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

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

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

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

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

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

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.ListIterator;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

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

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

public class Storage {

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

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

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

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

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

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

    }

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

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

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

```java
    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());
    }
    obj = parser.parse(new FileReader(fileName));
    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);
    }
    FileWriter file = new FileWriter(fileName);
    try {
      file.write(jsonObject.toJSONString());
    } catch (Exception e) {
      logger.error("Error al guardar el user index");
      logger.info(e.toString());
      e.printStackTrace();
    } finally {
      file.flush();
      file.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());
    }
    obj = parser.parse(new FileReader(fileName));
    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);
      }
    }
    FileWriter file = new FileWriter(fileName);
    try {
      file.write(jsonObject.toJSONString());
    } catch (Exception e) {
      logger.error("Error guardar el indice de hashtags");
      logger.info(e.toString());
      e.printStackTrace();
    } finally {
      file.flush();
      file.close();
    }
  }

  public String query(Command command) throws IOException, ParseException {
    List<String> resultList;
    String listString = "";
    if (String.valueOf(command.getMessage().charAt(0)).equals("#")) { // Es
                                      // consulta
                                      // por
                                      // hashtag
      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");
    }
    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 = null;

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

    Object obj = parser.parse(new FileReader(fileName));

    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);
```

```java
    returnList
        .add("Total de topics: " + String.valueOf(map.keySet().size()));
    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 null;
    }

    // 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());
    }
    obj = parser.parse(new FileReader(fileName));
    JSONObject jsonObject = (JSONObject) obj;
    JSONArray array = (JSONArray) jsonObject.get(key);

    System.out.println(array.toJSONString());

    BufferedReader br2;
    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();
        System.out.println("id: " + id);
        file = Constants.DB_DIR + "/" + id.substring(0, shardingFactor)
            + Constants.COMMAND_SCRIPT_EXTENSION;
        System.out.println("file: " + file);
        try (BufferedReader br = new BufferedReader(
            new FileReader(file))) {
          while ((line = br.readLine()) ≠ null
              ∧ remainingPost > 0) {
            System.out.println("line: " + line);
            obj2 = parser.parse(line);
            jsonObject2 = (JSONObject) obj2;
            messageList.add(jsonObject2.get(id).toString());
            remainingPost−−;
          }
        }
      }
    }
    // Retorno la lista con los mensajes encontrados
    return messageList;
  }

  public synchronized 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) {
        System.out.println("line: " + line);
        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()) {
```

```java
        String key = keyIt.next();
        Long comp1 = map.get(key);
        Long comp2 = val;

        if (comp1.equals(comp2)) {
          keyIt.remove();
          sortedMap.put(key, val);
          break;
        }
      }
    }
    Map<String, Long> map2 = sortedMap;
    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.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

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

import com.rabbitmq.client.AMQP.BasicProperties;
import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;

import ar.fiuba.taller.common.Command;
import ar.fiuba.taller.common.ConfigLoader;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.RemoteQueue;
import ar.fiuba.taller.common.Response;

public class StorageController extends DefaultConsumer implements Runnable {

  private Thread createControllerThread;
  private Thread queryControllerThread;
  private Thread removeControllerThread;
  private Thread responseControllerThread;
  private BlockingQueue<Command> queryQueue;
  private BlockingQueue<Command> removeQueue;
  private BlockingQueue<Command> createQueue;
  private BlockingQueue<Response> responseQueue;
  private ConfigLoader configLoader;
  private Storage storage;
  private RemoteQueue storageQueue;
  final static Logger logger = Logger.getLogger(StorageController.class);

  public StorageController(RemoteQueue storageQueue) {
    super(storageQueue.getChannel());
    configLoader = ConfigLoader.getInstance();
    storage = new Storage(configLoader.getShardingFactor(),
        configLoader.getQueryCountShowPosts(),
        configLoader.getTtCountShowPosts());
    this.storageQueue = storageQueue;
  }

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

    logger.info("Iniciando el storage controller");

    try {
      logger.info("Cargando la configuracion");
      configLoader.init(Constants.CONF_FILE);

      logger.info(
          "Creando las colas de consultas, removes, creates y response");
      queryQueue = new ArrayBlockingQueue<Command>(
          Constants.COMMAND_QUEUE_SIZE);
      removeQueue = new ArrayBlockingQueue<Command>(
          Constants.COMMAND_QUEUE_SIZE);
      createQueue = new ArrayBlockingQueue<Command>(
          Constants.COMMAND_QUEUE_SIZE);
      responseQueue = new ArrayBlockingQueue<Response>(
          Constants.COMMAND_QUEUE_SIZE);

      logger.info("Instancio los indices de usuarios y hashtags");

      logger.info("Creando los threads de query, remove y create");
```

```java
      queryControllerThread = new Thread(
          new QueryController(queryQueue, responseQueue, storage));
      removeControllerThread = new Thread(
          new RemoveController(removeQueue, responseQueue, storage));
      createControllerThread = new Thread(
          new CreateController(createQueue, responseQueue,
              configLoader.getShardingFactor(), storage));
      responseControllerThread = new Thread(
          new ResponseController(responseQueue));

      logger.info("Lanzando los threads de query, remove y create");
      queryControllerThread.start();
      removeControllerThread.start();
      createControllerThread.start();
      responseControllerThread.start();

      logger.info("Me pongo a comer de la cola: " + storageQueue.getHost()
          + " " + storageQueue.getQueueName());
      storageQueue.getChannel().basicConsume(storageQueue.getQueueName(),
          true, this);

    } catch (IOException e) {
      logger.error("Error al cargar el archivo de configuracion");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }

  @Override
  public void handleDelivery(String consumerTag, Envelope envelope,
      BasicProperties properties, byte[] body) throws IOException {
    super.handleDelivery(consumerTag, envelope, properties, body);
    Command command = new Command();
    try {
      command.deserialize(body);
      logger.info("Comando recibido con los siguientes parametros: "
          + "\nUUID: " + command.getUuid() + "\nUsuario: "
          + command.getUser() + "\nComando: " + command.getCommand()
          + "\nMensaje: " + command.getMessage());

      switch (command.getCommand()) {
      case PUBLISH:
        logger.info(
            "Comando recibido: PUBLISH. Insertando en la cola de creacion.");
        createQueue.put(command);
        break;
      case QUERY:
        logger.info(
            "Comando recibido: QUERY. Insertando en la cola de consultas.");
        queryQueue.put(command);
        break;
      case DELETE:
        logger.info(
            "Comando recibido: DELETE. Insertando en la cola de borrado.");
        removeQueue.put(command);
        break;
      default:
        logger.info("Comando recibido invalido. Comando descartado.");
      }
    } catch (ClassNotFoundException e) {
      logger.error("Error al deserializar el comando");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error("Error al deserializar el comando");
      logger.info(e.toString());
```

```java
        e.printStackTrace();
      } catch (InterruptedException e) {
        logger.error("Error al insertar el comando en alguna de las colas");
        logger.info(e.toString());
        e.printStackTrace();
      }
    }

}
```

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

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

import org.apache.log4j.Logger;

import ar.fiuba.taller.common.ConfigLoader;
import ar.fiuba.taller.common.RemoteQueue;
import ar.fiuba.taller.common.Response;

import java.io.IOException;
import java.util.*;

public class ResponseController implements Runnable {

  private BlockingQueue<Response> responseQueue;
  Response response;
  RemoteQueue remoteQueue;
  Map<String, RemoteQueue> usersMap;
  final static Logger logger = Logger.getLogger(ResponseController.class);

  public ResponseController(BlockingQueue<Response> responseQueue) {
    this.responseQueue = responseQueue;
    usersMap = new HashMap<String, RemoteQueue>();
  }

  public void run() {
    logger.info("Iniciando el response controller");
    while (true) {
      try {
        logger.info("Esperando siguiente respuesta");
        response = responseQueue.take();
        remoteQueue = usersMap.get(response.getUser());
        if (remoteQueue ≡ null) {
          // Creo la cola
          remoteQueue = new RemoteQueue(response.getUser(),
              ConfigLoader.getInstance().getUsersServer());
          remoteQueue.init();
          usersMap.put(response.getUser(), remoteQueue);
        }
        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");
        usersMap.get(response.getUser()).put(response);
        logger.info("Respuesta enviada");
      } catch (InterruptedException e) {
        logger.error(
            "Error al tomar respuestas de la cola responseQueue");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        logger.error("Error al insertar respuesta en la cola remota");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (TimeoutException e) {
        logger.error("Error al iniciar la cola remota del usuario");
        logger.info(e.toString());
        e.printStackTrace();
      }
    }

  }

}
```

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

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

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

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

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

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

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    String error_message = "Error al eliminar el mensaje";
    logger.info("Iniciando el remove controller");
    while (true) {
      try {
        command = removeQueue.take();
        response = new Response();
        response.setUuid(UUID.randomUUID());
        response.setUser(command.getUser());
        storage.delete(command);
        response.setMessage("Borrado exitoso");
        response.setResponse_status(RESPONSE_STATUS.OK);
      } catch (InterruptedException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.error("Error borrar el mensaje");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (ParseException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.error("Error borrar el mensaje");
        logger.info(e.toString());
        e.printStackTrace();
      } finally {
        try {
          responseQueue.put(response);
        } catch (InterruptedException e) {
          logger.error("No se pudo enviar la respuesta");
          logger.info(e.toString());
          e.printStackTrace();
        }
      }
    }
  }
}
```

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

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

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

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

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

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

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    String queryResult;
    // Este mensaje deberia ser configurable
    String error_message = "Error al consultar";
    logger.info("Iniciando el query controller");
    while (true) {
      try {
        command = queryQueue.take();
        response = new Response();
        response.setUuid(UUID.randomUUID());
        response.setUser(command.getUser());
        response.setMessage(storage.query(command));
        logger.debug(response.getMessage());
        response.setResponse_status(RESPONSE_STATUS.OK);
      } catch (InterruptedException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.error("Error al sacar comando de la cola removeQueue");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.error("Error borrar el mensaje");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (ParseException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.error("Error borrar el mensaje");
        logger.info(e.toString());
        e.printStackTrace();
      } finally {
        try {
          responseQueue.put(response);
        } catch (InterruptedException e) {
          logger.error("No se pudo enviar la respuesta");
          logger.info(e.toString());
          e.printStackTrace();
        }
      }
    }
  }

}
```

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

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

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

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

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

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

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

    while (true) {
      String error_message = "Error al crear el mensaje";
      try {
        command = createQueue.take();
        response = new Response();
        response.setUuid(UUID.randomUUID());
        response.setUser(command.getUser());
        storage.saveMessage(command);
        response.setMessage("Creacion exitosa");
        response.setResponse_status(RESPONSE_STATUS.OK);
      } catch (InterruptedException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.error("Error al leer un comando de la cola createQueue");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.error("Error al guardar el comando en la base de datos");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (ParseException e) {
        response.setResponse_status(RESPONSE_STATUS.ERROR);
        response.setMessage(error_message);
        logger.error("Error al actualizar alguno de los indices");
        logger.info(e.toString());
        e.printStackTrace();
      } finally {
        try {
          responseQueue.put(response);
```

```java
        } catch (InterruptedException e) {
          logger.error("No se pudo enviar la respuesta");
          logger.info(e.toString());
          e.printStackTrace();
        }
      }
    }
  }
}
```

```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.RemoteQueue;

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

  public static void main(String[] args) {
    PropertyConfigurator.configure(Constants.LOGGER_CONF);
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    try {
      ConfigLoader.getInstance().init(Constants.CONF_FILE);
      logger.info("Entablando conexion con el broker");
      RemoteQueue storageQueue = new RemoteQueue(
          ConfigLoader.getInstance().getStorageRequestQueueName(),
          ConfigLoader.getInstance().getStorageResquestQueueHost());
      storageQueue.init();
      logger.info("Disparando el storage controller");
      Thread storageControllerThread = new Thread(
          new StorageController(storageQueue));
      logger.info("Starteando el storage controller");
      storageControllerThread.start();
      logger.info("Joineando el storage controller");
      storageControllerThread.join();
    } catch (InterruptedException e) {
      logger.error("Error al joinear el storage controller");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error("Error al cargar la configuracion");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (TimeoutException e) {
      logger.error("Error al cerrar la cola storageQueue");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }
}
```

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

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

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

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

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

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

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

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

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

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

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

  public StorageController(BlockingQueue<Command> storageCommandQueue,
      RemoteQueue storageQueue) {
    this.storageCommandQueue = storageCommandQueue;
    this.storageQueue = storageQueue;
  }

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

    logger.info("Iniciando el storage controller");
    while (true) {
      try {
        command = storageCommandQueue.take();
        logger.info("Comando recibido con los siguientes parametros: "
            + "\nUsuario: " + command.getUser() + "\nComando: "
            + command.getCommand() + "\nMensaje: "
            + command.getMessage());
        storageQueue.put(command);
        logger.info("Comando enviado al storage");
      } catch (InterruptedException e) {
        logger.error(
            "Error al obtener el comando de la cola "
            + "storageCommandQueue");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        logger.error(
            "Error al insertar el comando de la cola storageQueue");
        logger.info(e.toString());
        e.printStackTrace();
      }
    }
  }
}
```

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

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

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

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

public class LoggerController implements Runnable {

  private BlockingQueue<Command> loggerCommandQueue;
  private RemoteQueue loggerQueue;

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

  public LoggerController(BlockingQueue<Command> loggerCommandQueue,
      RemoteQueue loggerQueue) {
    this.loggerCommandQueue = loggerCommandQueue;
    this.loggerQueue = loggerQueue;
  }

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

    logger.info("Iniciando el logger controller");
    while (true) {
      try {
        command = loggerCommandQueue.take();
        logger.info("Comando recibido con los siguientes parametros: "
            + "\nUsuario: " + command.getUser() + "\nComando: "
            + command.getCommand() + "\nMensaje: "
            + command.getMessage());
        loggerQueue.put(command);
        logger.info("Comando enviado al logger");
      } catch (InterruptedException e) {
        logger.error(
            "Error al obtener el comando de la cola "
            + "loggerCommandQueue");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        logger.error(
            "Error al insertar el comando de la cola loggerQueue");
        logger.info(e.toString());
        e.printStackTrace();
      }
    }
  }
}
```

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

import java.io.IOException;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
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.ConfigLoader;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.RemoteQueue;

public class Dispatcher implements Runnable {

  Thread analyzerControllerThread;
  Thread dispatcherControllerThread;
  Thread storageControllerThread;
  Thread loggerControllerThread;
  RemoteQueue dispatcherQueue;
  RemoteQueue storageQueue;
  RemoteQueue analyzerQueue;
  RemoteQueue loggerQueue;
  BlockingQueue<Command> storageCommandQueue;
  BlockingQueue<Command> analyzerCommandQueue;
  BlockingQueue<Command> loggerCommandQueue;
  ConfigLoader configLoader;
  final static Logger logger = Logger.getLogger(Dispatcher.class);

  public Dispatcher() {
    configLoader = ConfigLoader.getInstance();
  }

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

    logger.info("Iniciando el dispatcher");

    try {
      logger.info("Cargando la configuracion");
      configLoader.init(Constants.CONF_FILE);

      initDispatcher();
      startDispatcher();
      terminateDispatcher();

    } catch (InterruptedException e) {
      logger.error("Error al joinear los threads");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error("Error al cargar el archivo de configuracion");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (TimeoutException e) {
      logger.error("Error al iniciar las colas remotas");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }

  private void initDispatcher() throws IOException, TimeoutException {

    logger.info("Creando las colas internas");
    analyzerCommandQueue = new ArrayBlockingQueue<Command>(
        Constants.COMMAND_QUEUE_SIZE);
    storageCommandQueue = new ArrayBlockingQueue<Command>(
        Constants.COMMAND_QUEUE_SIZE);
    loggerCommandQueue = new ArrayBlockingQueue<Command>(
        Constants.COMMAND_QUEUE_SIZE);

    logger.info("Creando las conexiones a los brokers");
    logger.info("Creando la cola del dispatcher");
    dispatcherQueue = new RemoteQueue(configLoader.getDispatcherQueueName(),
        configLoader.getDispatcherQueueHost());
    dispatcherQueue.init();
    logger.info("Creando la cola hacia el analyzer");
    analyzerQueue = new RemoteQueue(configLoader.getAnalyzerQueueName(),
        configLoader.getAnalyzerQueueHost());
    analyzerQueue.init();
    logger.info("Creando la cola hacia el storage");
    storageQueue = new RemoteQueue(
        configLoader.getStorageRequestQueueName(),
        configLoader.getStorageResquestQueueHost());
    storageQueue.init();
    logger.info("Creando la cola hacia el logger");
    loggerQueue = new RemoteQueue(configLoader.getAuditLoggerQueueName(),
        configLoader.getAuditLoggerQueueHost());
    loggerQueue.init();

    logger.info("Creando los threads de los workers");
    analyzerControllerThread = new Thread(
        new AnalyzerController(analyzerCommandQueue, analyzerQueue));
    dispatcherControllerThread = new Thread(
        new DispatcherController(dispatcherQueue, storageCommandQueue,
            analyzerCommandQueue, loggerCommandQueue));
    storageControllerThread = new Thread(
        new StorageController(storageCommandQueue, storageQueue));
    loggerControllerThread = new Thread(
        new LoggerController(loggerCommandQueue, loggerQueue));

  }

  private void startDispatcher() {

    logger.info("Iniciando los threads de los workers");
    analyzerControllerThread.start();
    dispatcherControllerThread.start();
    storageControllerThread.start();
    loggerControllerThread.start();

  }

  private void terminateDispatcher() throws InterruptedException {

    logger.info("Joineando los threads de los workers");
    analyzerControllerThread.join();
    dispatcherControllerThread.join();
    storageControllerThread.join();
    loggerControllerThread.join();
  }
}
```

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

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

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

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;
import com.rabbitmq.client.AMQP.BasicProperties;

import ar.fiuba.taller.common.Command;
import ar.fiuba.taller.common.RemoteQueue;
import ar.fiuba.taller.common.Response;

public class DispatcherController extends DefaultConsumer implements Runnable {

  RemoteQueue dispatcherQueue;
  BlockingQueue<Command> storageCommandQueue;
  BlockingQueue<Command> analyzerCommandQueue;
  BlockingQueue<Command> loggerCommandQueue;
  final static Logger logger = Logger.getLogger(DispatcherController.class);

  public DispatcherController(RemoteQueue dispatcherQueue,
      BlockingQueue<Command> storageCommandQueue,
      BlockingQueue<Command> analyzerCommandQueue,
      BlockingQueue<Command> loggerCommandQueue) {
    super(dispatcherQueue.getChannel());
    this.storageCommandQueue = storageCommandQueue;
    this.analyzerCommandQueue = analyzerCommandQueue;
    this.loggerCommandQueue = loggerCommandQueue;
    this.dispatcherQueue = dispatcherQueue;
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    logger.info("Iniciando el dispatcher controller");
    // while(true) {
    try {
      dispatcherQueue.getChannel()
          .basicConsume(dispatcherQueue.getQueueName(), true, this);
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    // }
  }

  @Override
  public void handleDelivery(String consumerTag, Envelope envelope,
      BasicProperties properties, byte[] body) throws IOException {
    super.handleDelivery(consumerTag, envelope, properties, body);
    Command command = new Command();
    try {
      command.deserialize(body);
      logger.info("Comando recibido con los siguientes parametros: "
          + "\nUsuario: " + command.getUser() + "\nComando: "
          + command.getCommand() + "\nMensaje: "
          + command.getMessage());
      switch (command.getCommand()) {
      case PUBLISH:
        logger.info("Enviando mensaje a la cola del storage");
        storageCommandQueue.put(command);
        logger.info("Enviando mensaje a la cola del analyzer");
```

```java
          analyzerCommandQueue.put(command);
          logger.info("Enviando mensaje a la cola del logger");
          loggerCommandQueue.put(command);
          break;
        case QUERY:
          logger.info("Enviando mensaje a la cola del storage");
          storageCommandQueue.put(command);
          logger.info("Enviando mensaje a la cola del logger");
          loggerCommandQueue.put(command);
          break;
        case DELETE:
          logger.info("Enviando mensaje a la cola del storage");
          storageCommandQueue.put(command);
          logger.info("Enviando mensaje a la cola del logger");
          loggerCommandQueue.put(command);
          break;
        case FOLLOW:
          logger.info("Enviando mensaje a la cola del analyzer");
          analyzerCommandQueue.put(command);
          logger.info("Enviando mensaje a la cola del logger");
          loggerCommandQueue.put(command);
          break;
        default:
          logger.error("Comando invalido");
          break;
        }
      } catch (ClassNotFoundException e) {
        logger.info("Error al deserializar el comando");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        logger.info("Error al deserializar el comando");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (InterruptedException e) {
        logger.error("Error al insertar el comando en alguna de las colas");
        logger.info(e.toString());
        e.printStackTrace();
      }
    }
}
```

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

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

import ar.fiuba.taller.common.Constants;

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

  public static void main(String[] args) {
    PropertyConfigurator.configure(Constants.LOGGER_CONF);
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    logger.info("Disparando el dispatcher");
    Thread dispatcherThread = new Thread(new Dispatcher());
    dispatcherThread.start();
    try {
      dispatcherThread.join();
    } catch (InterruptedException e) {
      logger.error("Error al joinear el dispatcher");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }
}
```

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

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

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

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

public class AnalyzerController implements Runnable {
  private BlockingQueue<Command> analyzerCommandQueue;
  private RemoteQueue analyzerQueue;

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

  public AnalyzerController(BlockingQueue<Command> analyzerCommandQueue,
      RemoteQueue analyzerQueue) {
    this.analyzerCommandQueue = analyzerCommandQueue;
    this.analyzerQueue = analyzerQueue;
  }

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

    logger.info("Iniciando el analyzer controller");
    while (true) {
      try {
        command = analyzerCommandQueue.take();
        logger.info("Comando recibido con los siguientes parametros: "
            + "\nUsuario: " + command.getUser() + "\nComando: "
            + command.getCommand() + "\nMensaje: "
            + command.getMessage());
        analyzerQueue.put(command);
        logger.info("Comando enviado al analyzer");
      } catch (InterruptedException e) {
        logger.error(
            "Error al obtener el comando de la cola "
            + "analyzerCommandQueue");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        logger.error(
            "Error al insertar el comando de la cola analyzerQueue");
        logger.info(e.toString());
        e.printStackTrace();
      }
    }
  }
}
```

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

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

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

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

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

abr 02, 17 9:12 **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!" );
    }
}
```

abr 10, 17 5:27 **AppTest.java** Page 1/1

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

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

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

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

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

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

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

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

public class Response implements Serializable, ISerialize {

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

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

  public Response() {
    super();
    this.uuid = null;
    this.response_status = null;
    this.message = null;
  }

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

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

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

  public UUID getUuid() {
    return uuid;
  }

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

```java
  }

  public RESPONSE_STATUS getResponse_status() {
    return response_status;
  }

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

  public String getMessage() {
    return message;
  }

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

  public String getUser() {
    return user;
  }

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

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

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

import com.rabbitmq.client.*;

public class RemoteQueue {
  private String queueName;
  private String host;
  private ConnectionFactory factory;
  private Connection connection;
  private Channel channel;

  public Channel getChannel() {
    return channel;
  }

  public RemoteQueue(String queueName, String host) {
    this.queueName = queueName;
    this.host = host;
  }

  public String getQueueName() {
    return queueName;
  }

  public void setQueueName(String queueName) {
    this.queueName = queueName;
  }

  public String getHost() {
    return host;
  }

  public void setHost(String host) {
    this.host = host;
  }

  public void init() throws IOException, TimeoutException {
    factory = new ConnectionFactory();
    factory.setHost(host);
    connection = factory.newConnection();
    channel = connection.createChannel();
    channel.queueDeclareNoWait(queueName, false, false, false, null);
  }

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

  public void put(ISerialize message) throws IOException {
    channel.basicPublish("", queueName, null, message.serialize());
  }

}
```

```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;

}
```

```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 {

  public static final int COMMAND_QUEUE_SIZE = 100;
  public static final int RESPONSE_QUEUE_SIZE = 100;
  public static final String LOGGER_CONF = "conf/log4j.properties";
  public static final String COMMAND_SCRIPT_FOLDER = "scripts";
  public static final String COMMAND_SCRIPT_EXTENSION = ".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 = "conf/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 RESPONSE_QUEUE_HOST = "responseQueueHost";
  public static final String RESPONSE_QUEUE_NAME = "responseQueueName";
  public static final String DISPATCHER_QUEUE_HOST = "dispatcherQueueHost";
  public static final String DISPATCHER_QUEUE_NAME = "dispatcherQueueName";
  public static final String AUDIT_LOGGER_QUEUE_HOST = "auditLoggerQueueHost";
  public static final String AUDIT_LOGGER_QUEUE_NAME = "auditLoggerQueueName";
  public static final String STORAGE_REQUEST_QUEUE_HOST = "storageResquestQueueHost";
  public static final String STORAGE_REQUEST_QUEUE_NAME = "storageRequestQueueName";
  public static final String STORAGE_RESPONSE_QUEUE_HOST = "storageResponseQueueHost"
;
  public static final String STORAGE_RESPONSE_QUEUE_NAME = "storageResponseQueueName
";
  public static final String ANALYZER_QUEUE_HOST = "analyzerQueueHost";
  public static final String ANALYZER_QUEUE_NAME = "analyzerQueueName";
  public static final String USERS_SERVER = "usersServer";
  public static final String SHARDING_FACTOR = "shardingFactor";
  public static final String AUDIT_LOG_FILE = "log/audit.log";
  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 QUERY_COUNT_SHOW_POSTS = "queryCountShowPosts";
  public static final String TT_COUNT_SHOW_POST = "ttCountShowPosts";
  public static final String USER_READ_MODE = "r";
  public static final String USER_WRITE_MODE = "w";

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

  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);
```

```java
  }

  public static enum RESPONSE_STATUS {
    OK, ERROR, REGISTERED
  }

  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);
  }
}
```

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

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class ConfigLoader {

  private static ConfigLoader instance = null;
  private String responseQueueHost;
  private String responseQueueName;
  private String dispatcherQueueHost;
  private String dispatcherQueueName;
  private String auditLoggerQueueHost;
  private String auditLoggerQueueName;
  private String storageResquestQueueHost;
  private String storageRequestQueueName;
  private String storageResponseQueueHost;
  private String storageResponseQueueName;
  private String analyzerQueueHost;
  private String analyzerQueueName;
  private String usersServer;
  private int queryCountShowPosts;
  private int ttCountShowPosts;
  private int shardingFactor;

  public int getShardingFactor() {
    return shardingFactor;
  }

  protected ConfigLoader() {
    // TODO Auto−generated constructor stub
  }

  public static ConfigLoader getInstance() {
    if (instance ≡ null) {
      instance = new ConfigLoader();
    }
    return instance;
  }

  public void init(String configFile) throws IOException {
    Properties properties = new Properties();
    FileInputStream input = new FileInputStream(configFile);

    // cargamos el archivo de propiedades
    properties.load(input);

    // obtenemos las propiedades
    responseQueueHost = properties
        .getProperty(Constants.RESPONSE_QUEUE_HOST);
    responseQueueName = properties
        .getProperty(Constants.RESPONSE_QUEUE_NAME);
    dispatcherQueueHost = properties
        .getProperty(Constants.DISPATCHER_QUEUE_HOST);
    dispatcherQueueName = properties
        .getProperty(Constants.DISPATCHER_QUEUE_NAME);
    auditLoggerQueueHost = properties
        .getProperty(Constants.AUDIT_LOGGER_QUEUE_HOST);
    auditLoggerQueueName = properties
        .getProperty(Constants.AUDIT_LOGGER_QUEUE_NAME);
    storageResquestQueueHost = properties
        .getProperty(Constants.STORAGE_REQUEST_QUEUE_HOST);
    storageRequestQueueName = properties
        .getProperty(Constants.STORAGE_REQUEST_QUEUE_NAME);
    storageResponseQueueHost = properties
```

```java
        .getProperty(Constants.STORAGE_RESPONSE_QUEUE_HOST);
    storageResponseQueueName = properties
        .getProperty(Constants.STORAGE_RESPONSE_QUEUE_NAME);
    analyzerQueueHost = properties
        .getProperty(Constants.ANALYZER_QUEUE_HOST);
    analyzerQueueName = properties
        .getProperty(Constants.ANALYZER_QUEUE_NAME);
    usersServer = properties.getProperty(Constants.USERS_SERVER);
    shardingFactor = Integer
        .parseInt(properties.getProperty(Constants.SHARDING_FACTOR));
    queryCountShowPosts = Integer.parseInt(
        properties.getProperty(Constants.QUERY_COUNT_SHOW_POSTS));
    ttCountShowPosts = Integer
        .parseInt(properties.getProperty(Constants.TT_COUNT_SHOW_POST));
  }

  public String getResponseQueueHost() {
    return responseQueueHost;
  }

  public String getResponseQueueName() {
    return responseQueueName;
  }

  public String getDispatcherQueueHost() {
    return dispatcherQueueHost;
  }

  public String getDispatcherQueueName() {
    return dispatcherQueueName;
  }

  public String getAuditLoggerQueueHost() {
    return auditLoggerQueueHost;
  }

  public String getAuditLoggerQueueName() {
    return auditLoggerQueueName;
  }

  public String getStorageResquestQueueHost() {
    return storageResquestQueueHost;
  }

  public String getStorageRequestQueueName() {
    return storageRequestQueueName;
  }

  public String getStorageResponseQueueHost() {
    return storageResponseQueueHost;
  }

  public String getStorageResponseQueueName() {
    return storageResponseQueueName;
  }

  public String getAnalyzerQueueHost() {
    return analyzerQueueHost;
  }

  public String getAnalyzerQueueName() {
    return analyzerQueueName;
  }

  public int getQueryCountShowPosts() {
    return queryCountShowPosts;
  }

  public int getTtCountShowPosts() {
    return ttCountShowPosts;
  }

  public String getUsersServer() {
    return usersServer;
  }
}
```

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

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

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

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

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

  public Command() {
    this.command = null;
    this.user = null;
    this.message = null;
    this.uuid = null;
    this.timestamp = null;
  }

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

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

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

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

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

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

import ar.fiuba.taller.common.*;

public class UserConsole implements Runnable {
  private String username;
  private BlockingQueue<Command> commandQueue;
  private BlockingQueue<Response> responseQueue;
  private Thread scriptReaderThread;
  private Thread commandControllerThread;
  private Thread eventViewerThread;
  private Thread responseControllerThread;
  private RemoteQueue remoteUserResponseQueue;
  private RemoteQueue dispatcherQueue;
  private ConfigLoader configLoader = ConfigLoader.getInstance();
  private String mode;
  final static Logger logger = Logger.getLogger(UserConsole.class);

  public UserConsole(String username) {
    this.username = username;
  }

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

    try {
      logger.info("Iniciando usuario.");
      logger.info("Cargando la configuracion");
      configLoader.init(Constants.CONF_FILE);
      initUser();
      startUser();
      terminateUser();
    } catch (InterruptedException e) {
      logger.error("Error al joinear los threads");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error(
          "Error al cargar la configuracion o crear las colas remotas");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (TimeoutException e) {
      // TODO Auto−generated catch block
      e.printStackTrace();
    }
  }

  private void initUser() throws IOException, TimeoutException {
    logger.info("Creando cola de comandos leidos");
    commandQueue = new ArrayBlockingQueue<Command>(
        Constants.COMMAND_QUEUE_SIZE);
    logger.info("Creando cola del dispatcher");
    dispatcherQueue = new RemoteQueue(configLoader.getDispatcherQueueName(),
        configLoader.getDispatcherQueueHost());
    dispatcherQueue.init();
    logger.info("Creando lector de scripts");
    scriptReaderThread = new Thread(
        new ScriptReader(commandQueue,
            Constants.COMMAND_SCRIPT_FOLDER + "/" + username
                + Constants.COMMAND_SCRIPT_EXTENSION,
            username));
    logger.info("Creando controlador de comandos");
    commandControllerThread = new Thread(
        new CommandController(commandQueue, dispatcherQueue));
    logger.info("Creando cola de respuestas");
    responseQueue = new ArrayBlockingQueue<Response>(
        Constants.RESPONSE_QUEUE_SIZE);
    logger.info("Creando cola remota de respuestas");
    remoteUserResponseQueue = new RemoteQueue(username,
        configLoader.getResponseQueueHost());
    remoteUserResponseQueue.init();
    logger.info("Creando el controlador de respuestas");
    responseControllerThread = new Thread(
        new ResponseController(responseQueue, remoteUserResponseQueue));
    logger.info("Creando el visor de eventos");
    eventViewerThread = new Thread(new EventViewer(responseQueue, username,
        Constants.LOGS_DIR + "/" + username
            + Constants.EVENT_VIEWER_FILE_EXTENSION));
  }

  private void startUser() {
    logger.info("Iniciando el lector de scripts");
    scriptReaderThread.start();
    logger.info("Iniciando el controlador de comandos");
    commandControllerThread.start();
    logger.info("Iniciando el controlador de respuestas");
    responseControllerThread.start();
    logger.info("Iniciando el visor de eventos");
    eventViewerThread.start();
  }

  private void terminateUser()
      throws InterruptedException, IOException, TimeoutException {
    logger.info("Esperando al controlador de comandos");
    commandControllerThread.join();
    logger.info("controller finalizado!");
    logger.info("Esperando al reader");
    scriptReaderThread.join();
    logger.info("Reader finalizado!");
    logger.info("Esperando al controlador de respuestas");
    responseControllerThread.join();
    logger.info("controller controlador de respuestas!");
    logger.info("Esperando al visor de eventos");
    eventViewerThread.join();
    logger.info("visor de eventos finalizado!");
  }
}
```

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

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.concurrent.BlockingQueue;

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 ScriptReader implements Runnable {

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

  BlockingQueue<Command> commandQueue;
  String commandScript;
  String username;

  public ScriptReader(BlockingQueue<Command> commandQueue,
      String commandScript, String username) {
    this.commandQueue = commandQueue;
    this.commandScript = commandScript;
    this.username = username;
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    logger.info("Iniciando el script reader");
    try {
      JSONParser parser = new JSONParser();
      Object obj = parser.parse(new FileReader(commandScript));
      JSONObject jsonObject = (JSONObject) obj;
      JSONArray commandArray = (JSONArray) jsonObject
          .get(Constants.COMMAND_ARRAY);
      Iterator<JSONObject> iterator = commandArray.iterator();
      JSONObject commandObject;
      Command command;

      logger.info("Leyendo el command script: " + commandScript);
      while (iterator.hasNext()) {
        commandObject = iterator.next();
        command = new Command(
            (String) commandObject.get(Constants.COMMAND_KEY),
            username,
            (String) commandObject.get(Constants.MESSAGE_KEY), null,
            null);
        logger.info("Se inserto comando con los siguientes parametros: "
            + "\nUsuario: " + command.getUser() + "\nComando: "
            + command.getCommand() + "\nMensaje: "
            + command.getMessage());
        commandQueue.put(command);
      }
    } catch (InterruptedException e) {
      logger.error("Error al pushear comandos en la cola");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (FileNotFoundException e) {
      logger.error("No se encontro el archivo de comandos");
```

```java
            logger.info(e.toString());
            e.printStackTrace();
        } catch (IOException e) {
            logger.error("Error al leer el archivo de comandos");
            logger.info(e.toString());
            e.printStackTrace();
        } catch (ParseException e) {
            logger.error("Error al parsear el archivo de comandos");
            logger.info(e.toString());
            e.printStackTrace();
        }
    }
}
```

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

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

import com.rabbitmq.client.AMQP.BasicProperties;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;

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

public class ResponseController extends DefaultConsumer implements Runnable {

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

  public ResponseController(BlockingQueue<Response> responseQueue,
      RemoteQueue remoteResponseQueue) {
    super(remoteResponseQueue.getChannel());
    this.responseQueue = responseQueue;
    this.remoteResponseQueue = remoteResponseQueue;
  }

  @Override
  public void handleDelivery(String consumerTag, Envelope envelope,
      BasicProperties properties, byte[] body) throws IOException {
    super.handleDelivery(consumerTag, envelope, properties, body);
    Response response = new Response();
    try {
      response.deserialize(body);
      logger.info("Respuesta recibida con los siguientes valores: "
          + "\nUUID:" + response.getUuid() + "\nStatus:"
          + response.getResponse_status() + "\nMensaje:"
          + response.getMessage());
      responseQueue.put(response);
      logger.info("Respuesta pusheada en la cola responseQueue");
    } catch (ClassNotFoundException e) {
      logger.error("Error al deserializar la respuesta");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error("Error al deserializar la respuesta");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (InterruptedException e) {
      logger.error(
          "Error al insertar la respuesta en la cola responseQueue");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    logger.info("Iniciando el response controller");
    try {
      remoteResponseQueue.getChannel().basicConsume(
          remoteResponseQueue.getQueueName(), true, this);
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      }
    }

}
```

```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.concurrent.BlockingQueue;

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

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

public class EventViewer implements Runnable {
  BlockingQueue<Response> responseQueue;
  String username;
  String eventFile;
  final static Logger logger = Logger.getLogger(EventViewer.class);

  public EventViewer(BlockingQueue<Response> responseQueue, String username,
      String eventFile) {
    this.responseQueue = responseQueue;
    this.username = username;
    this.eventFile = eventFile;
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    Response response = null;
    FileWriter responseFile = null;
    PrintWriter pw;

    logger.info("Iniciando el event viewer");
    try {
      while (true) {
        logger.info("Esperando respuesta");
        response = responseQueue.take();
        pw = new PrintWriter(
            new BufferedWriter(new FileWriter(eventFile, true)));
        logger.info("Respuesta obtenida");
        pw.println("Evento recibido:\nUUID: " + response.getUuid()
            + "\nResponse Status: " + response.getResponse_status()
            + "\nMessage: " + response.getMessage());
        pw.println(
            "---------------------------------------------------"
            + "----------------------------------------------");
        pw.close();
      }
    } catch (InterruptedException e) {
      logger.error(
          "Error al tomar la respuesta en la cola responseQueue");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error("Error al abrir el archivo " + eventFile);
      logger.info(e.toString());
      e.printStackTrace();
    } finally {
      try {
        if (null ≠ responseFile)
          responseFile.close();
      } catch (Exception e2) {
        logger.error("Error al cerrar el archivo " + eventFile);
        logger.info(e2.toString());
        e2.printStackTrace();
      }
    }

  }

}
```

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

import java.io.IOException;
import java.lang.invoke.ConstantCallSite;
import java.sql.Timestamp;
import java.util.UUID;
import java.util.concurrent.BlockingQueue;

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

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

public class CommandController implements Runnable {
  private BlockingQueue<Command> commandQueue;
  private RemoteQueue dispatcherQueue;
  Timestamp timestamp;

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

  public CommandController(BlockingQueue<Command> commandQueue,
      RemoteQueue dispatcherQueue) {
    this.commandQueue = commandQueue;
    this.dispatcherQueue = dispatcherQueue;
  }

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

    logger.info("Iniciando el command controller");

    try {
      logger.info("Obteniendo comando de la cola");
      command = commandQueue.take();
      logger.info("Comando obtenido");
      logger.info("Comando recibido: " + command.getCommand());
      logger.info("Mensaje: " + command.getMessage());
      if (command.getMessage().length() ≤ 141) {
        logger.info("Generando UUID");
        command.setUuid(UUID.randomUUID());
        logger.info("Generando timestamp");
        timestamp = new Timestamp(System.currentTimeMillis());
        command.setTimestamp(Constants.SDF.format(timestamp));
        logger.info("UUID generado: " + command.getUuid());
        logger.info("Enviando el mensaje al dispatcher");
        dispatcherQueue.put(command);
        logger.info("Mensaje enviado");
      } else {
        logger.error("El mensaje contiene mas de 141 caracteres");
      }
    } catch (InterruptedException e) {
      logger.error("Error al sacar un comando de la cola commandQueue");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error("Error al enviar el mensaje al dispatcher");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }
}
```

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

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;

import org.apache.log4j.Logger;
import org.apache.log4j.MDC;
import org.apache.log4j.PropertyConfigurator;
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 App {
  final static Logger logger = Logger.getLogger(App.class);

  public static void main(String[] args) {
    PropertyConfigurator.configure(Constants.LOGGER_CONF);
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    List<Thread> usersList = new ArrayList<Thread>();
    String username, mode, reg;

    logger.info("Se inicia una nueva instancia de ClientConsole");

    try {
      // Obtengo los usuarios y los pongo a ejecutar el script
      JSONParser parser = new JSONParser();
      Object obj = parser.parse(new FileReader(Constants.USERS_FILE));
      JSONObject jsonObject = (JSONObject) obj;
      JSONArray arr = (JSONArray) jsonObject.get(Constants.USERS_KEY);
      Thread userConsoleThread;
      logger.info("Leyendo el archivo de usuarios a simular");
      Iterator<String> iterator = arr.iterator();
      StringTokenizer st;
      while (iterator.hasNext()) {
        username = iterator.next();
        logger.info("Siguiente usuario a crear: " + username);
        userConsoleThread = new Thread(new UserConsole(username));
        userConsoleThread.start();
        usersList.add(userConsoleThread);
        logger.info(
            "Usuario " + userConsoleThread.getId() + " creado!");
      }

      // Espero a que los usuarios hayan terminado de ejecutar
      logger.info("Esperando a que los usuarios terminen: "
          + usersList.size());
      for (Thread userThread : usersList) {
        userThread.join();
        logger.info("Usuario " + userThread.getId() + " finalizado!");
      }
    } catch (InterruptedException e) {
      logger.error("Error al joinear los threads de usuarios");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (FileNotFoundException e) {
      logger.error("No se encontro el archivo de usuarios");
      logger.info(e.toString());
      e.printStackTrace();
```

```java
    } catch (IOException e) {
      logger.error("Error al leer el archivo de usuarios");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (ParseException e) {
      logger.error("Error al parsear el archivo de usuarios");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }
}
```

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

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

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

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

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

```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.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.BlockingQueue;

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

import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;
import com.rabbitmq.client.AMQP.BasicProperties;

import ar.fiuba.taller.common.*;

public class AuditLogger extends DefaultConsumer implements Runnable {

  private Timestamp timestamp;
  private RemoteQueue loggerQueue;
  final static Logger logger = Logger.getLogger(AuditLogger.class);

  public AuditLogger(RemoteQueue loggerQueue) {
    super(loggerQueue.getChannel());
    ConfigLoader.getInstance();
    this.loggerQueue = loggerQueue;
  }

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

    logger.info("Iniciando el audit logger");
    try {
      PrintWriter pw = new PrintWriter(Constants.AUDIT_LOG_FILE, "UTF-8");
      pw.close();
      loggerQueue.getChannel().basicConsume(loggerQueue.getQueueName(),
          true, this);
    } catch (IOException e) {
      logger.error("Error consumir de la cola remota");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }

  @Override
  public void handleDelivery(String consumerTag, Envelope envelope,
      BasicProperties properties, byte[] body) throws IOException {
    super.handleDelivery(consumerTag, envelope, properties, body);
    PrintWriter pw = new PrintWriter(new BufferedWriter(
        new FileWriter(Constants.AUDIT_LOG_FILE, true)));
    Command command = new Command();
    try {
      command.deserialize(body);
      logger.info("Comando recibido con los siguientes parametros: "
          + "\nUsuario: " + command.getUser() + "\nComando: "
          + command.getCommand() + "\nMensaje: "
          + command.getMessage());
      logger.info("Escribiendo el mensaje en el archivo de log "
          + Constants.AUDIT_LOG_FILE);
      logger.info(getAuditLogEntry(command));
      pw.println(getAuditLogEntry(command));
      pw.close();
    } catch (ClassNotFoundException e) {
      logger.error("Error al deserializar el comando");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error("Error al deserializar el comando");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }

  private String getAuditLogEntry(Command command) {
    timestamp = new Timestamp(System.currentTimeMillis());
    return Constants.SDF.format(timestamp) + " − " + "UUID: "
        + command.getUuid() + " − Usuario: " + command.getUser()
        + " − Comando: " + command.getCommand() + " − Mensaje: "
        + command.getMessage();
  }

}
```

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

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.RemoteQueue;

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

  public static void main(String[] args) throws Exception {
    PropertyConfigurator.configure(Constants.LOGGER_CONF);
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    try {
      ConfigLoader.getInstance().init(Constants.CONF_FILE);
      logger.info("Conectando a la cola remota loggerQueue");
      RemoteQueue loggerQueue = new RemoteQueue(
          ConfigLoader.getInstance().getAuditLoggerQueueName(),
          ConfigLoader.getInstance().getAuditLoggerQueueHost());
      loggerQueue.init();
      Thread auditLoggerThread = new Thread(new AuditLogger(loggerQueue));
      logger.info("Disparando el audit logger");
      auditLoggerThread.start();
      auditLoggerThread.join();
    } catch (InterruptedException e) {
      logger.error("Error al joinear el audit logger");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (IOException e) {
      logger.error("Error al cargar la configuracion");
      logger.info(e.toString());
      e.printStackTrace();
    } catch (TimeoutException e) {
      logger.error("Error iniciar la cola remota");
      logger.info(e.toString());
      e.printStackTrace();
      throw new Exception();
    }
  }
}
```

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

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

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

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

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

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

import java.io.BufferedReader;
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() {
    // TODO Auto-generated constructor stub
  }

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

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

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

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

```
        JSONArray ar2 = new JSONArray();
        ar2.add(follower);
        jsonObject.put(updateKey, ar2);
      } else {
        array.add(follower);
        jsonObject.put(updateKey, array);
      }
      file = new FileWriter(tmpFile);
      try {
        file.write(jsonObject.toJSONString());
      } catch (Exception e) {
        logger.error("Error al guardar el index");
        logger.info(e.toString());
        e.printStackTrace();
      } finally {
        file.flush();
        file.close();
      }
    }

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

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

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

      System.out.println(array.toJSONString());

      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));
        logger.info("Topic sin #: " + word);
        jsonArray = (JSONArray) jsonObject.get(word);
        logger.info("arr: " + jsonArray);
        it = jsonArray.iterator();
        while (it.hasNext()) {
          followersList.add(it.next());
        }
      }
      return followersList;
    }
  }
```

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

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

import ar.fiuba.taller.common.Constants;

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

  public static void main(String[] args) {
    PropertyConfigurator.configure(Constants.LOGGER_CONF);
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    try {
      logger.info("Comenzando el analyzer");
      Thread analyzerThread = new Thread(new Analyzer());
      analyzerThread.start();
      analyzerThread.join();
    } catch (InterruptedException e) {
      logger.error("Error al joinear el analyzer");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }
}
```

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

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

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

import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;
import com.rabbitmq.client.AMQP.BasicProperties;

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

public class AnalyzerReciver extends DefaultConsumer implements Runnable {

  BlockingQueue<Response> responseQueue;
  Command command;
  Response response;
  UserRegistry userRegistry;
  RemoteQueue analyzerQueue;
  final static Logger logger = Logger.getLogger(AnalyzerReciver.class);

  public AnalyzerReciver(BlockingQueue<Response> responseQueue,
      RemoteQueue analyzerQueue, UserRegistry userRegistry) {
    super(analyzerQueue.getChannel());
    this.responseQueue = responseQueue;
    this.userRegistry = userRegistry;
    this.analyzerQueue = analyzerQueue;
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    logger.info("Iniciando el analyzer reciver");
    logger.info("Me pongo a comer de la cola: " + analyzerQueue.getHost()
        + " " + analyzerQueue.getQueueName());
    try {
      analyzerQueue.getChannel()
          .basicConsume(analyzerQueue.getQueueName(), true, this);
    } catch (IOException e) {
      logger.error("Error al comer de la cola");
      logger.info(e.toString());
      e.printStackTrace();
    }
  }

  @Override
  public void handleDelivery(String consumerTag, Envelope envelope,
      BasicProperties properties, byte[] body) throws IOException {
    super.handleDelivery(consumerTag, envelope, properties, body);
    Command command = new Command();
    try {
      command.deserialize(body);
      logger.info("Comando recibido con los siguientes parametros: "
          + "\nUUID: " + command.getUuid() + "\nUsuario: "
          + command.getUser() + "\nComando: " + command.getCommand()
          + "\nMensaje: " + command.getMessage());

      switch (command.getCommand()) {
      case PUBLISH:
        logger.info(
            "Comando recibido: PUBLISH. Insertando en la cola del "
```

```java
              + "analyzer dispatcher.");
          response = new Response();
          response.setUuid(command.getUuid());
          response.setUser(command.getUser());
          // Puede ser que de error en caso de hacer el update, entonces
          // hay que
          // mandarle error al usuario
          response.setResponse_status(RESPONSE_STATUS.OK);
          response.setMessage(command.getTimestamp() + "\n"
              + command.getUser() + "\n" + command.getMessage());
          responseQueue.put(response);
          break;
        case FOLLOW:
          logger.info(
              "Comando recibido: FOLLOW. Actualizando el user "
              + "registry.");
          userRegistry.update(command.getUser(), command.getMessage());
          response = new Response();
          response.setUuid(command.getUuid());
          response.setUser(command.getUser());
          response.setResponse_status(RESPONSE_STATUS.REGISTERED);
          response.setMessage("Seguidor registrado");
          responseQueue.put(response);
          break;
        default:
          logger.info("Comando recibido invalido. Comando descartado.");
        }
      } catch (ClassNotFoundException e) {
        logger.error("Error al deserializar el comando");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (IOException e) {
        logger.error("Error al deserializar el comando");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (InterruptedException e) {
        logger.error("Error al insertar el comando en alguna de las colas");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (ParseException e) {
        logger.error("Error al actualizar la base de usuarios");
        logger.info(e.toString());
        e.printStackTrace();
      }
    }

}
```

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

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

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

import ar.fiuba.taller.common.ConfigLoader;
import ar.fiuba.taller.common.Constants;
import ar.fiuba.taller.common.RemoteQueue;
import ar.fiuba.taller.common.Response;

public class Analyzer implements Runnable {

  private Thread analyzerDispatcherThread;
  private Thread analyzerReciverThread;
  private Thread responseControllerThread;
  private BlockingQueue<Response> responseQueue;
  private UserRegistry userRegistry;
  private ConfigLoader configLoader;
  private RemoteQueue analyzerQueue;
  final static Logger logger = Logger.getLogger(Analyzer.class);

  public Analyzer() {
    configLoader = ConfigLoader.getInstance();
  }

  public void run() {
    MDC.put("PID", String.valueOf(Thread.currentThread().getId()));
    try {
      configLoader.init(Constants.CONF_FILE);
      // Instancio la cola
      responseQueue = new ArrayBlockingQueue<Response>(
          Constants.COMMAND_QUEUE_SIZE);
      logger.debug(Constants.ANALYZER_QUEUE_NAME);
      logger.debug(Constants.ANALYZER_QUEUE_HOST);
      // Creo la cola remota en donde el anayzer recibe los comandos
      analyzerQueue = new RemoteQueue(
          ConfigLoader.getInstance().getAnalyzerQueueName(),
          ConfigLoader.getInstance().getAnalyzerQueueHost());
      analyzerQueue.init();

      // Instancio el registry
      userRegistry = new UserRegistry();

      // Hago una carga inicial del user registry

      // Instancio los threads
      analyzerReciverThread = new Thread(new AnalyzerReciver(
          responseQueue, analyzerQueue, userRegistry));
      analyzerDispatcherThread = new Thread(
          new AnalyzerDispatcher(responseQueue, userRegistry));

      // Inicio los threads
      analyzerReciverThread.start();
      analyzerDispatcherThread.start();

      // Me quedo esperando los threads
      analyzerReciverThread.join();
      analyzerDispatcherThread.join();

    } catch (IOException e) {
      logger.error("Error al cargar el archivo de configuracion");
```

```java
        logger.info(e.toString());
        e.printStackTrace();
      } catch (InterruptedException e) {
        logger.error("Error al dormir el thread");
        logger.info(e.toString());
        e.printStackTrace();
      } catch (TimeoutException e) {
        logger.error("Error al iniciar la cola remota");
        logger.info(e.toString());
        e.printStackTrace();
      }
    }

  }
```

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

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

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

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

public class AnalyzerDispatcher implements Runnable {

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

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

  public void run() {
    while (true) {
      try {
        response = responseQueue.take();
        logger.info("Nueva respuesta para enviar");
        logger.info("Nueva respuesta para enviar");
        logger.info("UUID: " + response.getUuid());
        logger.info("User: " + response.getUser());
        logger.info("Status: " + response.getResponse_status());
        logger.info("Message: " + response.getMessage());
        // Reviso si es un user register o un mensaje
        // Si da error o es una registracion, se lo devuelvo solamente
        // al usuario que envio el request
        if (response.getResponse_status() ≡ RESPONSE_STATUS.REGISTERED
            ∨ response
                .getResponse_status() ≡ RESPONSE_STATUS.ERROR)
        {
          logger.info("Enviando respuesta");
          remoteQueue = getUserQueue(response.getUser());
          remoteQueue.put(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
```

```java
                .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())).put(response);
            }
          }
        } catch (InterruptedException e) {
          logger.error(
              "Error al tomar respuestas de la cola responseQueue");
          logger.info(e.toString());
          e.printStackTrace();
        } catch (IOException e) {
          logger.error(
              "Error al insertar respuesta en la cola remota del "
              + "usuario:" + response.getUser());
          logger.info(e.toString());
          e.printStackTrace();
        } catch (ParseException e) {
          logger.error("Error al updatear los indices");
          logger.info(e.toString());
          e.printStackTrace();
        } catch (TimeoutException e) {
          logger.error("No se pudo enviar el mensaje al follower");
          logger.info(e.toString());
          e.printStackTrace();
        }
      }
    }

    private RemoteQueue getUserQueue(String username)
        throws IOException, TimeoutException {
      RemoteQueue tmpQueue;
      tmpQueue = usersMap.get(username);

      if (tmpQueue ≡ null) {
        tmpQueue = new RemoteQueue(username,
            ConfigLoader.getInstance().getUsersServer());
        tmpQueue.init();
        usersMap.put(username, tmpQueue);
      }
      return usersMap.get(username);
    }
  }
```

Table of Contents