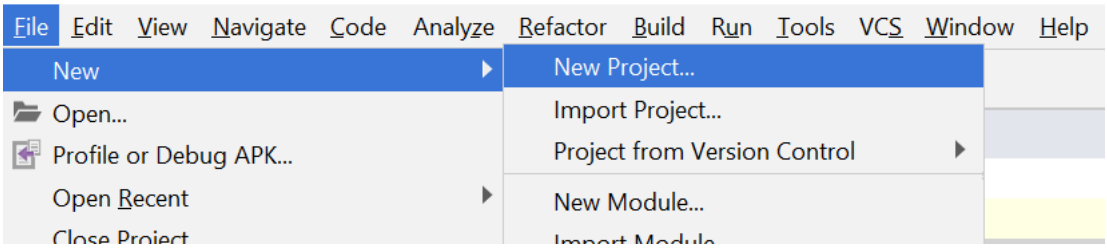
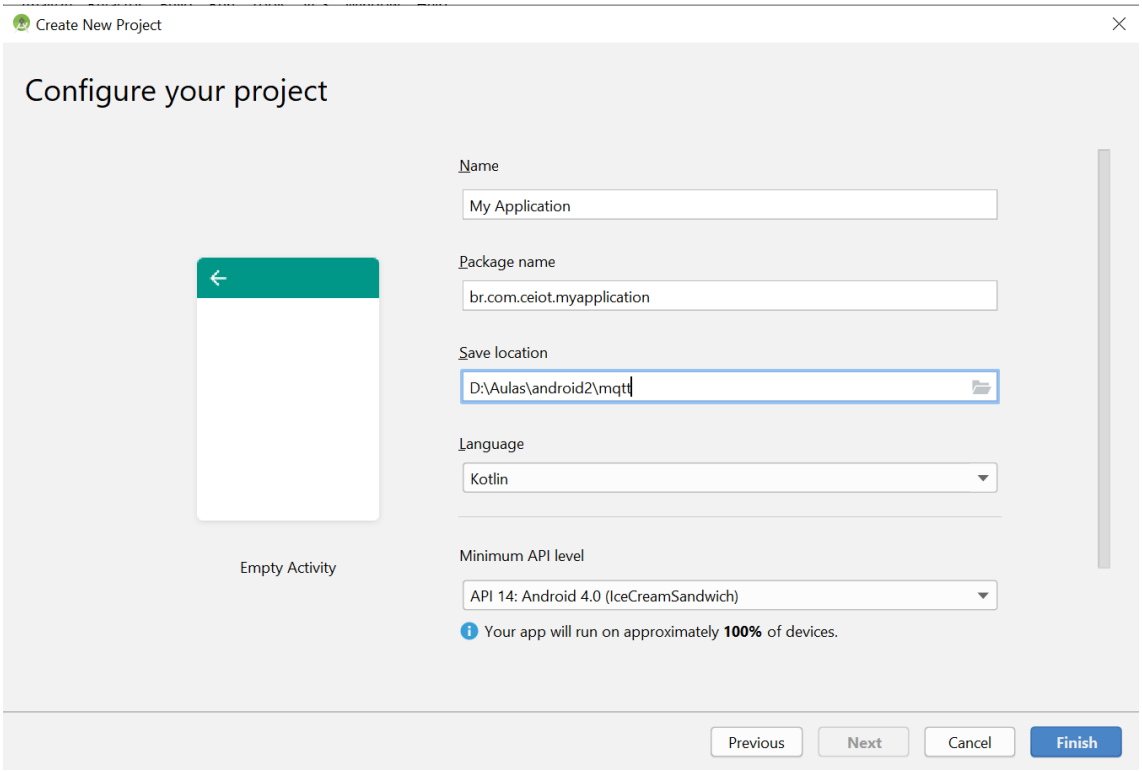
Tutorial MQTT Kotlin

1. Criar um projeto no Android Studio:



2. Selecionar Empty Activity



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

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

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

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

5. Registrar o serviço MQTT:

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

6. Registrar permissões:

```
<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" />
```

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

```
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"
    }
}
```

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

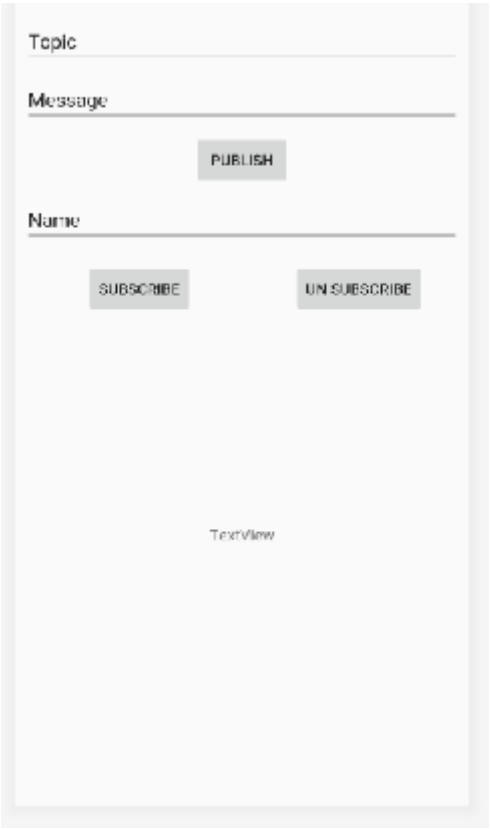
```
open class MqttCallbackHandler(private val context: Context, private val
clientHandle: String) : MqttCallbackExtended {

    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"
    }

}
```

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

```
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
```

```
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:




```
<?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>
```

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

```
try {
    this.mqttClient = AndroidMqttClient(
        this,
        "iot.eclipse.org",
        "1883",
        MqttCallBackActivity(this, "MainActivityMqttCallback")
    )
    val token = this.mqttClient!!.connect()

} catch (e: MqttException) {
    e.printStackTrace()
}
```

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

```
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
   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:

```
<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:

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

```
constructor(
    private router: Router,
    public httpClient: HttpClient) {
  }
```

   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 chamas rest.
      i. Em src/app, abrir o arquivos app.modules.ts e substituir a linha imports para adicionar HttpModules:

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

9) Declarar a variável messageList para receber as mensagens MQTT

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

10) Utilizar o código abaixo para configuração do MQTT:

```
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);
  }
```

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

```
<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>
```

```
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.

```dart
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'),
        ),
      ),
    );
  }
}
```

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

```dart
class MqttPage extends StatefulWidget {
  @override
  MqttState createState() => MqttState();
}

class MqttState extends State<MqttPage> {
  @override
  Widget build(BuildContext context) {
    return Text("Flutter MQTT");
  }
}
```

4) Adiciona código para exibir uma lista em tela:

```dart
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"),
```

```
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
4     String broker = 'test.mosquitto.org';
5     mqtt.MqttClient client;
6     mqtt.MqttConnectionState connectionState;
7     StreamSubscription subscription;
8     Set<String> topics = Set<String>();
9     String topic = "ceiot";
10  ...
```

8) Chamar o método connect ao iniciar.

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

9) Implementação MQTT
   a. Baseado no exemplo da página:

i. https://github.com/shamblett/mqtt_client/blob/master/example/flutter/lib/main.dart

b. Alterei apenas função `onMessage`.

```
...
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);
        }
      });
    }
  }
}
```