

Apr 25, 18 21:45

**UserStatisticsViewer.java**

Page 1/1

```

1  import com.google.gson.Gson;
2  import com.rabbitmq.client.*;
3
4  import java.io.IOException;
5  import java.util.concurrent.TimeoutException;
6
7  public class UserStatisticsViewer extends RabbitMQProcess {
8
9      public UserStatisticsViewer(String host) throws IOException, TimeoutException {
10         super(host);
11
12         // declare USERS_STATS exchange
13         getChannel().exchangeDeclare(Configuration.UsersStatisticsExchange,
14             BuiltinExchangeType.FANOUT);
15     }
16
17     @Override
18     public void run() throws IOException {
19         consumeStatistics();
20     }
21
22     private String consumeStatistics() throws IOException {
23         String statisticsQueue = getChannel().queueDeclare().getQueue();
24         getChannel().queueBind(statisticsQueue,
25             Configuration.UsersStatisticsExchange, "");
26
27         Consumer consumerStatistics = new DefaultConsumer(getChannel()) {
28             @Override
29             public void handleDelivery(String consumerTag, Envelope envelope,
30                 AMQP.BasicProperties properties,
31                 byte[] body) throws IOException {
32
33                 String json = new String(body, "UTF-8");
34                 UsersSecondsListenedStatistics statistics = new Gson().fromJson(
35                     json, UsersSecondsListenedStatistics.class);
36
37                 System.out.println("[x] Showing users who listened most: ");
38                 for (UserSecondsListened userStats :
39                     statistics.getUsersMostListenedSeconds()) {
40                     System.out.println(userStats.getUsername() + ": " +
41                         userStats.getSecondsListened());
42                 }
43             }
44         };
45
46         // consume de una cola temporal a traves de un exchange
47         // por lo que no tiene sentido ack manual
48         return getChannel().basicConsume(statisticsQueue, true,
49             consumerStatistics);
50     }
51
52     public static void main(String[] argv) throws Exception {
53         UserStatisticsViewer statisticsViewer =
54             new UserStatisticsViewer(Configuration.RabbitMQHost);
55         statisticsViewer.run();
56     }
57 }

```

Apr 25, 18 21:04

**UsersSecondsListenedStatistics.java**

Page 1/1

```

1  import java.util.Collection;
2  import java.util.LinkedList;
3  import java.util.List;
4
5  public class UsersSecondsListenedStatistics {
6
7      private List<UserSecondsListened> usersMostListenedSeconds;
8
9      public UsersSecondsListenedStatistics(Collection<UserSecondsListened> stats)
10     {
11         this.setUsersMostListenedSeconds(new LinkedList<>(stats));
12     }
13
14     public List<UserSecondsListened> getUsersMostListenedSeconds() {
15         return usersMostListenedSeconds;
16     }
17
18     public void setUsersMostListenedSeconds(List<UserSecondsListened> usersMostL
19         istenedSeconds) {
20         this.usersMostListenedSeconds = usersMostListenedSeconds;
21     }
22 }

```

Apr 25, 18 21:23 **UsersSecondsListenedComparator.java** Page 1/1

```

1  import java.util.Comparator;
2
3
4  public class UsersSecondsListenedComparator
5      implements Comparator<UserSecondsListened> {
6      @Override
7      public int compare(UserSecondsListened o1, UserSecondsListened o2) {
8
9          return (int) Math.signum(o2.getSecondsListened() -
10             o1.getSecondsListened());
11      }
12  }

```

Apr 25, 18 21:04 **UserSecondsListened.java** Page 1/1

```

1  public class UserSecondsListened {
2
3      private String username;
4      private long secondsListened;
5
6      public UserSecondsListened(String username, long secondsListened) {
7          this.setUsername(username);
8          this.setSecondsListened(secondsListened);
9      }
10
11     public String getUsername() {
12         return username;
13     }
14
15     public void setUsername(String username) {
16         this.username = username;
17     }
18
19     public long getSecondsListened() {
20         return secondsListened;
21     }
22
23     public void setSecondsListened(long secondsListened) {
24         this.secondsListened = secondsListened;
25     }
26 }

```

Apr 26, 18 2:51

## UsersDBHandler.java

Page 1/4

```

1  import com.google.gson.Gson;
2  import com.rabbitmq.client.*;
3
4  import java.io.IOException;
5  import java.util.*;
6  import java.util.concurrent.TimeoutException;
7
8  public class UsersDBHandler extends DBHandlerWithStatistics<User> {
9
10     private String queueName;
11
12     public UsersDBHandler(String host, Database<User> database,
13                          List<String> masks)
14         throws IOException, TimeoutException {
15         super(host, database);
16
17         // declare USERS_DB exchange
18         getChannel().exchangeDeclare(Configuration.UsersDBExchange,
19                                     BuiltinExchangeType.TOPIC);
20
21         // declare USERS_STATS exchange
22         getChannel().exchangeDeclare(Configuration.UsersStatisticsExchange,
23                                     BuiltinExchangeType.FANOUT);
24
25         this.queueName = Configuration.UsersDBExchange + "_" +
26                         Configuration.maskListToStr(masks);
27         getChannel().queueDeclare(queueName, true, false, false, null);
28         for (String mask : masks) {
29             getChannel().queueBind(queueName,
30                                   Configuration.UsersDBExchange, mask);
31         }
32     }
33
34     @Override
35     public void run() throws IOException {
36
37         Consumer usersConsumer = new DefaultConsumer(getChannel()) {
38             @Override
39             public void handleDelivery(String consumerTag, Envelope envelope,
40                                       AMQP.BasicProperties properties,
41                                       byte[] body) throws IOException {
42
43                 // parse request
44                 String jsonRequest = new String(body, "UTF-8");
45
46                 DatabaseRequest request = new Gson().fromJson(jsonRequest,
47                                                                DatabaseRequest.class);
48
49                 if (request.getType() == Configuration.UsersTypeConnect) {
50                     consumeConnection(request.getSerializedRequest());
51                 } else if (request.getType() ==
52                           Configuration.UsersTypeDisconnect) {
53                     consumeDisconnect(request.getSerializedRequest());
54                 } else if (request.getType() ==
55                           Configuration.UsersTypeKeepAlive) {
56                     consumeKeepAlive(request.getSerializedRequest());
57                 } else {
58                     Logger.output("Invalid request type received: " +
59                                  request.getType() + ", request: " +
60                                  request.getSerializedRequest());
61                 }
62
63                 getChannel().basicAck(envelope.getDeliveryTag(), false);
64             }
65         };
66
67         getChannel().basicConsume(queueName, false, usersConsumer);
68     }
69
70     @Override
71     protected List<StatisticTask> getStatistics() {
72         List<StatisticTask> operations = new LinkedList<>();
73     }

```

Apr 26, 18 2:51

## UsersDBHandler.java

Page 2/4

```

74     Runnable runnable = new Runnable() {
75         @Override
76         public void run() {
77             // get statistics
78             LimitedSortedSet<UserSecondsListened> usersMostListened =
79                 new LimitedSortedSet<>(Configuration.UserStatisticsN,
80                                       new UsersSecondsListenedComparator());
81             for (User row : database.getRows()) {
82                 UserSecondsListened userStats =
83                     new UserSecondsListened(row.username,
84                                             row.secondsListening);
85                 usersMostListened.add(userStats);
86             }
87
88             UsersSecondsListenedStatistics stats =
89                 new UsersSecondsListenedStatistics(usersMostListened);
90             String jsonStats = new Gson().toJson(stats);
91
92             try {
93                 getChannel().basicPublish(
94                     Configuration.UsersStatisticsExchange, "", null,
95                     jsonStats.getBytes());
96             } catch (IOException e) {
97                 Logger.output("IOException during statistics publish");
98             }
99         }
100     };
101
102     operations.add(new StatisticTask(runnable,
103                                     Configuration.UsersStatisticsPeriodSeconds));
104
105     return operations;
106 }
107
108 public void consumeConnection(String jsonRequest) throws IOException {
109
110     UserConnectRequest request = new Gson().fromJson(jsonRequest,
111                                                       UserConnectRequest.class);
112     UserConnectResponse response = new UserConnectResponse(request);
113
114     // get user record from DB
115     User user = database.getRow(request.getUsername());
116     if (user == null) {
117         System.out.println("User: " + request.getUsername() + " not " +
118                             "found, creating new user");
119         user = new User(request.getUsername());
120     }
121     // first check if any connection is not active and remove it
122     int connectionId = 0;
123     Date now = new Date();
124     ListIterator<UserRadioConnection> connIter =
125         user.connections.listIterator();
126     while(connIter.hasNext()) {
127         UserRadioConnection connection = connIter.next();
128
129         Date then = connection.keepAlive;
130         if (now.getTime() - then.getTime() >
131             Configuration.SecondsUntilDropConnection * 1000) {
132             UserDisconnectRequest disconnectRequest = new
133                 UserDisconnectRequest(request.getUsername(),
134                                     connection.radio,
135                                     connection.connectionID);
136             response.getClosedConnections().add(disconnectRequest);
137             connIter.remove();
138
139             System.out.println("[X] Closing old connection to: " +
140                                 connection.radio);
141         } else {
142             if (connection.connectionID > connectionId) {
143                 connectionId = connection.connectionID;
144             }
145         }
146     }

```

Apr 26, 18 2:51

## UsersDBHandler.java

Page 3/4

```

147     connectionId = (connectionId + 1) %
148         Configuration.MaxConnectionsPerUnlimitedUser;
149     // then check if user can connect to radio
150     if (user.connections.size() < user.connectionsLimit) {
151         UserRadioConnection connection =
152             new UserRadioConnection(response.getRadio(),
153                 new Date(), connectionId);
154         user.connections.add(connection);
155         response.setCouldConnect(true);
156         response.setConnectionId(connectionId);
157
158         System.out.println("[x] User: " + user.username +
159             " connected to: " + request.getRadio());
160     } else {
161         response.setCouldConnect(false);
162
163         System.out.println("[x] User: " + user.username +
164             " not connected to: " + request.getRadio());
165     }
166     // update user
167     database.updateRow(user);
168
169     String jsonResponse = new Gson().toJson(response);
170     getChannel().basicPublish("",
171         Configuration.ConnMgrUsersDBResponseQueue, null,
172         jsonResponse.getBytes());
173 }
174
175 public void consumeDisconnection(String jsonRequest) throws IOException {
176
177     UserDisconnectRequest request = new Gson().fromJson(jsonRequest,
178         UserDisconnectRequest.class);
179
180     // get user record from DB
181     User user = database.getRow(request.getUsername());
182     if (user != null) {
183         ListIterator<UserRadioConnection> connIter =
184             user.connections.listIterator();
185         while (connIter.hasNext()) {
186             UserRadioConnection connection = connIter.next();
187             if (connection.connectionID == request.getConnectionId() ^
188                 connection.radio.equals(request.getRadio())) {
189                 connIter.remove();
190                 System.out.println("[x] Removing connection " +
191                     "from: " + user.username + " to radio: "
192                     + connection.radio);
193                 break;
194             }
195         }
196     }
197     // update user
198     database.updateRow(user);
199 }
200
201 public void consumeKeepAlive(String jsonRequest) throws IOException {
202
203     KeepAliveRequest request = new Gson().fromJson(jsonRequest,
204         KeepAliveRequest.class);
205
206     // get user record from DB
207     User user = database.getRow(request.getUsername());
208     if (user == null) {
209         System.out.println("[x] Error: user who sent keep alive " +
210             "does not exist");
211         return;
212     }
213
214     // refresh keep alive
215     ListIterator<UserRadioConnection> connIter =
216         user.connections.listIterator();
217     while (connIter.hasNext()) {
218         UserRadioConnection connection = connIter.next();
219         if (connection.connectionID == request.getConnectionId() ^

```

Apr 26, 18 2:51

## UsersDBHandler.java

Page 4/4

```

220         connection.radio.equals(request.getRadio())) {
221             connection.keepAlive = new Date();
222             connIter.set(connection);
223             System.out.println("[x] Refreshing keepalive " +
224                 "from: " + user.username + " to radio: "
225                 + connection.radio + " id: " +
226                 connection.connectionID);
227             break;
228         }
229     }
230
231     // add to user total listened minutes
232     user.secondsListening += Configuration.KeepAlivePeriodSeconds;
233
234     // update user
235     database.updateRow(user);
236 }
237
238 public static void main(String[] argv) throws Exception {
239     if (argv.length < 1) {
240         System.out.println("Usage: UsersDBHandler mask1 mask2 mask3");
241         return;
242     }
243     List<String> masks = new LinkedList<>(Arrays.asList(argv));
244
245     // define database
246     Database<User> database = new DatabaseJson<>(
247         Configuration.UsersDBExchange + "-" +
248         Configuration.maskListToStr(masks), User.class);
249
250     // start database handler
251     UsersDBHandler handler = new UsersDBHandler(Configuration.RabbitMQHost,
252         database, masks);
253     handler.run();
254 }
255 }

```

Apr 25, 18 20:58

**UserRadioConnection.java**

Page 1/1

```

1  import java.util.Date;
2
3  public class UserRadioConnection {
4
5      public String radio;
6      // para diferenciar un usuario con varias conexiones a la misma radio
7      public int connectionID;
8      public Date keepAlive;
9
10     public UserRadioConnection(String radio, Date keepAlive,
11                                int connectionID){
12         this.radio = radio;
13         this.keepAlive = keepAlive;
14         this.connectionID = connectionID;
15     }
16 }

```

Apr 25, 18 20:58

**User.java**

Page 1/1

```

1  import java.util.LinkedList;
2  import java.util.List;
3
4  public class User extends DatabaseRow {
5
6      public String username;
7      public List<UserRadioConnection> connections = new LinkedList<>();
8      public long secondsListening;
9      public int connectionsLimit;
10
11     public User(String username) {
12         super(username);
13
14         this.username = username;
15         this.secondsListening = 0;
16         this.connectionsLimit = Configuration.MaxConnectionsPerFreeUser;
17     }
18 }

```

Apr 25, 18 21:04

**UserDisconnectRequest.java**

Page 1/1

```

1 public class UserDisconnectRequest {
2
3     // request
4     private String username;
5     private String radio;
6     private int connectionId;
7
8     public UserDisconnectRequest(String username, String radio, int
9         connectionId) {
10         this.setUsername(username);
11         this.setRadio(radio);
12         this.setConnectionId(connectionId);
13     }
14
15     public String toLogLine() {
16         return Configuration.LogsDisconnectionTag + " " + getUsername() + " " +
17             getRadio() + " " + getConnectionId();
18     }
19
20     public String getUsername() {
21         return username;
22     }
23
24     public void setUsername(String username) {
25         this.username = username;
26     }
27
28     public String getRadio() {
29         return radio;
30     }
31
32     public void setRadio(String radio) {
33         this.radio = radio;
34     }
35
36     public int getConnectionId() {
37         return connectionId;
38     }
39
40     public void setConnectionId(int connectionId) {
41         this.connectionId = connectionId;
42     }
43 }

```

Apr 25, 18 21:04

**UserConnectResponse.java**

Page 1/2

```

1 import java.util.LinkedList;
2 import java.util.List;
3
4 public class UserConnectResponse {
5
6     // request
7     private String username;
8     private String radio;
9
10    // id
11    private String returnQueueName;
12
13    // response
14    private boolean couldConnect = false;
15    private int connectionId;
16    private List<UserDisconnectRequest> closedConnections = new LinkedList<>();
17
18    public UserConnectResponse(UserConnectRequest request) {
19        this.setUsername(request.getUsername());
20        this.setRadio(request.getRadio());
21        this.setReturnQueueName(request.getReturnQueueName());
22    }
23
24    public String toLogLine() {
25        return Configuration.LogsConnectionTag + " " + getUsername() + " " +
26            getRadio() + " " + getConnectionId();
27    }
28
29    public String getUsername() {
30        return username;
31    }
32
33    public void setUsername(String username) {
34        this.username = username;
35    }
36
37    public String getRadio() {
38        return radio;
39    }
40
41    public void setRadio(String radio) {
42        this.radio = radio;
43    }
44
45    public String getReturnQueueName() {
46        return returnQueueName;
47    }
48
49    public void setReturnQueueName(String returnQueueName) {
50        this.returnQueueName = returnQueueName;
51    }
52
53    public boolean isCouldConnect() {
54        return couldConnect;
55    }
56
57    public void setCouldConnect(boolean couldConnect) {
58        this.couldConnect = couldConnect;
59    }
60
61    public int getConnectionId() {
62        return connectionId;
63    }
64
65    public void setConnectionId(int connectionId) {
66        this.connectionId = connectionId;
67    }
68
69    public List<UserDisconnectRequest> getClosedConnections() {
70        return closedConnections;
71    }
72
73    public void setClosedConnections(List<UserDisconnectRequest> closedConnectio

```

Apr 25, 18 21:04

**UserConnectResponse.java**

Page 2/2

```

ns) {
74     this.closedConnections = closedConnections;
75 }
76 }

```

Apr 26, 18 1:41

**UserConnectRequest.java**

Page 1/1

```

1
2 public class UserConnectRequest {
3
4     // request
5     private String username;
6     private String radio;
7
8     // id
9     private String returnQueueName;
10
11    public UserConnectRequest(String username, String radio,
12                               String returnQueueName) {
13        this.setUsername(username);
14        this.setRadio(radio);
15        this.setReturnQueueName(returnQueueName);
16    }
17
18    public UserConnectRequest(UserConnectResponse resp) {
19        this.username = resp.getUsername();
20        this.radio = resp.getRadio();
21        this.returnQueueName = resp.getReturnQueueName();
22    }
23
24    public String getUsername() {
25        return username;
26    }
27
28    public void setUsername(String username) {
29        this.username = username;
30    }
31
32    public String getRadio() {
33        return radio;
34    }
35
36    public void setRadio(String radio) {
37        this.radio = radio;
38    }
39
40    public String getReturnQueueName() {
41        return returnQueueName;
42    }
43
44    public void setReturnQueueName(String returnQueueName) {
45        this.returnQueueName = returnQueueName;
46    }
47 }

```

Apr 25, 18 22:21

**StatisticTask.java**

Page 1/1

```

1 public class StatisticTask {
2
3     private Runnable runnable;
4     private int period;
5
6     public StatisticTask(Runnable runnable, int period) {
7         this.runnable = runnable;
8         this.period = period;
9     }
10
11     public Runnable getRunnable() {
12         return runnable;
13     }
14
15     public void setRunnable(Runnable runnable) {
16         this.runnable = runnable;
17     }
18
19     public int getPeriod() {
20         return period;
21     }
22
23     public void setPeriod(int period) {
24         this.period = period;
25     }
26 }

```

Apr 26, 18 1:37

**RadiosUpdateRequest.java**

Page 1/1

```

1 public class RadiosUpdateRequest {
2
3     private String radio;
4     private String username;
5
6     public RadiosUpdateRequest(String radio, String username) {
7         this.setRadio(radio);
8         this.setUsername(username);
9     }
10
11     public RadiosUpdateRequest(UserConnectRequest req) {
12         this.radio = req.getRadio();
13         this.username = req.getUsername();
14     }
15
16     public RadiosUpdateRequest(UserDisconnectRequest req) {
17         this.radio = req.getRadio();
18         this.username = req.getRadio();
19     }
20
21     public String getRadio() {
22         return radio;
23     }
24
25     public void setRadio(String radio) {
26         this.radio = radio;
27     }
28
29     public String getUsername() {
30         return username;
31     }
32
33     public void setUsername(String username) {
34         this.username = username;
35     }
36 }

```



Apr 25, 18 21:45

## RadioStatisticsViewer.java

Page 1/1

```

1 import com.google.gson.Gson;
2 import com.rabbitmq.client.*;
3
4 import java.io.IOException;
5 import java.util.concurrent.TimeoutException;
6
7 public class RadioStatisticsViewer extends RabbitMQProcess {
8
9     public RadioStatisticsViewer(String host) throws IOException,
10         TimeoutException {
11         super(host);
12
13         // declare RADIOS_STATS exchange
14         getChannel().exchangeDeclare(Configuration.RadiosStatisticsExchange,
15             BuiltinExchangeType.FANOUT);
16     }
17
18     @Override
19     public void run() throws IOException {
20         consumeStatistics();
21     }
22
23     private String consumeStatistics() throws IOException {
24         String statisticsQueue = getChannel().queueDeclare().getQueue();
25         getChannel().queueBind(statisticsQueue,
26             Configuration.RadiosStatisticsExchange, "");
27
28         Consumer consumerStatistics = new DefaultConsumer(getChannel()) {
29             @Override
30             public void handleDelivery(String consumerTag, Envelope envelope,
31                 AMQP.BasicProperties properties,
32                 byte[] body) throws IOException {
33
34                 String json = new String(body, "UTF-8");
35                 RadiosConnectionsStatistics statistics = new Gson().fromJson(
36                     json, RadiosConnectionsStatistics.class);
37
38                 System.out.println("[x] Showing connections per radio: ");
39                 for (String radio : statistics.getRadioConnections().keySet()) {
40                     System.out.println(radio + ": " +
41                         statistics.getRadioConnections().get(radio));
42                 }
43             }
44         };
45         // consume de una cola temporal a traves de un exchange
46         // por lo que no tiene sentido ack manual
47         return getChannel().basicConsume(statisticsQueue,
48             true, consumerStatistics);
49     }
50
51     public static void main(String[] argv) throws Exception {
52         RadioStatisticsViewer statisticsViewer =
53             new RadioStatisticsViewer(Configuration.RabbitMQHost);
54         statisticsViewer.run();
55     }
56 }

```

Apr 26, 18 3:20

## RadioStation.java

Page 1/1

```

1 import com.rabbitmq.client.*;
2
3 import java.io.IOException;
4 import java.util.concurrent.*;
5
6 public class RadioStation extends RabbitMQProcess {
7
8     private String exchangeName;
9     private ScheduledExecutorService transmissionScheduler;
10    private ScheduledFuture<?> transmissionHandle;
11
12    private RadioSource source;
13
14    public RadioStation(String host, String radioName, RadioSource source) throws
15        s
16        IOException, TimeoutException {
17        super(host);
18
19        this.source = source;
20        source.init();
21
22        // declare BROADCAST exchange
23        exchangeName = Configuration.RadioExchangePrefix + radioName;
24        getChannel().exchangeDeclare(exchangeName, BuiltinExchangeType.FANOUT);
25    }
26
27    @Override
28    public void run() {
29        scheduleTransmission();
30    }
31
32    private void scheduleTransmission() {
33        transmissionScheduler = Executors
34            .newScheduledThreadPool(1);
35
36        transmissionHandle =
37            transmissionScheduler.scheduleAtFixedRate(new Runnable() {
38                @Override
39                public void run() {
40                    byte[] nextBlock = source.getNextByteBlock();
41
42                    try {
43                        getChannel().basicPublish(exchangeName, "",
44                            null, nextBlock);
45                    } catch (IOException e) {
46                        e.printStackTrace();
47                    }
48                }, Configuration.RadioSendPeriodMilliseconds,
49                    Configuration.RadioSendPeriodMilliseconds,
50                    TimeUnit.MILLISECONDS);
51    }
52
53    @Override
54    protected void close() throws IOException, TimeoutException {
55        super.close();
56
57        transmissionHandle.cancel(true);
58        transmissionScheduler.shutdown();
59        source.close();
60    }
61
62    public static void main(String[] argv) throws Exception {
63        RadioSource source = argv.length == 2 ?
64            new RadioSourceFile(argv[1]) :
65            new RadioSourceRandomNumbers();
66        RadioStation radio = new RadioStation(Configuration.RabbitMQHost, argv[0],
67            source);
68        radio.run();
69    }
70 }

```

Apr 26, 18 3:28

**RadioSourceRandomNumbers.java**

Page 1/1

```

1  import java.util.Base64;
2  import java.util.concurrent.ThreadLocalRandom;
3
4  public class RadioSourceRandomNumbers implements RadioSource {
5
6      @Override
7      public void init() {
8
9      }
10
11      @Override
12      public byte[] getNextByteBlock() {
13          int randomNum = ThreadLocalRandom.current().nextInt(0, 100 + 1);
14          Logger.output(" [x] Sent: " + randomNum);
15          return Base64.getEncoder().encode(Integer.toString(randomNum)
16              .getBytes());
17      }
18
19      @Override
20      public void close() {
21
22      }
23
24  }

```

Apr 12, 18 14:12

**RadioSource.java**

Page 1/1

```

1  import java.io.FileNotFoundException;
2
3  public interface RadioSource {
4
5      void init() throws FileNotFoundException;
6
7      byte[] getNextByteBlock();
8
9      void close();
10 }

```

Apr 26, 18 3:28

## RadioSourceFile.java

Page 1/1

```

1 import java.io.FileInputStream;
2 import java.io.FileNotFoundException;
3 import java.io.IOException;
4 import java.util.Base64;
5
6 public class RadioSourceFile implements RadioSource {
7
8     String filename;
9     FileInputStream audio;
10    int bytesPerRead;
11    byte[] buffer;
12
13    public RadioSourceFile(String filename) {
14        this.filename = filename;
15        this.bytesPerRead = 1000 * Configuration.RadioSendPeriodMilliseconds;
16        buffer = new byte[this.bytesPerRead];
17    }
18
19    @Override
20    public void init() throws FileNotFoundException {
21        audio = new FileInputStream(filename);
22    }
23
24    @Override
25    public byte[] getNextByteBlock() {
26        try {
27            int bytesSent = audio.read(buffer);
28            if (bytesSent == -1) {
29                audio.close();
30                audio = new FileInputStream(filename);
31            }
32            Logger.output("[X] Sent: " + bytesSent + " bytes");
33            return Base64.getEncoder().encode(buffer);
34        } catch (IOException e) {
35            Logger.output("IOException while reading blocks from file");
36        }
37        return "STATIC".getBytes();
38    }
39
40    @Override
41    public void close() {
42        try {
43            audio.close();
44        } catch (IOException e) {
45            Logger.output("IOException while closing");
46        }
47    }
48 }

```

Apr 26, 18 2:52

## RadiosDBHandler.java

Page 1/3

```

1 import com.google.gson.Gson;
2 import com.rabbitmq.client.*;
3
4 import java.io.IOException;
5 import java.util.Arrays;
6 import java.util.LinkedList;
7 import java.util.List;
8 import java.util.concurrent.*;
9
10 public class RadiosDBHandler extends DBHandlerWithStatistics<Radio> {
11
12     private String queueName;
13
14     public RadiosDBHandler(String host, Database<Radio> database,
15                             List<String> masks)
16         throws IOException, TimeoutException {
17         super(host, database);
18         this.database = database;
19
20         // declare RADIOS_DB exchange
21         getChannel().exchangeDeclare(Configuration.RadiosDBExchange,
22                                     BuiltinExchangeType.TOPIC);
23
24         // declare RADIOS_STATS exchange
25         getChannel().exchangeDeclare(Configuration.RadiosStatisticsExchange,
26                                     BuiltinExchangeType.FANOUT);
27
28         this.queueName = Configuration.RadiosDBExchange + "_" +
29                             Configuration.maskListToStr(masks);
30         getChannel().queueDeclare(queueName, true, false, false, null);
31         for (String mask : masks) {
32             getChannel().queueBind(queueName,
33                                   Configuration.RadiosDBExchange, mask);
34         }
35     }
36
37     @Override
38     public void run() throws IOException {
39
40         Consumer radiosConsumer = new DefaultConsumer(getChannel()) {
41             @Override
42             public void handleDelivery(String consumerTag, Envelope envelope,
43                                     AMQP.BasicProperties properties,
44                                     byte[] body) throws IOException {
45
46                 // parse request
47                 String jsonRequest = new String(body, "UTF-8");
48
49                 DatabaseRequest request = new Gson().fromJson(jsonRequest,
50                                                                DatabaseRequest.class);
51
52                 if (request.getType() == Configuration.UsersTypeConnect) {
53                     consumeConnection(request.getSerializedRequest());
54                 } else if (request.getType() ==
55                     Configuration.UsersTypeDisconnect) {
56                     consumeDisconnection(request.getSerializedRequest());
57                 } else {
58                     Logger.output("Invalid request type received: " +
59                                 request.getType() + ", request: " +
60                                 request.getSerializedRequest());
61                 }
62
63                 getChannel().basicAck(envelope.getDeliveryTag(), false);
64             }
65         };
66
67         getChannel().basicConsume(queueName, false, radiosConsumer);
68     }
69
70     @Override
71     protected List<StatisticTask> getStatistics() {
72         List<StatisticTask> operations = new LinkedList<>();
73     }

```

Apr 26, 18 2:52

## RadiosDBHandler.java

Page 2/3

```

74     Runnable runnable = new Runnable() {
75         @Override
76         public void run() {
77             // get statistics
78             RadiosConnectionsStatistics stats = new RadiosConnectionsStatist
79             ics();
80             for (Radio row : database.getRows()) {
81                 stats.getRadioConnections().put(row.getName(), row.getConne
82                 tedUsers());
83             }
84             String jsonStats = new Gson().toJson(stats);
85             try {
86                 getChannel().basicPublish(
87                     Configuration.RadiosStatisticsExchange, "",
88                     null, jsonStats.getBytes());
89             } catch (IOException e) {
90                 Logger.output("IOException during statistics publish");
91             }
92         };
93
94         operations.add(new StatisticTask(runnable,
95             Configuration.RadioStatisticsPeriodSeconds));
96
97         return operations;
98     }
99
100     private void consumeConnection(String jsonRequest) throws IOException {
101
102         UserConnectRequest request = new Gson().fromJson(jsonRequest,
103             UserConnectRequest.class);
104
105         // get radio record from DB
106         Radio radio = database.getRow(request.getRadio());
107         if (radio == null) {
108             radio = new Radio(request.getRadio());
109         }
110
111         // add one connection to counter
112         radio.setConnectedUsers(radio.getConnectedUsers() + 1);
113         System.out.println("[x] Adding one connection to radio: " +
114             radio.getName());
115
116         // save changes to db
117         database.updateRow(radio);
118     }
119
120     private void consumeDisconnection(String jsonRequest) throws IOException {
121
122         UserDisconnectRequest request = new Gson().fromJson(jsonRequest,
123             UserDisconnectRequest.class);
124
125         // get radio record from DB
126         Radio radio = database.getRow(request.getRadio());
127         if (radio == null) {
128             radio = new Radio(request.getRadio());
129         }
130
131         // add one connection to counter
132         radio.setConnectedUsers(radio.getConnectedUsers() - 1);
133         System.out.println("[x] Removing one connection from " +
134             "radio: " + radio.getName());
135
136         // save changes to db
137         database.updateRow(radio);
138     }
139
140     public static void main(String[] argv) throws Exception {
141         if (argv.length < 1) {
142             System.out.println("Usage: UsersDBHandler mask1 mask2 mask3");
143             return;
144         }

```

Apr 26, 18 2:52

## RadiosDBHandler.java

Page 3/3

```

145     List<String> masks = new LinkedList<>(Arrays.asList(argv));
146
147     // define database
148     Database<Radio> database = new DatabaseJson<>(
149         Configuration.RadiosDBExchange + "_" +
150         Configuration.maskListToStr(masks), Radio.class);
151
152     // start database handler
153     RadiosDBHandler handler = new RadiosDBHandler(Configuration.RabbitMQHost
154     ,
155         database, masks);
156     handler.run();
157 }

```

Apr 25, 18 21:01

**RadiosConnectionsStatistics.java**

Page 1/1

```

1  import java.util.HashMap;
2  import java.util.Map;
3
4  public class RadiosConnectionsStatistics {
5
6      private Map<String, Integer> radioConnections = new HashMap<>();
7
8      public Map<String, Integer> getRadioConnections() {
9          return radioConnections;
10     }
11
12     public void setRadioConnections(Map<String, Integer> radioConnections) {
13         this.radioConnections = radioConnections;
14     }
15 }

```

Apr 25, 18 21:02

**Radio.java**

Page 1/1

```

1  public class Radio extends DatabaseRow {
2      private String name;
3      private int connectedUsers;
4
5      public Radio(String name) {
6          super(name);
7
8          this.setName(name);
9          this.setConnectedUsers(0);
10     }
11
12     public String getName() {
13         return name;
14     }
15
16     public void setName(String name) {
17         this.name = name;
18     }
19
20     public int getConnectedUsers() {
21         return connectedUsers;
22     }
23
24     public void setConnectedUsers(int connectedUsers) {
25         this.connectedUsers = connectedUsers;
26     }
27 }

```

Apr 26, 18 2:55

**RabbitMQProcess.java**

Page 1/1

```

1 import com.rabbitmq.client.Channel;
2 import com.rabbitmq.client.Connection;
3 import com.rabbitmq.client.ConnectionFactory;
4
5 import java.io.IOException;
6 import java.util.concurrent.TimeoutException;
7
8 public abstract class RabbitMQProcess {
9
10     private Connection connection;
11     private Channel channel;
12
13     public RabbitMQProcess(String host) throws IOException, TimeoutException {
14
15         // load configuration
16         Configuration.loadConfiguration("config");
17
18         // init RabbitMQ connection and channel
19         ConnectionFactory factory = new ConnectionFactory();
20         factory.setHost(host);
21         connection = factory.newConnection();
22         channel = connection.createChannel();
23
24         Logger.init();
25
26         addShutdownHook();
27     }
28
29     protected Connection getConnection() {
30         return connection;
31     }
32
33     protected Channel getChannel() {
34         return channel;
35     }
36
37     public void addShutdownHook() {
38
39         RabbitMQProcess instance = this;
40         Thread mainThread = Thread.currentThread();
41         Runtime.getRuntime().addShutdownHook(new Thread() {
42             public void run() {
43
44                 System.out.println("Calling shutdown hook");
45
46                 try {
47                     instance.close();
48                 } catch (IOException e) {
49                     Logger.output("IOException during shutdown hook close");
50                 } catch (TimeoutException e) {
51                     Logger.output("TimeoutException during shutdown hook " +
52                                 "close");
53                 }
54                 try {
55                     mainThread.join();
56                 } catch (InterruptedException e) {
57                     Logger.output("InterruptedException during shutdown hook " +
58                                 "close");
59                 }
60             }
61         });
62     }
63
64     protected void close() throws IOException, TimeoutException {
65         channel.close();
66         connection.close();
67     }
68
69     public abstract void run() throws IOException;
70 }

```

Apr 25, 18 21:36

**Logger.java**

Page 1/2

```

1 import java.io.FileWriter;
2 import java.io.IOException;
3 import java.io.PrintStream;
4 import java.io.PrintWriter;
5 import java.text.SimpleDateFormat;
6 import java.util.Date;
7
8 public class Logger {
9
10     public static LogLevel currentLogLevel = LogLevel.INFO;
11     private static PrintWriter logWriter = null;
12     private static PrintStream out = null;
13
14     public enum LogLevel {
15         ERROR,
16         WARNING,
17         INFO
18     }
19
20     public static LogLevel intToLogLevel(int i) {
21         switch (i) {
22             case 0:
23                 return LogLevel.ERROR;
24             case 1:
25                 return LogLevel.WARNING;
26             case 2:
27                 return LogLevel.INFO;
28             default:
29                 return LogLevel.INFO;
30         }
31     }
32
33     private static String logLevelToString(LogLevel level) {
34         switch (level) {
35             case ERROR:
36                 return "[ERROR]";
37             case WARNING:
38                 return "[WARNING]";
39             case INFO:
40                 return "[INFO]";
41             default:
42                 return "[INVALID LOGLEVEL]";
43         }
44     }
45
46     public static void init() {
47         out = System.out;
48     }
49
50     public static void init(String filename) {
51         init();
52         try {
53             logWriter = new PrintWriter(new FileWriter(filename));
54
55             String timeStamp = new SimpleDateFormat(
56                 "yyyy/MM/dd/HH:mm:ss").format(new Date());
57             logWriter.println("*****" +
58                             timeStamp + "*****");
59         } catch (IOException e) {
60             log("Logger", "Couldn't open logfile for writing",
61                 LogLevel.ERROR);
62         }
63     }
64
65     public static void close() {
66         if (logWriter != null) {
67             logWriter.close();
68         }
69     }
70
71     public static void log(String name, String message, LogLevel level) {
72
73         String logLine = Thread.currentThread().getName() + "\t" +

```

Apr 25, 18 21:36

**Logger.java**

Page 2/2

```

74         logLevelToString(level) + "\t" + name + ":" + message;
75
76         // output to screen
77         if (currentLogLevel.ordinal() ≥ level.ordinal()) {
78             if (out ≠ null) {
79                 out.println(logLine);
80             } else {
81                 System.out.println(logLine);
82             }
83         }
84         // output to logfile
85         if (logWriter ≠ null) {
86             logWriter.println(logLine);
87         }
88     }
89
90     public static void output(String outString) {
91         if (out ≠ null) {
92             out.println(outString);
93         } else {
94             System.out.println(outString);
95         }
96         if (logWriter ≠ null) {
97             logWriter.println(outString);
98         }
99     }
100
101 }

```

Apr 12, 18 3:49

**LimitedSortedSet.java**

Page 1/1

```

1  import java.util.Collection;
2  import java.util.Comparator;
3  import java.util.TreeSet;
4
5  // un sorted set que automaticamente borra elementos
6  // de si mismo si se pasa del maximo
7  class LimitedSortedSet<E> extends TreeSet<E> {
8
9      private int maxSize;
10
11     LimitedSortedSet( int maxSize ) {
12         this.maxSize = maxSize;
13     }
14
15     LimitedSortedSet( int maxSize, Comparator<? super E> comparator ) {
16         super(comparator);
17         this.maxSize = maxSize;
18     }
19
20     @Override
21     public boolean addAll( Collection<? extends E> c ) {
22         boolean added = super.addAll( c );
23         if( size() > maxSize ) {
24             E firstToRemove = (E)toArray( )[maxSize];
25             removeAll( tailSet( firstToRemove ) );
26         }
27         return added;
28     }
29
30     @Override
31     public boolean add( E o ) {
32         boolean added = super.add( o );
33         while (size() > maxSize) {
34             remove(last());
35         }
36         return added;
37     }
38
39 }
40

```

Apr 25, 18 21:01

## KeepAliveRequest.java

Page 1/1

```

1 public class KeepAliveRequest {
2
3     private String username;
4     private int connectionId;
5     private String radio;
6
7     public KeepAliveRequest(String username, int connectionId, String radio) {
8         this.setUsername(username);
9         this.setConnectionId(connectionId);
10        this.setRadio(radio);
11    }
12
13    public String getUsername() {
14        return username;
15    }
16
17    public void setUsername(String username) {
18        this.username = username;
19    }
20
21    public int getConnectionId() {
22        return connectionId;
23    }
24
25    public void setConnectionId(int connectionId) {
26        this.connectionId = connectionId;
27    }
28
29    public String getRadio() {
30        return radio;
31    }
32
33    public void setRadio(String radio) {
34        this.radio = radio;
35    }
36 }

```

Apr 26, 18 0:58

## KeepAliveManager.java

Page 1/1

```

1 import com.google.gson.Gson;
2 import com.rabbitmq.client.*;
3
4 import java.io.IOException;
5 import java.util.concurrent.TimeoutException;
6
7 public class KeepAliveManager extends RabbitMQProcess {
8
9     public KeepAliveManager(String host) throws IOException, TimeoutException {
10        super(host);
11
12        // declare USERS_DB exchange
13        getChannel().exchangeDeclare(Configuration.UsersDBExchange,
14            BuiltinExchangeType.TOPIC);
15    }
16
17    @Override
18    public void run() throws IOException {
19        consumeKeepAlives();
20    }
21
22    private String consumeKeepAlives() throws IOException {
23        // KEEP ALIVE consumer
24        getChannel().queueDeclare(Configuration.KeepAliveQueue, true,
25            false, false, null);
26        Consumer consumer_keepalive = new DefaultConsumer(getChannel()) {
27            @Override
28            public void handleDelivery(String consumerTag, Envelope envelope,
29                AMQP.BasicProperties properties,
30                byte[] body) throws IOException {
31
32                String json = new String(body, "UTF-8");
33                KeepAliveRequest clientRequest = new Gson().fromJson(json,
34                    KeepAliveRequest.class);
35                System.out.println("[X] Received keep alive request from: "
36                    + clientRequest.getUsername() + " to: " +
37                    clientRequest.getRadio() + " id: " +
38                    clientRequest.getConnectionId());
39
40                // ask usersDB to register connection
41                DatabaseRequest usersdbRequest = new DatabaseRequest
42                    (Configuration.UsersTypeKeepAlive, json,
43                    clientRequest.getUsername());
44                getChannel().basicPublish(Configuration.UsersDBExchange,
45                    usersdbRequest.getRoutingKey(), null,
46                    new Gson().toJson(usersdbRequest).getBytes());
47
48                getChannel().basicAck(envelope.getDeliveryTag(), false);
49            }
50        };
51        return getChannel().basicConsume(Configuration.KeepAliveQueue, false,
52            consumer_keepalive);
53    }
54
55    public static void main(String[] argv) throws Exception {
56        KeepAliveManager manager =
57            new KeepAliveManager(Configuration.RabbitMQHost);
58        manager.run();
59    }
60 }

```



Apr 26, 18 2:47

## FileLogger.java

Page 1/1

```

1 import com.google.gson.Gson;
2 import com.rabbitmq.client.*;
3
4 import java.io.PrintWriter;
5 import java.io.IOException;
6 import java.io.PrintWriter;
7 import java.util.Date;
8 import java.util.List;
9 import java.util.ListIterator;
10 import java.util.concurrent.TimeoutException;
11
12 public abstract class FileLogger extends RabbitMQProcess {
13
14     PrintWriter logWriter;
15     String logsQueue;
16
17     public FileLogger(String host, String logFilename) throws
18         IOException, TimeoutException {
19         super(host);
20
21         // declare LOGS exchange
22         getChannel().exchangeDeclare(Configuration.LogsExchange,
23             BuiltinExchangeType.DIRECT);
24
25         logWriter = new PrintWriter(new FileWriter(logFilename, true));
26
27         logsQueue = getChannel().queueDeclare().getQueue();
28         for (String tag : getBindings()) {
29             getChannel().queueBind(logsQueue, Configuration.LogsExchange, tag);
30         }
31     }
32
33     @Override
34     public void run() throws IOException {
35         consumeLogs();
36     }
37
38     protected abstract List<String> getBindings();
39
40     public String consumeLogs() throws IOException {
41         // consume connection logs
42         Consumer connectConsumer = new DefaultConsumer(getChannel()) {
43             @Override
44             public void handleDelivery(String consumerTag, Envelope envelope,
45                 AMQP.BasicProperties properties,
46                 byte[] body) throws IOException {
47
48                 // write log to file
49                 String logLine = new String(body, "UTF-8");
50                 logWriter.println(logLine);
51
52                 System.out.println("[x] Received: " + logLine);
53             }
54         };
55
56         // consume de una cola temporal a traves de un exchange
57         // por lo que no tiene sentido ack manual
58         return getChannel().basicConsume(logsQueue, true, connectConsumer);
59     }
60
61     @Override
62     protected void close() throws IOException, TimeoutException {
63         super.close();
64         logWriter.close();
65     }
66 }

```

Apr 26, 18 1:48

## DisconnectionManager.java

Page 1/2

```

1 import com.google.gson.Gson;
2 import com.rabbitmq.client.*;
3
4 import java.io.IOException;
5 import java.util.concurrent.TimeoutException;
6
7 public class DisconnectionManager extends RabbitMQProcess {
8
9     public DisconnectionManager(String host) throws IOException,
10         TimeoutException {
11         super(host);
12
13         // declare USERS_DB exchange
14         getChannel().exchangeDeclare(Configuration.UsersDBExchange,
15             BuiltinExchangeType.TOPIC);
16
17         // declare RADIOS_DB exchange
18         getChannel().exchangeDeclare(Configuration.RadiosDBExchange,
19             BuiltinExchangeType.TOPIC);
20
21         // declare LOGS exchange
22         getChannel().exchangeDeclare(Configuration.LogsExchange,
23             BuiltinExchangeType.DIRECT);
24     }
25
26     @Override
27     public void run() throws IOException {
28         consumeDisconnections();
29     }
30
31     private String consumeDisconnections() throws IOException {
32         // DISCONNECTIONS consumer
33         getChannel().queueDeclare(Configuration.DisconnectionsQueue, true,
34             false, false, null);
35         Consumer consumer_disconnect = new DefaultConsumer(getChannel()) {
36             @Override
37             public void handleDelivery(String consumerTag, Envelope envelope,
38                 AMQP.BasicProperties properties,
39                 byte[] body) throws IOException {
40
41                 String json = new String(body, "UTF-8");
42                 UserDisconnectRequest clientRequest = new Gson().fromJson(json,
43                     UserDisconnectRequest.class);
44                 System.out.println("[x] Received request to disconnect user: " +
45                     " " + clientRequest.getUsername() + " from: " +
46                     clientRequest.getRadio());
47
48                 // assemble database request
49                 DatabaseRequest dbRequest = new DatabaseRequest
50                     (Configuration.UsersTypeDisconnect, json,
51                     clientRequest.getUsername());
52
53                 // ask usersDB to register disconnection
54                 getChannel().basicPublish(Configuration.UsersDBExchange,
55                     dbRequest.getRoutingKey(), null,
56                     new Gson().toJson(dbRequest).getBytes());
57
58                 // ask radiosDB to register disconnection
59                 getChannel().basicPublish(Configuration.RadiosDBExchange,
60                     dbRequest.getRoutingKey(), null,
61                     new Gson().toJson(dbRequest).getBytes());
62
63                 // send disconnects to file logger
64                 getChannel().basicPublish(Configuration.LogsExchange,
65                     Configuration.LogsDisconnectionTag, null,
66                     clientRequest.toLogLine().getBytes());
67
68                 getChannel().basicAck(envelope.getDeliveryTag(), false);
69             }
70         };
71         return getChannel().basicConsume(Configuration.DisconnectionsQueue,
72             false, consumer_disconnect);
73     }

```

Apr 26, 18 1:48

**DisconnectionManager.java**

Page 2/2

```

74
75     public static void main(String[] argv) throws Exception {
76         DisconnectionManager manager =
77             new DisconnectionManager(Configuration.RabbitMQHost);
78         manager.run();
79     }
80 }

```

Apr 25, 18 22:25

**DBHandlerWithStatistics.java**

Page 1/1

```

1  import java.io.IOException;
2  import java.util.LinkedList;
3  import java.util.List;
4  import java.util.concurrent.*;
5
6  public abstract class DBHandlerWithStatistics<T extends DatabaseRow>
7      extends RabbitMQProcess {
8
9      Database<T> database;
10     private ScheduledExecutorService statisticsScheduler = null;
11     private List<ScheduledFuture<?>> statisticsHandles = null;
12
13     public DBHandlerWithStatistics(String host, Database database) throws
14         IOException,
15         TimeoutException {
16         super(host);
17         this.database = database;
18
19         List<StatisticTask> statisticTasks = getStatistics();
20         //List<Integer> statisticTasksPeriods = getStatisticsPeriodsSeconds();
21         if (statisticTasks.size() > 0) {
22             statisticsScheduler = Executors
23                 .newScheduledThreadPool(
24                     Configuration.PoolSizeForDBstatistics);
25             statisticsHandles = new LinkedList<>();
26
27             for (StatisticTask task : statisticTasks) {
28                 Runnable r = task.getRunnable();
29                 int period = task.getPeriod();
30                 ScheduledFuture<?> statisticsHandle =
31                     statisticsScheduler.scheduleAtFixedRate(r, period,
32                         period, TimeUnit.SECONDS);
33                 statisticsHandles.add(statisticsHandle);
34             }
35         }
36     }
37
38     @Override
39     protected void close() throws IOException, TimeoutException {
40         super.close();
41
42         if (statisticsScheduler != null) {
43             for (ScheduledFuture<?> f : statisticsHandles) {
44                 f.cancel(true);
45             }
46             statisticsScheduler.shutdown();
47         }
48     }
49
50     protected abstract List<StatisticTask> getStatistics();
51 }

```

Apr 25, 18 21:01

**DatabaseRow.java**

Page 1/1

```

1 public abstract class DatabaseRow {
2
3     private String primary_key;
4
5     public DatabaseRow(String primary_key) {
6         this.setPrimary_key(primary_key);
7     }
8
9     public String getPrimary_key() {
10        return primary_key;
11    }
12
13    public void setPrimary_key(String primary_key) {
14        this.primary_key = primary_key;
15    }
16 }

```

Apr 26, 18 0:54

**DatabaseRequest.java**

Page 1/1

```

1 public class DatabaseRequest {
2
3     private int type;
4     private String serializedRequest;
5     private String routingKey;
6
7     public DatabaseRequest(int type, String serializedRequest, String
8         username) {
9         this.type = type;
10        this.serializedRequest = serializedRequest;
11        this.setRoutingKey(username.substring(0, 1));
12    }
13
14    public int getType() {
15        return type;
16    }
17
18    public void setType(int type) {
19        this.type = type;
20    }
21
22    public String getSerializedRequest() {
23        return serializedRequest;
24    }
25
26    public void setSerializedRequest(String serializedRequest) {
27        this.serializedRequest = serializedRequest;
28    }
29
30    public String getRoutingKey() {
31        return routingKey;
32    }
33
34    public void setRoutingKey(String routingKey) {
35        this.routingKey = routingKey;
36    }
37 }

```

Apr 25, 18 22:54

## DatabaseRAM.java

Page 1/1

```

1 import com.google.gson.internal.LinkedTreeMap;
2
3 import java.util.HashMap;
4 import java.util.LinkedList;
5 import java.util.List;
6 import java.util.Map;
7
8 public class DatabaseRAM<T extends DatabaseRow> implements Database<T> {
9
10     private Map<String, T> database = new HashMap<>();
11
12     public T getRow(String key) {
13         T row = database.getDefault(key, null);
14         return row;
15     }
16
17     @Override
18     public List<T> getRows() {
19         return new LinkedList<>(database.values());
20     }
21
22     @Override
23     public boolean createRow(T row) {
24         if (database.put(row.getPrimary_key(), row) == null) {
25             return true;
26         }
27         return false;
28     }
29
30     @Override
31     public boolean updateRow(T row) {
32         if (!database.containsKey(row.getPrimary_key())) {
33             return createRow(row);
34         }
35         database.put(row.getPrimary_key(), row);
36         return true;
37     }
38
39     @Override
40     public boolean removeRow(String primary_key) {
41         if (database.remove(primary_key) != null) {
42             return true;
43         }
44         return false;
45     }
46
47     protected Map<String, T> getDatabase() {
48         return database;
49     }
50
51     protected void setDatabase(Map<String, T> db) {
52         this.database = db;
53     }
54 }

```

Apr 25, 18 23:17

## DatabaseJson.java

Page 1/2

```

1 import com.google.gson.Gson;
2
3 import java.io.*;
4 import java.util.HashMap;
5 import java.util.Map;
6
7 public class DatabaseJson<T extends DatabaseRow> extends DatabaseRAM<T> {
8
9     private String filename;
10    private Class<T> classOfT;
11
12    public DatabaseJson(String filename, Class<T> classOfT) {
13        this.filename = filename;
14        this.classOfT = classOfT;
15
16        loadFromFile();
17    }
18
19    @Override
20    public boolean createRow(T row) {
21        boolean result = super.createRow(row);
22        if (result) {
23            saveToFile();
24        }
25        return result;
26    }
27
28    @Override
29    public boolean updateRow(T row) {
30        boolean result = super.updateRow(row);
31        if (result) {
32            saveToFile();
33        }
34        return result;
35    }
36
37    @Override
38    public boolean removeRow(String primary_key) {
39        boolean result = super.removeRow(primary_key);
40        if (result) {
41            saveToFile();
42        }
43        return result;
44    }
45
46    private void loadFromFile() {
47        try {
48
49            BufferedReader br = new BufferedReader(new FileReader(filename));
50            Map<String, T> db = new HashMap<>();
51
52            Gson gson = new Gson();
53            String line;
54            while ((line = br.readLine()) != null) {
55                Logger.output(line);
56                T row = gson.fromJson(line, classOfT);
57                db.put(row.getPrimary_key(), row);
58            }
59            br.close();
60            setDatabase(db);
61
62        } catch (IOException e) {
63            Logger.output("Unable to load database from file: " + filename);
64        }
65    }
66
67    private void saveToFile() {
68        try {
69            PrintWriter writer = new PrintWriter(new FileWriter(filename));
70            Map<String, T> db = getDatabase();
71            Gson gson = new Gson();
72
73            for (String key : db.keySet()) {

```

Apr 25, 18 23:17

**DatabaseJson.java**

Page 2/2

```
74     String jsonRow = gson.toJson(db.get(key), classOfT);
75     writer.println(jsonRow);
76 }
77 writer.close();
78 } catch (IOException e) {
79     Logger.output("Unable to save database to file: " + filename);
80 }
81 }
82 }
```

Apr 12, 18 2:38

**Database.java**

Page 1/1

```
1 import java.util.List;
2
3 public interface Database<T extends DatabaseRow> {
4
5     T getRow(String key);
6
7     List<T> getRows();
8
9     boolean createRow(T row);
10
11     boolean updateRow(T row);
12
13     boolean removeRow(String key);
14 }
```

Apr 26, 18 2:46

## ConnectionManager.java

Page 1/2

```

1  import com.google.gson.Gson;
2  import com.rabbitmq.client.*;
3  import sun.security.krb5.Config;
4
5  import java.io.IOException;
6  import java.util.concurrent.TimeoutException;
7
8  public class ConnectionManager extends RabbitMQProcess {
9
10     public ConnectionManager(String host) throws IOException, TimeoutException {
11         super(host);
12
13         // declare USERS_DB exchange
14         getChannel().exchangeDeclare(Configuration.UsersDBExchange,
15             BuiltinExchangeType.TOPIC);
16
17         // declare usersDB responses queue
18         getChannel().queueDeclare(Configuration.ConnMgrUsersDBResponseQueue,
19             true, false, null);
20
21         // declare RADIOS_DB exchange
22         getChannel().exchangeDeclare(Configuration.RadiosDBExchange,
23             BuiltinExchangeType.TOPIC);
24
25         // declare LOGS exchange
26         getChannel().exchangeDeclare(Configuration.LogsExchange,
27             BuiltinExchangeType.DIRECT);
28     }
29
30     @Override
31     public void run() throws IOException {
32         //consumeConnections();
33         consumeUsersDB();
34     }
35
36     private String consumeUsersDB() throws IOException {
37         // usersDB consume
38         Consumer consumer_usersdb = new DefaultConsumer(getChannel()) {
39             @Override
40             public void handleDelivery(String consumerTag, Envelope envelope,
41                 AMQP.BasicProperties properties,
42                 byte[] body) throws IOException {
43
44                 String json = new String(body, "UTF-8");
45                 UserConnectResponse response = new Gson().fromJson(json,
46                     UserConnectResponse.class);
47
48                 for (UserDisconnectRequest disconn :
49                     response.getClosedConnections()) {
50
51                     // register closed connections in radios DB
52                     DatabaseRequest dbRequest = new DatabaseRequest
53                         (Configuration.UsersTypeDisconnect, new Gson()
54                             .toJson(disconn), disconn.getUsername());
55                     getChannel().basicPublish(Configuration.RadiosDBExchange,
56                         dbRequest.getRoutingKey(), null,
57                         new Gson().toJson(dbRequest).getBytes());
58
59                     // send disconnects to file logger
60                     getChannel().basicPublish(Configuration.LogsExchange,
61                         Configuration.LogsDisconnectionTag, null,
62                         disconn.toLogLine().getBytes());
63                 }
64
65                 if (response.isCouldConnect()) {
66                     // send connect to file logger
67                     getChannel().basicPublish(Configuration.LogsExchange,
68                         Configuration.LogsConnectionTag, null,
69                         response.toLogLine().getBytes());
70                 }
71
72                 // register connection in radios DB
73                 DatabaseRequest dbRequest = new DatabaseRequest
                     (Configuration.UsersTypeConnect, new Gson()

```

Apr 26, 18 2:46

## ConnectionManager.java

Page 2/2

```

74         .toJson(new UserConnectRequest(response)),
75         response.getUsername());
76         getChannel().basicPublish(Configuration.RadiosDBExchange,
77             dbRequest.getRoutingKey(), null,
78             new Gson().toJson(dbRequest).getBytes());
79
80         System.out.println(" [X] User: " + response.getUsername() +
81             " connected to radio: " + response.getRadio());
82     } else {
83         System.out.println(" [X] User: " + response.getUsername() +
84             " denied connection to radio: " + response.getRadio());
85     }
86
87     String jsonResponse = new Gson().toJson(response);
88     getChannel().basicPublish("", response.getReturnQueueName(), null,
89         jsonResponse.getBytes());
90
91     getChannel().basicAck(envelope.getDeliveryTag(), false);
92 }
93 };
94 return getChannel().basicConsume(Configuration.ConnMgrUsersDBResponseQueue,
95     false, consumer_usersdb);
96 }
97
98 public static void main(String[] argv) throws Exception {
99
100     ConnectionManager manager =
101         new ConnectionManager(Configuration.RabbitMQHost);
102     manager.run();
103 }
104
105 }

```

Apr 25, 18 20:52

## ConnDisconnFileLogger.java

Page 1/1

```

1 import java.io.IOException;
2 import java.util.LinkedList;
3 import java.util.List;
4 import java.util.concurrent.TimeoutException;
5
6 public class ConnDisconnFileLogger extends FileLogger {
7
8     public ConnDisconnFileLogger(String host, String logFilename) throws
9         IOException, TimeoutException {
10         super(host, logFilename);
11     }
12
13     @Override
14     protected List<String> getBindings() {
15         List<String> bindings = new LinkedList<>();
16         bindings.add(Configuration.LogsConnectionTag);
17         bindings.add(Configuration.LogsDisconnectionTag);
18         return bindings;
19     }
20
21     public static void main(String[] argv) throws Exception {
22
23         ConnDisconnFileLogger fileLogger =
24             new ConnDisconnFileLogger(Configuration.RabbitMQHost,
25                 argv[0]);
26         fileLogger.run();
27     }
28 }

```

Apr 26, 18 3:24

## Configuration.java

Page 1/2

```

1 import com.google.gson.Gson;
2 import com.google.gson.GsonBuilder;
3
4 import java.io.BufferedReader;
5 import java.io.FileNotFoundException;
6 import java.io.FileReader;
7 import java.io.IOException;
8 import java.util.List;
9
10 public class Configuration {
11
12     public static int RadioSendPeriodMilliseconds = 1000;
13     public static int RadioAudiofileBytesPerSecond = 102400;
14     public static int KeepAlivePeriodSeconds = 5;
15     public static int SecondsUntilDropConnection = 10;
16     public static int MaxConnectionsPerFreeUser = 3;
17     public static int MaxConnectionsPerUnlimitedUser = 999;
18
19     public static String RabbitMQHost = "localhost";
20
21     public static int PoolSizeForDBstatistics = 1;
22
23     public static String UsersDBExchange = "USERS_DB";
24     public static int UsersTypeConnect = 1;
25     public static int UsersTypeDisconnect = 2;
26     public static int UsersTypeKeepAlive = 3;
27     public static String UsersStatisticsExchange = "USERS_STATS";
28     public static int UsersStatisticsPeriodSeconds = 10;
29     public static int UserStatisticsN = 100;
30
31     public static String RadiosDBExchange = "RADIOS_DB";
32     public static String RadiosStatisticsExchange = "RADIOS_STATS";
33     public static int RadioStatisticsPeriodSeconds = 10;
34
35     public static String ConnMgrUsersDBResponseQueue =
36         "usersDBResponseQueueName";
37
38     public static String ConnectionsQueue = "CONNECTIONS";
39     public static String DisconnectionsQueue = "DISCONNECTIONS";
40     public static String KeepAliveQueue = "KEEP_ALIVE";
41
42     public static String RadioExchangePrefix = "BROADCAST-";
43
44     public static String LogsExchange = "LOGS";
45     public static String LogsConnectionTag = "connect";
46     public static String LogsDisconnectionTag = "disconnect";
47
48     public static String maskListToStr(List<String> masks) {
49         return String.join(",", masks)
50             .replace(".", "")
51             .replace("#", "");
52     }
53
54     public static boolean loadConfiguration(String configFilename) {
55
56         try {
57             // read json config
58             BufferedReader br = new BufferedReader(
59                 new FileReader( configFilename));
60             String jsonString = "";
61             String s;
62             while ((s = br.readLine()) != null) {
63                 jsonString += s;
64             }
65
66             // esto es para que gson serialize variables estaticas
67             GsonBuilder gsonBuilder = new GsonBuilder();
68             gsonBuilder.excludeFieldsWithModifiers(
69                 java.lang.reflect.Modifier.TRANSIENT);
70
71             Gson gson = gsonBuilder.create();
72             // load to object
73             Configuration config = gson.fromJson(jsonString,

```

Apr 26, 18 3:24

## Configuration.java

Page 2/2

```

74         Configuration.class);
75     }
76     catch (FileNotFoundException e) {
77         Logger.output ("FileNotFoundException loading " +
78             "configuration file, using defaults");
79     } catch (IOException e) {
80         Logger.output ("IOException loading configuration file, using " +
81             "defaults");
82     } finally {
83         return true;
84     }
85 }
86 }

```

Apr 26, 18 3:28

## Client.java

Page 1/4

```

1  import com.google.gson.Gson;
2  import com.rabbitmq.client.*;
3  import java.util.Base64;
4
5  import java.io.FileOutputStream;
6  import java.io.IOException;
7  import java.text.SimpleDateFormat;
8  import java.util.Date;
9  import java.util.NoSuchElementException;
10 import java.util.Scanner;
11 import java.util.concurrent.*;
12
13 public class Client extends RabbitMQProcess {
14
15     private String radioExchange = "";
16
17     private String username = "";
18     private String radio;
19     private int connectionId;
20     private String radioConsumeTag = "";
21     FileOutputStream transmissionWriter = null;
22
23     private ScheduledExecutorService keepAliveScheduler =
24         Executors.newScheduledThreadPool(1);
25     private ScheduledFuture<?> keepAliveHandle;
26
27     public Client(String host) throws IOException, TimeoutException {
28         super(host);
29     }
30
31     public void setUsername(String username) {
32         this.username = username;
33     }
34
35     public void setRadio(String radio) {
36         this.radio = radio;
37     }
38
39     public boolean requestConnectionToRadio() throws IOException,
40         InterruptedException {
41
42         if (username.equals("")) {
43             Logger.output ("ERROR: Did you specify a username?");
44             return false;
45         }
46
47         // define callback queue
48         String callbackQueueName = getChannel().queueDeclare().getQueue();
49
50         // create request
51         UserConnectRequest request = new UserConnectRequest(username, radio,
52             callbackQueueName);
53         String requestJson = new Gson().toJson(request);
54
55         // publish to usersDB exchange to start register connection operation
56         DatabaseRequest usersdbRequest = new DatabaseRequest
57             (Configuration.UsersTypeConnect, requestJson, username);
58         getChannel().basicPublish(Configuration.UsersDBExchange,
59             usersdbRequest.getRoutingKey(), null,
60             new Gson().toJson(usersdbRequest).getBytes());
61
62         final BlockingQueue<String> responseQueue =
63             new ArrayBlockingQueue<String>(1);
64
65         // es una cola temporaria, no sirve de nada el ack
66         String callbackTag = getChannel().basicConsume(callbackQueueName, true,
67             new DefaultConsumer(getChannel()) {
68                 @Override
69                 public void handleDelivery(String consumerTag, Envelope envelope,
70                     AMQP.BasicProperties properties,
71                     byte[] body) throws IOException {
72                     responseQueue.offer(new String(body, "UTF-8"));
73                 }
74             });

```



Apr 26, 18 3:28

Client.java

Page 2/4

```

74     });
75
76     String jsonResponse = responseQueue.take();
77     getChannel().basicCancel(callbackTag);
78     UserConnectResponse response = new Gson().fromJson(jsonResponse,
79         UserConnectResponse.class);
80     if (!response.isCouldConnect()) {
81         Logger.output("ERROR: Connection refused, are " +
82             "you already connected on 3 devices?");
83         return false;
84     }
85
86     connectionId = response.getConnectionId();
87     radioExchange = Configuration.RadioExchangePrefix + response.getRadio();
88     return true;
89 }
90
91 public boolean listenToRadio() throws IOException {
92
93     if (radioExchange.equals("")) {
94         return false;
95     }
96
97     // declare radio broadcast exchange
98     getChannel().exchangeDeclare(radioExchange, BuiltinExchangeType.FANOUT);
99
100    // declare temporary queue and bind
101    String queueName = getChannel().queueDeclare().getQueue();
102    getChannel().queueBind(queueName, radioExchange, "");
103    Logger.output("Creating queue: " + queueName);
104
105    // open new file for transmission
106    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss");
107    String transmissionName = "client" + "-" + username + "-" + radio +
108        "-" + connectionId + "-" + sdf.format(new Date()) + ".wav";
109    transmissionWriter = new FileOutputStream(transmissionName);
110
111    Consumer consumer = new DefaultConsumer(getChannel()) {
112        @Override
113        public void handleDelivery(String consumerTag, Envelope envelope,
114            AMQP.BasicProperties properties,
115            byte[] body) throws IOException {
116            String message = new String(body, "UTF-8");
117            Logger.output("[x] Received '" + message + "'");
118            byte[] decodedBody = Base64.getDecoder().decode(body);
119            transmissionWriter.write(decodedBody);
120        }
121    };
122    // es una cola temporaria, no sirve de nada el ack
123    radioConsumeTag = getChannel().basicConsume(queueName, true,
124        consumer);
125    return true;
126 }
127
128 public void scheduleKeepAlive() {
129     final Runnable sendKeepAlive = new Runnable() {
130         @Override
131         public void run() {
132             KeepAliveRequest request = new KeepAliveRequest(username,
133                 connectionId, radio);
134             String requestJson = new Gson().toJson(request);
135             try {
136                 getChannel().basicPublish("",
137                     Configuration.KeepAliveQueue, null,
138                     requestJson.getBytes());
139             } catch (IOException e) {
140                 Logger.output("IOException while listening to " +
141                     "radio: " + radio + ", user: " + username +
142                     ", connection id: " + connectionId);
143             }
144         }
145     };
146     keepAliveHandle = keepAliveScheduler.scheduleAtFixedRate

```

Apr 26, 18 3:28

Client.java

Page 3/4

```

147         (sendKeepAlive, Configuration.KeepAlivePeriodSeconds,
148             Configuration.KeepAlivePeriodSeconds, TimeUnit.SECONDS);
149     }
150
151     public void stopKeepAlive() {
152         keepAliveHandle.cancel(true);
153     }
154
155     public void stopListeningToRadio() throws IOException {
156         if (radioConsumeTag.equals("")) {
157             Logger.output("ERROR: not listening to radio");
158         } else {
159             // create request
160             UserDisconnectRequest request = new UserDisconnectRequest(username,
161                 radio, connectionId);
162             String requestJson = new Gson().toJson(request);
163
164             // publish to DISCONNECTIONS queue
165             getChannel().basicPublish("", Configuration.DisconnectionsQueue,
166                 null, requestJson.getBytes());
167
168             // stop receiving transmission
169             getChannel().basicCancel(radioConsumeTag);
170             radioConsumeTag = "";
171
172             // close transmission file
173             transmissionWriter.close();
174             transmissionWriter = null;
175         }
176     }
177
178     public void printOptions() {
179         Logger.output("\n");
180         Logger.output("Choose an action: ");
181         Logger.output("\t" + "1. Set user");
182         Logger.output("\t" + "2. Connect to radio");
183         Logger.output("\t" + "3. Disconnect from radio");
184         Logger.output("\t" + "4. Exit");
185     }
186
187     public boolean mainMenu(Scanner in) throws IOException,
188         InterruptedException {
189         String choiceStr = in.nextLine();
190         int choice = Integer.parseInt(choiceStr);
191         switch (choice) {
192             case 1:
193                 System.out.print("Please specify a username: ");
194                 String username = in.nextLine();
195                 setUsername(username);
196                 break;
197             case 2:
198                 Logger.output("Please specify a radio: ");
199                 String radio = in.nextLine();
200                 setRadio(radio);
201                 if (!requestConnectionToRadio()) {
202                     break;
203                 }
204                 listenToRadio();
205                 scheduleKeepAlive();
206                 break;
207             case 3:
208                 stopListeningToRadio();
209                 stopKeepAlive();
210                 break;
211             case 4:
212                 Logger.output("Press CTRL+C to exit");
213                 return true;
214             default:
215                 Logger.output("ERROR: Invalid option");
216                 break;
217         }
218
219         printOptions();

```

Apr 26, 18 3:28

## Client.java

Page 4/4

```

220     return false;
221 }
222
223 @Override
224 protected void close() throws IOException, TimeoutException {
225     super.close();
226     if (transmissionWriter != null) {
227         transmissionWriter.close();
228     }
229 }
230
231 @Override
232 public void run() throws IOException {
233     Scanner in = new Scanner(System.in);
234     printOptions();
235
236     boolean end = false;
237     while (!end) {
238         try {
239             end = mainMenu(in);
240             // esta excepcion aparece al apretar ctrl+c
241         } catch (NoSuchElementException e) {
242             end = true;
243         } catch (InterruptedException e) {
244             end = true;
245         }
246     }
247 }
248
249 public static void main(String[] argv) throws Exception {
250     Client client = new Client(Configuration.RabbitMQHost);
251     client.run();
252 }
253
254
255 }

```

May 01, 18 20:41

## Table of Content

Page 1/1

1	<b>Table of Contents</b>			
2	1	UserStatisticsViewer.java	sheets 1 to 1 ( 1) pages 1- 1	58 lines
3	2	UsersSecondsListenedStatistics.java	sheets 1 to 1 ( 1) pages 2- 2	21 lines
4	3	UsersSecondsListenedComparator.java	sheets 2 to 2 ( 1) pages 3- 3	13 lines
5	4	UserSecondsListened.java	sheets 2 to 2 ( 1) pages 4- 4	27 lines
6	5	UsersDBHandler.java	sheets 3 to 4 ( 2) pages 5- 8	256 lines
7	6	UserRadioConnection.java	sheets 5 to 5 ( 1) pages 9- 9	17 lines
8	7	User.java	sheets 5 to 5 ( 1) pages 10- 10	19 lines
9	8	UserDisconnectRequest.java	sheets 6 to 6 ( 1) pages 11- 11	44 lines
10	9	UserConnectResponse.java	sheets 6 to 7 ( 2) pages 12- 13	77 lines
11	10	UserConnectRequest.java	sheets 7 to 7 ( 1) pages 14- 14	48 lines
12	11	StatisticTask.java	sheets 8 to 8 ( 1) pages 15- 15	27 lines
13	12	RadiosUpdateRequest.java	sheets 8 to 8 ( 1) pages 16- 16	37 lines
14	13	RadioStatisticsViewer.java	sheets 9 to 9 ( 1) pages 17- 17	57 lines
15	14	RadioStation.java	sheets 9 to 9 ( 1) pages 18- 18	71 lines
16	15	RadioSourceRandomNumbers.java	sheets 10 to 10 ( 1) pages 19- 19	25 lines
17	16	RadioSource.java	sheets 10 to 10 ( 1) pages 20- 20	11 lines
18	17	RadioSourceFile.java	sheets 11 to 11 ( 1) pages 21- 21	49 lines
19	18	RadiosDBHandler.java	sheets 11 to 12 ( 2) pages 22- 24	158 lines
20	19	RadiosConnectionsStatistics.java	sheets 13 to 13 ( 1) pages 25- 25	16 lines
21	20	Radio.java	sheets 13 to 13 ( 1) pages 26- 26	28 lines
22	21	RabbitMQProcess.java	sheets 14 to 14 ( 1) pages 27- 27	71 lines
23	22	Logger.java	sheets 14 to 15 ( 2) pages 28- 29	102 lines
24	23	LimitedSortedSet.java	sheets 15 to 15 ( 1) pages 30- 30	41 lines
25	24	KeepAliveRequest.java	sheets 16 to 16 ( 1) pages 31- 31	37 lines
26	25	KeepAliveManager.java	sheets 16 to 16 ( 1) pages 32- 32	61 lines
27	26	FileLogger.java	sheets 17 to 17 ( 1) pages 33- 33	67 lines
28	27	DisconnectionManager.java	sheets 17 to 18 ( 2) pages 34- 35	81 lines
29	28	DBHandlerWithStatistics.java	sheets 18 to 18 ( 1) pages 36- 36	52 lines
30	29	DatabaseRow.java	sheets 19 to 19 ( 1) pages 37- 37	17 lines
31	30	DatabaseRequest.java	sheets 19 to 19 ( 1) pages 38- 38	38 lines
32	31	DatabaseRAM.java	sheets 20 to 20 ( 1) pages 39- 39	55 lines
33	32	DatabaseJson.java	sheets 20 to 21 ( 2) pages 40- 41	83 lines
34	33	Database.java	sheets 21 to 21 ( 1) pages 42- 42	15 lines
35	34	ConnectionManager.java	sheets 22 to 22 ( 1) pages 43- 44	106 lines
36	35	ConnDisconnFileLogger.java	sheets 23 to 23 ( 1) pages 45- 45	29 lines
37	36	Configuration.java	sheets 23 to 24 ( 2) pages 46- 47	87 lines
38	37	Client.java	sheets 24 to 26 ( 3) pages 48- 51	256 lines