package com.dai.webServer.Mqtt; import java.sql.SQLException;

import com.dai.db.Database;

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.apache.commons.text.StringEscapeUtils; 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.MqttExceptior
import org.eclipse.paho.client.mqttv3.MqttMessage; import org.springframework.beans.factory.annotation.Autow
import java.util.Random; import static org.apache.commons.text.CharacterPredicates.LETTERS;

public class Listener 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 = "#";
private Database db = new Database();


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(db.getCurrentTimeStamp());

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

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

            System.out.println(message);
            db.insert(message);


 }
}
```