## **Functional Requirements**

The service should return a list of strings matching the prefix.

## **Non-Functional Requirements**

1. Latency should be low. The service should be able to deliver results in real-time, as the user types in the query.
2. The system should be scalable to millions of users.
3. The system should be highly available.

## **Data structure to store the words:**

Each server stores the words in a Trie tree and in memory. Refer: https://github.com/bhargMV/PrefixSearch/blob/main/main.go

## **Capacity Requirements**

Entire English dictionary has ~172000 words and average word length is 5 (10 bytes). Each server just need 10 MB memory for O(1) prefix query search of words.

## **Architecture:**

Assumptions:

1. The application servers are dedicated only to serve the prefix searches, and the amount of data that’s required to serve the queries fits in the memory of application servers.
2. Trie tree is already constructed and stored in application servers and DB.



**Availability** is addressed by replicated application servers. **Latency** is addressed by the load balancer forwarding the request to the least busy server and the O(1) query time.

**Scalability**: If the number of users grow, more application servers could be added. If the system needs to support any word, not just the word present in the English dictionary, Trie tree can be broken into subtrees and stored on different application server. For example, the Trie tree with all the words starting with A can go to server 1,2,3 and the Trie tree with all the words starting with B can go to server 4,5,6 etc. Configuration of all the servers is managed by Apache Zookeeper. Basically, these application servers are Zookeeper instances.

**Database:**

Trie tree is stored in a document store such as MongoDB and is replicated. Whenever a new application server is added or an application server is restarted after a crash, a copy of Trie tree is fetched from DB through Database servers. For high availability, there are multiple database servers in the system. Requests from application server to the database servers goes through a load balancer.

## **How to support adding new words?**

Application servers expose a new API to add word to the trie. Load balancer forwards the request to the appropriate application server to add the word based on the server load and starting character/characters. Application server updates the local trie and simultaneously updates the trie tree in the database through database server. Primary and secondary database are eventually synced for consistency.