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

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StringReader;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.channels.FileLock;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.ListIterator;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

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

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

public class Storage {

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

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

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

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

```java
    FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ);
    FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
    try {
      ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
      fileChannel.read(buffer);
      buffer.position(0);
      StringBuilder sb = new StringBuilder();
          while (buffer.hasRemaining()) {
              sb.append((char) buffer.get());
          }
          String tmp = sb.toString();
          if((tmp.split("}", -1).length - 1) > 1) {
            tmp = tmp.substring(0, tmp.indexOf("}")+1);
          }
      obj = parser.parse(new StringReader(tmp));
      JSONObject jsonObject = (JSONObject) obj;
      int count = 0;
      String regexPattern = "(#\\w+)";
      Pattern p = Pattern.compile(regexPattern);
      Matcher m = p.matcher(command.getMessage());
      String hashtag;
      while (m.find()) {
        hashtag = m.group(1);
        hashtag = hashtag.substring(1, hashtag.length());
        Long obj2 = (Long) jsonObject.get(hashtag);
        if (obj2 ≡ null) {
          // La entrada no existe y hay que crearla
          jsonObject.put(hashtag, 1);
        } else {
          obj2++;
          jsonObject.put(hashtag, obj2);
        }
      }
      lock.release();
      fileChannel.close();
      logger.debug("sssssss" + jsonObject.toJSONString());
      fileChannel = FileChannel.open(path, StandardOpenOption.WRITE,
                StandardOpenOption.TRUNCATE_EXISTING);
      lock = fileChannel.lock(); // gets an exclusive lock
        buffer = ByteBuffer.wrap(jsonObject.toJSONString().getBytes());
        fileChannel.write(buffer);
    } catch (Exception e) {
      logger.error("Error guardar el indice de TT: " + e);
    } finally {
      lock.release();
      fileChannel.close();
    }
  }

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

    logger.info("Guardando el comando en la base de datos: " + fileName);
    logger.info("Contenido del registro: " + command.toJson());
    File tmpFile = new File(fileName);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
    }
    JSONObject obj2 = new JSONObject();
    obj2.put("command", command.getCommand().toString());
    obj2.put("user", command.getUser());
    obj2.put("message", command.getMessage());
    obj2.put("timestamp", command.getTimestamp());
    JSONObject jsonObject = new JSONObject();
    jsonObject.put(command.getUuid().toString(), obj2);

        Path path = Paths.get(fileName);
        FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.WRITE,
              StandardOpenOption.APPEND);
        FileLock lock = fileChannel.lock(); // gets an exclusive lock
    try {
      ByteBuffer buffer = ByteBuffer.wrap((jsonObject.toJSONString() + String.format("%n")).getBytes());
      fileChannel.write(buffer);
    } catch (Exception e) {
      logger.error("Error guardar la base de datos: " + e);
    } finally {
      lock.release();
      fileChannel.close();
    }
    // Una vez que persisto el mensaje, actualizo los indices y el TT
    updateUserIndex(command);
    updateHashTagIndex(command);
    updateTT(command);
  }

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

    logger.info("Actualizando el inice de usuarios");
    File tmpFile = new File(fileName);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
      oFile.write("{}".getBytes());
    }

        Path path = Paths.get(fileName);
        FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ);
        FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
        try {
    ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
    fileChannel.read(buffer);
    buffer.position(0);
    StringBuilder sb = new StringBuilder();
        while (buffer.hasRemaining()) {
            sb.append((char) buffer.get());
        }
        String tmp = sb.toString();
        if((tmp.split("}", -1).length - 1) > 1) {
          tmp = tmp.substring(0, tmp.indexOf("}")+1);
        }
    obj = parser.parse(new StringReader(tmp));
    JSONObject jsonObject = (JSONObject) obj;
    JSONArray array = (JSONArray) jsonObject.get(command.getUser());
    if (array ≡ null) {
      // Hay que crear la entrada en el indice
      JSONArray ar2 = new JSONArray();
      ar2.add(command.getUuid().toString());
      jsonObject.put(command.getUser(), ar2);
    } else {
      array.add(command.getUuid().toString());
```

```
      jsonObject.put(command.getUser(), array);
    }
    lock.release();
    fileChannel.close();
    fileChannel = FileChannel.open(path, StandardOpenOption.WRITE,
              StandardOpenOption.TRUNCATE_EXISTING);
    lock = fileChannel.lock(); // gets an exclusive lock
      buffer = ByteBuffer.wrap(jsonObject.toJSONString().getBytes());
      fileChannel.write(buffer);
    } catch (Exception e) {
      logger.error("Error guardar el indice de u: " + e);
    } finally {
      lock.release();
      fileChannel.close();
    }
  }

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

    logger.info("Actualizando el inice de hashtags");
    File tmpFile = new File(fileName);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
      oFile.write("{}".getBytes());
    }

        Path path = Paths.get(fileName);
        FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ);
        FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
        try {
    ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
    fileChannel.read(buffer);
    buffer.position(0);
    StringBuilder sb = new StringBuilder();
        while (buffer.hasRemaining()) {
            sb.append((char) buffer.get());
        }
        String tmp = sb.toString();
        if((tmp.split("}", -1).length - 1) > 1) {
          tmp = tmp.substring(0, tmp.indexOf("}")+1);
        }
    obj = parser.parse(new StringReader(tmp));
    JSONObject jsonObject = (JSONObject) obj;
    JSONArray array;
    String regexPattern = "(#\\w+)";
    Pattern p = Pattern.compile(regexPattern);
    Matcher m = p.matcher(command.getMessage());
    String hashtag;
    JSONArray ar2;
    while (m.find()) {
      hashtag = m.group(1);
      hashtag = hashtag.substring(1, hashtag.length());
      array = (JSONArray) jsonObject.get(hashtag);
      if (array ≡ null) {
        // Hay que crear la entrada en el indice
        ar2 = new JSONArray();
        ar2.add(command.getUuid().toString());
        jsonObject.put(hashtag, ar2);
      } else {
        array.add(command.getUuid().toString());
        jsonObject.put(hashtag, array);
      }
    }
    lock.release();
    fileChannel.close();
    fileChannel = FileChannel.open(path, StandardOpenOption.WRITE,
              StandardOpenOption.TRUNCATE_EXISTING);
    lock = fileChannel.lock(); // gets an exclusive lock
      buffer = ByteBuffer.wrap(jsonObject.toJSONString().getBytes());
      fileChannel.write(buffer);
    } catch (Exception e) {
      logger.error("Error guardar el indice de hashtags: " + e);
    } finally {
      lock.release();
      fileChannel.close();
    }
  }

  public String query(Command command) throws IOException, ParseException {
    List<String> resultList = new ArrayList<String>();
    String listString = "";
    if (String.valueOf(command.getMessage().charAt(0)).equals("#")) { // #
      resultList = queryBy(command.getMessage().substring(1,
          command.getMessage().length()), "HASHTAG");
    } else if (command.getMessage().equals("TT")) { // Es consulta por TT
      resultList = queryTT(command.getMessage());
    } else { // Es consulta por usuario
      resultList = queryBy(command.getMessage(), "USER");
    }
    if(¬resultList.isEmpty()) {
      for (String element : resultList) {
        listString += element + "\n";
      }
    }
    return listString;
  }

  private List<String> queryTT(String hashTag)
      throws FileNotFoundException, IOException, ParseException {
    Map<String, Long> map = new HashMap<String, Long>();
    String fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_TT;
    List<String> returnList = new ArrayList<String>();

    // Levantar el json
    JSONParser parser = new JSONParser();

        Path path = Paths.get(fileName);
        FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ
);
        FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
        try {
        ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
    fileChannel.read(buffer);
    buffer.position(0);
    StringBuilder sb = new StringBuilder();
        while (buffer.hasRemaining()) {
            sb.append((char) buffer.get());
        }

    Object obj = parser.parse(new StringReader(sb.toString()));

    JSONObject jsonObject = (JSONObject) obj;

    // Crear un map
    for (Iterator iterator = jsonObject.keySet().iterator(); iterator
        .hasNext();) {
```

```
      String key = (String) iterator.next();
      map.put(key, (Long) jsonObject.get(key));
    }

    returnList = sortHashMapByValues(map);
    returnList
        .add("Total de topics: " + String.valueOf(map.keySet().size()));
        } catch(Exception e) {
          // Do nothing
        } finally {
    lock.release();
    fileChannel.close();
        }
    return returnList;
  }

  private List<String> queryBy(String key, String type)
      throws IOException, ParseException {
    String fileName;
    JSONParser parser = new JSONParser();
    Object obj, obj2;
    List<String> messageList = new ArrayList<String>();
    String file, id;

    if (type.equals("USER")) {
      logger.info("Consultando por user");
      fileName = Constants.DB_INDEX_DIR + "/" + Constants.DB_USER_INDEX;
    } else if (type.equals("HASHTAG")) {
      logger.info("Consultando por hashtag");
      fileName = Constants.DB_INDEX_DIR + "/"
          + Constants.DB_HASHTAG_INDEX;
    } else {
      return messageList;
    }

    // Obtengo la lista de archivos que contienen el user

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

        Path path = Paths.get(fileName);
        FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ
);
        FileLock lock = fileChannel.lock(0, Long.MAX_VALUE, true);
        try {
    ByteBuffer buffer = ByteBuffer.allocate(((int) fileChannel.size()));
    fileChannel.read(buffer);
    buffer.position(0);
    StringBuilder sb = new StringBuilder();
        while (buffer.hasRemaining()) {
            sb.append((char) buffer.get());
        }

    obj = parser.parse(new StringReader(sb.toString()));
    JSONObject jsonObject = (JSONObject) obj;
    JSONArray array = (JSONArray) jsonObject.get(key);

    String line, reg;
    JSONObject jsonObject2;
    int remainingPost = queryCountShowPosts;
    // Abro archivo por archivo y recupero los mensajes
    if (array ≠ null) {
      ListIterator<String> iterator = array.listIterator(array.size());
      while (iterator.hasPrevious() ∧ remainingPost > 0) {
        id = iterator.previous();
        file = Constants.DB_DIR + "/" + id.substring(0, shardingFactor)
            + Constants.COMMAND_SCRIPT_EXTENSION;
        Path path2 = Paths.get(file);
        FileChannel fileChannel2 = FileChannel.open(path2, StandardOpenOption.READ);
        FileLock lock2 = fileChannel2.lock(0, Long.MAX_VALUE, true);
        ByteBuffer buffer2 = ByteBuffer.allocate(((int) fileChannel2.size()));
        fileChannel2.read(buffer2);
        buffer2.position(0);
        StringBuilder sb2 = new StringBuilder();
        while (buffer2.hasRemaining()) {
          sb2.append((char) buffer2.get());
        }
        try (
          BufferedReader br = new BufferedReader(
          new StringReader(sb2.toString()))
        ) {
          while ((line = br.readLine()) ≠ null ∧ remainingPost > 0
              ∧ ¬("").equals(line.trim())) {
            System.out.println("line: " + line);
            obj2 = parser.parse(line);
            jsonObject2 = (JSONObject) obj2;
            if (jsonObject2.get(id) ≠ null) {
              messageList.add(jsonObject2.get(id).toString());
            }
            remainingPost--;
          }
        }
        lock2.release();
        fileChannel2.close();
      }
    }
        }catch(Exception e) {
          // Do nothing
        } finally {
    lock.release();
    fileChannel.close();
        }
    // Retorno la lista con los mensajes encontrados
    return messageList;
  }

  public void delete(Command command)
      throws IOException, ParseException {
    String file = Constants.DB_DIR + "/"
        + command.getMessage().substring(0, shardingFactor)
        + Constants.COMMAND_SCRIPT_EXTENSION;
    String fileTmp = file + ".tmp";
    JSONParser parser = new JSONParser();
    Object obj2;
    String line, key;
    JSONObject jsonObject2;

    // Creo un archivo temporal
    PrintWriter pw = new PrintWriter(
        new BufferedWriter(new FileWriter(fileTmp)));

    logger.info("Eleiminando registro");

    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
      while ((line = br.readLine()) ≠ null) {
        obj2 = parser.parse(line);
        jsonObject2 = (JSONObject) obj2;
        key = (String) jsonObject2.keySet().iterator().next();
```

```
        if (¬(key.equals(command.getMessage()))) {
          // Si no es la clave a borrar, guardo el registro en un
          // archivo temporal
          pw.println(jsonObject2);
        }
      }
    }
    pw.close();
    // Borro el archvio original y renombro el tmp
    File fileToDelete = new File(file);
    File newFile = new File(fileTmp);
    if (fileToDelete.delete()) {
      logger.info("Archivo original borrado");
      logger.info("Renombrado el archivo temporal al original");
      if (newFile.renameTo(fileToDelete)) {
        logger.info("Archivo renombrado con exito");
      } else {
        logger.error("No se ha podido renombrar el archivo");
        throw new IOException();
      }
    } else {
      logger.error(
          "No se ha podido borrar el registro. Se aborta la operacion");
      throw new IOException();
    }
  }

  private List<String> sortHashMapByValues(Map<String, Long> map) {
    List<String> mapKeys = new ArrayList<String>(map.keySet());
    List<Long> mapValues = new ArrayList<Long>(map.values());
    Collections.sort(mapValues);
    Collections.sort(mapKeys);

    LinkedHashMap<String, Long> sortedMap = new LinkedHashMap<String, Long>();

    java.util.Iterator<Long> valueIt = mapValues.iterator();
    while (valueIt.hasNext()) {
      Long val = valueIt.next();
      java.util.Iterator<String> keyIt = mapKeys.iterator();

      while (keyIt.hasNext()) {
        String key = keyIt.next();
        Long comp1 = map.get(key);
        Long comp2 = val;

        if (comp1.equals(comp2)) {
          keyIt.remove();
          sortedMap.put(key, val);
          break;
        }
      }
    }
    List<String> tt = new ArrayList<String>();
    ArrayList<String> keys = new ArrayList<String>(sortedMap.keySet());
    int i = keys.size() - 1;
    int j = ttCountShowPosts;
    while (i ≥ 0 ∧ j > 0) {
      tt.add(keys.get(i));
      j--;
      i--;
    }
    return tt;
  }

}
```

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

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

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

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

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

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

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

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

          } catch (ClassNotFoundException | IOException e) {
            logger.error("No se ha podido deserializar el mensaje");
          }
        }
      }
    } catch (InterruptedException e) {
      // Do nothing
      logger.error("Error al analizar comando: " + e);
    } finally {
      Iterator it = usersMap.entrySet().iterator();
      while (it.hasNext()) {
          Map.Entry pair = (Map.Entry)it.next();
```

```java
          WritingRemoteQueue userQueue = (WritingRemoteQueue) pair.getValue();
          try {
        userQueue.close();
      } catch (IOException | TimeoutException e) {
        // Do nothing
        logger.error("Error al cerrar una response user queue: " + e);
      }
          it.remove(); // avoids a ConcurrentModificationException
      }
    }
    logger.info("Storgae Controller terminado");
  }

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

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

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

  private void sendResponse(Response response) {
    logger.info("Siguiente respuesta");
    WritingRemoteQueue currentUserRemoteQueue;
    currentUserRemoteQueue = usersMap.get(response.getUser());
    if (currentUserRemoteQueue ≡ null) {
      // Creo la cola
      try {
        currentUserRemoteQueue = new WritingRemoteQueue(
            response.getUser(), config.get(Constants.KAFKA_WRITE_PROPERTIES));
      } catch (IOException e) {
        logger.error("No se han podido crear las colas de kafka: " + e);
        System.exit(1);
      }
      usersMap.put(response.getUser(), currentUserRemoteQueue);
    }
    logger.info(
        "Enviando respuesta al usuario: " + response.getUser());
    logger.info("UUID: " + response.getUuid());
    logger.info("Status de la respuesta: "
        + response.getResponse_status());
    logger.info(
        "Contenido de la respuesta: " + response.getMessage());
    logger.info("Esperando siguiente respuesta");
    try {
      usersMap.get(response.getUser()).push(response);
      logger.info("Respuesta enviada: " + response.getUser() + ":" + response.getMess
age()
      + ":" + response.getResponse_status() + ":" + response.getUuid());
    } catch (IOException e) {
      logger.error(
          "No se ha podido enviar la respuesta al usuario "
              + response.getUser());
    }
  }
}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

public class DispatcherController {

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

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

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

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


    logger.info("Iniciando el dispatcher controller");
    try {
      while (¬Thread.interrupted()) {
        messageList = dispatcherQueue.pop();
        Iterator<byte[]> it = messageList.iterator();
        while (it.hasNext()) {
          try {
            command = new Command();
            command.deserialize(it.next());
            logger.info(
                "Comando recibido con los siguientes parametros: "
                    + "\nUsuario: " + command.getUser()
                    + "\nComando: " + command.getCommand()
                    + "\nMensaje: " + command.getMessage());
            switch (command.getCommand()) {
            case PUBLISH:
```

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

```java
                storageQueue.push(command);
                analyzerQueue.push(command);
                loggerQueue.push(command);
                logger.info("Comando enviado al publish: "
                    + "\nUsuario: " + command.getUser()
                    + "\nComando: " + command.getCommand()
                    + "\nMensaje: " + command.getMessage());
                break;
              case QUERY:
                storageQueue.push(command);
                loggerQueue.push(command);
                logger.info("Comando enviado al query: "
                    + "\nUsuario: " + command.getUser()
                    + "\nComando: " + command.getCommand()
                    + "\nMensaje: " + command.getMessage());
                break;
              case DELETE:
                logger.info("Comando enviado al delete: "
                    + "\nUsuario: " + command.getUser()
                    + "\nComando: " + command.getCommand()
                    + "\nMensaje: " + command.getMessage());
                storageQueue.push(command);
                loggerQueue.push(command);
                break;
              case FOLLOW:
                logger.info("Comando enviado al follow: "
                    + "\nUsuario: " + command.getUser()
                    + "\nComando: " + command.getCommand()
                    + "\nMensaje: " + command.getMessage());
                analyzerQueue.push(command);
                loggerQueue.push(command);
                break;
              default:
                logger.error("Comando invalido");
                break;
              }
            } catch (ClassNotFoundException | IOException e) {
              logger.error("No se ha podido deserializar el mensaje: " + e);
            }
          }
        }
      } finally {
        try {
          storageQueue.close();
          dispatcherQueue.close();
          analyzerQueue.close();
        } catch (IOException | TimeoutException e) {
          // Do nothing
          logger.error("No se ha podido cerrar alguna de las colas");
          logger.debug(e);
        }
      }
      logger.info("Dispatcher controller terminado");
    }
  }
```

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

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

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

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

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

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

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

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

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

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

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

}
```

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

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

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

public class Response implements Serializable, ISerialize {

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

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

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

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

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

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

  public UUID getUuid() {
    return uuid;
  }

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

  public RESPONSE_STATUS getResponse_status() {
    return response_status;
  }

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

  public String getMessage() {
    return message;
  }

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

  public String getUser() {
    return user;
  }

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

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

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

public abstract class RemoteQueue {

  public abstract void close() throws IOException, TimeoutException;

}
```

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

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

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

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

  public class ReadingRemoteQueueException extends WakeupException {
  }

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

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

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

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

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

```

```java
}
```

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

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

import java.io.IOException;

public interface ISerialize {

  public byte[] serialize() throws IOException;

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

}
```

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

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

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

public class Constants {

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

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

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

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

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

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

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

  // Constantes para el analyzer
  public static final String ANALYZER_QUEUE_HOST = "analyzer.queue.host";
  public static final String ANALYZER_QUEUE_NAME = "analyzer.queue.name";
  public static final long ANALYZER_THREAD_WAIT_TIME = 5000;

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

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

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

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

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

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

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

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

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

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

public class ConfigLoader {

  private Map<String, String> propertiesMap;

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

    propertiesMap = Collections.unmodifiableMap(propertiesMap);
  }

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

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

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

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

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

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

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

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

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

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

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

```

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

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

  public String getUser() {
    return user;
  }

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

  public String getMessage() {
    return message;
  }

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

  public UUID getUuid() {
    return uuid;
  }

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

  public String getTimestamp() {
    return timestamp;
  }

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

  public String toJson() {
    String tmp;

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

  public void fromJson(String jsonString) {

  }
}
```

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

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

import org.apache.log4j.Logger;
import org.apache.log4j.MDC;
import org.apache.log4j.PropertyConfigurator;
import ar.fiuba.taller.common.ConfigLoader;
import ar.fiuba.taller.common.Constants;

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

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

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

    String mode = args[0];

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

    if (mode.equals(Constants.INTERACTIVE_MODE)) {
      if ((args[1] ≡ null ∨ ("").equals(args[1]))
          ∧ (args[2] ≡ null ∨ ("").equals(args[2]))) {
        displayHelp();
      }
      System.out.printf(
          "Iniciando el Client console en modo interactivo para el usuario %s",
          args[1]);
      InteractiveUser interactiveUser = new InteractiveUser(configLoader.getProp
erties(), args[1], args[2]);
      interactiveUser.run();
    } else if (mode.equals(Constants.BATCH_MODE)) {
      try {
        usersAmount = Integer.parseInt(args[1]);
      } catch (NumberFormatException e) {
        System.out.printf("Argumento invalido");
        System.exit(1);
      }
      ExecutorService executor = Executors.newFixedThreadPool(usersAmount);
      System.out.printf("Iniciando el Client console en modo batch");
      for (int i = 0; i < Integer.parseInt(args[1]); i++) {
        usersSet.add(new BatchUser(configLoader.getProperties(),
            "user" + i, configLoader.getProperties().get(Constants.USERS_RESPONSE
_HOST)));
      }
      try {
        executor.invokeAll(usersSet);
```

```java
      }catch (Exception e) {
        logger.error("Error al invocar a los usuarios: " + e);
      } finally {
        executor.shutdownNow();
        try {
          executor.awaitTermination(
              Constants.USER_THREAD_WAIT_TIME,
              TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) {
          // Do nothing
        }
      }
    }
  }

  private static void displayHelp() {
    System.out.printf(
        "Client console%n*************%nSintaxis:%n./ClientConsole <params>%nParametros:%ni [usernam
e] [host]: Inicia el cliente en modo interactivo%nusername: Nombre del usuario%nhost: Nombre y puerto del servidor a
 conectar (ej. localhost:9092)%n%nb [usersamount] [host]: Inicia el cliente en modo batch%nusersamount: Cantidad de
 usuarios a simular%nhost: Nombre y puerto del servidor a conectar (ej. localhost:9092)%n%n");
    System.exit(Constants.EXIT_FAILURE);
  }

}
```

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

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Map;
import java.util.concurrent.TimeoutException;
import ar.fiuba.taller.common.Command;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.ReadingRemoteQueue;
import ar.fiuba.taller.common.WritingRemoteQueue;

public class InteractiveUser {
  String userName;
  private CommandController commandController;
  private Thread eventViewerThread;
  private ReadingRemoteQueue remoteUserResponseQueue;
  private WritingRemoteQueue dispatcherQueue;

  public InteractiveUser(Map<String, String> config, String userName,
      String userHost) {
    this.userName = userName;
    try {
      dispatcherQueue = new WritingRemoteQueue(
          config.get(Constants.DISPATCHER_QUEUE_NAME),
          config.get(Constants.KAFKA_WRITE_PROPERTIES));
      remoteUserResponseQueue = new ReadingRemoteQueue(userName, config.get(Cons
tants.KAFKA_READ_PROPERTIES));
    } catch (IOException e) {
      System.out.printf("No se han podido inicializar las colas de kafka: %s", e);
      System.exit(1);
    }
    commandController =
        new CommandController(dispatcherQueue,
            Integer.parseInt(config.get(Constants.MAX_LENGTH_MSG)),
            Constants.LOGS_DIR + "/" + userName
                + Constants.COMMANDS_FILE_EXTENSION);
    eventViewerThread = new Thread(new EventWriter(
        Constants.LOGS_DIR + "/" + userName
            + Constants.EVENT_VIEWER_FILE_EXTENSION,
            remoteUserResponseQueue));
  }

  public void run() {
    BufferedReader br = null;
    String[] msgParts;

    eventViewerThread.start();
    br = new BufferedReader(new InputStreamReader(System.in));
    while (¬Thread.interrupted()) {
      try {
        System.out.print("Enter command: ");
        String input = br.readLine();
        msgParts = input.split(":");
        commandController.sendMessage(new Command(msgParts[0], userName,
            msgParts[1], null, null));
      } catch (IOException e) {
        System.out.println(
            "Error: No se ha podido procesar el comando");
      }
    }

    remoteUserResponseQueue.shutDown();
    try {
      remoteUserResponseQueue.close();
    } catch (IOException | TimeoutException e) {
```

```java
      // Do nothing
    }
    eventViewerThread.interrupt();
    try {
      eventViewerThread.join(Constants.USER_THREAD_WAIT_TIME);
    } catch (InterruptedException e1) {
      // Do nothing
    }
  }
}
```

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

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

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

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

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

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

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

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

```

```java
  }

}
```

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

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

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

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

  public void sendMessage(Command command) {
    PrintWriter pw;

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

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

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

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

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

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

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

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

    eventViewerThread.start();

    try {
```

```java
        JSONParser parser = new JSONParser();
        Object obj = parser.parse(new FileReader(Constants.COMMAND_SCRIPT));
        JSONObject jsonObject = (JSONObject) obj;
        JSONArray commandArray = (JSONArray) jsonObject
            .get(Constants.COMMAND_ARRAY);
        JSONObject commandObject;
        Command command;
        List<Integer> commandIndexList = getCommandIndexList(commandAmount,
            commandArray.size());
        Iterator<Integer> iterator = commandIndexList.iterator();

        while (iterator.hasNext()) {
          commandObject = (JSONObject) commandArray.get(iterator.next());
          command = new Command(
              (String) commandObject.get(Constants.COMMAND_KEY),
              userName,
              (String) commandObject.get(Constants.MESSAGE_KEY), null,
              null);
          logger.debug("COMANDO: " + count
              + ".Se inserto comando con los siguientes parametros: "
              + "\nUsuario: " + command.getUser() + "\nComando: "
              + command.getCommand() + "\nMensaje: "
              + command.getMessage());
          commandController.sendMessage(command);
          ++count;
        }
      } catch (ParseException | IOException e) {
        logger.error("Error al tratar el script de comandos: " + e);
      }
      return null;
    }

    private List<Integer> getCommandIndexList(int commandListIndexSize,
        int maxCommandsAvailable) {
      List<Integer> commandIndexList = new ArrayList<Integer>();

      for (int i = 0; i < commandListIndexSize; i++) {
        commandIndexList.add((int) (Math.random() * maxCommandsAvailable));
      }

      return commandIndexList;
    }

}
```

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

import java.io.IOException;
import org.apache.log4j.Logger;
import org.apache.log4j.MDC;
import org.apache.log4j.PropertyConfigurator;

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

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

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

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

    final ReadingRemoteQueue loggerQueue = new ReadingRemoteQueue(
        configLoader.getProperties()
            .get(Constants.AUDIT_LOGGER_QUEUE_NAME),
        configLoader.getProperties()
            .get(Constants.KAFKA_READ_PROPERTIES));

    AuditLogger auditLogger = new AuditLogger(loggerQueue, configLoader.getPrope
rties());
    auditLogger.run();
    loggerQueue.shutDown();
    loggerQueue.close();
  }
}
```

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

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

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

import ar.fiuba.taller.common.*;

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

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

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

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

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

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

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

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

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

}
```

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

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

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

import ar.fiuba.taller.common.Constants;

public class UserRegistry {

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

  public UserRegistry() {
  }

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

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

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

    obj = parser.parse(new FileReader(tmpFile));
    jsonObject = (JSONObject) obj;
    JSONArray array = (JSONArray) jsonObject.get(updateKey);
    if (array ≡ null) {
      // Hay que crear la entrada en el indice
      JSONArray ar2 = new JSONArray();
      ar2.add(follower);
```

```java
      jsonObject.put(updateKey, ar2);
    } else {
      array.add(follower);
      jsonObject.put(updateKey, array);
    }
    file = new FileWriter(tmpFile);
    try {
      file.write(jsonObject.toJSONString());
    } catch (Exception e) {
      logger.error("Error al guardar el index: " + e);
    } finally {
      file.flush();
      try {
        file.close();
      } catch (IOException e) {
        logger.error("No se ha podido cerrar el archivo de registro: " + e);
      }
    }
  }

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

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

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

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

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

    File tmpFile = new File(hashtagFile);
    if (tmpFile.createNewFile()) {
      FileOutputStream oFile = new FileOutputStream(tmpFile, false);
      oFile.write("{}".getBytes());
    }
    logger.info("Obteniendo hashtags de " + followed);
    obj = parser.parse(new FileReader(hashtagFile));
    jsonObject = (JSONObject) obj;
    String regexPattern = "(#\\w+)";
    Pattern p = Pattern.compile(regexPattern);
    Matcher m = p.matcher(followed);
    while (m.find()) {
      word = m.group(1).substring(1, m.group(1).length());
      logger.info("Hashtag: " + m.group(1));
      jsonArray = (JSONArray) jsonObject.get(word);
      logger.info("arr: " + jsonArray);
      if (jsonArray ≠ null) {
        it = jsonArray.iterator();
        while (it.hasNext()) {
          followersList.add(it.next());
        }
      }
    }
    return followersList;
  }
}
```

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

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

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

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

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

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

    logger.info("Iniciando el analyzer");

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

    ReadingRemoteQueue analyzerQueue = null;
    try {
      analyzerQueue = new ReadingRemoteQueue(
          configLoader.getProperties().get(Constants.ANALYZER_QUEUE_NAME),
          configLoader.getProperties().get(Constants.KAFKA_READ_PROPERTIES));
    } catch (IOException e1) {
      logger.error("No se ha podido inicializar la cola de kafka: " + e1);
      System.exit(Constants.EXIT_FAILURE);
    }

    AnalyzerController analyzerController = new AnalyzerController(
        configLoader.getProperties(), analyzerQueue);
    analyzerController.run();
    analyzerQueue.shutDown();
    try {
      analyzerQueue.close();
    } catch (IOException | TimeoutException e) {
      // Do nothing
      logger.error("No se ha podido cerrar la cola del analyzer: " + e);
    }
  }
}
```

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

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

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

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

public class AnalyzerController {

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

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

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

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

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

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

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

    if (tmpQueue ≡ null) {
      tmpQueue = new WritingRemoteQueue(username, config.get(Constants.KAFKA_WRI
TE_PROPERTIES));
      usersMap.put(username, tmpQueue);
    }
    return usersMap.get(username);
  }

}
```

Table of Contents