Mobile and the real-time cloud

Overview

This is the fourth lab exercise in this course. After completing this lab you will be able to use MQTT to support the communication functionality in your mobile app. MQTT is currently one of the most important publish/subscribe protocols for interconnected computing, especially in mobile and IoT. MQTT is lightweight and efficient, designed for constrained computing, but has a good selection of features to satisfy most applications and still be a magnitude simpler than several of the alternatives like for example AMQP.

Links, starting points:

Rather than starting from scratch, you can use the following sample project as a starting base:

https://github.com/gokr/mqtt-painter (Länkar till en externa sida.)Länkar till en externa sida.

Just clone it, then drag and drop the evothings.json file onto Evothings Studio.

For concise info on MQTT, see:

https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot (Länkar till en externa sida.)Länkar till en externa sida.

A good test client for investigating what goes in in your MQTT communication:

http://www.hivemq.com/demos/websocket-client/ (Länkar till en externa sida.)Länkar till en externa sida.

Just enter the proper host name for the MQTT server, in this case mqtt.evothings.com, change port to **8083**, (and skip the SSL checkbox), and press CONNECT.

Exercise 1 (6 points)

Modify the painter app to become a single chat room. The communication principles would be the same, but instead of everyone seeing each other's painting strokes - you would instead see everyone's chat messages.

The default topic is paint/ and you probably want to call **your topic** something else, pick something unique for your lab group or else you will have conflicts with other students.

You should be able to set your nickname in the app. Focus on a clear, simple interface for this lab, striving to make it appealing to the user.

NOTE: If you're not running off of CG Tek viewer, or the Evothings viewer on mobile, **you need to edit the example a bit more**; typically the command "cordova.uuid" won't be available, and you need to create your own unique ID for each client some other way.

Note the try-catch construct, and use it to log errors!

```
app.onMessageArrived = function (message) {
try {
var o = JSON.parse(message.payloadString)
app.ctx.beginPath()
app.ctx.moveTo(o.from.x, o.from.y)
app.ctx.lineTo(o.to.x, o.to.y)
app.ctx.strokeStyle = o.color
app.ctx.stroke()
} catch (e) {
console.log('Bad message: ' + message.payloadString)
}
}
```

Exercise 2 (2 points)

Extend the chat application with at least one of the following features:

- Ability to send a chat message only seen by one or a few specific participants. Perhaps simply by prefixing with the nickname(s) of the users targeted.

- Ability to set a “goodbye!” message in your client that will be shown as a regular chat message if your client goes offline (hint: “Last Will”), something along these lines:
- ```
  var last_will = new Paho.MQTT.Message(JSON.stringify({
  msg: "good bye", uuid: app.uuid, nickname: app.nickname,
  color: app.color }));
    msg.destinationName = YOUR-TOPIC-HERE  // e.g.
  'music/' + app.uuid + '/evt'
    var options = {
      willMessage: msg,
      useSSL: true,
      onSuccess: app.onConnect,
      onFailure: app.onConnectFailure
    }
  ```
- Ability to, when connecting, immediately see the list of online nicknames in the chat room. (hint: “retained = true”)

Good luck!
  ```