**BBC CountMeUp Implementation notes**

This project contains my solution to the BBC CountMeUp technical challenge. This document gives details of how to call the webservice, the assumptions behind the implementation and my reasoning behind the technical decisions made.

**How to run call the CountMeUp webservice**

The webservice has been implemented using Spring Boot. This allows easy unit testing and this is the recommended method to test the provided scenario. The application should be opened in an appropriate IDE (tested using Eclipse). The CountMeUpUnitTests class can then be run as a unit test.

**Assumptions**

The provided scenario did not specify in what format the votes would be provided to the CountMeUp webservice. I have therefore assumed that a List of votes has been provided through a separate webservice outside the scope of the challenge but have not yet been processed. The unit test above manually supplies an ArrayList of the 10 million votes.

I have concentrated on the server-side part of the development. This is mainly due to a mixture of my relative lack of experience of front-end development and the fact that the challenge seemed mainly concerned with factors affected by server-side operations (such as performance and voter validation)

**Technical Decisions and Performance**

To minimize required boilerplate code I used Spring Boot to implement the webservice. As well as an efficient way to set up a webservice (including an embedded server) it allows easy unit testing.

Throughout development performance was a key concern. I therefore decided to use collections with good performance for retrieving and adding objects. However, once initial implementation was complete, I discovered that the performance of the application didn’t quite make fulfil the requirements. There were several options I considered for improving performance. One example considered was using multithreading to process different votes on different threads, however this was not possible as whether a vote is valid is dependent on whether the voter has voted three times before. Therefore the votes have to be processed synchronously.

After analysis of the performance bottlenecks it was identified that putting voter names into a HashMap was the key problem. This was surprising as normally operations such as this on a HashMap have a complexity of O(1) however there were several possible reasons why this could be the case:

* There could be a large number of HashCode collisions. It is unlikely that there would be more collisions than necessary as the String class has a well-refined HashCode.
* The HashMap size could need to be expanded many times. With such a large number of voters this was going to be the case. Therefore the starting size and load factor were changed to improve performance.
* The JVM could lack the memory necessary to easily process such a large HashMap. Therefore, in the JVM launch arguments the initial and maximum heap size were increased.

With these changes the performance greatly improved and was normally 0.9-1.1s dependent on the number of unique voters. This value is obviously highly dependent on the specifications of the machine the code is run on.