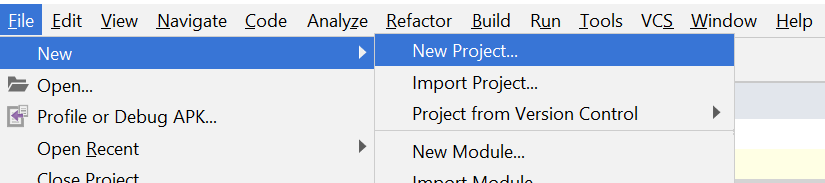
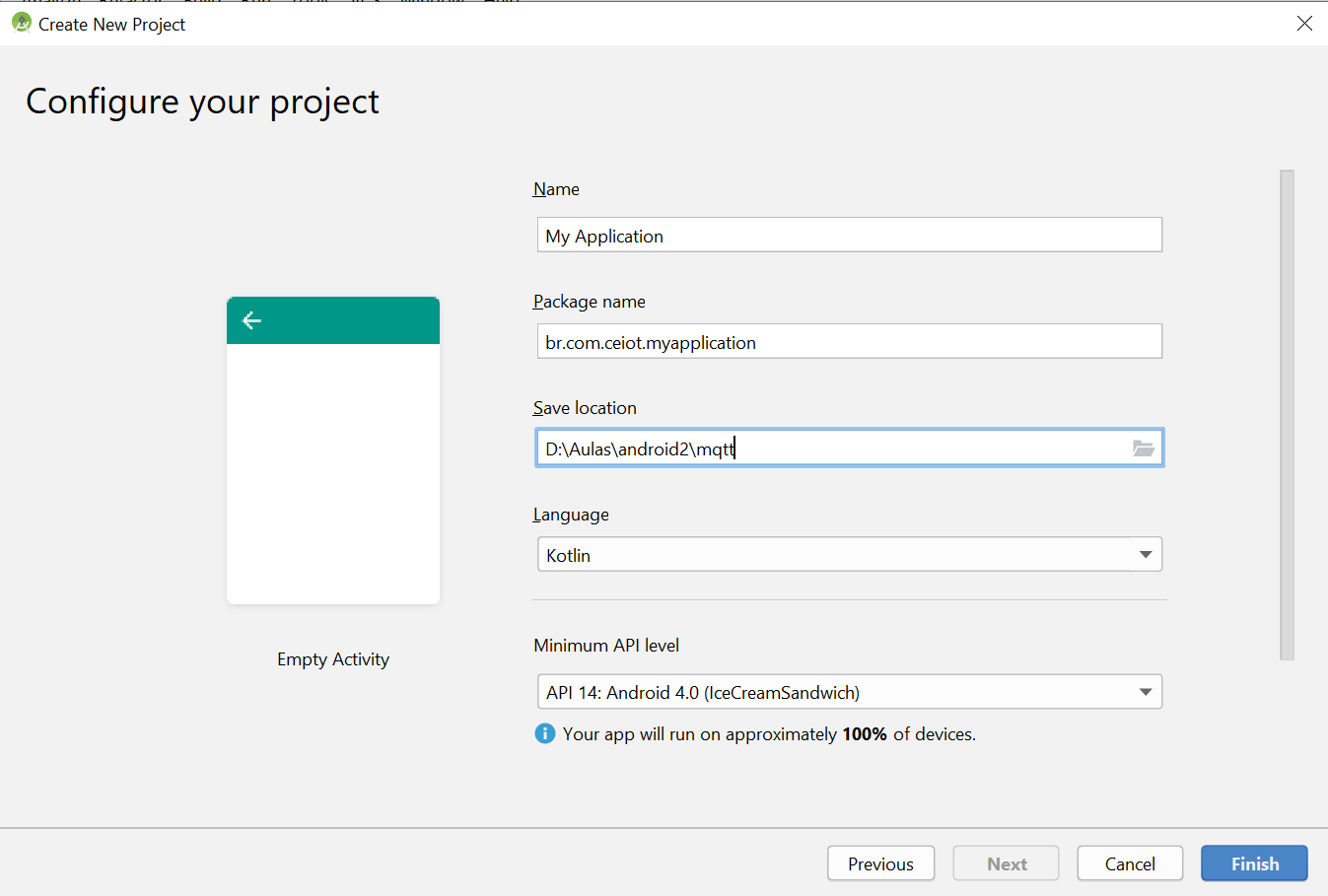
**Tutorial MQTT Kotlin**

1. Criar um projeto no Android Studio:



1. Selecionar Empty Activity



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

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

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

|  |  |
| --- | --- |
| 1  2 | implementation **'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'** implementation **'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'** |

1. Registrar o serviço MQTT:

|  |  |
| --- | --- |
| 1 | <**service android:name="org.eclipse.paho.android.service.MqttService"**/> |

1. Registrar permissões:

|  |  |
| --- | --- |
| 1  2  3  4 | <**uses-permission android:name="android.permission.WAKE\_LOCK"** /> <**uses-permission android:name="android.permission.INTERNET"** /> <**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"** /> <**uses-permission android:name="android.permission.READ\_PHONE\_STATE"** /> |

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

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

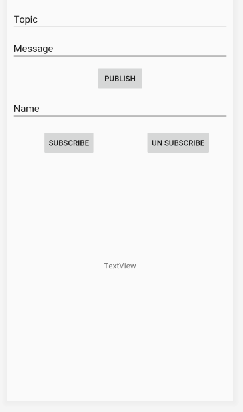
1. Criar classe MqttCallbackHandler que implementa MqttCallbackExtended (Lib MQTT). Esta classe é utilizada para tratar mensagens de conexão, conexão perdida, Mensagem enviada/recebida etc.

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

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115 | **class** AndroidMqttClient {   **private var mqttClient**: MqttAndroidClient? = **null  private var brokerURL**: String? = **null  private var brokerPort**: String? = **null  internal var context**: Context   */\*\*  \* Construtora que inicia serviço para um determinado broker.  \** ***@param context*** *\** ***@param brokerURL*** *\** ***@param brokerPort*** *\*/* **internal constructor**(context: Context, brokerURL: String, brokerPort: String) {  **this**.**context** = context  **this**.**brokerURL** = brokerURL  **this**.**brokerPort** = brokerPort  createMqttClient(MqttCallbackHandler(**this**.**context**.*applicationContext*, **"AppCEIOT Callback"**))  }   **internal constructor**(context: Context, brokerURL: String, brokerPort: String, mqttCallback: MqttCallback) {  **this**.**context** = context  **this**.**brokerURL** = brokerURL  **this**.**brokerPort** = brokerPort  createMqttClient(mqttCallback)  }   */\*\*  \* Inicializa client.  \** ***@return*** *\*/* **fun** createMqttClient(mqttCallback: MqttCallback): MqttAndroidClient {  **val** clientId = MqttClient.generateClientId()  **this**.**mqttClient** = MqttAndroidClient(  **this**.**context**.*applicationContext*,  **"tcp://"** + **this**.**brokerURL** + **":"** + **this**.**brokerPort**,  clientId  )  **this**.**mqttClient**!!.setCallback(mqttCallback)  **return this**.**mqttClient**!!  }   */\*\*  \* Realiza conexão  \** ***@return*** *\** ***@throws MqttException*** *\*/* @Throws(MqttException::**class**)  **fun** connect(): IMqttToken {  **val** options = MqttConnectOptions()  options.*mqttVersion* = MqttConnectOptions.*MQTT\_VERSION\_3\_1* options.*isAutomaticReconnect* = **true  val** token = **mqttClient**!!.connect(options)  token.*actionCallback* = ActionListener(**"MqttConnect"**)  **return** token  }   */\*\*  \* Implementa desconexão  \** ***@throws MqttException*** *\*/* @Throws(MqttException::**class**)  **fun** disconnect() {  **val** mqttToken = **mqttClient**!!.disconnect()  mqttToken.*actionCallback* = ActionListener(**"MqttDisconnect"**)  }   */\*\*  \* Publica uma mensagem no broker  \** ***@param message*** *\** ***@param qos*** *\** ***@param topic*** *\** ***@throws MqttException*** *\** ***@throws UnsupportedEncodingException*** *\*/* @Throws(MqttException::**class**, UnsupportedEncodingException::**class**)  **fun** publishMessage(message: String, qos: Int, topic: String) {  **var** encodedPayload = ByteArray(0)  encodedPayload = message.*toByteArray*(*charset*(**"UTF-8"**))  **val** encodedMessage = MqttMessage(encodedPayload)  **mqttClient**!!.publish(topic, encodedMessage)  }   */\*\*  \* Se inscreve para escutar um determinado tópico  \** ***@param topic*** *\** ***@param qos*** *\** ***@throws MqttException*** *\*/* @Throws(MqttException::**class**)  **fun** subscribe(topic: String, qos: Int) {  **val** token = **mqttClient**!!.subscribe(topic, qos)  token.*actionCallback* = ActionListener(**"MqttSubscribe"**)  }   */\*\*  \* Cancela inscrição em um determinado tópico  \** ***@param topic*** *\** ***@throws MqttException*** *\*/* @Throws(MqttException::**class**)  **fun** unSubscribe(topic: String) {  **val** token = **mqttClient**!!.unsubscribe(topic)  token.*actionCallback* = ActionListener(**"MqttUnSubscribe"**)  }   **companion object** {   **private val TAG** = **"AndroidMqttClient"** }  } |

1. Em activity\_main.xml, criar o seguinte layout. Com o código abaixo:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119 | *<?***xml version="1.0" encoding="utf-8"***?>* <**android.support.constraint.ConstraintLayout  xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  xmlns:app="http://schemas.android.com/apk/res-auto"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context=".MainActivity"**>   <**Button  android:id="@+id/button\_publish"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="8dp"  android:layout\_marginLeft="8dp"  android:layout\_marginRight="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  android:onClick="publish"  android:text="Publish"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toBottomOf="@+id/editText\_publish\_msg"** />   <**Button  android:id="@+id/button\_subscribe"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="8dp"  android:layout\_marginLeft="8dp"  android:layout\_marginRight="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="16dp"  android:onClick="subscribe"  android:text="Subscribe"  app:layout\_constraintEnd\_toStartOf="@+id/button\_unsubscribe"  app:layout\_constraintHorizontal\_bias="0.405"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toBottomOf="@+id/editText\_subscribe\_topic"** />   <**EditText  android:id="@+id/editText\_publish\_topic"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="8dp"  android:layout\_marginLeft="8dp"  android:layout\_marginRight="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="16dp"  android:ems="10"  android:inputType="textPersonName"  android:text="Topic"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent"** />   <**EditText  android:id="@+id/editText\_publish\_msg"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="8dp"  android:layout\_marginLeft="8dp"  android:layout\_marginRight="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  android:ems="10"  android:inputType="textPersonName"  android:text="Message"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toBottomOf="@+id/editText\_publish\_topic"** />   <**EditText  android:id="@+id/editText\_subscribe\_topic"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="8dp"  android:layout\_marginLeft="8dp"  android:layout\_marginRight="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  android:ems="10"  android:inputType="textPersonName"  android:text="Name"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toBottomOf="@+id/button\_publish"** />   <**Button  android:id="@+id/button\_unsubscribe"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="40dp"  android:layout\_marginRight="40dp"  android:layout\_marginTop="16dp"  android:onClick="unsubscribe"  android:text="Un Subscribe"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintTop\_toBottomOf="@+id/editText\_subscribe\_topic"** />   <**TextView  android:id="@+id/textView\_result"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginLeft="8dp"  android:layout\_marginRight="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  android:text="TextView"  app:layout\_constraintBottom\_toBottomOf="parent"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="@+id/button\_subscribe"** />  </**android.support.constraint.ConstraintLayout**> |

1. Na MainActivity, inicializar o MQTT:
   1. Adicionar o código abaixo no método onCreate

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

* 1. Implementar os métodos que realizam publish, subscribe, etc...

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

**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
   1. <https://unpkg.com/mqtt@2.18.8/dist/mqtt.min.js>
   2. <https://github.com/mqttjs/MQTT.js#browser>
   3. 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> |

1. Em src/app/home, abrir o arquivo [home.page.ts](http://home.page.ts)
2. Adicionar o código abaixo e seus respectivos imports:

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

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

* 1. “private router: Router” servirá para fazer o redirecionamento de páginas.
     1. Se necessário usar: this.router.navigate(['/nome\_da\_pagina]);
  2. “public httpClient: HttpClient,” servirá para fazer chamas rest.
     1. Em src/app, abrir o arquivos [app.modules.ts](http://home.page.ts) e substituir a linha imports para adicionar HttpModules:

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

1. Declarar a variável messageList para receber as mensagens MQTT

|  |  |
| --- | --- |
| 1 | messageList: any[] = []; |

1. Utilizar o código abaixo para configuração do MQTT:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | ngOnInit() {  this.mqttConnect();  }  /\*  Configuração para MQTT  \*/  mqttConnect() {  try {    let that = this; //Referência para chamar variáveis do angular  //Configuração do Broker. (Websockets)  var options = {  clientId: 'testCeiot\_1',  connectTimeout: 5000,  hostname: 'test.mosquitto.org',  port: 8080,  path: '/mqtt'  };    //Conexão  var client = mqtt.connect(options);    //Se inscreve em um tópico ao se conectar ao broker  client.on('connect', function () {  client.subscribe('ceiot', function (err) {  if (!err) {  client.publish('ceiot', 'Hello mqtt')  }  })  });  //Tratamento ao receber mensagem  client.on('message', function (topic, message) {  that.receiveMessage(topic, message);  });  } catch (e) {  console.log(e);  }  }  /\*\*  \* Adiciona tópico e mensagem no array messageList  \* @param topic  \* @param message  \*/  receiveMessage(topic, message)  {  console.log(message.toString())  var obj = {};  obj['topic'] = topic;  obj['message'] = message;  this.messageList.push(obj);  } |

1. Atualizar o código HTML em src/app/home home.page

|  |  |
| --- | --- |
| 1  2  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <ion-header>  <ion-toolbar>  <ion-title>  Exemplo MQTT  </ion-title>  </ion-toolbar>  </ion-header>  <ion-content>  <div class="ion-padding">  Lista de mensagens  <ion-grid>  <ion-row>  <ion-col><div>Tópico</div></ion-col>  <ion-col><div>Mensagem</div></ion-col>  </ion-row>  <ion-row \*ngFor="let item of messageList">  <ion-col>{{item.topic}}</ion-col>  <ion-col>{{item.message}}</ion-col>  </ion-row>  </ion-grid>  </div>  </ion-content> |

**Tutorial MQTT Flutter**

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

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

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

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

1. Adiciona código para exibir uma lista em tela:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | class MqttPage extends StatefulWidget {  @override  MqttState createState() => MqttState();  }  class MqttMessage {  const MqttMessage({ this.topic, this.message });  final String topic;  final String message;  }  class MqttState extends State<MqttPage> {  List<MqttMessage> messages = <MqttMessage>[  new MqttMessage(topic: "AA",message: "AA"),  new MqttMessage(topic: "BB",message: "BB"),  new MqttMessage(topic: "CC",message: "CC")];  Widget buildList() {  return  ListView.builder(  itemCount: messages.length,  itemBuilder: (context, position) {  return \_buildRow(messages[position]);  });  }    @override  Widget build(BuildContext context) {  return buildList();  }  Widget \_buildRow(MqttMessage message) {  return ListTile(  title: Text(message.message)  );  } |

1. Para iniciar a implementação do MQTT deve-se adicionar a dependência em pubspec.yaml mqtt\_client: ^5.5.3
2. Adicionar imports:
   1. Se continuar com erro, reiniciar a IDE.

|  |  |
| --- | --- |
| 1  2  3 | import 'dart:async';  import 'package:flutter/material.dart';  import 'package:mqtt\_client/mqtt\_client.dart' as mqtt; |

1. Declarar o seguinte código.

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

1. Chamar o método connect ao iniciar.

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

1. Implementação MQTT
   1. Baseado no exemplo da página:
      1. <https://github.com/shamblett/mqtt_client/blob/master/example/flutter/lib/main.dart>
   2. Alterei apenas função onMessage.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110 | ...  void \_connect() async {  /// First create a client  client = mqtt.MqttClient(broker, '');  /// Set logging on if needed, defaults to off  client.logging(on: true);  /// Keep alive value  client.keepAlivePeriod = 30;  /// Add the unsolicited disconnection callback  client.onDisconnected = \_onDisconnected;  /// Create a connection message to use or use the default one.  final mqtt.MqttConnectMessage connMess = mqtt.MqttConnectMessage()  .withClientIdentifier('Mqtt\_MyClientUniqueId2')  // Must agree with the keep alive set above or not set  .startClean() // Non persistent session for testing  .keepAliveFor(30)  // If you set this you must set a will message  .withWillTopic('willtopic')  .withWillMessage('My Will message')  .withWillQos(mqtt.MqttQos.atLeastOnce);  print('MQTT client connecting....');  client.connectionMessage = connMess;  /// Connect the client  try {  await client.connect();  } catch (e) {  print(e);  \_disconnect();  }  /// Check if we are connected  if (client != null && client.connectionState == mqtt.MqttConnectionState.connected) {  print('MQTT client connected');  setState(() {  connectionState = client.connectionState;  });  \_subscribeToTopic(topic);  } else {  print('ERROR: MQTT client connection failed - '  'disconnecting, state is ${client.connectionState}');  \_disconnect();  }  /// Message Listener  subscription = client.updates.listen(\_onMessage);  }  void \_disconnect() {  client.disconnect();  \_onDisconnected();  }  void \_onDisconnected() {  setState(() {  topics.clear();  connectionState = client.connectionState;  client = null;  subscription.cancel();  subscription = null;  });  print('MQTT client disconnected');  }  void \_onMessage(List<mqtt.MqttReceivedMessage> event) {  print(event.length);  final mqtt.MqttPublishMessage recMess =  event[0].payload as mqtt.MqttPublishMessage;  final String message =  mqtt.MqttPublishPayload.bytesToStringAsString(recMess.payload.message);  print('MQTT message: topic is <${event[0].topic}>, '  'payload is <-- ${message} -->');  print(client.connectionState);  setState(() {  messages.add(MqttMessage(  topic: event[0].topic,  message: message  ));  });  }  void \_subscribeToTopic(String topic) {  if (connectionState == mqtt.MqttConnectionState.connected) {  setState(() {  if (topics.add(topic.trim())) {  print('Subscribing to ${topic.trim()}');  client.subscribe(topic, mqtt.MqttQos.exactlyOnce);  }  });  }  }  void \_unsubscribeFromTopic(String topic) {  if (connectionState == mqtt.MqttConnectionState.connected) {  setState(() {  if (topics.remove(topic.trim())) {  print('Unsubscribing from ${topic.trim()}');  client.unsubscribe(topic);  }  });  }  }  } |