3. Caching can be enabled along with LAST\_MODIFIED\_DATE and ETAGS and return the response immediately as well during Search. (but the code isn’t implemented with this).

When the service is called, it checks whether HashMap is empty (which might be in case of re-boot or first call).

In that case, HashMap will be populated first reading the file level data and return the response.

1. If the string to be registered is a new String, the corresponding string Id will be calculated as follows for the requirements
2. Intuitively, if there are no duplicates in string, then the sum of the code points can be calculated for ‘abc’ as 2\*(codepoint of a+codepoint of b+ codepoint of c) – codepoint of c.

This can be proved by linear algebra like below: -

**abc = a+(a+b)+(b+c) = 2a + 2b + c**

If there are duplicates and the characters are strictly in increasing order, for eg. abbc, here b is repeated only after a followed by c.

We can prove by linear algebra that

**Sum of code points of all characters – ((sum of code points of all characters – sum of**

**Code points of distinct characters)/2)**

Ie, for abbc

From above equation, sum of code points of all characters,

**2(a+b+b+c) –c = 2a+4b+c**

Sum of code points of distinct chars

2(a+b+c) –c = 2a+2b+c

Their difference

2a+4b+c – (2a+2b+c) = 2b

Divide by 2,

2b/2 = b

So, now use above formula

**2a+4b+c – b = 2a+3b+c (which is the right answer as per a+(a+b)+b+(b+c) )**

But if has duplicates and not increasing order, say, ‘abab’, we cannot use this equation. So, loop can be avoided using Java 8 foreach and Lambda for duplicate scenario.

1. For Search operation, the string Id is checked in HashMap and the corresponding strings in the HashSet are returned as JSON data. If HashMap is empty, it gets populated by reading the strings from file StringsRegister.txt. The value is returned then.

**Suggestions for Improvements**

1. Caching can be enabled along with **LAST\_MODIFIED\_DATE** and **ETAGS** and return the response immediately as well during Search.