```
1  package ar.fiuba.taller.storage;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.File;
6  import java.io.FileNotFoundException;
7  import java.io.FileOutputStream;
8  import java.io.FileReader;
9  import java.io.FileWriter;
10 import java.io.IOException;
11 import java.io.PrintWriter;
12 import java.io.RandomAccessFile;
13 import java.io.StringReader;
14 import java.nio.ByteBuffer;
15 import java.nio.channels.FileChannel;
16 import java.nio.channels.FileLock;
17 import java.nio.file.Path;
18 import java.nio.file.Paths;
19 import java.nio.file.StandardOpenOption;
20 import java.util.ArrayList;
21 import java.util.Collections;
22 import java.util.HashMap;
23 import java.util.Iterator;
24 import java.util.LinkedHashMap;
25 import java.util.List;
26 import java.util.ListIterator;
27 import java.util.Map;
28 import java.util.regex.Matcher;
29 import java.util.regex.Pattern;
30
31 import org.apache.log4j.Logger;
32 import org.apache.log4j.MDC;
33 import org.json.simple.JSONArray;
34 import org.json.simple.JSONObject;
35 import org.json.simple.parser.JSONParser;
36 import org.json.simple.parser.ParseException;
37
38 import ar.fiuba.taller.common.Command;
39 import ar.fiuba.taller.common.Constants;
40
41 public class Storage {
42
43   private int shardingFactor;
44   private int queryCountShowPosts;
45   private int ttCountShowPosts;
46   final static Logger logger = Logger.getLogger(Storage.class);
47
48   public Storage(int shardingFactor, int queryCountShowPosts,
49       int ttCountShowPosts) {
50     this.shardingFactor = shardingFactor;
51     this.queryCountShowPosts = queryCountShowPosts;
52     this.ttCountShowPosts = ttCountShowPosts;
53     MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
54   }
55
56   private void updateTT(Command command) throws IOException, ParseException {
57     String fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_TT;
58     JSONParser parser = new JSONParser();
59
60     logger.info("Actualizando los TT");
61     RandomAccessFile aFile = new RandomAccessFile(fileName, "rw");
62     try (FileChannel fileChannel = aFile.getChannel()) {
63       FileLock lock = fileChannel.lock();
64       ByteBuffer buffer = null;
65       String tmp = loadFile(fileChannel, buffer);
66       Object obj = parser.parse(new StringReader(tmp));
```



```
67       JSONObject jsonObject = (JSONObject) obj;
68       int count = 0;
69       String regexPattern = "(#\\w+)";
70       Pattern p = Pattern.compile(regexPattern);
71       Matcher m = p.matcher(command.getMessage());
72       String hashtag;
73       while (m.find()) {
74         hashtag = m.group(1);
75         hashtag = hashtag.substring(1, hashtag.length());
76         Long obj2 = (Long) jsonObject.get(hashtag);
77         if (obj2 == null) {
78           // La entrada no existe y hay que crearla
79           jsonObject.put(hashtag, 1);
80         } else {
81           obj2++;
82           jsonObject.put(hashtag, obj2);
83         }
84       }
85       fileChannel.truncate(0);
86       buffer = ByteBuffer.allocate((int) jsonObject.toJSONString().length()));
87       buffer.put(jsonObject.toJSONString().getBytes());
88       buffer.flip();
89           while(buffer.hasRemaining()) {
90             fileChannel.write(buffer);
91           }
92     } catch (Exception e) {
93       logger.error("Error guardar el indice de TT: " + e);
94     }
95   }
96
97   public void saveMessage(Command command)
98       throws IOException, ParseException {
99     String fileName = Constants.DB_DIR + "/"
100        + command.getUuid().toString().substring(0, shardingFactor)
101        + Constants.COMMAND_SCRIPT_EXTENSION;
102    JSONParser parser = new JSONParser();
103    Object obj;
104
105    logger.info("Guardando el comando en la base de datos: " + fileName);
106    logger.info("Contenido del registro: " + command.toJson());
107    File tmpFile = new File(fileName);
108    if (tmpFile.createNewFile()) {
109      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
110    }
111    JSONObject obj2 = new JSONObject();
112    obj2.put("command", command.getCommand().toString());
113    obj2.put("user", command.getUser());
114    obj2.put("message", command.getMessage());
115    obj2.put("timestamp", command.getTimestamp());
116    JSONObject jsonObject = new JSONObject();
117    jsonObject.put(command.getUuid().toString(), obj2);
118
119    RandomAccessFile aFile     = new RandomAccessFile(fileName, "rw");
120    FileChannel      fileChannel = aFile.getChannel();
121    FileLock lock = fileChannel.lock();
122
123    try {
124      ByteBuffer buffer = ByteBuffer.wrap((jsonObject.toJSONString() + String.fo
     rmat("%n")).getBytes());
125      fileChannel.write(buffer);
126    } catch (Exception e) {
127      logger.error("Error guardar la base de datos: " + e);
128    } finally {
129      // Una vez que persisto el mensaje, actualizo los indices y el TT
130      updateUserIndex(command);
131      updateHashTagIndex(command);
```

```
132         updateTT(command);
133         lock.release();
134         fileChannel.close();
135       }
136     }
137
138     private void updateUserIndex(Command command)
139         throws IOException, ParseException {
140       String fileName = Constants.DB_INDEX_DIR + "/"
141           + Constants.DB_USER_INDEX;
142       JSONParser parser = new JSONParser();
143
144       logger.info("Actualizando el inice de usuarios");
145       RandomAccessFile aFile = new RandomAccessFile(fileName, "rw");
146           try (FileChannel fileChannel = aFile.getChannel()) {
147         FileLock lock = fileChannel.lock();
148         ByteBuffer buffer = null;
149         String tmp = loadFile(fileChannel, buffer);
150         Object obj = parser.parse(new StringReader(tmp));
151         JSONObject jsonObject = (JSONObject) obj;
152         JSONArray array = (JSONArray) jsonObject.get(command.getUser());
153         if (array == null) {
154           // Hay que crear la entrada en el indice
155           JSONArray ar2 = new JSONArray();
156           ar2.add(command.getUuid().toString());
157           jsonObject.put(command.getUser(), ar2);
158         } else {
159           array.add(command.getUuid().toString());
160           jsonObject.put(command.getUser(), array);
161         }
162         fileChannel.truncate(0);
163         buffer = ByteBuffer.allocate(((int) jsonObject.toJSONString().length()));
164         buffer.put(jsonObject.toJSONString().getBytes());
165         buffer.flip();
166             while(buffer.hasRemaining()) {
167               fileChannel.write(buffer);
168             }
169       } catch (Exception e) {
170         logger.error("Error guardar el indice de usuarios: " + e);
171       }
172     }
173
174     private void updateHashTagIndex(Command command)
175         throws IOException, ParseException {
176       String fileName = Constants.DB_INDEX_DIR + "/"
177           + Constants.DB_HASHTAG_INDEX;
178       JSONParser parser = new JSONParser();
179
180       logger.info("Actualizando el inice de hashtags");
181       RandomAccessFile aFile = new RandomAccessFile(fileName, "rw");
182           try (FileChannel fileChannel = aFile.getChannel()) {
183         FileLock lock = fileChannel.lock();
184         ByteBuffer buffer = null;
185         String tmp = loadFile(fileChannel, buffer);
186         Object obj = parser.parse(new StringReader(tmp));
187         JSONObject jsonObject = (JSONObject) obj;
188         JSONArray array;
189         String regexPattern = "(#\\w+)";
190         Pattern p = Pattern.compile(regexPattern);
191         Matcher m = p.matcher(command.getMessage());
192         String hashtag;
193         JSONArray ar2;
194         while (m.find()) {
195           hashtag = m.group(1);
196           hashtag = hashtag.substring(1, hashtag.length());
197           array = (JSONArray) jsonObject.get(hashtag);
```



```
198           if (array == null) {
199             // Hay que crear la entrada en el indice
200             ar2 = new JSONArray();
201             ar2.add(command.getUuid().toString());
202             jsonObject.put(hashtag, ar2);
203           } else {
204             array.add(command.getUuid().toString());
205             jsonObject.put(hashtag, array);
206           }
207         }
208         fileChannel.truncate(0);
209         buffer = ByteBuffer.allocate(((int) jsonObject.toJSONString().length()));
210         buffer.put(jsonObject.toJSONString().getBytes());
211         buffer.flip();
212             while(buffer.hasRemaining()) {
213               fileChannel.write(buffer);
214             }
215       } catch (Exception e) {
216         logger.error("Error guardar el indice de hashtags: " + e);
217       }
218     }
219
220     public String query(Command command) throws IOException, ParseException {
221       List<String> resultList = new ArrayList<String>();
222       String listString = "";
223       if (String.valueOf(command.getMessage().charAt(0)).equals("#")) { // #
224         resultList = queryBy(command.getMessage().substring(1,
225             command.getMessage().length()), "HASHTAG");
226       } else if (command.getMessage().equals("TT")) { // Es consulta por TT
227         resultList = queryTT(command.getMessage());
228       } else { // Es consulta por usuario
229         resultList = queryBy(command.getMessage(), "USER");
230       }
231       if(¬resultList.isEmpty()) {
232         for (String element : resultList) {
233           listString += element + "\n";
234         }
235       }
236       return listString;
237     }
238
239     private List<String> queryTT(String hashTag)
240         throws FileNotFoundException, IOException, ParseException {
241       Map<String, Long> map = new HashMap<String, Long>();
242       String fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_TT;
243       List<String> returnList = new ArrayList<String>();
244
245       // Levantar el json
246       JSONParser parser = new JSONParser();
247
248       RandomAccessFile aFile = new RandomAccessFile(fileName, "rw");
249           try (FileChannel fileChannel = aFile.getChannel()) {
250             FileLock lock = fileChannel.lock();
251             ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
252         fileChannel.read(buffer);
253         buffer.position(0);
254         StringBuilder sb = new StringBuilder();
255             while (buffer.hasRemaining()) {
256                 sb.append((char) buffer.get());
257             }
258
259         Object obj = parser.parse(new StringReader(sb.toString()));
260
261         JSONObject jsonObject = (JSONObject) obj;
262
263         // Crear un map
```

```
264         for (Iterator iterator = jsonObject.keySet().iterator(); iterator
265             .hasNext();) {
266           String key = (String) iterator.next();
267           map.put(key, (Long) jsonObject.get(key));
268         }
269
270         returnList = sortHashMapByValues(map);
271         returnList
272             .add("Total de topics: " + String.valueOf(map.keySet().size()));
273           } catch(Exception e) {
274           // Do nothing
275           }
276       return returnList;
277     }
278
279     private List<String> queryBy(String key, String type)
280         throws IOException, ParseException {
281       String fileName;
282       JSONParser parser = new JSONParser();
283       Object obj2;
284       List<String> messageList = new ArrayList<String>();
285       String file, id;
286
287       if (type.equals("USER")) {
288         logger.info("Consultando por user");
289         fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_USER_INDEX;
290       } else if (type.equals("HASHTAG")) {
291         logger.info("Consultando por hashtag");
292         fileName = Constants.DB_INDEX_DIR + "/"
293             + Constants.DB_HASHTAG_INDEX;
294       } else {
295         return messageList;
296       }
297
298       // Obtengo la lista de archivos que contienen el user
299
300       RandomAccessFile aFile = new RandomAccessFile(fileName, "rw");
301           try (FileChannel fileChannel = aFile.getChannel()) {
302         FileLock lock = fileChannel.lock();
303         ByteBuffer buffer = null;
304         String tmp = loadFile(fileChannel, buffer);
305         Object obj = parser.parse(new StringReader(tmp));
306
307         JSONObject jsonObject = (JSONObject) obj;
308         JSONArray array = (JSONArray) jsonObject.get(key);
309
310         String line, reg;
311         JSONObject jsonObject2;
312         int remainingPost = queryCountShowPosts;
313         // Abro archivo por archivo y recupero los mensajes
314         if (array ≠ null) {
315           ListIterator<String> iterator = array.listIterator(array.size());
316           while (iterator.hasPrevious() ∧ remainingPost > 0) {
317             id = iterator.previous();
318             file = Constants.DB_DIR + "/" + id.substring(0, shardingFactor)
319                 + Constants.COMMAND_SCRIPT_EXTENSION;
320             Path path2 = Paths.get(file);
321             try(FileChannel fileChannel2 = FileChannel.open(path2, StandardOpenOpt
    ion.READ)) {
322               FileLock lock2 = fileChannel2.lock(0, Long.MAX_VALUE, true);
323               ByteBuffer buffer2 = ByteBuffer.allocate(((int) fileChannel2.size())
    );
324               fileChannel2.read(buffer2);
325               buffer2.position(0);
326               StringBuilder sb2 = new StringBuilder();
327               while (buffer2.hasRemaining()) {
```



```
328                 sb2.append((char) buffer2.get());
329               }
330               try (
331                 BufferedReader br = new BufferedReader(
332                 new StringReader(sb2.toString()))
333               ) {
334                 while ((line = br.readLine()) ≠ null ∧ remainingPost > 0
335                     ∧ ¬("").equals(line.trim())) {
336                   System.out.println("line: " + line);
337                   obj2 = parser.parse(line);
338                   jsonObject2 = (JSONObject) obj2;
339                   if (jsonObject2.get(id) ≠ null) {
340                     messageList.add(jsonObject2.get(id).toString());
341                   }
342                   remainingPost--;
343                 }
344               }
345             }
346           }
347         }
348           }catch(Exception e) {
349             // Do nothing
350           }
351       // Retorno la lista con los mensajes encontrados
352       return messageList;
353     }
354
355     private List<String> sortHashMapByValues(Map<String, Long> map) {
356       List<String> mapKeys = new ArrayList<String>(map.keySet());
357       List<Long> mapValues = new ArrayList<Long>(map.values());
358       Collections.sort(mapValues);
359       Collections.sort(mapKeys);
360
361       LinkedHashMap<String, Long> sortedMap = new LinkedHashMap<String, Long>();
362
363       java.util.Iterator<Long> valueIt = mapValues.iterator();
364       while (valueIt.hasNext()) {
365         Long val = valueIt.next();
366         java.util.Iterator<String> keyIt = mapKeys.iterator();
367
368         while (keyIt.hasNext()) {
369           String key = keyIt.next();
370           Long comp1 = map.get(key);
371           Long comp2 = val;
372
373           if (comp1.equals(comp2)) {
374             keyIt.remove();
375             sortedMap.put(key, val);
376             break;
377           }
378         }
379       }
380       List<String> tt = new ArrayList<String>();
381       ArrayList<String> keys = new ArrayList<String>(sortedMap.keySet());
382       int i = keys.size() - 1;
383       int j = ttCountShowPosts;
384       while (i ≥ 0 ∧ j > 0) {
385         tt.add(keys.get(i));
386         j--;
387         i--;
388       }
389       return tt;
390     }
391
392     private String loadFile(FileChannel fileChannel, ByteBuffer buffer) throws IOE
    xception {
```

Oct 14, 17 18:19 **Storage.java** Page 7/7

```
393       buffer = ByteBuffer.allocate(((int) fileChannel.size()));
394       fileChannel.read(buffer);
395       buffer.position(0);
396       StringBuilder sb = new StringBuilder();
397           while (buffer.hasRemaining()) {
398               sb.append((char) buffer.get());
399           }
400           String tmp = sb.toString();
401           if((tmp.split("}", -1).length - 1) > 1) {
402             tmp = tmp.substring(0, tmp.indexOf("}")+1);
403           }
404           return tmp;
405     }
406   }
```

Oct 14, 17 9:45 **StorageController.java** Page 1/3

```
1   package ar.fiuba.taller.storage;
2
3   import java.io.IOException;
4   import java.util.HashMap;
5   import java.util.Iterator;
6   import java.util.List;
7   import java.util.Map;
8   import java.util.UUID;
9   import java.util.concurrent.TimeoutException;
10
11  import org.apache.log4j.Logger;
12  import org.apache.log4j.MDC;
13  import org.json.simple.parser.ParseException;
14
15  import ar.fiuba.taller.common.Command;
16  import ar.fiuba.taller.common.Constants;
17  import ar.fiuba.taller.common.ReadingRemoteQueue;
18  import ar.fiuba.taller.common.Response;
19  import ar.fiuba.taller.common.WritingRemoteQueue;
20  import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
21
22  public class StorageController {
23    private Map<String, String> config;
24    private Storage storage;
25    private ReadingRemoteQueue storageQueue;
26    private Map<String, WritingRemoteQueue> usersMap;
27    final static Logger logger = Logger.getLogger(StorageController.class);
28
29    public StorageController(Map<String, String> config,
30        ReadingRemoteQueue storageQueue) {
31      MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
32      this.config = config;
33      storage = new Storage(
34          Integer.parseInt(config.get(Constants.SHARDING_FACTOR)),
35          Integer.parseInt(config.get(Constants.QUERY_COUNT_SHOW_POSTS)),
36          Integer.parseInt(config.get(Constants.TT_COUNT_SHOW)));
37      this.storageQueue = storageQueue;
38      usersMap = new HashMap<String, WritingRemoteQueue>();
39    }
40
41    public void run() {
42      Command command;
43      List<byte[]> messageList = null;
44
45      logger.info("Consumiendo de la storageQueue");
46      try {
47        while (¬Thread.interrupted()) {
48          messageList = storageQueue.pop();
49          for (byte[] message : messageList) {
50            try {
51              command = new Command();
52              command.deserialize(message);
53              analyzeCommand(command);
54
55            } catch (ClassNotFoundException | IOException e) {
56              logger.error("No se ha podido deserializar el mensaje");
57            }
58          }
59        }
60      } catch (InterruptedException e) {
61        // Do nothing
62        logger.error("Error al analizar comando: " + e);
63      } finally {
64          Iterator it = usersMap.entrySet().iterator();
65          while (it.hasNext()) {
66              Map.Entry pair = (Map.Entry)it.next();
```

Oct 14, 17 9:45 **StorageController.java** Page 2/3

```
67              WritingRemoteQueue userQueue = (WritingRemoteQueue) pair.getValue();
68              try {
69                userQueue.close();
70              } catch (IOException | TimeoutException e) {
71                // Do nothing
72                logger.error("Error al cerrar una response user queue: " + e);
73              }
74                it.remove(); // avoids a ConcurrentModificationException
75              }
76        }
77        logger.info("Storgae Controller terminado");
78      }
79
80      private void analyzeCommand(Command command) throws InterruptedException {
81        String error_message = "Error al crear el mensaje";
82        Response response = new Response();
83
84        logger.info("Comando recibido con los siguientes parametros: "
85            + "\nUUID: " + command.getUuid() + "\nUsuario: "
86            + command.getUser() + "\nComando: " + command.getCommand()
87            + "\nMensaje: " + command.getMessage());
88
89        response.setUuid(UUID.randomUUID());
90        response.setUser(command.getUser());
91        try {
92          switch (command.getCommand()) {
93          case PUBLISH:
94            logger.info(
95                "Comando recibido: PUBLISH. Insertando en la cola de creacion.");
96            storage.saveMessage(command);
97            response.setMessage("Creacion exitosa");
98            response.setResponse_status(RESPONSE_STATUS.OK);
99            break;
100         case QUERY:
101           logger.info(
102               "Comando recibido: QUERY. Insertando en la cola de consultas.");
103           response.setMessage(storage.query(command));
104           logger.debug(response.getMessage());
105           response.setResponse_status(RESPONSE_STATUS.OK);
106           break;
107         default:
108           logger.info("Comando recibido invalido. Comando descartado.");
109         }
110       } catch (IOException e) {
111         response.setResponse_status(RESPONSE_STATUS.ERROR);
112         response.setMessage(error_message);
113         logger.error(e);
114       } catch (ParseException e) {
115         response.setResponse_status(RESPONSE_STATUS.ERROR);
116         response.setMessage(error_message);
117         e.printStackTrace();
118         logger.error(e);
119       } finally {
120         if (response ≠ null) {
121           sendResponse(response);
122           response = null;
123         }
124       }
125     }
126
127     private void sendResponse(Response response) {
128       logger.info("Siguiente respuesta");
129       WritingRemoteQueue currentUserRemoteQueue;
130       currentUserRemoteQueue = usersMap.get(response.getUser());
131       if (currentUserRemoteQueue ≡ null) {
132         // Creo la cola
```

Oct 14, 17 9:45 **StorageController.java** Page 3/3

```
133         try {
134           currentUserRemoteQueue = new WritingRemoteQueue(
135               response.getUser(), config.get(Constants.KAFKA_WRITE_PROPERTIES));
136         } catch (IOException e) {
137           logger.error("No se han podido crear las colas de kafka: " + e);
138           System.exit(1);
139         }
140         usersMap.put(response.getUser(), currentUserRemoteQueue);
141       }
142       logger.info(
143           "Enviando respuesta al usuario: " + response.getUser());
144       logger.info("UUID: " + response.getUuid());
145       logger.info("Status de la respuesta: "
146           + response.getResponse_status());
147       logger.info(
148           "Contenido de la respuesta: " + response.getMessage());
149       logger.info("Esperando siguiente respuesta");
150       try {
151         usersMap.get(response.getUser()).push(response);
152         logger.info("Respuesta enviada: " + response.getUser() + ":" + response.getMess
    age()
153           + ":" + response.getResponse_status() + ":" + response.getUuid());
154       } catch (IOException e) {
155         logger.error(
156             "No se ha podido enviar la respuesta al usuario "
157                 + response.getUser());
158       }
159     }
160   }
```

Oct 01, 17 11:55 **MainStorage.java** Page 1/1

```java
package ar.fiuba.taller.storage;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

import org.apache.log4j.Logger;
import org.apache.log4j.MDC;
import org.apache.log4j.PropertyConfigurator;

import ar.fiuba.taller.common.ConfigLoader;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.ReadingRemoteQueue;

public class MainStorage {
  final static Logger logger = Logger.getLogger(MainStorage.class);

  public static void main(String[] args) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    PropertyConfigurator.configure(Constants.LOGGER_CONF);
    ConfigLoader configLoader = null;

    try {
      configLoader = new ConfigLoader(Constants.CONF_FILE);
    } catch (IOException e) {
      logger.error("Error al cargar la configuracion");
      System.exit(Constants.EXIT_FAILURE);
    }
    ReadingRemoteQueue storageQueue = null;
    try {
      storageQueue = new ReadingRemoteQueue(
          configLoader.getProperties().get(Constants.STORAGE_QUEUE_NAME),
          configLoader.getProperties().get(Constants.KAFKA_READ_PROPERTIES));
    } catch (IOException e1) {
      logger.error("No se han podido inicializar las colas de kafka: " + e1);
      System.exit(1);
    }

    StorageController storageController = new StorageController(
        configLoader.getProperties(), storageQueue);

    storageController.run();
    storageQueue.shutDown();
    try {
      storageQueue.close();
    } catch (IOException | TimeoutException e) {
      // Do nothing
      logger.error("No se ha podido cerrar la cola de entrada al storage: " + e);
    }
  }
}
```

Oct 01, 17 11:52 **MainDispatcher.java** Page 1/1

```java
package ar.fiuba.taller.dispatcher;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

import org.apache.log4j.Logger;
import org.apache.log4j.MDC;
import org.apache.log4j.PropertyConfigurator;

import ar.fiuba.taller.common.ConfigLoader;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.ReadingRemoteQueue;

public class MainDispatcher {
  final static Logger logger = Logger.getLogger(MainDispatcher.class);

  public static void main(String[] args) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    PropertyConfigurator.configure(Constants.LOGGER_CONF);
    ConfigLoader configLoader = null;

    try {
      configLoader = new ConfigLoader(Constants.CONF_FILE);
    } catch (IOException e) {
      logger.error("Error al cargar la configuracion");
      System.exit(Constants.EXIT_FAILURE);
    }

    ReadingRemoteQueue dispatcherQueue = null;
    try {
      dispatcherQueue = new ReadingRemoteQueue(
          configLoader.getProperties()
              .get(Constants.DISPATCHER_QUEUE_NAME),
          configLoader.getProperties()
              .get(Constants.KAFKA_READ_PROPERTIES));
    } catch (IOException e1) {
      logger.error("No se han podido inicializar las colas de kafka: " + e1);
      System.exit(1);
    }

    DispatcherController dispatcherController = new DispatcherController(
        configLoader.getProperties(), dispatcherQueue);

    dispatcherController.run();
    dispatcherQueue.shutDown();
    try {
      dispatcherQueue.close();
    } catch (IOException | TimeoutException e) {
      // Do nothing
      logger.error("No se ha podido cerrar la cola del dispatcher");
      logger.debug(e);
    }
  }
}
```

Oct 01, 17 11:50 **DispatcherController.java** Page 1/2

```java
package ar.fiuba.taller.dispatcher;

import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.concurrent.TimeoutException;

import org.apache.log4j.Logger;
import org.apache.log4j.MDC;

import ar.fiuba.taller.common.Command;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.ReadingRemoteQueue;
import ar.fiuba.taller.common.WritingRemoteQueue;

public class DispatcherController {

  private ReadingRemoteQueue dispatcherQueue;
  private WritingRemoteQueue storageQueue;
  private WritingRemoteQueue analyzerQueue;
  private WritingRemoteQueue loggerQueue;

  final static Logger logger = Logger.getLogger(DispatcherController.class);

  public DispatcherController(Map<String, String> config,
      ReadingRemoteQueue dispatcherQueue) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    this.dispatcherQueue = dispatcherQueue;
    try {
      this.storageQueue = new WritingRemoteQueue(
          config.get(Constants.STORAGE_QUEUE_NAME),
          config.get(Constants.KAFKA_WRITE_PROPERTIES));
      this.loggerQueue = new WritingRemoteQueue(
          config.get(Constants.AUDIT_LOGGER_QUEUE_NAME),
          config.get(Constants.KAFKA_WRITE_PROPERTIES));
      this.analyzerQueue = new WritingRemoteQueue(
          config.get(Constants.ANALYZER_QUEUE_NAME),
          config.get(Constants.KAFKA_WRITE_PROPERTIES));
    } catch (IOException e) {
      logger.error("No se han podido inicializar las colas de kafka: " + e);
      System.exit(1);
    }
  }

  public void run() {
    Command command = new Command();
    List<byte[]> messageList = null;


    logger.info("Iniciando el dispatcher controller");
    try {
      while (¬Thread.interrupted()) {
        messageList = dispatcherQueue.pop();
        Iterator<byte[]> it = messageList.iterator();
        while (it.hasNext()) {
          try {
            command = new Command();
            command.deserialize(it.next());
            logger.info(
                "Comando recibido con los siguientes parametros: "
                    + "\nUsuario: " + command.getUser()
                    + "\nComando: " + command.getCommand()
                    + "\nMensaje: " + command.getMessage());
            switch (command.getCommand()) {
            case PUBLISH:
              storageQueue.push(command);
              analyzerQueue.push(command);
              loggerQueue.push(command);
              logger.info("Comando enviado al publish: "
                  + "\nUsuario: " + command.getUser()
                  + "\nComando: " + command.getCommand()
                  + "\nMensaje: " + command.getMessage());
              break;
            case QUERY:
              storageQueue.push(command);
              loggerQueue.push(command);
              logger.info("Comando enviado al query: "
                  + "\nUsuario: " + command.getUser()
                  + "\nComando: " + command.getCommand()
                  + "\nMensaje: " + command.getMessage());
              break;
            case DELETE:
              logger.info("Comando enviado al delete: "
                  + "\nUsuario: " + command.getUser()
                  + "\nComando: " + command.getCommand()
                  + "\nMensaje: " + command.getMessage());
              storageQueue.push(command);
              loggerQueue.push(command);
              break;
            case FOLLOW:
              logger.info("Comando enviado al follow: "
                  + "\nUsuario: " + command.getUser()
                  + "\nComando: " + command.getCommand()
                  + "\nMensaje: " + command.getMessage());
              analyzerQueue.push(command);
              loggerQueue.push(command);
              break;
            default:
              logger.error("Comando invalido");
              break;
            }
          } catch (ClassNotFoundException | IOException e) {
            logger.error("No se ha podido deserializar el mensaje: " + e);
          }
        }
      }
    } finally {
      try {
        storageQueue.close();
        dispatcherQueue.close();
        analyzerQueue.close();
      } catch (IOException | TimeoutException e) {
        // Do nothing
        logger.error("No se ha podido cerrar alguna de las colas");
        logger.debug(e);
      }
    }
    logger.info("Dispatcher controller terminado");
  }
}
```

Oct 01, 17 11:50 **DispatcherController.java** Page 2/2

Sep 16, 17 8:01 **App.java** Page 1/1

```java
package ar.fiuba.taller.crea_deploy;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

Oct 01, 17 11:39 **WritingRemoteQueue.java** Page 1/1

```java
package ar.fiuba.taller.common;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
import java.util.concurrent.TimeoutException;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class WritingRemoteQueue extends RemoteQueue {
  private Producer<byte[], byte[]> producer;
  private String queueName;

  public WritingRemoteQueue(String queueName,
      String propertiesFile) throws IOException {
    Properties props = new Properties();
    this.queueName = queueName;

    InputStream input = null;
    input = new FileInputStream(propertiesFile);
    props.load(input);
    producer = new KafkaProducer<byte[], byte[]>(props);
    input.close();
  }

  public void close() throws IOException, TimeoutException {
    producer.close();
  }

  public void push(ISerialize message) throws IOException {
    ProducerRecord<byte[], byte[]> data = new ProducerRecord<byte[], byte[]>(
        queueName, message.serialize());
    producer.send(data);
  }

}
```

Oct 01, 17 10:46 **Response.java** Page 1/2

```java
package ar.fiuba.taller.common;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.UUID;

import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;

public class Response implements Serializable, ISerialize {

  private UUID uuid;
  private String user;
  private RESPONSE_STATUS response_status;
  private String message;

  public Response(UUID uuid, RESPONSE_STATUS response_status,
      String message) {
    super();
    this.uuid = uuid;
    this.response_status = response_status;
    this.message = message;
  }

  public Response() {
    super();
    this.uuid = new UUID(0,0);
    this.response_status = RESPONSE_STATUS.EMPTY;
    this.message = "";
  }

  public byte[] serialize() throws IOException {
    ByteArrayOutputStream os = new ByteArrayOutputStream();
    ObjectOutput objOut = new ObjectOutputStream(os);

    objOut.writeObject(this);
    byte responseArray[] = os.toByteArray();
    objOut.close();
    os.close();
    return responseArray;
  }

  public void deserialize(byte[] responseArray)
      throws IOException, ClassNotFoundException {
    ByteArrayInputStream is = new ByteArrayInputStream(responseArray);
    ObjectInput objIn = new ObjectInputStream(is);
    Response tmp;
    tmp = (Response) objIn.readObject();
    objIn.close();
    is.close();
    uuid = tmp.getUuid();
    response_status = tmp.getResponse_status();
    message = tmp.getMessage();
  }

  public UUID getUuid() {
    return uuid;
  }

  public void setUuid(UUID uuid) {
    this.uuid = uuid;
```

Oct 01, 17 10:46 **Response.java** Page 2/2

```java
  }

  public RESPONSE_STATUS getResponse_status() {
    return response_status;
  }

  public void setResponse_status(RESPONSE_STATUS response_status) {
    this.response_status = response_status;
  }

  public String getMessage() {
    return message;
  }

  public void setMessage(String message) {
    this.message = message;
  }

  public String getUser() {
    return user;
  }

  public void setUser(String user) {
    this.user = user;
  }
}
```

Sep 16, 17 8:01 **RemoteQueue.java** Page 1/1

```java
package ar.fiuba.taller.common;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

public abstract class RemoteQueue {

  public abstract void close() throws IOException, TimeoutException;

}
```

Oct 01, 17 11:36 **ReadingRemoteQueue.java** Page 1/2

```java
package ar.fiuba.taller.common;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.concurrent.TimeoutException;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.errors.WakeupException;

public class ReadingRemoteQueue extends RemoteQueue {
  private KafkaConsumer<byte[], byte[]> consumer;

  public class ReadingRemoteQueueException extends WakeupException {
  }

  public ReadingRemoteQueue(String queueName,
      String propertiesFile) throws IOException {
    Properties consumerConfig = new Properties();
    InputStream input = null;
    input = new FileInputStream(propertiesFile);
    consumerConfig.load(input);
    consumer = new KafkaConsumer<byte[], byte[]>(consumerConfig);
    consumer.subscribe(Collections.singletonList(queueName));
    input.close();
  }

  @Override
  public void close() throws IOException, TimeoutException {
    consumer.close();
  }

  public void shutDown() {
    consumer.wakeup();
  }

  public List<byte[]> pop() throws ReadingRemoteQueueException {
    List<byte[]> msgList = null;

    try {
      while (msgList == null) {
        ConsumerRecords<byte[], byte[]> records = consumer
            .poll(Long.MAX_VALUE);
        if (¬records.isEmpty()) {
          msgList = new ArrayList<byte[]>();
          for (ConsumerRecord<byte[], byte[]> record : records) {
            msgList.add(record.value());
          }
          consumer.commitSync();
        }
      }
    } catch (WakeupException e) {
      throw new ReadingRemoteQueueException();
    }
    return msgList;
  }

```

Oct 01, 17 11:36 **ReadingRemoteQueue.java** Page 2/2

```
67  }
```

Sep 16, 17 8:01 **ISerialize.java** Page 1/1

```
1   package ar.fiuba.taller.common;
2
3   import java.io.IOException;
4
5   public interface ISerialize {
6
7     public byte[] serialize() throws IOException;
8
9     public void deserialize(byte[] byteForm)
10        throws IOException, ClassNotFoundException;
11
12  }
```

Oct 01, 17 11:31 **Constants.java** Page 1/2

```java
package ar.fiuba.taller.common;

import java.text.SimpleDateFormat;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class Constants {

  // Constantes globales
  public static final int COMMAND_QUEUE_SIZE = 1000;
  public static final int RESPONSE_QUEUE_SIZE = 1000;
  public static final String LOGGER_CONF = "conf/log4j.properties";

  public static final String COMMAND_SCRIPT = "scripts/script.json";
  public static final String COMMAND_ARRAY = "commands";
  public static final String COMMAND_KEY = "command";
  public static final String USER_KEY = "user";
  public static final String NAME_KEY = "name";
  public static final String USERS_KEY = "users";
  public static final String MESSAGE_KEY = "message";
  public static final String USERS_FILE = "conf/users.json";
  public static final String CONF_FILE = "configuration.properties";
  public static final String LOGS_DIR = "log";
  public static final String EVENT_VIEWER_FILE = "user_";
  public static final String EVENT_VIEWER_FILE_EXTENSION = ".events";
  public static final String COMMANDS_FILE_EXTENSION = ".commands";

  public static final String KAFKA_READ_PROPERTIES = "kafka.read.properties";
  public static final String KAFKA_WRITE_PROPERTIES = "kafka.write.properties";

  // Constantes para el usuario
  public static final String INTERACTIVE_MODE = "i";
  public static final String BATCH_MODE = "b";
  public static final String MAX_LENGTH_MSG = "max.length.msg";
  public static final String COMMAND_AMOUNT = "command.amount";
  public static final String BATCH_DELAY_TIME = "batch.delay.time";
  public static final long USER_THREAD_WAIT_TIME = 5000;

  // Constantes para el storage
  public static final String STORAGE_QUEUE_NAME = "storage.queue.name";
  public static final String STORAGE_QUERY_RESULT_QUEUE_NAME = "storage.query.result.queue.name";
  public static final String STORAGE_QUEUE_HOST = "storage.queue.host";
  public static final String STORAGE_QUERY_RESULT_QUEUE_HOST = "storage.query.result.queue.host";
  public static final String USERS_RESPONSE_HOST = "users.response.host";
  public static final long STORAGE_THREAD_WAIT_TIME = 5000;
  public static final String SHARDING_FACTOR = "sharding.factor";
  public static final String QUERY_COUNT_SHOW_POSTS = "query.count.show.posts";
  public static final String TT_COUNT_SHOW = "tt.count.show";
  public static final String COMMAND_SCRIPT_EXTENSION = ".json";

  // Constantes para el audit logger
  public static final String AUDIT_LOGGER_QUEUE_HOST = "audit.logger.queue.host";
  public static final String AUDIT_LOGGER_QUEUE_NAME = "audit.logger.queue.name";
  public static final long AUDIT_LOGGER_THREAD_WAIT_TIME = 5000;
  public static final String AUDIT_LOG_FILE = "audit.log.file";

  // Constantes para el dispatcher
  public static final String DISPATCHER_QUEUE_NAME = "dispatcher.queue.name";
  public static final String DISPATCHER_QUEUE_HOST = "dispatcher.queue.host";
  public static final long DISPATCHER_THREAD_WAIT_TIME = 5000;

  // Constantes para el analyzer
  public static final String ANALYZER_QUEUE_HOST = "analyzer.queue.host";
```

Oct 01, 17 11:31 **Constants.java** Page 2/2

```java
  public static final String ANALYZER_QUEUE_NAME = "analyzer.queue.name";
  public static final long ANALYZER_THREAD_WAIT_TIME = 5000;

  public static final String DB_DIR = "db";
  public static final String DB_INDEX_DIR = "idx";
  public static final String DB_USER_INDEX = "user.json";
  public static final String DB_HASHTAG_INDEX = "hashtag.json";
  public static final String DB_TT = "tt.json";
  public static final SimpleDateFormat SDF = new SimpleDateFormat(
      "yyyy-MM-dd HH:mm:ss");

  public static final String USER_READ_MODE = "r";
  public static final String USER_WRITE_MODE = "w";

  public static final String ACKS_CONFIG = "acks.config";
  public static final String RETRIES_CONFIG = "retries.config";
  public static final String KEY_SERIALIZER_CLASS_CONFIG = "key.serializer.class.config";
  public static final String VALUE_SERIALIZER_CLASS_CONFIG = "value.serializer.class.config";
  public static final String KEY_DESERIALIZER_CLASS_CONFIG = "key.deserializer.class.config";
  public static final String VALUE_DESERIALIZER_CLASS_CONFIG = "value.deserializer.class.config";
  public static final String GROUP_ID_CONFIG = "group.id.config";
  public static final String AUTO_OFFSET_RESET_CONFIG = "auto.offset.reset.config";

  public static enum COMMAND {
    PUBLISH, QUERY, DELETE, FOLLOW, EMPTY
  };

  public static Map<String, COMMAND> COMMAND_MAP;
  static {
    Map<String, COMMAND> tmpMap = new HashMap<String, Constants.COMMAND>();
    tmpMap.put("PUBLISH", COMMAND.PUBLISH);
    tmpMap.put("QUERY", COMMAND.QUERY);
    tmpMap.put("DELETE", COMMAND.DELETE);
    tmpMap.put("FOLLOW", COMMAND.FOLLOW);
    COMMAND_MAP = Collections.unmodifiableMap(tmpMap);
  }

  public static enum RESPONSE_STATUS {
    OK, ERROR, REGISTERED, EMPTY
  }

  public static Map<String, RESPONSE_STATUS> RESPONSE_STATUS_MAP;
  static {
    Map<String, RESPONSE_STATUS> tmpMap1 = new HashMap<String, RESPONSE_STATUS>();
    tmpMap1 = new HashMap<String, Constants.RESPONSE_STATUS>();
    tmpMap1.put("OK", RESPONSE_STATUS.OK);
    tmpMap1.put("ERROR", RESPONSE_STATUS.ERROR);
    tmpMap1.put("REGISTERED", RESPONSE_STATUS.REGISTERED);
    RESPONSE_STATUS_MAP = Collections.unmodifiableMap(tmpMap1);
  }

  public static final int EXIT_SUCCESS = 0;
  public static final int EXIT_FAILURE = 1;
}
```

Sep 16, 17 8:01 **ConfigLoader.java** Page 1/1

```java
package ar.fiuba.taller.common;

import java.io.IOException;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

public class ConfigLoader {

  private Map<String, String> propertiesMap;

  public ConfigLoader(String configFile) throws IOException {
    propertiesMap = new HashMap<String, String>();
    Properties properties = new Properties();
    try {
      properties.load(Thread.currentThread().getContextClassLoader()
          .getResourceAsStream(Constants.CONF_FILE));
    } catch (IOException e) {
      System.err.println(
          "No ha sido posible cargar el archivo de propiedades");
      throw new IOException();
    }
    for (String key : properties.stringPropertyNames()) {
      String value = properties.getProperty(key);
      propertiesMap.put(key, value);
    }

    propertiesMap = Collections.unmodifiableMap(propertiesMap);
  }

  public Map<String, String> getProperties() {
    return propertiesMap;
  }
}
```

Oct 01, 17 9:21 **Command.java** Page 1/2

```java
package ar.fiuba.taller.common;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.UUID;

import ar.fiuba.taller.common.Constants.COMMAND;

@SuppressWarnings("serial")
public class Command implements Serializable, ISerialize {

  private UUID uuid;
  private COMMAND command;
  private String user;
  private String message;
  private String timestamp;

  public Command() {
    this.command = COMMAND.EMPTY;
    this.user = "";
    this.message = "";
    this.uuid = new UUID(0,0);
    this.timestamp = "";
  }

  public Command(String command, String user, String message, UUID uuid,
      String timestamp) {
    this.command = Constants.COMMAND_MAP.get(command);
    this.user = user;
    this.message = message;
    this.uuid = uuid;
    this.timestamp = timestamp;
  }

  public byte[] serialize() throws IOException {
    ByteArrayOutputStream os = new ByteArrayOutputStream();
    ObjectOutput objOut = new ObjectOutputStream(os);

    objOut.writeObject(this);
    byte byteForm[] = os.toByteArray();
    objOut.close();
    os.close();
    return byteForm;
  }

  public void deserialize(byte[] byteForm)
      throws IOException, ClassNotFoundException {
    ByteArrayInputStream is = new ByteArrayInputStream(byteForm);
    ObjectInput objIn = new ObjectInputStream(is);
    Command tmp;
    tmp = (Command) objIn.readObject();
    objIn.close();
    is.close();
    uuid = tmp.getUuid();
    command = tmp.getCommand();
    user = tmp.getUser();
    message = tmp.getMessage();
    timestamp = tmp.getTimestamp();
  }

```

Oct 01, 17 9:21 **Command.java** Page 2/2

```
67    public COMMAND getCommand() {
68      return command;
69    }
70
71    public void setCommand(COMMAND command) {
72      this.command = command;
73    }
74
75    public String getUser() {
76      return user;
77    }
78
79    public void setUser(String user) {
80      this.user = user;
81    }
82
83    public String getMessage() {
84      return message;
85    }
86
87    public void setMessage(String message) {
88      this.message = message;
89    }
90
91    public UUID getUuid() {
92      return uuid;
93    }
94
95    public void setUuid(UUID uuid) {
96      this.uuid = uuid;
97    }
98
99    public String getTimestamp() {
100     return timestamp;
101   }
102
103   public void setTimestamp(String timestamp) {
104     this.timestamp = timestamp;
105   }
106
107   public String toJson() {
108     String tmp;
109
110     tmp = "{command:" + command.toString() + ",user:" + user + ",message:"
111         + message + ",timestamp:" + timestamp + "}";
112     return tmp;
113   }
114
115   public void fromJson(String jsonString) {
116
117   }
118 }
```

Oct 01, 17 9:38 **MainClientConsole.java** Page 1/2

```
1  package ar.fiuba.taller.ClientConsole;
2
3  import java.io.IOException;
4  import java.util.HashSet;
5  import java.util.Set;
6  import java.util.concurrent.Callable;
7  import java.util.concurrent.ExecutorService;
8  import java.util.concurrent.Executors;
9  import java.util.concurrent.TimeUnit;
10
11 import org.apache.log4j.Logger;
12 import org.apache.log4j.MDC;
13 import org.apache.log4j.PropertyConfigurator;
14 import ar.fiuba.taller.common.ConfigLoader;
15 import ar.fiuba.taller.common.Constants;
16
17 public class MainClientConsole {
18   final static Logger logger = Logger.getLogger(MainClientConsole.class);
19
20   public static void main(String[] args) {
21     PropertyConfigurator.configure(Constants.LOGGER_CONF);
22     MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
23     Set<Callable<String>> usersSet = new HashSet<Callable<String>>();
24     int usersAmount = 0;
25     ConfigLoader configLoader = null;
26
27     if (args.length ≡ 0) {
28       displayHelp();
29     }
30
31     String mode = args[0];
32
33     try {
34       configLoader = new ConfigLoader(Constants.CONF_FILE);
35     } catch (IOException e) {
36       logger.error("Error al cargar la configuracion");
37       System.exit(Constants.EXIT_FAILURE);
38     }
39
40     if (mode.equals(Constants.INTERACTIVE_MODE)) {
41       if ((args[1] ≡ null ∨ ("").equals(args[1]))
42           ∧ (args[2] ≡ null ∨ ("").equals(args[2]))) {
43         displayHelp();
44       }
45       System.out.printf(
46           "Iniciando el Client console en modo interactivo para el usuario %s",
47           args[1]);
48       InteractiveUser interactiveUser = new InteractiveUser(configLoader.getProp
   erties(), args[1], args[2]);
49       interactiveUser.run();
50     } else if (mode.equals(Constants.BATCH_MODE)) {
51       try {
52         usersAmount = Integer.parseInt(args[1]);
53       } catch (NumberFormatException e) {
54         System.out.printf("Argumento invalido");
55         System.exit(1);
56       }
57       ExecutorService executor = Executors.newFixedThreadPool(usersAmount);
58       System.out.printf("Iniciando el Client console en modo batch");
59       for (int i = 0; i < Integer.parseInt(args[1]); i++) {
60         usersSet.add(new BatchUser(configLoader.getProperties(),
61             "user" + i, configLoader.getProperties().get(Constants.USERS_RESPONSE
   _HOST)));
62       }
63       try {
64         executor.invokeAll(usersSet);
```

Oct 01, 17 9:38 **MainClientConsole.java** Page 2/2

```
65          }catch (Exception e) {
66            logger.error("Error al invocar a los usuarios: " + e);
67          } finally {
68            executor.shutdownNow();
69            try {
70              executor.awaitTermination(
71                  Constants.USER_THREAD_WAIT_TIME,
72                  TimeUnit.MILLISECONDS);
73            } catch (InterruptedException e) {
74              // Do nothing
75            }
76          }
77        }
78      }
79
80      private static void displayHelp() {
81        System.out.printf(
82            "Client console%n**************%nSintaxis:%n./ClientConsole <params>%nParametros:%ni [usernam
    e] [host]: Inicia el cliente en modo interactivo%nusername: Nombre del usuario%nhost: Nombre y puerto del servidor a
     conectar (ej. localhost:9092)%n%nb [usersamount] [host]: Inicia el cliente en modo batch%nusersamount: Cantidad de
     usuarios a simular%nhost: Nombre y puerto del servidor a conectar (ej. localhost:9092)%n%n");
83        System.exit(Constants.EXIT_FAILURE);
84      }
85
86  }
```

Oct 01, 17 11:48 **InteractiveUser.java** Page 1/2

```
1   package ar.fiuba.taller.ClientConsole;
2
3   import java.io.BufferedReader;
4   import java.io.IOException;
5   import java.io.InputStreamReader;
6   import java.util.Map;
7   import java.util.concurrent.TimeoutException;
8   import ar.fiuba.taller.common.Command;
9   import ar.fiuba.taller.common.Constants;
10  import ar.fiuba.taller.common.ReadingRemoteQueue;
11  import ar.fiuba.taller.common.WritingRemoteQueue;
12
13  public class InteractiveUser {
14    String userName;
15    private CommandController commandController;
16    private Thread eventViewerThread;
17    private ReadingRemoteQueue remoteUserResponseQueue;
18    private WritingRemoteQueue dispatcherQueue;
19
20    public InteractiveUser(Map<String, String> config, String userName,
21        String userHost) {
22      this.userName = userName;
23      try {
24        dispatcherQueue = new WritingRemoteQueue(
25            config.get(Constants.DISPATCHER_QUEUE_NAME),
26            config.get(Constants.KAFKA_WRITE_PROPERTIES));
27        remoteUserResponseQueue = new ReadingRemoteQueue(userName, config.get(Cons
    tants.KAFKA_READ_PROPERTIES));
28      } catch (IOException e) {
29        System.out.printf("No se han podido inicializar las colas de kafka: %s", e);
30        System.exit(1);
31      }
32      commandController =
33          new CommandController(dispatcherQueue,
34              Integer.parseInt(config.get(Constants.MAX_LENGTH_MSG)),
35              Constants.LOGS_DIR + "/" + userName
36                  + Constants.COMMANDS_FILE_EXTENSION);
37      eventViewerThread = new Thread(new EventWriter(
38          Constants.LOGS_DIR + "/" + userName
39              + Constants.EVENT_VIEWER_FILE_EXTENSION,
40              remoteUserResponseQueue));
41    }
42
43    public void run() {
44      BufferedReader br = null;
45      String[] msgParts;
46
47      eventViewerThread.start();
48      br = new BufferedReader(new InputStreamReader(System.in));
49      while (¬Thread.interrupted()) {
50        try {
51          System.out.print("Enter command: ");
52          String input = br.readLine();
53          msgParts = input.split(":");
54          commandController.sendMessage(new Command(msgParts[0], userName,
55              msgParts[1], null, null));
56        } catch (IOException e) {
57          System.out.println(
58              "Error: No se ha podido procesar el comando");
59        }
60      }
61
62      remoteUserResponseQueue.shutDown();
63      try {
64        remoteUserResponseQueue.close();
65      } catch (IOException | TimeoutException e) {
```

Oct 01, 17 11:48 **InteractiveUser.java** Page 2/2

```
66          // Do nothing
67        }
68        eventViewerThread.interrupt();
69        try {
70          eventViewerThread.join(Constants.USER_THREAD_WAIT_TIME);
71        } catch (InterruptedException e1) {
72          // Do nothing
73        }
74      }
75    }
```

Oct 14, 17 17:34 **EventWriter.java** Page 1/1

```
1   package ar.fiuba.taller.ClientConsole;
2
3   import java.io.BufferedWriter;
4   import java.io.FileWriter;
5   import java.io.IOException;
6   import java.io.PrintWriter;
7   import java.util.List;
8
9   import org.apache.log4j.Logger;
10  import org.apache.log4j.MDC;
11
12  import ar.fiuba.taller.common.ReadingRemoteQueue;
13  import ar.fiuba.taller.common.Response;
14
15  public class EventWriter implements Runnable {
16    private ReadingRemoteQueue remoteResponseQueue;
17    private String eventFile;
18    final static Logger logger = Logger.getLogger(EventWriter.class);
19
20    public EventWriter(
21        String eventFile, ReadingRemoteQueue remoteResponseQueue) {
22      MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
23      this.remoteResponseQueue = remoteResponseQueue;
24      this.eventFile = eventFile;
25    }
26
27    @SuppressWarnings("null")
28    public void run() {
29      Response response = new Response();
30      List<byte[]> messageList = null;
31
32      logger.debug("Iniciando el event viewer");
33      while (¬Thread.interrupted()) {
34        messageList = remoteResponseQueue.pop();
35        try (PrintWriter pw = new PrintWriter(new BufferedWriter(
36            new FileWriter(eventFile, true)))) {
37          for (byte[] message : messageList) {
38            response.deserialize(message);
39            pw.printf(
40                "Evento recibido − UUID: {%s} − Status: {%s} − Mensaje: {%s}%n−−−−−−−−−−−−−−−−−−−−
    −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−%n",
41                response.getUuid(), response.getResponse_status(),
42                response.getMessage());
43          }
44        } catch (IOException | ClassNotFoundException e) {
45          logger.error("No se ha podido escribir la respuesta: " + e);
46        }
47      }
48    }
49
50  }
```

Oct 14, 17 17:35 **CommandController.java** Page 1/1

```java
package ar.fiuba.taller.ClientConsole;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Timestamp;
import java.util.UUID;
import ar.fiuba.taller.common.Command;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.WritingRemoteQueue;

public class CommandController {
  private WritingRemoteQueue dispatcherQueue;
  private int maxlengthMsg;
  private Timestamp timestamp;
  private String commandFile;

  public CommandController(
      WritingRemoteQueue dispatcherQueue, int maxlengthMsg,
      String commandFile) {
    this.dispatcherQueue = dispatcherQueue;
    this.maxlengthMsg = maxlengthMsg;
    this.commandFile = commandFile;
  }

  public void sendMessage(Command command) {
    try {
      if (command.getMessage().length() ≤ maxlengthMsg) {
        command.setUuid(UUID.randomUUID());
        timestamp = new Timestamp(System.currentTimeMillis());
        command.setTimestamp(Constants.SDF.format(timestamp));
        dispatcherQueue.push(command);
        try (PrintWriter pw = new PrintWriter(new BufferedWriter(
            new FileWriter(commandFile, true)))) {
          pw.printf(
              "Evento enviado − UUID: {%s} − Timestamp: {%s} − Comando: {%s} − Mensaje: {%s}%n−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−%n",
              command.getUuid(), command.getTimestamp(),
              command.getCommand(), command.getMessage());
          System.out.printf(
              "Comando enviado − UUID: {%s} − Comando: {%s} − Usuario: {%s} − Mensaje: {%s} − Timestamp: {%s}",
              command.getUuid().toString(),
              command.getCommand().toString(),
              command.getUser(), command.getMessage(),
              command.getTimestamp());
        } catch(IOException e) {
          System.out.printf("No ha sido posible abrir el archivo de impresion de comandos: " + e);
        }
      } else {
        System.out.printf(
            "El mensaje contiene mas de 141 caracteres");
      }
    } catch (IOException e) {
      System.out.printf("Error al enviar el mensaje al dispatcher");
    }
  }
}
```

Oct 01, 17 11:48 **BatchUser.java** Page 1/2

```java
package ar.fiuba.taller.ClientConsole;

import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.concurrent.Callable;

import org.apache.log4j.Logger;
import org.apache.log4j.MDC;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import ar.fiuba.taller.common.Command;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.ReadingRemoteQueue;
import ar.fiuba.taller.common.WritingRemoteQueue;

public class BatchUser implements Callable {
  private String userName;
  private int commandAmount;
  private CommandController commandController;
  private Thread eventViewerThread;
  private ReadingRemoteQueue remoteUserResponseQueue;
  private WritingRemoteQueue dispatcherQueue;
  private long delayTime;
  final static Logger logger = Logger.getLogger(BatchUser.class);

  public BatchUser(Map<String, String> config, String userName,
      String userHost) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    this.userName = userName;
    commandAmount = Integer.parseInt(config.get(Constants.COMMAND_AMOUNT));
    try {
      dispatcherQueue = new WritingRemoteQueue(
          config.get(Constants.DISPATCHER_QUEUE_NAME),
          config.get(Constants.KAFKA_WRITE_PROPERTIES));
      remoteUserResponseQueue = new ReadingRemoteQueue(userName, config.get(Constants.KAFKA_READ_PROPERTIES));
    } catch (IOException e) {
      logger.error("No se han podido inicializar las colas de kafka: " + e);
      System.exit(1);
    }
    commandController =
        new CommandController(dispatcherQueue,
            Integer.parseInt(config.get(Constants.MAX_LENGTH_MSG)),
            Constants.LOGS_DIR + "/" + userName
                + Constants.COMMANDS_FILE_EXTENSION);
    eventViewerThread = new Thread(new EventWriter(
        Constants.LOGS_DIR + "/" + userName
            + Constants.EVENT_VIEWER_FILE_EXTENSION, remoteUserResponseQueue));
    delayTime = Long.parseLong(config.get(Constants.BATCH_DELAY_TIME));
  }

  @Override
  public Object call() throws Exception {
    logger.debug("Iniciando el script reader");
    int count = 0;

    eventViewerThread.start();

    try {
```

Oct 01, 17 11:48 **BatchUser.java** Page 2/2

```
66          JSONParser parser = new JSONParser();
67          Object obj = parser.parse(new FileReader(Constants.COMMAND_SCRIPT));
68          JSONObject jsonObject = (JSONObject) obj;
69          JSONArray commandArray = (JSONArray) jsonObject
70              .get(Constants.COMMAND_ARRAY);
71          JSONObject commandObject;
72          Command command;
73          List<Integer> commandIndexList = getCommandIndexList(commandAmount,
74              commandArray.size());
75          Iterator<Integer> iterator = commandIndexList.iterator();
76
77          while (iterator.hasNext()) {
78            commandObject = (JSONObject) commandArray.get(iterator.next());
79            command = new Command(
80                (String) commandObject.get(Constants.COMMAND_KEY),
81                userName,
82                (String) commandObject.get(Constants.MESSAGE_KEY), null,
83                null);
84            logger.debug("COMANDO: " + count
85                + ".Se inserto comando con los siguientes parametros: "
86                + "\nUsuario: " + command.getUser() + "\nComando: "
87                + command.getCommand() + "\nMensaje: "
88                + command.getMessage());
89            commandController.sendMessage(command);
90            ++count;
91          }
92        } catch (ParseException | IOException e) {
93          logger.error("Error al tratar el script de comandos: " + e);
94        }
95        return null;
96      }
97
98      private List<Integer> getCommandIndexList(int commandListIndexSize,
99          int maxCommandsAvailable) {
100       List<Integer> commandIndexList = new ArrayList<Integer>();
101
102       for (int i = 0; i < commandListIndexSize; i++) {
103         commandIndexList.add((int) (Math.random() * maxCommandsAvailable));
104       }
105
106       return commandIndexList;
107     }
108
109 }
```

Oct 01, 17 11:44 **MainAuditLogger.java** Page 1/1

```
1  package ar.fiuba.taller.auditLogger;
2
3  import java.io.IOException;
4  import org.apache.log4j.Logger;
5  import org.apache.log4j.MDC;
6  import org.apache.log4j.PropertyConfigurator;
7
8  import ar.fiuba.taller.common.ConfigLoader;
9  import ar.fiuba.taller.common.Constants;
10 import ar.fiuba.taller.common.ReadingRemoteQueue;
11
12 public class MainAuditLogger {
13   final static Logger logger = Logger.getLogger(MainAuditLogger.class);
14
15   public static void main(String[] args) throws Exception {
16     PropertyConfigurator.configure(Constants.LOGGER_CONF);
17     MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
18     ConfigLoader configLoader = null;
19
20     try {
21       configLoader = new ConfigLoader(Constants.CONF_FILE);
22     } catch (IOException e) {
23       logger.error("Error al cargar la configuracion");
24       System.exit(Constants.EXIT_FAILURE);
25     }
26
27     final ReadingRemoteQueue loggerQueue = new ReadingRemoteQueue(
28         configLoader.getProperties()
29             .get(Constants.AUDIT_LOGGER_QUEUE_NAME),
30         configLoader.getProperties()
31             .get(Constants.KAFKA_READ_PROPERTIES));
32
33     AuditLogger auditLogger = new AuditLogger(loggerQueue, configLoader.getPrope
   rties());
34     auditLogger.run();
35     loggerQueue.shutDown();
36     loggerQueue.close();
37   }
38 }
```

Oct 01, 17 9:29 **AuditLogger.java** Page 1/2

```
1  package ar.fiuba.taller.auditLogger;
2
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.io.IOException;
6  import java.io.PrintWriter;
7  import java.sql.Timestamp;
8  import java.util.List;
9  import java.util.Map;
10
11 import org.apache.log4j.Logger;
12 import org.apache.log4j.MDC;
13
14 import ar.fiuba.taller.common.*;
15
16 public class AuditLogger {
17   private Timestamp timestamp;
18   private ReadingRemoteQueue loggerQueue;
19   private Map<String, String> config;
20   final static Logger logger = Logger.getLogger(AuditLogger.class);
21
22   public AuditLogger(ReadingRemoteQueue loggerQueue,
23       Map<String, String> config) {
24     this.loggerQueue = loggerQueue;
25     this.config = config;
26   }
27
28   public void run() {
29     MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
30     List<byte[]> messageList = null;
31     Command command = new Command();
32     PrintWriter pw = null;
33
34     logger.info("Iniciando el audit logger");
35
36     try {
37       // Si no existe el archivo lo creo
38       pw = new PrintWriter(config.get(Constants.AUDIT_LOG_FILE), "UTF-8");
39       pw.close();
40
41       // Lo abro para realizar append
42       pw = new PrintWriter(new BufferedWriter(new FileWriter(
43           config.get(Constants.AUDIT_LOG_FILE), true)));
44
45       while (¬Thread.interrupted()) {
46         messageList = loggerQueue.pop();
47         for (byte[] message : messageList) {
48           try {
49             command.deserialize(message);
50             logger.info("Comando recibido: "
51                 + getAuditLogEntry(command));
52             pw.println(getAuditLogEntry(command));
53             pw.flush();
54           } catch (ClassNotFoundException | IOException e) {
55             logger.error("No se ha podido deserializar el mensaje");
56           }
57         }
58       }
59     } catch (IOException e) {
60       logger.error("No se ha podido abrir el archivo de log: " + e);
61     }
62     logger.info("Audit logger terminado");
63   }
64
65   private String getAuditLogEntry(Command command) {
66     timestamp = new Timestamp(System.currentTimeMillis());
```

Oct 01, 17 9:29 **AuditLogger.java** Page 2/2

```
67     return Constants.SDF.format(timestamp) + " - " + "UUID: "
68         + command.getUuid() + " - Usuario: " + command.getUser()
69         + " - Comando: " + command.getCommand() + " - Mensaje: "
70         + command.getMessage();
71   }
72
73 }
```

Oct 01, 17 9:32 **UserRegistry.java** Page 1/3

```
1  package ar.fiuba.taller.analyzer;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.FileOutputStream;
6  import java.io.FileReader;
7  import java.io.FileWriter;
8  import java.io.IOException;
9  import java.util.ArrayList;
10 import java.util.Iterator;
11 import java.util.List;
12 import java.util.regex.Matcher;
13 import java.util.regex.Pattern;
14
15 import org.apache.log4j.Logger;
16 import org.json.simple.JSONArray;
17 import org.json.simple.JSONObject;
18 import org.json.simple.parser.JSONParser;
19 import org.json.simple.parser.ParseException;
20
21 import ar.fiuba.taller.common.Constants;
22
23 public class UserRegistry {
24
25   final static Logger logger = Logger.getLogger(UserRegistry.class);
26
27   public UserRegistry() {
28   }
29
30   public void update(String follower, String followed)
31       throws IOException, ParseException {
32     String updateFile;
33     String updateKey;
34     JSONParser parser = new JSONParser();
35     ;
36     Object obj;
37     JSONObject jsonObject;
38     JSONArray jsonArray;
39     FileWriter file;
40
41     if (String.valueOf(followed.charAt(0)).equals("#")) {
42       // Si sigo un hastag => actualizo la base de seguidores del hashtag
43       updateFile = Constants.DB_DIR + "/" + Constants.DB_HASHTAG_INDEX;
44       updateKey = followed.substring(1, followed.length());
45     } else {
46       // Si no, asumo que es un usuario => actualizo la base de seguidores
47       // del usuario
48       updateFile = Constants.DB_DIR + "/" + Constants.DB_USER_INDEX;
49       updateKey = followed;
50     }
51
52     logger.info(
53         "Actualizando el inice: " + updateFile + " con " + updateKey);
54     File tmpFile = new File(updateFile);
55     if (tmpFile.createNewFile()) {
56       FileOutputStream oFile = new FileOutputStream(tmpFile, false);
57       oFile.write("{}".getBytes());
58     }
59
60     obj = parser.parse(new FileReader(tmpFile));
61     jsonObject = (JSONObject) obj;
62     JSONArray array = (JSONArray) jsonObject.get(updateKey);
63     if (array == null) {
64       // Hay que crear la entrada en el indice
65       JSONArray ar2 = new JSONArray();
66       ar2.add(follower);
```

Oct 01, 17 9:32 **UserRegistry.java** Page 2/3

```
67       jsonObject.put(updateKey, ar2);
68     } else {
69       array.add(follower);
70       jsonObject.put(updateKey, array);
71     }
72     file = new FileWriter(tmpFile);
73     try {
74       file.write(jsonObject.toJSONString());
75     } catch (Exception e) {
76       logger.error("Error al guardar el index: " + e);
77     } finally {
78       file.flush();
79       try {
80         file.close();
81       } catch (IOException e) {
82         logger.error("No se ha podido cerrar el archivo de registro: " + e);
83       }
84     }
85   }
86
87   public List<String> getUserFollowers(String followed)
88       throws FileNotFoundException, IOException, ParseException {
89     String usersFile = Constants.DB_DIR + "/" + Constants.DB_USER_INDEX;
90     JSONParser parser = new JSONParser();
91     Object obj;
92     JSONObject jsonObject;
93
94     logger.info("Buscando followers del usuario");
95
96     File tmpFile = new File(usersFile);
97     if (tmpFile.createNewFile()) {
98       FileOutputStream oFile = new FileOutputStream(tmpFile, false);
99       oFile.write("{}".getBytes());
100    }
101    obj = parser.parse(new FileReader(usersFile));
102    jsonObject = (JSONObject) obj;
103    JSONArray array = (JSONArray) jsonObject.get(followed);
104    if (array == null) {
105      array = new JSONArray();
106    }
107    return array;
108  }
109
110  public List<String> getHashtagFollowers(String followed)
111      throws FileNotFoundException, IOException, ParseException {
112    String hashtagFile = Constants.DB_DIR + "/"
113        + Constants.DB_HASHTAG_INDEX;
114    List<String> followersList = new ArrayList<String>();
115    JSONParser parser = new JSONParser();
116    Object obj;
117    JSONObject jsonObject;
118    JSONArray jsonArray;
119    Iterator<String> it;
120    String word;
121
122    logger.info("Buscando followers del hashtag");
123
124    File tmpFile = new File(hashtagFile);
125    if (tmpFile.createNewFile()) {
126      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
127      oFile.write("{}".getBytes());
128    }
129    logger.info("Obteniendo hashtags de " + followed);
130    obj = parser.parse(new FileReader(hashtagFile));
131    jsonObject = (JSONObject) obj;
132    String regexPattern = "(#\\w+)";
```

Oct 01, 17 9:32 **UserRegistry.java** Page 3/3

```
133       Pattern p = Pattern.compile(regexPattern);
134       Matcher m = p.matcher(followed);
135       while (m.find()) {
136         word = m.group(1).substring(1, m.group(1).length());
137         logger.info("Hashtag: " + m.group(1));
138         jsonArray = (JSONArray) jsonObject.get(word);
139         logger.info("arr: " + jsonArray);
140         if (jsonArray ≠ null) {
141           it = jsonArray.iterator();
142           while (it.hasNext()) {
143             followersList.add(it.next());
144           }
145         }
146       }
147       return followersList;
148     }
149   }
```

Oct 01, 17 11:43 **AnalyzerMain.java** Page 1/1

```
1   package ar.fiuba.taller.analyzer;
2
3   import java.io.IOException;
4   import java.util.concurrent.TimeoutException;
5
6   import org.apache.log4j.Logger;
7   import org.apache.log4j.MDC;
8   import org.apache.log4j.PropertyConfigurator;
9
10  import ar.fiuba.taller.common.ConfigLoader;
11  import ar.fiuba.taller.common.Constants;
12  import ar.fiuba.taller.common.ReadingRemoteQueue;
13
14  public class AnalyzerMain {
15    final static Logger logger = Logger.getLogger(AnalyzerMain.class);
16
17    public static void main(String[] args) {
18      MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
19      PropertyConfigurator.configure(Constants.LOGGER_CONF);
20      ConfigLoader configLoader = null;
21
22      logger.info("Iniciando el analyzer");
23
24      try {
25        configLoader = new ConfigLoader(Constants.CONF_FILE);
26      } catch (IOException e) {
27        logger.error("Error al cargar la configuracion");
28        System.exit(Constants.EXIT_FAILURE);
29      }
30
31      ReadingRemoteQueue analyzerQueue = null;
32      try {
33        analyzerQueue = new ReadingRemoteQueue(
34            configLoader.getProperties().get(Constants.ANALYZER_QUEUE_NAME),
35            configLoader.getProperties().get(Constants.KAFKA_READ_PROPERTIES));
36      } catch (IOException e1) {
37        logger.error("No se ha podido inicializar la cola de kafka: " + e1);
38        System.exit(Constants.EXIT_FAILURE);
39      }
40
41      AnalyzerController analyzerController = new AnalyzerController(
42          configLoader.getProperties(), analyzerQueue);
43      analyzerController.run();
44      analyzerQueue.shutDown();
45      try {
46        analyzerQueue.close();
47      } catch (IOException | TimeoutException e) {
48        // Do nothing
49        logger.error("No se ha podido cerrar la cola del analyzer: " + e);
50      }
51    }
52  }
```

Oct 01, 17 11:41 **AnalyzerController.java** Page 1/3

```java
package ar.fiuba.taller.analyzer;

import java.io.IOException;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.TimeoutException;

import org.apache.log4j.Logger;
import org.apache.log4j.MDC;
import org.json.simple.parser.ParseException;

import ar.fiuba.taller.common.Command;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
import ar.fiuba.taller.common.ReadingRemoteQueue;
import ar.fiuba.taller.common.Response;
import ar.fiuba.taller.common.WritingRemoteQueue;

public class AnalyzerController {

  private Map<String, String> config;
  private ReadingRemoteQueue analyzerQueue;
  private Map<String, WritingRemoteQueue> usersMap;
  private WritingRemoteQueue remoteQueue;
  private UserRegistry userRegistry;
  private List<String> userFollowers;
  private List<String> hashtagFollowers;
  private Set<String> usersSet;
  final static Logger logger = Logger.getLogger(AnalyzerController.class);

  public AnalyzerController(Map<String, String> config,
      ReadingRemoteQueue analyzerQueue) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    this.analyzerQueue = analyzerQueue;
    this.usersMap = new HashMap<String, WritingRemoteQueue>();
    this.config = config;
  }

  public void run() {
    Command command = new Command();
    Response response = new Response();
    List<byte[]> messageList = null;
    userRegistry = new UserRegistry();

    try {
      while (¬Thread.interrupted()) {
        messageList = analyzerQueue.pop();
        for (byte[] message : messageList) {
          try {
            command.deserialize(message);
            logger.info(
                "Comando recibido con los siguientes parametros: "
                    + "\nUUID: " + command.getUuid()
                    + "\nUsuario: " + command.getUser()
                    + "\nComando: " + command.getCommand()
                    + "\nMensaje: " + command.getMessage());
            response = new Response();
            response.setUuid(command.getUuid());
            response.setUser(command.getUser());
            switch (command.getCommand()) {
            case PUBLISH:
              response.setResponse_status(RESPONSE_STATUS.OK);
```

Oct 01, 17 11:41 **AnalyzerController.java** Page 2/3

```java
              response.setMessage(command.getTimestamp() + "\n"
                  + command.getUser() + "\n"
                  + command.getMessage());
              sendResponse(response);
              break;
            case FOLLOW:
              userRegistry.update(command.getUser(),
                  command.getMessage());
              response.setResponse_status(
                  RESPONSE_STATUS.REGISTERED);
              response.setMessage("Seguidor registrado");
              sendResponse(response);
              break;
            default:
              logger.info(
                  "Comando recibido invalido. Comando descartado.");
            }
          } catch (IOException | ParseException
              | ClassNotFoundException | TimeoutException e) {
            logger.error("Error al tratar el mensaje recibido: " + e);
          }
        }
      }
    } finally {
      Iterator it = usersMap.entrySet().iterator();
      while (it.hasNext()) {
        Map.Entry pair = (Map.Entry)it.next();
        WritingRemoteQueue userQueue = (WritingRemoteQueue) pair.getValue();
        try {
          userQueue.close();
        } catch (IOException | TimeoutException e) {
          // Do nothing
          logger.error("Error al cerrar una response user queue: " + e);
        }
        it.remove(); // avoids a ConcurrentModificationException
      }
    }
    logger.info("Analyzer reciver finalizado");
  }

  private void sendResponse(Response response) throws IOException, TimeoutException, ParseException {
    // Reviso si es un user register o un mensaje
    // Si da error o es una registracion, se lo devuelvo
    // solamente
    // al usuario que envio el request
    if (response
        .getResponse_status() ≡ RESPONSE_STATUS.REGISTERED
        ∨ response
            .getResponse_status() ≡ RESPONSE_STATUS.ERROR) {
      logger.info("Enviando respuesta");
      remoteQueue = getUserQueue(response.getUser());
      remoteQueue.push(response);
    } else {
      // Por Ok, hago anycast a los followers
      logger.info("Anycast a los followers");
      usersSet = new HashSet<String>();
      userFollowers = userRegistry
          .getUserFollowers(response.getUser());
      hashtagFollowers = userRegistry
          .getHashtagFollowers(response.getMessage());
      for (String follower : userFollowers) {
        usersSet.add(follower);
      }
      for (String follower : hashtagFollowers) {
        usersSet.add(follower);
```

Oct 01, 17 11:41 **AnalyzerController.java** Page 3/3

```
132         }
133         // Fowardeo el mensaje a los followers
134         Iterator<String> it = usersSet.iterator();
135         while (it.hasNext()) {
136           (getUserQueue(it.next())).push(response);
137         }
138       }
139     }
140
141     private WritingRemoteQueue getUserQueue(String username)
142         throws IOException, TimeoutException {
143       WritingRemoteQueue tmpQueue;
144       logger.info("Ususario a fowardear: " + username);
145       tmpQueue = usersMap.get(username);
146
147       if (tmpQueue == null) {
148         tmpQueue = new WritingRemoteQueue(username, config.get(Constants.KAFKA_WRI
    TE_PROPERTIES));
149         usersMap.put(username, tmpQueue);
150       }
151       return usersMap.get(username);
152     }
153
154 }
```

Oct 16, 17 12:15 **Table of Content** Page 1/1