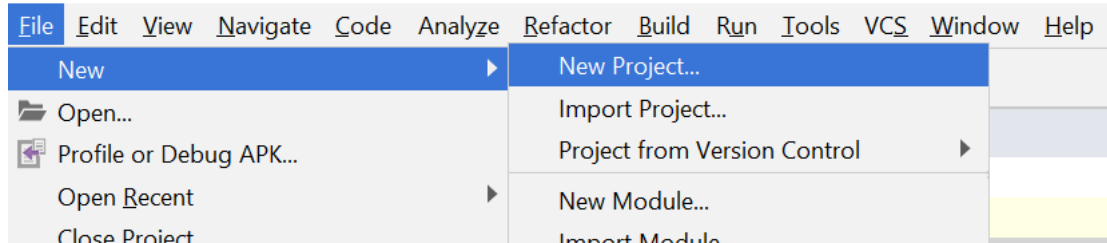
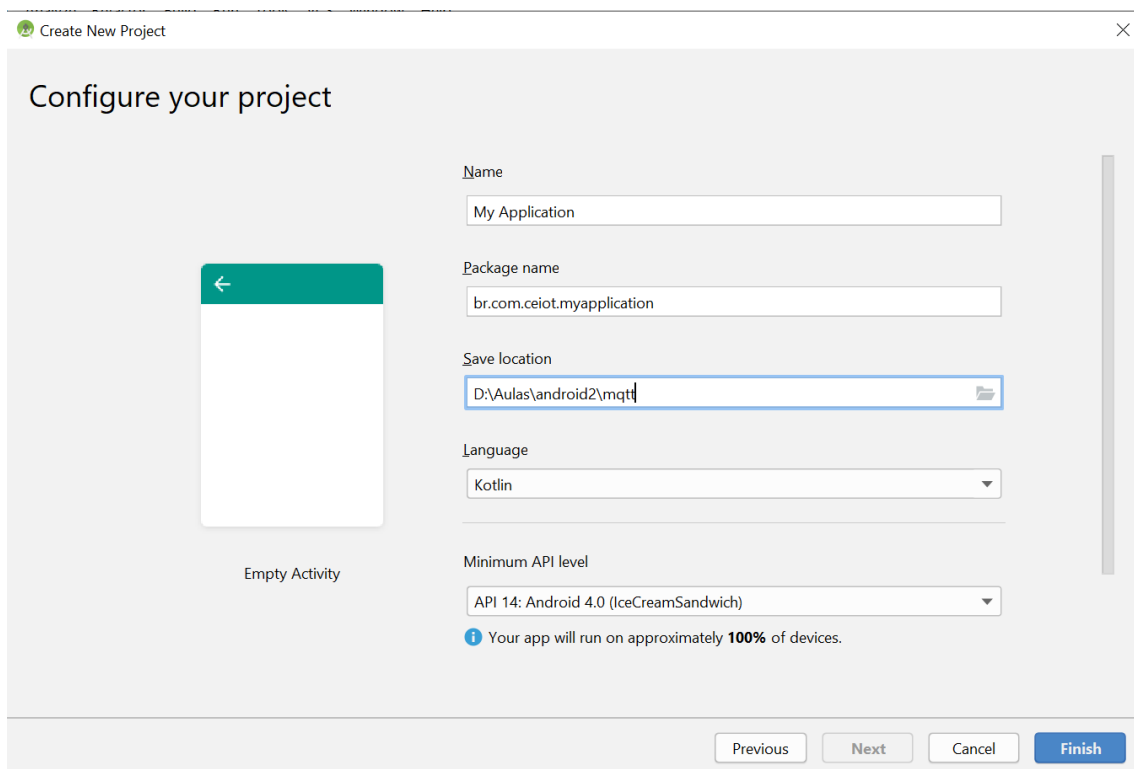


Tutorial MQTT Kotlin

1. Criar um projeto no Android Studio:



2. Selecionar Empty Activity



3. Em build gradle (Project: mqtt), adicionar o repositório:

```
1 maven {  
2     url "https://repo.eclipse.org/content/repositories/paho-snapshots/"  
3 }
```

4. Em build gradle (Project: app), adicionar as dependências:

```
1 implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'  
2 implementation 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'
```

5. Registrar o serviço MQTT:

```
1 <service android:name="org.eclipse.paho.android.service.MqttService"/>
```

6. Registrar permissões:

```
1 <uses-permission android:name="android.permission.WAKE_LOCK" />
2 <uses-permission android:name="android.permission.INTERNET" />
3 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
4 <uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

7. Criar classe ActionListener que implementa IMqttActionListener (Lib MQTT). Esta classe é utilizada para tratar mensagens de sucesso ou falha.

```
1 class ActionListener(private val name: String) : IMqttActionListener {
2
3     override fun onSuccess(asyncActionToken: IMqttToken) {
4         Log.d(TAG, "onSuccess: $name")
5     }
6
7     override fun onFailure(asyncActionToken: IMqttToken, exception: Throwable) {
8         Log.d(TAG, "onFailure: $name")
9     }
10
11     companion object {
12
13         private val TAG = "ActionListener"
14     }
15 }
```

8. Criar classe MqttCallbackHandler que implementa MqttCallbackExtended (Lib MQTT). Esta classe é utilizada para tratar mensagens de conexão, conexão perdida, Mensagem enviada/recebida etc.

```
1 open class MqttCallbackHandler(private val context: Context, private val
2 clientHandle: String) : MqttCallbackExtended {
3
4     override fun connectComplete(reconnect: Boolean, serverURI: String) {
5         Log.d(TAG, "connectComplete: $clientHandle")
6     }
7
8     override fun connectionLost(cause: Throwable) {
9         Log.d(TAG, "connectionLost: $clientHandle")
10    }
11
12    @Throws(Exception::class)
13    override fun messageArrived(topic: String, message: MqttMessage) {
14        Log.d(TAG, "messageArrived: $clientHandle")
15    }
16
17
18    override fun deliveryComplete(token: IMqttDeliveryToken) {
19        Log.d(TAG, "deliveryComplete: $clientHandle")
20    }
21
22    companion object {
23        private val TAG = "MqttCallbackHandler"
24    }
25
26 }
```

9. Criar classe AndroidMqttClient. Esta classe implementa as rotinas para conexão, envio de mensagens, etc.

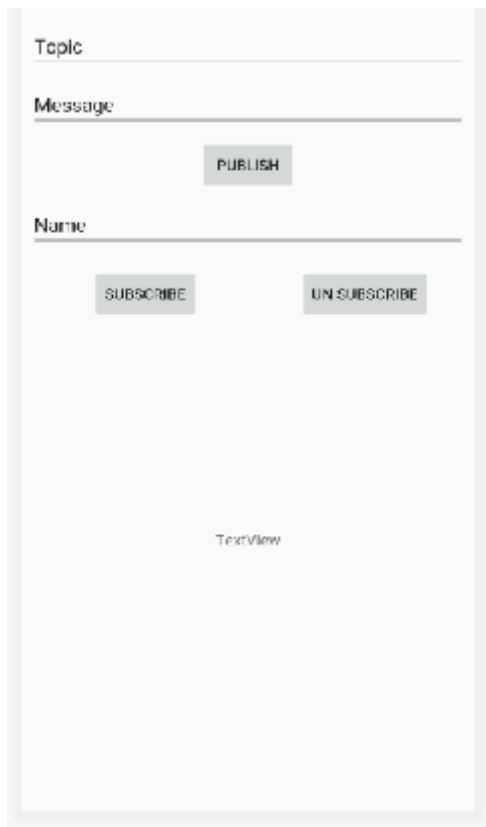
```
1  class AndroidMqttClient {
2
3      private var mqttClient: MqttAndroidClient? = null
4      private var brokerURL: String? = null
5      private var brokerPort: String? = null
6      internal var context: Context
7
8      /**
9       * Construtora que inicia serviço para um determinado broker.
10      * @param context
11      * @param brokerURL
12      * @param brokerPort
13      */
14      internal constructor(context: Context, brokerURL: String, brokerPort: String)
15      {
16          this.context = context
17          this.brokerURL = brokerURL
18          this.brokerPort = brokerPort
19          createMqttClient(MqttCallbackHandler(this.context.applicationContext,
20 "AppCEIoT Callback"))
21      }
22
23      internal constructor(context: Context, brokerURL: String, brokerPort: String,
24 mqttCallback: MqttCallback) {
25          this.context = context
26          this.brokerURL = brokerURL
27          this.brokerPort = brokerPort
28          createMqttClient(mqttCallback)
29      }
30
31      /**
32      * Inicializa client.
33      * @return
34      */
35      fun createMqttClient(mqttCallback: MqttCallback): MqttAndroidClient {
36          val clientId = MqttClient.generateClientId()
37          this.mqttClient = MqttAndroidClient(
38              this.context.applicationContext,
39              "tcp://" + this.brokerURL + ":" + this.brokerPort,
40              clientId
41          )
42          this.mqttClient!!.setCallback(mqttCallback)
43          return this.mqttClient!!
44      }
45
46      /**
47      * Realiza conexão
48      * @return
49      * @throws MqttException
50      */
51      @Throws(MqttException::class)
52      fun connect(): IMqttToken {
53          val options = MqttConnectOptions()
54          options.mqttVersion = MqttConnectOptions.MQTT_VERSION_3_1
55          options.isAutomaticReconnect = true
56          val token = mqttClient!!.connect(options)
57          token.actionCallback = ActionListener("MqttConnect")
58          return token
59      }
60
61      /**
62      * Implementa desconexão
63      * @throws MqttException
64      */
65      @Throws(MqttException::class)
66      fun disconnect() {
67          val mqttToken = mqttClient!!.disconnect()
68          mqttToken.actionCallback = ActionListener("MqttDisconnect")
69      }
70
71      /**
72      * Publica uma mensagem no broker
73      * @param message
```

```

74      * @param qos
75      * @param topic
76      * @throws MqttException
77      * @throws UnsupportedEncodingException
78      */
79      @Throws(MqttException::class, UnsupportedEncodingException::class)
80      fun publishMessage(message: String, qos: Int, topic: String) {
81          var encodedPayload = ByteArray(0)
82          encodedPayload = message.toByteArray(charset("UTF-8"))
83          val encodedMessage = MqttMessage(encodedPayload)
84          mqttClient!!.publish(topic, encodedMessage)
85      }
86
87      /**
88       * Se inscreve para escutar um determinado tópico
89       * @param topic
90       * @param qos
91       * @throws MqttException
92       */
93      @Throws(MqttException::class)
94      fun subscribe(topic: String, qos: Int) {
95          val token = mqttClient!!.subscribe(topic, qos)
96          token.actionCallback = ActionListener("MqttSubscribe")
97      }
98
99      /**
100       * Cancela inscrição em um determinado tópico
101       * @param topic
102       * @throws MqttException
103       */
104      @Throws(MqttException::class)
105      fun unsubscribe(topic: String) {
106          val token = mqttClient!!.unsubscribe(topic)
107          token.actionCallback = ActionListener("MqttUnSubscribe")
108      }
109
110      companion object {
111
112          private val TAG = "AndroidMqttClient"
113      }
114
115  }

```

10. Em activity_main.xml, criar o seguinte layout. Com o código abaixo:



```

1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:tools="http://schemas.android.com/tools"
5      xmlns:app="http://schemas.android.com/apk/res-auto"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9
10     <Button
11         android:id="@+id/button_publish"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:layout_marginEnd="8dp"
15         android:layout_marginLeft="8dp"
16         android:layout_marginRight="8dp"
17         android:layout_marginStart="8dp"
18         android:layout_marginTop="8dp"
19         android:onClick="publish"
20         android:text="Publish"
21         app:layout_constraintEnd_toEndOf="parent"
22         app:layout_constraintStart_toStartOf="parent"
23         app:layout_constraintTop_toBottomOf="@+id/editText_publish_msg" />
24
25     <Button
26         android:id="@+id/button_subscribe"
27         android:layout_width="wrap_content"
28         android:layout_height="wrap_content"
29         android:layout_marginEnd="8dp"
30         android:layout_marginLeft="8dp"
31         android:layout_marginRight="8dp"
32         android:layout_marginStart="8dp"
33         android:layout_marginTop="16dp"
34         android:onClick="subscribe"
35         android:text="Subscribe"
36         app:layout_constraintEnd_toStartOf="@+id/button_unsubscribe"
37         app:layout_constraintHorizontal_bias="0.405"
38         app:layout_constraintStart_toStartOf="parent"
39         app:layout_constraintTop_toBottomOf="@+id/editText_subscribe_topic"
40     />
41
42     <EditText

```

```

43         android:id="@+id/editText_publish_topic"
44         android:layout_width="0dp"
45         android:layout_height="wrap_content"
46         android:layout_marginEnd="8dp"
47         android:layout_marginLeft="8dp"
48         android:layout_marginRight="8dp"
49         android:layout_marginStart="8dp"
50         android:layout_marginTop="16dp"
51         android:ems="10"
52         android:inputType="textPersonName"
53         android:text="Topic"
54         app:layout_constraintEnd_toEndOf="parent"
55         app:layout_constraintStart_toStartOf="parent"
56         app:layout_constraintTop_toTopOf="parent" />
57
58     <EditText
59         android:id="@+id/editText_publish_msg"
60         android:layout_width="0dp"
61         android:layout_height="wrap_content"
62         android:layout_marginEnd="8dp"
63         android:layout_marginLeft="8dp"
64         android:layout_marginRight="8dp"
65         android:layout_marginStart="8dp"
66         android:layout_marginTop="8dp"
67         android:ems="10"
68         android:inputType="textPersonName"
69         android:text="Message"
70         app:layout_constraintEnd_toEndOf="parent"
71         app:layout_constraintStart_toStartOf="parent"
72         app:layout_constraintTop_toBottomOf="@+id/editText_publish_topic" />
73
74     <EditText
75         android:id="@+id/editText_subscribe_topic"
76         android:layout_width="0dp"
77         android:layout_height="wrap_content"
78         android:layout_marginEnd="8dp"
79         android:layout_marginLeft="8dp"
80         android:layout_marginRight="8dp"
81         android:layout_marginStart="8dp"
82         android:layout_marginTop="8dp"
83         android:ems="10"
84         android:inputType="textPersonName"
85         android:text="Name"
86         app:layout_constraintEnd_toEndOf="parent"
87         app:layout_constraintStart_toStartOf="parent"
88         app:layout_constraintTop_toBottomOf="@+id/button_publish" />
89
90     <Button
91         android:id="@+id/button_unsubscribe"
92         android:layout_width="wrap_content"
93         android:layout_height="wrap_content"
94         android:layout_marginEnd="40dp"
95         android:layout_marginRight="40dp"
96         android:layout_marginTop="16dp"
97         android:onClick="unsubscribe"
98         android:text="Un Subscribe"
99         app:layout_constraintEnd_toEndOf="parent"
100        app:layout_constraintTop_toBottomOf="@+id/editText_subscribe_topic"
101    />
102
103    <TextView
104        android:id="@+id/textView_result"
105        android:layout_width="wrap_content"
106        android:layout_height="wrap_content"
107        android:layout_marginBottom="8dp"
108        android:layout_marginEnd="8dp"
109        android:layout_marginLeft="8dp"
110        android:layout_marginRight="8dp"
111        android:layout_marginStart="8dp"
112        android:layout_marginTop="8dp"
113        android:text="TextView"
114        app:layout_constraintBottom_toBottomOf="parent"
115        app:layout_constraintEnd_toEndOf="parent"
116        app:layout_constraintStart_toStartOf="parent"
117        app:layout_constraintTop_toTopOf="@+id/button_subscribe" />
118
119 </android.support.constraint.ConstraintLayout>

```

11. Na MainActivity, inicializar o MQTT:

a. Adicionar o código abaixo no método onCreate

```
1  try {
2      this.mqttClient = AndroidMqttClient(
3          this,
4          "iot.eclipse.org",
5          "1883",
6          MqttCallbackActivity(this, "MainActivityMqttCallback")
7      )
8      val token = this.mqttClient!!.connect()
9  }
10 } catch (e: MqttException) {
11     e.printStackTrace()
12 }
```

b. Implementar os métodos que realizam publish, subscribe, etc...

```
1  fun publish(view: View) {
2      val topic = editText_publish_topic.getText().toString()
3      val message = editText_publish_msg.getText().toString()
4      try {
5          mqttClient!!.publishMessage(message, 0, topic)
6      } catch (e: UnsupportedEncodingException) {
7          e.printStackTrace()
8      } catch (e: MqttException) {
9          e.printStackTrace()
10     }
11 }
12
13
14 fun subscribe(view: View) {
15     val topic = editText_subscribe_topic.getText().toString()
16     try {
17         mqttClient!!.subscribe(topic, 0)
18     } catch (e: MqttException) {
19         e.printStackTrace()
20     }
21 }
22
23
24 fun unsubscribe(view: View) {
25     val topic = editText_subscribe_topic.getText().toString()
26     try {
27         mqttClient!!.unsubscribe(topic)
28     } catch (e: MqttException) {
29         e.printStackTrace()
30     }
31 }
32
33 inner class MqttCallbackActivity(context: Context, clientHandle: String) :
34     MqttCallbackHandler(context, clientHandle) {
35     @Throws(Exception::class)
36     override fun messageArrived(topic: String, message: MqttMessage) {
37         super.messageArrived(topic, message)
38         textView_result.setText("$topic:$message")
39     }
40 }
```

Tutorial MQTT Ionic

- 1) Instalar o Nodejs.
- 2) Instalar o ionic com o comando “npm install -g ionic cordova”
- 3) Para criar uma aplicação, utilize o comando “ionic start ceiotMqtt blank”
- 4) Abrir a pasta gerada com o Visual Studio Code.
- 5) Baixe o arquivo mqtt.min.js e copie para a pasta src/js
 - a. <https://unpkg.com/mqtt@2.18.8/dist/mqtt.min.js>
 - b. <https://github.com/mqttjs/MQTT.js#browser>
 - c. Alternativa: <https://www.eclipse.org/paho/clients/js/>

- 6) Em src/index.html, adicione a linha abaixo:

```
1 <script src="js/mqtt.min.js"></script>
```

- 7) Em src/app/home, abrir o arquivo [home.page.ts](#)

- 8) Adicionar o código abaixo e seus respectivos imports:

```
1 import { Router } from '@angular/router';
2 import { HttpClient } from '@angular/common/http';
3 import * as mqtt from '../js/mqtt.min';
```

```
1 constructor(
2   private router: Router,
3   public httpClient: HttpClient) {
4 }
```

- a. “private router: Router” servirá para fazer o redirecionamento de páginas.

- i. Se necessário usar: `this.router.navigate(['/nome_da_pagina']);`

- b. “public httpClient: HttpClient,” servirá para fazer chamadas REST.

- i. Em src/app, abrir o arquivos [app.modules.ts](#) e substituir a linha imports para adicionar HttpClientModule:

```
1 imports: [BrowserModule, IonicModule.forRoot(),
  AppRoutingModule,HttpClientModule],
```

- 9) Declarar a variável messageList para receber as mensagens MQTT

```
1 messageList: any[] = [];
```


10) Utilizar o código abaixo para configuração do MQTT:

```
1  ngOnInit() {
2      this.mqttConnect();
3  }
4
5  /*
6  Configuração para MQTT
7  */
8  mqttConnect() {
9      try {
10
11          let that = this; //Referência para chamar variáveis do angular
12
13          //Configuração do Broker. (Websockets)
14          var options = {
15              clientId: 'testCeiote_1',
16              connectTimeout: 5000,
17              hostname: 'test.mosquitto.org',
18              port: 8080,
19              path: '/mqtt'
20          };
21
22          //Conexão
23          var client = mqtt.connect(options);
24
25          //Se inscreve em um tópico ao se conectar ao broker
26          client.on('connect', function () {
27              client.subscribe('ceiot', function (err) {
28                  if (!err) {
29                      client.publish('ceiot', 'Hello mqtt')
30                  }
31              })
32          });
33
34          //Tratamento ao receber mensagem
35          client.on('message', function (topic, message) {
36              that.receiveMessage(topic, message);
37          });
38      } catch (e) {
39          console.log(e);
40      }
41  }
42
43  /**
44   * Adiciona tópico e mensagem no array messageList
45   * @param topic
46   * @param message
47   */
48  receiveMessage(topic, message)
49  {
50      console.log(message.toString())
51      var obj = {};
52      obj['topic'] = topic;
53      obj['message'] = message;
54      this.messageList.push(obj);
55  }
```

11) Atualizar o código HTML em src/app/home home.page

```
1  <ion-header>
2      <ion-toolbar>
4          <ion-title>
5              Exemplo MQTT
6          </ion-title>
7      </ion-toolbar>
8  </ion-header>
9
10 <ion-content>
11     <div class="ion-padding">
12         Lista de mensagens
13         <ion-grid>
14             <ion-row>
15                 <ion-col><div>Tópico</div></ion-col>
16                 <ion-col><div>Mensagem</div></ion-col>
```

```
17         </ion-row>
18         <ion-row *ngFor="let item of messageList">
19             <ion-col>{{item.topic}}</ion-col>
20             <ion-col>{{item.message}}</ion-col>
21         </ion-row>
22     </ion-grid>
23 </div>
24 </ion-content>
```

Tutorial MQTT Flutter

- 1) Criar uma aplicação conforme, o link abaixo
 - a. <https://flutter.dev/docs/get-started/test-drive?tab=vscode#create-app>
- 2) Substituir o conteúdo do arquivo main.dart pelo código abaixo.

```
1 import 'package:flutter/material.dart';
2
3 void main() => runApp(MyApp());
4
5 class MyApp extends StatelessWidget {
6   @override
7   Widget build(BuildContext context) {
8     return MaterialApp(
9       title: 'Flutter MQTT',
10      theme: ThemeData(
11        primarySwatch: Colors.blue,
12      ),
13      home: Scaffold(
14        appBar: AppBar(
15          title: Text('Exemplo Flutter MQTT'),
16        ),
17        body: Center(
18          child: Text('Flutter MQTT'),
19        ),
20      ),
21    );
22  }
23 }
```

- 3) Adicionar uma página com código abaixo. Em seguida, mudar a referência de “child: Text('Flutter MQTT')” para “child: MqttPage(),”

```
1 class MqttPage extends StatefulWidget {
2   @override
3   MqttState createState() => MqttState();
4 }
5
6
7 class MqttState extends State<MqttPage> {
8   @override
9   Widget build(BuildContext context) {
10    return Text("Flutter MQTT");
11  }
12 }
```

- 4) Adiciona código para exibir uma lista em tela:

```
1 class MqttPage extends StatefulWidget {
2   @override
3   MqttState createState() => MqttState();
4 }
5
6 class MqttMessage {
7   const MqttMessage({ this.topic, this.message });
8   final String topic;
9   final String message;
10 }
11
12 class MqttState extends State<MqttPage> {
13
14   List<MqttMessage> messages = <MqttMessage>[
15     new MqttMessage(topic: "AA",message: "AA"),
```

```

16     new MqttMessage(topic: "BB",message: "BB"),
17     new MqttMessage(topic: "CC",message: "CC")];
18
19   Widget buildList() {
20     return
21     ListView.builder(
22       itemCount: messages.length,
23       itemBuilder: (context, position) {
24         return _buildRow(messages[position]);
25       });
26   }
27
28   @override
29   Widget build(BuildContext context) {
30     return buildList();
31   }
32
33   Widget _buildRow(MqttMessage message) {
34     return ListTile(
35       title: Text(message.message)
36     );
37   }

```

- 5) Para iniciar a implementação do MQTT deve-se adicionar a dependência em pubspec.yaml
- ```
mqtt_client: ^5.5.3
```

- 6) Adicionar imports:

- a. Se continuar com erro, reiniciar a IDE.

```

1 import 'dart:async';
2 import 'package:flutter/material.dart';
3 import 'package:mqtt_client/mqtt_client.dart' as mqtt;

```

- 7) Declarar o seguinte código.

```

1 class MqttState extends State<MqttPage> {
2
3 String broker = 'test.mosquitto.org';
4 mqtt.MqttClient client;
5 mqtt.MqttConnectionState connectionState;
6 StreamSubscription subscription;
7 Set<String> topics = Set<String>();
8 String topic = "ceiot";
9 ...
10

```

- 8) Chamar o método connect ao iniciar.

```

1 ...
2 @override
3 Widget build(BuildContext context) {
4 if (client?.connectionState == mqtt.MqttConnectionState.connected) {
5 //_disconnect();
6 } else {
7 _connect();
8 }
9 return buildList();
10 }
11 ...
12

```

- 9) Implementação MQTT

- a. Baseado no exemplo da página:

- i. [https://github.com/shamblott/mqtt\\_client/blob/master/example/flutter/lib/main.dart](https://github.com/shamblott/mqtt_client/blob/master/example/flutter/lib/main.dart)
- b. Alterei apenas função `onMessage`.

```
1 ...
2 void _connect() async {
3
4 /// First create a client
5 client = mqtt.MqttClient(broker, '');
6
7 /// Set logging on if needed, defaults to off
8 client.logging(on: true);
9
10 /// Keep alive value
11 client.keepAlivePeriod = 30;
12
13 /// Add the unsolicited disconnection callback
14 client.onDisconnected = _onDisconnected;
15
16 /// Create a connection message to use or use the default one.
17 final mqtt.MqttConnectMessage connMess = mqtt.MqttConnectMessage()
18 .withClientIdentifier('Mqtt_MyClientUniqueId2')
19 // Must agree with the keep alive set above or not set
20 .startClean() // Non persistent session for testing
21 .keepAliveFor(30)
22 // If you set this you must set a will message
23 .withWillTopic('willtopic')
24 .withWillMessage('My Will message')
25 .withWillQos(mqtt.MqttQos.atLeastOnce);
26 print('MQTT client connecting....');
27 client.connectionMessage = connMess;
28
29 /// Connect the client
30 try {
31 await client.connect();
32 } catch (e) {
33 print(e);
34 _disconnect();
35 }
36
37 /// Check if we are connected
38 if (client != null && client.connectionState ==
39 mqtt.MqttConnectionState.connected) {
40 print('MQTT client connected');
41 setState(() {
42 connectionState = client.connectionState;
43 });
44 _subscribeToTopic(topic);
45 } else {
46 print('ERROR: MQTT client connection failed - '
47 'disconnecting, state is ${client.connectionState}');
48 _disconnect();
49 }
50
51 /// Message Listener
52 subscription = client.updates.listen(_onMessage);
53 }
54
55 void _disconnect() {
56 client.disconnect();
57 _onDisconnected();
58 }
59
60 void _onDisconnected() {
61 setState(() {
62 topics.clear();
63 connectionState = client.connectionState;
64 client = null;
65 subscription.cancel();
66 subscription = null;
67 });
68 print('MQTT client disconnected');
69 }
70 }
```

```

71 void _onMessage(List<mqtt.MqttReceivedMessage> event) {
72 print(event.length);
73 final mqtt.MqttPublishMessage recMess =
74 event[0].payload as mqtt.MqttPublishMessage;
75 final String message =
76 mqtt.MqttPublishPayload.bytesToStringAsString(recMess.payload.message);
77
78 print('MQTT message: topic is <${event[0].topic}>, '
79 'payload is <-- ${message} -->');
80 print(client.connectionState);
81 setState(() {
82 messages.add(MqttMessage(
83 topic: event[0].topic,
84 message: message
85));
86 });
87 }
88
89 void _subscribeToTopic(String topic) {
90 if (connectionState == mqtt.MqttConnectionState.connected) {
91 setState(() {
92 if (topics.add(topic.trim())) {
93 print('Subscribing to ${topic.trim()}');
94 client.subscribe(topic, mqtt.MqttQos.exactlyOnce);
95 }
96 });
97 }
98 }
99
100 void _unsubscribeFromTopic(String topic) {
101 if (connectionState == mqtt.MqttConnectionState.connected) {
102 setState(() {
103 if (topics.remove(topic.trim())) {
104 print('Unsubscribing from ${topic.trim()}');
105 client.unsubscribe(topic);
106 }
107 });
108 }
109 }
110 }

```