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

import java.io.BufferedReader;
import java.io.BufferedWriter;
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.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.ListIterator;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

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;

public class Storage {

  private int shardingFactor;
  private int queryCountShowPosts;
  private int ttCountShowPosts;
  final static Logger logger = Logger.getLogger(Storage.class);

  public Storage(int shardingFactor, int queryCountShowPosts,
      int ttCountShowPosts) {
    this.shardingFactor = shardingFactor;
    this.queryCountShowPosts = queryCountShowPosts;
    this.ttCountShowPosts = ttCountShowPosts;
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
  }

  public synchronized void create(Command command)
      throws IOException, ParseException {
    saveMessage(command);
  }

  private void updateTT(Command command) throws IOException, ParseException {
    String fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_TT;
    JSONParser parser = new JSONParser();
    Object obj;

    logger.info("Actualizando los TT");
    File tmpFile = new File(fileName);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
      oFile.write("{}".getBytes());
    }

    obj = parser.parse(new FileReader(fileName));
    JSONObject jsonObject = (JSONObject) obj;
```



```java
    int count = 0;
    String regexPattern = "(#\\w+)";
    Pattern p = Pattern.compile(regexPattern);
    Matcher m = p.matcher(command.getMessage());
    String hashtag;
    while (m.find()) {
      hashtag = m.group(1);
      hashtag = hashtag.substring(1, hashtag.length());
      Long obj2 = (Long) jsonObject.get(hashtag);
      if (obj2 == null) {
        // La entrada no existe y hay que crearla
        jsonObject.put(hashtag, 1);
      } else {
        obj2++;
        jsonObject.put(hashtag, obj2);
      }

    }

    FileWriter file = new FileWriter(fileName);
    try {
      file.write(jsonObject.toJSONString());
    } catch (Exception e) {
      logger.error("Error guardar el indice de hashtags");
      logger.info(e.toString());
      e.printStackTrace();
    } finally {
      file.flush();
      file.close();
    }
  }

  public void saveMessage(Command command)
      throws IOException, ParseException {
    String fileName = Constants.DB_DIR + "/"
        + command.getUuid().toString().substring(0, shardingFactor)
        + Constants.COMMAND_SCRIPT_EXTENSION;
    JSONParser parser = new JSONParser();
    Object obj;

    logger.info("Guardando el comando en la base de datos: " + fileName);
    logger.info("Contenido del registro: " + command.toJson());
    File tmpFile = new File(fileName);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
    }
    JSONObject obj2 = new JSONObject();
    obj2.put("command", command.getCommand().toString());
    obj2.put("user", command.getUser());
    obj2.put("message", command.getMessage());
    obj2.put("timestamp", command.getTimestamp());
    JSONObject jsonObject = new JSONObject();
    jsonObject.put(command.getUuid().toString(), obj2);
    FileWriter file = new FileWriter(fileName, true);
    try {
      file.write(jsonObject.toJSONString() + String.format("%n"));
    } catch (Exception e) {
      logger.error("Error guardar la base de datos");
      logger.info(e.toString());
      e.printStackTrace();
    } finally {
      file.flush();
      file.close();
    }
    // Una vez que persisto el mensaje, actualizo los indices y el TT
    updateUserIndex(command);
```

```
133       updateHashTagIndex(command);
134       updateTT(command);
135     }
136
137     private void updateUserIndex(Command command)
138         throws IOException, ParseException {
139       String fileName = Constants.DB_INDEX_DIR + "/"
140           + Constants.DB_USER_INDEX;
141       JSONParser parser = new JSONParser();
142       Object obj;
143
144       logger.info("Actualizando el inice de usuarios");
145       File tmpFile = new File(fileName);
146       if (tmpFile.createNewFile()) {
147         FileOutputStream oFile = new FileOutputStream(tmpFile, false);
148         oFile.write("{}".getBytes());
149       }
150       obj = parser.parse(new FileReader(fileName));
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       FileWriter file = new FileWriter(fileName);
163       try {
164         file.write(jsonObject.toJSONString());
165       } catch (Exception e) {
166         logger.error("Error al guardar el user index");
167         logger.info(e.toString());
168         e.printStackTrace();
169       } finally {
170         file.flush();
171         file.close();
172       }
173     }
174
175     private void updateHashTagIndex(Command command)
176         throws IOException, ParseException {
177       String fileName = Constants.DB_INDEX_DIR + "/"
178           + Constants.DB_HASHTAG_INDEX;
179       JSONParser parser = new JSONParser();
180       Object obj;
181
182       logger.info("Actualizando el inice de hashtags");
183       File tmpFile = new File(fileName);
184       if (tmpFile.createNewFile()) {
185         FileOutputStream oFile = new FileOutputStream(tmpFile, false);
186         oFile.write("{}".getBytes());
187       }
188       obj = parser.parse(new FileReader(fileName));
189       JSONObject jsonObject = (JSONObject) obj;
190       JSONArray array;
191       String regexPattern = "(#\\w+)";
192       Pattern p = Pattern.compile(regexPattern);
193       Matcher m = p.matcher(command.getMessage());
194       String hashtag;
195       JSONArray ar2;
196       while (m.find()) {
197         hashtag = m.group(1);
198         hashtag = hashtag.substring(1, hashtag.length());
```



```
199         array = (JSONArray) jsonObject.get(hashtag);
200         if (array == null) {
201           // Hay que crear la entrada en el indice
202           ar2 = new JSONArray();
203           ar2.add(command.getUuid().toString());
204           jsonObject.put(hashtag, ar2);
205         } else {
206           array.add(command.getUuid().toString());
207           jsonObject.put(hashtag, array);
208         }
209       }
210       FileWriter file = new FileWriter(fileName);
211       try {
212         file.write(jsonObject.toJSONString());
213       } catch (Exception e) {
214         logger.error("Error guardar el indice de hashtags");
215         logger.info(e.toString());
216         e.printStackTrace();
217       } finally {
218         file.flush();
219         file.close();
220       }
221     }
222
223     public String query(Command command) throws IOException, ParseException {
224       List<String> resultList;
225       String listString = "";
226       if (String.valueOf(command.getMessage().charAt(0)).equals("#")) { // Es
227                                                 // consulta
228                                                 // por
229                                                 // hashtag
230         resultList = queryBy(command.getMessage().substring(1,
231             command.getMessage().length()), "HASHTAG");
232       } else if (command.getMessage().equals("TT")) { // Es consulta por TT
233         resultList = queryTT(command.getMessage());
234       } else { // Es consulta por usuario
235         resultList = queryBy(command.getMessage(), "USER");
236       }
237       for (String element : resultList) {
238         listString += element + "\n";
239       }
240
241       return listString;
242     }
243
244     private List<String> queryTT(String hashTag)
245         throws FileNotFoundException, IOException, ParseException {
246       Map<String, Long> map = new HashMap<String, Long>();
247       String fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_TT;
248       List<String> returnList = null;
249
250       // Levantar el json
251       JSONParser parser = new JSONParser();
252
253       Object obj = parser.parse(new FileReader(fileName));
254
255       JSONObject jsonObject = (JSONObject) obj;
256
257       // Crear un map
258       for (Iterator iterator = jsonObject.keySet().iterator(); iterator
259           .hasNext();) {
260         String key = (String) iterator.next();
261         map.put(key, (Long) jsonObject.get(key));
262       }
263
264       returnList = sortHashMapByValues(map);
```

```
265       returnList
266           .add("Total de topics: " + String.valueOf(map.keySet().size()));
267       return returnList;
268     }
269
270     private List<String> queryBy(String key, String type)
271         throws IOException, ParseException {
272       String fileName;
273       JSONParser parser = new JSONParser();
274       Object obj, obj2;
275       List<String> messageList = new ArrayList<String>();
276       String file, id;
277
278       if (type.equals("USER")) {
279         logger.info("Consultando por user");
280         fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_USER_INDEX;
281       } else if (type.equals("HASHTAG")) {
282         logger.info("Consultando por hashtag");
283         fileName = Constants.DB_INDEX_DIR + "/"
284             + Constants.DB_HASHTAG_INDEX;
285       } else {
286         return null;
287       }
288
289       // Obtengo la lista de archivos que contienen el user
290
291       File tmpFile = new File(fileName);
292       if (tmpFile.createNewFile()) {
293         FileOutputStream oFile = new FileOutputStream(tmpFile, false);
294         oFile.write("{}".getBytes());
295       }
296       obj = parser.parse(new FileReader(fileName));
297       JSONObject jsonObject = (JSONObject) obj;
298       JSONArray array = (JSONArray) jsonObject.get(key);
299
300       String line, reg;
301       JSONObject jsonObject2;
302       int remainingPost = queryCountShowPosts;
303       // Abro archivo por archivo y recupero los mensajes
304       if (array ≠ null) {
305         ListIterator<String> iterator = array.listIterator(array.size());
306         while (iterator.hasPrevious() ∧ remainingPost > 0) {
307           id = iterator.previous();
308           System.out.println("id: " + id);
309           file = Constants.DB_DIR + "/" + id.substring(0, shardingFactor)
310               + Constants.COMMAND_SCRIPT_EXTENSION;
311           System.out.println("file: " + file);
312           try (BufferedReader br = new BufferedReader(
313               new FileReader(file))) {
314             while ((line = br.readLine()) ≠ null ∧ remainingPost > 0
315                 ∧ ¬("").equals(line.trim())) {
316               System.out.println("line: " + line);
317               obj2 = parser.parse(line);
318               jsonObject2 = (JSONObject) obj2;
319               if (jsonObject2.get(id) ≠ null) {
320                 messageList.add(jsonObject2.get(id).toString());
321               }
322               remainingPost--;
323             }
324           }
325         }
326       }
327       // Retorno la lista con los mensajes encontrados
328       return messageList;
329     }
330
```



```
331     public synchronized void delete(Command command)
332         throws IOException, ParseException {
333       String file = Constants.DB_DIR + "/"
334           + command.getMessage().substring(0, shardingFactor)
335           + Constants.COMMAND_SCRIPT_EXTENSION;
336       String fileTmp = file + ".tmp";
337       JSONParser parser = new JSONParser();
338       Object obj2;
339       String line, key;
340       JSONObject jsonObject2;
341
342       // Creo un archivo temporal
343       PrintWriter pw = new PrintWriter(
344           new BufferedWriter(new FileWriter(fileTmp)));
345
346       logger.info("Eleiminando registro");
347
348       try (BufferedReader br = new BufferedReader(new FileReader(file))) {
349         while ((line = br.readLine()) ≠ null) {
350           System.out.println("line: " + line);
351           obj2 = parser.parse(line);
352           jsonObject2 = (JSONObject) obj2;
353           key = (String) jsonObject2.keySet().iterator().next();
354           if (¬(key.equals(command.getMessage()))) {
355             // Si no es la clave a borrar, guardo el registro en un
356             // archivo temporal
357             pw.println(jsonObject2);
358           }
359         }
360       }
361       pw.close();
362       // Borro el archvio original y renombro el tmp
363       File fileToDelete = new File(file);
364       File newFile = new File(fileTmp);
365       if (fileToDelete.delete()) {
366         logger.info("Archivo original borrado");
367         logger.info("Renombrado el archivo temporal al original");
368         if (newFile.renameTo(fileToDelete)) {
369           logger.info("Archivo renombrado con exito");
370         } else {
371           logger.error("No se ha podido renombrar el archivo");
372           throw new IOException();
373         }
374       } else {
375         logger.error(
376             "No se ha podido borrar el registro. Se aborta la operacion");
377         throw new IOException();
378       }
379     }
380
381     private List<String> sortHashMapByValues(Map<String, Long> map) {
382       List<String> mapKeys = new ArrayList<String>(map.keySet());
383       List<Long> mapValues = new ArrayList<Long>(map.values());
384       Collections.sort(mapValues);
385       Collections.sort(mapKeys);
386
387       LinkedHashMap<String, Long> sortedMap = new LinkedHashMap<String, Long>();
388
389       java.util.Iterator<Long> valueIt = mapValues.iterator();
390       while (valueIt.hasNext()) {
391         Long val = valueIt.next();
392         java.util.Iterator<String> keyIt = mapKeys.iterator();
393
394         while (keyIt.hasNext()) {
395           String key = keyIt.next();
396           Long comp1 = map.get(key);
```

sep 14, 17 6:59 **Storage.java** Page 7/7

```
397           Long comp2 = val;
398
399           if (comp1.equals(comp2)) {
400             keyIt.remove();
401             sortedMap.put(key, val);
402             break;
403           }
404         }
405       }
406       List<String> tt = new ArrayList<String>();
407       ArrayList<String> keys = new ArrayList<String>(sortedMap.keySet());
408       int i = keys.size() − 1;
409       int j = ttCountShowPosts;
410       while (i ≥ 0 ∧ j > 0) {
411         tt.add(keys.get(i));
412         j−−;
413         i−−;
414       }
415       return tt;
416     }
417
418 }
```

sep 14, 17 6:59 **StorageController.java** Page 1/2

```
1  package ar.fiuba.taller.storage;
2
3  import java.io.IOException;
4  import java.util.List;
5  import java.util.Map;
6  import java.util.concurrent.ArrayBlockingQueue;
7  import java.util.concurrent.BlockingQueue;
8  import org.apache.log4j.Logger;
9  import org.apache.log4j.MDC;
10
11 import ar.fiuba.taller.common.Command;
12 import ar.fiuba.taller.common.Constants;
13 import ar.fiuba.taller.common.ReadingRemoteQueue;
14 import ar.fiuba.taller.common.ReadingRemoteQueueException;
15 import ar.fiuba.taller.common.Response;
16
17 public class StorageController implements Runnable {
18   private Thread createControllerThread;
19   private Thread queryControllerThread;
20   private Thread removeControllerThread;
21   private Thread responseControllerThread;
22   private BlockingQueue<Command> queryQueue;
23   private BlockingQueue<Command> removeQueue;
24   private BlockingQueue<Command> createQueue;
25   private BlockingQueue<Response> responseQueue;
26   private Storage storage;
27   private ReadingRemoteQueue storageQueue;
28   final static Logger logger = Logger.getLogger(StorageController.class);
29
30   public StorageController(Map<String, String> config,
31       ReadingRemoteQueue storageQueue) {
32     MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
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     queryQueue = new ArrayBlockingQueue<Command>(
39         Constants.COMMAND_QUEUE_SIZE);
40     removeQueue = new ArrayBlockingQueue<Command>(
41         Constants.COMMAND_QUEUE_SIZE);
42     createQueue = new ArrayBlockingQueue<Command>(
43         Constants.COMMAND_QUEUE_SIZE);
44     responseQueue = new ArrayBlockingQueue<Response>(
45         Constants.RESPONSE_QUEUE_SIZE);
46     queryControllerThread = new Thread(
47         new QueryController(queryQueue, responseQueue, storage));
48     removeControllerThread = new Thread(
49         new RemoveController(removeQueue, responseQueue, storage));
50     createControllerThread = new Thread(
51         new CreateController(createQueue, responseQueue,
52             Integer.parseInt(config.get(Constants.SHARDING_FACTOR)),
53             storage));
54     responseControllerThread = new Thread(
55         new ResponseController(responseQueue, config));
56   }
57
58   public void run() {
59     Command command;
60     List<byte[]> messageList = null;
61
62     logger.info("Lanzando los threads de query, remove y create");
63     queryControllerThread.start();
64     removeControllerThread.start();
65     createControllerThread.start();
66     responseControllerThread.start();
```

sep 14, 17 6:59 **StorageController.java** Page 2/2

```
67
68      logger.info("Consumiendo de la storageQueue");
69      try {
70        while (¬Thread.interrupted()) {
71          messageList = storageQueue.pop();
72          for (byte[] message : messageList) {
73            try {
74              command = new Command();
75              command.deserialize(message);
76              analyzeCommand(command);
77
78            } catch (ClassNotFoundException | IOException e) {
79              logger.error("No se ha podido deserializar el mensaje");
80            }
81          }
82        }
83      } catch (ReadingRemoteQueueException | InterruptedException e) {
84        queryControllerThread.interrupt();
85        removeControllerThread.interrupt();
86        createControllerThread.interrupt();
87        responseControllerThread.interrupt();
88        try {
89          queryControllerThread.join(Constants.STORAGE_THREAD_WAIT_TIME);
90          removeControllerThread.join(Constants.STORAGE_THREAD_WAIT_TIME);
91          createControllerThread.join(Constants.STORAGE_THREAD_WAIT_TIME);
92          responseControllerThread
93              .join(Constants.STORAGE_THREAD_WAIT_TIME);
94        } catch (InterruptedException e1) {
95          logger.error("Fallo el join de alguno de los threads");
96          logger.debug(e1);
97        }
98      }
99      logger.info("Storgae Controller terminado");
100   }
101
102   private void analyzeCommand(Command command) throws InterruptedException {
103     logger.info("Comando recibido con los siguientes parametros: "
104         + "\nUUID: " + command.getUuid() + "\nUsuario: "
105         + command.getUser() + "\nComando: " + command.getCommand()
106         + "\nMensaje: " + command.getMessage());
107
108     switch (command.getCommand()) {
109     case PUBLISH:
110       logger.info(
111           "Comando recibido: PUBLISH. Insertando en la cola de creacion.");
112       createQueue.put(command);
113       break;
114     case QUERY:
115       logger.info(
116           "Comando recibido: QUERY. Insertando en la cola de consultas.");
117       queryQueue.put(command);
118       break;
119     case DELETE:
120       logger.info(
121           "Comando recibido: DELETE. Insertando en la cola de borrado.");
122       removeQueue.put(command);
123       break;
124     default:
125       logger.info("Comando recibido invalido. Comando descartado.");
126     }
127   }
128 }
```

sep 09, 17 18:46 **ResponseController.java** Page 1/1

```
1  package ar.fiuba.taller.storage;
2
3  import java.util.concurrent.BlockingQueue;
4  import org.apache.log4j.Logger;
5
6  import ar.fiuba.taller.common.Response;
7  import ar.fiuba.taller.common.WritingRemoteQueue;
8
9  import java.io.IOException;
10 import java.util.*;
11
12 public class ResponseController implements Runnable {
13
14   private BlockingQueue<Response> responseQueue;
15   private Map<String, WritingRemoteQueue> usersMap;
16   private Map<String, String> config;
17   final static Logger logger = Logger.getLogger(ResponseController.class);
18
19   public ResponseController(BlockingQueue<Response> responseQueue,
20       Map<String, String> config) {
21     this.responseQueue = responseQueue;
22     usersMap = new HashMap<String, WritingRemoteQueue>();
23     this.config = config;
24   }
25
26   public void run() {
27     logger.info("Iniciando el response controller");
28     Response response = new Response();
29     WritingRemoteQueue currentUserRemoteQueue;
30
31     try {
32       while (¬Thread.interrupted()) {
33         logger.info("Esperando siguiente respuesta");
34         response = responseQueue.take();
35         currentUserRemoteQueue = usersMap.get(response.getUser());
36         if (currentUserRemoteQueue ≡ null) {
37           // Creo la cola
38           currentUserRemoteQueue = new WritingRemoteQueue(
39               response.getUser(), "localhost:9092", config);
40           usersMap.put(response.getUser(), currentUserRemoteQueue);
41         }
42         logger.info(
43             "Enviando respuesta al usuario: " + response.getUser());
44         logger.info("UUID: " + response.getUuid());
45         logger.info("Status de la respuesta: "
46             + response.getResponse_status());
47         logger.info(
48             "Contenido de la respuesta: " + response.getMessage());
49         logger.info("Esperando siguiente respuesta");
50         try {
51           usersMap.get(response.getUser()).push(response);
52           logger.info("Respuesta enviada");
53         } catch (IOException e) {
54           logger.error(
55               "No se ha podido enviar la respuesta al usuario "
56                   + response.getUser());
57         }
58       }
59     } catch (InterruptedException e) {
60       logger.info("ResponseController interrumpido");
61     }
62
63   }
64
65 }
```

sep 14, 17 6:59 **RemoveController.java** Page 1/1

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

import java.io.IOException;
import java.util.UUID;
import java.util.concurrent.BlockingQueue;

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

public class RemoveController implements Runnable {
  private BlockingQueue<Command> removeQueue;
  private BlockingQueue<Response> responseQueue;
  private Storage storage;
  private Command command;
  private Response response;
  final static Logger logger = Logger.getLogger(StorageController.class);

  public RemoveController(BlockingQueue<Command> removeQueue,
      BlockingQueue<Response> responseQueue, Storage storage) {
    super();
    this.removeQueue = removeQueue;
    this.storage = storage;
    this.responseQueue = responseQueue;
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    String error_message = "Error al eliminar el mensaje";
    logger.info("Iniciando el remove controller");
    try {
      while (¬Thread.interrupted()) {
        try {
          command = removeQueue.take();
          response = new Response();
          response.setUuid(UUID.randomUUID());
          response.setUser(command.getUser());
          storage.delete(command);
          response.setMessage("Borrado exitoso");
          response.setResponse_status(RESPONSE_STATUS.OK);
        } 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);
          logger.error(e);
        } finally {
          if (response ≠ null) {
            responseQueue.put(response);
            response = null;
          }
        }
      }
    } catch (InterruptedException e) {
      logger.info("Remove Controller interrumpido");
    }
  }
}
```

sep 14, 17 6:59 **QueryController.java** Page 1/1

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

import java.io.IOException;
import java.util.UUID;
import java.util.concurrent.BlockingQueue;

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

public class QueryController implements Runnable {
  private BlockingQueue<Command> queryQueue;
  private BlockingQueue<Response> responseQueue;
  private Storage storage;
  private Command command;
  private Response response;
  final static Logger logger = Logger.getLogger(QueryController.class);

  public QueryController(BlockingQueue<Command> queryQueue,
      BlockingQueue<Response> responseQueue, Storage storage) {
    super();
    this.queryQueue = queryQueue;
    this.responseQueue = responseQueue;
    this.storage = storage;
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    String error_message = "Error al consultar";
    logger.info("Iniciando el query controller");
    try {
      while (¬Thread.interrupted()) {
        try {
          command = queryQueue.take();
          response = new Response();
          response.setUuid(UUID.randomUUID());
          response.setUser(command.getUser());
          response.setMessage(storage.query(command));
          logger.debug(response.getMessage());
          response.setResponse_status(RESPONSE_STATUS.OK);
        } 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);
          logger.error(e);
          e.printStackTrace();
        } finally {
          if (response ≠ null) {
            responseQueue.put(response);
            response = null;
          }
        }
      }
    } catch (InterruptedException e) {
      logger.info("Query Controller interrumpido");
    }
  }
}
```

sep 14, 17 6:59 **MainStorage.java** Page 1/1

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

import java.io.IOException;
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);
    }
    final ReadingRemoteQueue storageQueue = new ReadingRemoteQueue(
        configLoader.getProperties().get(Constants.STORAGE_QUEUE_NAME),
        configLoader.getProperties().get(Constants.STORAGE_QUEUE_HOST),
        configLoader.getProperties());
    final Thread storageControllerThread = new Thread(new StorageController(
        configLoader.getProperties(), storageQueue));

    Runtime.getRuntime().addShutdownHook(new Thread() {
      @Override
      public void run() {
        storageQueue.shutDown();
        storageControllerThread.interrupt();
        try {
          storageControllerThread
              .join(Constants.STORAGE_THREAD_WAIT_TIME);
        } catch (InterruptedException e) {
          // Do nothing
        }
      }
    });
    storageControllerThread.start();
  }
}
```

sep 14, 17 6:59 **CreateController.java** Page 1/1

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

import java.io.IOException;
import java.util.UUID;
import java.util.concurrent.BlockingQueue;

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

public class CreateController implements Runnable {
  private BlockingQueue<Command> createQueue;
  private BlockingQueue<Response> responseQueue;
  private Command command;
  private Storage storage;
  private Response response;
  final static Logger logger = Logger.getLogger(CreateController.class);

  public CreateController(BlockingQueue<Command> createQueue,
      BlockingQueue<Response> responseQueue, int shardingFactor,
      Storage storage) {
    super();
    this.createQueue = createQueue;
    this.responseQueue = responseQueue;
    this.storage = storage;
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    logger.info("Iniciando el create controller");

    try {
      while (¬Thread.interrupted()) {
        String error_message = "Error al crear el mensaje";
        try {
          command = createQueue.take();
          response = new Response();
          response.setUuid(UUID.randomUUID());
          response.setUser(command.getUser());
          storage.saveMessage(command);
          response.setMessage("Creacion exitosa");
          response.setResponse_status(RESPONSE_STATUS.OK);
        } 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);
          logger.error(e);
        } finally {
          if (response ≠ null) {
            responseQueue.put(response);
            response = null;
          }
        }
      }
    } catch (InterruptedException e) {
      logger.info("Create controller interrumpido");
    }
  }
}
```

sep 14, 17 6:59 **StorageController.java** Page 1/1

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

import java.io.IOException;
import java.util.Map;
import java.util.concurrent.BlockingQueue;

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.WritingRemoteQueue;

public class StorageController implements Runnable {
  private BlockingQueue<Command> storageCommandQueue;
  private WritingRemoteQueue storageQueue;

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

  public StorageController(BlockingQueue<Command> storageCommandQueue,
      Map<String, String> config) {
    this.storageCommandQueue = storageCommandQueue;
    this.storageQueue = new WritingRemoteQueue(
        config.get(Constants.STORAGE_QUEUE_NAME),
        config.get(Constants.STORAGE_QUEUE_HOST), config);
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    Command command;

    logger.info("Iniciando el storage controller");
    try {
      while (¬Thread.interrupted()) {
        command = new Command();
        try {
          command = storageCommandQueue.take();
          logger.info(
              "Comando recibido con los siguientes parametros: "
                  + "\nUsuario: " + command.getUser()
                  + "\nComando: " + command.getCommand()
                  + "\nMensaje: " + command.getMessage());
          storageQueue.push(command);
          logger.info("Comando enviado al storage");
        } catch (IOException e) {
          logger.error(e);
        }
      }
    } catch (InterruptedException e) {
      logger.info("Storage controller interrumpido");
    }
  }
}
```

sep 14, 17 6:59 **MainDispatcher.java** Page 1/1

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

import java.io.IOException;

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);
    }

    final ReadingRemoteQueue dispatcherQueue = new ReadingRemoteQueue(
        configLoader.getProperties()
            .get(Constants.DISPATCHER_QUEUE_NAME),
        configLoader.getProperties()
            .get(Constants.DISPATCHER_QUEUE_HOST),
        configLoader.getProperties());

    final Thread dispatcherThread = new Thread(new DispatcherController(
        configLoader.getProperties(), dispatcherQueue));
    Runtime.getRuntime().addShutdownHook(new Thread() {
      @Override
      public void run() {
        dispatcherQueue.shutDown();
        dispatcherThread.interrupt();
        try {
          dispatcherThread.join(Constants.STORAGE_THREAD_WAIT_TIME);
        } catch (InterruptedException e) {
          // Do nothing
        }
      }
    });

    dispatcherThread.start();
  }
}
```

sep 14, 17 6:59 **LoggerController.java** Page 1/1

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

import java.io.IOException;
import java.util.Map;
import java.util.concurrent.BlockingQueue;

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.WritingRemoteQueue;

public class LoggerController implements Runnable {
  private BlockingQueue<Command> loggerCommandQueue;
  private WritingRemoteQueue loggerQueue;
  final static Logger logger = Logger.getLogger(LoggerController.class);

  public LoggerController(BlockingQueue<Command> loggerCommandQueue,
      Map<String, String> config) {
    this.loggerCommandQueue = loggerCommandQueue;
    loggerQueue = new WritingRemoteQueue(
        config.get(Constants.AUDIT_LOGGER_QUEUE_NAME),
        config.get(Constants.AUDIT_LOGGER_QUEUE_HOST), config);
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    Command command;

    logger.info("Iniciando el logger controller");
    try {
      while (¬Thread.interrupted()) {
        try {
          command = loggerCommandQueue.take();
          logger.info(
              "Comando recibido con los siguientes parametros: "
                  + "\nUsuario: " + command.getUser()
                  + "\nComando: " + command.getCommand()
                  + "\nMensaje: " + command.getMessage());
          loggerQueue.push(command);
          logger.info("Comando enviado al logger");
        } catch (IOException e) {
          logger.error(e);
        }
      }
    } catch (InterruptedException e) {
      logger.info("Logger controller interrumpido");
    }
    logger.info("Logger controller terminado");
  }
}
```

sep 14, 17 6:59 **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.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

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.ReadingRemoteQueueException;

public class DispatcherController implements Runnable {

  private ReadingRemoteQueue dispatcherQueue;
  private BlockingQueue<Command> storageCommandQueue;
  private BlockingQueue<Command> analyzerCommandQueue;
  private BlockingQueue<Command> loggerCommandQueue;
  private Thread analyzerControllerThread;
  private Thread storageControllerThread;
  private Thread loggerControllerThread;
  final static Logger logger = Logger.getLogger(DispatcherController.class);

  public DispatcherController(Map<String, String> config,
      ReadingRemoteQueue dispatcherQueue) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    analyzerCommandQueue = new ArrayBlockingQueue<Command>(
        Constants.COMMAND_QUEUE_SIZE);
    storageCommandQueue = new ArrayBlockingQueue<Command>(
        Constants.COMMAND_QUEUE_SIZE);
    loggerCommandQueue = new ArrayBlockingQueue<Command>(
        Constants.COMMAND_QUEUE_SIZE);
    analyzerControllerThread = new Thread(
        new AnalyzerController(analyzerCommandQueue, config));
    storageControllerThread = new Thread(
        new StorageController(storageCommandQueue, config));
    loggerControllerThread = new Thread(
        new LoggerController(loggerCommandQueue, config));
    this.dispatcherQueue = dispatcherQueue;
  }

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

    analyzerControllerThread.start();
    storageControllerThread.start();
    loggerControllerThread.start();

    logger.info("Iniciando el dispatcher controller");
    try {
      while (¬Thread.interrupted()) {
        messageList = dispatcherQueue.pop();
        Iterator<byte[]> it = messageList.iterator();
        while (it.hasNext()) {
          // for (byte[] message : messageList) {
          try {
            command = new Command();
            command.deserialize(it.next());
            logger.info(
                "Comando recibido con los siguientes parametros: "
```

sep 14, 17 6:59 **DispatcherController.java** Page 2/2

```
                        + "\nUsuario: " + command.getUser()
                        + "\nComando: " + command.getCommand()
                        + "\nMensaje: " + command.getMessage());
                switch (command.getCommand()) {
                case PUBLISH:
                  storageCommandQueue.put(command);
                  analyzerCommandQueue.put(command);
                  loggerCommandQueue.put(command);
                  logger.info("Comando enviado al publish: "
                      + "\nUsuario: " + command.getUser()
                      + "\nComando: " + command.getCommand()
                      + "\nMensaje: " + command.getMessage());
                  break;
                case QUERY:
                  storageCommandQueue.put(command);
                  loggerCommandQueue.put(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());
                  storageCommandQueue.put(command);
                  loggerCommandQueue.put(command);
                  break;
                case FOLLOW:
                  logger.info("Comando enviado al follow: "
                      + "\nUsuario: " + command.getUser()
                      + "\nComando: " + command.getCommand()
                      + "\nMensaje: " + command.getMessage());
                  analyzerCommandQueue.put(command);
                  loggerCommandQueue.put(command);
                  break;
                default:
                  logger.error("Comando invalido");
                  break;
                }
              } catch (ClassNotFoundException | IOException e) {
                logger.error("No se ha podido deserializar el mensaje");
                logger.debug(e);
                e.printStackTrace();
              }
            }
          }
        } catch (ReadingRemoteQueueException | InterruptedException e) {
          analyzerControllerThread.interrupt();
          storageControllerThread.interrupt();
          loggerControllerThread.interrupt();
          try {
            analyzerControllerThread.join();
            storageControllerThread.join();
            loggerControllerThread.join();
          } catch (InterruptedException e1) {
            // Do nothing
          }
        }
        logger.info("Dispatcher controller terminado");
      }
    }
```

sep 14, 17 6:58 **AnalyzerController.java** Page 1/1

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

import java.io.IOException;
import java.util.Map;
import java.util.concurrent.BlockingQueue;

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.WritingRemoteQueue;

public class AnalyzerController implements Runnable {
  private BlockingQueue<Command> analyzerCommandQueue;
  private WritingRemoteQueue analyzerQueue;
  final static Logger logger = Logger.getLogger(AnalyzerController.class);

  public AnalyzerController(BlockingQueue<Command> analyzerCommandQueue,
      Map<String, String> config) {
    this.analyzerCommandQueue = analyzerCommandQueue;
    this.analyzerQueue = new WritingRemoteQueue(
        config.get(Constants.ANALYZER_QUEUE_NAME),
        config.get(Constants.ANALYZER_QUEUE_HOST), config);
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    Command command;

    logger.info("Iniciando el analyzer controller");
    try {
      while (¬Thread.interrupted()) {
        try {
          command = analyzerCommandQueue.take();
          logger.info(
              "Comando recibido con los siguientes parametros: "
                  + "\nUsuario: " + command.getUser()
                  + "\nComando: " + command.getCommand()
                  + "\nMensaje: " + command.getMessage());
          analyzerQueue.push(command);
          logger.info("Comando enviado al analyzer");
        } catch (IOException e) {
          logger.error(e);
        }
      }
    } catch (InterruptedException e) {
      logger.info("Analyzer controller interrumpido");
    }
    logger.info("Analyzer controller terminado");
  }
}
```

sep 01, 17 21:18 **AppTest.java** Page 1/1

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

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Unit test for simple App.
 */
public class AppTest
    extends TestCase
{
    /**
     * Create the test case
     *
     * @param testName name of the test case
     */
    public AppTest( String testName )
    {
        super( testName );
    }

    /**
     * @return the suite of tests being tested
     */
    public static Test suite()
    {
        return new TestSuite( AppTest.class );
    }

    /**
     * Rigourous Test :-)
     */
    public void testApp()
    {
        assertTrue( true );
    }
}
```

sep 01, 17 21:18 **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!" );
    }
}
```

sep 14, 17 6:57 **WritingRemoteQueue.java** Page 1/1

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

import java.io.IOException;
import java.util.Map;
import java.util.Properties;
import java.util.UUID;
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.ProducerConfig;
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 queueHost,
      Map<String, String> params) {
    Properties props = new Properties();
    this.queueName = queueName;
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, queueHost);
    props.put(ProducerConfig.ACKS_CONFIG,
        params.get(Constants.ACKS_CONFIG));
    props.put(ProducerConfig.RETRIES_CONFIG,
        Integer.parseInt(params.get(Constants.RETRIES_CONFIG)));
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
        params.get(Constants.VALUE_SERIALIZER_CLASS_CONFIG));
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
        params.get(Constants.KEY_SERIALIZER_CLASS_CONFIG));
    producer = new KafkaProducer<byte[], byte[]>(props);
  }

  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);
  }

}
```

sep 01, 17 21:18 **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 = null;
    this.response_status = null;
    this.message = null;
  }

  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;
```

sep 01, 17 21:18 **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 02, 17 20:30 **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  }
```

sep 14, 17 6:57 **ReadingRemoteQueue.java** Page 1/1

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

import java.io.IOException;
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 ReadingRemoteQueue(String queueName, String queueHost,
      Map<String, String> params) {
    Properties consumerConfig = new Properties();
    consumerConfig.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, queueHost);
    consumerConfig.put(ConsumerConfig.GROUP_ID_CONFIG,
        params.get(Constants.GROUP_ID_CONFIG));
    consumerConfig.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
        params.get(Constants.AUTO_OFFSET_RESET_CONFIG));
    consumerConfig.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
        params.get(Constants.KEY_DESERIALIZER_CLASS_CONFIG));
    consumerConfig.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
        params.get(Constants.VALUE_DESERIALIZER_CLASS_CONFIG));
    consumer = new KafkaConsumer<byte[], byte[]>(consumerConfig);
    consumer.subscribe(Collections.singletonList(queueName));
  }

  @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;
  }

}
```

sep 03, 17 9:06 **ReadingRemoteQueueException.java** Page 1/1

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

import org.apache.kafka.common.errors.WakeupException;

public class ReadingRemoteQueueException extends WakeupException {

}
```

sep 01, 17 21:18 **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;

}
```

sep 14, 17 6:57 **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";

  // 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 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";
  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";
```

sep 14, 17 6:57 **Constants.java** Page 2/2

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

sep 14, 17 6:57 **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 }
```

sep 01, 17 21:18 **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 = null;
    this.user = null;
    this.message = null;
    this.uuid = null;
    this.timestamp = null;
  }

  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();
  }

```

sep 01, 17 21:18 **Command.java** Page 2/2

```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) {

  }
}
```

sep 14, 17 6:57 **ResponseController.java** Page 1/1

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

import java.io.IOException;
import java.util.List;
import java.util.concurrent.BlockingQueue;
import org.apache.log4j.Logger;
import org.apache.log4j.MDC;

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

public class ResponseController implements Runnable {

  private BlockingQueue<Response> responseQueue;
  private ReadingRemoteQueue remoteResponseQueue;
  final static Logger logger = Logger.getLogger(ResponseController.class);

  public ResponseController(BlockingQueue<Response> responseQueue,
      ReadingRemoteQueue remoteResponseQueue) {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    this.responseQueue = responseQueue;
    this.remoteResponseQueue = remoteResponseQueue;
  }

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

    logger.debug("Iniciando el response controller");
    try {
      while (¬Thread.interrupted()) {
        messageList = remoteResponseQueue.pop();
        for (byte[] message : messageList) {
          try {
            response.deserialize(message);
            responseQueue.put(response);
          } catch (IOException | ClassNotFoundException e) {
            logger.error(
                "No se ha podido obtener el mensaje de la cola del usuario");
            logger.debug(e);
          }
        }
      }
    } catch (InterruptedException e) {
      // Do nothing
    }
    logger.debug("Iniciando el response controller");
  }
}
```

sep 14, 17 6:57 **MainClientConsole.java** Page 1/2

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

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

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;

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

  public static void main(String[] args) {
    PropertyConfigurator.configure(Constants.LOGGER_CONF);
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    Set<Callable<String>> usersSet = new HashSet<Callable<String>>();
    int usersAmount = 0;
    ConfigLoader configLoader = null;

    if (args.length ≡ 0) {
      displayHelp();
    }

    final String mode = args[0];

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

    try {
      usersAmount = Integer.parseInt(args[1]);
    } catch (NumberFormatException e) {
      // Do nothing
    }

    final Thread userThread = createUser(mode, configLoader.getProperties(),
        args[1], args[2]);
    final ExecutorService executor = createUsers(mode, usersAmount);

    if (mode.equals(Constants.INTERACTIVE_MODE)) {
      System.out.printf(
          "Iniciando el Client console en modo interactivo para el usuario %s",
          args[1]);
      userThread.start();
    } else if (mode.equals(Constants.BATCH_MODE)) {
      System.out.printf("Iniciando el Client console en modo batch");
      try {
        for (int i = 0; i < Integer.parseInt(args[1]); i++) {
          usersSet.add(new BatchUser(configLoader.getProperties(),
              "user" + i, "localhost:9092"));
        }
        executor.invokeAll(usersSet);
      } catch (InterruptedException e) {
```

sep 14, 17 6:57 **MainClientConsole.java** Page 2/2

```
67          // Do nothing
68        }
69      } else {
70        displayHelp();
71      }
72
73      Runtime.getRuntime().addShutdownHook(new Thread() {
74        @Override
75        public void run() {
76          if (mode.equals(Constants.INTERACTIVE_MODE)) {
77            userThread.interrupt();
78            try {
79              userThread.join(Constants.USER_THREAD_WAIT_TIME);
80            } catch (InterruptedException e) {
81              // Do nothing
82            }
83          } else {
84            executor.shutdownNow();
85            try {
86              executor.awaitTermination(
87                  Constants.USER_THREAD_WAIT_TIME,
88                  TimeUnit.MILLISECONDS);
89            } catch (InterruptedException e) {
90              // Do nothing
91            }
92          }
93        }
94      });
95    }
96
97    private static Thread createUser(String mode, Map<String, String> config,
98        String userName, String hostName) {
99      if (mode.equals(Constants.INTERACTIVE_MODE)) {
100       if ((userName ≡ null ∨ ("").equals(userName))
101           ∧ (hostName ≡ null ∨ ("").equals(hostName))) {
102         displayHelp();
103       }
104       ;
105       System.out.printf(
106           "Iniciando el Client console en modo interactivo para el usuario %s",
107           userName);
108       return new Thread(new InteractiveUser(config, userName, hostName));
109     } else {
110       return null;
111     }
112   }
113
114   private static ExecutorService createUsers(String mode, int userAmount) {
115     if (mode.equals(Constants.BATCH_MODE)) {
116       ExecutorService executor = Executors.newFixedThreadPool(userAmount);
117       return executor;
118     } else {
119       return null;
120     }
121   }
122
123   private static void displayHelp() {
124     System.out.printf(
125         "Client console%n**********%nSintaxis:%n./ClientConsole <params>%nParametros:%ni [username] [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");
126     System.exit(Constants.EXIT_FAILURE);
127   }
128
129 }
```

sep 14, 17 6:56 **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.ArrayBlockingQueue;
8  import java.util.concurrent.BlockingQueue;
9
10 import ar.fiuba.taller.common.Command;
11 import ar.fiuba.taller.common.Constants;
12 import ar.fiuba.taller.common.ReadingRemoteQueue;
13 import ar.fiuba.taller.common.Response;
14 import ar.fiuba.taller.common.WritingRemoteQueue;
15
16 public class InteractiveUser implements Runnable {
17   String userName;
18   private BlockingQueue<Command> commandQueue;
19   private BlockingQueue<Response> responseQueue;
20   private Thread commandControllerThread;
21   private Thread eventViewerThread;
22   private Thread responseControllerThread;
23   private ReadingRemoteQueue remoteUserResponseQueue;
24   private WritingRemoteQueue dispatcherQueue;
25
26   public InteractiveUser(Map<String, String> config, String userName,
27       String userHost) {
28     this.userName = userName;
29     commandQueue = new ArrayBlockingQueue<Command>(
30         Constants.COMMAND_QUEUE_SIZE);
31     dispatcherQueue = new WritingRemoteQueue(
32         config.get(Constants.DISPATCHER_QUEUE_NAME),
33         config.get(Constants.DISPATCHER_QUEUE_HOST), config);
34     commandControllerThread = new Thread(
35         new CommandController(commandQueue, dispatcherQueue,
36             Integer.parseInt(config.get(Constants.MAX_LENGTH_MSG)),
37             Constants.LOGS_DIR + "/" + userName
38                 + Constants.COMMANDS_FILE_EXTENSION));
39     responseQueue = new ArrayBlockingQueue<Response>(
40         Constants.RESPONSE_QUEUE_SIZE);
41     remoteUserResponseQueue = new ReadingRemoteQueue(userName, userHost,
42         config);
43     responseControllerThread = new Thread(
44         new ResponseController(responseQueue, remoteUserResponseQueue));
45     eventViewerThread = new Thread(new EventWriter(responseQueue, userName,
46         Constants.LOGS_DIR + "/" + userName
47             + Constants.EVENT_VIEWER_FILE_EXTENSION));
48   }
49
50   public void run() {
51     BufferedReader br = null;
52     String[] msgParts;
53
54     commandControllerThread.start();
55     eventViewerThread.start();
56     responseControllerThread.start();
57
58     try {
59       br = new BufferedReader(new InputStreamReader(System.in));
60       while (¬Thread.interrupted()) {
61         try {
62           System.out.print("Enter command: ");
63           String input = br.readLine();
64           msgParts = input.split(":");
65           commandQueue.put(new Command(msgParts[0], userName,
66               msgParts[1], null, null));
```

sep 14, 17 6:56 **InteractiveUser.java** Page 2/2

```
67            } catch (IOException e) {
68              System.out.println(
69                   "Error: No se ha podido procesar el comando");
70            }
71          }
72        } catch (InterruptedException e) {
73          remoteUserResponseQueue.shutDown();
74          commandControllerThread.interrupt();
75          eventViewerThread.interrupt();
76          responseControllerThread.interrupt();
77          try {
78            commandControllerThread.join(Constants.USER_THREAD_WAIT_TIME);
79            eventViewerThread.join(Constants.USER_THREAD_WAIT_TIME);
80            responseControllerThread.join(Constants.USER_THREAD_WAIT_TIME);
81          } catch (InterruptedException e1) {
82            // Do nothing
83          }
84        }
85      }
86    }
```

sep 09, 17 18:55 **EventWriter.java** Page 1/2

```
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.concurrent.BlockingQueue;
8
9   import org.apache.log4j.Logger;
10  import org.apache.log4j.MDC;
11
12  import ar.fiuba.taller.common.Response;
13
14  public class EventWriter implements Runnable {
15    BlockingQueue<Response> responseQueue;
16    String username;
17    String eventFile;
18    final static Logger logger = Logger.getLogger(EventWriter.class);
19
20    public EventWriter(BlockingQueue<Response> responseQueue, String username,
21        String eventFile) {
22      MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
23      this.responseQueue = responseQueue;
24      this.username = username;
25      this.eventFile = eventFile;
26    }
27
28    public void run() {
29      Response response = null;
30      FileWriter responseFile = null;
31      PrintWriter pw;
32
33      logger.debug("Iniciando el event viewer");
34      try {
35        while (¬Thread.interrupted()) {
36          logger.debug("Esperando respuesta");
37          response = responseQueue.take();
38          try {
39            pw = new PrintWriter(new BufferedWriter(
40                new FileWriter(eventFile, true)));
41            logger.debug("Respuesta obtenida");
42            pw.printf(
43                "Evento recibido − UUID: {%s} − Status: {%s} − Mensaje: {%s}%n−−−−−−−−−−−−−−−−−−−−
    −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−%n",
44                response.getUuid(), response.getResponse_status(),
45                response.getMessage());
46            pw.close();
47          } catch (IOException e) {
48            logger.error("No se ha podido escribir la respuesta");
49            logger.debug(e);
50          }
51        }
52      } catch (InterruptedException e) {
53        // Do nothing
54      } finally {
55        try {
56          if (null ≠ responseFile)
57            responseFile.close();
58        } catch (Exception e2) {
59          logger.error("Error al cerrar el archivo " + eventFile);
60          logger.debug(e2);
61        }
62      }
63
64    }
65
```

sep 09, 17 18:55 **EventWriter.java** Page 2/2

```
66 }
```

sep 14, 17 6:56 **CommandController.java** Page 1/2

```
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.sql.Timestamp;
8  import java.util.UUID;
9  import java.util.concurrent.BlockingQueue;
10
11 import org.apache.log4j.Logger;
12 import org.apache.log4j.MDC;
13
14 import ar.fiuba.taller.common.Command;
15 import ar.fiuba.taller.common.Constants;
16 import ar.fiuba.taller.common.WritingRemoteQueue;
17
18 public class CommandController implements Runnable {
19   private BlockingQueue<Command> commandQueue;
20   private WritingRemoteQueue dispatcherQueue;
21   private int maxlengthMsg;
22   private Timestamp timestamp;
23   private String commandFile;
24
25   final static Logger logger = Logger.getLogger(CommandController.class);
26
27   public CommandController(BlockingQueue<Command> commandQueue,
28       WritingRemoteQueue dispatcherQueue, int maxlengthMsg,
29       String commandFile) {
30     this.commandQueue = commandQueue;
31     this.dispatcherQueue = dispatcherQueue;
32     this.maxlengthMsg = maxlengthMsg;
33     this.commandFile = commandFile;
34   }
35
36   public void run() {
37     MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
38     Command command;
39     FileWriter responseFile = null;
40     PrintWriter pw;
41
42     logger.debug("Iniciando el command controller");
43     try {
44       while (¬Thread.interrupted()) {
45         try {
46           logger.debug("Obteniendo comando de la cola");
47           command = commandQueue.take();
48           logger.debug("Comando obtenido");
49           logger.debug("Comando recibido: " + command.getCommand());
50           logger.debug("Mensaje: " + command.getMessage());
51           if (command.getMessage().length() ≤ maxlengthMsg) {
52             logger.debug("Generando UUID");
53             command.setUuid(UUID.randomUUID());
54             logger.debug("Generando timestamp");
55             timestamp = new Timestamp(System.currentTimeMillis());
56             command.setTimestamp(Constants.SDF.format(timestamp));
57             logger.debug("UUID generado: " + command.getUuid());
58             logger.debug("Enviando el mensaje al dispatcher");
59             dispatcherQueue.push(command);
60             logger.debug("Mensaje enviado");
61             pw = new PrintWriter(new BufferedWriter(
62                 new FileWriter(commandFile, true)));
63             logger.debug("Respuesta obtenida");
64             pw.printf(
65                 "Evento enviado − UUID: {%s} − Timestamp: {%s} − Comando: {%s} − Mensaje: {%s}%n−
   ————————————————————————————————————————%n",
```

```
66                      command.getUuid(), command.getTimestamp(),
67                      command.getCommand(), command.getMessage());
68                  pw.close();
69                  System.out.printf(
70                      "Comando enviado − UUID: {%s} − Comando: {%s} − Usuario: {%s} − Mensaje: {%s} − Ti
    mestamp: {%s}",
71                      command.getUuid().toString(),
72                      command.getCommand().toString(),
73                      command.getUser(), command.getMessage(),
74                      command.getTimestamp());
75              } else {
76                  logger.error(
77                      "El mensaje contiene mas de 141 caracteres");
78              }
79            } catch (IOException e) {
80              logger.error("Error al enviar el mensaje al dispatcher");
81              logger.debug(e);
82            }
83          }
84        } catch (InterruptedException e) {
85          logger.error("Error al sacar un comando de la cola commandQueue");
86          logger.debug(e);
87        }
88        logger.debug("Command controller terminado");
89      }
90    }
```



```
1   package ar.fiuba.taller.ClientConsole;
2
3   import java.io.FileReader;
4   import java.io.IOException;
5   import java.util.ArrayList;
6   import java.util.Iterator;
7   import java.util.List;
8   import java.util.Map;
9   import java.util.concurrent.ArrayBlockingQueue;
10  import java.util.concurrent.BlockingQueue;
11  import java.util.concurrent.Callable;
12
13  import org.apache.log4j.Logger;
14  import org.apache.log4j.MDC;
15  import org.json.simple.JSONArray;
16  import org.json.simple.JSONObject;
17  import org.json.simple.parser.JSONParser;
18  import org.json.simple.parser.ParseException;
19
20  import ar.fiuba.taller.common.Command;
21  import ar.fiuba.taller.common.Constants;
22  import ar.fiuba.taller.common.ReadingRemoteQueue;
23  import ar.fiuba.taller.common.Response;
24  import ar.fiuba.taller.common.WritingRemoteQueue;
25
26  public class BatchUser implements Callable {
27    private String userName;
28    private int commandAmount;
29    private BlockingQueue<Command> commandQueue;
30    private BlockingQueue<Response> responseQueue;
31    private Thread commandControllerThread;
32    private Thread eventViewerThread;
33    private Thread responseControllerThread;
34    private ReadingRemoteQueue remoteUserResponseQueue;
35    private WritingRemoteQueue dispatcherQueue;
36    private long delayTime;
37    final static Logger logger = Logger.getLogger(BatchUser.class);
38
39    public BatchUser(Map<String, String> config, String userName,
40        String userHost) {
41      MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
42      this.userName = userName;
43      commandAmount = Integer.parseInt(config.get(Constants.COMMAND_AMOUNT));
44      commandQueue = new ArrayBlockingQueue<Command>(
45          Constants.COMMAND_QUEUE_SIZE);
46      dispatcherQueue = new WritingRemoteQueue(
47          config.get(Constants.DISPATCHER_QUEUE_NAME),
48          config.get(Constants.DISPATCHER_QUEUE_HOST), config);
49      commandControllerThread = new Thread(
50          new CommandController(commandQueue, dispatcherQueue,
51              Integer.parseInt(config.get(Constants.MAX_LENGTH_MSG)),
52              Constants.LOGS_DIR + "/" + userName
53                  + Constants.COMMANDS_FILE_EXTENSION));
54      responseQueue = new ArrayBlockingQueue<Response>(
55          Constants.RESPONSE_QUEUE_SIZE);
56      remoteUserResponseQueue = new ReadingRemoteQueue(userName, userHost,
57          config);
58      responseControllerThread = new Thread(
59          new ResponseController(responseQueue, remoteUserResponseQueue));
60      eventViewerThread = new Thread(new EventWriter(responseQueue, userName,
61          Constants.LOGS_DIR + "/" + userName
62              + Constants.EVENT_VIEWER_FILE_EXTENSION));
63      delayTime = Long.parseLong(config.get(Constants.BATCH_DELAY_TIME));
64    }
65
66    @Override
```

sep 14, 17 6:56 **BatchUser.java** Page 2/2

```
67    public Object call() throws Exception {
68      logger.debug("Iniciando el script reader");
69      int count = 0;
70
71      commandControllerThread.start();
72      eventViewerThread.start();
73      responseControllerThread.start();
74      try {
75        JSONParser parser = new JSONParser();
76        Object obj = parser.parse(new FileReader(Constants.COMMAND_SCRIPT));
77        JSONObject jsonObject = (JSONObject) obj;
78        JSONArray commandArray = (JSONArray) jsonObject
79            .get(Constants.COMMAND_ARRAY);
80        JSONObject commandObject;
81        Command command;
82        List<Integer> commandIndexList = getCommandIndexList(commandAmount,
83            commandArray.size());
84        Iterator<Integer> iterator = commandIndexList.iterator();
85
86        while (iterator.hasNext()) {
87          commandObject = (JSONObject) commandArray.get(iterator.next());
88          command = new Command(
89              (String) commandObject.get(Constants.COMMAND_KEY),
90              userName,
91              (String) commandObject.get(Constants.MESSAGE_KEY), null,
92              null);
93          logger.debug("COMANDO: " + count
94              + ".Se inserto comando con los siguientes parametros: "
95              + "\nUsuario: " + command.getUser() + "\nComando: "
96              + command.getCommand() + "\nMensaje: "
97              + command.getMessage());
98          commandQueue.put(command);
99          ++count;
100       }
101     } catch (InterruptedException e) {
102       logger.error("Thread interrumpido");
103       logger.debug(e);
104     } catch (ParseException | IOException e) {
105       logger.error("Error al tratar el script de comandos");
106       logger.debug(e);
107     }
108     return null;
109   }
110
111   private List<Integer> getCommandIndexList(int commandListIndexSize,
112       int maxCommandsAvailable) {
113     List<Integer> commandIndexList = new ArrayList<Integer>();
114
115     for (int i = 0; i < commandListIndexSize; i++) {
116       commandIndexList.add((int) (Math.random() * maxCommandsAvailable));
117     }
118
119     return commandIndexList;
120   }
121
122 }
```

sep 14, 17 6:58 **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.AUDIT_LOGGER_QUEUE_HOST),
32          configLoader.getProperties());
33
34      final Thread auditLoggerThread = new Thread(
35          new AuditLogger(loggerQueue, configLoader.getProperties()));
36
37      Runtime.getRuntime().addShutdownHook(new Thread() {
38        @Override
39        public void run() {
40          loggerQueue.shutDown();
41          auditLoggerThread.interrupt();
42          try {
43            auditLoggerThread
44                .join(Constants.AUDIT_LOGGER_THREAD_WAIT_TIME);
45          } catch (InterruptedException e) {
46            // Do nothing
47          }
48        }
49      });
50
51      auditLoggerThread.start();
52    }
53  }
```

sep 14, 17 6:58 **AuditLogger.java** Page 1/2

```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 implements Runnable {
  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(e);
    } catch (ReadingRemoteQueueException e) {
      pw.close();
    }
    logger.info("Audit logger terminado");
  }

```

sep 14, 17 6:58 **AuditLogger.java** Page 2/2

```java
  private String getAuditLogEntry(Command command) {
    timestamp = new Timestamp(System.currentTimeMillis());
    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 synchronized 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");
      logger.info(e.toString());
      e.printStackTrace();
    } finally {
      file.flush();
      file.close();
    }
  }

  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+)";
    Pattern p = Pattern.compile(regexPattern);
    Matcher m = p.matcher(followed);
```

sep 14, 17 6:58 **UserRegistry.java** Page 2/3

sep 14, 17 6:58 **UserRegistry.java** Page 3/3

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

sep 14, 17 6:58 **AnalyzerReciver.java** Page 1/2

```
1   package ar.fiuba.taller.analyzer;
2
3   import java.io.IOException;
4   import java.util.List;
5   import java.util.Map;
6   import java.util.concurrent.ArrayBlockingQueue;
7   import java.util.concurrent.BlockingQueue;
8
9   import org.apache.log4j.Logger;
10  import org.apache.log4j.MDC;
11  import org.json.simple.parser.ParseException;
12
13  import ar.fiuba.taller.common.Command;
14  import ar.fiuba.taller.common.Constants;
15  import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
16  import ar.fiuba.taller.common.ReadingRemoteQueue;
17  import ar.fiuba.taller.common.ReadingRemoteQueueException;
18  import ar.fiuba.taller.common.Response;
19
20  public class AnalyzerReciver implements Runnable {
21
22    private Map<String, String> config;
23    private ReadingRemoteQueue analyzerQueue;
24    final static Logger logger = Logger.getLogger(AnalyzerReciver.class);
25
26    public AnalyzerReciver(Map<String, String> config,
27        ReadingRemoteQueue analyzerQueue) {
28      MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
29      this.analyzerQueue = analyzerQueue;
30      this.config = config;
31    }
32
33    public void run() {
34      Command command = new Command();
35      Response response = new Response();
36      List<byte[]> messageList = null;
37      BlockingQueue<Response> responseQueue = new ArrayBlockingQueue<Response>(
38          Constants.RESPONSE_QUEUE_SIZE);
39      UserRegistry userRegistry = new UserRegistry();
40      Thread analyzerDispatcherThread = new Thread(
41          new AnalyzerDispatcher(responseQueue, userRegistry, config));
42
43      logger.info("Iniciando el analyzer reciver");
44      analyzerDispatcherThread.start();
45
46      try {
47        while (¬Thread.interrupted()) {
48          messageList = analyzerQueue.pop();
49          for (byte[] message : messageList) {
50            try {
51              command.deserialize(message);
52              logger.info(
53                  "Comando recibido con los siguientes parametros: "
54                      + "\nUUID: " + command.getUuid()
55                      + "\nUsuario: " + command.getUser()
56                      + "\nComando: " + command.getCommand()
57                      + "\nMensaje: " + command.getMessage());
58              switch (command.getCommand()) {
59              case PUBLISH:
60                response = new Response();
61                response.setUuid(command.getUuid());
62                response.setUser(command.getUser());
63                response.setResponse_status(RESPONSE_STATUS.OK);
64                response.setMessage(command.getTimestamp() + "\n"
65                    + command.getUser() + "\n"
66                    + command.getMessage());
```

sep 14, 17 6:58 **AnalyzerReciver.java** Page 2/2

```
67                responseQueue.put(response);
68                break;
69              case FOLLOW:
70                userRegistry.update(command.getUser(),
71                    command.getMessage());
72                response = new Response();
73                response.setUuid(command.getUuid());
74                response.setUser(command.getUser());
75                response.setResponse_status(
76                    RESPONSE_STATUS.REGISTERED);
77                response.setMessage("Seguidor registrado");
78                responseQueue.put(response);
79                break;
80              default:
81                logger.info(
82                    "Comando recibido invalido. Comando descartado.");
83              }
84            } catch (IOException | ParseException
85                | ClassNotFoundException e) {
86              logger.error("Error al tratar el mensaje recibido.");
87            }
88          }
89        }
90      } catch (ReadingRemoteQueueException | InterruptedException e) {
91        analyzerDispatcherThread.interrupt();
92        try {
93          analyzerDispatcherThread.join();
94        } catch (InterruptedException e1) {
95          // Do nothing
96        }
97      }
98      logger.info("Analyzer reciver finalizado");
99    }
100
101 }
```

sep 14, 17 6:58 **AnalyzerMain.java** Page 1/1

```
1  package ar.fiuba.taller.analyzer;
2
3  import java.io.IOException;
4
5  import org.apache.log4j.Logger;
6  import org.apache.log4j.MDC;
7  import org.apache.log4j.PropertyConfigurator;
8
9  import ar.fiuba.taller.common.ConfigLoader;
10 import ar.fiuba.taller.common.Constants;
11 import ar.fiuba.taller.common.ReadingRemoteQueue;
12
13 public class AnalyzerMain {
14   final static Logger logger = Logger.getLogger(AnalyzerMain.class);
15
16   public static void main(String[] args) {
17     MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
18     PropertyConfigurator.configure(Constants.LOGGER_CONF);
19     ConfigLoader configLoader = null;
20
21     logger.info("Iniciando el analyzer");
22
23     try {
24       configLoader = new ConfigLoader(Constants.CONF_FILE);
25     } catch (IOException e) {
26       logger.error("Error al cargar la configuracion");
27       System.exit(Constants.EXIT_FAILURE);
28     }
29
30     final ReadingRemoteQueue analyzerQueue = new ReadingRemoteQueue(
31         configLoader.getProperties().get(Constants.ANALYZER_QUEUE_NAME),
32         configLoader.getProperties().get(Constants.ANALYZER_QUEUE_HOST),
33         configLoader.getProperties());
34
35     final Thread analyzerReciverThread = new Thread(new AnalyzerReciver(
36         configLoader.getProperties(), analyzerQueue));
37     Runtime.getRuntime().addShutdownHook(new Thread() {
38       @Override
39       public void run() {
40         analyzerQueue.shutDown();
41         analyzerReciverThread.interrupt();
42         try {
43           analyzerReciverThread
44               .join(Constants.STORAGE_THREAD_WAIT_TIME);
45         } catch (InterruptedException e) {
46           // Do nothing
47         } finally {
48           logger.info("Analyzer terminado");
49         }
50       }
51     });
52
53     analyzerReciverThread.start();
54   }
55 }
```

sep 14, 17 6:58 **AnalyzerDispatcher.java** Page 1/2

```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.BlockingQueue;
import java.util.concurrent.TimeoutException;

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

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

public class AnalyzerDispatcher implements Runnable {

  private BlockingQueue<Response> responseQueue;
  private Response response;
  private Map<String, WritingRemoteQueue> usersMap;
  private WritingRemoteQueue remoteQueue;
  private UserRegistry userRegistry;
  private List<String> userFollowers;
  private List<String> hashtagFollowers;
  private Set<String> usersSet;
  private Map<String, String> config;
  final static Logger logger = Logger.getLogger(AnalyzerDispatcher.class);

  public AnalyzerDispatcher(BlockingQueue<Response> responseQueue,
      UserRegistry userRegistry, Map<String, String> config) {
    this.responseQueue = responseQueue;
    this.userRegistry = userRegistry;
    usersMap = new HashMap<String, WritingRemoteQueue>();
    this.config = config;
  }

  public void run() {
    logger.info("Iniciando el Analyzer dispatcher");
    try {
      while (¬Thread.interrupted()) {
        try {
          response = responseQueue.take();
          logger.info("Nueva respuesta para enviar");
          logger.info("Nueva respuesta para enviar");
          logger.info("UUID: " + response.getUuid());
          logger.info("User: " + response.getUser());
          logger.info("Status: " + response.getResponse_status());
          logger.info("Message: " + response.getMessage());
          // 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");
```

sep 14, 17 6:58 **AnalyzerDispatcher.java** Page 2/2

```java
            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);
            }
            // Fowardeo el mensaje a los followers
            Iterator<String> it = usersSet.iterator();
            while (it.hasNext()) {
              (getUserQueue(it.next())).push(response);
            }
          }
        } catch (IOException | ParseException | TimeoutException e) {
          logger.error(
              "Error al insertar respuesta en la cola remota del "
                  + "usuario:" + response.getUser());
          logger.error(e);
        }
      }
    } catch (InterruptedException e) {
      logger.info("Analyzer dispatcher interrumpido");
    }
    logger.info("Analyzer dispatcher finalizado");
  }

  private WritingRemoteQueue getUserQueue(String username)
      throws IOException, TimeoutException {
    WritingRemoteQueue tmpQueue;
    logger.info("Ususario a fowardear: " + username);
    tmpQueue = usersMap.get(username);

    if (tmpQueue ≡ null) {
      tmpQueue = new WritingRemoteQueue(username, "localhost:9092",
          config);
      usersMap.put(username, tmpQueue);
    }
    return usersMap.get(username);
  }
}
```

sep 14, 17 7:06 **Table of Content** Page 1/1

Table of Contents