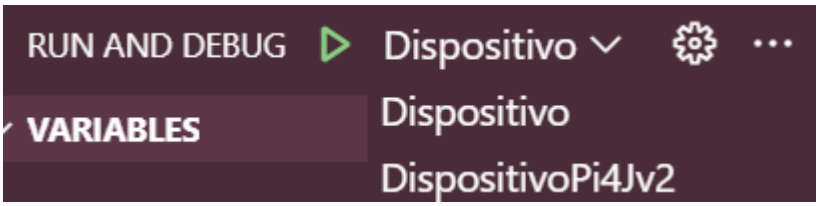
SEMINARIO 2

Aclaración inicial: se decidió construir el proyecto final sobre Maven debido a la diferencia de sistemas operativos de los integrantes, así todos podían correr y probar el proyecto en sus respectivos dispositivos.

1. Ejecución de la RasperryPi

Para ello se debe abrir el panel “Run & Debug”

y seleccionar “Dispositivo”



Luego correr el archivo para ver la ejecución.

2. Ejercicio 5.1

Se añadió la función f3 al dispositivo que por defecto inicia en el modo “parpadeo” en la clase “DispositivoIniciador” de la siguiente manera:

```
IFuncion f3 = Funcion.build(id:"f3", FuncionStatus.BLINK, deviceId, publisher);
d.addFuncion(f3);
```

3. Ejercicio 5.2

El objetivo era implementar el mensaje de estado de un dispositivo en la API REST, para ello en la clase “Dispositivo_Recurso” se implementó el método que transforma el dispositivo y sus funciones en un objeto JSON:

```
for (IFuncion funcion : dispositivo.getFunciones()) {
    JSONObject funcionJson = new JSONObject();
    funcionJson.put(key:"id", funcion.getId());
    funcionJson.put(key:"estado", funcion.getStatus());

    arrayFunciones.put(funcionJson);
}
```

En esta se construye el JSON con el id del dispositivo, se recorre la lista de funciones “IFuncion” y se añade cada una con su estado actual, y el método “GET” de la API REST al hacer la petición “/dispositivo” devuelve este JSON como respuesta.

4. Ejercicio 5.3

El objetivo era implementar el mensaje de estado de una función en la API REST, para ello se modificó la clase “Funcion Recurso” para que retorne en la petición “GET” adicional al id también el estado:

```
public static JSONObject serialize(IFuncion f) {
    JSONObject jsonResult = new JSONObject();
    try {
        jsonResult.put(key:"id", f.getId());
        jsonResult.put(key:"estado", f.getStatus());
    } catch (JSONException e) {
    }
    return jsonResult;
```

Para obtener el resultado se hace a partir de la ruta “/dispositivo/funcion/<funcion-id>, el estado actualizado se devuelve siempre en formato JSON.

5. Ejercicio 5.4

Se añadió al dispositivo la capacidad de estar habilitado o deshabilitado, de manera que si está habilitado se permite el control de todas sus funciones, de lo contrario se bloquea la modificación de sus funciones. Para ello en la clase “Dispositivo” se añadió la propiedad “habilitado” con sus respectivos métodos encargados de actualizar todas las funciones asociadas al dispositivo:

```
protected String deviceId = null;
private boolean habilitado = true;
```

```
@Override
public Boolean isHabilitado() {
    return this.habilitado;
}
```

```java
@Override
public Boolean habilitar() {
    habilitado = true;
    if (this.getFunciones() != null) {
        for (IFuncion f : this.getFunciones()) {
            f.habilitar();
        }
    }
    return habilitado;
}
```

```java
@Override
public Boolean deshabilitar() {
    habilitado = false;
    if (this.getFunciones() != null) {
        for (IFuncion f : this.getFunciones()) {
            f.deshabilitar();
        }
    }
    return habilitado;
}
```

Además en la clase “Funcion” se añadieron los métodos “habilitar” y “deshabilitar” que actualizan el estado de la función a ON/OFF y gestionan el flag interno “habilitado”.

```java
private boolean habilitado = true;
```

```java
@Override
public IFuncion encender() {

    if (!habilitado) return this;
    MySimpleLogger.info(this.loggerId, msg:"==> Encender");
    this.setStatus(FuncionStatus.ON);
    return this;
}
```

```java
@Override
public Boolean habilitar() {
    this.habilitado = true;
    return true;
}
```

```java
@Override
public Boolean deshabilitar() {
    this.habilitado = false;
    return true;
}
```

```java
@Override
public Boolean isHabilitado() {
    return this.habilitado;
}
```

6. Ejercicio 5.5

Se extendió la API REST “Dispositivo_Recurso” para permitir habilitar o deshabilitar el dispositivo mediante peticiones PUT. Además se añadió el campo “habilitado” al JSON de salida mediante la petición “/dispositivo”:

```java
public static JSONObject serialize(IDispositivo dispositivo) {
    JSONObject jsonResult = new JSONObject();

    try {
        jsonResult.put(key:"id", dispositivo.getId());
        jsonResult.put(key:"habilitado", dispositivo.isHabilitado());

        if (dispositivo.getFunciones() != null) {
            JSONArray arrayFunciones = new JSONArray();

            for (IFuncion funcion : dispositivo.getFunciones()) {
                JSONObject funcionJson = new JSONObject();
                funcionJson.put(key:"id", funcion.getId());
                funcionJson.put(key:"estado", funcion.getStatus());

                arrayFunciones.put(funcionJson);
            }


            jsonResult.put(key:"funciones", arrayFunciones);
        }
    } catch (JSONException e) {
    }

    return jsonResult;
}
```

7. Ejercicio 5.6

La propiedad “TOPIC_BASE” en la clase “Configuracion” define el profijo común de los topic MQTT utilizados por el dispositivo, es decir, establece la ruta base bajo la cual se publican y se reciben todos los mensajes MQTT relacionados con los dispositivos y sus funciones. De esta manera todas los topics se generan de forma consistente.

8. Ejercicio 5.7

El objetivo fue implementar la API MQTT del dispositivo, de forma que las funciones pudieran ser controladas mediante mensajes JSON recibidos en topics específicos. Para ello se modificó la clase Dispositivo_APIMQTT encargada de conectarse al bróker MQTT indicado para que se encargue de suscribirse a los topics de comandos de cada función del dispositivo (“es/upv/inf/muiinf/ina/dispositivo/{deviceId}/funcion/{funcionId}/comandos”), procesar mensajes recibidos en formato JSON y ejecutar las acciones correspondientes sobre la función:

```java
@Override
public void messageArrived(String topic, MqttMessage message) throws Exception {
    JSONObject payload = null;
    try {
        payload = new JSONObject(new String(message.getPayload()));
    } catch (JSONException e) {
        MySimpleLogger.warn(this.loggerId, "Mensaje no válido en topic " + topic);
        return;
    }
```

```java
String action = payload.getString("accion");
```

9. Ejercicio 5.8

Siguiendo la lógica anterior, ahora además de poderse modificar el estado de una función también se puede habilitar o deshabilitar el dispositivo, de modo que al acceder a la ruta de comandos (“es/upv/inf/muiinf/ina/dispositivo/{deviceId}/funcion/{funcionId}/comandos”) se pueda cambiar la acción del dispositivo mediante un JSON:

```java
String[] topicNiveles = topic.split(regex:"/");
if (topicNiveles[topicNiveles.length - 1].equalsIgnoreCase(anotherString:"comandos")) {
    if (action == null || action.isBlank()) {
        MySimpleLogger.warn(this.loggerId, "Mensaje sin acción en topic " + topic);
        return;
    }
}
```

```java
if (Arrays.stream(topicNiveles).anyMatch("funcion"::equals)) {
    String funcionId = topicNiveles[topicNiveles.length - 2];
    IFuncion f = this.dispositivo.getFuncion(funcionId);
    if (f == null) {
        MySimpleLogger.warn(this.loggerId, "No encontrada funcion " + funcionId);
        return;
    }
    switch (action.toLowerCase()) {
        case "encender":
            f.encender();
            break;
        case "apagar":
            f.apagar();
            break;
        case "parpadear":
            f.parpadear();
            break;
        default:
            MySimpleLogger.warn(this.loggerId, "Acción '" + action + "' no reconocida para
    }
```

```
} else {
    switch (action.toLowerCase()) {
        case "habilitar":
            dispositivo.habilitar();
            break;
        case "deshabilitar":
            dispositivo.deshabilitar();
            break;
        default:
            MySimpleLogger.warn(this.loggerId, "Acción '" + action + "' no reconocida para
    }
```

```
public void iniciar() {

    if (this.myClient == null || !this.myClient.isConnected())
        this.connect();

    if (this.dispositivo == null)
        return;

    for (IFuncion f : this.dispositivo.getFunciones())
        this.subscribe(this.calculateCommandTopic(f));

    // subscribe to device-level command topic
    this.subscribe(this.calculateDeviceCommandTopic());

}
```

10. Ejercicio 5.9

Ahora además de aceptar comandos, el dispositivo debe poder publicar información de estado de sus funciones en formato JSON, a través de MQTT. Ahora se definen dos tipos de topics asociados a cada función: comandos (“es/upv/inf/muiinf/ina/dispositivo/{deviceId}/funcion/{funcionId}/comandos”) para el control de la función, e información (“es/upv/inf/muiinf/ina/dispositivo/{deviceId}/funcion/{funcionId}/info”) para la publicación del estado actual de la función. Para este fin se definió la clase FuncionPublisher_APIMQTT:

```
public class FuncionPublisher_APIMQTT {
    protected MqttClient myClient;
    protected MqttConnectOptions connOpt;
    protected String clientId = null;
    protected String mqttBroker = null;

    public FuncionPublisher_APIMQTT(String brokerUrl, String clientId) throws MqttException {
        this.clientId = clientId;
        myClient = new MqttClient(brokerUrl, clientId);
        connOpt = new MqttConnectOptions();
        connOpt.setCleanSession(true);
        myClient.connect(connOpt);
    }

    public void publish_status(String topic, String funcionId, String estado) {
        String json = String.format("{\"id\":\"%s\", \"estado\":\"%s\"}", funcionId, estado);
        byte[] payload = json.getBytes(StandardCharsets.UTF_8);
```

```
    try {
        myClient.publish(
            topic, // topic
            payload, // payload
            1, // QoS level
            false // retained?
        );
    } catch (MqttException e) {
        MySimpleLogger.warn("FuncionPublisher", "Error publicando estado en topic " + topic + ":
    }
}

public void disconnect() {
    try {
        if (myClient.isConnected()) {
            myClient.disconnect();
        }
    } catch (MqttException e) {
        MySimpleLogger.warn("FuncionPublisher", "Error al desconectar: " + e.getMessage());
    }
}
```

11. Ejercicio 5.10

Se desarrolló un controlador semafórico que coordina dos dispositivos IoT, sem1 y sem2, cada uno representando un semáforo en una calle de un cruce. Cada semáforo tiene tres funciones: f1 para rojo, f2 para amarrillo y f3 para verde.

La solución incluye:

- task10-run.sh: script encargado de arrancar ambos semáforos y el ciclo de control.
- semaforo-controller.sh: script encargado de publicar comando MQTT que siguen la secuencia del semáforo.

Para ejecutarlo se debe abrir una terminal “GIT BASH” y utilizar los siguientes comandos:

- chmod +x task10-run.sh semaforo-controller.sh
- ./task10-run.sh

12. Ejercicio 5.11

El objetivo de este ejercicio era implementar un controlador maestro-esclavos que permitiera replicar el estado de una función de un dispositivo maestro en varios dispositivos esclavos, para ello se construyó un controlador que se suscribe al topic del maestro, puede recibir mensajes para cambiar el estado del dispositivo y replicar estos cambios en cada esclavo.

Para ejecutarlo se debe tener un bróker MQTT local corriendo en “localhost:1883”, ay ejecutar los siguiente comandos en una terminal:

- cd ejercicio_11
- mvn clean package
- java -jar target/maestro-esclavo-1.0.0.jar / mvn exec:java -Dexec.mainClass="ejercicio11.MaestroEsclavoController" (puede ser cualquiera de los 2)
- javac -cp "lib/*" -d bin src/main/java/ejercicio11/MaestroEsclavoController.java
- java -cp "bin;lib/*" ejercicio11.MaestroEsclavoController tcp://localhost:1883 ttmi050 ttmi051,ttmi052 f1

Una vez se tiene esto se puede probar el funcionamiento, para ello se deben abrir 4 terminales:

a. Terminal A: puede ver todo lo que publica el maestro.

mosquitto_sub -h localhost -t "es/upv/inf/muiinf/ina/dispositivo/ttmi050/funcion/f1/#" -v

b. Terminal B: puede ver todos los comandos enviados a los esclavos.

mosquitto_sub -h localhost -t "es/upv/inf/muiinf/ina/dispositivo/+/funcion/f1/comandos" -v

c. Terminal C: puede escuchar solo al esclavo 1.

mosquitto_sub -h localhost -t "es/upv/inf/muiinf/ina/dispositivo/ttmi051/funcion/f1/comandos" -v

d. Terminal D: puede escuchar solo al esclavo 2.

mosquitto_sub -h localhost -t "es/upv/inf/muiinf/ina/dispositivo/ttmi052/funcion/f1/comandos" -v

Luego desde otra terminal se podrá probar enviar mensajes al maestro con la siguiente estructura:

mosquitto_pub -h localhost -t "es/upv/inf/muiinf/ina/dispositivo/ttmi050/funcion/f1/info" -m '{"id":"<idfuncion>","estado":"<estado>"}'