

Apr 12, 18 3:44

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         channel.exchangeDeclare(Configuration.UsersStatisticsExchange,
14             BuiltinExchangeType.FANOUT);
15
16         consumeStatistics();
17     }
18
19     private String consumeStatistics() throws IOException {
20         String statisticsQueue = channel.queueDeclare().getQueue();
21         channel.queueBind(statisticsQueue,
22             Configuration.UsersStatisticsExchange, "");
23
24         Consumer consumerStatistics = new DefaultConsumer(channel) {
25             @Override
26             public void handleDelivery(String consumerTag, Envelope envelope,
27                 AMQP.BasicProperties properties,
28                 byte[] body) throws IOException {
29
30                 String json = new String(body, "UTF-8");
31                 UsersSecondsListenedStatistics statistics = new Gson().fromJson(
32                     json, UsersSecondsListenedStatistics.class);
33
34                 System.out.println(" [x] Showing connections per radio: ");
35                 for (UserSecondsListened userStats :
36                     statistics.usersMostListenedSeconds) {
37                     System.out.println(userStats.username + ":" +
38                         userStats.secondsListened);
39                 }
40             }
41         };
42         return channel.basicConsume(statisticsQueue, true, consumerStatistics);
43     }
44
45     public static void main(String[] argv) throws Exception {
46         UserStatisticsViewer statisticsViewer =
47             new UserStatisticsViewer(Configuration.RabbitMQHost);
48     }
49 }

```

Apr 12, 18 3:51

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      public List<UserSecondsListened> usersMostListenedSeconds;
8
9      public UsersSecondsListenedStatistics(Collection<UserSecondsListened> stats)
10     {
11         this.usersMostListenedSeconds = new LinkedList<>(stats);
12     }

```

Apr 12, 18 3:37 **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         if (o2.secondsListened < o1.secondsListened) {
10             return -1;
11         }
12         if (o2.secondsListened > o1.secondsListened) {
13             return 1;
14         }
15         return 0;
16     }
17 }
```

Apr 12, 18 3:40 **UserSecondsListened.java** Page 1/1

```
1 public class UserSecondsListened {
2
3     public String username;
4     public long secondsListened;
5
6     public UserSecondsListened(String username, long secondsListened) {
7         this.username = username;
8         this.secondsListened = secondsListened;
9     }
10 }
```

Apr 11, 18 21:38

UsersDBRowRadioConnection.java

Page 1/1

```

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

```

Apr 11, 18 23:20

UsersDBRow.java

Page 1/1

```

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

```

Apr 12, 18 13:41

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.Date;
6  import java.util.LinkedList;
7  import java.util.List;
8  import java.util.ListIterator;
9  import java.util.concurrent.TimeoutException;
10
11 public class UsersDBHandler extends DBHandlerWithStatistics<UsersDBRow> {
12
13     public UsersDBHandler(String host, Database<UsersDBRow> database) throws
14         IOException,
15         TimeoutException {
16         super(host, database);
17
18         // declare USERS_DB exchange
19         channel.exchangeDeclare(Configuration.UsersDBExchange,
20             BuiltInExchangeType.DIRECT);
21
22         // declare USERS_STATS exchange
23         channel.exchangeDeclare(Configuration.UsersStatisticsExchange,
24             BuiltInExchangeType.FANOUT);
25
26         // consumer methods
27         consumeConnections();
28         consumeDisconnections();
29         consumeKeepAlive();
30     }
31
32     @Override
33     protected List<Runnable> getStatisticsOperations() {
34         List<Runnable> operations = new LinkedList<>();
35         operations.add(new Runnable() {
36             @Override
37             public void run() {
38                 // get statistics
39                 LimitedSortedSet<UserSecondsListened> usersMostListened =
40                     new LimitedSortedSet<>(Configuration.UserStatisticsN,
41                         new UsersSecondsListenedComparator());
42                 for (UsersDBRow row : database.getRows()) {
43                     UserSecondsListened userStats =
44                         new UserSecondsListened(row.username,
45                             row.secondsListening);
46                     usersMostListened.add(userStats);
47                 }
48
49                 UsersSecondsListenedStatistics stats =
50                     new UsersSecondsListenedStatistics(usersMostListened);
51                 String jsonStats = new Gson().toJson(stats);
52
53                 try {
54                     channel.basicPublish(Configuration.UsersStatisticsExchange,
55                         "", null, jsonStats.getBytes());
56                 } catch (IOException e) {
57                     e.printStackTrace();
58                 }
59             }
60         });
61
62         return operations;
63     }
64
65     @Override
66     protected List<Integer> getStatisticsPeriodsSeconds() {
67         List<Integer> operationsPeriods = new LinkedList<>();
68         operationsPeriods.add(Configuration.UsersStatisticsPeriodSeconds);
69         return operationsPeriods;
70     }
71
72     public String consumeConnections() throws IOException {
73         // consume CONNECT_TO_RADIO requests

```

Apr 12, 18 13:41

UsersDBHandler.java

Page 2/4

```

74     String connectUsersQueue = channel.queueDeclare().getQueue();
75     channel.queueBind(connectUsersQueue, Configuration.UsersDBExchange,
76         Configuration.UsersDBConnectTag);
77     Consumer connectConsumer = new DefaultConsumer(channel) {
78         @Override
79         public void handleDelivery(String consumerTag, Envelope envelope,
80             AMQP.BasicProperties properties,
81             byte[] body) throws IOException {
82
83         // parse request
84         String jsonRequest = new String(body, "UTF-8");
85
86         UserConnectRequest request = new Gson().fromJson(jsonRequest,
87             UserConnectRequest.class);
88         UserConnectResponse response = new UserConnectResponse(request);
89
90         // get user record from DB
91         UsersDBRow user = database.getRow(request.username);
92         if (user == null) {
93             System.out.println("User: " + request.username + " not " +
94                 "found, creating new user");
95             user = new UsersDBRow(request.username);
96
97         // first check if any connection is not active and remove it
98         int connectionId = 0;
99         Date now = new Date();
100         ListIterator<UsersDBRowRadioConnection> connIter =
101             user.connections.listIterator();
102         while (connIter.hasNext()) {
103             UsersDBRowRadioConnection connection = connIter.next();
104
105             Date then = connection.keepAlive();
106             if (now.getTime() - then.getTime() >
107                 Configuration.SecondsUntilDropConnection * 1000) {
108                 UserDisconnectRequest disconnectRequest = new
109                     UserDisconnectRequest(request.username,
110                         connection.radio,
111                         connection.connectionId);
112                 response.closedConnections.add(disconnectRequest);
113                 connIter.remove();
114
115                 System.out.println("[x] Closing old connection to: " +
116                     connection.radio);
117             } else {
118                 if (connection.connectionId > connectionId) {
119                     connectionId = connection.connectionId;
120                 }
121             }
122         }
123         connectionId = (connectionId + 1) %
124             Configuration.MaxConnectionsPerUnlimitedUser;
125         // then check if user can connect to radio
126         if (user.connections.size() < user.connectionsLimit) {
127             UsersDBRowRadioConnection connection =
128                 new UsersDBRowRadioConnection(response.radio,
129                     new Date(), connectionId);
130             user.connections.add(connection);
131             response.couldConnect = true;
132             response.connectionId = connectionId;
133
134             System.out.println("[x] User: " + user.username +
135                 " connected to: " + request.radio);
136         } else {
137             response.couldConnect = false;
138
139             System.out.println("[x] User: " + user.username +
140                 " not connected to: " + request.radio);
141         }
142         // update user
143         database.updateRow(user);
144
145         String jsonResponse = new Gson().toJson(response);
146         channel.basicPublish("",

```

Apr 12, 18 13:41

UsersDBHandler.java

Page 3/4

```

147         Configuration.ConnMgrUsersDBResponseQueue, null,
148         jsonResponse.getBytes());
149     }
150 };
151     return channel.basicConsume(connectUsersQueue, true, connectConsumer);
152 }
153
154     public String consumeDisconnections() throws IOException {
155         // consume DISCONNECT_FROM_RADIO requests
156         String disconnectUsersQueue = channel.queueDeclare().getQueue();
157         channel.queueBind(disconnectUsersQueue, Configuration.UsersDBExchange,
158             Configuration.UsersDBDisconnectTag);
159         Consumer disconnectConsumer = new DefaultConsumer(channel) {
160             @Override
161             public void handleDelivery(String consumerTag, Envelope envelope,
162                 AMQP.BasicProperties properties,
163                 byte[] body) throws IOException {
164
165                 // parse request
166                 String jsonRequest = new String(body, "UTF-8");
167
168                 UserDisconnectRequest request = new Gson().fromJson(jsonRequest,
169                     UserDisconnectRequest.class);
170
171                 // get user record from DB
172                 UsersDBRow user = database.getRow(request.username);
173                 if (user != null) {
174                     ListIterator<UsersDBRowRadioConnection> connIter =
175                         user.connections.listIterator();
176                     while (connIter.hasNext()) {
177                         UsersDBRowRadioConnection connection = connIter.next();
178                         if (connection.connectionID == request.connectionId ^
179                             connection.radio.equals(request.radio)) {
180                             connIter.remove();
181                             System.out.println(" [x] Removing connection " +
182                                 "from: " + user.username + " to radio: "
183                                 + connection.radio);
184                             break;
185                         }
186                     }
187                 }
188                 // update user
189                 database.updateRow(user);
190             }
191         };
192         return channel.basicConsume(disconnectUsersQueue, true,
193             disconnectConsumer);
194     }
195
196     public String consumeKeepAlive() throws IOException {
197         // consume KEEP_ALIVE requests
198         String keepaliveUsersQueue = channel.queueDeclare().getQueue();
199         channel.queueBind(keepaliveUsersQueue, Configuration.UsersDBExchange,
200             Configuration.UsersDBKeepAliveTag);
201         Consumer keepaliveConsumer = new DefaultConsumer(channel) {
202             @Override
203             public void handleDelivery(String consumerTag, Envelope envelope,
204                 AMQP.BasicProperties properties,
205                 byte[] body) throws IOException {
206
207                 // parse request
208                 String jsonRequest = new String(body, "UTF-8");
209
210                 KeepAliveRequest request = new Gson().fromJson(jsonRequest,
211                     KeepAliveRequest.class);
212
213                 // get user record from DB
214                 UsersDBRow user = database.getRow(request.username);
215                 if (user == null) {
216                     System.out.println(" [x] Error: user who sent keep alive " +
217                         "does not exist");
218                     return;
219                 }

```

Apr 12, 18 13:41

UsersDBHandler.java

Page 4/4

```

220
221         // refresh keep alive
222         ListIterator<UsersDBRowRadioConnection> connIter =
223             user.connections.listIterator();
224         while (connIter.hasNext()) {
225             UsersDBRowRadioConnection connection = connIter.next();
226             if (connection.connectionID == request.connectionId ^
227                 connection.radio.equals(request.radio)) {
228                 connection.keepAlive = new Date();
229                 connIter.set(connection);
230                 System.out.println(" [x] Refreshing keepalive " +
231                     "from: " + user.username + " to radio: "
232                     + connection.radio + " id: " +
233                     connection.connectionID);
234                 break;
235             }
236         }
237
238         // add to user total listened minutes
239         user.secondsListening += Configuration.KeepAlivePeriodSeconds;
240
241         // update user
242         database.updateRow(user);
243     }
244 };
245     return channel.basicConsume(keepaliveUsersQueue, true,
246         keepaliveConsumer);
247 }
248
249     public static void main(String[] argv) throws Exception {
250
251         // define database
252         Database<UsersDBRow> database = new DatabaseRAM();
253
254         // start database handler
255         UsersDBHandler handler = new UsersDBHandler(Configuration.RabbitMQHost,
256             database);
257     }
258 }

```

Apr 12, 18 13:28

UserDisconnectRequest.java

Page 1/1

```

1 public class UserDisconnectRequest {
2
3     // request
4     public String username;
5     public String radio;
6     public int connectionId;
7
8     public UserDisconnectRequest(String username, String radio, int
9         connectionId) {
10         this.username = username;
11         this.radio = radio;
12         this.connectionId = connectionId;
13     }
14
15     public String toLogLine() {
16         return Configuration.LogsDisconnectionTag + " " + username + " " +
17             radio + " " + connectionId;
18     }
19 }

```

Apr 12, 18 13:31

UserConnectResponse.java

Page 1/1

```

1 import java.util.LinkedList;
2 import java.util.List;
3
4 public class UserConnectResponse {
5
6     // request
7     public String username;
8     public String radio;
9
10    // id
11    public String returnQueueName;
12
13    // response
14    public boolean couldConnect = false;
15    public int connectionId;
16    public List<UserDisconnectRequest> closedConnections = new LinkedList<>();
17
18    public UserConnectResponse(UserConnectRequest request) {
19        this.username = request.username;
20        this.radio = request.radio;
21        this.returnQueueName = request.returnQueueName;
22    }
23
24    public String toLogLine() {
25        return Configuration.LogsConnectionTag + " " + username + " " +
26            radio + " " + connectionId;
27    }
28 }

```

Apr 11, 18 20:36

UserConnectRequest.java

Page 1/1

```
1
2 public class UserConnectRequest {
3
4     // request
5     public String username;
6     public String radio;
7
8     // id
9     public String returnQueueName;
10
11     public UserConnectRequest(String username, String radio,
12                               String returnQueueName) {
13         this.username = username;
14         this.radio = radio;
15         this.returnQueueName = returnQueueName;
16     }
17 }
```

Apr 12, 18 13:31

RadiosUpdateRequest.java

Page 1/1

```
1 public class RadiosUpdateRequest {
2
3     public String radio;
4     public String username;
5
6     public RadiosUpdateRequest(String radio, String username) {
7         this.radio = radio;
8         this.username = username;
9     }
10 }
```

Apr 12, 18 2:53

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         channel.exchangeDeclare(Configuration.RadiosStatisticsExchange,
15             BuiltinExchangeType.FANOUT);
16
17         consumeStatistics();
18     }
19
20     private String consumeStatistics() throws IOException {
21         String statisticsQueue = channel.queueDeclare().getQueue();
22         channel.queueBind(statisticsQueue,
23             Configuration.RadiosStatisticsExchange, "");
24
25         Consumer consumerStatistics = new DefaultConsumer(channel) {
26             @Override
27             public void handleDelivery(String consumerTag, Envelope envelope,
28                 AMQP.BasicProperties properties,
29                 byte[] body) throws IOException {
30
31                 String json = new String(body, "UTF-8");
32                 RadiosConnectionsStatistics statistics = new Gson().fromJson(
33                     json, RadiosConnectionsStatistics.class);
34
35                 System.out.println("[X] Showing connections per radio: ");
36                 for (String radio : statistics.radioConnections.keySet()) {
37                     System.out.println(radio + ": " +
38                         statistics.radioConnections.get(radio));
39                 }
40             }
41         };
42         return channel.basicConsume(statisticsQueue, true, consumerStatistics);
43     }
44
45     public static void main(String[] argv) throws Exception {
46         RadioStatisticsViewer statisticsViewer =
47             new RadioStatisticsViewer(Configuration.RabbitMQHost);
48     }
49 }

```

Apr 12, 18 13:55

RadioSourceRandomNumbers.java

Page 1/1

```

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

```


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 12, 18 14:29

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, int bitrate) {
14         this.filename = filename;
15         this.bytesPerRead = 1000 * Configuration.RadioSendPeriodSeconds;
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             audio.read(buffer);
28             return Base64.getEncoder().encode(buffer);
29         } catch (IOException e) {
30             e.printStackTrace();
31         }
32         return "STATIC".getBytes();
33     }
34
35     @Override
36     public void close() {
37     }
38 }
39 }
```

Apr 11, 18 19:58

RadiosDBRow.java

Page 1/1

```

1 import java.util.LinkedList;
2 import java.util.List;
3
4 public class RadiosDBRow extends DatabaseRow {
5     public String name;
6     public int connectedUsers;
7
8     public RadiosDBRow(String name) {
9         super(name);
10
11         this.name = name;
12         this.connectedUsers = 0;
13     }
14 }

```

Apr 12, 18 3:24

RadiosDBHandler.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.LinkedList;
6 import java.util.List;
7 import java.util.concurrent.*;
8
9 public class RadiosDBHandler extends DBHandlerWithStatistics<RadiosDBRow> {
10
11     public RadiosDBHandler(String host, Database<RadiosDBRow> database) throws
12         IOException,
13         TimeoutException {
14         super(host, database);
15         this.database = database;
16
17         // declare RADIOS_DB exchange
18         channel.exchangeDeclare(Configuration.RadiosDBExchange,
19             BuiltinExchangeType.DIRECT);
20
21         // declare RADIOS_STATS exchange
22         channel.exchangeDeclare(Configuration.RadiosStatisticsExchange,
23             BuiltinExchangeType.FANOUT);
24
25         consumeConnections();
26         consumeDisconnections();
27     }
28
29     @Override
30     protected List<Runnable> getStatisticsOperations() {
31         List<Runnable> operations = new LinkedList<>();
32         operations.add(new Runnable() {
33             @Override
34             public void run() {
35                 // get statistics
36                 RadiosConnectionsStatistics stats = new RadiosConnectionsStatist
37                     ics();
38                 for (RadiosDBRow row : database.getRows()) {
39                     stats.radioConnections.put(row.name, row.connectedUsers);
40                 }
41                 String jsonStats = new Gson().toJson(stats);
42                 try {
43                     channel.basicPublish(Configuration.RadiosStatisticsExchange,
44                         "",
45                         null, jsonStats.getBytes());
46                 } catch (IOException e) {
47                     e.printStackTrace();
48                 }
49             }
50         });
51         return operations;
52     }
53
54     @Override
55     protected List<Integer> getStatisticsPeriodsSeconds() {
56         List<Integer> operationsPeriods = new LinkedList<>();
57         operationsPeriods.add(Configuration.RadioStatisticsPeriodSeconds);
58         return operationsPeriods;
59     }
60
61     private String consumeConnections() throws IOException {
62         // consume CONNECT_TO_RADIO requests
63         String connectUsersQueue = channel.queueDeclare().getQueue();
64         channel.queueBind(connectUsersQueue, Configuration.RadiosDBExchange,
65             Configuration.RadiosDBConnectTag);
66         Consumer connectConsumer = new DefaultConsumer(channel) {
67             @Override
68             public void handleDelivery(String consumerTag, Envelope envelope,
69                 AMQP.BasicProperties properties,
70                 byte[] body) throws IOException {
71

```

Apr 12, 18 3:24

RadiosDBHandler.java

Page 2/2

```

72 // parse request
73 String jsonRequest = new String(body, "UTF-8");
74
75 RadiosUpdateRequest request = new Gson().fromJson(jsonRequest,
76     RadiosUpdateRequest.class);
77
78 // get radio record from DB
79 RadiosDBRow radio = database.getRow(request.radio);
80 if (radio == null) {
81     radio = new RadiosDBRow(request.radio);
82 }
83
84 // add one connection to counter
85 radio.connectedUsers++;
86 System.out.println("[x] Adding one connection to radio: " +
87     radio.name);
88
89 // save changes to db
90 database.updateRow(radio);
91 }
92 };
93 return channel.basicConsume(connectUsersQueue, true, connectConsumer);
94 }
95
96 private String consumeDisconnections() throws IOException {
97     // consume DISCONNECT_FROM_RADIO requests
98     String disconnectUsersQueue = channel.queueDeclare().getQueue();
99     channel.queueBind(disconnectUsersQueue, Configuration.RadiosDBExchange,
100         Configuration.RadiosDBDisconnectTag);
101     Consumer disconnectConsumer = new DefaultConsumer(channel) {
102         @Override
103         public void handleDelivery(String consumerTag, Envelope envelope,
104             AMQP.BasicProperties properties,
105             byte[] body) throws IOException {
106
107             // parse request
108             String jsonRequest = new String(body, "UTF-8");
109
110             RadiosUpdateRequest request = new Gson().fromJson(jsonRequest,
111                 RadiosUpdateRequest.class);
112
113             // get radio record from DB
114             RadiosDBRow radio = database.getRow(request.radio);
115             if (radio == null) {
116                 radio = new RadiosDBRow(request.radio);
117             }
118
119             // add one connection to counter
120             radio.connectedUsers--;
121             System.out.println("[x] Removing one connection from " +
122                 "radio: " + radio.name);
123
124             // save changes to db
125             database.updateRow(radio);
126
127         }
128     };
129     return channel.basicConsume(disconnectUsersQueue, true,
130         disconnectConsumer);
131 }
132
133 public static void main(String[] argv) throws Exception {
134     // define database
135     Database<RadiosDBRow> database = new DatabaseRAM();
136
137     // start database handler
138     RadiosDBHandler handler = new RadiosDBHandler(Configuration.RabbitMQHost
139         ,
140         database);
141 }

```

Apr 12, 18 2:36

RadiosConnectionsStatistics.java

Page 1/1

```

1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class RadiosConnectionsStatistics {
5
6     public Map<String, Integer> radioConnections = new HashMap<>();
7 }

```

Apr 12, 18 14:20

Radio.java

Page 1/1

```

1 import com.rabbitmq.client.*;
2
3 import java.io.IOException;
4 import java.util.concurrent.*;
5
6 public class Radio extends RabbitMQProcess {
7
8     private String exchangeName;
9     private ScheduledExecutorService transmissionScheduler;
10    private ScheduledFuture<?> transmissionHandle;
11
12    private RadioSource source;
13
14    public Radio(String host, String radioName, RadioSource source) throws
15        IOException, TimeoutException {
16        super(host);
17
18        this.source = source;
19        source.init();
20
21        // declare BROADCAST exchange
22        exchangeName = Configuration.RadioExchangePrefix + radioName;
23        channel.exchangeDeclare(exchangeName, BuiltinExchangeType.FANOUT);
24
25        scheduleTransmission();
26    }
27
28    private void scheduleTransmission() {
29        transmissionScheduler = Executors
30            .newScheduledThreadPool(1);
31
32        transmissionHandle =
33            transmissionScheduler.scheduleAtFixedRate(new Runnable() {
34                @Override
35                public void run() {
36                    byte[] nextBlock = source.getNextByteBlock();
37
38                    try {
39                        channel.basicPublish(exchangeName, "", null, nextBlock);
40                    } catch (IOException e) {
41                        e.printStackTrace();
42                    }
43                }, Configuration.RadioSendPeriodSeconds,
44                    Configuration.RadioSendPeriodSeconds,
45                    TimeUnit.SECONDS);
46    }
47
48    @Override
49    protected void close() throws IOException, TimeoutException {
50        super.close();
51
52        transmissionHandle.cancel(true);
53        transmissionScheduler.shutdown();
54        source.close();
55    }
56
57    public static void main(String[] argv) throws Exception {
58        RadioSource source = argv.length == 3 ?
59            new RadioSourceFile(argv[1],
60                Integer.parseInt(argv[2]) * 1024) :
61            new RadioSourceRandomNumbers();
62        Radio radio = new Radio(Configuration.RabbitMQHost, argv[0], source);
63    }
64
65
66 }

```

Apr 12, 18 13:53

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 class RabbitMQProcess {
9
10    protected Connection connection;
11    protected 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        addShutdownHook();
25    }
26
27    public void addShutdownHook() {
28
29        RabbitMQProcess instance = this;
30        Thread mainThread = Thread.currentThread();
31        Runtime.getRuntime().addShutdownHook(new Thread() {
32            public void run() {
33
34                System.out.println("Calling shutdown hook");
35
36                try {
37                    instance.close();
38                } catch (IOException e) {
39                    e.printStackTrace();
40                } catch (TimeoutException e) {
41                    e.printStackTrace();
42                }
43                try {
44                    mainThread.join();
45                } catch (InterruptedException e) {
46                    e.printStackTrace();
47                }
48            }
49        });
50    }
51
52    protected void close() throws IOException, TimeoutException {
53        channel.close();
54        connection.close();
55    }
56
57 }

```

Apr 07, 18 11:39

Logger.java

Page 1/2

```

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

```

Apr 07, 18 11:39

Logger.java

Page 2/2

```

74         if (logWriter != null) {
75             logWriter.println(logLine);
76         }
77     }
78
79     public static void output(String outString) {
80         System.out.println(outString);
81         logWriter.println(outString);
82     }
83
84 }

```

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 11, 18 21:34

KeepAliveRequest.java

Page 1/1

```

1  public class KeepAliveRequest {
2
3      public String username;
4      public int connectionId;
5      public String radio;
6
7      public KeepAliveRequest( String username, int connectionId, String radio ) {
8          this.username = username;
9          this.connectionId = connectionId;
10         this.radio = radio;
11     }
12 }

```

Apr 12, 18 2:00

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         channel.exchangeDeclare(Configuration.UsersDBExchange,
14             BuiltinExchangeType.DIRECT);
15
16         consumeKeepAlives();
17     }
18
19     private String consumeKeepAlives() throws IOException {
20         // KEEP ALIVE consumer
21         channel.queueDeclare(Configuration.KeepAliveQueue, true, false, false,
22             null);
23         Consumer consumer_keepalive = new DefaultConsumer(channel) {
24             @Override
25             public void handleDelivery(String consumerTag, Envelope envelope,
26                 AMQP.BasicProperties properties,
27                 byte[] body) throws IOException {
28
29                 String json = new String(body, "UTF-8");
30                 KeepAliveRequest request = new Gson().fromJson(json,
31                     KeepAliveRequest.class);
32
33                 System.out.println("[X] Received keep alive request from: "
34                     + request.username + " to: " + request.radio + " id: " +
35                     request.connectionId);
36
37                 // ask usersDB to register connection
38                 channel.basicPublish(Configuration.UsersDBExchange,
39                     Configuration.UsersDBKeepAliveTag, null,
40                     new Gson().toJson(request).getBytes());
41             }
42         };
43         return channel.basicConsume(Configuration.KeepAliveQueue, true,
44             consumer_keepalive);
45     }
46
47     public static void main(String[] argv) throws Exception {
48         KeepAliveManager manager =
49             new KeepAliveManager(Configuration.RabbitMQHost);
50     }
51 }

```

Apr 12, 18 13:55

FileLogger.java

Page 1/1

```

1 import com.google.gson.Gson;
2 import com.rabbitmq.client.*;
3
4 import java.io.FileWriter;
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         channel.exchangeDeclare(Configuration.LogsExchange,
23             BuiltinExchangeType.DIRECT);
24
25         logWriter = new PrintWriter(new FileWriter(logFilename, true));
26
27         logsQueue = channel.queueDeclare().getQueue();
28         for (String tag : getBindings()) {
29             channel.queueBind(logsQueue, Configuration.LogsExchange, tag);
30         }
31         consumeLogs();
32     }
33
34     protected abstract List<String> getBindings();
35
36     public String consumeLogs() throws IOException {
37         // consume connection logs
38         Consumer connectConsumer = new DefaultConsumer(channel) {
39             @Override
40             public void handleDelivery(String consumerTag, Envelope envelope,
41                 AMQP.BasicProperties properties,
42                 byte[] body) throws IOException {
43
44                 // write log to file
45                 String logLine = new String(body, "UTF-8");
46                 logWriter.println(logLine);
47
48                 System.out.println("[X] Received: " + logLine);
49             }
50         };
51         return channel.basicConsume(logsQueue, true, connectConsumer);
52     }
53
54     @Override
55     protected void close() throws IOException, TimeoutException {
56         super.close();
57         logWriter.close();
58     }
59 }

```

Apr 12, 18 12:19

DisconnectionManager.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 DisconnectionManager extends RabbitMQProcess {
8
9     public DisconnectionManager(String host) throws IOException,
10         TimeoutException {
11         super(host);
12
13         // declare USERS_DB exchange
14         channel.exchangeDeclare(Configuration.UsersDBExchange,
15             BuiltinExchangeType.DIRECT);
16
17         // declare RADIOS_DB exchange
18         channel.exchangeDeclare(Configuration.RadiosDBExchange,
19             BuiltinExchangeType.DIRECT);
20
21         // declare LOGS exchange
22         channel.exchangeDeclare(Configuration.LogsExchange,
23             BuiltinExchangeType.DIRECT);
24
25         consumeDisconnections();
26
27     private String consumeDisconnections() throws IOException {
28         // DISCONNECTIONS consumer
29         channel.queueDeclare(Configuration.DisconnectionsQueue, true, false, false,
30             null);
31         Consumer consumer_disconnect = new DefaultConsumer(channel) {
32             @Override
33             public void handleDelivery(String consumerTag, Envelope envelope,
34                 AMQP.BasicProperties properties,
35                 byte[] body) throws IOException {
36
37                 String json = new String(body, "UTF-8");
38                 UserDisconnectRequest request = new Gson().fromJson(json,
39                     UserDisconnectRequest.class);
40
41                 System.out.println("[X] Received request to disconnect user: " +
42                     " " + request.username + " from: " + request.radio);
43
44                 // ask usersDB to register disconnection
45                 channel.basicPublish(Configuration.UsersDBExchange,
46                     Configuration.UsersDBDisconnectTag, null,
47                     new Gson().toJson(request).getBytes());
48
49                 // ask radiosDB to register disconnection
50                 channel.basicPublish(Configuration.RadiosDBExchange,
51                     Configuration.RadiosDBDisconnectTag, null,
52                     new Gson().toJson(request).getBytes());
53
54                 // send disconnects to file logger
55                 channel.basicPublish(Configuration.LogsExchange,
56                     Configuration.LogsDisconnectionTag, null,
57                     request.toLogLine().getBytes());
58             }
59         };
60         return channel.basicConsume(Configuration.DisconnectionsQueue, true,
61             consumer_disconnect);
62     }
63
64     public static void main(String[] argv) throws Exception {
65         DisconnectionManager manager =
66             new DisconnectionManager(Configuration.RabbitMQHost);
67     }
68 }

```

Apr 12, 18 13:55

DBHandlerWithStatistics.java

Page 1/1

```

1 import com.google.gson.Gson;
2
3 import java.io.IOException;
4 import java.util.LinkedList;
5 import java.util.List;
6 import java.util.concurrent.*;
7
8 public abstract class DBHandlerWithStatistics<T extends DatabaseRow>
9     extends RabbitMQProcess {
10
11     Database<T> database;
12     private ScheduledExecutorService statisticsScheduler = null;
13     private List<ScheduledFuture<?>> statisticsHandles = null;
14
15     public DBHandlerWithStatistics(String host, Database database) throws
16         IOException,
17         TimeoutException {
18         super(host);
19         this.database = database;
20
21         List<Runnable> statisticTasks = getStatisticsOperations();
22         List<Integer> statisticTasksPeriods = getStatisticsPeriodsSeconds();
23         if (statisticTasks.size() > 0) {
24             statisticsScheduler = Executors
25                 .newScheduledThreadPool(1);
26             statisticsHandles = new LinkedList<>();
27
28             for (int i = 0; i < statisticTasks.size(); ++i) {
29                 Runnable r = statisticTasks.get(i);
30                 int period = statisticTasksPeriods.get(i);
31                 ScheduledFuture<?> statisticsHandle =
32                     statisticsScheduler.scheduleAtFixedRate(r, period,
33                         period, TimeUnit.SECONDS);
34                 statisticsHandles.add(statisticsHandle);
35             }
36         }
37     }
38
39     @Override
40     protected void close() throws IOException, TimeoutException {
41         super.close();
42
43         if (statisticsScheduler != null) {
44             for (ScheduledFuture<?> f : statisticsHandles) {
45                 f.cancel(true);
46             }
47             statisticsScheduler.shutdown();
48         }
49     }
50
51     protected abstract List<Runnable> getStatisticsOperations();
52
53     protected abstract List<Integer> getStatisticsPeriodsSeconds();
54 }

```


Apr 11, 18 19:12

DatabaseRow.java

Page 1/1

```

1 public abstract class DatabaseRow {
2
3     public String primary_key;
4
5     public DatabaseRow(String primary_key) {
6         this.primary_key = primary_key;
7     }
8 }

```

Apr 12, 18 2:38

DatabaseRAM.java

Page 1/1

```

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

```

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 12, 18 13:34

ConnectionManager.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 ConnectionManager extends RabbitMQProcess {
8
9     public ConnectionManager(String host) throws IOException, TimeoutException {
10         super(host);
11
12         // declare USERS_DB exchange
13         channel.exchangeDeclare(Configuration.UsersDBExchange,
14             BuiltinExchangeType.DIRECT);
15
16         // declare usersDB responses queue
17         channel.queueDeclare(Configuration.ConnMgrUsersDBResponseQueue,
18             true, false, false, null);
19
20         // declare RADIOS_DB exchange
21         channel.exchangeDeclare(Configuration.RadiosDBExchange,
22             BuiltinExchangeType.DIRECT);
23
24         // declare LOGS exchange
25         channel.exchangeDeclare(Configuration.LogsExchange,
26             BuiltinExchangeType.DIRECT);
27
28         consumeConnections();
29         consumeUsersDB();
30     }
31
32     private String consumeConnections() throws IOException {
33         // CONNECTIONS consumer
34         channel.queueDeclare(Configuration.ConnectionsQueue,
35             true, false, false, null);
36         Consumer consumer_connect = new DefaultConsumer(channel) {
37             @Override
38             public void handleDelivery(String consumerTag, Envelope envelope,
39                 AMQP.BasicProperties properties,
40                 byte[] body) throws IOException {
41
42                 String json = new String(body, "UTF-8");
43                 UserConnectRequest request = new Gson().fromJson(json,
44                     UserConnectRequest.class);
45
46                 System.out.println("[X] Received connection request from: "
47                     + request.username + " to: " + request.radio);
48
49                 // ask usersDB to register connection
50                 channel.basicPublish(Configuration.UsersDBExchange,
51                     Configuration.UsersDBConnectTag, null,
52                     new Gson().toJson(request).getBytes());
53             }
54         };
55         return channel.basicConsume(Configuration.ConnectionsQueue,
56             true, consumer_connect);
57     }
58
59     private String consumeUsersDB() throws IOException {
60         // usersDB consume
61         Consumer consumer_usersdb = new DefaultConsumer(channel) {
62             @Override
63             public void handleDelivery(String consumerTag, Envelope envelope,
64                 AMQP.BasicProperties properties,
65                 byte[] body) throws IOException {
66
67                 String json = new String(body, "UTF-8");
68                 UserConnectResponse response = new Gson().fromJson(json,
69                     UserConnectResponse.class);
70
71                 for (UserDisconnectRequest disconn :
72                     response.closedConnections) {
73

```

Apr 12, 18 13:34

ConnectionManager.java

Page 2/2

```

74 // register closed connections in radios DB
75 RadiosUpdateRequest radiosRequest =
76     new RadiosUpdateRequest(disconn.radio,
77         disconn.username);
78 channel.basicPublish(Configuration.RadiosDBExchange,
79     Configuration.RadiosDBDisconnectTag, null,
80     new Gson().toJson(radiosRequest).getBytes());
81
82 // send disconnects to file logger
83 channel.basicPublish(Configuration.LogsExchange,
84     Configuration.LogsDisconnectionTag, null,
85     disconn.toLogLine().getBytes());
86
87
88 if (response.couldConnect) {
89     // send connect to file logger
90     channel.basicPublish(Configuration.LogsExchange,
91         Configuration.LogsConnectionTag, null,
92         response.toLogLine().getBytes());
93
94     // register connection in radios DB
95     RadiosUpdateRequest radioConnectRequest =
96         new RadiosUpdateRequest(response.radio,
97             response.username);
98     channel.basicPublish(Configuration.RadiosDBExchange,
99         Configuration.RadiosDBConnectTag, null,
100         new Gson().toJson(radioConnectRequest).getBytes());
101
102     System.out.println(" [X] User: " + response.username +
103         " connected to radio: " + response.radio);
104 } else {
105     System.out.println(" [X] User: " + response.username +
106         " denied connection to radio: " + response.radio);
107 }
108
109 String jsonResponse = new Gson().toJson(response);
110 channel.basicPublish("", response.returnQueueName, null,
111     jsonResponse.getBytes());
112 }
113 };
114 return channel.basicConsume(Configuration.ConnMgrUsersDBResponseQueue,
115     true, consumer_usersdb);
116 }
117
118 public static void main(String[] argv) throws Exception {
119
120     ConnectionManager manager =
121         new ConnectionManager(Configuration.RabbitMQHost);
122 }
123
124 }

```

Apr 12, 18 13:21

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     }
27 }

```

Apr 12, 18 14:18

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
9  public class Configuration {
10
11     public static int RadioSendPeriodSeconds = 2;
12     public static int KeepAlivePeriodSeconds = 5;
13     public static int SecondsUntilDropConnection = 10;
14     public static int MaxConnectionsPerFreeUser = 3;
15     public static int MaxConnectionsPerUnlimitedUser = 999;
16
17     public static String RabbitMQHost = "localhost";
18
19     public static String UsersDBExchange = "USERS_DB";
20     public static String UsersDBConnectTag = "connect";
21     public static String UsersDBDisconnectTag = "disconnect";
22     public static String UsersDBKeepAliveTag = "keepalive";
23     public static String UsersStatisticsExchange = "USERS_STATS";
24     public static int UsersStatisticsPeriodSeconds = 10;
25     public static int UserStatisticsN = 100;
26
27     public static String RadiosDBExchange = "RADIOS_DB";
28     public static String RadiosDBConnectTag = "connect";
29     public static String RadiosDBDisconnectTag = "disconnect";
30     public static String RadiosStatisticsExchange = "RADIOS_STATS";
31     public static int RadioStatisticsPeriodSeconds = 10;
32
33     public static String ConnMgrUsersDBResponseQueue =
34         "usersDBResponseQueueName";
35
36     public static String ConnectionsQueue = "CONNECTIONS";
37     public static String DisconnectionsQueue = "DISCONNECTIONS";
38     public static String KeepAliveQueue = "KEEP_ALIVE";
39
40     public static String RadioExchangePrefix = "BROADCAST-";
41
42     public static String LogsExchange = "LOGS";
43     public static String LogsConnectionTag = "connect";
44     public static String LogsDisconnectionTag = "disconnect";
45
46     public static boolean loadConfiguration(String configFilename) {
47
48         try {
49             // read json config
50             BufferedReader br = new BufferedReader(
51                 new FileReader( configFilename));
52             String jsonString = "";
53             String s;
54             while ((s = br.readLine()) != null) {
55                 jsonString += s;
56             }
57
58             // esto es para que gson serialize variables estaticas
59             GsonBuilder gsonBuilder = new GsonBuilder();
60             gsonBuilder.excludeFieldsWithModifiers(
61                 java.lang.reflect.Modifier.TRANSIENT);
62
63             Gson gson = gsonBuilder.create();
64             // load to object
65             Configuration config = gson.fromJson(jsonString,
66                 Configuration.class);
67
68             return true;
69         }
70         catch (FileNotFoundException e) {
71             /*Logger.log("Monitor", "config file: " + configFilename +
72                 " not found", Logger.logLevel.ERROR);*/
73         } catch (IOException e) {

```

Apr 12, 18 14:18

Configuration.java

Page 2/2

```

74             /*Logger.log("Monitor", "config file: " + configFilename +
75                 " could not be read", Logger.logLevel.ERROR);*/
76         }
77
78         return false;
79     }
80 }

```

Apr 12, 18 14:32

Client.java

Page 1/4

```

1 import com.google.gson.Gson;
2 import com.rabbitmq.client.*;
3
4 import java.io.FileOutputStream;
5 import java.io.FileWriter;
6 import java.io.IOException;
7 import java.text.SimpleDateFormat;
8 import java.util.Base64;
9 import java.util.Date;
10 import java.util.NoSuchElementException;
11 import java.util.Scanner;
12 import java.util.concurrent.*;
13
14 public class Client extends RabbitMQProcess {
15
16     private String radioExchange = "";
17
18     private String username = "";
19     private String radio;
20     private int connectionId;
21     private String radioConsumeTag = "";
22     FileOutputStream transmissionWriter = null;
23
24     private ScheduledExecutorService keepAliveScheduler =
25         Executors.newScheduledThreadPool(1);
26     private ScheduledFuture<?> keepAliveHandle;
27
28     public Client(String host) throws IOException, TimeoutException {
29         super(host);
30     }
31
32     public void setUsername(String username) {
33         this.username = username;
34     }
35
36     public void setRadio(String radio) {
37         this.radio = radio;
38     }
39
40     public boolean requestConnectionToRadio() throws IOException,
41         InterruptedException {
42
43         if (username.equals("")) {
44             System.out.println("ERROR: Did you specify a username?");
45             return false;
46         }
47
48         // define callback queue
49         String callbackQueueName = channel.queueDeclare().getQueue();
50
51         // create request
52         UserConnectRequest request = new UserConnectRequest(username, radio,
53             callbackQueueName);
54         String requestJson = new Gson().toJson(request);
55
56         // publish to CONNECTIONS queue
57         channel.basicPublish("", Configuration.ConnectionsQueue, null,
58             requestJson.getBytes());
59
60         final BlockingQueue<String> responseQueue =
61             new ArrayBlockingQueue<String>(1);
62
63         String callbackTag = channel.basicConsume(callbackQueueName, true,
64             new DefaultConsumer(channel) {
65                 @Override
66                 public void handleDelivery(String consumerTag, Envelope envelope,
67                     AMQP.BasicProperties properties,
68                     byte[] body) throws IOException {
69                     responseQueue.offer(new String(body, "UTF-8"));
70                 }
71             });
72
73         String jsonResponse = responseQueue.take();

```

Apr 12, 18 14:32

Client.java

Page 2/4

```

74         channel.basicCancel(callbackTag);
75         UserConnectResponse response = new Gson().fromJson(jsonResponse,
76             UserConnectResponse.class);
77         if (!response.couldConnect) {
78             System.out.println("ERROR: Connection refused, are " +
79                 "you already connected on 3 devices?");
80             return false;
81         }
82
83         connectionId = response.connectionId;
84         radioExchange = Configuration.RadioExchangePrefix + response.radio;
85         return true;
86     }
87
88     public boolean listenToRadio() throws IOException {
89
90         if (radioExchange.equals("")) {
91             return false;
92         }
93
94         // declare radio broadcast exchange
95         channel.exchangeDeclare(radioExchange, BuiltinExchangeType.FANOUT);
96
97         // declare temporary queue and bind
98         String queueName = channel.queueDeclare().getQueue();
99         channel.queueBind(queueName, radioExchange, "");
100         System.out.println("Creating queue: " + queueName);
101
102         // open new file for transmission
103         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss");
104         String transmissionName = "client" + "-" + username + "-" + radio +
105             "-" + connectionId + "-" + sdf.format(new Date()) + ".wav";
106         transmissionWriter = new FileOutputStream(transmissionName);
107
108         Consumer consumer = new DefaultConsumer(channel) {
109             @Override
110             public void handleDelivery(String consumerTag, Envelope envelope,
111                 AMQP.BasicProperties properties,
112                 byte[] body) throws IOException {
113                 String message = new String(body, "UTF-8");
114                 System.out.println("[x] Received '" + message + "'");
115                 transmissionWriter.write(body);
116             }
117         };
118         radioConsumeTag = channel.basicConsume(queueName, true, consumer);
119         return true;
120     }
121
122     public void scheduleKeepAlive() {
123         final Runnable sendKeepAlive = new Runnable() {
124             @Override
125             public void run() {
126                 KeepAliveRequest request = new KeepAliveRequest(username,
127                     connectionId, radio);
128                 String requestJson = new Gson().toJson(request);
129                 try {
130                     channel.basicPublish("", Configuration.KeepAliveQueue,
131                         null, requestJson.getBytes());
132                 } catch (IOException e) {
133                     e.printStackTrace();
134                 }
135             }
136         };
137         keepAliveHandle = keepAliveScheduler.scheduleAtFixedRate
138             (sendKeepAlive, 5, 5, TimeUnit.SECONDS);
139     }
140
141     public void stopKeepAlive() {
142         keepAliveScheduler.schedule(new Runnable() {
143             @Override
144             public void run() {
145                 keepAliveHandle.cancel(true);
146             }

```

Apr 12, 18 14:32

Client.java

Page 3/4

```

147     }, 0, TimeUnit.SECONDS);
148 }
149
150 public void stopListeningToRadio() throws IOException {
151     if (radioConsumeTag.equals("")) {
152         System.out.println("ERROR: not listening to radio");
153     } else {
154         // create request
155         UserDisconnectRequest request = new UserDisconnectRequest(username,
156             radio, connectionId);
157         String requestJson = new Gson().toJson(request);
158
159         // publish to DISCONNECTIONS queue
160         channel.basicPublish("", Configuration.DisconnectionsQueue, null,
161             requestJson.getBytes());
162
163         // stop receiving transmission
164         channel.basicCancel(radioConsumeTag);
165         radioConsumeTag = "";
166
167         // close transmission file
168         transmissionWriter.close();
169         transmissionWriter = null;
170     }
171 }
172
173 public void printOptions() {
174     System.out.println("\n");
175     System.out.println("Choose an action: ");
176     System.out.println("\t" + "1. Set user");
177     System.out.println("\t" + "2. Connect to radio");
178     System.out.println("\t" + "3. Disconnect from radio");
179     System.out.println("\t" + "4. Exit");
180 }
181
182 public boolean mainMenu(Scanner in) throws IOException,
183     InterruptedException {
184     String choiceStr = in.nextLine();
185     int choice = Integer.parseInt(choiceStr);
186     switch (choice) {
187         case 1:
188             System.out.print("Please specify a username: ");
189             String username = in.nextLine();
190             setUsername(username);
191             break;
192         case 2:
193             System.out.println("Please specify a radio: ");
194             String radio = in.nextLine();
195             setRadio(radio);
196             if (!requestConnectionToRadio()) {
197                 break;
198             }
199             listenToRadio();
200             scheduleKeepAlive();
201             break;
202         case 3:
203             stopListeningToRadio();
204             stopKeepAlive();
205             break;
206         case 4:
207             System.out.println("Press CTRL+C to exit");
208             return true;
209         default:
210             System.out.println("ERROR: Invalid option");
211             break;
212     }
213
214     printOptions();
215     return false;
216 }
217
218 @Override
219 protected void close() throws IOException, TimeoutException {

```

Apr 12, 18 14:32

Client.java

Page 4/4

```

220     super.close();
221     if (transmissionWriter != null) {
222         transmissionWriter.close();
223     }
224 }
225
226
227 public static void main(String[] argv) throws Exception {
228
229     Scanner in = new Scanner(System.in);
230
231     Client client = new Client(Configuration.RabbitMQHost);
232     client.printOptions();
233
234     boolean end = false;
235     while (!end) {
236         try {
237             end = client.mainMenu(in);
238         } catch (NoSuchElementException e) {
239             end = true;
240         }
241     }
242 }
243
244 }

```

Apr 12, 18 14:52	Table of Content	Page 1/1
1	Table of Contents	
2	1 <i>UserStatisticsViewer.java</i> sheets 1 to 1 (1) pages 1- 1 50 lines	
3	2 <i>UsersSecondsListenedStatistics.java</i> sheets 1 to 1 (1) pages 2- 2 13 lines	
4	3 <i>UsersSecondsListenedComparator.java</i> sheets 2 to 2 (1) pages 3- 3 18 lines	
5	4 <i>UserSecondsListened.java</i> sheets 2 to 2 (1) pages 4- 4 11 lines	
6	5 <i>UsersDBRowRadioConnection.java</i> sheets 3 to 3 (1) pages 5- 5 17 lines	
7	6 <i>UsersDBRow.java</i> sheets 3 to 3 (1) pages 6- 6 19 lines	
8	7 <i>UsersDBHandler.java</i> sheets 4 to 5 (2) pages 7- 10 259 lines	
9	8 <i>UserDisconnectRequest.java</i> sheets 6 to 6 (1) pages 11- 11 20 lines	
10	9 <i>UserConnectResponse.java</i> sheets 6 to 6 (1) pages 12- 12 29 lines	
11	10 <i>UserConnectRequest.java</i> sheets 7 to 7 (1) pages 13- 13 18 lines	
12	11 <i>RadiosUpdateRequest.java</i> sheets 7 to 7 (1) pages 14- 14 11 lines	
13	12 <i>RadioStatisticsViewer.java</i> sheets 8 to 8 (1) pages 15- 15 50 lines	
14	13 <i>RadioSourceRandomNumbers.java</i> sheets 8 to 8 (1) pages 16- 16 23 lines	
15	14 <i>RadioSource.java</i> sheets 9 to 9 (1) pages 17- 17 11 lines	
16	15 <i>RadioSourceFile.java</i> sheets 9 to 9 (1) pages 18- 18 40 lines	
17	16 <i>RadiosDBRow.java</i> sheets 10 to 10 (1) pages 19- 19 15 lines	
18	17 <i>RadiosDBHandler.java</i> sheets 10 to 11 (2) pages 20- 21 142 lines	
19	18 <i>RadiosConnectionsStatistics.java</i> sheets 11 to 11 (1) pages 22- 22 8 lines	
20	19 <i>Radio.java</i> sheets 12 to 12 (1) pages 23- 23 67 lines	
21	20 <i>RabbitMQProcess.java</i> sheets 12 to 12 (1) pages 24- 24 58 lines	
22	21 <i>Logger.java</i> sheets 13 to 13 (1) pages 25- 26 85 lines	
23	22 <i>LimitedSortedSet.java</i> sheets 14 to 14 (1) pages 27- 27 41 lines	
24	23 <i>KeepAliveRequest.java</i> sheets 14 to 14 (1) pages 28- 28 13 lines	
25	24 <i>KeepAliveManager.java</i> sheets 15 to 15 (1) pages 29- 29 52 lines	
26	25 <i>FileLogger.java</i> sheets 15 to 15 (1) pages 30- 30 60 lines	
27	26 <i>DisconnectionManager.java</i> sheets 16 to 16 (1) pages 31- 31 70 lines	
28	27 <i>DBHandlerWithStatistics.java</i> sheets 16 to 16 (1) pages 32- 32 55 lines	
29	28 <i>DatabaseRow.java</i> sheets 17 to 17 (1) pages 33- 33 9 lines	
30	29 <i>DatabaseRAM.java</i> sheets 17 to 17 (1) pages 34- 34 45 lines	
31	30 <i>Database.java</i> sheets 18 to 18 (1) pages 35- 35 15 lines	
32	31 <i>ConnectionManager.java</i> sheets 18 to 19 (2) pages 36- 37 125 lines	
33	32 <i>ConnDisconnFileLogger.java</i> sheets 19 to 19 (1) pages 38- 38 28 lines	
34	33 <i>Configuration.java</i> .. sheets 20 to 20 (1) pages 39- 40 81 lines	
35	34 <i>Client.java</i> sheets 21 to 22 (2) pages 41- 44 245 lines	