1. **log4j2.xml**

Defines default configuration of log4j2 logger.

1. **MongoDBConfiguration.cfg**

One sample configuration is given below:

**hostIP = 192.168.1.253**

**hostPort = 27017**

**dataBaseName = newServerApplicationDB**

**userName = admin**

**passWord = admin123**

Now, hostIP and hostPort defined the IP address and Port of the remote server where mongodb is installed.

**databaseName** defines the databaseName which is used for project.

userName and passWord are user name and password for accessing the specified database.

Now, every variable in MongoDBConfiguration.cfg is reloadable.

1. **Classes:**

**i) PatternMatchingOptions.java**

This is a class which defines an interface

public interface PatternMatchingOptions

{

public static final char CASE\_INSENSITIVE='i';

public static final char INCLUDE\_ANCHOR='m';

public static final char ALLOW\_DOT\_CHARACTER\_TO\_MATCH\_ALL\_CHARACTER=

's';

}

In MongoDBHandler, there’s are functions for fetching, updating and deleting data based on sql like query type query. I.e. we search if a field’s value is matched with some predefined regular expression pattern or not. Now, for regex matching, these 3 options can be set.

**ii) DebugSignalHandler**

Now, since, there’s configuration file, we support reload configuration file option. Now, do perform reload Configuration dynamically, we provide a mechanism. Suppose, some user needs to reload configuration file, when the application is already running. He/She will first change the configuration file. After it, he/she has to check on the process id of the current process. This can be done using

Netstat -nlp|grep port\_number

Where port+number is the current port number in which the process is running

After that

Do, sudo kill -12 pid\_of\_process

Now, the signal will be received by the process. Now, DebugSignalHandler module will process the receiving signal and if the receiving signal is SIGUSR2 (12 represents SIGUSR2 signal) , it reloads the fetched data from the configuration file.

**iii) MongoDBTester**

It contains the queries for testing. This class is not important at all.

**Iv) MongoDBConfiguration**

This class fetches data from Configuration file.

The functions of the class are following:

1. **Private MongoDBConfiguration():** constructor of the class. Initialises the variables associated with the class.

Now, the constructor is private, since, we want to ensure that only one instance of this class is active all the time.

**b) public static MongoDBConfiguration getMongoDBConfigurationInstance()**

Returns the instance of the class. (Since, the class MongoDBConfiguration is a singleton class)

**c)private boolean loadConfig():** This class fetches data from the configuration file and initialises fetched data to class variables.

**d) private boolean setAndValidateConfig(String field, String value):**

Whenever, a configuration variable name and it’s corresponding value is fetched from the configuration file, this function is called to ensure their validity.

**e) private boolean validateIP(String ipAddress)**

This validates IP since, hostIP needs to be fetched from Configuration file

**f) public boolean reloadConfig()**

Now, reloadConfig function is actually for other class to ensure that configuration data is fetched. Since, loadConfig() function is private.

Now, there’s a lass access time variable in the class. If current access time - last access time > 10 seconds, it calls loadConfig function fetch the configuration data (once again). It is present so that, configuration file cannot be very frequently reloaded.

Now, the following functions are basically getter functions associated to the class variables:

**g) public String returnHostIP()**

Returns the hostIP

**h) public short returnHostPort()**

Returns the hostPort

Now, hostIP and hostPort are IP Address and port number of the remote server in which mongodb is installed and in which the current application’s database is residing.

**I) public String returnDataBaseName()**

**Returns the database Name.** (the application’s data base)

**J) public String returnUserName()**

Returns the user name of the database.

**K) public String returnPassword()**

Returns the password of the database.

**v) MongoDBHandler**

**1) private MongoDBHandler(String hostIP, short hostPort, String databaseName, String userName, String password)**

Private constructor of the mongoDBHandler. Constructor is private since we want to ensure that only one instance is active at a time.

**2) public static MongoDBHandler initMongoDBHandler()**

Function to get an instance of the class.

**3)public static boolean reinitMongoDBHandler()**

Suppose, configuration function is reloaded. Now, some variables of this class (as this class establishes connection to database, authenticates an user in database etc) must be reloaded too. For this purpose, mongoDBHandler class instance must be re instantiated.

**4) public boolean createATable(String tableName, Set<String> uniqueKeys)**

**5) public boolean createATable(String tableName)**

Now, both functions create a table. First function additionally accepts a set of uniqueKeys. uniqueKeys is the set of field names whose value is supposed to be unique. So, records containing duplicate values of these fields cannot be inserted.

**6) public synchronized void showExistingTable()**

This shows the list of existing tables in the application’s database.

**7) public boolean insertIntoTable(String tableName, BasicDBObject toInsertData)**

**8)** **public boolean insertIntoTable(String tableName, HashMap<String, Object> data)**

Inserts a record in JSON string format in the collection/table.

**9) public ArrayList<String> fetchAllRows(String tableName)**

This function helps to get all the documents inserted in a particular table

**10) public ArrayList<String> fetchSpecificRowsWhereBasicQuery(String tableName,HashMap<String, Object> whereQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age is=23

Then, you have to call fetchSpecificRowsWhereBasicQuery(“tableName”,

whereQueryDetails,null,null)

Suppose, you want to display first n number of records as the result of already mentioned query.

You have to specify limit=n

Suppose, you want to skip first n number of records as the result of already mentioned query.

You have to specify skipFirstFew=n

**11) public ArrayList<String> fetchSpecificRowsWhereNotEqualToQuery(String tableName,**

**HashMap<String, Object> whereNotEqualToQueryDetails, Integer limit,**

**Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age!=23.

For queries like these, we need this function.

We already mention the use of limit and skipFirstFew.

**12) public ArrayList<String> fetchSpecificRowsWhereInQuery(String tableName,**

**HashMap<String, List> whereInQueryDetails, Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age in (23,24,25). For queries like these, we need this function.

We already mention the use of limit and skipFirstFew.

**13) public ArrayList<String> fetchSpecificRowsWhereNotInQuery(String tableName,**

**HashMap<String, List> whereNotInQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age is not in (23,24,25). For queries like these, we need this function.

**14) public ArrayList<String> fetchSpecificRowsWhereGreaterThanQuery(String tableName,**

**HashMap<String, Object> whereGreaterThanQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age >23

. For queries like these, we need this function.

**15) public ArrayList<String> fetchSpecificRowsWhereGreaterThanOrEqualQuery(String tableName, HashMap<String, Object> whereGreaterThanOrEqualQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age >=23

. For queries like these, we need this function.

**16) public ArrayList<String> fetchSpecificRowsWhereLessThanQuery(String tableName,**

**HashMap<String, Object> whereLessThanQueryDetails, Integer limit,**

**Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age<29

. For queries like these, we need this function.

**17) public ArrayList<String> fetchSpecificRowsWhereLessThanOrEqualQuery(String tableName,**

**HashMap<String, Object> whereLessThanOrEqualQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age<=29

. For queries like these, we need this function.

**18) public ArrayList<String> fetchSpecificRowsInBetweenExclusiveQuery(String tableName,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age is >23 and <29

. For queries like these, we need this function.

**19) public ArrayList<String> fetchSpecificRowsInBetweenInclusiveQuery(String tableName,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose age is >=23 and <=29

. For queries like these, we need this function.

**20) public ArrayList<String> fetchSpecificRowsWhereLikeQueryCaseSensitive(String tableName,**

**HashMap<String, Object> whereLikeQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose name is Like “^S.\*”. I.e. Whose name starts with s. For queries like these, we need this function. Now, the regex matching will be done in case sensitive manner.

**21) public ArrayList<String> fetchSpecificRowsWhereLikeQueryCaseInSensitive(String tableName,**

**HashMap<String, Object> whereLikeQueryDetails,**

**Integer limit, Integer skipFirstFew)**

Suppose, you want to display the records of all persons whose name is Like “^S.\*”. I.e. Whose name starts with s. For queries like these, we need this function. Now, the regex matching will be done in case insensitive manner.

**22) public ArrayList<String> fetchSpecificRowsWhereLikeQuery(String tableName,**

**HashMap<String, HashMap<Object, Object>> whereLikeQueryDetails,**

**Integer limit, Integer skipFirstFew)**

You can additionally provide regex options for like query in this function. For more information, check TestingFinalAPI.docx

**23) public ArrayList<String> fetchSpecificRowsWhereQuery(String tableName,**

**HashMap<String, Object> whereBasicQueryDetails,**

**HashMap<String, Object> whereGreaterThanOrEqualQueryDetails,**

**HashMap<String, Object> whereGreaterThanQueryDetails,**

**HashMap<String, Object> whereLessThanQueryDetails,**

**HashMap<String, Object> whereLessThanOrEqualQueryDetails,**

**HashMap<String, List> whereInQueryDetails,**

**HashMap<String, List> whereNotInQueryDetails,**

**HashMap<String, Object> whereNotEqualToQueryDetails,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenExclusiveQueryDetails,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenInclusiveQueryDetails,**

**HashMap<String, HashMap<Object, Object>> whereLikeQueryDetails,**

**Integer limit,**

**Integer skipFirstFew**

**)**

Now, suppose you have multiple queries. Like, you want to fetch the records of all person whose Name starts with S and who are male. For queries like these, you need this function.

**24) public boolean deleteSpecificRowsWhereBasicQuery(String tableName,**

**HashMap<String, Object> whereQueryDetails)**

Suppose, you want to delete records of all persons whose age=23. For queries like these, you need this function.

**25) public boolean deleteSpecificRowsWhereNotEqualToQuery(String tableName, HashMap<String, Object> whereNotEqualToQueryDetails)**

Suppose, you want to delete records of all persons whose age!=23. For queries like these, you need this function.

**26) public boolean deleteSpecificRowsWhereInQuery(String tableName, HashMap<String, List> whereInQueryDetails)**

Suppose, you want to delete records of all persons whose age is In (23,24). For queries like these, you need this function.

**27) public boolean deleteSpecificRowsWhereNotInQuery(String tableName, HashMap<String, List> whereNotInQueryDetails)**

Suppose, you want to delete records of all persons whose age is not In (23,24). For queries like these, you need this function.

**28) public boolean deleteSpecificRowsWhereGreaterThanQuery(String tableName, HashMap<String, Object> whereGreaterThanQueryDetails)**

Suppose, you want to delete records of all persons whose age >23. For queries like these, you need this function.

**29) public boolean deleteSpecificRowsWhereGreaterThanOrEqualQuery(String tableName, HashMap<String, Object> whereGreaterThanOrEqualQueryDetails)**

Suppose, you want to delete records of all persons whose age >= 23. For queries like these, you need this function.

**30) public boolean deleteSpecificRowsWhereLessThanQuery(String tableName, HashMap<String, Object> whereLessThanQueryDetails)**

Suppose, you want to delete records of all persons whose age <29. For queries like these, you need this function.

**31) public boolean deleteSpecificRowsWhereLessThanOrEqualQuery(String tableName, HashMap<String, Object> whereLessThanOrEqualQueryDetails)**

Suppose, you want to delete records of all persons whose age <=29. For queries like these, you need this function.

**32) public boolean deleteSpecificRowsInBetweenExclusiveQuery(String tableName,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenQueryDetails)**

Suppose, you want to delete records of all persons whose age is >24 and <29. For queries like these, you need this function.

**33) public boolean deteteSpecificRowsInBetweenInclusiveQuery(String tableName,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenQueryDetails)**

Suppose, you want to delete records of all persons whose age is >=24 and <=29. For queries like these, you need this function.

**34) public boolean deleteSpecificRowsWhereLikeQueryCaseSensitive(String tableName,**

**HashMap<String, Object> whereLikeQueryDetails)**

Suppose, you want to delete records all all persons whose name is like

“^S.\*” or whose name starts with S. For queries like these, you need this function. The regex comparison is done in case sensitive manner.

**35) public boolean deleteSpecificRowsWhereLikeQueryCaseInSensitive(String tableName,**

**HashMap<String, Object> whereLikeQueryDetails)**

Suppose, you want to delete records all all persons whose name is like

“^S.\*” or whose name starts with S. For queries like these, you need this function. The regex comparison is done in case insensitive manner.

**36) public boolean deleteSpecificRowsWhereLikeQueryCaseInSensitive(String tableName,**

**HashMap<String, Object> whereLikeQueryDetails)**

This function provides additional options for regex comparison based deletion. For more info, check TestingFinalAPI.

**37) public boolean deleteSpecificRowsWhereQuery(String tableName,**

**HashMap<String, Object> whereBasicQueryDetails,**

**HashMap<String, Object> whereGreaterThanOrEqualQueryDetails,**

**HashMap<String, Object> whereGreaterThanQueryDetails,**

**HashMap<String, Object> whereLessThanQueryDetails,**

**HashMap<String, Object> whereLessThanOrEqualQueryDetails,**

**HashMap<String, List> whereInQueryDetails,**

**HashMap<String, List> whereNotInQueryDetails,**

**HashMap<String, Object> whereNotEqualToQueryDetails,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenExclusiveQueryDetails,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenInclusiveQueryDetails,**

**HashMap<String, HashMap<Object, Object>> whereLikeQueryDetails**

**)**

Suppose, you want to delete records of all persons whose name starts with S and age is >23 and <29. For queries like these, we need this function.

**38) public boolean drop\_a\_table(String tableName)**

This is to drop a table/collection.

**39) public boolean updateSpecificRowsWhereBasicQuery(String tableName,**

**HashMap<String, Object> whereQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 24 whose name is Sayak. For update queries like these, you need this function.

**40) public boolean updateSpecificRowsWhereNotEqualToQuery(String tableName,**

**HashMap<String, Object> whereNotEqualToQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 26 whose id!=2.

For update queries like these, you need this function.

**41) public boolean updateSpecificRowsWhereInQuery(String tableName, HashMap<String, List> whereInQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 24 whose id in(2,3). or whose id is either 2 or 3. For update queries like these, you need this function.

**42) public boolean updateSpecificRowsWhereNotInQuery(String tableName,**

**HashMap<String, List> whereNotInQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 24 whose id is not in(2,3). For update queries like these, you need this function.

**43) public boolean updateSpecificRowsWhereGreaterThanQuery(String tableName,**

**HashMap<String, Object> whereGreaterThanQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 25 whose id is>3.

For update queries like these, you need this function.

**44) public boolean updateSpecificRowsWhereGreaterThanOrEqualQuery(String tableName,**

**HashMap<String, Object> whereGreaterThanOrEqualQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 26 whose id is>=3.

For update queries like these, you need this function.

**45) public boolean updateSpecificRowsWhereLessThanQuery(String tableName,**

**HashMap<String, Object> whereLessThanQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 20 whose id is<4.

For update queries like these, you need this function.

**46) public boolean updateSpecificRowsWhereLessThanOrEqualQuery(String tableName,**

**HashMap<String, Object> whereLessThanOrEqualQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 21 whose id<=3.

For update queries like these, you need this function.

**47) public boolean updateSpecificRowsInBetweenExclusiveQuery(String tableName,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 21 whose id>3 and id<7.

For update queries like these, you need this function.

**48)** **public boolean updateSpecificRowsInBetweenInclusiveQuery(String tableName,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 21 whose id=>3 and id<=7.

For update queries like these, you need this function.

**49) public boolean updateSpecificRowsWhereLikeQueryCaseSensitive(String tableName,**

**HashMap<String, Object> whereLikeQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 29 whose name starts with S (in a case sensitive manner). For update queries like these, you need this function.

**50)** **public boolean updateSpecificRowsWhereLikeQueryCaseInSensitive(String tableName,**

**HashMap<String, Object> whereLikeQueryDetails,**

**HashMap<String, Object> updateDetails)**

Suppose, you want to update all person’s age to 29 whose name starts with s (in a case insensitive manner). For update queries like these, you need this function.

**51) public boolean updateSpecificRowsWhereLikeQuery(String tableName,**

**HashMap<String, HashMap<Object, Object>> whereLikeQueryDetails,**

**HashMap<String, Object> updateDetails)**

This provides additional regex options to do regex based updates.

**52) public boolean updateSpecificRowsWhereQuery(String tableName,**

**HashMap<String, Object> whereBasicQueryDetails,**

**HashMap<String, Object> whereGreaterThanOrEqualQueryDetails,**

**HashMap<String, Object> whereGreaterThanQueryDetails,**

**HashMap<String, Object> whereLessThanQueryDetails,**

**HashMap<String, Object> whereLessThanOrEqualQueryDetails,**

**HashMap<String, List> whereInQueryDetails,**

**HashMap<String, List> whereNotInQueryDetails,**

**HashMap<String, Object> whereNotEqualToQueryDetails,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenExclusiveQueryDetails,**

**HashMap<String, HashMap<Object, Object>> whereInBetweenInclusiveQueryDetails,**

**HashMap<String, HashMap<Object,Object>> whereLikeQueryDetails,**

**HashMap<String, Object> updateDetails**

**)**

Suppose, you want to update all persons’ age whose name starts with S and whose id >2 and id<7. For complex update queries like these, you need this function.

**53) public void destroyMongoDBHandler()**

This closes the connection to the database.