-----API Specifications------

- HTTP server that is interactive in browser
- JSON for sending and receiving data
- ZeroMQ for messaging between nodes (Allows for very large messages as well as automatic JSON encoding and decoding)
- Modifiable config.json file - Number of workers (Reducers and Mappers), Server IP Address, Ports for nodes

------Command Structure-------

Commands can be invoked by utilizing proper URL structure in the browser or by using curl: (URL Port and IP can be modified in the configuration file)

Methods:
  word-count:
    Required Flags: book - Book must exist in the "books" folder, book name given should be the same as the file name
    Optional Flags: specifier - Filtering response based on the given word
    Examples:
      http://localhost:8080/?method=word-count&book=russian-short-stories.txt

http://localhost:8080/?method=word-count&book=russian-short-stories.txt&specifier=ebook
  Inverted Index:
    Required Flags: directory - this directory should exist within the assignment 2 folder, only text files or no files allowed
    Optional Flags: specifier - Filtering response based on the given word
    Examples:
      http://localhost:8080/?method=word-count&directory=books2
      http://localhost:8080/?method=word-count&directory=books2&specifier=fox

------Process Communication------

Uses ZeroMQ utilizing JSON structure to communicate between nodes and the API over the network

- Partitioning
  Data is never stored in shared folders/files minus the books, the master node has access to this folder, which provides
  checks that ensure the book exists. Information is communicated between nodes using the ZeroMQ sockets, so in theory,
  these actions could be performed on separate devices.
- Message Formats

Messages are shared utilizing JSON, which allows for better classification and accessing of data within a message, for example:

```
{
    "method":"map word count",
    "task":"Hello, this is a test word count"
}
```

The above is a mock message sent from the master node to the worker nodes that are responsible for mapping.

```
{
    "method":"reduce word count ",
    "task":[{LIST OF FILE PATHS}]
}
```

The above is a mock message sent from the master node to the worker nodes that are responsible for reducing.

------Inner Workings------

1. webserver.py is ran
2. Master Node is created
3. Master Node creates number of workers (num_mappers, num_reducers)
4. The master node creates a ZMQ connection with each worker separately and stores all of these connections in a dictionary
5. The workers prepare themselves to receive request from master node
6. Master Node establishes ZeroMQ server and waits for request
7. webserver establishes connection with master node over ZeroMQ
8. webserver establishes it's own HTTP server and prepares to accept requests
9. When a GET request is made to the HTTP server, it parses the passed URL looking for the method called, and passes the given parameters to the proper handlers
10. These handlers then send the data to the Master Node over ZeroMQ
    1. This data will contain things like - The Method, the file queried, and specifiers on the method
11. The master node then parses the method and does either a word count on the given file or a Inverted index, behavior varies, but is similar to the following

    --------------WORD COUNT----------------

    1. Master Node prepares file from the specified file path
    2. Master Node partitions based on the file lines and number of mappers
    3. Master Node sends the workers the string to map over ZeroMQ, this number of mappers is specified in the configuration file
    4. Master waits for response from every mapper
    5. Mappers performs the mapping on the string sent, and then organize the keys based on the ASCII Value of the string modulus the number of reducers
    6. Mappers send the results to the corresponding to the groupby created in the last step
    7. Reducers receive the information

8. Reducers respond to the mappers saying they have received the information
9. Once Reducers have received results = # of Mappers, they perform the reducing
10. The reducer sends the completed results to the master node
11. The master gathers this data from all of the reducers and performs last minute specifier calculations

--------------Inverted Index----------------

1. Master Node accesses given directory
2. Master Node partitions based on the number of files and number of mappers
3. Master Node sends the workers a list of file paths to access
4. Master waits for response from every mapper
5. Mappers performs the mapping on every file in the list sent, and then organize the keys based on the ASCII Value of the string modulus the number of reducers
6. Mappers send the results to the corresponding to the groupby created in the last step
7. Reducers receive the information
8. Reducers respond to the mappers saying they have received the information
9. Once Reducers have received results = # of Mappers, they perform the reducing
10. The reducer sends the completed results to the master node
11. The master gathers this data from all of the reducers and performs last minute specifier calculations

12. The master node sends it's reply back to the API over ZeroMQ
13. The API formats the response in JSON structure and responds to the requestor

------Logging------

Each process creates and keeps a real time log of actions and events that occur on the system. All of these logs are stored
inside the "logging" folder. This folder contains logs for every worker, master node and webserver named accordingly.
Every log is time stamped.

------Limitations & Assumptions------

- Data is partioned based on size of data that is being passed through. Rather than using lines to determine what is sent to each mapper,
  the master distributes every mapper an equal amount of data, this data is determined by the size of the file.
- The master can create any number of workers, but for the workers to be created successfully,
  it needs access to a range of ports equal to the number of workers all of these ports must be specified in the configuration file
  These ports are not specified if they are a reducer port or a mapper port, that is decided by the master at runtime
  Because fault tolerance is optional for undergraduates, I am making the assumption that the port will always be accessed and reserved properly.

------How to Run------
1. Open the folder
2. Ensure zmq is installed, and the correct executable is set for pythonCommand in the configuration file
3. Modify the configuration file, ensuring ports are unassigned and num_workers = num_mappers + num_reducers
4. Run webserver.py
5. Send request to: "http://{ip_address}:{server port}/?method=...&..."
OR
Run start.bash/start.bat depending on linux or windows, ensure the correct python command is set there as well

-----How to test-----

Word Count:
 Add custom text file to /books directory, request that book from the webserver
Inverted Index:
 Add custom directory to the assignment 2 folder, add custom text files to directory, request that directory from the webserver

[x] Nodes are processes
[x] Must communicate over network - ZMQ
[x] Master node which coordinates all other processes
[x] Well defined API - HTTP Server using JSON
[x] Word Count
[x] Inverted Index
[x] Testing
[x] Scripts to run
[x] Report