```
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.StringReader;
13 import java.nio.ByteBuffer;
14 import java.nio.channels.FileChannel;
15 import java.nio.channels.FileLock;
16 import java.nio.file.Path;
17 import java.nio.file.Paths;
18 import java.nio.file.StandardOpenOption;
19 import java.util.ArrayList;
20 import java.util.Collections;
21 import java.util.HashMap;
22 import java.util.Iterator;
23 import java.util.LinkedHashMap;
24 import java.util.List;
25 import java.util.ListIterator;
26 import java.util.Map;
27 import java.util.regex.Matcher;
28 import java.util.regex.Pattern;
29
30 import org.apache.log4j.Logger;
31 import org.apache.log4j.MDC;
32 import org.json.simple.JSONArray;
33 import org.json.simple.JSONObject;
34 import org.json.simple.parser.JSONParser;
35 import org.json.simple.parser.ParseException;
36
37 import ar.fiuba.taller.common.Command;
38 import ar.fiuba.taller.common.Constants;
39
40 public class Storage {
41
42   private int shardingFactor;
43   private int queryCountShowPosts;
44   private int ttCountShowPosts;
45   final static Logger logger = Logger.getLogger(Storage.class);
46
47   public Storage(int shardingFactor, int queryCountShowPosts,
48       int ttCountShowPosts) {
49     this.shardingFactor = shardingFactor;
50     this.queryCountShowPosts = queryCountShowPosts;
51     this.ttCountShowPosts = ttCountShowPosts;
52     MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
53   }
54
55   private void updateTT(Command command) throws IOException, ParseException {
56     String fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_TT;
57     JSONParser parser = new JSONParser();
58     Object obj;
59
60     logger.info("Actualizando los TT");
61     File tmpFile = new File(fileName);
62     if (tmpFile.createNewFile()) {
63       FileOutputStream oFile = new FileOutputStream(tmpFile, false);
64       oFile.write("{}".getBytes());
65     }
66     Path path = Paths.get(fileName);
```



```
67     FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ);
68     FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
69     try {
70       ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
71       fileChannel.read(buffer);
72       buffer.position(0);
73       StringBuilder sb = new StringBuilder();
74           while (buffer.hasRemaining()) {
75               sb.append((char) buffer.get());
76           }
77           String tmp = sb.toString();
78           if((tmp.split("}", -1).length - 1) > 1) {
79             tmp = tmp.substring(0, tmp.indexOf("}")+1);
80           }
81       obj = parser.parse(new StringReader(tmp));
82       JSONObject jsonObject = (JSONObject) obj;
83       int count = 0;
84       String regexPattern = "(#\\w+)";
85       Pattern p = Pattern.compile(regexPattern);
86       Matcher m = p.matcher(command.getMessage());
87       String hashtag;
88       while (m.find()) {
89         hashtag = m.group(1);
90         hashtag = hashtag.substring(1, hashtag.length());
91         Long obj2 = (Long) jsonObject.get(hashtag);
92         if (obj2 == null) {
93           // La entrada no existe y hay que crearla
94           jsonObject.put(hashtag, 1);
95         } else {
96           obj2++;
97           jsonObject.put(hashtag, obj2);
98         }
99       }
100      lock.release();
101      fileChannel.close();
102      logger.debug("sssssss" + jsonObject.toJSONString());
103      fileChannel = FileChannel.open(path, StandardOpenOption.WRITE,
104                StandardOpenOption.TRUNCATE_EXISTING);
105      lock = fileChannel.lock(); // gets an exclusive lock
106        buffer = ByteBuffer.wrap(jsonObject.toJSONString().getBytes());
107        fileChannel.write(buffer);
108    } catch (Exception e) {
109      logger.error("Error guardar el indice de TT: " + e);
110    } finally {
111      lock.release();
112      fileChannel.close();
113    }
114  }
115
116  public void saveMessage(Command command)
117      throws IOException, ParseException {
118    String fileName = Constants.DB_DIR + "/"
119        + command.getUuid().toString().substring(0, shardingFactor)
120        + Constants.COMMAND_SCRIPT_EXTENSION;
121    JSONParser parser = new JSONParser();
122    Object obj;
123
124    logger.info("Guardando el comando en la base de datos: " + fileName);
125    logger.info("Contenido del registro: " + command.toJson());
126    File tmpFile = new File(fileName);
127    if (tmpFile.createNewFile()) {
128      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
129    }
130    JSONObject obj2 = new JSONObject();
131    obj2.put("command", command.getCommand().toString());
132    obj2.put("user", command.getUser());
```

```
133       obj2.put("message", command.getMessage());
134       obj2.put("timestamp", command.getTimestamp());
135       JSONObject jsonObject = new JSONObject();
136       jsonObject.put(command.getUuid().toString(), obj2);
137
138           Path path = Paths.get(fileName);
139           FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.WRIT
    E,
140                   StandardOpenOption.APPEND);
141           FileLock lock = fileChannel.lock(); // gets an exclusive lock
142       try {
143         ByteBuffer buffer = ByteBuffer.wrap((jsonObject.toJSONString() + String.fo
    rmat("%n")).getBytes());
144         fileChannel.write(buffer);
145       } catch (Exception e) {
146         logger.error("Error guardar la base de datos: " + e);
147       } finally {
148         lock.release();
149         fileChannel.close();
150       }
151       // Una vez que persisto el mensaje, actualizo los indices y el TT
152       updateUserIndex(command);
153       updateHashTagIndex(command);
154       updateTT(command);
155     }
156
157     private void updateUserIndex(Command command)
158         throws IOException, ParseException {
159       String fileName = Constants.DB_INDEX_DIR + "/"
160           + Constants.DB_USER_INDEX;
161       JSONParser parser = new JSONParser();
162       Object obj;
163
164       logger.info("Actualizando el inice de usuarios");
165       File tmpFile = new File(fileName);
166       if (tmpFile.createNewFile()) {
167         FileOutputStream oFile = new FileOutputStream(tmpFile, false);
168         oFile.write("{}".getBytes());
169       }
170
171           Path path = Paths.get(fileName);
172           FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ
    );
173           FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
174           try {
175       ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
176       fileChannel.read(buffer);
177       buffer.position(0);
178       StringBuilder sb = new StringBuilder();
179           while (buffer.hasRemaining()) {
180               sb.append((char) buffer.get());
181           }
182           String tmp = sb.toString();
183           if((tmp.split("}", -1).length - 1) > 1) {
184             tmp = tmp.substring(0, tmp.indexOf("}")+1);
185           }
186       obj = parser.parse(new StringReader(tmp));
187       JSONObject jsonObject = (JSONObject) obj;
188       JSONArray array = (JSONArray) jsonObject.get(command.getUser());
189       if (array == null) {
190         // Hay que crear la entrada en el indice
191         JSONArray ar2 = new JSONArray();
192         ar2.add(command.getUuid().toString());
193         jsonObject.put(command.getUser(), ar2);
194       } else {
195         array.add(command.getUuid().toString());
```



```
196         jsonObject.put(command.getUser(), array);
197       }
198       lock.release();
199       fileChannel.close();
200       fileChannel = FileChannel.open(path, StandardOpenOption.WRITE,
201                 StandardOpenOption.TRUNCATE_EXISTING);
202       lock = fileChannel.lock(); // gets an exclusive lock
203         buffer = ByteBuffer.wrap(jsonObject.toJSONString().getBytes());
204         fileChannel.write(buffer);
205       } catch (Exception e) {
206         logger.error("Error guardar el indice de u: " + e);
207       } finally {
208         lock.release();
209         fileChannel.close();
210       }
211     }
212
213     private void updateHashTagIndex(Command command)
214         throws IOException, ParseException {
215       String fileName = Constants.DB_INDEX_DIR + "/"
216           + Constants.DB_HASHTAG_INDEX;
217       JSONParser parser = new JSONParser();
218       Object obj;
219
220       logger.info("Actualizando el inice de hashtags");
221       File tmpFile = new File(fileName);
222       if (tmpFile.createNewFile()) {
223         FileOutputStream oFile = new FileOutputStream(tmpFile, false);
224         oFile.write("{}".getBytes());
225       }
226
227           Path path = Paths.get(fileName);
228           FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ
    );
229           FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
230           try {
231       ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
232       fileChannel.read(buffer);
233       buffer.position(0);
234       StringBuilder sb = new StringBuilder();
235           while (buffer.hasRemaining()) {
236               sb.append((char) buffer.get());
237           }
238           String tmp = sb.toString();
239           if((tmp.split("}", -1).length - 1) > 1) {
240             tmp = tmp.substring(0, tmp.indexOf("}")+1);
241           }
242       obj = parser.parse(new StringReader(tmp));
243       JSONObject jsonObject = (JSONObject) obj;
244       JSONArray array;
245       String regexPattern = "(#\\w+)";
246       Pattern p = Pattern.compile(regexPattern);
247       Matcher m = p.matcher(command.getMessage());
248       String hashtag;
249       JSONArray ar2;
250       while (m.find()) {
251         hashtag = m.group(1);
252         hashtag = hashtag.substring(1, hashtag.length());
253         array = (JSONArray) jsonObject.get(hashtag);
254         if (array == null) {
255           // Hay que crear la entrada en el indice
256           ar2 = new JSONArray();
257           ar2.add(command.getUuid().toString());
258           jsonObject.put(hashtag, ar2);
259         } else {
260           array.add(command.getUuid().toString());
```

```
261           jsonObject.put(hashtag, array);
262         }
263       }
264       lock.release();
265       fileChannel.close();
266       fileChannel = FileChannel.open(path, StandardOpenOption.WRITE,
267                 StandardOpenOption.TRUNCATE_EXISTING);
268       lock = fileChannel.lock(); // gets an exclusive lock
269         buffer = ByteBuffer.wrap(jsonObject.toJSONString().getBytes());
270         fileChannel.write(buffer);
271       } catch (Exception e) {
272         logger.error("Error guardar el indice de hashtags: " + e);
273       } finally {
274         lock.release();
275         fileChannel.close();
276       }
277     }
278
279     public String query(Command command) throws IOException, ParseException {
280       List<String> resultList = new ArrayList<String>();
281       String listString = "";
282       if (String.valueOf(command.getMessage().charAt(0)).equals("#")) { // #
283         resultList = queryBy(command.getMessage().substring(1,
284             command.getMessage().length()), "HASHTAG");
285       } else if (command.getMessage().equals("TT")) { // Es consulta por TT
286         resultList = queryTT(command.getMessage());
287       } else { // Es consulta por usuario
288         resultList = queryBy(command.getMessage(), "USER");
289       }
290       if(¬resultList.isEmpty()) {
291         for (String element : resultList) {
292           listString += element + "\n";
293         }
294       }
295       return listString;
296     }
297
298     private List<String> queryTT(String hashTag)
299         throws FileNotFoundException, IOException, ParseException {
300       Map<String, Long> map = new HashMap<String, Long>();
301       String fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_TT;
302       List<String> returnList = new ArrayList<String>();
303
304       // Levantar el json
305       JSONParser parser = new JSONParser();
306
307           Path path = Paths.get(fileName);
308           FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ
    );
309           FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
310           try {
311           ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
312       fileChannel.read(buffer);
313       buffer.position(0);
314       StringBuilder sb = new StringBuilder();
315           while (buffer.hasRemaining()) {
316               sb.append((char) buffer.get());
317           }
318
319       Object obj = parser.parse(new StringReader(sb.toString()));
320
321       JSONObject jsonObject = (JSONObject) obj;
322
323       // Crear un map
324       for (Iterator iterator = jsonObject.keySet().iterator(); iterator
325           .hasNext();) {
```



```
326         String key = (String) iterator.next();
327         map.put(key, (Long) jsonObject.get(key));
328       }
329
330       returnList = sortHashMapByValues(map);
331       returnList
332           .add("Total de topics: " + String.valueOf(map.keySet().size()));
333           } catch(Exception e) {
334             // Do nothing
335           } finally {
336       lock.release();
337       fileChannel.close();
338           }
339       return returnList;
340     }
341
342     private List<String> queryBy(String key, String type)
343         throws IOException, ParseException {
344       String fileName;
345       JSONParser parser = new JSONParser();
346       Object obj, obj2;
347       List<String> messageList = new ArrayList<String>();
348       String file, id;
349
350       if (type.equals("USER")) {
351         logger.info("Consultando por user");
352         fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_USER_INDEX;
353       } else if (type.equals("HASHTAG")) {
354         logger.info("Consultando por hashtag");
355         fileName = Constants.DB_INDEX_DIR + "/"
356             + Constants.DB_HASHTAG_INDEX;
357       } else {
358         return messageList;
359       }
360
361       // Obtengo la lista de archivos que contienen el user
362
363       File tmpFile = new File(fileName);
364       if (tmpFile.createNewFile()) {
365         FileOutputStream oFile = new FileOutputStream(tmpFile, false);
366         oFile.write("{}".getBytes());
367       }
368
369           Path path = Paths.get(fileName);
370           FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ
    );
371           FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
372           try {
373       ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
374       fileChannel.read(buffer);
375       buffer.position(0);
376       StringBuilder sb = new StringBuilder();
377           while (buffer.hasRemaining()) {
378               sb.append((char) buffer.get());
379           }
380
381       obj = parser.parse(new StringReader(sb.toString()));
382       JSONObject jsonObject = (JSONObject) obj;
383       JSONArray array = (JSONArray) jsonObject.get(key);
384
385       String line, reg;
386       JSONObject jsonObject2;
387       int remainingPost = queryCountShowPosts;
388       // Abro archivo por archivo y recupero los mensajes
389       if (array ≠ null) {
390         ListIterator<String> iterator = array.listIterator(array.size());
```

```
391         while (iterator.hasPrevious() ∧ remainingPost > 0) {
392           id = iterator.previous();
393           file = Constants.DB_DIR + "/" + id.substring(0, shardingFactor)
394               + Constants.COMMAND_SCRIPT_EXTENSION;
395           Path path2 = Paths.get(file);
396           FileChannel fileChannel2 = FileChannel.open(path2, StandardOpenOption.RE
    AD);
397           FileLock lock2 = fileChannel2.lock(0, Long.MAX_VALUE, true);
398           ByteBuffer buffer2 = ByteBuffer.allocate(((int) fileChannel2.size()));
399           fileChannel2.read(buffer2);
400           buffer2.position(0);
401           StringBuilder sb2 = new StringBuilder();
402           while (buffer2.hasRemaining()) {
403             sb2.append((char) buffer2.get());
404           }
405           try (
406             BufferedReader br = new BufferedReader(
407             new StringReader(sb2.toString()))
408           ) {
409             while ((line = br.readLine()) ≠ null ∧ remainingPost > 0
410                 ∧ ¬("").equals(line.trim())) {
411               System.out.println("line: " + line);
412               obj2 = parser.parse(line);
413               jsonObject2 = (JSONObject) obj2;
414               if (jsonObject2.get(id) ≠ null) {
415                 messageList.add(jsonObject2.get(id).toString());
416               }
417               remainingPost--;
418             }
419           }
420           lock2.release();
421           fileChannel2.close();
422         }
423       }
424           }catch(Exception e) {
425             // Do nothing
426           } finally {
427       lock.release();
428       fileChannel.close();
429           }
430       // Retorno la lista con los mensajes encontrados
431       return messageList;
432     }
433
434     public void delete(Command command)
435         throws IOException, ParseException {
436       String file = Constants.DB_DIR + "/"
437           + command.getMessage().substring(0, shardingFactor)
438           + Constants.COMMAND_SCRIPT_EXTENSION;
439       String fileTmp = file + ".tmp";
440       JSONParser parser = new JSONParser();
441       Object obj2;
442       String line, key;
443       JSONObject jsonObject2;
444
445       // Creo un archivo temporal
446       PrintWriter pw = new PrintWriter(
447           new BufferedWriter(new FileWriter(fileTmp)));
448
449       logger.info("Eleiminando registro");
450
451       try (BufferedReader br = new BufferedReader(new FileReader(file))) {
452         while ((line = br.readLine()) ≠ null) {
453           obj2 = parser.parse(line);
454           jsonObject2 = (JSONObject) obj2;
455           key = (String) jsonObject2.keySet().iterator().next();
```



```
456           if (¬(key.equals(command.getMessage()))) {
457             // Si no es la clave a borrar, guardo el registro en un
458             // archivo temporal
459             pw.println(jsonObject2);
460           }
461         }
462       }
463       pw.close();
464       // Borro el archvio original y renombro el tmp
465       File fileToDelete = new File(file);
466       File newFile = new File(fileTmp);
467       if (fileToDelete.delete()) {
468         logger.info("Archivo original borrado");
469         logger.info("Renombrado el archivo temporal al original");
470         if (newFile.renameTo(fileToDelete)) {
471           logger.info("Archivo renombrado con exito");
472         } else {
473           logger.error("No se ha podido renombrar el archivo");
474           throw new IOException();
475         }
476       } else {
477         logger.error(
478             "No se ha podido borrar el registro. Se aborta la operacion");
479         throw new IOException();
480       }
481     }
482
483     private List<String> sortHashMapByValues(Map<String, Long> map) {
484       List<String> mapKeys = new ArrayList<String>(map.keySet());
485       List<Long> mapValues = new ArrayList<Long>(map.values());
486       Collections.sort(mapValues);
487       Collections.sort(mapKeys);
488
489       LinkedHashMap<String, Long> sortedMap = new LinkedHashMap<String, Long>();
490
491       java.util.Iterator<Long> valueIt = mapValues.iterator();
492       while (valueIt.hasNext()) {
493         Long val = valueIt.next();
494         java.util.Iterator<String> keyIt = mapKeys.iterator();
495
496         while (keyIt.hasNext()) {
497           String key = keyIt.next();
498           Long comp1 = map.get(key);
499           Long comp2 = val;
500
501           if (comp1.equals(comp2)) {
502             keyIt.remove();
503             sortedMap.put(key, val);
504             break;
505           }
506         }
507       }
508       List<String> tt = new ArrayList<String>();
509       ArrayList<String> keys = new ArrayList<String>(sortedMap.keySet());
510       int i = keys.size() - 1;
511       int j = ttCountShowPosts;
512       while (i ≥ 0 ∧ j > 0) {
513         tt.add(keys.get(i));
514         j--;
515         i--;
516       }
517       return tt;
518     }
519
520   }
```

Oct 01, 17 20:08 **StorageController.java** Page 1/3

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

import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.UUID;
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.ReadingRemoteQueue;
import ar.fiuba.taller.common.Response;
import ar.fiuba.taller.common.WritingRemoteQueue;
import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;

public class StorageController {
  private Map<String, String> config;
  private Storage storage;
  private ReadingRemoteQueue storageQueue;
  private Map<String, WritingRemoteQueue> usersMap;
  final static Logger logger = Logger.getLogger(StorageController.class);

  public StorageController(Map<String, String> config,
      ReadingRemoteQueue storageQueue) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    this.config = config;
    storage = new Storage(
        Integer.parseInt(config.get(Constants.SHARDING_FACTOR)),
        Integer.parseInt(config.get(Constants.QUERY_COUNT_SHOW_POSTS)),
        Integer.parseInt(config.get(Constants.TT_COUNT_SHOW)));
    this.storageQueue = storageQueue;
    usersMap = new HashMap<String, WritingRemoteQueue>();
  }

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

    logger.info("Consumiendo de la storageQueue");
    try {
      while (¬Thread.interrupted()) {
        messageList = storageQueue.pop();
        for (byte[] message : messageList) {
          try {
            command = new Command();
            command.deserialize(message);
            analyzeCommand(command);

          } catch (ClassNotFoundException | IOException e) {
            logger.error("No se ha podido deserializar el mensaje");
          }
        }
      }
    } catch (InterruptedException e) {
      // Do nothing
      logger.error("Error al analizar comando: " + 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("Storgae Controller terminado");
  }

  private void analyzeCommand(Command command) throws InterruptedException {
    String error_message = "Error al crear el mensaje";
    Response response = new Response();

    logger.info("Comando recibido con los siguientes parametros: "
        + "\nUUID: " + command.getUuid() + "\nUsuario: "
        + command.getUser() + "\nComando: " + command.getCommand()
        + "\nMensaje: " + command.getMessage());

    response.setUuid(UUID.randomUUID());
    response.setUser(command.getUser());
    try {
      switch (command.getCommand()) {
      case PUBLISH:
        logger.info(
            "Comando recibido: PUBLISH. Insertando en la cola de creacion.");
        storage.saveMessage(command);
        response.setMessage("Creacion exitosa");
        response.setResponse_status(RESPONSE_STATUS.OK);
        break;
      case QUERY:
        logger.info(
            "Comando recibido: QUERY. Insertando en la cola de consultas.");
        response.setMessage(storage.query(command));
        logger.debug(response.getMessage());
        response.setResponse_status(RESPONSE_STATUS.OK);
        break;
      case DELETE:
        logger.info(
            "Comando recibido: DELETE. Insertando en la cola de borrado.");
        storage.delete(command);
        response.setMessage("Borrado exitoso");
        response.setResponse_status(RESPONSE_STATUS.OK);
        break;
      default:
        logger.info("Comando recibido invalido. Comando descartado.");
      }
    } catch (IOException e) {
      response.setResponse_status(RESPONSE_STATUS.ERROR);
      response.setMessage(error_message);
      logger.error(e);
    } catch (ParseException e) {
      response.setResponse_status(RESPONSE_STATUS.ERROR);
      response.setMessage(error_message);
      e.printStackTrace();
      logger.error(e);
    } finally {
      if (response ≠ null) {
        sendResponse(response);
        response = null;
      }
    }
  }
```

Oct 01, 17 20:08 **StorageController.java** Page 2/3

Oct 01, 17 20:08 **StorageController.java** Page 3/3

```
133
134   private void sendResponse(Response response) {
135     logger.info("Siguiente respuesta");
136     WritingRemoteQueue currentUserRemoteQueue;
137     currentUserRemoteQueue = usersMap.get(response.getUser());
138     if (currentUserRemoteQueue == null) {
139       // Creo la cola
140       try {
141         currentUserRemoteQueue = new WritingRemoteQueue(
142             response.getUser(), config.get(Constants.KAFKA_WRITE_PROPERTIES));
143       } catch (IOException e) {
144         logger.error("No se han podido crear las colas de kafka: " + e);
145         System.exit(1);
146       }
147       usersMap.put(response.getUser(), currentUserRemoteQueue);
148     }
149     logger.info(
150         "Enviando respuesta al usuario: " + response.getUser());
151     logger.info("UUID: " + response.getUuid());
152     logger.info("Status de la respuesta: "
153         + response.getResponse_status());
154     logger.info(
155         "Contenido de la respuesta: " + response.getMessage());
156     logger.info("Esperando siguiente respuesta");
157     try {
158       usersMap.get(response.getUser()).push(response);
159       logger.info("Respuesta enviada: " + response.getUser() + ":" + response.getMess
    age()
160       + ":" + response.getResponse_status() + ":" + response.getUuid());
161     } catch (IOException e) {
162       logger.error(
163           "No se ha podido enviar la respuesta al usuario "
164               + response.getUser());
165     }
166   }
167 }
```

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

```
1  package ar.fiuba.taller.storage;
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 MainStorage {
15   final static Logger logger = Logger.getLogger(MainStorage.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     try {
23       configLoader = new ConfigLoader(Constants.CONF_FILE);
24     } catch (IOException e) {
25       logger.error("Error al cargar la configuracion");
26       System.exit(Constants.EXIT_FAILURE);
27     }
28     ReadingRemoteQueue storageQueue = null;
29     try {
30       storageQueue = new ReadingRemoteQueue(
31           configLoader.getProperties().get(Constants.STORAGE_QUEUE_NAME),
32           configLoader.getProperties().get(Constants.KAFKA_READ_PROPERTIES));
33     } catch (IOException e1) {
34       logger.error("No se han podido inicializar las colas de kafka: " + e1);
35       System.exit(1);
36     }
37
38     StorageController storageController = new StorageController(
39         configLoader.getProperties(), storageQueue);
40
41     storageController.run();
42     storageQueue.shutDown();
43     try {
44       storageQueue.close();
45     } catch (IOException | TimeoutException e) {
46       // Do nothing
47       logger.error("No se ha podido cerrar la cola de entrada al storage: " + e);
48     }
49   }
50 }
```

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:
```

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

```
67                  storageQueue.push(command);
68                  analyzerQueue.push(command);
69                  loggerQueue.push(command);
70                  logger.info("Comando enviado al publish: "
71                      + "\nUsuario: " + command.getUser()
72                      + "\nComando: " + command.getCommand()
73                      + "\nMensaje: " + command.getMessage());
74                  break;
75                case QUERY:
76                  storageQueue.push(command);
77                  loggerQueue.push(command);
78                  logger.info("Comando enviado al query: "
79                      + "\nUsuario: " + command.getUser()
80                      + "\nComando: " + command.getCommand()
81                      + "\nMensaje: " + command.getMessage());
82                  break;
83                case DELETE:
84                  logger.info("Comando enviado al delete: "
85                      + "\nUsuario: " + command.getUser()
86                      + "\nComando: " + command.getCommand()
87                      + "\nMensaje: " + command.getMessage());
88                  storageQueue.push(command);
89                  loggerQueue.push(command);
90                  break;
91                case FOLLOW:
92                  logger.info("Comando enviado al follow: "
93                      + "\nUsuario: " + command.getUser()
94                      + "\nComando: " + command.getCommand()
95                      + "\nMensaje: " + command.getMessage());
96                  analyzerQueue.push(command);
97                  loggerQueue.push(command);
98                  break;
99                default:
100                 logger.error("Comando invalido");
101                 break;
102               }
103             } catch (ClassNotFoundException | IOException e) {
104               logger.error("No se ha podido deserializar el mensaje: " + e);
105             }
106           }
107         }
108       } finally {
109         try {
110           storageQueue.close();
111           dispatcherQueue.close();
112           analyzerQueue.close();
113         } catch (IOException | TimeoutException e) {
114           // Do nothing
115           logger.error("No se ha podido cerrar alguna de las colas");
116           logger.debug(e);
117         }
118       }
119       logger.info("Dispatcher controller terminado");
120     }
121   }
```

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

```
1  package ar.fiuba.taller.crea_deploy;
2
3  /**
4   * Hello world!
5   *
6   */
7  public class App
8  {
9      public static void main( String[] args )
10     {
11         System.out.println( "Hello World!" );
12     }
13 }
```

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

```
67    }
68
69    public RESPONSE_STATUS getResponse_status() {
70      return response_status;
71    }
72
73    public void setResponse_status(RESPONSE_STATUS response_status) {
74      this.response_status = response_status;
75    }
76
77    public String getMessage() {
78      return message;
79    }
80
81    public void setMessage(String message) {
82      this.message = message;
83    }
84
85    public String getUser() {
86      return user;
87    }
88
89    public void setUser(String user) {
90      this.user = user;
91    }
92  }
```

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

```
1  package ar.fiuba.taller.common;
2
3  import java.io.IOException;
4  import java.util.concurrent.TimeoutException;
5
6  public abstract class RemoteQueue {
7
8    public abstract void close() throws IOException, TimeoutException;
9
10 }
```

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

```
1  package ar.fiuba.taller.common;
2
3  import java.io.FileInputStream;
4  import java.io.FileNotFoundException;
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.util.ArrayList;
8  import java.util.Collections;
9  import java.util.List;
10 import java.util.Map;
11 import java.util.Properties;
12 import java.util.concurrent.TimeoutException;
13
14 import org.apache.kafka.clients.consumer.ConsumerConfig;
15 import org.apache.kafka.clients.consumer.ConsumerRecord;
16 import org.apache.kafka.clients.consumer.ConsumerRecords;
17 import org.apache.kafka.clients.consumer.KafkaConsumer;
18 import org.apache.kafka.common.errors.WakeupException;
19
20 public class ReadingRemoteQueue extends RemoteQueue {
21   private KafkaConsumer<byte[], byte[]> consumer;
22
23   public class ReadingRemoteQueueException extends WakeupException {
24   }
25
26   public ReadingRemoteQueue(String queueName,
27       String propertiesFile) throws IOException {
28     Properties consumerConfig = new Properties();
29     InputStream input = null;
30     input = new FileInputStream(propertiesFile);
31     consumerConfig.load(input);
32     consumer = new KafkaConsumer<byte[], byte[]>(consumerConfig);
33     consumer.subscribe(Collections.singletonList(queueName));
34     input.close();
35   }
36
37   @Override
38   public void close() throws IOException, TimeoutException {
39     consumer.close();
40   }
41
42   public void shutDown() {
43     consumer.wakeup();
44   }
45
46   public List<byte[]> pop() throws ReadingRemoteQueueException {
47     List<byte[]> msgList = null;
48
49     try {
50       while (msgList == null) {
51         ConsumerRecords<byte[], byte[]> records = consumer
52             .poll(Long.MAX_VALUE);
53         if (!records.isEmpty()) {
54           msgList = new ArrayList<byte[]>();
55           for (ConsumerRecord<byte[], byte[]> record : records) {
56             msgList.add(record.value());
57           }
58           consumer.commitSync();
59         }
60       }
61     } catch (WakeupException e) {
62       throw new ReadingRemoteQueueException();
63     }
64     return msgList;
65   }
66
```

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

```
67 }
```

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

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

import java.io.IOException;

public interface ISerialize {

  public byte[] serialize() throws IOException;

  public void deserialize(byte[] byteForm)
      throws IOException, ClassNotFoundException;

}
```

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

```
65    public static final String ANALYZER_QUEUE_NAME = "analyzer.queue.name";
66    public static final long ANALYZER_THREAD_WAIT_TIME = 5000;
67
68    public static final String DB_DIR = "db";
69    public static final String DB_INDEX_DIR = "idx";
70    public static final String DB_USER_INDEX = "user.json";
71    public static final String DB_HASHTAG_INDEX = "hashtag.json";
72    public static final String DB_TT = "tt.json";
73    public static final SimpleDateFormat SDF = new SimpleDateFormat(
74          "yyyy-MM-dd HH:mm:ss");
75
76    public static final String USER_READ_MODE = "r";
77    public static final String USER_WRITE_MODE = "w";
78
79    public static final String ACKS_CONFIG = "acks.config";
80    public static final String RETRIES_CONFIG = "retries.config";
81    public static final String KEY_SERIALIZER_CLASS_CONFIG = "key.serializer.class.config";
82    public static final String VALUE_SERIALIZER_CLASS_CONFIG = "value.serializer.class.confi
    g";
83    public static final String KEY_DESERIALIZER_CLASS_CONFIG = "key.deserializer.class.conf
    ig";
84    public static final String VALUE_DESERIALIZER_CLASS_CONFIG = "value.deserializer.class.
    config";
85    public static final String GROUP_ID_CONFIG = "group.id.config";
86    public static final String AUTO_OFFSET_RESET_CONFIG = "auto.offset.reset.config";
87
88    public static enum COMMAND {
89      PUBLISH, QUERY, DELETE, FOLLOW, EMPTY
90    };
91
92    public static Map<String, COMMAND> COMMAND_MAP;
93    static {
94      Map<String, COMMAND> tmpMap = new HashMap<String, Constants.COMMAND>();
95      tmpMap.put("PUBLISH", COMMAND.PUBLISH);
96      tmpMap.put("QUERY", COMMAND.QUERY);
97      tmpMap.put("DELETE", COMMAND.DELETE);
98      tmpMap.put("FOLLOW", COMMAND.FOLLOW);
99      COMMAND_MAP = Collections.unmodifiableMap(tmpMap);
100   }
101
102   public static enum RESPONSE_STATUS {
103     OK, ERROR, REGISTERED, EMPTY
104   }
105
106   public static Map<String, RESPONSE_STATUS> RESPONSE_STATUS_MAP;
107   static {
108     Map<String, RESPONSE_STATUS> tmpMap1 = new HashMap<String, RESPONSE_STATUS>(
    );
109     tmpMap1 = new HashMap<String, Constants.RESPONSE_STATUS>();
110     tmpMap1.put("OK", RESPONSE_STATUS.OK);
111     tmpMap1.put("ERROR", RESPONSE_STATUS.ERROR);
112     tmpMap1.put("REGISTERED", RESPONSE_STATUS.REGISTERED);
113     RESPONSE_STATUS_MAP = Collections.unmodifiableMap(tmpMap1);
114   }
115
116   public static final int EXIT_SUCCESS = 0;
117   public static final int EXIT_FAILURE = 1;
118 }
```

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

```
1  package ar.fiuba.taller.common;
2
3  import java.io.IOException;
4  import java.util.Collections;
5  import java.util.HashMap;
6  import java.util.Map;
7  import java.util.Properties;
8
9  public class ConfigLoader {
10
11   private Map<String, String> propertiesMap;
12
13   public ConfigLoader(String configFile) throws IOException {
14     propertiesMap = new HashMap<String, String>();
15     Properties properties = new Properties();
16     try {
17       properties.load(Thread.currentThread().getContextClassLoader()
18           .getResourceAsStream(Constants.CONF_FILE));
19     } catch (IOException e) {
20       System.err.println(
21           "No ha sido posible cargar el archivo de propiedades");
22       throw new IOException();
23     }
24     for (String key : properties.stringPropertyNames()) {
25       String value = properties.getProperty(key);
26       propertiesMap.put(key, value);
27     }
28
29     propertiesMap = Collections.unmodifiableMap(propertiesMap);
30   }
31
32   public Map<String, String> getProperties() {
33     return propertiesMap;
34   }
35 }
```

```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();
  }

```



```java
  public COMMAND getCommand() {
    return command;
  }

  public void setCommand(COMMAND command) {
    this.command = command;
  }

  public String getUser() {
    return user;
  }

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

  public String getMessage() {
    return message;
  }

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

  public UUID getUuid() {
    return uuid;
  }

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

  public String getTimestamp() {
    return timestamp;
  }

  public void setTimestamp(String timestamp) {
    this.timestamp = timestamp;
  }

  public String toJson() {
    String tmp;

    tmp = "{command:" + command.toString() + ",user:" + user + ",message:"
        + message + ",timestamp:" + timestamp + "}";
    return tmp;
  }

  public void fromJson(String jsonString) {

  }
}
```

```
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);
```



```
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 }
```

```
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) {
```



```
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 }
```

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

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

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

import ar.fiuba.taller.common.ReadingRemoteQueue;
import ar.fiuba.taller.common.Response;

public class EventWriter implements Runnable {
  private ReadingRemoteQueue remoteResponseQueue;
  private String eventFile;
  final static Logger logger = Logger.getLogger(EventWriter.class);

  public EventWriter(
      String eventFile, ReadingRemoteQueue remoteResponseQueue) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    this.remoteResponseQueue = remoteResponseQueue;
    this.eventFile = eventFile;
  }

  @SuppressWarnings("null")
  public void run() {
    Response response = new Response();
    FileWriter responseFile = null;
    PrintWriter pw;
    List<byte[]> messageList = null;

    logger.debug("Iniciando el event viewer");
    try {
      while (¬Thread.interrupted()) {
        messageList = remoteResponseQueue.pop();
          try {
            for (byte[] message : messageList) {
              response.deserialize(message);
              pw = new PrintWriter(new BufferedWriter(
                  new FileWriter(eventFile, true)));
              pw.printf(
                  "Evento recibido − UUID: {%s} − Status: {%s} − Mensaje: {%s}%n−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−%n",
                  response.getUuid(), response.getResponse_status(),
                  response.getMessage());
              pw.close();
            }
          } catch (IOException | ClassNotFoundException e) {
            logger.error("No se ha podido escribir la respuesta: " + e);
          }
      }
    } finally {
      try {
        if (null ≠ responseFile)
          try {
            responseFile.close();
          } catch (IOException e) {
            logger.error("No se ha podido cerrar el response file: " + e);
          }
      } catch (Exception e2) {
        logger.error("Error al cerrar el archivo " + eventFile + ":" + e2);
      }
    }

  }

}
```

Oct 01, 17 9:27 **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) {
    PrintWriter pw;

    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 {
          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());
          pw.close();
          System.out.printf(
              "Comando enviado − UUID: {%s} − Comando: {%s} − Usuario: {%s} − Mensaje: {%s} − Times
tamp: {%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(Cons
tants.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 }
```

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

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Timestamp;
import java.util.List;
import java.util.Map;

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

import ar.fiuba.taller.common.*;

public class AuditLogger {
  private Timestamp timestamp;
  private ReadingRemoteQueue loggerQueue;
  private Map<String, String> config;
  final static Logger logger = Logger.getLogger(AuditLogger.class);

  public AuditLogger(ReadingRemoteQueue loggerQueue,
      Map<String, String> config) {
    this.loggerQueue = loggerQueue;
    this.config = config;
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    List<byte[]> messageList = null;
    Command command = new Command();
    PrintWriter pw = null;

    logger.info("Iniciando el audit logger");

    try {
      // Si no existe el archivo lo creo
      pw = new PrintWriter(config.get(Constants.AUDIT_LOG_FILE), "UTF-8");
      pw.close();

      // Lo abro para realizar append
      pw = new PrintWriter(new BufferedWriter(new FileWriter(
          config.get(Constants.AUDIT_LOG_FILE), true)));

      while (¬Thread.interrupted()) {
        messageList = loggerQueue.pop();
        for (byte[] message : messageList) {
          try {
            command.deserialize(message);
            logger.info("Comando recibido: "
                + getAuditLogEntry(command));
            pw.println(getAuditLogEntry(command));
            pw.flush();
          } catch (ClassNotFoundException | IOException e) {
            logger.error("No se ha podido deserializar el mensaje");
          }
        }
      }
    } catch (IOException e) {
      logger.error("No se ha podido abrir el archivo de log: " + e);
    }
    logger.info("Audit logger terminado");
  }

  private String getAuditLogEntry(Command command) {
    timestamp = new Timestamp(System.currentTimeMillis());
```



```java
    return Constants.SDF.format(timestamp) + " - " + "UUID: "
        + command.getUuid() + " - Usuario: " + command.getUser()
        + " - Comando: " + command.getCommand() + " - Mensaje: "
        + command.getMessage();
  }

}
```

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

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.log4j.Logger;
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.Constants;

public class UserRegistry {

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

  public UserRegistry() {
  }

  public void update(String follower, String followed)
      throws IOException, ParseException {
    String updateFile;
    String updateKey;
    JSONParser parser = new JSONParser();
    ;
    Object obj;
    JSONObject jsonObject;
    JSONArray jsonArray;
    FileWriter file;

    if (String.valueOf(followed.charAt(0)).equals("#")) {
      // Si sigo un hastag => actualizo la base de seguidores del hashtag
      updateFile = Constants.DB_DIR + "/" + Constants.DB_HASHTAG_INDEX;
      updateKey = followed.substring(1, followed.length());
    } else {
      // Si no, asumo que es un usuario => actualizo la base de seguidores
      // del usuario
      updateFile = Constants.DB_DIR + "/" + Constants.DB_USER_INDEX;
      updateKey = followed;
    }

    logger.info(
        "Actualizando el inice: " + updateFile + " con " + updateKey);
    File tmpFile = new File(updateFile);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
      oFile.write("{}".getBytes());
    }

    obj = parser.parse(new FileReader(tmpFile));
    jsonObject = (JSONObject) obj;
    JSONArray array = (JSONArray) jsonObject.get(updateKey);
    if (array == null) {
      // Hay que crear la entrada en el indice
      JSONArray ar2 = new JSONArray();
      ar2.add(follower);
      jsonObject.put(updateKey, ar2);
    } else {
      array.add(follower);
      jsonObject.put(updateKey, array);
    }
    file = new FileWriter(tmpFile);
    try {
      file.write(jsonObject.toJSONString());
    } catch (Exception e) {
      logger.error("Error al guardar el index: " + e);
    } finally {
      file.flush();
      try {
        file.close();
      } catch (IOException e) {
        logger.error("No se ha podido cerrar el archivo de registro: " + e);
      }
    }
  }

  public List<String> getUserFollowers(String followed)
      throws FileNotFoundException, IOException, ParseException {
    String usersFile = Constants.DB_DIR + "/" + Constants.DB_USER_INDEX;
    JSONParser parser = new JSONParser();
    Object obj;
    JSONObject jsonObject;

    logger.info("Buscando followers del usuario");

    File tmpFile = new File(usersFile);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
      oFile.write("{}".getBytes());
    }
    obj = parser.parse(new FileReader(usersFile));
    jsonObject = (JSONObject) obj;
    JSONArray array = (JSONArray) jsonObject.get(followed);
    if (array == null) {
      array = new JSONArray();
    }
    return array;
  }

  public List<String> getHashtagFollowers(String followed)
      throws FileNotFoundException, IOException, ParseException {
    String hashtagFile = Constants.DB_DIR + "/"
        + Constants.DB_HASHTAG_INDEX;
    List<String> followersList = new ArrayList<String>();
    JSONParser parser = new JSONParser();
    Object obj;
    JSONObject jsonObject;
    JSONArray jsonArray;
    Iterator<String> it;
    String word;

    logger.info("Buscando followers del hashtag");

    File tmpFile = new File(hashtagFile);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
      oFile.write("{}".getBytes());
    }
    logger.info("Obteniendo hashtags de " + followed);
    obj = parser.parse(new FileReader(hashtagFile));
    jsonObject = (JSONObject) obj;
    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 02, 17 6:46 **Table of Content** Page 1/1