

# Manual de IoT Labs

Primera versión

Centro de  
Investigación de  
Ingeniería  
Mecatrónica  
(CIDIMEC)

Javier Contreras

Este manual busca servir de guía y referencia para el uso de la plataforma IoT Labs. En las subsiguientes secciones se mostraran casos de uso, ejemplos y descripciones de procedimientos para el uso correcto del sistema IoT Labs.

# Índice general

<b>1. Introducción</b>	<b>2</b>
<b>2. Interfaces de Laboratorio</b>	<b>3</b>
2.1. Herramientas de IoT Labs . . . . .	3
2.1.1. Editor IoT Labs IDE . . . . .	3
2.1.2. IoT Labs CLI . . . . .	4
2.2. Qt Markup Language (QML) . . . . .	6
2.2.1. Estructura de un archivo QML . . . . .	6
2.2.2. Layout . . . . .	7
2.2.3. Anchors . . . . .	9
2.2.4. Controles . . . . .	11
2.2.5. Property binding . . . . .	14
2.2.6. Elementos personalizados . . . . .	15
2.2.7. Cargar a la plataforma . . . . .	18
<b>3. Sistema de comunicación</b>	<b>25</b>
3.1. Obteniendo credenciales para el sistema de comunicación . . . . .	25
3.2. API de comunicación para las interfaces de laboratorio . . . . .	29
3.3. Cliente de pruebas . . . . .	31
<b>Referencias</b>	<b>35</b>

# Capítulo 1

## Introducción

El sistema IoT Labs surge como un proyecto de grado de Ingeniería Mecatrónica que busca proporcionar a los estudiantes una plataforma en la cual puedan estudiar el protocolo MQTT aplicado al Internet de las Cosas (IoT) [1].

Para la realización del proyecto se utilizaron diversas tecnologías, algunas de ellas se exponen al usuario de modo que sea utilizable por los estudiantes. Este es el caso de Qt, se trata de un framework sobre el cual se construye el sistema que permite a los usuarios desarrollar y desplegar páginas web sin necesidad de utilizar HTML, CSS ni JavaScript (web) [5].

Para fines de comunicación entre páginas y dispositivos IoT Labs dispone de un sistema de comunicación en tiempo real con los protocolos MQTT y WebSocket. Dado que WebSocket se distribuye de forma nativa con HTML5 fue utilizado para la construcción de una API a ser utilizada en las páginas web o "Interfaces de Laboratorio" de ahora en adelante. Dicha API cuenta con métodos similares a los encontrados en MQTT de modo que la comunicación resulte similar en ambos lados de cada laboratorio (Interfaz de laboratorio y dispositivo).

# Capítulo 2

## Interfaces de Laboratorio

Las interfaces de laboratorio son páginas web que el estudiante puede elaborar con el lenguaje QML, es decir que utilizan el framework de Qt para la plataforma WebAssembly. Sin embargo, su uso no requiere de la instalación de ninguna herramienta de Qt dado que la plataforma se encarga de compilar el programa. Para su elaboración se puede utilizar cualquier editor de texto. No obstante, se recomienda la utilización de las herramientas proporcionadas por IoT Labs puesto que el editor "IoT Labs IDE" tiene un visualizador y un entorno de ejecución que permitirá visualizar y probar la Interfaz de laboratorio antes de ser cargada a la plataforma. Dicha herramienta se encuentra disponible para su descarga en el apartado de descargas del sitio web [2].

### 2.1. Herramientas de IoT Labs

#### 2.1.1. Editor IoT Labs IDE

IoT Labs IDE es la herramienta de edición y pruebas desarrollada por IoT Labs para la elaboración de interfaces de laboratorio. Dispone de las funciones básicas de un editor de texto y un entorno de visualización y ejecución de las interfaces de laboratorio.

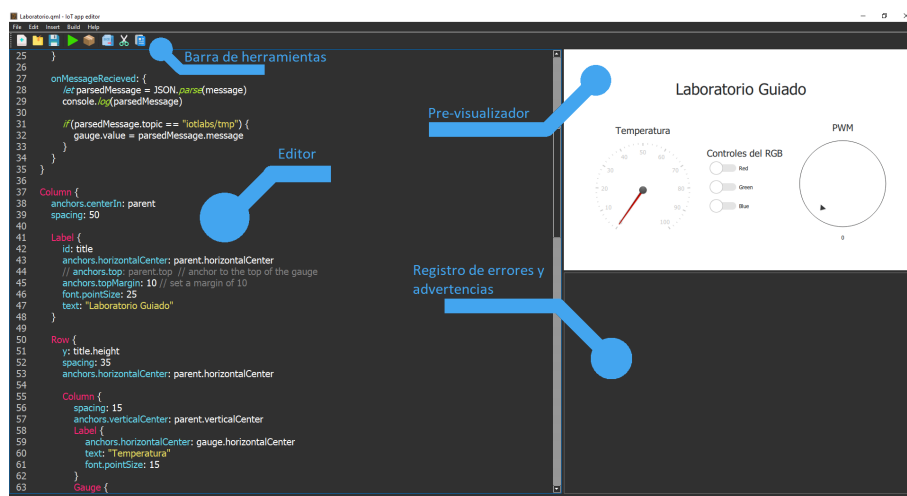


Figura 2.1: Captura de pantalla de IoT Labs IDE

La interfaz gráfica de IoT Labs IDE se divide en cuatro partes como se observa en la figura 2.1:

- Barra de herramientas: La barra de herramientas contiene accesos directos a las herramientas de edición, ejecución y empaquetado de interfaces de laboratorio.
- Editor: El editor dispone de una sección para editar el código.
- Pre-visualizador: Es un panel que permite la visualización de lo que se encuentra en el código del editor.
- Registro de errores y advertencias: Muestra los errores que se han encontrado y las líneas de código en las que se encuentran.

### 2.1.2. IoT Labs CLI

Como alternativa a IoT Labs IDE se elaboro un CLI o *Command Line Interface* con el objetivo de proporcionar una herramienta que facilite el desarrollo de interfaces de laboratorio con otros editores como VSCode, Atom o Sublime Text. El CLI se encuentra disponible en el mismo instalador descargable en el sitio web de IoT Labs [2].

El CLI dispone de las herramientas necesarias para inicializar, empaquetar y ejecutar una interfaz de laboratorio ya sea de un solo archivo o con múltiples archivos y recursos. En la figura 2.2 se puede observar el uso del CLI con el argumento **-help**, el cual muestra al usuario las opciones disponibles y los argumentos posicionales.

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Javie>iotlabs --help
Usage: iotlabs [options] command project_file
IoT Labs Command Line Interface

Options:
  -?, -h, --help    Displays help on commandline options.
  --help-all        Displays help including Qt specific options.
  -v, --version      Displays version information.

Arguments:
  command           Command to run: init, build, run, help
  project_file       Project file to use

C:\Users\Javie>

```

Figura 2.2: CLI Help

### Inicializando un proyecto de interfaz de laboratorio

Para inicializar un proyecto utilizando el CLI basta con navegar a la carpeta donde deseemos crear el proyecto y ejecutar el siguiente comando:

---

```
1 iotlabs init
```

---

El efecto que tendrá el comando se encuentra en la generación de dos archivos. Uno .json y otro .qml. El archivo .json le permitirá al CLI tener la lista de archivos del proyecto.

Como se observa el primer argumento posicional es el comando a ejecutar, en este caso *init*. Mientras que el segundo argumento posicional fue omitido con lo que el CLI utiliza uno por omisión. En caso de que especifique el segundo argumento posicional el archivo QML generado tomara ese nombre.

### Anatomía del archivo de configuración

El archivo `.json` generado tiene tres elementos como se muestra en el ejemplo. El campo `"main"` contiene el nombre del archivo principal de la interfaz de laboratorio, es decir el archivo principal. El campo `"components"` contiene un *array* con los nombres de los elementos personalizados, es posible que este vacío. El campo `"resources"` contiene los nombres de los archivos que no son código, por ejemplo imágenes.

---

```
1  {
2    "components": [],
3    "main": "Lab.qml",
4    "resources": []
5  }
```

---

Para agregar componentes personalizados basta con crear el archivo y añadir una entrada en el campo `"components"` en el archivo de configuración al igual que para agregar imágenes.

### Empaquetando la interfaz de laboratorio

Para empaquetar la interfaz de laboratorio el usuario deberá tener el archivo de compilación actualizado y ejecutar el comando *build*.

---

```
1  iotlabs build
```

---

Como resultado se generará un archivo ZIP, el cual contiene todos los archivos necesarios por la plataforma y por consiguiente se puede cargar a la plataforma.

### Ejecutando la interfaz de laboratorio

El comando *run* del CLI le permite al usuario ejecutar la interfaz de laboratorio. Para esto no es necesario que el archivo de configuración este actualizado, sin embargo se recomienda mantener el archivo actualizado para evitar futuros problemas.

---

```
1  iotlabs run
```

---

El comando *run* creará una ventana en la que se mostrará la interfaz de laboratorio generada por el usuario con toda la funcionalidad que tendría en la página web como se muestra en la figura 2.3.

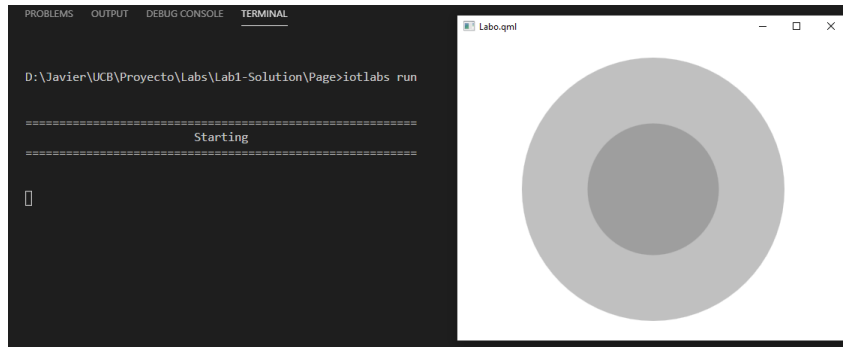


Figura 2.3: CLI run

## 2.2. Qt Markup Language (QML)

El lenguaje utilizado para la elaboración se denomina QML o Qt Markup Language. Se trata de un lenguaje declarativo creado en el marco del framework Qt. La sintaxis de utiliza es similar a la de JavaScript pero presenta Tipos propios al igual que una semántica propia. A continuación se explicara de forma concisa como se compone un archivo QML para IoT Labs. En caso de requerir más información se recomienda visitar el sitio web oficial de Qt [4].

### 2.2.1. Estructura de un archivo QML

Los archivos QML se pueden identificar observando su extensión (.qml). Todo archivo QML tiene dos partes:

- Importación de módulos: Sirve para importar los módulos que se desean utilizar en el archivo QML. Es importante considerar que los módulos tienen nombre y versión, puesto que existen múltiples versiones de cada módulo disponible.
- Cuerpo del documento: El cuerpo del documento es el lugar donde se escribe el código QML de la interfaz de laboratorio, normalmente inicia con el elemento `Item`.

---

```

1  import QtQuick 2.12
2  import QtQuick.Controls 2.12
3  import QtQuick.Layouts 1.15
4  import Controls 1.0
5  import IMT.IoTLabsWS 1.0
6
7  // Styling imports
8  import QtQuick.Controls.Material 2.12
9
10
11  Item {
12      id: rootItem
13      // La interfaz de laboratorio va aquí
14  }
```

---

En este caso concreto las líneas 1-8 son importaciones de módulos y el cuerpo del documento se encuentra en las líneas 11-14.

### 2.2.2. Layout

Qt ofrece un conjunto de elementos dentro del framework QtQuick que pueden ser utilizados para construir Layouts. Muchos de estos tienen usos avanzados por lo que en este manual no se mostrarán todos los elementos de layout. Por el contrario, solo se detallarán los elementos *Row* y *Column*.

Los elementos *Row* y *Column* son los elementos más básicos que se utilizan para construir Layouts. Sin embargo, son suficientes para muchos escenarios.

Cuando se busca contener varios elementos en una fila el elemento *Row* es recomendable.

---

```
1      import QtQuick 2.12
2      import QtQuick.Controls 2.12
3      import QtQuick.Layouts 1.15
4      import Controls 1.0
5      import IMT.IoTLabsWS 1.0
6
7      // Styling imports
8      import QtQuick.Controls.Material 2.12
9
10
11     Item {
12         id: rootItem
13
14         Row {
15             width: 100
16             height: 50
17             spacing: 10
18
19             Button {
20                 text: "Button 1"
21             }
22             Button {
23                 text: "Button 2"
24             }
25             Button {
26                 text: "Button 3"
27             }
28             Button {
29                 text: "Button 4"
30             }
31         }
32     }
```

---





Figura 2.4: Resultado del ejemplo Row

En caso de que se necesite colocar varios elementos en una columna el elemento Column puede ser utilizado.

---

```

1      import QtQuick 2.12
2      import QtQuick.Controls 2.12
3      import QtQuick.Layouts 1.15
4      import Controls 1.0
5      import IMT.IoTLabsWS 1.0
6
7      // Styling imports
8      import QtQuick.Controls.Material 2.12
9
10
11     Item {
12         id: rootItem
13
14         Column {
15             width: 100
16             height: 50
17             spacing: 10
18
19             Button {
20                 text: "Button 1"
21             }
22             Button {
23                 text: "Button 2"
24             }
25             Button {
26                 text: "Button 3"
27             }
28             Button {
29                 text: "Button 4"
30             }
31         }
32     }

```

---

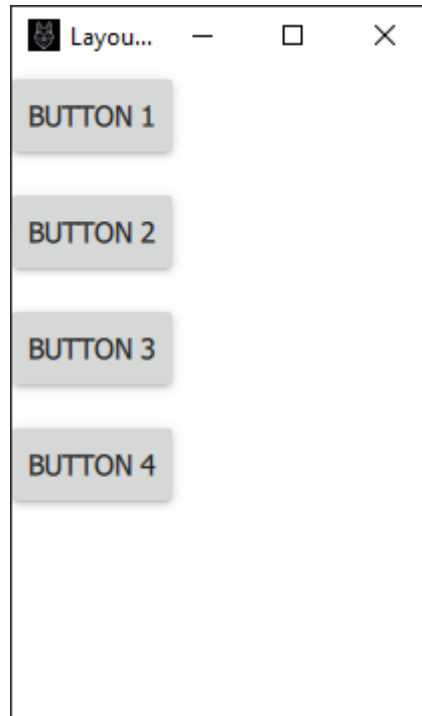


Figura 2.5: Resultado del ejemplo Column

Como nota adicional se puede mencionar que ambos elementos, Row y Column, tienen propiedades similares.

### 2.2.3. Anchors

Las anclas o Anchors son una característica que se puede aplicar sobre cualquier elemento visual. El propósito es, como su nombre lo indica, anclar un elemento visual a otro elemento visual.

Existen muchos tipos de Anchors que sirven para distintas cosas. Algunos ejemplos de Anchors son:

---

```

1      import QtQuick 2.12
2      import QtQuick.Controls 2.12
3      import QtQuick.Layouts 1.15
4      import Controls 1.0
5      import IMT.IoTLabsWS 1.0
6
7      // Styling imports
8      import QtQuick.Controls.Material 2.12
9
10
11     Item {
12         id: rootItem
13
14         Rectangle {
15             id: mainRectangle

```

```

16         width: 200
17         height: 200
18
19         // también es posible utilizar códigos de color.
20         // #ff0000 para el color rojo
21         color: 'red'
22
23         // Ancla el rectángulo al centro del elemento padre.
24         // En este caso el elemento root
25         anchors.centerIn: parent
26
27     Rectangle {
28         width: 50
29         height: 50
30         color: 'green'
31
32         // Ancla el rectángulo al centro del elemento padre.
33         // En este caso el rectángulo con id mainRectangle
34         anchors.centerIn: parent
35     }
36
37     Rectangle {
38         width: 50
39         height: 50
40         color: 'yellow'
41
42         // Ancla el lado izquierdo del rectángulo con el lado
43         // izquierdo del elemento padre
44         anchors.left: parent.left
45     }
46 }
47
48 Rectangle {
49     width: 50
50     height: 50
51     color: 'blue'
52
53     // En conjunto estas dos anclas hacen que la esquina superior
54     // izquierda coincida con la esquina inferior derecha del
55     // rectángulo con id mainRectangle
56     anchors.top: mainRectangle.bottom
57     anchors.left: mainRectangle.right
58 }
59
60 Rectangle {
61     width: 50
62     height: 50
63     color: '#00ffff'

```

```
64
65     // Las anclas tambien pueden utilizarse para colocar
66     // margenes entre elementos
67     // En este caso se ancla el rectangulo con el rectangulo
68     // mainRectangle con una separacion de 50
69     anchors.top: mainRectangle.bottom
70     anchors.topMargin: 50
71     // Esta ancla alinea los centros horizontales de
72     // ambos rectangulos
73     anchors.horizontalCenter: mainRectangle.horizontalCenter
74 }
75 }
```

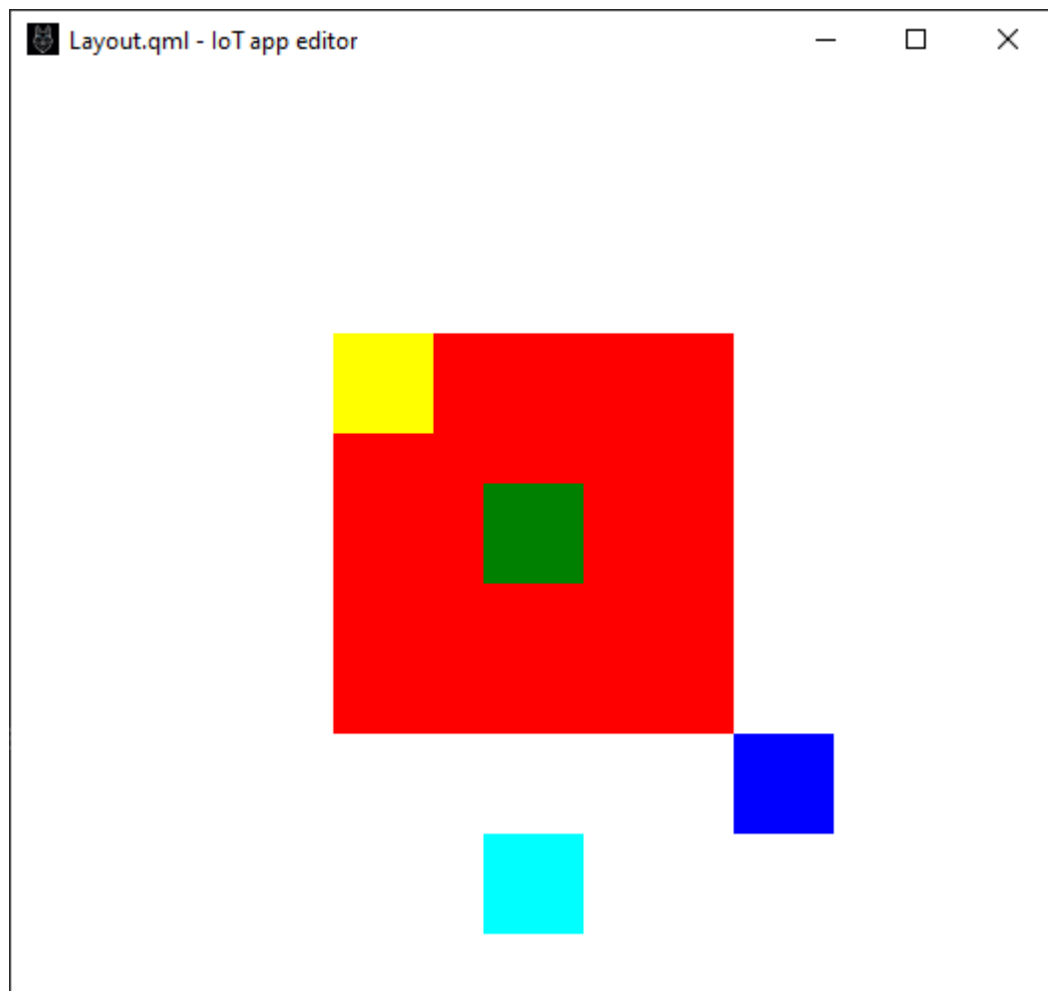


Figura 2.6: Resultado del ejemplo Anchors

## 2.2.4. Controles

En QML los controles son definidos como los elementos visuales que interactúan con el usuario. QtQuick existen muchos controles que sirven para muchas cosas.

---

```
1  import QtQuick 2.12
2  import QtQuick.Controls 2.12
3  import QtQuick.Layouts 1.15
4  import Controls 1.0
5  import IMT.IoTLabsWS 1.0
6
7  // Styling imports
8  import QtQuick.Controls.Material 2.12
9
10
11  Item {
12      id: rootItem
13
14      Column {
15          anchors.fill: parent
16
17          Button {
18              text: "Button text"
19              onClicked: {
20                  console.log("Button was clicked")
21              }
22          }
23
24          Switch {
25              text: "Switch text"
26              onCheckedChanged: {
27                  console.log(`Switch toggled`)
28                  // checked property holds switch state
29              }
30          }
31
32          Dial {
33              value: 0
34              from: 0
35              to: 1
36              onValueChanged: {
37                  // this is a javascript function
38                  // provided as callback for on value changed event
39                  console.log(`value changed`)
40                  // access to this dial value property
41              }
42          }
43
44          Slider {
45              value: 0
46              from: 0
47              to: 1
48              onValueChanged: {
```

```

49         // this is a javascript function
50         // provided as callback for on value changed event
51         console.log(`value changed`)
52         // access to this dial value property
53     }
54 }
55
56 Label {
57     text: "Label text"
58 }
59
60 TextField {
61     onAccepted: {
62         console.log("Accepted")
63     }
64 }
65
66 Gauge {
67     value: 50
68     width: parent.width / 2
69     height: parent.width / 2
70 }
71 }
72 }
```

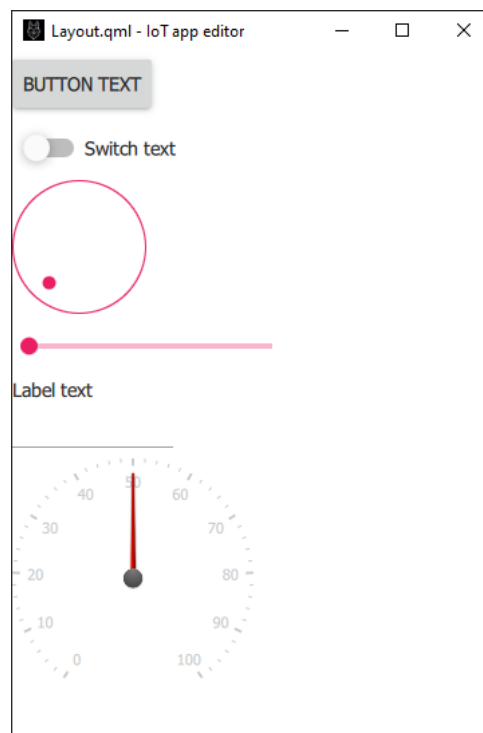


Figura 2.7: Ejemplos de Controles

### 2.2.5. Property binding

La conexión de propiedades o *property binding* es una característica de QML que facilita la propagación de eventos entre elementos. En lugar de utilizar los callbacks del evento se puede tomar la propiedad deseada como fuente de otra propiedad en otro elemento.

---

```

1      import QtQuick 2.12
2      import QtQuick.Controls 2.12
3      import QtQuick.Layouts 1.15
4      import Controls 1.0
5      import IMT.IoTLabsWS 1.0
6
7      // Styling imports
8      import QtQuick.Controls.Material 2.12
9
10
11     Item {
12         id: rootItem
13
14         Gauge {
15             id: gauge
16             width: 200
17             height: 200
18             min: 0
19             max: 100
20             anchors.centerIn: parent
21
22             // gauge.value is binded to sourceDial.value
23             // Whenever the dial is moved gauge value
24             // gets updated automatically
25             value: sourceDial.value
26         }
27
28         Dial {
29             id: sourceDial
30             value: 0
31             from: 0
32             to: 100
33             anchors.horizontalCenter: parent.horizontalCenter
34             anchors.top: gauge.bottom
35             anchors.topMargin: 20
36
37             // In this case we are using property binding
38             // No need for callback
39             onValueChanged: {
40                 // this is a javascript function provided as callback for on value changed event
41                 console.log(`value changed`) // access to this dial value property
42             }

```

```
43     }  
44 }
```

---

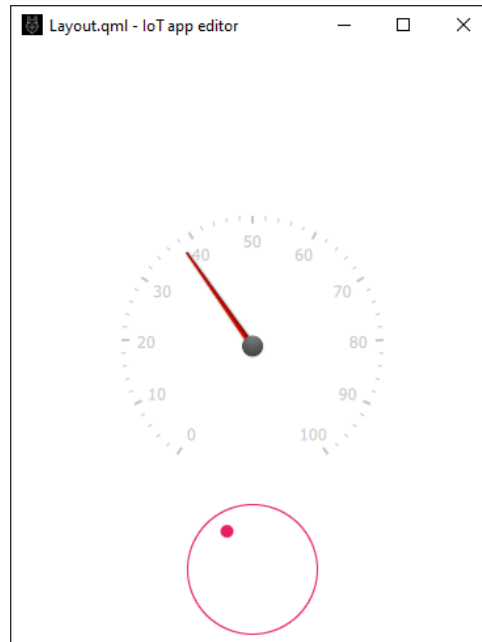


Figura 2.8: Ejemplo de property binding

### 2.2.6. Elementos personalizados

En QML es posible crear elementos personalizados utilizando otros elementos. En el siguiente ejemplo se muestra la construcción de un botón sin la utilización del elemento `Button`. El propósito del siguiente ejemplo es el de mostrar que es posible crear elementos incluso más complejos utilizando los mismos conceptos.

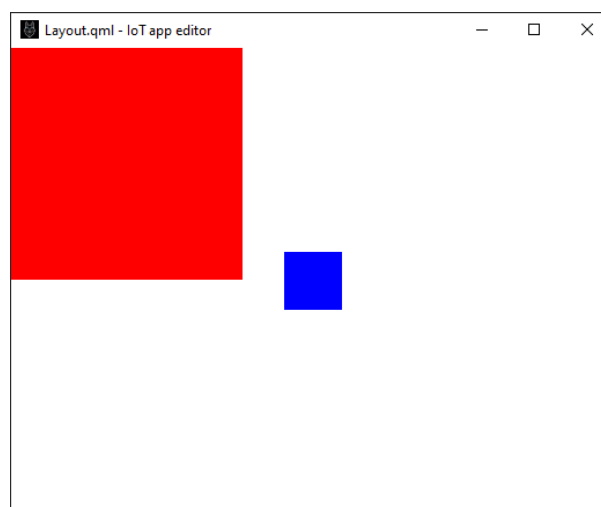


Figura 2.9: Ejemplo de Elemento personalizado



---

```
1  import QtQuick 2.12
2  import QtQuick.Controls 2.12
3  import QtQuick.Layouts 1.15
4  import Controls 1.0
5  import IMT.IoTLabsWS 1.0
6
7  // Styling imports
8  import QtQuick.Controls.Material 2.12
9
10
11  Rectangle {
12      width: 200
13      height: 200
14      color: 'red'
15
16      // Utilizando la palabra clave signal se puede
17      // definir un evento
18
19      // Cuando el evento customButtonClicked
20      // es disparado el callback se llama
21      // onCustomButtonClicked
22      // el prefijo on se añade automáticamente y
23      // la primera letra se convierte en mayúscula
24      signal customButtonClicked
25
26      TapHandler {
27          onPressed: customButtonClicked()
28      }
29  }
```

---

Listing 1: CustomButton.qml

---

```
1  import QtQuick 2.12
2  import QtQuick.Controls 2.12
3  import QtQuick.Layouts 1.15
4  import Controls 1.0
5  import IMT.IoTLabsWS 1.0
6
7  // Styling imports
8  import QtQuick.Controls.Material 2.12
9
10
11 Item {
12     id: rootItem
13
14     CustomButton {
15         onClicked: {
16             rect.colorDefiner = rect.colorDefiner + 1
17         }
18     }
19
20     Rectangle {
21         id: rect
22         width: 50
23         height: 50
24         color: colorDefiner%2 == 0? 'red': 'blue'
25
26         property int colorDefiner: 0
27
28         anchors.centerIn: parent
29     }
30 }
```

---

Listing 2: userApp.qml

En el ejemplo se utilizan dos ficheros QML, userApp.qml y CustomButton.qml <sup>1</sup>. En CustomButton.qml se define el elemento que deseamos crear. La palabra signal se utiliza para definir un evento, o como se denominan en Qt una señal.

Las señales pueden ser consumidas mediante callbacks o conexiones. En el primer caso el callback se denomina como la señal y pero añadiendo la palabra *on* y la primera letra se convierte en mayúscula.

Por otro lado el archivo userApp.qml es la fuente de la interfaz de laboratorio. Dentro se crea una instancia del elemento CustomButton <sup>2</sup> y un elemento Rectangle.

En la interfaz el rectángulo más grande, que corresponde al elemento personalizado, responde a eventos de click. Y cada vez se presiona se dispara el evento on-

---

<sup>1</sup>Es necesario que ambos archivos se encuentren en la misma carpeta

<sup>2</sup>Es necesario que el nombre del archivo del elemento personalizado empiece con una letra mayúscula, de otra forma no podrían ser reconocidos

CustomButtonClicked, el cual se consume para modificar el valor de la propiedad colorDefiner del rectángulo ocasionando un cambio de color.

### 2.2.7. Cargar a la plataforma

Al momento de cargar a la plataforma existen dos casos:

- Cuando solo existe un archivo
- Cuando se tiene más de un archivo

#### Caso de un solo archivo

El procedimiento a seguir para el primero caso, cuando existe un solo archivo, es el siguiente <sup>3</sup>:

- Modificar el nombre del archivo a userApp.qml: Es posible que el usuario utilice nombres diferentes en el proceso de desarrollo. Sin embargo, el sistema de compilación requiere que el archivo principal, generado por el usuario, se llame userApp.qml
- Crear un archivo ZIP con userApp.qml como contenido
- Cargar el archivo ZIP a la plataforma: En la tarjeta del laboratorio existe un botón con un icono de nube. Dicho botón despliega la ventana modal para cargar el archivo ZIP

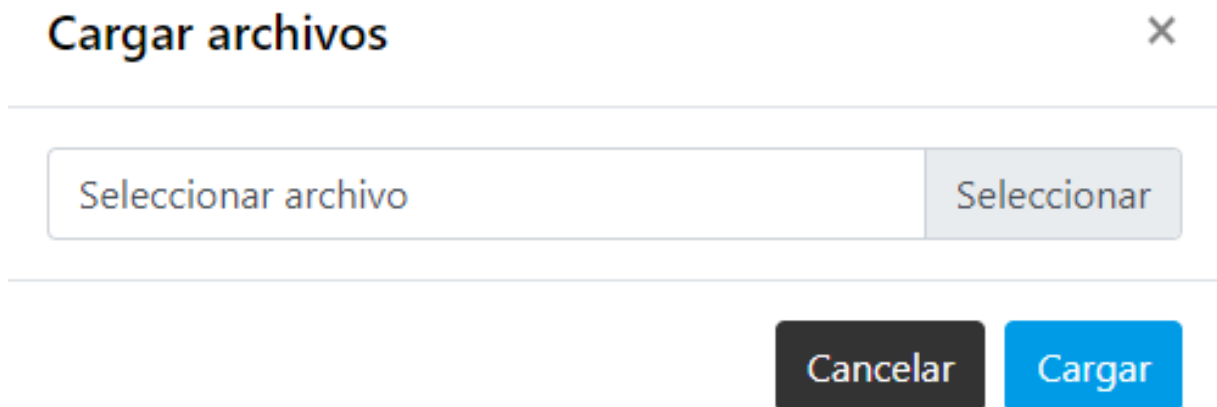


Figura 2.10: Modal para cargar el archivo ZIP en la plataforma

- Esperar los resultados de la compilación: Cuando el proceso de compilación termina se envía un correo electrónico al usuario notificando que el proceso ha concluido. El correo electrónico puede contener un mensaje satisfactorio o el registro de errores que han sido detectados.

<sup>3</sup>Para el caso de un solo archivo el editor IoT Labs IDE proporciona una función que realiza los primeros dos pasos.



Figura 2.11: Notificación de los resultados de la compilación

### Caso de múltiples archivos

En el caso de múltiples archivos es necesario crear un archivo adicional denominado *qml.qrc*. El archivo es utilizado por el compilador para incluir archivos adicionales. Estos archivos adicionales pueden imágenes o archivos QML.

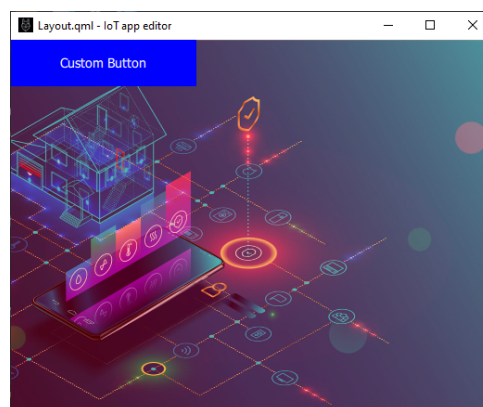


Figura 2.12: Interfaz de laboratorio con múltiples archivos

La estructura de ficheros de la siguiente interfaz de laboratorio es la siguiente:

- userApp.qml (Listing 3)
- CustomButton.qml (Listing 4)
- assets/
  - img/
    - login\_background.jpg (Figura 2.13)

Es decir que existen dos archivos QML en la carpeta base y una imagen en la carpeta img/ dentro de la carpeta assets/.



Figura 2.13: assets/img/login\_background.jpg

Se ha modificado el elemento CustomButton utilizado anteriormente para agregar una propiedad de tipo string, el texto del botón.

Si se desea cargar esta interfaz de laboratorio a la plataforma es necesario crear el manifiesto de todos estos archivos. Para ellos basta con crear un archivo con el nombre qml.qrc. Una vez creado el archivo la estructura de ficheros quedará de la siguiente forma:

- userApp.qml (Listing 3)
- CustomButton.qml (Listing 4)
- qml.qrc (Listing 5)
- assets/
  - img/
    - login\_background.jpg (Figura 2.13)

El contenido del archivo es similar al de un archivo XML. Como se ve en el archivo 5 el contenido se divide en tres niveles.

Los tags (RCC) y (qresource prefix="/") siempre deben estar presentes ya que son necesarios para el compilador. Finalmente los tags de tipo (file) contienen una lista

de los ficheros de la interfaz de laboratorio. Es necesario mencionar que algunos de estos archivos son generados por IoT Labs para realizar la compilación por lo que deben estar presentes en el archivo `qml.qrc`. Las siguientes entradas son necesarias para el correcto funcionamiento de la interfaz de laboratorio.

- `main.qml`
- `Controls/qmldir`
- `Controls/plugins.qmltypes`
- `Controls/Gauge.qml`
- `userApp.qml`

De las cuales solo `userApp.qml` es generado por el usuario mientras que el resto son generados al momento de compilar la interfaz de laboratorio. Por otro lado los archivos adicionales que se agreguen al proyecto deben estar listados en este archivo. Ese es el caso del elemento `CustomButton.qml` y de la imagen `assets/img/login_background.jpg` <sup>4</sup>.

Por ultimo se debe crear un archivo ZIP con todos los archivos y carpetas como contenido. Dicho archivo se debe cargar al igual que en el tercer paso del caso anterior (Figura 2.10).

---

<sup>4</sup>Es importante que los archivos QML adicionales se encuentren en la misma carpeta que `userApp.qml`. No se deberían poner los archivos QML en niveles de carpetas más profundos

---

```
1  import QtQuick 2.12
2  import QtQuick.Controls 2.12
3  import QtQuick.Layouts 1.15
4  import Controls 1.0
5  import IMT.IoTLabsWS 1.0
6
7  // Styling imports
8  import QtQuick.Controls.Material 2.12
9
10
11  Item {
12      id: rootItem
13
14      Image {
15          anchors.fill: parent
16          source: "assets/img/login_background.jpg"
17          fillMode: Image.PreserveAspectCrop
18
19          Rectangle {
20              id: rect
21              width: 50
22              height: 50
23              color: colorDefiner%2 == 0? 'red': 'blue'
24              opacity: 0.3
25
26              property int colorDefiner: 0
27
28              anchors.fill: parent
29          }
30      }
31
32      CustomButton {
33          height: 50
34          text: 'Custom Button'
35          color: 'blue'
36
37          onClickCustomButtonClicked: {
38              rect.colorDefiner = rect.colorDefiner + 1
39          }
40      }
41  }
```

---

Listing 3: userApp.qml

---

```
1  import QtQuick 2.12
2  import QtQuick.Controls 2.12
3  import QtQuick.Layouts 1.15
4  import Controls 1.0
5  import IMT.IoTLabsWS 1.0
6
7  // Styling imports
8  import QtQuick.Controls.Material 2.12
9
10
11  Rectangle {
12      width: 200
13      height: 200
14      color: 'red'
15
16      property string text: ''
17
18      signal customButtonClicked
19
20      Label {
21          text: parent.text
22          anchors.centerIn: parent
23          color: 'white'
24      }
25
26      TapHandler {
27          onTap: customButtonClicked()
28      }
29  }
```

---

Listing 4: CustomButton.qml



---

```
1 <RCC>
2     <qresource prefix="/">
3         <file>main.qml</file>
4         <file>userApp.qml</file>
5         <file>Controls/qmlDir</file>
6         <file>Controls/plugins.qmltypes</file>
7         <file>Controls/Gauge.qml</file>
8         <file>assets/img/login_background.jpg</file>
9         <file>CustomButton.qml</file>
10    </qresource>
11 </RCC>
```

---

Listing 5: qml.qrc

# Capítulo 3

## Sistema de comunicación

El sistema de comunicación en tiempo real que proporciona IoT Labs a sus usuarios utiliza los protocolos MQTT y WebSocket. Ambos se encuentran conectados, es decir que si se publica un mensaje por MQTT será enviado a todos sus suscriptores tanto por MQTT como por WebSocket y viceversa.

En el caso de utilizar el protocolo MQTT se debe utilizar un cliente que utilice las versiones 3.1 o 3.1.1 establecidas en el estándar de OASIS [3]. Algunos clientes disponibles son:

- PahoMQTT
- PubSubClient
- Particle MQTT

### 3.1. Obteniendo credenciales para el sistema de comunicación

La barra de navegación de la plataforma tiene 5 páginas disponibles para distintas tareas. De todas esas páginas las primeras dos son necesarias para utilizar el sistema de comunicación.

- La primera página se pueden crear, editar, borrar y visualizar las interfaces de laboratorio.
- La segunda página se pueden registrar, editar, borrar y visualizar los dispositivos.

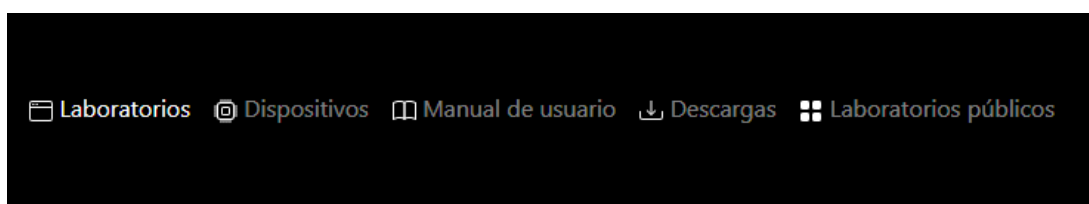


Figura 3.1: Barra de navegación

En caso de utilizar MQTT el sistema toma a dicho cliente como dispositivo. No obstante, es posible utilizar MQTT en otras plataformas. Por ejemplo, para la creación de un servicio Web o para la integración con aplicaciones externas.

Los clientes MQTT tienen propiedades para almacenar usuarios y contraseñas y utilizarlas para autenticarse con el Broker MQTT al que se desean conectar. Estas propiedades se utilizan en el Broker de IoT Labs para autenticar a los dispositivos que registran los usuarios. Cuando se registra un dispositivo en IoT Labs se generan credenciales de forma automática.

Para registrar un dispositivo se debe utilizar el botón [+] como se muestra en la figura 3.2.

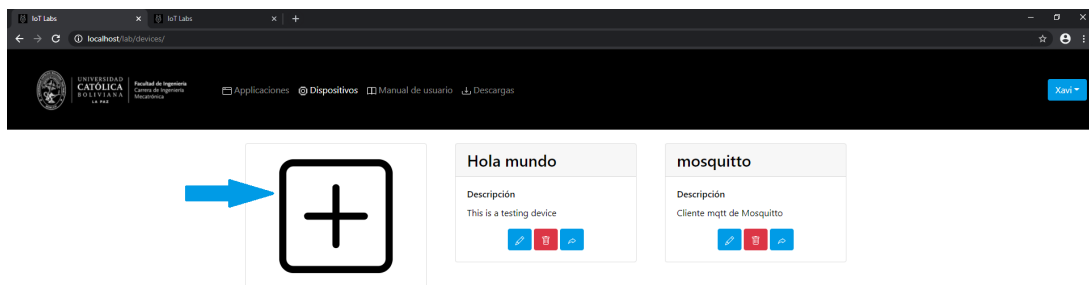


Figura 3.2: Página de dispositivos

Luego se debe llenar el formulario que se muestra en la vista modal en la figura 3.3.

### Crear dispositivo

Nombre

Descripción

Cancelar

Crear

Figura 3.3: Formulario de registro de dispositivo

Finalmente se obtiene el Token presionando el botón “Copiar API key” como se muestra en la figura 3.4.



Figura 3.4: Obtener el Token del dispositivo

Este procedimiento colocará las credenciales del dispositivo en el portapapeles del sistema. Por lo que el siguiente paso es el de pegar el contenido del portapapeles en el cliente MQTT.

El formato de las credenciales es el siguiente:

---

```

1  deviceID: "604e75bbbe51a6096999a6f0"
2  apiKey:  "ed36d975165caea183030db01be1206092aa2d3a7942eefe917762355e8866f0"

```

---

Para los equivalentes en los clientes MQTT el campo "deviceID" es el nombre de usuario mientras que el campo "apiKey" es la contraseña.

Por otro lado el protocolo WebSocket esta pensado para ser utilizado en las interfaces de laboratorio por lo que el procedimiento para obtener las credenciales se lleva a cabo en la página de Laboratorios.

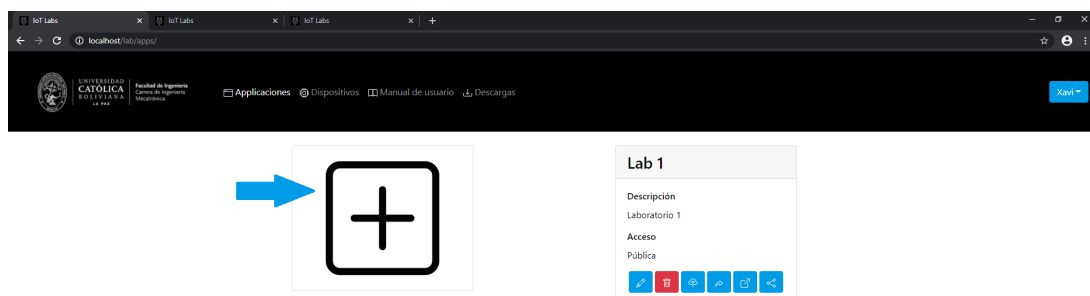
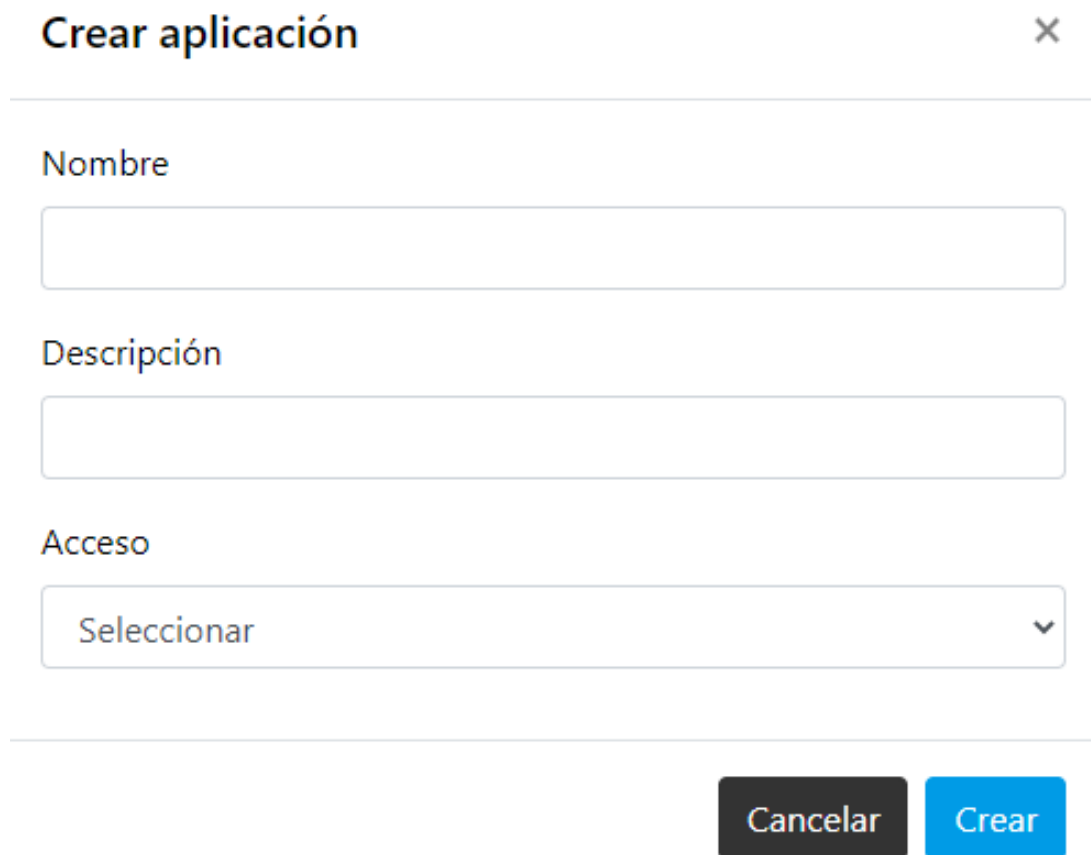


Figura 3.5: Página de laboratorios

Al igual que los dispositivos, las interfaces de laboratorio deben ser registradas antes de poder conectarse al sistema de comunicación de IoT Labs. Una interfaz se puede registrar utilizando el botón [+] de la página de Laboratorios como se observa en la figura 3.5.

Luego de debe llenar el formulario de registro de interfaces de laboratorio que se muestra en la figura 3.6. El campo "Acceso" se refiere al estado de publicación de la interfaz de laboratorio entre "Público" y "Privado".



El formulario se titula "Crear aplicación" y tiene un botón de cerrar (X) en la esquina superior derecha. Contiene tres campos de entrada: "Nombre" (un campo de texto simple), "Descripción" (un campo de texto simple), y "Acceso" (un menú desplegable con la opción "Seleccionar" y una flecha hacia abajo). En la parte inferior derecha hay dos botones: "Cancelar" (gris oscuro) y "Crear" (azul).

Figura 3.6: Formulario de registro de interfaces de laboratorio

Finalmente se debe utilizar el botón "Copiar API Key" como se muestra en la figura como se muestra en la figura 3.7.

El formato de las credenciales obtenidas es similar al formato para dispositivos. En este caso los nombres de los campos corresponden a los nombres de las propiedades utilizadas en la API de comunicación para interfaces de laboratorio.

---

<sup>1</sup> `appID: "604e8c7dbe51a6096999a6f1"`

<sup>2</sup> `apiKey: "11732260025cc73bd6a2ffc72cb684b786a47fce584d2d0292d68ea05f66e58f"`

---



Figura 3.7: Obtener el Token de la interfaz de laboratorio

## 3.2. API de comunicación para las interfaces de laboratorio

Las interfaces de laboratorio se encuentran construidas sobre QtQuick por lo que la API de comunicación es accesible en el lenguaje QML. Se trata de un cliente que utiliza el protocolo WebSocket para enviar y recibir mensajes.

El cliente presenta dos métodos que cumplen funciones similares a las encontradas en MQTT, “publish” y “subscribe”. Estos métodos permiten al usuario enviar y recibir mensajes directamente en las interfaces de laboratorio.

Para tener acceso a la API de comunicación es necesario importar el módulo con el siguiente código

---

```
1 import IMT.IoTLabsWS 1.0
```

---

Una vez que el módulo se ha importado es posible crear instancias del elemento IoT LabsWS.

---

```

1      IoTLabsWS {
2          id: client
3          host: "wss://imt-iotlabs.net"
4          appID: "604e8c7dbe51a6096999a6f1"
5          apiKey: "11732260025cc73bd6a2ffc72cb684b786a47fce584d2d0292d68ea05f66e58f"
6
7          Component.onCompleted: {
8              connectServer()
9          }
10
11         onConnectionEstablished: {
12             // subscribe("topicToSubscribe")    // subscribe to login responses
13         }
14
15         onCredentialsRejected: {
16             // console.log("Provided credentials were rejected by the server")
17         }
18
19         onMessageRecieved: {
20             let parsedMessage = JSON.parse(message)
21             // console.log(parsedMessage.topic)
22             // console.log(parsedMessage.message)
23         }
24     }

```

---

Cada vez que se instancia el elemento IoT LabsWS es necesario inicializar todas las propiedades y callbacks.

- La propiedad `id` es común a todos los elementos instanciados en QML, sirve para acceder a la instancia y sus propiedades o métodos.
- La propiedad `host` indica la URL del servidor de WebSockets.
- La propiedad `appID` es proporcionada en las credenciales de la interfaz de laboratorio.
- La propiedad `apiKey` es proporcionada en las credenciales de la interfaz de laboratorio.
- El callback `Component.onCompleted` se ejecuta cuando el elemento ha sido instanciado y sirve para establecer conexión con el servidor.
- El callback `onConnectionEstablished` se ejecuta cuando se ha establecido conexión con el servidor. Puede ser utilizado para notificar que el cliente esta conectado y enviar mensajes o suscribirse a topics.
- El callback `onCredentialsRejected` se ejecuta cuando las credenciales han sido rechazadas. En ese caso deberás verificar las credenciales en la plataforma.

- El callback `onMessageReceived` se ejecuta cuando se recibe un mensaje. El mensaje debe ser procesado con `JSON.parse` como se ve en el ejemplo. Luego se podrá acceder al topic y payload (message) para ser procesadas por el resto de la interfaz de laboratorio.

Luego de instanciar el cliente de WebSocket se pueden enviar mensajes a cualquier topic utilizando el método `publish` y suscribirse a cualquier topic utilizando el método `subscribe`. La sintaxis de ambos métodos es la siguiente:

```
1 client.publish ("topic/to/publish", "message")
2 client.subscribe ("topic/to/subscribe")
```

### 3.3. Cliente de pruebas

La plataforma IoT Labs proporciona entre sus herramientas un cliente MQTT con el que los estudiantes puedan enviar y recibir mensajes con el objetivo de probar sus laboratorios antes de terminar una característica o durante el desarrollo. Este cliente se encuentra en el instalador disponible en la página de descargas [2].

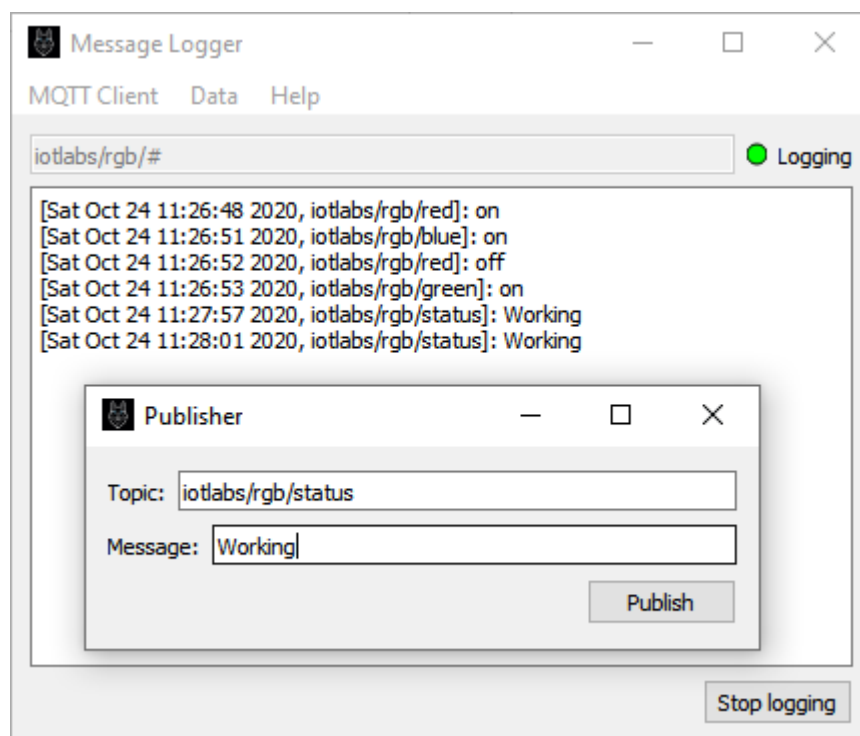


Figura 3.8: Interfaz gráfica del cliente de pruebas

En la figura 3.8 se puede observar la interfaz del cliente de pruebas "MessageLogger" enviando y recibiendo mensajes con el protocolo MQTT.

Para empezar a utilizar la herramienta "MessageLogger" es necesario registrar un dispositivo en la plataforma. Recordando que cualquier cliente que utilice el protocolo MQTT para comunicarse es considerado un dispositivo en la plataforma IoT Labs.



Una vez que se haya registrado el dispositivo y se hayan obtenido sus credenciales se puede proceder a la conexión del cliente de pruebas.

El menú “MQTT Client” existe una función para establecer conexión. Cuando se ejecuta dicha función se despliega una ventana para ingresar las credenciales. Luego de ingresar las credenciales y presionar el botón “Connect” se debe observar el indicador en la esquina superior derecha de la ventana principal. El indicador muestra el estado de conectividad de MessageLogger mediante tres colores y palabras:

- Disconnected (Rojo): MessageLogger esta desconectado
- Ready (Amarillo): MessageLogger esta conectado y listo para enviar y recibir mensajes
- Logging (Verde): MessageLogger esta conectado y recibiendo mensajes.

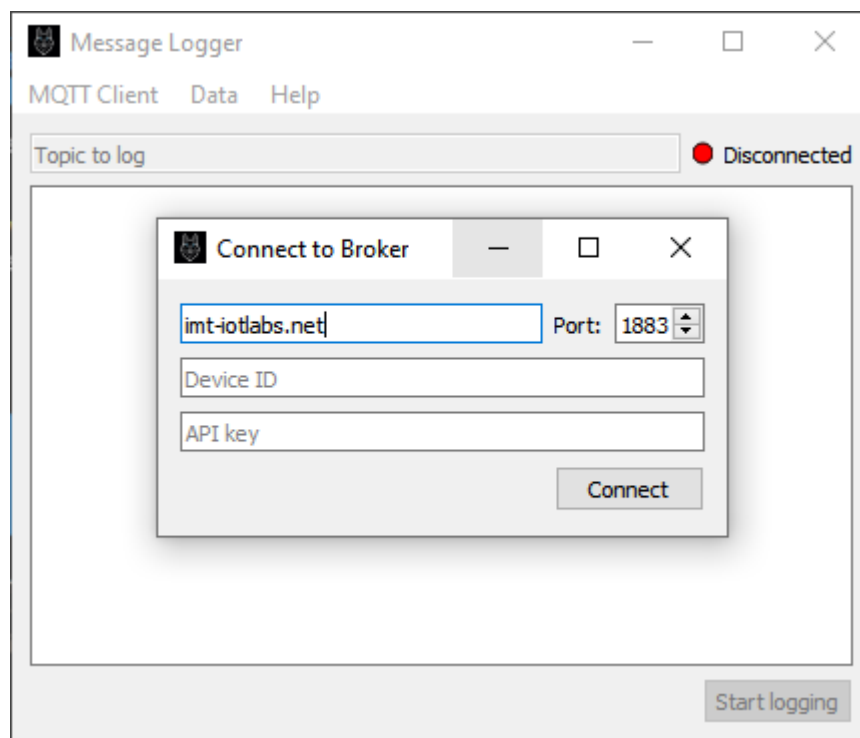


Figura 3.9: Proceso de conexión de MessageLogger

Una vez que se ha pasado al estado “Ready” el usuario puede iniciar una suscripción escribiendo el topic en el campo “Topic to log” y presionando el botón “Start logging”. Cuando se haya iniciado una suscripción el estado cambiará a “Logging” como se ve en la figura 3.10.

Mientras MessageLogger se encuentre en los estados “Ready” o “Logging” es posible enviar mensajes a cualquier topic. Para ello es necesario abrir la ventana del publicador en el menú “MQTT Client” como se ve en la figura 3.11.

Finalmente, el menú “Data” de MessageLogger tiene dos funciones que pueden resultar útiles. La primera sirve para limpiar la lista de todos los mensajes que han sido recibidos. Y la otra función sirve para exportar todos los mensajes de la lista a un archivo CSV como el de la figura 3.12.

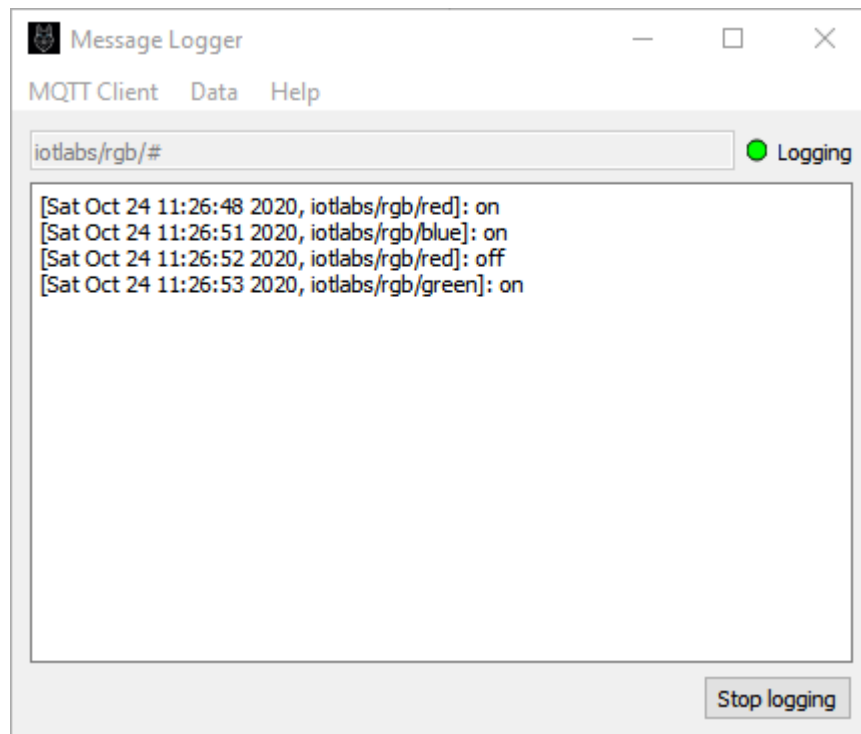


Figura 3.10: Suscripción iniciada

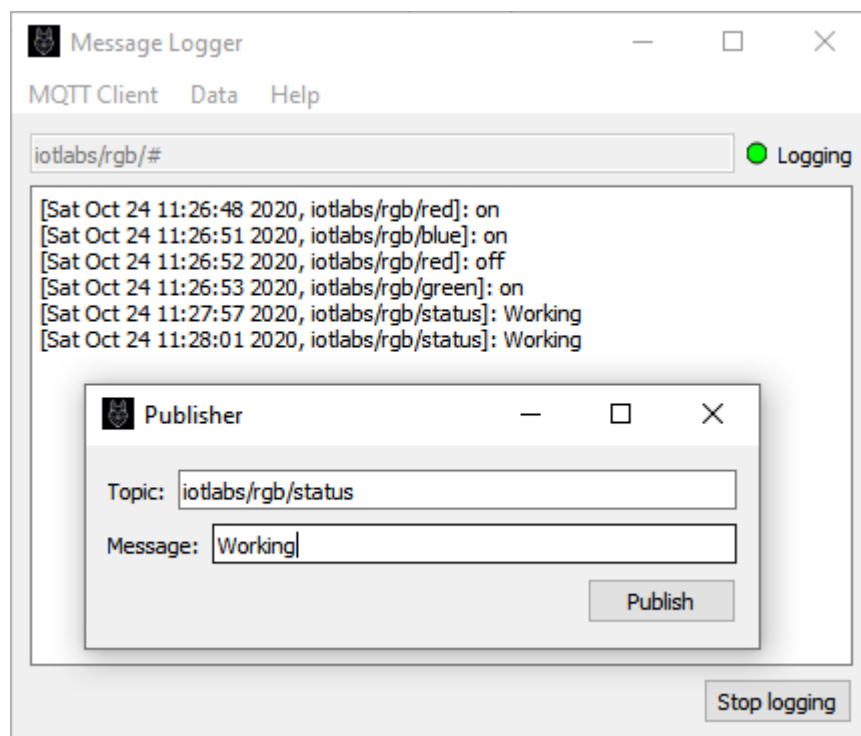


Figura 3.11: Publicador de MessageLogger

```

Timestamp, Topic, Message
Sat Oct 24 12:04:44 2020, iotlabs/rgb/red, on
Sat Oct 24 12:04:45 2020, iotlabs/rgb/blue, on
Sat Oct 24 12:04:45 2020, iotlabs/pwm, 0.044095100866552994
Sat Oct 24 12:04:45 2020, iotlabs/pwm, 0.043970868925026434
Sat Oct 24 12:04:45 2020, iotlabs/pwm, 0.045937082549562316
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.047946466972002555
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.054151214222561105
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.06071827997249402
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.07647823936863465
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.16460114907948284
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.18011837632915625
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.1944657254367239
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.20268976485034113
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.2104545278944419
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.21801263786749892
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.22498619213243273
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.22745206441403154
Sat Oct 24 12:04:46 2020, iotlabs/pwm, 0.22952652921209446
Sat Oct 24 12:04:46 2020, iotlabs/rgb/green, on
Sat Oct 24 12:04:47 2020, iotlabs/rgb/blue, off
Sat Oct 24 12:04:48 2020, iotlabs/pwm, 0.2622911716840371
Sat Oct 24 12:04:48 2020, iotlabs/pwm, 0.2648001160939205
Sat Oct 24 12:04:48 2020, iotlabs/pwm, 0.26569326280992894
Sat Oct 24 12:04:48 2020, iotlabs/pwm, 0.27267136495450595
Sat Oct 24 12:04:48 2020, iotlabs/pwm, 0.2773286350454939
Sat Oct 24 12:04:48 2020, iotlabs/pwm, 0.2847562472319659
Sat Oct 24 12:04:48 2020, iotlabs/pwm, 0.2911662526002329
Sat Oct 24 12:04:48 2020, iotlabs/pwm, 0.30476521871612827
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5365135421446611
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5424145189514078
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5511109493101783
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5559421471509022
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.56144982940974
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.565890562255972
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5711228643060188
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5837161162871485
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5885501705902599
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5920434260769482
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.5975841809241752
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.6027927731489574
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.6069406238535534
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.6109785442361554
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.6129565315468389
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.6149073796001214
Sat Oct 24 12:04:49 2020, iotlabs/pwm, 0.6168312443734946
Sat Oct 24 12:04:49 2020, iotlabs/rgb/red, off
Sat Oct 24 12:04:50 2020, iotlabs/pwm, 0.9110876322903839
Sat Oct 24 12:04:51 2020, iotlabs/rgb/blue, on
Sat Oct 24 12:04:52 2020, iotlabs/rgb/green, off
Sat Oct 24 12:04:52 2020, iotlabs/rgb/red, on

```

Figura 3.12: Archivo CSV exportado desde MessageLogger

# Bibliografía

- [1] J. CONTRERAS, *Laboratorio educativo para aplicaciones IoT*, Universidad Católica Boliviana "San Pablo", 2021.
- [2] I. LABS, *IoT Labs - Descargas*. <https://imt-iotlabs.net/downloads/>. Visitado: 13-03-2021.
- [3] OASIS, *MQTT version 3.1.1*. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Visitado: 2020-10-01.
- [4] QT, *QML*. <https://doc.qt.io/qt-5/qtqml-index.html>. Visitado: 13-03-2021.
- [5] —, *Qt for WebAssembly - Qt Wiki*. [https://wiki.qt.io/Qt\\_for\\_WebAssembly](https://wiki.qt.io/Qt_for_WebAssembly). Visitado: 05-10-2020.