

abr 10, 17 5:28

AppTest.java

Page 1/1

```

1 package ar.fiuba.taller.storage;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8  * Unit test for simple App.
9  */
10 public class AppTest extends TestCase {
11     /**
12      * Create the test case
13      *
14      * @param testName
15      *      name of the test case
16      */
17     public AppTest(String testName) {
18         super(testName);
19     }
20
21     /**
22      * @return the suite of tests being tested
23      */
24     public static Test suite() {
25         return new TestSuite(AppTest.class);
26     }
27
28     /**
29      * Rigourous Test :-)
30      */
31     public void testApp() {
32         assertTrue(true);
33     }
34 }

```

abr 10, 17 5:29

Storage.java

Page 1/7

```

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

```

abr 10, 17 5:29

Storage.java

Page 2/7

```

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

```

abr 10, 17 5:29

Storage.java

Page 3/7

```

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 indice 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 indice 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());

```

abr 10, 17 5:29

Storage.java

Page 4/7

```

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);

```

abr 10, 17 5:29

Storage.java

Page 5/7

```

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     System.out.println(array.toJSONString());
301
302     BufferedReader br2;
303     String line, reg;
304     JSONObject jsonObject2;
305     int remainingPost = queryCountShowPosts;
306     // Abro archivo por archivo y recupero los mensajes
307     if (array != null) {
308         ListIterator<String> iterator = array.listIterator(array.size());
309         while (iterator.hasPrevious() ^ remainingPost > 0) {
310             id = iterator.previous();
311             System.out.println("id: " + id);
312             file = Constants.DB_DIR + "/" + id.substring(0, shardingFactor)
313                 + Constants.COMMAND_SCRIPT_EXTENSION;
314             System.out.println("file: " + file);
315             try (BufferedReader br = new BufferedReader(
316                 new FileReader(file))) {
317                 while ((line = br.readLine()) != null
318                     ^ remainingPost > 0) {
319                     System.out.println("line: " + line);
320                     obj2 = parser.parse(line);
321                     jsonObject2 = (JSONObject) obj2;
322                     messageList.add(jsonObject2.get(id).toString());
323                     remainingPost--;
324                 }
325             }
326         }
327     }
328     // Retorno la lista con los mensajes encontrados
329     return messageList;
330 }

```

abr 10, 17 5:29

Storage.java

Page 6/7

```

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

```

abr 10, 17 5:29

Storage.java

Page 7/7

```

397         String key = keyIt.next();
398         Long comp1 = map.get(key);
399         Long comp2 = val;
400
401         if (comp1.equals(comp2)) {
402             keyIt.remove();
403             sortedMap.put(key, val);
404             break;
405         }
406     }
407 }
408 Map<String, Long> map2 = sortedMap;
409 List<String> tt = new ArrayList<String>();
410 ArrayList<String> keys = new ArrayList<String>(sortedMap.keySet());
411 int i = keys.size() - 1;
412 int j = ttCountShowPosts;
413 while (i ≥ 0 ^ j > 0) {
414     tt.add(keys.get(i));
415     j--;
416     i--;
417 }
418 return tt;
419 }
420
421 }

```

abr 10, 17 5:28

StorageController.java

Page 1/3

```

1 package ar.fiuba.taller.storage;
2
3 import java.io.IOException;
4 import java.util.concurrent.ArrayBlockingQueue;
5 import java.util.concurrent.BlockingQueue;
6
7 import org.apache.log4j.Logger;
8 import org.apache.log4j.MDC;
9
10 import com.rabbitmq.client.AMQP.BasicProperties;
11 import com.rabbitmq.client.DefaultConsumer;
12 import com.rabbitmq.client.Envelope;
13
14 import ar.fiuba.taller.common.Command;
15 import ar.fiuba.taller.common.ConfigLoader;
16 import ar.fiuba.taller.common.Constants;
17 import ar.fiuba.taller.common.RemoteQueue;
18 import ar.fiuba.taller.common.Response;
19
20 public class StorageController extends DefaultConsumer implements Runnable {
21
22     private Thread createControllerThread;
23     private Thread queryControllerThread;
24     private Thread removeControllerThread;
25     private Thread responseControllerThread;
26     private BlockingQueue<Command> queryQueue;
27     private BlockingQueue<Command> removeQueue;
28     private BlockingQueue<Command> createQueue;
29     private BlockingQueue<Response> responseQueue;
30     private ConfigLoader configLoader;
31     private Storage storage;
32     private RemoteQueue storageQueue;
33     final static Logger logger = Logger.getLogger(StorageController.class);
34
35     public StorageController(RemoteQueue storageQueue) {
36         super(storageQueue.getChannel());
37         configLoader = ConfigLoader.getInstance();
38         storage = new Storage(configLoader.getShardingFactor(),
39             configLoader.getQueryCountShowPosts(),
40             configLoader.getTtCountShowPosts());
41         this.storageQueue = storageQueue;
42     }
43
44     public void run() {
45         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
46
47         logger.info("Iniciando el storage controller");
48
49         try {
50             logger.info("Cargando la configuracion");
51             configLoader.init(Constants.CONF_FILE);
52
53             logger.info(
54                 "Creando las colas de consultas, removes, creates y response");
55             queryQueue = new ArrayBlockingQueue<Command>(
56                 Constants.COMMAND_QUEUE_SIZE);
57             removeQueue = new ArrayBlockingQueue<Command>(
58                 Constants.COMMAND_QUEUE_SIZE);
59             createQueue = new ArrayBlockingQueue<Command>(
60                 Constants.COMMAND_QUEUE_SIZE);
61             responseQueue = new ArrayBlockingQueue<Response>(
62                 Constants.COMMAND_QUEUE_SIZE);
63
64             logger.info("Instancio los indices de usuarios y hashtags");
65
66             logger.info("Creando los threads de query, remove y create");

```

abr 10, 17 5:28

StorageController.java

Page 2/3

```

67     queryControllerThread = new Thread(
68         new QueryController(queryQueue, responseQueue, storage));
69     removeControllerThread = new Thread(
70         new RemoveController(removeQueue, responseQueue, storage));
71     createControllerThread = new Thread(
72         new CreateController(createQueue, responseQueue,
73             configLoader.getShardingFactor(), storage));
74     responseControllerThread = new Thread(
75         new ResponseController(responseQueue));
76
77     logger.info("Lanzando los threads de query, remove y create");
78     queryControllerThread.start();
79     removeControllerThread.start();
80     createControllerThread.start();
81     responseControllerThread.start();
82
83     logger.info("Me pongo a comer de la cola: " + storageQueue.getHost()
84         + " " + storageQueue.getQueueName());
85     storageQueue.getChannel().basicConsume(storageQueue.getQueueName(),
86         true, this);
87
88     } catch (IOException e) {
89         logger.error("Error al cargar el archivo de configuracion");
90         logger.info(e.toString());
91         e.printStackTrace();
92     }
93 }
94
95 @Override
96 public void handleDelivery(String consumerTag, Envelope envelope,
97     BasicProperties properties, byte[] body) throws IOException {
98     super.handleDelivery(consumerTag, envelope, properties, body);
99     Command command = new Command();
100     try {
101         command.deserialize(body);
102         logger.info("Comando recibido con los siguientes parametros: "
103             + "\nUUID: " + command.getUuid() + "\nUsuario: "
104             + command.getUser() + "\nComando: " + command.getCommand()
105             + "\nMensaje: " + command.getMessage());
106
107         switch (command.getCommand()) {
108             case PUBLISH:
109                 logger.info(
110                     "Comando recibido: PUBLISH. Insertando en la cola de creacion.");
111                 createQueue.put(command);
112                 break;
113             case QUERY:
114                 logger.info(
115                     "Comando recibido: QUERY. Insertando en la cola de consultas.");
116                 queryQueue.put(command);
117                 break;
118             case DELETE:
119                 logger.info(
120                     "Comando recibido: DELETE. Insertando en la cola de borrado.");
121                 removeQueue.put(command);
122                 break;
123             default:
124                 logger.info("Comando recibido invalido. Comando descartado.");
125         }
126     } catch (ClassNotFoundException e) {
127         logger.error("Error al deserializar el comando");
128         logger.info(e.toString());
129         e.printStackTrace();
130     } catch (IOException e) {
131         logger.error("Error al deserializar el comando");
132         logger.info(e.toString());

```

abr 10, 17 5:28

StorageController.java

Page 3/3

```

133     e.printStackTrace();
134 } catch (InterruptedException e) {
135     logger.error("Error al insertar el comando en alguna de las colas");
136     logger.info(e.toString());
137     e.printStackTrace();
138 }
139 }
140
141 }
```

abr 10, 17 5:28

ResponseController.java

Page 1/2

```

1 package ar.fiuba.taller.storage;
2
3 import java.util.concurrent.BlockingQueue;
4 import java.util.concurrent.TimeoutException;
5
6 import org.apache.log4j.Logger;
7
8 import ar.fiuba.taller.common.ConfigLoader;
9 import ar.fiuba.taller.common.RemoteQueue;
10 import ar.fiuba.taller.common.Response;
11
12 import java.io.IOException;
13 import java.util.*;
14
15 public class ResponseController implements Runnable {
16
17     private BlockingQueue<Response> responseQueue;
18     Response response;
19     RemoteQueue remoteQueue;
20     Map<String, RemoteQueue> usersMap;
21     final static Logger logger = Logger.getLogger(ResponseController.class);
22
23     public ResponseController(BlockingQueue<Response> responseQueue) {
24         this.responseQueue = responseQueue;
25         usersMap = new HashMap<String, RemoteQueue>();
26     }
27
28     public void run() {
29         logger.info("Iniciando el response controller");
30         while (true) {
31             try {
32                 logger.info("Esperando siguiente respuesta");
33                 response = responseQueue.take();
34                 remoteQueue = usersMap.get(response.getUser());
35                 if (remoteQueue == null) {
36                     // Creo la cola
37                     remoteQueue = new RemoteQueue(response.getUser(),
38                         ConfigLoader.getInstance().getUsersServer());
39                     remoteQueue.init();
40                     usersMap.put(response.getUser(), remoteQueue);
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                 usersMap.get(response.getUser()).put(response);
51                 logger.info("Respuesta enviada");
52             } catch (InterruptedException e) {
53                 logger.error(
54                     "Error al tomar respuestas de la cola responseQueue");
55                 logger.info(e.toString());
56                 e.printStackTrace();
57             } catch (IOException e) {
58                 logger.error("Error al insertar respuesta en la cola remota");
59                 logger.info(e.toString());
60                 e.printStackTrace();
61             } catch (TimeoutException e) {
62                 logger.error("Error al iniciar la cola remota del usuario");
63                 logger.info(e.toString());
64                 e.printStackTrace();
65             }
66         }
67     }
68 }
```

abr 10, 17 5:28

ResponseController.java

Page 2/2

```

67     }
68 }
69
70 }

```

abr 10, 17 5:29

RemoveController.java

Page 1/2

```

1  package ar.fiuba.taller.storage;
2
3  import java.io.IOException;
4  import java.util.UUID;
5  import java.util.concurrent.BlockingQueue;
6
7  import org.apache.log4j.Logger;
8  import org.apache.log4j.MDC;
9  import org.json.simple.parser.ParseException;
10
11 import ar.fiuba.taller.common.Command;
12 import ar.fiuba.taller.common.Response;
13 import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
14
15 public class RemoveController implements Runnable {
16     private BlockingQueue<Command> removeQueue;
17     private BlockingQueue<Response> responseQueue;
18     private Storage storage;
19     private Command command;
20     private Response response;
21     final static Logger logger = Logger.getLogger(StorageController.class);
22
23     public RemoveController(BlockingQueue<Command> removeQueue,
24                             BlockingQueue<Response> responseQueue, Storage storage) {
25         super();
26         this.removeQueue = removeQueue;
27         this.storage = storage;
28         this.responseQueue = responseQueue;
29     }
30
31     public void run() {
32         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
33         String error_message = "Error al eliminar el mensaje";
34         logger.info("Iniciando el remove controller");
35         while (true) {
36             try {
37                 command = removeQueue.take();
38                 response = new Response();
39                 response.setUuid(UUID.randomUUID());
40                 response.setUser(command.getUser());
41                 storage.delete(command);
42                 response.setMessage("Borrado exitoso");
43                 response.setResponse_status(RESPONSE_STATUS.OK);
44             } catch (InterruptedException e) {
45                 response.setResponse_status(RESPONSE_STATUS.ERROR);
46                 response.setMessage(error_message);
47                 logger.info(e.toString());
48                 e.printStackTrace();
49             } catch (IOException e) {
50                 response.setResponse_status(RESPONSE_STATUS.ERROR);
51                 response.setMessage(error_message);
52                 logger.error("Error borrar el mensaje");
53                 logger.info(e.toString());
54                 e.printStackTrace();
55             } catch (ParseException e) {
56                 response.setResponse_status(RESPONSE_STATUS.ERROR);
57                 response.setMessage(error_message);
58                 logger.error("Error borrar el mensaje");
59                 logger.info(e.toString());
60                 e.printStackTrace();
61             } finally {
62                 try {
63                     responseQueue.put(response);
64                 } catch (InterruptedException e) {
65                     logger.error("No se pudo enviar la respuesta");
66                     logger.info(e.toString());

```

abr 10, 17 5:29

RemoveController.java

Page 2/2

```

67         e.printStackTrace();
68     }
69 }
70 }
71 }
72 }

```

abr 10, 17 5:28

QueryController.java

Page 1/2

```

1  package ar.fiuba.taller.storage;
2
3  import java.io.IOException;
4  import java.util.UUID;
5  import java.util.concurrent.BlockingQueue;
6
7  import org.apache.log4j.Logger;
8  import org.apache.log4j.MDC;
9  import org.json.simple.parser.ParseException;
10
11 import ar.fiuba.taller.common.Command;
12 import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
13 import ar.fiuba.taller.common.Response;
14
15 public class QueryController implements Runnable {
16     private BlockingQueue<Command> queryQueue;
17     private BlockingQueue<Response> responseQueue;
18     private Storage storage;
19     private Command command;
20     private Response response;
21     final static Logger logger = Logger.getLogger(QueryController.class);
22
23     public QueryController(BlockingQueue<Command> queryQueue,
24         BlockingQueue<Response> responseQueue, Storage storage) {
25         super();
26         this.queryQueue = queryQueue;
27         this.responseQueue = responseQueue;
28         this.storage = storage;
29     }
30
31     public void run() {
32         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
33         String queryResult;
34         // Este mensaje deberia ser configurable
35         String error_message = "Error al consultar";
36         logger.info("Iniciando el query controller");
37         while (true) {
38             try {
39                 command = queryQueue.take();
40                 response = new Response();
41                 response.setUuid(UUID.randomUUID());
42                 response.setUser(command.getUser());
43                 response.setMessage(storage.query(command));
44                 logger.debug(response.getMessage());
45                 response.setResponse_status(RESPONSE_STATUS.OK);
46             } catch (InterruptedException e) {
47                 response.setResponse_status(RESPONSE_STATUS.ERROR);
48                 response.setMessage(error_message);
49                 logger.error("Error al sacar comando de la cola removeQueue");
50                 logger.info(e.toString());
51                 e.printStackTrace();
52             } catch (IOException e) {
53                 response.setResponse_status(RESPONSE_STATUS.ERROR);
54                 response.setMessage(error_message);
55                 logger.error("Error borrar el mensaje");
56                 logger.info(e.toString());
57                 e.printStackTrace();
58             } catch (ParseException e) {
59                 response.setResponse_status(RESPONSE_STATUS.ERROR);
60                 response.setMessage(error_message);
61                 logger.error("Error borrar el mensaje");
62                 logger.info(e.toString());
63                 e.printStackTrace();
64             } finally {
65                 try {
66                     responseQueue.put(response);

```


abr 10, 17 5:28

QueryController.java

Page 2/2

```

67     } catch (InterruptedException e) {
68         logger.error("No se pudo enviar la respuesta");
69         logger.info(e.toString());
70         e.printStackTrace();
71     }
72 }
73 }
74 }
75 }
76 }

```

abr 10, 17 5:28

CreateController.java

Page 1/2

```

1  package ar.fiuba.taller.storage;
2
3  import java.io.IOException;
4  import java.util.UUID;
5  import java.util.concurrent.BlockingQueue;
6
7  import org.apache.log4j.Logger;
8  import org.apache.log4j.MDC;
9  import org.json.simple.parser.ParseException;
10
11 import ar.fiuba.taller.common.Command;
12 import ar.fiuba.taller.common.Response;
13 import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
14
15 public class CreateController implements Runnable {
16     private BlockingQueue<Command> createQueue;
17     private BlockingQueue<Response> responseQueue;
18     private Command command;
19     private Storage storage;
20     private Response response;
21     final static Logger logger = Logger.getLogger(CreateController.class);
22
23     public CreateController(BlockingQueue<Command> createQueue,
24                             BlockingQueue<Response> responseQueue, int shardingFactor,
25                             Storage storage) {
26         super();
27         this.createQueue = createQueue;
28         this.responseQueue = responseQueue;
29         this.storage = storage;
30     }
31
32     public void run() {
33         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
34         logger.info("Iniciando el create controller");
35
36         while (true) {
37             String error_message = "Error al crear el mensaje";
38             try {
39                 command = createQueue.take();
40                 response = new Response();
41                 response.setUuid(UUID.randomUUID());
42                 response.setUser(command.getUser());
43                 storage.saveMessage(command);
44                 response.setMessage("Creacion exitosa");
45                 response.setResponse_status(RESPONSE_STATUS.OK);
46             } catch (InterruptedException e) {
47                 response.setResponse_status(RESPONSE_STATUS.ERROR);
48                 response.setMessage(error_message);
49                 logger.error("Error al leer un comando de la cola createQueue");
50                 logger.info(e.toString());
51                 e.printStackTrace();
52             } catch (IOException e) {
53                 response.setResponse_status(RESPONSE_STATUS.ERROR);
54                 response.setMessage(error_message);
55                 logger.error("Error al guardar el comando en la base de datos");
56                 logger.info(e.toString());
57                 e.printStackTrace();
58             } catch (ParseException e) {
59                 response.setResponse_status(RESPONSE_STATUS.ERROR);
60                 response.setMessage(error_message);
61                 logger.error("Error al actualizar alguno de los indices");
62                 logger.info(e.toString());
63                 e.printStackTrace();
64             } finally {
65                 try {
66                     responseQueue.put(response);

```

abr 10, 17 5:28

CreateController.java

Page 2/2

```

67     } catch (InterruptedException e) {
68         logger.error("No se pudo enviar la respuesta");
69         logger.info(e.toString());
70         e.printStackTrace();
71     }
72 }
73 }
74 }
75 }

```

abr 10, 17 5:28

App.java

Page 1/1

```

1  package ar.fiuba.taller.storage;
2
3  import java.io.IOException;
4  import java.util.concurrent.TimeoutException;
5
6  import org.apache.log4j.Logger;
7  import org.apache.log4j.MDC;
8  import org.apache.log4j.PropertyConfigurator;
9
10 import ar.fiuba.taller.common.ConfigLoader;
11 import ar.fiuba.taller.common.Constants;
12 import ar.fiuba.taller.common.RemoteQueue;
13
14 public class App {
15     final static Logger logger = Logger.getLogger(App.class);
16
17     public static void main(String[] args) {
18         PropertyConfigurator.configure(Constants.LOGGER_CONF);
19         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
20         try {
21             ConfigLoader.getInstance().init(Constants.CONF_FILE);
22             logger.info("Entablando conexion con el broker");
23             RemoteQueue storageQueue = new RemoteQueue(
24                 ConfigLoader.getInstance().getStorageRequestQueueName(),
25                 ConfigLoader.getInstance().getStorageResquestQueueHost());
26             storageQueue.init();
27             logger.info("Disparando el storage controller");
28             Thread storageControllerThread = new Thread(
29                 new StorageController(storageQueue));
30             logger.info("Starteando el storage controller");
31             storageControllerThread.start();
32             logger.info("Joineando el storage controller");
33             storageControllerThread.join();
34         } catch (InterruptedException e) {
35             logger.error("Error al joinear el storage controller");
36             logger.info(e.toString());
37             e.printStackTrace();
38         } catch (IOException e) {
39             logger.error("Error al cargar la configuracion");
40             logger.info(e.toString());
41             e.printStackTrace();
42         } catch (TimeoutException e) {
43             logger.error("Error al cerrar la cola storageQueue");
44             logger.info(e.toString());
45             e.printStackTrace();
46         }
47     }
48 }

```

abr 10, 17 5:28

AppTest.java

Page 1/1

```

1 package ar.fiuba.taller.dispatcher;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8  * Unit test for simple App.
9  */
10 public class AppTest extends TestCase {
11     /**
12      * Create the test case
13      *
14      * @param testName
15      *      name of the test case
16      */
17     public AppTest(String testName) {
18         super(testName);
19     }
20
21     /**
22      * @return the suite of tests being tested
23      */
24     public static Test suite() {
25         return new TestSuite(AppTest.class);
26     }
27
28     /**
29      * Rigourous Test :-)
30      */
31     public void testApp() {
32         assertTrue(true);
33     }
34 }

```

abr 10, 17 5:28

StorageController.java

Page 1/1

```

1 package ar.fiuba.taller.dispatcher;
2
3 import java.io.IOException;
4 import java.util.concurrent.BlockingQueue;
5
6 import org.apache.log4j.Logger;
7 import org.apache.log4j.MDC;
8
9 import ar.fiuba.taller.common.Command;
10 import ar.fiuba.taller.common.RemoteQueue;
11
12 public class StorageController implements Runnable {
13     private BlockingQueue<Command> storageCommandQueue;
14     private RemoteQueue storageQueue;
15
16     final static Logger logger = Logger.getLogger(StorageController.class);
17
18     public StorageController(BlockingQueue<Command> storageCommandQueue,
19         RemoteQueue storageQueue) {
20         this.storageCommandQueue = storageCommandQueue;
21         this.storageQueue = storageQueue;
22     }
23
24     public void run() {
25         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
26         Command command;
27
28         logger.info("Iniciando el storage controller");
29         while (true) {
30             try {
31                 command = storageCommandQueue.take();
32                 logger.info("Comando recibido con los siguientes parametros: "
33                     + "\nUsuario: " + command.getUser() + "\nComando: "
34                     + command.getCommand() + "\nMensaje: "
35                     + command.getMessage());
36                 storageQueue.put(command);
37                 logger.info("Comando enviado al storage");
38             } catch (InterruptedException e) {
39                 logger.error(
40                     "Error al obtener el comando de la cola "
41                     + "storageCommandQueue");
42                 logger.info(e.toString());
43                 e.printStackTrace();
44             } catch (IOException e) {
45                 logger.error(
46                     "Error al insertar el comando de la cola storageQueue");
47                 logger.info(e.toString());
48                 e.printStackTrace();
49             }
50         }
51     }
52 }

```

abr 10, 17 5:28

LoggerController.java

Page 1/1

```

1 package ar.fiuba.taller.dispatcher;
2
3 import java.io.IOException;
4 import java.util.concurrent.BlockingQueue;
5
6 import org.apache.log4j.Logger;
7 import org.apache.log4j.MDC;
8
9 import ar.fiuba.taller.common.Command;
10 import ar.fiuba.taller.common.RemoteQueue;
11
12 public class LoggerController implements Runnable {
13
14     private BlockingQueue<Command> loggerCommandQueue;
15     private RemoteQueue loggerQueue;
16
17     final static Logger logger = Logger.getLogger(LoggerController.class);
18
19     public LoggerController(BlockingQueue<Command> loggerCommandQueue,
20         RemoteQueue loggerQueue) {
21         this.loggerCommandQueue = loggerCommandQueue;
22         this.loggerQueue = loggerQueue;
23     }
24
25     public void run() {
26         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
27         Command command;
28
29         logger.info("Iniciando el logger controller");
30         while (true) {
31             try {
32                 command = loggerCommandQueue.take();
33                 logger.info("Comando recibido con los siguientes parametros: "
34                     + "\nUsuario: " + command.getUser() + "\nComando: "
35                     + command.getCommand() + "\nMensaje: "
36                     + command.getMessage());
37                 loggerQueue.put(command);
38                 logger.info("Comando enviado al logger");
39             } catch (InterruptedException e) {
40                 logger.error(
41                     "Error al obtener el comando de la cola "
42                     + "loggerCommandQueue");
43                 logger.info(e.toString());
44                 e.printStackTrace();
45             } catch (IOException e) {
46                 logger.error(
47                     "Error al insertar el comando de la cola loggerQueue");
48                 logger.info(e.toString());
49                 e.printStackTrace();
50             }
51         }
52     }
53 }

```

abr 10, 17 5:28

Dispatcher.java

Page 1/2

```

1 package ar.fiuba.taller.dispatcher;
2
3 import java.io.IOException;
4 import java.util.concurrent.ArrayBlockingQueue;
5 import java.util.concurrent.BlockingQueue;
6 import java.util.concurrent.TimeoutException;
7
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.ConfigLoader;
13 import ar.fiuba.taller.common.Constants;
14 import ar.fiuba.taller.common.RemoteQueue;
15
16 public class Dispatcher implements Runnable {
17
18     Thread analyzerControllerThread;
19     Thread dispatcherControllerThread;
20     Thread storageControllerThread;
21     Thread loggerControllerThread;
22     RemoteQueue dispatcherQueue;
23     RemoteQueue storageQueue;
24     RemoteQueue analyzerQueue;
25     RemoteQueue loggerQueue;
26     BlockingQueue<Command> storageCommandQueue;
27     BlockingQueue<Command> analyzerCommandQueue;
28     BlockingQueue<Command> loggerCommandQueue;
29     ConfigLoader configLoader;
30     final static Logger logger = Logger.getLogger(Dispatcher.class);
31
32     public Dispatcher() {
33         configLoader = ConfigLoader.getInstance();
34     }
35
36     public void run() {
37         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
38
39         logger.info("Iniciando el dispatcher");
40
41         try {
42             logger.info("Cargando la configuracion");
43             configLoader.init(Constants.CONF_FILE);
44
45             initDispatcher();
46             startDispatcher();
47             terminateDispatcher();
48
49         } catch (InterruptedException e) {
50             logger.error("Error al joinear los threads");
51             logger.info(e.toString());
52             e.printStackTrace();
53         } catch (IOException e) {
54             logger.error("Error al cargar el archivo de configuracion");
55             logger.info(e.toString());
56             e.printStackTrace();
57         } catch (TimeoutException e) {
58             logger.error("Error al iniciar las colas remotas");
59             logger.info(e.toString());
60             e.printStackTrace();
61         }
62     }
63
64     private void initDispatcher() throws IOException, TimeoutException {
65
66         logger.info("Creando las colas internas");

```

abr 10, 17 5:28

Dispatcher.java

Page 2/2

```

67 analyzerCommandQueue = new ArrayBlockingQueue<Command>(
68     Constants.COMMAND_QUEUE_SIZE);
69 storageCommandQueue = new ArrayBlockingQueue<Command>(
70     Constants.COMMAND_QUEUE_SIZE);
71 loggerCommandQueue = new ArrayBlockingQueue<Command>(
72     Constants.COMMAND_QUEUE_SIZE);
73
74 logger.info("Creando las conexiones a los brokers");
75 logger.info("Creando la cola del dispatcher");
76 dispatcherQueue = new RemoteQueue(configLoader.getDispatcherQueueName(),
77     configLoader.getDispatcherQueueHost());
78 dispatcherQueue.init();
79 logger.info("Creando la cola hacia el analyzer");
80 analyzerQueue = new RemoteQueue(configLoader.getAnalyzerQueueName(),
81     configLoader.getAnalyzerQueueHost());
82 analyzerQueue.init();
83 logger.info("Creando la cola hacia el storage");
84 storageQueue = new RemoteQueue(
85     configLoader.getStorageRequestQueueName(),
86     configLoader.getStorageResquestQueueHost());
87 storageQueue.init();
88 logger.info("Creando la cola hacia el logger");
89 loggerQueue = new RemoteQueue(configLoader.getAuditLoggerQueueName(),
90     configLoader.getAuditLoggerQueueHost());
91 loggerQueue.init();
92
93 logger.info("Creando los threads de los workers");
94 analyzerControllerThread = new Thread(
95     new AnalyzerController(analyzerCommandQueue, analyzerQueue));
96 dispatcherControllerThread = new Thread(
97     new DispatcherController(dispatcherQueue, storageCommandQueue,
98     analyzerCommandQueue, loggerCommandQueue));
99 storageControllerThread = new Thread(
100     new StorageController(storageCommandQueue, storageQueue));
101 loggerControllerThread = new Thread(
102     new LoggerController(loggerCommandQueue, loggerQueue));
103
104 }
105
106 private void startDispatcher() {
107
108     logger.info("Iniciando los threads de los workers");
109     analyzerControllerThread.start();
110     dispatcherControllerThread.start();
111     storageControllerThread.start();
112     loggerControllerThread.start();
113
114 }
115
116 private void terminateDispatcher() throws InterruptedException {
117
118     logger.info("Joiando los threads de los workers");
119     analyzerControllerThread.join();
120     dispatcherControllerThread.join();
121     storageControllerThread.join();
122     loggerControllerThread.join();
123
124 }

```

abr 10, 17 5:28

DispatcherController.java

Page 1/2

```

1 package ar.fiuba.taller.dispatcher;
2
3 import java.io.IOException;
4 import java.util.concurrent.BlockingQueue;
5
6 import org.apache.log4j.Logger;
7 import org.apache.log4j.MDC;
8
9 import com.rabbitmq.client.Channel;
10 import com.rabbitmq.client.DefaultConsumer;
11 import com.rabbitmq.client.Envelope;
12 import com.rabbitmq.client.AMQP.BasicProperties;
13
14 import ar.fiuba.taller.common.Command;
15 import ar.fiuba.taller.common.RemoteQueue;
16 import ar.fiuba.taller.common.Response;
17
18 public class DispatcherController extends DefaultConsumer implements Runnable {
19
20     RemoteQueue dispatcherQueue;
21     BlockingQueue<Command> storageCommandQueue;
22     BlockingQueue<Command> analyzerCommandQueue;
23     BlockingQueue<Command> loggerCommandQueue;
24     final static Logger logger = Logger.getLogger(DispatcherController.class);
25
26     public DispatcherController(RemoteQueue dispatcherQueue,
27         BlockingQueue<Command> storageCommandQueue,
28         BlockingQueue<Command> analyzerCommandQueue,
29         BlockingQueue<Command> loggerCommandQueue) {
30         super(dispatcherQueue.getChannel());
31         this.storageCommandQueue = storageCommandQueue;
32         this.analyzerCommandQueue = analyzerCommandQueue;
33         this.loggerCommandQueue = loggerCommandQueue;
34         this.dispatcherQueue = dispatcherQueue;
35     }
36
37     public void run() {
38         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
39         logger.info("Iniciando el dispatcher controller");
40         // while(true) {
41         try {
42             dispatcherQueue.getChannel()
43                 .basicConsume(dispatcherQueue.getQueueName(), true, this);
44         } catch (IOException e) {
45             // TODO Auto-generated catch block
46             e.printStackTrace();
47         }
48         // }
49     }
50
51 @Override
52 public void handleDelivery(String consumerTag, Envelope envelope,
53     BasicProperties properties, byte[] body) throws IOException {
54     super.handleDelivery(consumerTag, envelope, properties, body);
55     Command command = new Command();
56     try {
57         command.deserialize(body);
58         logger.info("Comando recibido con los siguientes parametros: "
59             + "\nUsuario: " + command.getUser() + "\nComando: "
60             + command.getCommand() + "\nMensaje: "
61             + command.getMessage());
62         switch (command.getCommand()) {
63             case PUBLISH:
64                 logger.info("Enviando mensaje a la cola del storage");
65                 storageCommandQueue.put(command);
66                 logger.info("Enviando mensaje a la cola del analyzer");

```

abr 10, 17 5:28

DispatcherController.java

Page 2/2

```

67     analyzerCommandQueue.put(command);
68     logger.info("Enviando mensaje a la cola del logger");
69     loggerCommandQueue.put(command);
70     break;
71     case QUERY:
72         logger.info("Enviando mensaje a la cola del storage");
73         storageCommandQueue.put(command);
74         logger.info("Enviando mensaje a la cola del logger");
75         loggerCommandQueue.put(command);
76         break;
77     case DELETE:
78         logger.info("Enviando mensaje a la cola del storage");
79         storageCommandQueue.put(command);
80         logger.info("Enviando mensaje a la cola del logger");
81         loggerCommandQueue.put(command);
82         break;
83     case FOLLOW:
84         logger.info("Enviando mensaje a la cola del analyzer");
85         analyzerCommandQueue.put(command);
86         logger.info("Enviando mensaje a la cola del logger");
87         loggerCommandQueue.put(command);
88         break;
89     default:
90         logger.error("Comando invalido");
91         break;
92     }
93     catch (ClassNotFoundException e) {
94         logger.info("Error al deserializar el comando");
95         logger.info(e.toString());
96         e.printStackTrace();
97     } catch (IOException e) {
98         logger.info("Error al deserializar el comando");
99         logger.info(e.toString());
100        e.printStackTrace();
101    } catch (InterruptedException e) {
102        logger.error("Error al insertar el comando en alguna de las colas");
103        logger.info(e.toString());
104        e.printStackTrace();
105    }
106 }
107 }

```

abr 10, 17 5:28

App.java

Page 1/1

```

1  package ar.fiuba.taller.dispatcher;
2
3  import org.apache.log4j.Logger;
4  import org.apache.log4j.MDC;
5  import org.apache.log4j.PropertyConfigurator;
6
7  import ar.fiuba.taller.common.Constants;
8
9  public class App {
10     final static Logger logger = Logger.getLogger(App.class);
11
12     public static void main(String[] args) {
13         PropertyConfigurator.configure(Constants.LOGGER_CONF);
14         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
15         logger.info("Disparando el dispatcher");
16         Thread dispatcherThread = new Thread(new Dispatcher());
17         dispatcherThread.start();
18         try {
19             dispatcherThread.join();
20         } catch (InterruptedException e) {
21             logger.error("Error al joinear el dispatcher");
22             logger.info(e.toString());
23             e.printStackTrace();
24         }
25     }
26 }

```

abr 10, 17 5:28

AnalyzerController.java

Page 1/1

```

1 package ar.fiuba.taller.dispatcher;
2
3 import java.io.IOException;
4 import java.util.concurrent.BlockingQueue;
5
6 import org.apache.log4j.Logger;
7 import org.apache.log4j.MDC;
8
9 import ar.fiuba.taller.common.Command;
10 import ar.fiuba.taller.common.RemoteQueue;
11
12 public class AnalyzerController implements Runnable {
13     private BlockingQueue<Command> analyzerCommandQueue;
14     private RemoteQueue analyzerQueue;
15
16     final static Logger logger = Logger.getLogger(AnalyzerController.class);
17
18     public AnalyzerController(BlockingQueue<Command> analyzerCommandQueue,
19                             RemoteQueue analyzerQueue) {
20         this.analyzerCommandQueue = analyzerCommandQueue;
21         this.analyzerQueue = analyzerQueue;
22     }
23
24     public void run() {
25         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
26         Command command;
27
28         logger.info("Iniciando el analyzer controller");
29         while (true) {
30             try {
31                 command = analyzerCommandQueue.take();
32                 logger.info("Comando recibido con los siguientes parametros: "
33                     + "\nUsuario: " + command.getUser() + "\nComando: "
34                     + command.getCommand() + "\nMensaje: "
35                     + command.getMessage());
36                 analyzerQueue.put(command);
37                 logger.info("Comando enviado al analyzer");
38             } catch (InterruptedException e) {
39                 logger.error(
40                     "Error al obtener el comando de la cola "
41                     + "analyzerCommandQueue");
42                 logger.info(e.toString());
43                 e.printStackTrace();
44             } catch (IOException e) {
45                 logger.error(
46                     "Error al insertar el comando de la cola analyzerQueue");
47                 logger.info(e.toString());
48                 e.printStackTrace();
49             }
50         }
51     }
52 }

```

abr 02, 17 9:12

AppTest.java

Page 1/1

```

1 package ar.fiuba.taller.crea_deploy;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8  * Unit test for simple App.
9  */
10 public class AppTest
11     extends TestCase
12 {
13     /**
14      * Create the test case
15      * @param testName name of the test case
16      */
17     public AppTest( String testName )
18     {
19         super( testName );
20     }
21
22     /**
23      * @return the suite of tests being tested
24      */
25     public static Test suite()
26     {
27         return new TestSuite( AppTest.class );
28     }
29
30     /**
31      * Rigourous Test :-)
32      */
33     public void testApp()
34     {
35         assertTrue( true );
36     }
37 }

```

abr 02, 17 9:12

App.java

Page 1/1

```

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

```

abr 10, 17 5:27

AppTest.java

Page 1/1

```

1 package ar.fiuba.taller.common;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8  * Unit test for simple App.
9  */
10 public class AppTest extends TestCase {
11     /**
12      * Create the test case
13      *
14      * @param testName
15      *        name of the test case
16      */
17     public AppTest(String testName) {
18         super(testName);
19     }
20
21     /**
22      * @return the suite of tests being tested
23      */
24     public static Test suite() {
25         return new TestSuite(AppTest.class);
26     }
27
28     /**
29      * Rigourous Test :-)
30      */
31     public void testApp() {
32         assertTrue(true);
33     }
34 }

```


abr 10, 17 5:27

Response.java

Page 1/2

```

1 package ar.fiuba.taller.common;
2
3 import java.io.ByteArrayInputStream;
4 import java.io.ByteArrayOutputStream;
5 import java.io.IOException;
6 import java.io.ObjectInput;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutput;
9 import java.io.ObjectOutputStream;
10 import java.io.Serializable;
11 import java.util.UUID;
12
13 import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
14
15 public class Response implements Serializable, ISerialize {
16
17     private UUID uuid;
18     private String user;
19     private RESPONSE_STATUS response_status;
20     private String message;
21
22     public Response(UUID uuid, RESPONSE_STATUS response_status,
23                     String message) {
24         super();
25         this.uuid = uuid;
26         this.response_status = response_status;
27         this.message = message;
28     }
29
30     public Response() {
31         super();
32         this.uuid = null;
33         this.response_status = null;
34         this.message = null;
35     }
36
37     public byte[] serialize() throws IOException {
38         ByteArrayOutputStream os = new ByteArrayOutputStream();
39         ObjectOutput objOut = new ObjectOutputStream(os);
40
41         objOut.writeObject(this);
42         byte responseArray[] = os.toByteArray();
43         objOut.close();
44         os.close();
45         return responseArray;
46     }
47
48     public void deserialize(byte[] responseArray)
49         throws IOException, ClassNotFoundException {
50         ByteArrayInputStream is = new ByteArrayInputStream(responseArray);
51         ObjectInput objIn = new ObjectInputStream(is);
52         Response tmp;
53         tmp = (Response) objIn.readObject();
54         objIn.close();
55         is.close();
56         uuid = tmp.getUuid();
57         response_status = tmp.getResponse_status();
58         message = tmp.getMessage();
59     }
60
61     public UUID getUuid() {
62         return uuid;
63     }
64
65     public void setUuid(UUID uuid) {
66         this.uuid = uuid;

```

abr 10, 17 5:27

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 }

```

abr 10, 17 5:27

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 import com.rabbitmq.client.*;
7
8 public class RemoteQueue {
9     private String queueName;
10    private String host;
11    private ConnectionFactory factory;
12    private Connection connection;
13    private Channel channel;
14
15    public Channel getChannel() {
16        return channel;
17    }
18
19    public RemoteQueue(String queueName, String host) {
20        this.queueName = queueName;
21        this.host = host;
22    }
23
24    public String getQueueName() {
25        return queueName;
26    }
27
28    public void setQueueName(String queueName) {
29        this.queueName = queueName;
30    }
31
32    public String getHost() {
33        return host;
34    }
35
36    public void setHost(String host) {
37        this.host = host;
38    }
39
40    public void init() throws IOException, TimeoutException {
41        factory = new ConnectionFactory();
42        factory.setHost(host);
43        connection = factory.newConnection();
44        channel = connection.createChannel();
45        channel.queueDeclareNoWait(queueName, false, false, false, null);
46    }
47
48    public void close() throws IOException, TimeoutException {
49        channel.close();
50        connection.close();
51    }
52
53    public void put(ISerialize message) throws IOException {
54        channel.basicPublish("", queueName, null, message.serialize());
55    }
56
57 }

```

abr 10, 17 5:27

ISerialize.java

Page 1/1

```

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

```

abr 10, 17 5:27

Constants.java

Page 1/2

```

1 package ar.fiuba.taller.common;
2
3 import java.text.SimpleDateFormat;
4 import java.util.Collections;
5 import java.util.HashMap;
6 import java.util.Map;
7
8 public class Constants {
9
10     public static final int COMMAND_QUEUE_SIZE = 100;
11     public static final int RESPONSE_QUEUE_SIZE = 100;
12     public static final String LOGGER_CONF = "conf/log4j.properties";
13     public static final String COMMAND_SCRIPT_FOLDER = "scripts";
14     public static final String COMMAND_SCRIPT_EXTENSION = ".json";
15     public static final String COMMAND_ARRAY = "commands";
16     public static final String COMMAND_KEY = "command";
17     public static final String USER_KEY = "user";
18     public static final String NAME_KEY = "name";
19     public static final String USERS_KEY = "users";
20     public static final String MESSAGE_KEY = "message";
21     public static final String USERS_FILE = "conf/users.json";
22     public static final String CONF_FILE = "conf/configuration.properties";
23     public static final String LOGS_DIR = "log";
24     public static final String EVENT_VIEWER_FILE = "user";
25     public static final String EVENT_VIEWER_FILE_EXTENSION = ".events";
26     public static final String RESPONSE_QUEUE_HOST = "responseQueueHost";
27     public static final String RESPONSE_QUEUE_NAME = "responseQueueName";
28     public static final String DISPATCHER_QUEUE_HOST = "dispatcherQueueHost";
29     public static final String DISPATCHER_QUEUE_NAME = "dispatcherQueueName";
30     public static final String AUDIT_LOGGER_QUEUE_HOST = "auditLoggerQueueHost";
31     public static final String AUDIT_LOGGER_QUEUE_NAME = "auditLoggerQueueName";
32     public static final String STORAGE_REQUEST_QUEUE_HOST = "storageRequestQueueHost";
33     public static final String STORAGE_REQUEST_QUEUE_NAME = "storageRequestQueueName";
34     public static final String STORAGE_RESPONSE_QUEUE_HOST = "storageResponseQueueHost";
35
36     public static final String STORAGE_RESPONSE_QUEUE_NAME = "storageResponseQueueName";
37
38     public static final String ANALYZER_QUEUE_HOST = "analyzerQueueHost";
39     public static final String ANALYZER_QUEUE_NAME = "analyzerQueueName";
40     public static final String USERS_SERVER = "usersServer";
41     public static final String SHARDING_FACTOR = "shardingFactor";
42     public static final String AUDIT_LOG_FILE = "log/audit.log";
43     public static final String DB_DIR = "db";
44     public static final String DB_INDEX_DIR = "idx";
45     public static final String DB_USER_INDEX = "user.json";
46     public static final String DB_HASHTAG_INDEX = "hashtag.json";
47     public static final String DB_TT = "tt.json";
48     public static final SimpleDateFormat SDF = new SimpleDateFormat(
49         "yyyy-MM-dd HH:mm:ss");
50     public static final String QUERY_COUNT_SHOW_POSTS = "queryCountShowPosts";
51     public static final String TT_COUNT_SHOW_POST = "ttCountShowPosts";
52     public static final String USER_READ_MODE = "r";
53     public static final String USER_WRITE_MODE = "w";
54
55     public static enum COMMAND {
56         PUBLISH, QUERY, DELETE, FOLLOW
57     };
58
59     public static Map<String, COMMAND> COMMAND_MAP;
60     static {
61         Map<String, COMMAND> tmpMap = new HashMap<String, COMMAND>();
62         tmpMap.put("PUBLISH", COMMAND.PUBLISH);
63         tmpMap.put("QUERY", COMMAND.QUERY);
64         tmpMap.put("DELETE", COMMAND.DELETE);
65         tmpMap.put("FOLLOW", COMMAND.FOLLOW);
66         COMMAND_MAP = Collections.unmodifiableMap(tmpMap);
67     }
68 }

```

abr 10, 17 5:27

Constants.java

Page 2/2

```

65     }
66
67     public static enum RESPONSE_STATUS {
68         OK, ERROR, REGISTERED
69     }
70
71     public static Map<String, RESPONSE_STATUS> RESPONSE_STATUS_MAP;
72     static {
73         Map<String, RESPONSE_STATUS> tmpMap1 = new HashMap<String, RESPONSE_STATUS>();
74
75         tmpMap1 = new HashMap<String, Constants.RESPONSE_STATUS>();
76         tmpMap1.put("OK", RESPONSE_STATUS.OK);
77         tmpMap1.put("ERROR", RESPONSE_STATUS.ERROR);
78         tmpMap1.put("REGISTERED", RESPONSE_STATUS.REGISTERED);
79         RESPONSE_STATUS_MAP = Collections.unmodifiableMap(tmpMap1);
80     }
81 }

```

abr 10, 17 5:27

ConfigLoader.java

Page 1/3

```

1 package ar.fiuba.taller.common;
2
3 import java.io.FileInputStream;
4 import java.io.IOException;
5 import java.util.Properties;
6
7 public class ConfigLoader {
8
9     private static ConfigLoader instance = null;
10    private String responseQueueHost;
11    private String responseQueueName;
12    private String dispatcherQueueHost;
13    private String dispatcherQueueName;
14    private String auditLoggerQueueHost;
15    private String auditLoggerQueueName;
16    private String storageResquestQueueHost;
17    private String storageRequestQueueName;
18    private String storageResponseQueueHost;
19    private String storageResponseQueueName;
20    private String analyzerQueueHost;
21    private String analyzerQueueName;
22    private String usersServer;
23    private int queryCountShowPosts;
24    private int ttCountShowPosts;
25    private int shardingFactor;
26
27    public int getShardingFactor() {
28        return shardingFactor;
29    }
30
31    protected ConfigLoader() {
32        // TODO Auto-generated constructor stub
33    }
34
35    public static ConfigLoader getInstance() {
36        if (instance == null) {
37            instance = new ConfigLoader();
38        }
39        return instance;
40    }
41
42    public void init(String configFile) throws IOException {
43        Properties properties = new Properties();
44        FileInputStream input = new FileInputStream(configFile);
45
46        // cargamos el archivo de propiedades
47        properties.load(input);
48
49        // obtenemos las propiedades
50        responseQueueHost = properties
51            .getProperty(Constants.RESPONSE_QUEUE_HOST);
52        responseQueueName = properties
53            .getProperty(Constants.RESPONSE_QUEUE_NAME);
54        dispatcherQueueHost = properties
55            .getProperty(Constants.DISPATCHER_QUEUE_HOST);
56        dispatcherQueueName = properties
57            .getProperty(Constants.DISPATCHER_QUEUE_NAME);
58        auditLoggerQueueHost = properties
59            .getProperty(Constants.AUDIT_LOGGER_QUEUE_HOST);
60        auditLoggerQueueName = properties
61            .getProperty(Constants.AUDIT_LOGGER_QUEUE_NAME);
62        storageResquestQueueHost = properties
63            .getProperty(Constants.STORAGE_REQUEST_QUEUE_HOST);
64        storageRequestQueueName = properties
65            .getProperty(Constants.STORAGE_REQUEST_QUEUE_NAME);
66        storageResponseQueueHost = properties

```

abr 10, 17 5:27

ConfigLoader.java

Page 2/3

```

67        .getProperty(Constants.STORAGE_RESPONSE_QUEUE_HOST);
68        storageResponseQueueName = properties
69            .getProperty(Constants.STORAGE_RESPONSE_QUEUE_NAME);
70        analyzerQueueHost = properties
71            .getProperty(Constants.ANALYZER_QUEUE_HOST);
72        analyzerQueueName = properties
73            .getProperty(Constants.ANALYZER_QUEUE_NAME);
74        usersServer = properties.getProperty(Constants.USERS_SERVER);
75        shardingFactor = Integer
76            .parseInt(properties.getProperty(Constants.SHARDING_FACTOR));
77        queryCountShowPosts = Integer.parseInt(
78            properties.getProperty(Constants.QUERY_COUNT_SHOW_POSTS));
79        ttCountShowPosts = Integer
80            .parseInt(properties.getProperty(Constants.TT_COUNT_SHOW_POST));
81    }
82
83    public String getResponseQueueHost() {
84        return responseQueueHost;
85    }
86
87    public String getResponseQueueName() {
88        return responseQueueName;
89    }
90
91    public String getDispatcherQueueHost() {
92        return dispatcherQueueHost;
93    }
94
95    public String getDispatcherQueueName() {
96        return dispatcherQueueName;
97    }
98
99    public String getAuditLoggerQueueHost() {
100        return auditLoggerQueueHost;
101    }
102
103    public String getAuditLoggerQueueName() {
104        return auditLoggerQueueName;
105    }
106
107    public String getStorageResquestQueueHost() {
108        return storageResquestQueueHost;
109    }
110
111    public String getStorageRequestQueueName() {
112        return storageRequestQueueName;
113    }
114
115    public String getStorageResponseQueueHost() {
116        return storageResponseQueueHost;
117    }
118
119    public String getStorageResponseQueueName() {
120        return storageResponseQueueName;
121    }
122
123    public String getAnalyzerQueueHost() {
124        return analyzerQueueHost;
125    }
126
127    public String getAnalyzerQueueName() {
128        return analyzerQueueName;
129    }
130
131    public int getQueryCountShowPosts() {
132        return queryCountShowPosts;

```

abr 10, 17 5:27

ConfigLoader.java

Page 3/3

```

133     }
134
135     public int getTtCountShowPosts() {
136         return ttCountShowPosts;
137     }
138
139     public String getUsersServer() {
140         return usersServer;
141     }
142 }

```

abr 10, 17 5:27

Command.java

Page 1/2

```

1  package ar.fiuba.taller.common;
2
3  import java.io.ByteArrayInputStream;
4  import java.io.ByteArrayOutputStream;
5  import java.io.IOException;
6  import java.io.ObjectInput;
7  import java.io.ObjectInputStream;
8  import java.io.ObjectOutput;
9  import java.io.ObjectOutputStream;
10 import java.io.Serializable;
11 import java.util.UUID;
12
13 import ar.fiuba.taller.common.Constants.COMMAND;
14
15 @SuppressWarnings("serial")
16 public class Command implements Serializable, ISerialize {
17
18     private UUID uuid;
19     private COMMAND command;
20     private String user;
21     private String message;
22     private String timestamp;
23
24     public Command() {
25         this.command = null;
26         this.user = null;
27         this.message = null;
28         this.uuid = null;
29         this.timestamp = null;
30     }
31
32     public Command(String command, String user, String message, UUID uuid,
33         String timestamp) {
34         this.command = Constants.COMMAND_MAP.get(command);
35         this.user = user;
36         this.message = message;
37         this.uuid = uuid;
38         this.timestamp = timestamp;
39     }
40
41     public byte[] serialize() throws IOException {
42         ByteArrayOutputStream os = new ByteArrayOutputStream();
43         ObjectOutput objOut = new ObjectOutputStream(os);
44
45         objOut.writeObject(this);
46         byte byteForm[] = os.toByteArray();
47         objOut.close();
48         os.close();
49         return byteForm;
50     }
51
52     public void deserialize(byte[] byteForm)
53         throws IOException, ClassNotFoundException {
54         ByteArrayInputStream is = new ByteArrayInputStream(byteForm);
55         ObjectInput objIn = new ObjectInputStream(is);
56         Command tmp;
57         tmp = (Command) objIn.readObject();
58         objIn.close();
59         is.close();
60         uuid = tmp.getUuid();
61         command = tmp.getCommand();
62         user = tmp.getUser();
63         message = tmp.getMessage();
64         timestamp = tmp.getTimestamp();
65     }
66 }

```

abr 10, 17 5:27

Command.java

Page 2/2

```

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

```

abr 10, 17 5:26

UserConsole.java

Page 1/2

```

1 package ar.fiuba.taller.ClientConsole;
2
3 import java.io.IOException;
4 import java.util.concurrent.ArrayBlockingQueue;
5 import java.util.concurrent.BlockingQueue;
6 import java.util.concurrent.TimeoutException;
7
8 import org.apache.log4j.Logger;
9 import org.apache.log4j.MDC;
10
11 import ar.fiuba.taller.common.*;
12
13 public class UserConsole implements Runnable {
14     private String username;
15     private BlockingQueue<COMMAND> commandQueue;
16     private BlockingQueue<Response> responseQueue;
17     private Thread scriptReaderThread;
18     private Thread commandControllerThread;
19     private Thread eventViewerThread;
20     private Thread responseControllerThread;
21     private RemoteQueue remoteUserResponseQueue;
22     private RemoteQueue dispatcherQueue;
23     private ConfigLoader configLoader = ConfigLoader.getInstance();
24     private String mode;
25     final static Logger logger = Logger.getLogger(UserConsole.class);
26
27     public UserConsole(String username) {
28         this.username = username;
29     }
30
31     public void run() {
32         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
33
34         try {
35             logger.info("Iniciando usuario.");
36             logger.info("Cargando la configuracion");
37             configLoader.init(Constants.CONF_FILE);
38             initUser();
39             startUser();
40             terminateUser();
41         } catch (InterruptedException e) {
42             logger.error("Error al joinear los threads");
43             logger.info(e.toString());
44             e.printStackTrace();
45         } catch (IOException e) {
46             logger.error(
47                 "Error al cargar la configuracion o crear las colas remotas");
48             logger.info(e.toString());
49             e.printStackTrace();
50         } catch (TimeoutException e) {
51             // TODO Auto-generated catch block
52             e.printStackTrace();
53         }
54     }
55
56     private void initUser() throws IOException, TimeoutException {
57         logger.info("Creando cola de comandos leidos");
58         commandQueue = new ArrayBlockingQueue<COMMAND>(
59             Constants.COMMAND_QUEUE_SIZE);
60         logger.info("Creando cola del dispatcher");
61         dispatcherQueue = new RemoteQueue(configLoader.getDispatcherQueueName(),
62             configLoader.getDispatcherQueueHost());
63         dispatcherQueue.init();
64         logger.info("Creando lector de scripts");
65         scriptReaderThread = new Thread(
66             new ScriptReader(commandQueue,

```

abr 10, 17 5:26

UserConsole.java

Page 2/2

```

67         Constants.COMMAND_SCRIPT_FOLDER + "/" + username
68         + Constants.COMMAND_SCRIPT_EXTENSION,
69         username));
70     logger.info("Creando controlador de comandos");
71     commandControllerThread = new Thread(
72         new CommandController(commandQueue, dispatcherQueue));
73     logger.info("Creando cola de respuestas");
74     responseQueue = new ArrayBlockingQueue<Response>(
75         Constants.RESPONSE_QUEUE_SIZE);
76     logger.info("Creando cola remota de respuestas");
77     remoteUserResponseQueue = new RemoteQueue(username,
78         configLoader.getResponseQueueHost());
79     remoteUserResponseQueue.init();
80     logger.info("Creando el controlador de respuestas");
81     responseControllerThread = new Thread(
82         new ResponseController(responseQueue, remoteUserResponseQueue));
83     logger.info("Creando el visor de eventos");
84     eventViewerThread = new Thread(new EventViewer(responseQueue, username,
85         Constants.LOGS_DIR + "/" + username
86         + Constants.EVENT_VIEWER_FILE_EXTENSION));
87 }
88
89 private void startUser() {
90     logger.info("Iniciando el lector de scripts");
91     scriptReaderThread.start();
92     logger.info("Iniciando el controlador de comandos");
93     commandControllerThread.start();
94     logger.info("Iniciando el controlador de respuestas");
95     responseControllerThread.start();
96     logger.info("Iniciando el visor de eventos");
97     eventViewerThread.start();
98 }
99
100 private void terminateUser()
101     throws InterruptedException, IOException, TimeoutException {
102     logger.info("Esperando al controlador de comandos");
103     commandControllerThread.join();
104     logger.info("controller finalizado!");
105     logger.info("Esperando al reader");
106     scriptReaderThread.join();
107     logger.info("Reader finalizado!");
108     logger.info("Esperando al controlador de respuestas");
109     responseControllerThread.join();
110     logger.info("controller controlador de respuestas!");
111     logger.info("Esperando al visor de eventos");
112     eventViewerThread.join();
113     logger.info("visor de eventos finalizado!");
114 }
115 }

```

abr 10, 17 5:26

ScriptReader.java

Page 1/2

```

1  package ar.fiuba.taller.ClientConsole;
2
3  import java.io.FileNotFoundException;
4  import java.io.FileReader;
5  import java.io.IOException;
6  import java.util.Iterator;
7  import java.util.concurrent.BlockingQueue;
8
9  import org.apache.log4j.Logger;
10 import org.apache.log4j.MDC;
11 import org.json.simple.JSONArray;
12 import org.json.simple.JSONObject;
13 import org.json.simple.parser.JSONParser;
14 import org.json.simple.parser.ParseException;
15
16 import ar.fiuba.taller.common.Command;
17 import ar.fiuba.taller.common.Constants;
18
19 public class ScriptReader implements Runnable {
20
21     final static Logger logger = Logger.getLogger(ScriptReader.class);
22
23     BlockingQueue<Command> commandQueue;
24     String commandScript;
25     String username;
26
27     public ScriptReader(BlockingQueue<Command> commandQueue,
28         String commandScript, String username) {
29         this.commandQueue = commandQueue;
30         this.commandScript = commandScript;
31         this.username = username;
32     }
33
34     public void run() {
35         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
36         logger.info("Iniciando el script reader");
37         try {
38             JSONParser parser = new JSONParser();
39             Object obj = parser.parse(new FileReader(commandScript));
40             JSONObject jsonObject = (JSONObject) obj;
41             JSONArray commandArray = (JSONArray) jsonObject
42                 .get(Constants.COMMAND_ARRAY);
43             Iterator<JSONObject> iterator = commandArray.iterator();
44             JSONObject commandObject;
45             Command command;
46
47             logger.info("Leyendo el command script: " + commandScript);
48             while (iterator.hasNext()) {
49                 commandObject = iterator.next();
50                 command = new Command(
51                     (String) commandObject.get(Constants.COMMAND_KEY),
52                     username,
53                     (String) commandObject.get(Constants.MESSAGE_KEY), null,
54                     null);
55                 logger.info("Se inserto comando con los siguientes parametros: "
56                     + "\nUsuario: " + command.getUser() + "\nComando: "
57                     + command.getCommand() + "\nMensaje: "
58                     + command.getMessage());
59                 commandQueue.put(command);
60             }
61         } catch (InterruptedException e) {
62             logger.error("Error al pushear comandos en la cola");
63             logger.info(e.toString());
64             e.printStackTrace();
65         } catch (FileNotFoundException e) {
66             logger.error("No se encontro el archivo de comandos");

```

abr 10, 17 5:26

ScriptReader.java

Page 2/2

```

67     logger.info(e.toString());
68     e.printStackTrace();
69 } catch (IOException e) {
70     logger.error("Error al leer el archivo de comandos");
71     logger.info(e.toString());
72     e.printStackTrace();
73 } catch (ParseException e) {
74     logger.error("Error al parsear el archivo de comandos");
75     logger.info(e.toString());
76     e.printStackTrace();
77 }
78 }
79 }

```

abr 10, 17 5:26

ResponseController.java

Page 1/2

```

1  package ar.fiuba.taller.ClientConsole;
2
3  import java.io.IOException;
4  import java.util.concurrent.BlockingQueue;
5  import org.apache.log4j.Logger;
6  import org.apache.log4j.MDC;
7
8  import com.rabbitmq.client.AMQP.BasicProperties;
9  import com.rabbitmq.client.Channel;
10 import com.rabbitmq.client.DefaultConsumer;
11 import com.rabbitmq.client.Envelope;
12
13 import ar.fiuba.taller.common.RemoteQueue;
14 import ar.fiuba.taller.common.Response;
15
16 public class ResponseController extends DefaultConsumer implements Runnable {
17
18     private BlockingQueue<Response> responseQueue;
19     private RemoteQueue remoteResponseQueue;
20     final static Logger logger = Logger.getLogger(ResponseController.class);
21
22     public ResponseController(BlockingQueue<Response> responseQueue,
23                             RemoteQueue remoteResponseQueue) {
24         super(remoteResponseQueue.getChannel());
25         this.responseQueue = responseQueue;
26         this.remoteResponseQueue = remoteResponseQueue;
27     }
28
29     @Override
30     public void handleDelivery(String consumerTag, Envelope envelope,
31                               BasicProperties properties, byte[] body) throws IOException {
32         super.handleDelivery(consumerTag, envelope, properties, body);
33         Response response = new Response();
34         try {
35             response.deserialize(body);
36             logger.info("Respuesta recibida con los siguientes valores: "
37                       + "\nUUID:" + response.getUuid() + "\nStatus:"
38                       + response.getResponse_status() + "\nMensaje:"
39                       + response.getMessage());
40             responseQueue.put(response);
41             logger.info("Respuesta pusheada en la cola responseQueue");
42         } catch (ClassNotFoundException e) {
43             logger.error("Error al deserializar la respuesta");
44             logger.info(e.toString());
45             e.printStackTrace();
46         } catch (IOException e) {
47             logger.error("Error al deserializar la respuesta");
48             logger.info(e.toString());
49             e.printStackTrace();
50         } catch (InterruptedException e) {
51             logger.error(
52                 "Error al insertar la respuesta en la cola responseQueue");
53             logger.info(e.toString());
54             e.printStackTrace();
55         }
56     }
57
58     public void run() {
59         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
60         logger.info("Iniciando el response controller");
61         try {
62             remoteResponseQueue.getChannel().basicConsume(
63                 remoteResponseQueue.getQueueName(), true, this);
64         } catch (IOException e) {
65             // TODO Auto-generated catch block
66             e.printStackTrace();

```


abr 10, 17 5:26

ResponseController.java

Page 2/2

```

67     }
68   }
69
70 }

```

abr 10, 17 5:27

EventViewer.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.Constants;
13 import ar.fiuba.taller.common.Response;
14
15 public class EventViewer implements Runnable {
16     BlockingQueue<Response> responseQueue;
17     String username;
18     String eventFile;
19     final static Logger logger = Logger.getLogger(EventViewer.class);
20
21     public EventViewer(BlockingQueue<Response> responseQueue, String username,
22         String eventFile) {
23         this.responseQueue = responseQueue;
24         this.username = username;
25         this.eventFile = eventFile;
26     }
27
28     public void run() {
29         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
30         Response response = null;
31         FileWriter responseFile = null;
32         PrintWriter pw;
33
34         logger.info("Iniciando el event viewer");
35         try {
36             while (true) {
37                 logger.info("Esperando respuesta");
38                 response = responseQueue.take();
39                 pw = new PrintWriter(
40                     new BufferedWriter(new FileWriter(eventFile, true)));
41                 logger.info("Respuesta obtenida");
42                 pw.println("Evento recibido:\nUUID: " + response.getUuid()
43                     + "\nResponse Status: " + response.getResponse_status()
44                     + "\nMessage: " + response.getMessage());
45                 pw.println(
46                     "-----"
47                     + "-----");
48                 pw.close();
49             }
50         } catch (InterruptedException e) {
51             logger.error(
52                 "Error al tomar la respuesta en la cola responseQueue");
53             logger.info(e.toString());
54             e.printStackTrace();
55         } catch (IOException e) {
56             logger.error("Error al abrir el archivo " + eventFile);
57             logger.info(e.toString());
58             e.printStackTrace();
59         } finally {
60             try {
61                 if (null != responseFile)
62                     responseFile.close();
63             } catch (Exception e2) {
64                 logger.error("Error al cerrar el archivo " + eventFile);
65                 logger.info(e2.toString());
66                 e2.printStackTrace();

```

abr 10, 17 5:27

EventViewer.java

Page 2/2

```

67     }
68     }
69
70     }
71
72     }

```

abr 10, 17 5:26

CommandController.java

Page 1/1

```

1  package ar.fiuba.taller.ClientConsole;
2
3  import java.io.IOException;
4  import java.lang.invoke.ConstantCallSite;
5  import java.sql.Timestamp;
6  import java.util.UUID;
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.Command;
13 import ar.fiuba.taller.common.Constants;
14 import ar.fiuba.taller.common.RemoteQueue;
15
16 public class CommandController implements Runnable {
17     private BlockingQueue<Command> commandQueue;
18     private RemoteQueue dispatcherQueue;
19     Timestamp timestamp;
20
21     final static Logger logger = Logger.getLogger(CommandController.class);
22
23     public CommandController(BlockingQueue<Command> commandQueue,
24                             RemoteQueue dispatcherQueue) {
25         this.commandQueue = commandQueue;
26         this.dispatcherQueue = dispatcherQueue;
27     }
28
29     public void run() {
30         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
31         Command command;
32
33         logger.info("Iniciando el command controller");
34
35         try {
36             logger.info("Obteniendo comando de la cola");
37             command = commandQueue.take();
38             logger.info("Comando obtenido");
39             logger.info("Comando recibido: " + command.getCommand());
40             logger.info("Mensaje: " + command.getMessage());
41             if (command.getMessage().length() ≤ 141) {
42                 logger.info("Generando UUID");
43                 command.setUuid(UUID.randomUUID());
44                 logger.info("Generando timestamp");
45                 timestamp = new Timestamp(System.currentTimeMillis());
46                 command.setTimestamp(Constants.SDF.format(timestamp));
47                 logger.info("UUID generado: " + command.getUuid());
48                 logger.info("Enviando el mensaje al dispatcher");
49                 dispatcherQueue.put(command);
50                 logger.info("Mensaje enviado");
51             } else {
52                 logger.error("El mensaje contiene mas de 141 caracteres");
53             }
54         } catch (InterruptedException e) {
55             logger.error("Error al sacar un comando de la cola commandQueue");
56             logger.info(e.toString());
57             e.printStackTrace();
58         } catch (IOException e) {
59             logger.error("Error al enviar el mensaje al dispatcher");
60             logger.info(e.toString());
61             e.printStackTrace();
62         }
63     }
64 }

```

abr 10, 17 5:26

App.java

Page 1/2

```

1 package ar.fiuba.taller.ClientConsole;
2
3 import java.io.FileNotFoundException;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.util.ArrayList;
7 import java.util.Iterator;
8 import java.util.List;
9 import java.util.StringTokenizer;
10
11 import org.apache.log4j.Logger;
12 import org.apache.log4j.MDC;
13 import org.apache.log4j.PropertyConfigurator;
14 import org.json.simple.JSONArray;
15 import org.json.simple.JSONObject;
16 import org.json.simple.parser.JSONParser;
17 import org.json.simple.parser.ParseException;
18
19 import ar.fiuba.taller.common.Constants;
20
21 public class App {
22     final static Logger logger = Logger.getLogger(App.class);
23
24     public static void main(String[] args) {
25         PropertyConfigurator.configure(Constants.LOGGER_CONF);
26         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
27         List<Thread> usersList = new ArrayList<Thread>();
28         String username, mode, reg;
29
30         logger.info("Se inicia una nueva instancia de ClientConsole");
31
32         try {
33             // Obtengo los usuarios y los pongo a ejecutar el script
34             JSONParser parser = new JSONParser();
35             Object obj = parser.parse(new FileReader(Constants.USERS_FILE));
36             JSONObject jsonObject = (JSONObject) obj;
37             JSONArray arr = (JSONArray) jsonObject.get(Constants.USERS_KEY);
38             Thread userConsoleThread;
39             logger.info("Leyendo el archivo de usuarios a simular");
40             Iterator<String> iterator = arr.iterator();
41             StringTokenizer st;
42             while (iterator.hasNext()) {
43                 username = iterator.next();
44                 logger.info("Siguiente usuario a crear: " + username);
45                 userConsoleThread = new Thread(new UserConsole(username));
46                 userConsoleThread.start();
47                 usersList.add(userConsoleThread);
48                 logger.info(
49                     "Usuario " + userConsoleThread.getId() + " creado!");
50             }
51
52             // Espero a que los usuarios hayan terminado de ejecutar
53             logger.info("Esperando a que los usuarios terminen: "
54                 + usersList.size());
55             for (Thread userThread : usersList) {
56                 userThread.join();
57                 logger.info("Usuario " + userThread.getId() + " finalizado!");
58             }
59             } catch (InterruptedException e) {
60                 logger.error("Error al joinear los threads de usuarios");
61                 logger.info(e.toString());
62                 e.printStackTrace();
63             } catch (FileNotFoundException e) {
64                 logger.error("No se encontro el archivo de usuarios");
65                 logger.info(e.toString());
66                 e.printStackTrace();

```

abr 10, 17 5:26

App.java

Page 2/2

```

67         } catch (IOException e) {
68             logger.error("Error al leer el archivo de usuarios");
69             logger.info(e.toString());
70             e.printStackTrace();
71         } catch (ParseException e) {
72             logger.error("Error al parsear el archivo de usuarios");
73             logger.info(e.toString());
74             e.printStackTrace();
75         }
76     }
77 }

```

abr 10, 17 5:25

AppTest.java

Page 1/1

```

1 package ar.fiuba.taller.auditLogger;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8  * Unit test for simple App.
9  */
10 public class AppTest extends TestCase {
11     /**
12      * Create the test case
13      *
14      * @param testName
15      *      name of the test case
16      */
17     public AppTest(String testName) {
18         super(testName);
19     }
20
21     /**
22      * @return the suite of tests being tested
23      */
24     public static Test suite() {
25         return new TestSuite(AppTest.class);
26     }
27
28     /**
29      * Rigourous Test :-))
30      */
31     public void testApp() {
32         assertTrue(true);
33     }
34 }

```

abr 10, 17 5:25

AuditLogger.java

Page 1/2

```

1 package ar.fiuba.taller.auditLogger;
2
3 import java.io.BufferedWriter;
4 import java.io.FileWriter;
5 import java.io.IOException;
6 import java.io.PrintWriter;
7 import java.sql.Timestamp;
8 import java.text.DateFormat;
9 import java.text.SimpleDateFormat;
10 import java.util.Date;
11 import java.util.concurrent.BlockingQueue;
12
13 import org.apache.log4j.Logger;
14 import org.apache.log4j.MDC;
15
16 import com.rabbitmq.client.DefaultConsumer;
17 import com.rabbitmq.client.Envelope;
18 import com.rabbitmq.client.AMQP.BasicProperties;
19
20 import ar.fiuba.taller.common.*;
21
22 public class AuditLogger extends DefaultConsumer implements Runnable {
23
24     private Timestamp timestamp;
25     private RemoteQueue loggerQueue;
26     final static Logger logger = Logger.getLogger(AuditLogger.class);
27
28     public AuditLogger(RemoteQueue loggerQueue) {
29         super(loggerQueue.getChannel());
30         ConfigLoader.getInstance();
31         this.loggerQueue = loggerQueue;
32     }
33
34     public void run() {
35         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
36
37         logger.info("Iniciando el audit logger");
38         try {
39             PrintWriter pw = new PrintWriter(Constants.AUDIT_LOG_FILE, "UTF-8");
40             pw.close();
41             loggerQueue.getChannel().basicConsume(loggerQueue.getQueueName(),
42                 true, this);
43         } catch (IOException e) {
44             logger.error("Error consumir de la cola remota");
45             logger.info(e.toString());
46             e.printStackTrace();
47         }
48     }
49
50     @Override
51     public void handleDelivery(String consumerTag, Envelope envelope,
52         BasicProperties properties, byte[] body) throws IOException {
53         super.handleDelivery(consumerTag, envelope, properties, body);
54         PrintWriter pw = new PrintWriter(new BufferedWriter(
55             new FileWriter(Constants.AUDIT_LOG_FILE, true)));
56         Command command = new Command();
57         try {
58             command.deserialize(body);
59             logger.info("Comando recibido con los siguientes parametros: "
60                 + "\nUsuario: " + command.getUser() + "\nComando: "
61                 + command.getCommand() + "\nMensaje: "
62                 + command.getMessage());
63             logger.info("Escribiendo el mensaje en el archivo de log "
64                 + Constants.AUDIT_LOG_FILE);
65             logger.info(getAuditLogEntry(command));
66             pw.println(getAuditLogEntry(command));

```

abr 10, 17 5:25

AuditLogger.java

Page 2/2

```

67     pw.close();
68     } catch (ClassNotFoundException e) {
69         logger.error("Error al deserializar el comando");
70         logger.info(e.toString());
71         e.printStackTrace();
72     } catch (IOException e) {
73         logger.error("Error al deserializar el comando");
74         logger.info(e.toString());
75         e.printStackTrace();
76     }
77 }
78
79 private String getAuditLogEntry(Command command) {
80     timestamp = new Timestamp(System.currentTimeMillis());
81     return Constants.SDF.format(timestamp) + "-" + "UUID: "
82         + command.getUuid() + "-Usuario: " + command.getUser()
83         + "-Comando: " + command.getCommand() + "-Mensaje: "
84         + command.getMessage();
85 }
86
87 }

```

abr 10, 17 5:25

App.java

Page 1/1

```

1  package ar.fiuba.taller.auditLogger;
2
3  import java.io.IOException;
4  import java.util.concurrent.TimeoutException;
5
6  import org.apache.log4j.Logger;
7  import org.apache.log4j.MDC;
8  import org.apache.log4j.PropertyConfigurator;
9
10 import ar.fiuba.taller.common.ConfigLoader;
11 import ar.fiuba.taller.common.Constants;
12 import ar.fiuba.taller.common.RemoteQueue;
13
14 public class App {
15     final static Logger logger = Logger.getLogger(App.class);
16
17     public static void main(String[] args) throws Exception {
18         PropertyConfigurator.configure(Constants.LOGGER_CONF);
19         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
20         try {
21             ConfigLoader.getInstance().init(Constants.CONF_FILE);
22             logger.info("Conectando a la cola remota loggerQueue");
23             RemoteQueue loggerQueue = new RemoteQueue(
24                 ConfigLoader.getInstance().getAuditLoggerQueueName(),
25                 ConfigLoader.getInstance().getAuditLoggerQueueHost());
26             loggerQueue.init();
27             Thread auditLoggerThread = new Thread(new AuditLogger(loggerQueue));
28             logger.info("Disparando el audit logger");
29             auditLoggerThread.start();
30             auditLoggerThread.join();
31         } catch (InterruptedException e) {
32             logger.error("Error al joinear el audit logger");
33             logger.info(e.toString());
34             e.printStackTrace();
35         } catch (IOException e) {
36             logger.error("Error al cargar la configuracion");
37             logger.info(e.toString());
38             e.printStackTrace();
39         } catch (TimeoutException e) {
40             logger.error("Error iniciar la cola remota");
41             logger.info(e.toString());
42             e.printStackTrace();
43             throw new Exception();
44         }
45     }
46 }

```

abr 10, 17 5:24

AppTest.java

Page 1/1

```

1 package ar.fiuba.taller.analyzer;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8  * Unit test for simple App.
9  */
10 public class AppTest extends TestCase {
11     /**
12      * Create the test case
13      *
14      * @param testName
15      *      name of the test case
16      */
17     public AppTest(String testName) {
18         super(testName);
19     }
20
21     /**
22      * @return the suite of tests being tested
23      */
24     public static Test suite() {
25         return new TestSuite(AppTest.class);
26     }
27
28     /**
29      * Rigourous Test :-)
30      */
31     public void testApp() {
32         assertTrue(true);
33     }
34 }

```

abr 10, 17 5:24

UserRegistry.java

Page 1/3

```

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

```

abr 10, 17 5:24

UserRegistry.java

Page 2/3

```

67     JSONArray ar2 = new JSONArray();
68     ar2.add(follower);
69     jsonObject.put(updateKey, ar2);
70 } else {
71     array.add(follower);
72     jsonObject.put(updateKey, array);
73 }
74 file = new FileWriter(tmpFile);
75 try {
76     file.write(jsonObject.toJSONString());
77 } catch (Exception e) {
78     logger.error("Error al guardar el index");
79     logger.info(e.toString());
80     e.printStackTrace();
81 } finally {
82     file.flush();
83     file.close();
84 }
85 }
86
87 public List<String> getUserFollowers(String followed)
88     throws FileNotFoundException, IOException, ParseException {
89     String usersFile = Constants.DB_DIR + "/" + Constants.DB_USER_INDEX;
90     JSONParser parser = new JSONParser();
91     Object obj;
92     JSONObject jsonObject;
93
94     logger.info("Buscando followers del usuario");
95
96     File tmpFile = new File(usersFile);
97     if (tmpFile.createNewFile()) {
98         FileOutputStream oFile = new FileOutputStream(tmpFile, false);
99         oFile.write("{}".getBytes());
100     }
101     obj = parser.parse(new FileReader(usersFile));
102     jsonObject = (JSONObject) obj;
103     JSONArray array = (JSONArray) jsonObject.get(followed);
104
105     System.out.println(array.toJSONString());
106
107     return array;
108 }
109
110 public List<String> getHashtagFollowers(String followed)
111     throws FileNotFoundException, IOException, ParseException {
112     String hashtagFile = Constants.DB_DIR + "/"
113         + Constants.DB_HASHTAG_INDEX;
114     List<String> followersList = new ArrayList<String>();
115     JSONParser parser = new JSONParser();
116     Object obj;
117     JSONObject jsonObject;
118     JSONArray jsonArray;
119     Iterator<String> it;
120     String word;
121
122     logger.info("Buscando followers del hashtag");
123
124     File tmpFile = new File(hashtagFile);
125     if (tmpFile.createNewFile()) {
126         FileOutputStream oFile = new FileOutputStream(tmpFile, false);
127         oFile.write("{}".getBytes());
128     }
129     logger.info("Obteniendo hashtags de " + followed);
130     obj = parser.parse(new FileReader(hashtagFile));
131     jsonObject = (JSONObject) obj;
132     String regexPattern = "(#\\w+)";

```

abr 10, 17 5:24

UserRegistry.java

Page 3/3

```

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

```

abr 10, 17 5:24

App.java

Page 1/1

```

1 package ar.fiuba.taller.analyzer;
2
3 import org.apache.log4j.Logger;
4 import org.apache.log4j.MDC;
5 import org.apache.log4j.PropertyConfigurator;
6
7 import ar.fiuba.taller.common.Constants;
8
9 public class App {
10     final static Logger logger = Logger.getLogger(App.class);
11
12     public static void main(String[] args) {
13         PropertyConfigurator.configure(Constants.LOGGER_CONF);
14         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
15         try {
16             logger.info("Comenzando el analyzer");
17             Thread analyzerThread = new Thread(new Analyzer());
18             analyzerThread.start();
19             analyzerThread.join();
20         } catch (InterruptedException e) {
21             logger.error("Error al joinar el analyzer");
22             logger.info(e.toString());
23             e.printStackTrace();
24         }
25     }
26 }

```

abr 10, 17 5:25

AnalyzerReciver.java

Page 1/2

```

1 package ar.fiuba.taller.analyzer;
2
3 import java.io.IOException;
4 import java.util.concurrent.BlockingQueue;
5
6 import org.apache.log4j.Logger;
7 import org.apache.log4j.MDC;
8 import org.json.simple.parser.ParseException;
9
10 import com.rabbitmq.client.DefaultConsumer;
11 import com.rabbitmq.client.Envelope;
12 import com.rabbitmq.client.AMQP.BasicProperties;
13
14 import ar.fiuba.taller.common.Command;
15 import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
16 import ar.fiuba.taller.common.RemoteQueue;
17 import ar.fiuba.taller.common.Response;
18
19 public class AnalyzerReciver extends DefaultConsumer implements Runnable {
20
21     BlockingQueue<Response> responseQueue;
22     Command command;
23     Response response;
24     UserRegistry userRegistry;
25     RemoteQueue analyzerQueue;
26     final static Logger logger = Logger.getLogger(AnalyzerReciver.class);
27
28     public AnalyzerReciver(BlockingQueue<Response> responseQueue,
29         RemoteQueue analyzerQueue, UserRegistry userRegistry) {
30         super(analyzerQueue.getChannel());
31         this.responseQueue = responseQueue;
32         this.userRegistry = userRegistry;
33         this.analyzerQueue = analyzerQueue;
34     }
35
36     public void run() {
37         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
38         logger.info("Iniciando el analyzer reciver");
39         logger.info("Me pongo a comer de la cola: " + analyzerQueue.getHost()
40             + " " + analyzerQueue.getQueueName());
41         try {
42             analyzerQueue.getChannel()
43                 .basicConsume(analyzerQueue.getQueueName(), true, this);
44         } catch (IOException e) {
45             logger.error("Error al comer de la cola");
46             logger.info(e.toString());
47             e.printStackTrace();
48         }
49     }
50
51     @Override
52     public void handleDelivery(String consumerTag, Envelope envelope,
53         BasicProperties properties, byte[] body) throws IOException {
54         super.handleDelivery(consumerTag, envelope, properties, body);
55         Command command = new Command();
56         try {
57             command.deserialize(body);
58             logger.info("Comando recibido con los siguientes parametros: "
59                 + "\nUUID: " + command.getUuid() + "\nUsuario: "
60                 + command.getUser() + "\nComando: " + command.getCommand()
61                 + "\nMensaje: " + command.getMessage());
62
63             switch (command.getCommand()) {
64                 case PUBLISH:
65                     logger.info(
66                         "Comando recibido: PUBLISH. Insertando en la cola del "

```


abr 10, 17 5:25

AnalyzerReciver.java

Page 2/2

```

67         + "analyzer dispatcher.");
68     response = new Response();
69     response.setUuid(command.getUuid());
70     response.setUser(command.getUser());
71     // Puede ser que de error en caso de hacer el update, entonces
72     // hay que
73     // mandarle error al usuario
74     response.setResponse_status(RESPONSE_STATUS.OK);
75     response.setMessage(command.getTimestamp() + "\n"
76         + command.getUser() + "\n" + command.getMessage());
77     responseQueue.put(response);
78     break;
79 case FOLLOW:
80     logger.info(
81         "Comando recibido: FOLLOW. Actualizando el user "
82         + "registry.");
83     userRegistry.update(command.getUser(), command.getMessage());
84     response = new Response();
85     response.setUuid(command.getUuid());
86     response.setUser(command.getUser());
87     response.setResponse_status(RESPONSE_STATUS.REGISTERED);
88     response.setMessage("Seguidor registrado");
89     responseQueue.put(response);
90     break;
91 default:
92     logger.info("Comando recibido invalido. Comando descartado.");
93 }
94 } catch (ClassNotFoundException e) {
95     logger.error("Error al deserializar el comando");
96     logger.info(e.toString());
97     e.printStackTrace();
98 } catch (IOException e) {
99     logger.error("Error al deserializar el comando");
100    logger.info(e.toString());
101    e.printStackTrace();
102 } catch (InterruptedException e) {
103    logger.error("Error al insertar el comando en alguna de las colas");
104    logger.info(e.toString());
105    e.printStackTrace();
106 } catch (ParseException e) {
107    logger.error("Error al actualizar la base de usuarios");
108    logger.info(e.toString());
109    e.printStackTrace();
110 }
111 }
112 }
113 }

```

abr 10, 17 5:24

Analyzer.java

Page 1/2

```

1 package ar.fiuba.taller.analyzer;
2
3 import java.io.IOException;
4 import java.util.concurrent.ArrayBlockingQueue;
5 import java.util.concurrent.BlockingQueue;
6 import java.util.concurrent.TimeoutException;
7
8 import org.apache.log4j.Logger;
9 import org.apache.log4j.MDC;
10
11 import ar.fiuba.taller.common.ConfigLoader;
12 import ar.fiuba.taller.common.Constants;
13 import ar.fiuba.taller.common.RemoteQueue;
14 import ar.fiuba.taller.common.Response;
15
16 public class Analyzer implements Runnable {
17
18     private Thread analyzerDispatcherThread;
19     private Thread analyzerReciverThread;
20     private Thread responseControllerThread;
21     private BlockingQueue<Response> responseQueue;
22     private UserRegistry userRegistry;
23     private ConfigLoader configLoader;
24     private RemoteQueue analyzerQueue;
25     final static Logger logger = Logger.getLogger(Analyzer.class);
26
27     public Analyzer() {
28         configLoader = ConfigLoader.getInstance();
29     }
30
31     public void run() {
32         MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
33         try {
34             configLoader.init(Constants.CONF_FILE);
35             // Instancio la cola
36             responseQueue = new ArrayBlockingQueue<Response>(
37                 Constants.COMMAND_QUEUE_SIZE);
38             logger.debug(Constants.ANALYZER_QUEUE_NAME);
39             logger.debug(Constants.ANALYZER_QUEUE_HOST);
40             // Creo la cola remota en donde el analyzer recibe los comandos
41             analyzerQueue = new RemoteQueue(
42                 ConfigLoader.getInstance().getAnalyzerQueueName(),
43                 ConfigLoader.getInstance().getAnalyzerQueueHost());
44             analyzerQueue.init();
45
46             // Instancio el registry
47             userRegistry = new UserRegistry();
48
49             // Hago una carga inicial del user registry
50
51             // Instancio los threads
52             analyzerReciverThread = new Thread(new AnalyzerReciver(
53                 responseQueue, analyzerQueue, userRegistry));
54             analyzerDispatcherThread = new Thread(
55                 new AnalyzerDispatcher(responseQueue, userRegistry));
56
57             // Inicio los threads
58             analyzerReciverThread.start();
59             analyzerDispatcherThread.start();
60
61             // Me quedo esperando los threads
62             analyzerReciverThread.join();
63             analyzerDispatcherThread.join();
64
65         } catch (IOException e) {
66             logger.error("Error al cargar el archivo de configuracion");
67         }
68     }
69 }

```

abr 10, 17 5:24

Analyzer.java

Page 2/2

```

67     logger.info(e.toString());
68     e.printStackTrace();
69 } catch (InterruptedException e) {
70     logger.error("Error al dormir el thread");
71     logger.info(e.toString());
72     e.printStackTrace();
73 } catch (TimeoutException e) {
74     logger.error("Error al iniciar la cola remota");
75     logger.info(e.toString());
76     e.printStackTrace();
77 }
78 }
79
80 }
```

abr 10, 17 5:24

AnalyzerDispatcher.java

Page 1/2

```

1  package ar.fiuba.taller.analyzer;
2
3  import java.io.IOException;
4  import java.util.HashMap;
5  import java.util.HashSet;
6  import java.util.Iterator;
7  import java.util.List;
8  import java.util.Map;
9  import java.util.Set;
10 import java.util.concurrent.BlockingQueue;
11 import java.util.concurrent.TimeoutException;
12
13 import org.apache.log4j.Logger;
14 import org.json.simple.parser.ParseException;
15
16 import ar.fiuba.taller.common.RemoteQueue;
17 import ar.fiuba.taller.common.Response;
18 import ar.fiuba.taller.common.ConfigLoader;
19 import ar.fiuba.taller.common.Constants.RESPONSE_STATUS;
20
21 public class AnalyzerDispatcher implements Runnable {
22
23     BlockingQueue<Response> responseQueue;
24     Response response;
25     Map<String, RemoteQueue> usersMap;
26     RemoteQueue remoteQueue;
27     UserRegistry userRegistry;
28     List<String> userFollowers;
29     List<String> hashtagFollowers;
30     Set<String> usersSet;
31     final static Logger logger = Logger.getLogger(AnalyzerDispatcher.class);
32
33     public AnalyzerDispatcher(BlockingQueue<Response> responseQueue,
34                               UserRegistry userRegistry) {
35         this.responseQueue = responseQueue;
36         this.userRegistry = userRegistry;
37         usersMap = new HashMap<String, RemoteQueue>();
38     }
39
40     public void run() {
41         while (true) {
42             try {
43                 response = responseQueue.take();
44                 logger.info("Nueva respuesta para enviar");
45                 logger.info("Nueva respuesta para enviar");
46                 logger.info("UUID: " + response.getUuid());
47                 logger.info("User: " + response.getUser());
48                 logger.info("Status: " + response.getResponse_status());
49                 logger.info("Message: " + response.getMessage());
50                 // Reviso si es un user register o un mensaje
51                 // Si da error o es una registracion, se lo devuelvo solamente
52                 // al usuario que envio el request
53                 if (response.getResponse_status() == RESPONSE_STATUS.REGISTERED
54                     || response
55                         .getResponse_status() == RESPONSE_STATUS.ERROR)
56                 {
57                     logger.info("Enviando respuesta");
58                     remoteQueue = getUserQueue(response.getUser());
59                     remoteQueue.put(response);
60                 } else {
61                     // Por Ok, hago anycast a los followers
62                     logger.info("Anycast a los followers");
63                     usersSet = new HashSet<String>();
64                     userFollowers = userRegistry
65                         .getUserFollowers(response.getUser());
66                     hashtagFollowers = userRegistry
```

abr 10, 17 5:24

AnalyzerDispatcher.java

Page 2/2

```

67         .getHashtagFollowers(response.getMessage());
68         for (String follower : userFollowers) {
69             usersSet.add(follower);
70         }
71         for (String follower : hashtagFollowers) {
72             usersSet.add(follower);
73         }
74         // Fowardeo el mensaje a los followers
75         Iterator<String> it = usersSet.iterator();
76         while (it.hasNext()) {
77             (getUserQueue(it.next())).put(response);
78         }
79     }
80 } catch (InterruptedException e) {
81     logger.error(
82         "Error al tomar respuestas de la cola responseQueue");
83     logger.info(e.toString());
84     e.printStackTrace();
85 } catch (IOException e) {
86     logger.error(
87         "Error al insertar respuesta en la cola remota del "
88         + "usuario:" + response.getUser());
89     logger.info(e.toString());
90     e.printStackTrace();
91 } catch (ParseException e) {
92     logger.error("Error al updatear los indices");
93     logger.info(e.toString());
94     e.printStackTrace();
95 } catch (TimeoutException e) {
96     logger.error("No se pudo enviar el mensaje al follower");
97     logger.info(e.toString());
98     e.printStackTrace();
99 }
100 }
101 }
102
103 private RemoteQueue getUserQueue(String username)
104     throws IOException, TimeoutException {
105     RemoteQueue tmpQueue;
106     tmpQueue = usersMap.get(username);
107
108     if (tmpQueue == null) {
109         tmpQueue = new RemoteQueue(username,
110             ConfigLoader.getInstance().getUsersServer());
111         tmpQueue.init();
112         usersMap.put(username, tmpQueue);
113     }
114     return usersMap.get(username);
115 }
116 }

```

abr 10, 17 7:12

Table of Content

Page 1/1

1	Table of Contents				
2	1 AppTest.java.....	sheets	1 to 1 (1)	pages	1- 1 35 lines
3	2 Storage.java.....	sheets	1 to 4 (4)	pages	2- 8 422 lines
4	3 StorageController.java	sheets	5 to 6 (2)	pages	9- 11 142 lines
5	4 ResponseController.java	sheets	6 to 7 (2)	pages	12- 13 71 lines
6	5 RemoveController.java	sheets	7 to 8 (2)	pages	14- 15 73 lines
7	6 QueryController.java	sheets	8 to 9 (2)	pages	16- 17 77 lines
8	7 CreateController.java	sheets	9 to 10 (2)	pages	18- 19 76 lines
9	8 App.java.....	sheets	10 to 10 (1)	pages	20- 20 49 lines
10	9 AppTest.java.....	sheets	11 to 11 (1)	pages	21- 21 35 lines
11	10 StorageController.java	sheets	11 to 11 (1)	pages	22- 22 53 lines
12	11 LoggerController.java	sheets	12 to 12 (1)	pages	23- 23 54 lines
13	12 Dispatcher.java.....	sheets	12 to 13 (2)	pages	24- 25 125 lines
14	13 DispatcherController.java	sheets	13 to 14 (2)	pages	26- 27 108 lines
15	14 App.java.....	sheets	14 to 14 (1)	pages	28- 28 27 lines
16	15 AnalyzerController.java	sheets	15 to 15 (1)	pages	29- 29 53 lines
17	16 AppTest.java.....	sheets	15 to 15 (1)	pages	30- 30 39 lines
18	17 App.java.....	sheets	16 to 16 (1)	pages	31- 31 14 lines
19	18 AppTest.java.....	sheets	16 to 16 (1)	pages	32- 32 35 lines
20	19 Response.java.....	sheets	17 to 17 (1)	pages	33- 34 93 lines
21	20 RemoteQueue.java....	sheets	18 to 18 (1)	pages	35- 35 58 lines
22	21 ISerialize.java.....	sheets	18 to 18 (1)	pages	36- 36 13 lines
23	22 Constants.java.....	sheets	19 to 19 (1)	pages	37- 38 81 lines
24	23 ConfigLoader.java....	sheets	20 to 21 (2)	pages	39- 41 143 lines
25	24 Command.java.....	sheets	21 to 22 (2)	pages	42- 43 119 lines
26	25 AppTest.java.....	sheets	22 to 22 (1)	pages	43- 43 1 lines
27	26 UserConsole.java....	sheets	22 to 23 (2)	pages	44- 45 116 lines
28	27 ScriptReader.java....	sheets	23 to 24 (2)	pages	46- 47 80 lines
29	28 ResponseController.java	sheets	24 to 25 (2)	pages	48- 49 71 lines
30	29 EventViewer.java....	sheets	25 to 26 (2)	pages	50- 51 73 lines
31	30 CommandController.java	sheets	26 to 26 (1)	pages	52- 52 65 lines
32	31 App.java.....	sheets	27 to 27 (1)	pages	53- 54 78 lines
33	32 AppTest.java.....	sheets	28 to 28 (1)	pages	55- 55 35 lines
34	33 AuditLogger.java....	sheets	28 to 29 (2)	pages	56- 57 88 lines
35	34 App.java.....	sheets	29 to 29 (1)	pages	58- 58 47 lines
36	35 AppTest.java.....	sheets	30 to 30 (1)	pages	59- 59 35 lines
37	36 UserRegistry.java....	sheets	30 to 31 (2)	pages	60- 62 149 lines
38	37 App.java.....	sheets	32 to 32 (1)	pages	63- 63 27 lines
39	38 AnalyzerReciver.java	sheets	32 to 33 (2)	pages	64- 65 114 lines
40	39 Analyzer.java.....	sheets	33 to 34 (2)	pages	66- 67 81 lines
41	40 AnalyzerDispatcher.java	sheets	34 to 35 (2)	pages	68- 69 117 lines