High level description  
  
**1) Architecture:**  
The server runs 1 + N threads.  
a) Manager thread (1) - listens for incoming connections and passes on the request to the worker threads pool.  
b) N worker threads – handles the requests (configurable).

**2) General Design notes and considerations:**a) Perform everything that is possible before the server is up and running (for run time consideration).  
b) Use threads pool to save the creation/deletion of “run-time” threads.  
c) Use “thread-per-request” pattern for concurrent handling of requests.  
  
**3) Main objects of the server:**  
a) **HashDbManager** – contains all the implementation of:  
# extracting the words from the txt file.  
# store them in an efficient manner.  
# retrieve the list of matched words for a given word (which is generated from a similar?word=XXX request).  
  
The general idea is to hold two maps:  
- Map that maps between a word (string) to an hash code (integer value) - m\_wordToPermutationNumberMap  
- Map that maps between the hash code to the list of matched words (list<string>) - m\_permutationNumberToPermutationsListMap

# when constructing the maps, before taking the hash code for each word, **the word is sorted** (using std::sort) 🡪 thus causing each word that composed of the **exact same set of characters** to receive the **same hash code**.  
# Then all words with the same hash code are stored in a single list, within the second map, with their key being the hash code.

For example, given the following DB file = { apple, pelpa, bgc, cgb, aah}

\* The first map will be the following (<word :: hash\_code>):  
< aah :: 9491188875533077872>  
<apple :: 15385942197106809753>  
<bgc :: 13147583528078288608>  
<cgb :: 13147583528078288608>  
<pelpa :: 15385942197106809753>

\* The second map will be like so: <hash\_code :: [lsimilar\_words\_list]>  
<9491188875533077872 :: [aah]>

<13147583528078288608 :: [bgc, cgb]>

<15385942197106809753 :: [apple, pelpa]>  
  
b) **BasicSimilarWordsResolver** – responsible to retrieve from the *HashDbManager* the list of similar words for a given word.  
# According to the way that the list of similar words are being kept – all that it has to do is to (upon receiving the list of similar words for a given word) is to remove the word that was sent for the request (as it should not be returned).  
  
c) **BasicHttpRequestHandler** – executes the requests, as well as:  
# responsible to filter unsupported requests.  
# send the actual payload back according to their type (OK/Error).  
  
d) **BasicStatsRequestResolver** – responsible to retrieve the needed information for a stats request.  
# Also updates the respective indicators (average request handling time & number of requests) upon the a similar?word request.  
# Due to the fact that the functions that read & write to these indicators will run from different context (of different worker threads), mutual exclusion is desired here.  
  
  
**4) General assumption:**  
a) All the words in the dictionary are all lower case letters.  
b) All the words sent in a request similar?word will be as well lower case letters.