package com.dai.webServer.Mqtt; import java.sql.PreparedStatement; import java.sql.SQLException; import org.json.simple.*; import org.json.simple.parser.JSONParser; import org.json.simple.parser.ParseException;

import com.dai.db.AnalyticsDB; import com.dai.webServer.Conexao.Conexao; import com.dai.webServer.Conexao.Email; import com.dai.webServer.Mqtt.*; import org.json.simple.parser.ParseException; import org.apache.commons.text.RandomStringGenerator;*

/**********

* Licensed under the Apache License, Version 2.0 (the "License"); * you may not use this file except in compliance with the License. * You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.

**********/

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken; import org.eclipse.paho.client.mqttv3.MqttCallback; import org.eclipse.paho.client.mqttv3.MqttClient; import org.eclipse.paho.client.mqttv3.MqttConnectOptions; import org.eclipse.paho.client.mqttv3.MqttException; import org.eclipse.paho.client.mqttv3.MqttMessage; import org.springframework.beans.factory.annotation.Autowired; import java.util.Random; import static org.apache.commons.text.CharacterPredicates.LETTERS;

public class ReceiveRequests implements MqttCallback {

```
@Autowired

/** The broker url. */
private static final String brokerUrl = "tcp://alvesvitor.ddns.net:80";

/** The client id. */
Random rand = new Random();

/** The topic. */
public static final String topic = "request/#";
//private AnalyticsDB db = new AnalyticsDB();
private ApproveRequests ap = new ApproveRequests();

public String random() {
RandomStringGenerator generator = new RandomStringGenerator.Builder()
        .withinRange('0', 'z')
        .filteredBy(LETTERS)
        .build();
String a = generator.toString();
return a;
}
```

```
public void subscribe(String clientId) {


    try {
        String a = random();
        MqttClient sampleClient = new MqttClient(brokerUrl, a);
        MqttConnectOptions connOpts = new MqttConnectOptions();
        connOpts.setMqttVersion(MqttConnectOptions.MQTT_VERSION_3_1_1);
        connOpts.setUserName("dai");
        String password = "12345678";
        connOpts.setPassword(password.toCharArray());

        connOpts.setCleanSession(true);

        System.out.println("checking");

        System.out.println("Mqtt Connecting to broker: " + brokerUrl);
        sampleClient.connect(connOpts);
        System.out.println("Mqtt Connected");

        sampleClient.setCallback(this);
        sampleClient.subscribe(topic);

        System.out.println("Subscribed");
        System.out.println("Listening");

    } catch (MqttException me) {

        System.out.println("Mqtt reason " + me.getReasonCode());
        System.out.println("Mqtt msg " + me.getMessage());
        System.out.println("Mqtt loc " + me.getLocalizedMessage());
        System.out.println("Mqtt cause " + me.getCause());
        System.out.println("Mqtt excep " + me);
    }
}
```

```
public void connectionLost(Throwable arg0) {
    try {
        wait(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
```

```
  this.subscribe(topic);
}

public void deliveryComplete(IMqttDeliveryToken arg0) {

}

public void messageArrived(String topic, MqttMessage message) throws Exception  {

    System.out.println("Mqtt topic : " + topic);

    System.out.println("Mqtt msg : " + message.toString());
    byte[] hey = message.getPayload();
    String str = new String(hey, "UTF-8"); // for UTF-8 encoding
    insert(str, topic);
}

 public void insert(String message , String topic) throws ParseException, SQLException  {

    String tag = topic.substring(topic.lastIndexOf('/')+1);

    String correctedTopic = topic.substring(topic.lastIndexOf('/')+1);


    AnalyticsDB armed = new AnalyticsDB();




    System.out.println("is--------- Armed");




    AnalyticsDB db = new AnalyticsDB();


    AnalyticsDB mail = new AnalyticsDB();
    System.out.println("It's here now");
    System.out.println(correctedTopic);
            String outcome = db.approve(message, correctedTopic);

    System.out.println(outcome);
    String responseTopic = "response/" + correctedTopic;

    String openDoor = "relay/" + correctedTopic + "/rele1";
```

```
   String payLoadOpen = "3";
   String approved = "Bem vindo a casa " + outcome;

   String denied = "Por favor tente de novo";

   if (outcome != null){

       ap.sendMessage(responseTopic, approved, message);

       ap.sendMessage(openDoor, payLoadOpen, message);
       System.out.println("door opened");
       closeDoor(openDoor, message);


       AnalyticsDB insert = new AnalyticsDB();
       insert.insertDB(message, outcome);


       Long id_division = armed.findIdDivision(correctedTopic);

       AnalyticsDB updateArm = new AnalyticsDB();

       updateArm.updateArm("false", id_division);


   }else{

        String email = mail.readEmail(correctedTopic);
           System.out.println(email);


       Email a = new Email();

       a.sendEmail(email, "<img src='https://i.imgur.com/dbsHhsJ.png' alt='deu merda a carı

       AnalyticsDB insert = new AnalyticsDB();
       insert.insertDBNot(message, outcome);
       ap.sendMessage(responseTopic, denied, message);

   }
}

public void closeDoor(String openDoor, String message){
```

```
    try { Thread.sleep (1000); } catch (InterruptedException ex) {}


    ap.sendMessage(openDoor, "2", message);
    ap.sendMessage(openDoor, "1", message);

    try { Thread.sleep (1000); } catch (InterruptedException ex) {}


    ap.sendMessage(openDoor, "2", message);

    System.out.println("door closed");

 }
}
```