



# Les objets connectés

## (Partie 4 : Blynk)



LE CREN Anthony

### 1 Qu'est-ce que Blynk ?

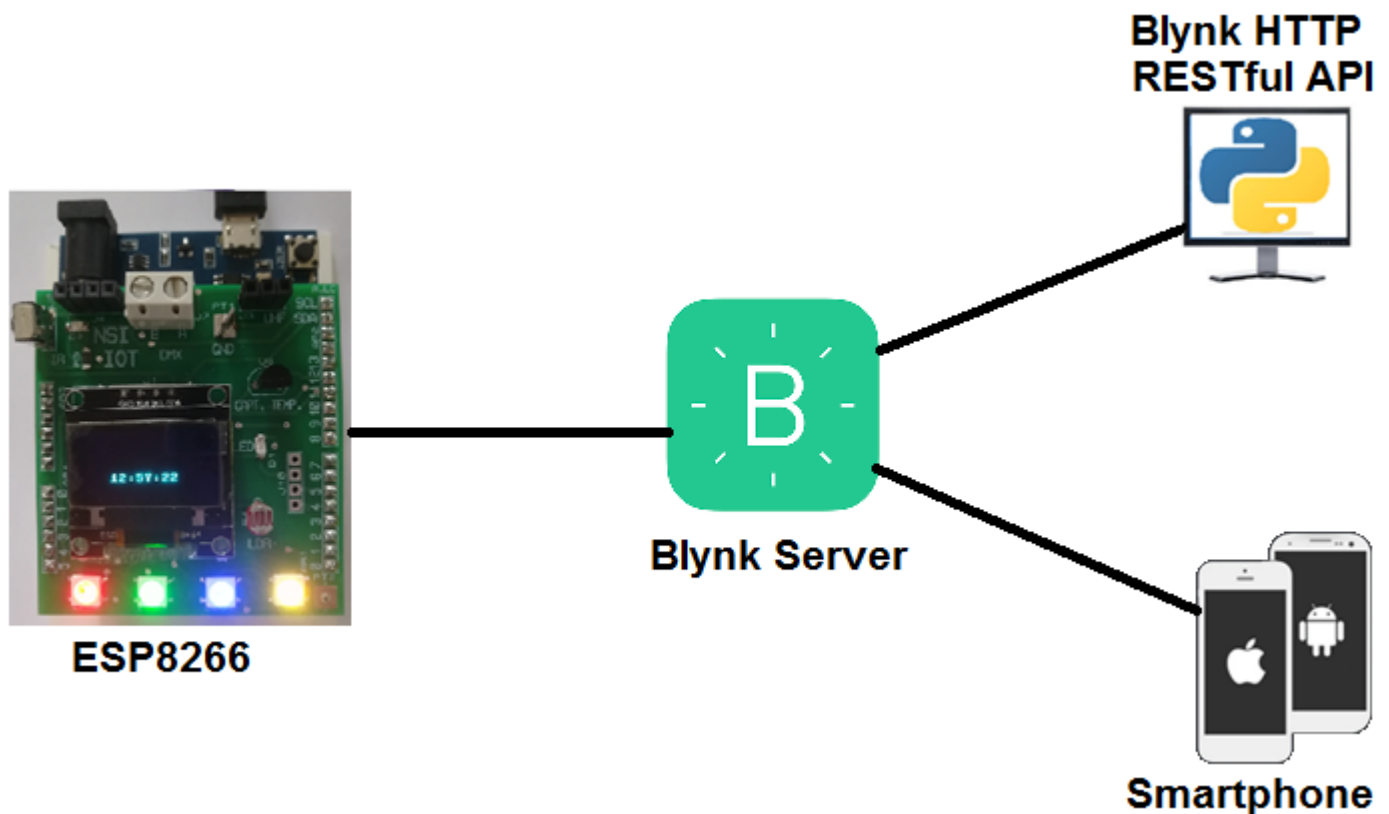
[Blynk](#) est une plate-forme pour l'Internet des Objets (IoT). Cette plateforme permet de concevoir une application mobile (Android et iOS) pour contrôler et visualiser les données d'un système embarqué via un serveur cloud public ou privé.

La conception de l'application mobile (Android et iOS) à base de widgets (éléments graphiques) est réalisée par simple glisser & déposer sans écrire une ligne de code dans le smartphone.

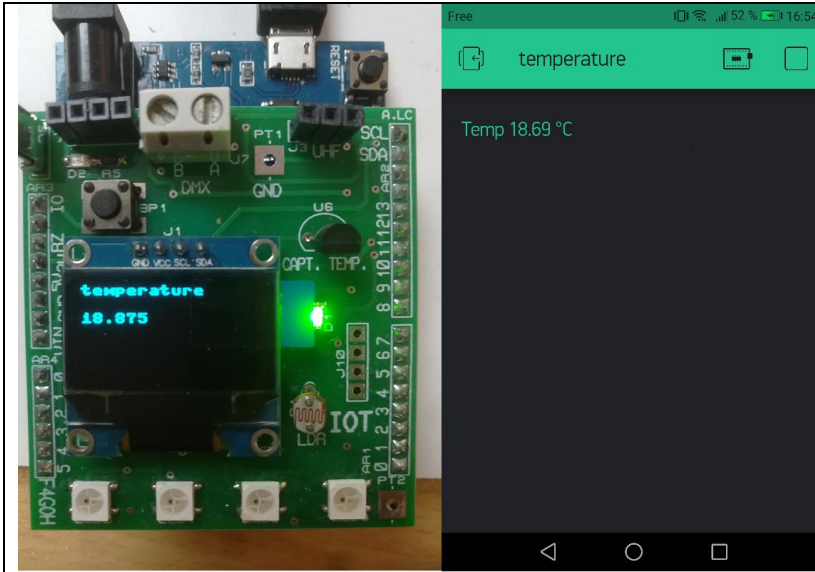
Le logiciel à concevoir est principalement réalisé dans l'objet connecté à l'aide de micro python ou C++ dans l'ESP8266.

Mise en situation :

Les données des capteurs connectés à l'esp8266 sont envoyées au serveur Blynk périodiquement. L'application installée dans un smartphone permet de consulter ces données ou de commander des actionneurs sur l'objet connecté. La communication peut ainsi être bidirectionnelle. De plus une application sur PC utilisant l'API REST (Application Programming Interface, Representational State Transfer) permettra également de consulter les données de IOT.



## 2 Lecture de température



Etape 1 : Le smartphone utilise l'application blynk en configurant ses widgets (ex un label pour afficher une température)

Etape 2 : l'IOT se connecte sur le serveur blynk à l'aide d'un programme et d'une bibliothèque sous micropython

Etape 3 : La Température de l'IOT est envoyée périodiquement au serveur blynk, puis vers le smartphone.

**Q1** A l'aide de l'annexe 1, **Installer** et **configurer** l'application blynk sur le smartphone.

**Q2** A l'aide de l'annexe 2, **Installer** la bibliothèque `blynklib_mp.mpy` dans l'IOT avec l'IDE Thonny.

**Q3** Exécuter le programme suivant dans l'esp8266 avec votre propre SSID, mot de passe et clé API. Valider l'affichage de la température sur le smartphone.

```
import blynklib_mp as blynklib
from machine import Timer
import network
import utime as time
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
from onewire import OneWire
from ds18x20 import DS18X20
from time import sleep_ms
import sys

WIFI_SSID = 'NSI'
WIFI_PASS = 'touchard'
BLYNK_AUTH = '8ttqxxxxxxxxxxxxxxxxxxxxhYOKe4cg'

#Check Wifi Connexion
print("Connecting to WiFi network")
'{}'.format(WIFI_SSID))
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(WIFI_SSID, WIFI_PASS)
while not wifi.isconnected():
    time.sleep(1)
    print('WiFi connect retry ...')
print('WiFi IP:', wifi.ifconfig()[0])

print("Connecting to Blynk server...")
blynk = blynklib.Blynk(BLYNK_AUTH, log=print)

#OLED and ds18s20 init
i2c = I2C(scl=Pin(5), sda=Pin(4))
oled = SSD1306_I2C(128, 64, i2c, 0x3c)
bus = OneWire(Pin(12))
ds = DS18X20(bus)
capteur_temperature = ds.scan()
```

```
#Timer init
tim = Timer(-1)

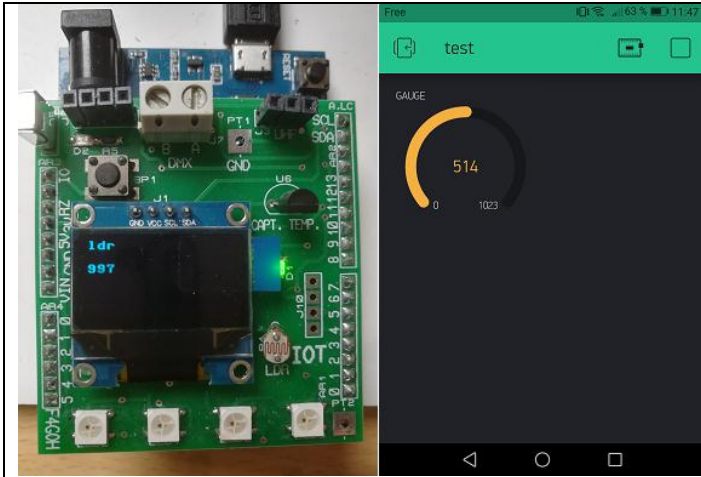
#Virtual Pin definition for Blynk interface
TEMP_VPIN = 5

#Interrupt handler executed every 5 seconds
def handleInterrupt(timer):
    ds.convert_temp()
    sleep_ms(750)
    temp_celsius = ds.read_temp(capteur_temperature[0])
    print("Température : ", temp_celsius)
    oled.fill(0)
    oled.text("temperature", 0, 0)
    oled.text(str(temp_celsius), 0, 20)
    oled.show()
    blynk.virtual_write(TEMP_VPIN, temp_celsius)

#Timer configuration
tim.init(period=5000, mode=Timer.PERIODIC,
callback=handleInterrupt)

while True:
    try:
        blynk.run()
    except KeyboardInterrupt:
        print('Ctrl-C pressed...exiting')
        tim.deinit()
        sys.exit()
```

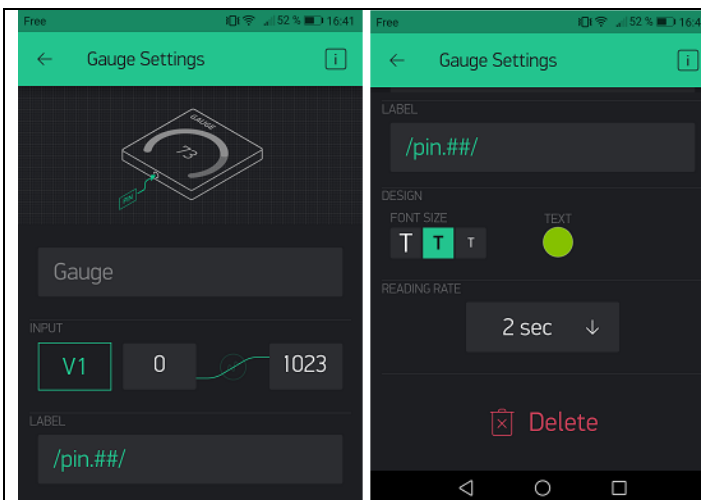
### 3 Détection de luminosité



Etape 1 : Le smartphone utilise l'application blynk avec un widget (ex gauge afficher la luminosité).

Etape 2 : l'IOT se connecte sur le serveur blynk à l'aide d'un programme et d'une bibliothèque sous micropython .

Etape 3 : La luminosité de l'IOT est envoyée périodiquement au serveur blynk, puis vers le smartphone.



Configurer le widget gauge en utilisant la broche virtuelle V1 entre 0 et 1023.

Option de formatage avec deux décimales.

Paramétrer le rafraîchissement (reading rate) à 2 secondes.

#### Options de Formatage

On suppose que le capteur envoie le nombre 12.6789 à l'application Blynk. Le formatage du nombre affiché sur le smartphone peut être configuré :

/pin/ - affiche la valeur sans formatage (12.6789)

/pin./ - affiche la valeur sans la partie décimale (13)

/pin.#/ - affiche la valeur avec une décimale (12.7)

/pin.##/ - affiche la valeur avec deux décimales (12.68)

**Q4** Exécuter le programme suivant dans l'esp8266 avec son propre SSID, mot de passe et clé API. Valider l'affichage de la luminosité sur le smartphone.

<pre>import blynklib as blynklib import network import utime as time from machine import ADC, Pin, I2C from ssd1306 import SSD1306_I2C  WIFI_SSID = 'NSI' WIFI_PASS = 'touchard' BLYNK_AUTH = '8ttqxxxxxxxxxxxxxxxxxxxxhYOKe4cg'  #Check Wifi Connexion print("Connecting to WiFi network") print('{}' .format(WIFI_SSID)) wifi = network.WLAN(network.STA_IF) wifi.active(True) wifi.connect(WIFI_SSID, WIFI_PASS) while not wifi.isconnected():     time.sleep(1)     print('WiFi connect retry ...') print('WiFi IP:', wifi.ifconfig()[0])  print("Connecting to Blynk server...") blynk = blynklib.Blynk(BLYNK_AUTH, log=print)  #OLED and adc init i2c = I2C(scl=Pin(5), sda=Pin(4)) oled = SSD1306_I2C(128, 64, i2c, 0x3c) adc = ADC(0)</pre>	<pre>#Virtual Pin definition for Blynk interface LDR_VPIN = 1  #Define 2 RGB colors LOW_COLOR = '#f5b041' HIGH_COLOR = '#85c1e9'  #This function is called at each smartphone request @blynk.handle_event('read V{}'.format(LDR_VPIN)) def read_handler(vpin):     ldr = adc.read()     oled.fill(0)     oled.text("ldr", 0, 0)     oled.text(str(ldr), 0, 20)     oled.show()     print('ldr value=', ldr)     if ldr &lt; 600:         blynk.set_property(LDR_VPIN, 'color', LOW_COLOR)     else:         blynk.set_property(LDR_VPIN, 'color', HIGH_COLOR)     blynk.virtual_write(vpin, ldr)  while True:     blynk.run()</pre>
---	---

### Remarque :

Dans le code python, si la valeur numérique issue du capteur LDR devient inférieure à 600, la propriété du widget Gauge va changer de couleur. Cela est rendu possible grâce à la méthode `set_property`.

Quelques exemples pour changer les [propriétés d'un Widget](#)

```
#Change la couleur de la LED
blynk.set_property(LDR_VPIN, 'color', '#85c1e9')

#Change le nom du label de la led
blynk.set_property(LDR_VPIN, 'label', 'nouveau nom')

#Change la valeur max du widget gauge
blynk.set_property(LDR_VPIN, 'max', 2000)

#Change la valeur min du widget gauge
blynk.set_property(LDR_VPIN, 'min', 100)
```

Pour les boutons simples

`onLabel` / `offLabel` est une chaîne pour le label ON / OFF du bouton;

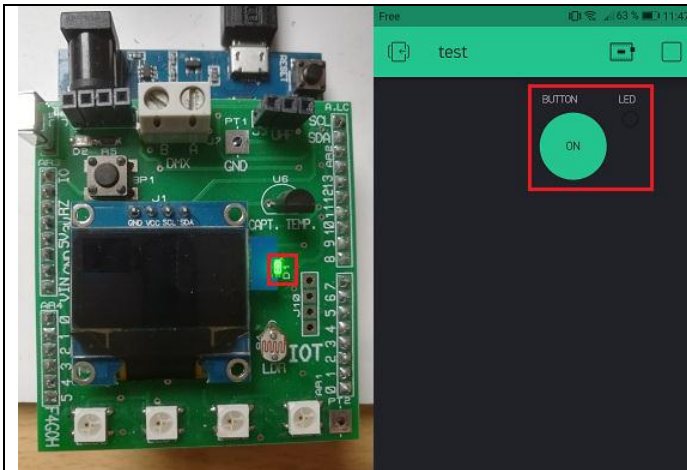
Pour les boutons stylés

`onLabel` / `offLabel` est une chaîne pour le label ON / OFF du bouton;

`onColor` / `offColor` est une chaîne au format HEX pour les couleurs ON / OFF du bouton;

`onBackColor` / `offBackColor` est une chaîne au format HEX pour les couleurs ON / OFF de l'arrière-plan du bouton.

## 4 Commande d'une led.

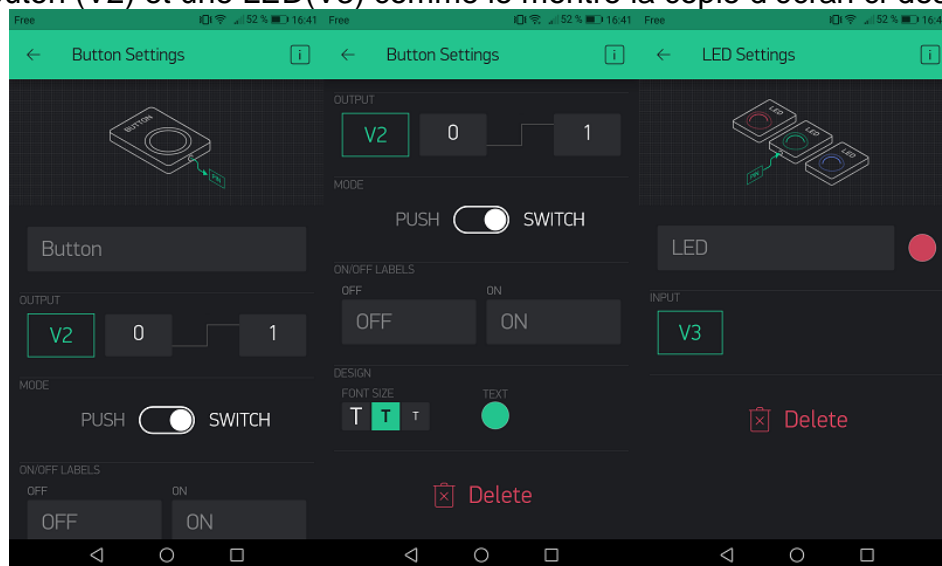


Etape 1 : Le smartphone utilise l'application blynk avec deux widgets (ex bouton et LED).

Etape 2 : l'IOT se connecte sur le serveur blynk à l'aide d'un programme et d'une bibliothèque sous micropython.

Etape 3 : L'utilisateur peut allumer ou éteindre la led D1 de l'IOT avec le smartphone. La commande est également validée par un changement d'état d'une LED virtuelle sur le smartphone.

Configurer un bouton (V2) et une LED(V3) comme le montre la copie d'écran ci-dessous.



**Q5** Exécuter le programme suivant dans l'esp8266 avec son propre SSID, mot de passe et clé API. Valider la commande de la LED sur le smartphone.

```
import blynklib as blynklib
import network
import utime as time
from machine import Pin

WIFI_SSID = 'NSI'
WIFI_PASS = 'touchard'
BLYNK_AUTH = '8ttqxxxxxxxxxxxxxxxxxxhYOKe4cg'

#Check Wifi Connexion
print("Connecting to WiFi network")
'{}'.format(WIFI_SSID))
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(WIFI_SSID, WIFI_PASS)
while not wifi.isconnected():
    time.sleep(1)
    print('WiFi connect retry ...')
print('WiFi IP:', wifi.ifconfig()[0])

print("Connecting to Blynk server...")
blynk = blynklib.Blynk(BLYNK_AUTH, log=print)
```

```
#LED init
led = Pin(0, Pin.OUT)

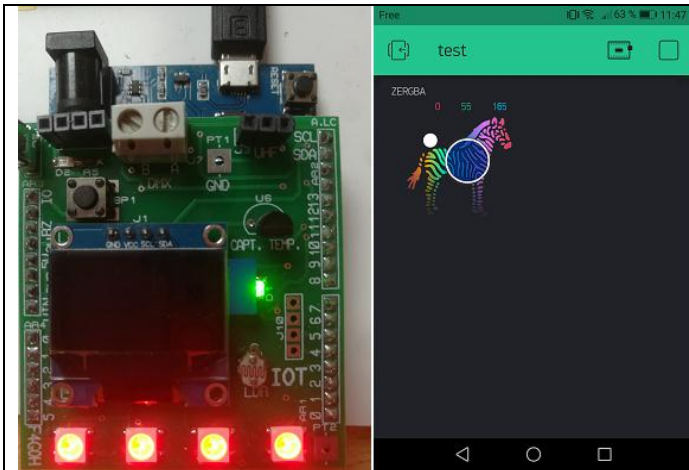
#Virtual Pin definition for Blynk interface
BUTTON_VPIN = 2
LED_VPIN = 3

#This function is called at each smartphone request
@blynk.handle_event('write V{}'.format(BUTTON_VPIN))
def write_virtual_pin_handler(vpin, value):
    print("pin", vpin, "value", value)
    if value[0] == '1':
        led.on()
        blynk.virtual_write(LED_VPIN, 255)
    else:
        led.off()
        blynk.virtual_write(LED_VPIN, 0)

while True:
    blynk.run()
```



## 5 Commande RGB de LED neopixels.

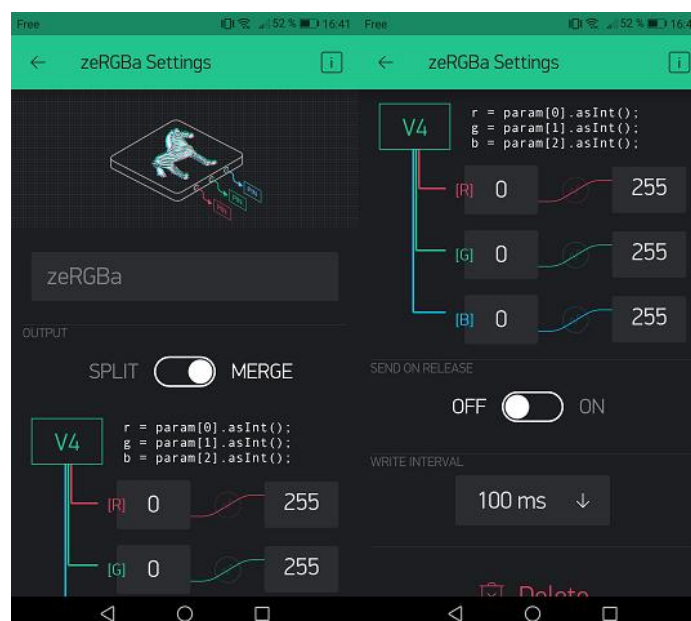


Etape 1 : Le smartphone utilise l'application blynk avec un widget (zeRGBa).

Etape 2 : l'IOT se connecte sur le serveur blynk à l'aide d'un programme et d'une bibliothèque sous micropython.

Etape 3 : L'utilisateur peut contrôler la couleur des 4 LEDs de l'IOT à partir du smartphone.

Configurer le color picker zeRGBa (V4) comme le montre la copie d'écran ci-dessous.



**Q6** Exécuter le programme suivant dans l'esp8266 avec son propre SSID, mot de passe et clé API. Valider la commande des LEDs sur le smartphone.

```
import blynklib as blynklib
import network
import utime as time
from neopixel import NeoPixel
from machine import Pin

# Check Wifi Connexion
WIFI_SSID = 'NSI'
WIFI_PASS = 'touchard'
BLYNK_AUTH = '8ttqxxxxxxxxxxxxxxxxxxxxhYOKe4cg'

print("Connecting to WiFi network")
print('{}' .format(WIFI_SSID))
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(WIFI_SSID, WIFI_PASS)
while not wifi.isconnected():
    time.sleep(1)
    print('WiFi connect retry ...')
print('WiFi IP:', wifi.ifconfig()[0])

print("Connecting to Blynk server...")
blynk = blynklib.Blynk(BLYNK_AUTH, log=print)
```

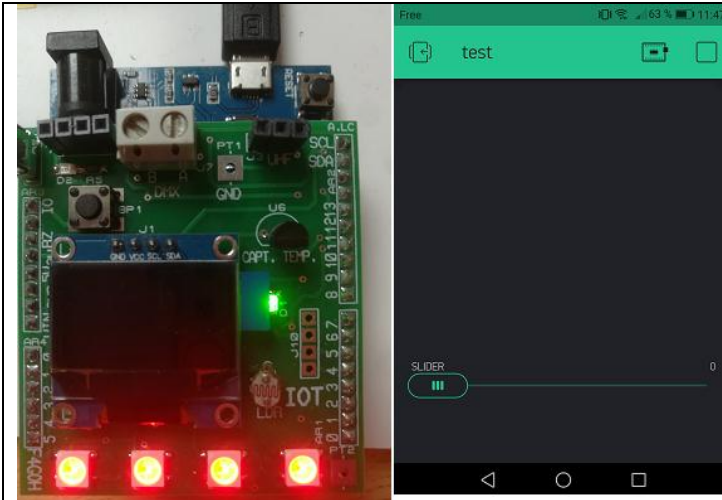
```
# Neopixels init
np = NeoPixel(Pin(14), 4)

# Virtual Pin definition for Blynk interface
RGB_VPIN = 4

# This function is called at each smartphone request
@blynk.handle_event('write V{}'.format(RGB_VPIN))
def write_virtual_pin_handler(vpin, value):
    print("RGB", value)
    for n in range(0, 4):
        np[n] = int(value[0]), int(value[1]),
    int(value[2])
    np.write()

while True:
    blynk.run()
```

## 6 Commande gradateur RGB une couleur.

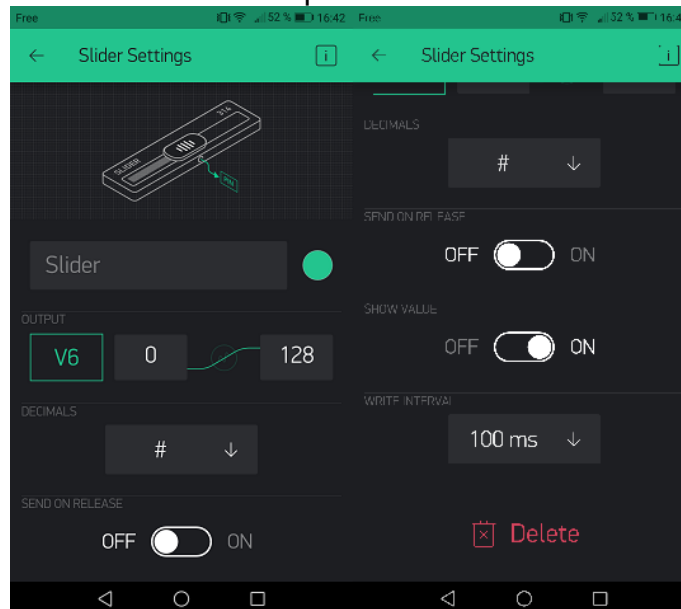


Etape 1 : Le smartphone utilise l'application blynk avec un widget (slider).

Etape 2 : l'IOT se connecte sur le serveur blynk à l'aide d'un programme et d'une bibliothèque sous micropython.

Etape 3 : L'utilisateur peut contrôler le niveau de luminosité avec le slider à partir du smartphone.

Configurer le slider (V6) comme le montre la copie d'écran ci-dessous.



**Q7** Exécuter le programme suivant dans l'esp8266 avec son propre SSID, mot de passe et clé API. Valider la commande de la LED sur le smartphone.

```
import blynklib as blynklib
import network
import utime as time
from neopixel import NeoPixel
from machine import Pin

WIFI_SSID = 'NSI'
WIFI_PASS = 'touchard'
BLYNK_AUTH = '8ttqxxxxxxxxxxxxxxxxxxhYOKe4cg'

# Check Wifi Connexion
print("Connecting to WiFi network")
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(WIFI_SSID, WIFI_PASS)
while not wifi.isconnected():
    time.sleep(1)
    print('WiFi connect retry ...')
print('WiFi IP:', wifi.ifconfig()[0])

print("Connecting to Blynk server...")
blynk = blynklib.Blynk(BLYNK_AUTH, log=print)
```

```
# Neopixels init
np = NeoPixel(Pin(14), 4)

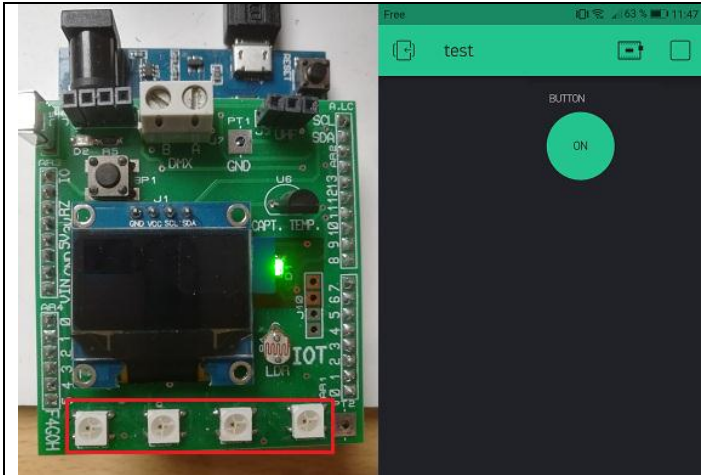
# Virtual Pin definition for Blynk interface
LEVEL_VPIN = 6

def update_neopixels(level):
    for n in range(0, 4):
        color = level, 0, level
        np[n] = color
    np.write()

@blynk.handle_event('write V{}'.format(LEVEL_VPIN))
def write_virtual_pin_handler(vpin, value):
    global level
    print("Level", value[0])
    level = int(value[0])
    update_neopixels(level)

while True:
    blynk.run()
```

## 7 Exemple de mini projet : commande de défilement RGB.



Etape 1 : Le smartphone utilise l'application blynk avec un widget (Button).

Etape 2 : L'IOT se connecte sur le serveur blynk à l'aide d'un programme et d'une bibliothèque sous micropython.

Etape 3 : L'utilisateur commande le défilement des LEDs RGB avec le smartphone.

**Q8** Exécuter le programme suivant dans l'esp8266 avec son propre SSID, mot de passe et clé API. Valider la commande de défilement sur le smartphone.

```
import blynklib as blynklib
import network
import utime as time
from neopixel import NeoPixel
from machine import Pin
from time import sleep_ms

WIFI_SSID = 'NSI'
WIFI_PASS = 'touchard'
BLYNK_AUTH = '8ttqxxxxxxxxxxxxxxxxxxxxhYOKe4cg'

print("Connecting to WiFi network")
'{}'.format(WIFI_SSID))
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(WIFI_SSID, WIFI_PASS)
while not wifi.isconnected():
    time.sleep(1)
    print('WiFi connect retry ...')
print('WiFi IP:', wifi.ifconfig()[0])

print("Connecting to Blynk server...")
blynk = blynklib.Blynk(BLYNK_AUTH, log=print)

# Neopixels init
NBPIXELS = 4
np = NeoPixel(Pin(14), 4)

# Virtual Pin definition for Blynk interface
BUTTON_VPIN = 2
```

```
# leds scrolling state
etat = 0

@blynk.handle_event('write V{}'.format(BUTTON_VPIN))
def write_virtual_pin_handler(vpin, value):
    global etat
    print("pin", vpin, "value", value)
    etat = int(value[0])

def defilement(color):
    for p in range(NBPIXELS):
        np[p] = color
        np[(p - 1) % NBPIXELS] = (0, 0, 0)
        np.write()
        sleep_ms(100)

def stop():
    for p in range(NBPIXELS):
        np[p] = (0, 0, 0)
        np.write()

while True:
    blynk.run()
    if etat == 1:
        defilement((0, 255, 0))
    else:
        stop()
```

**Q7** Utiliser un widget « SuperChart » pour afficher la température.



**Q8** Utiliser 4 boutons pour allumer séparément les LEDs RGB.



**Q9** Utiliser un widget textInput pour afficher un texte sur l'écran OLED.





## 8 Utilisation de l'API REST sur l'ordinateur PC

API REST (Representational State Transfer Application Program Interface) est un style architectural qui permet aux logiciels de communiquer entre eux sur un réseau ou sur un même appareil. Le plus souvent les développeurs utilisent des API REST pour créer des services web. Souvent appelé services web RESTful, REST utilise des méthodes **HTTP** pour récupérer et publier des données entre un périphérique client et un serveur.

En utilisant le protocole **HTTP**, les **API REST** permettent au serveur blynk de communiquer avec les logiciels d'un autre appareil (PC), même s'ils utilisent des systèmes d'exploitation et des architectures différents. Le client (PC) peut demander des ressources avec un langage que le serveur comprend, et le serveur renvoie la ressource avec un langage que le client accepte. Le serveur renvoie la ressource au format JSON (JavaScript Object Notation), XML (Extensible Markup Language) ou texte.

Ainsi, pour dialoguer avec le serveur Blynk avec le protocole http, on utilisera la bibliothèque `requests` de python qui comprend les deux méthodes suivantes :

- GET : pour demander des données au serveur.
- POST : pour soumettre les données à traiter au serveur.

Exemple utilisant la méthode GET appliquée sur la gauge page 3 :

<pre>import requests, json  proxies = {     'http': 'http://172.30.137.29:3128', } # Authentification token BLYNK_AUTH = '8ttqxxxxxxxxxxxxxxxxxxhYOKe4cg'  # URL de base BLYNK_URL = "http://blynk-cloud.com/"  def write(pin,value):     # url de la requete     url = BLYNK_URL + BLYNK_AUTH + "/update/" + pin+     "?value=" + str(value)     print("l'url est",url)     response = requests.get(url)     #response = requests.get(url,proxies=proxies)     print(response)  def property(pin,prop,value):     # url de la requete     url = BLYNK_URL + BLYNK_AUTH + "/update/" + pin+     "?"+prop+"=" + value     print("l'url est",url)     response = requests.get(url)     #response = requests.get(url,proxies=proxies)     print(response)</pre>	<pre>def read(pin):     # url de la requete     url = BLYNK_URL + BLYNK_AUTH + "/get/" + pin     print("l'url est",url)     response = requests.get(url)     #response = requests.get(url,proxies=proxies)     #response qui va être convertit en liste     print(response)     lst = response.json()     print(lst[0])  #lecture de la gauge read("V1")  #mets à jour la gauge V1 à 325 write("V1",325)  #change la couleur de la gauge property("V1","color","%2385c100")</pre>
--	---

Remarque :

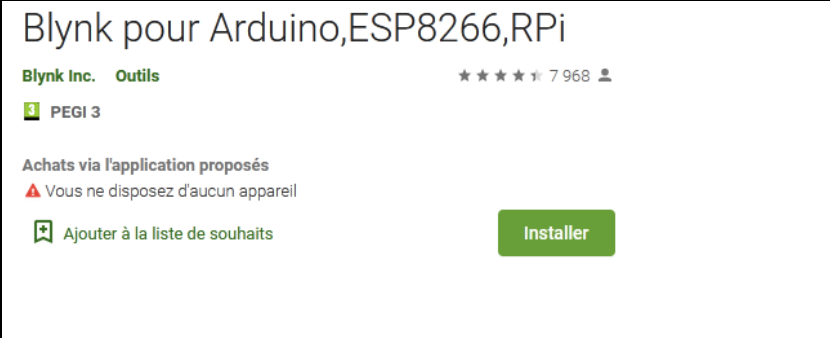
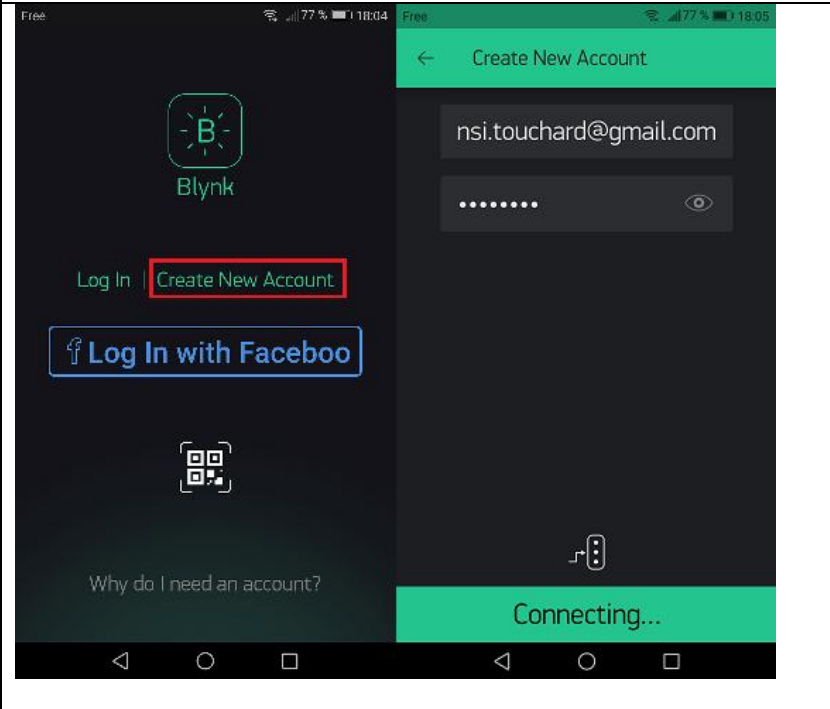
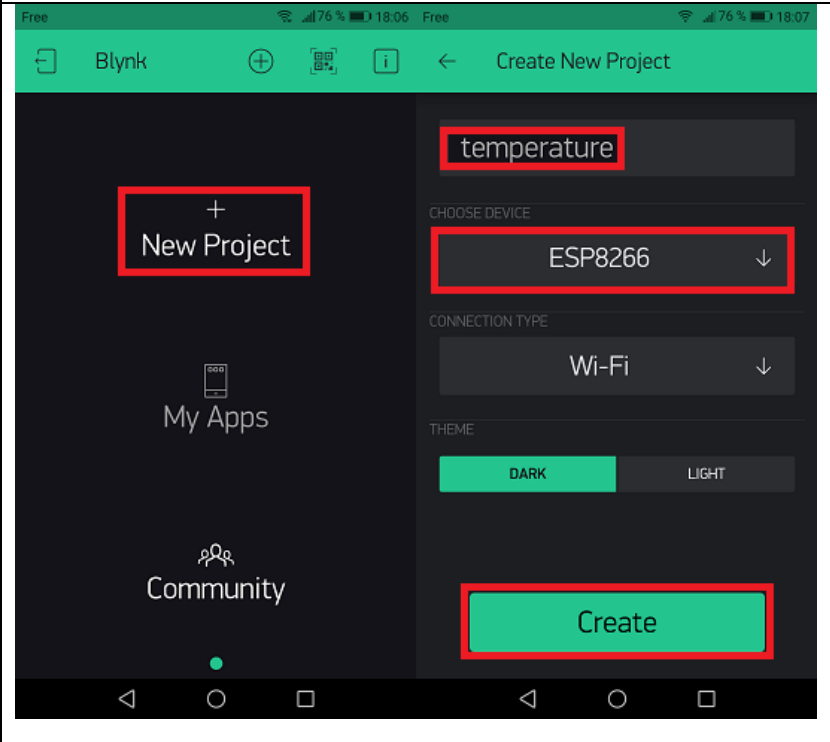
Pour utiliser la bibliothèque `request` à l'école, remplacer la ligne suivante

```
response = requests.get(url)
```

par `response = requests.get(url,proxies=proxies)`

en renseignant les paramètres de proxies.

## Annexe 1 : Configuration initiale de l'application blynk

	<p>Télécharger l'application blynk sur <a href="#">Android</a> ou <a href="#">ios</a></p>
	<p>Créer un nouveau compte</p> <p>Une confirmation par mail est ensuite envoyée</p> <p>Hi there,</p> <p>Welcome to Blynk, a platform to build your next awesome IOT project.</p> <p>Get Started:</p> <ol style="list-style-type: none"> <li>1. Install Blynk Library. <a href="#">Here is</a> a step-by-step guide on how to do that;</li> <li>2. Check <a href="#">Blynk examples</a> for your hardware;</li> </ol>
	<p>Créer un nouveau projet avec un esp8266</p> <p>Nom du projet par exemple « température »</p>

Une fois le projet créé, un mail est automatiquement envoyé avec la clé api (**Auth Token**) qu'il faudra utiliser dans l'application IOT avec micropython

**Auth Token : 8ttqxxxxxxxxxxxxxxxxxxxxhY0Ke4cg**

Happy Blynking!

-

Getting Started Guide -> <https://www.blynk.cc/getting-started>

Documentation -> <http://docs.blynk.cc/>

Sketch generator -> <https://examples.blynk.cc/>

Latest Blynk library -> [https://github.com/blynk/blynk-library/releases/download/v0.6.1/Blynk\\_Release\\_v0.6.1.zip](https://github.com/blynk/blynk-library/releases/download/v0.6.1/Blynk_Release_v0.6.1.zip)

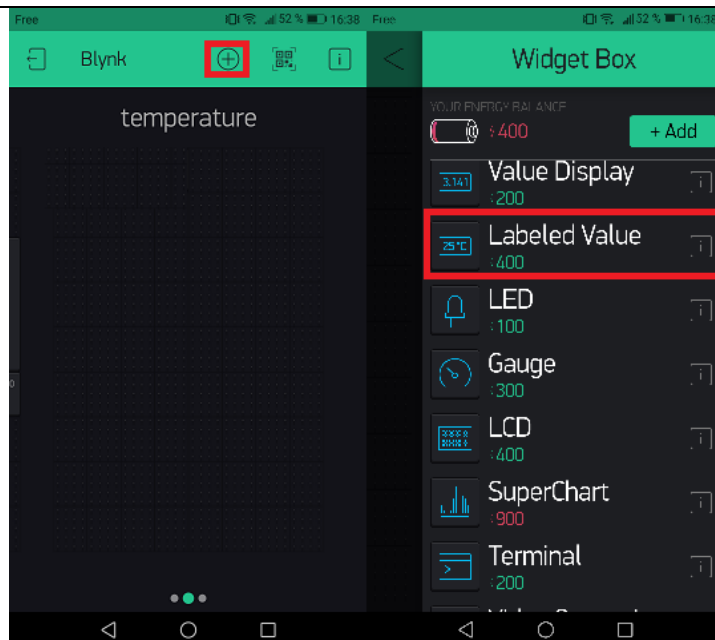
Latest Blynk server -> <https://github.com/blynk/blynk-server/releases/download/v0.41.13/server-0.41.13.jar>

-

<https://www.blynk.cc>

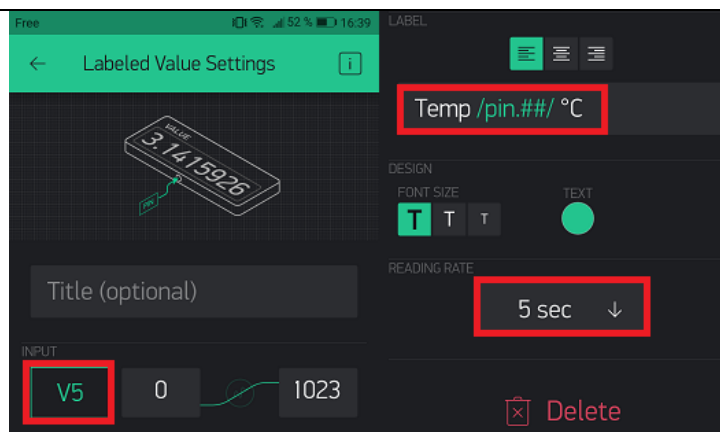
[twitter.com/blynk\\_app](https://twitter.com/blynk_app)

[www.facebook.com/blynkapp](https://www.facebook.com/blynkapp)



Dans le projet nouvellement créé, ajouter un label (Labeled Value)

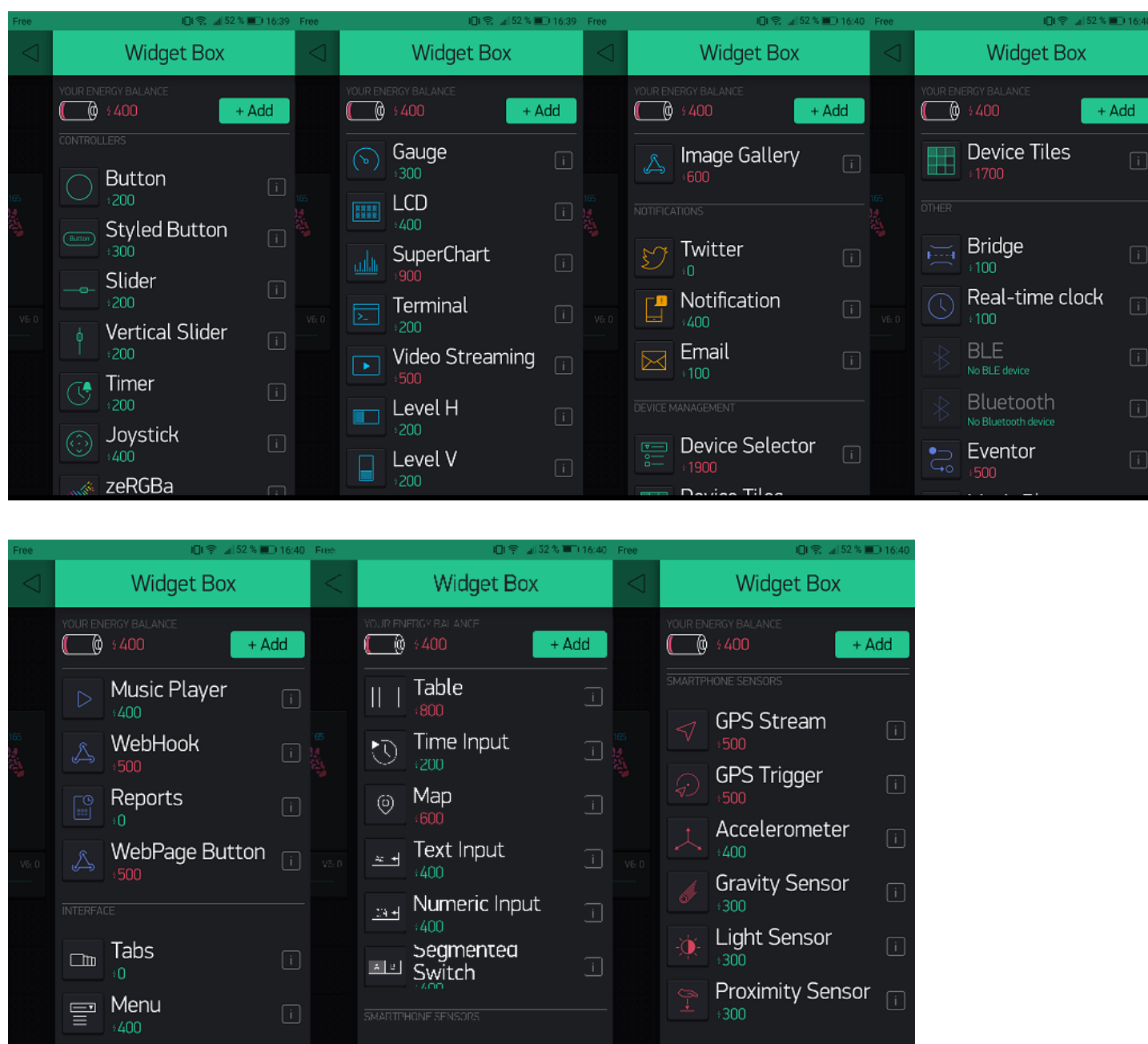
Puis cliquer brièvement sur le label pour accéder à sa configuration.



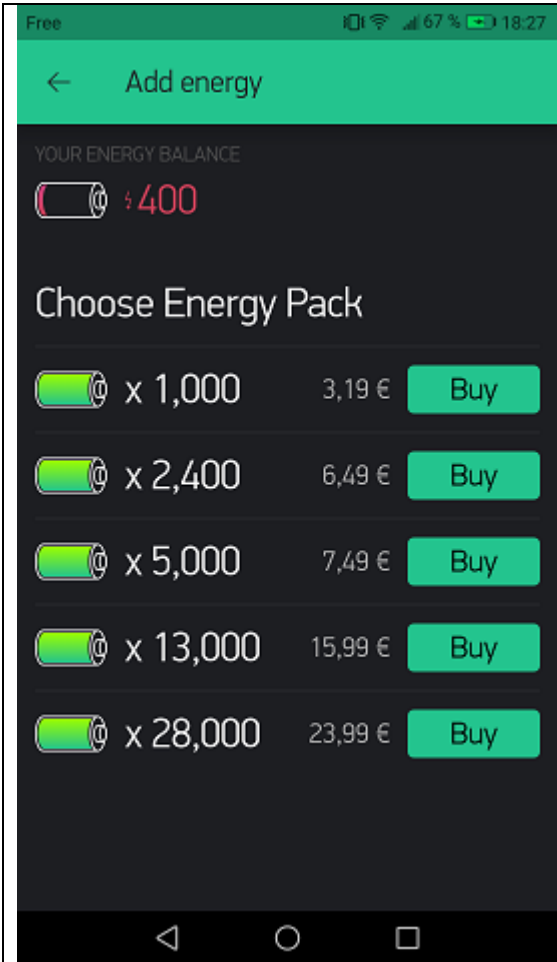
Paramétrer le label comme le montre la copie d'écran ci-contre.

	
<p>Pour exécuter l'application cliquer sur le <b>triangle</b>.</p>	<p>Pour stopper l'application, cliquer sur le <b>carré</b>. Une notification en rouge apparait si l'IOT n'est pas connecté au serveur.</p>


### Liste des widgets disponibles



Remarque :



YOUR ENERGY BALANCE


:400

+ Add

Une exploitation commerciale n'est pas possible avec le plan Prototype. D'autre part, Blynk utilise un système d'énergie (Blynk.Energy) pour limiter l'usage de ses fonctionnalités. Au départ, on dispose de 2000 points d'énergie par compte que l'on dépense pour les widgets d'interface utilisateur et autres fonctionnalités Blynk. Lorsque l'on supprime des widgets, l'énergie dépensée est entièrement restituée . Mais avec certaines fonctionnalités (le partage par exemple), ce n'est pas le cas.

Attention : Partager coûte 1000 d'énergie et cette énergie n'est pas récupérable, même si aucun partage n'est effectué.

Liens utiles :

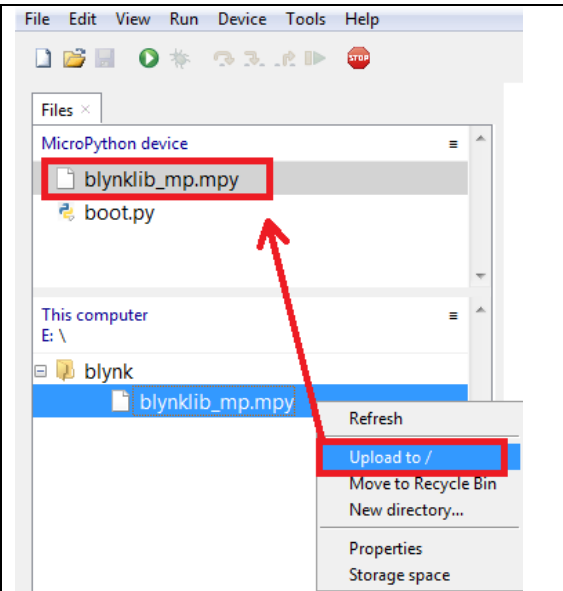
- L'API [RESTful HTTP](#) vous permet de lire et écrire facilement les valeurs des broches dans les applications et hardwares connectés à Blynk.
- [Tutoriel1 en français](#)
- [Tutoriel2 en français](#)



## Annexe 2 : Installation de la bibliothèque `blynklib_mp.mpy` dans l'IOT

		<p>Avec l'IDE Thonny, activer le menu « Files »</p> <p>Si le menu n'apparaît pas, effectuer une mise à jour du logiciel Thonny</p>
--	--	--

<pre>MicroPython v1.13 on 2020-09-11; ESP module with ESP8266  Type "help()" for more information. &gt;&gt;&gt;</pre>	<p>Vérifier la version du firmware <b>v1.13</b> dans l'esp8266.</p> <p>Rappel : Touches <b>CTRL+F2</b> pour se connecter à l'IOT.</p>
---	---

	<p>Effectuer une copie du fichier <code>blynklib_mp.mpy</code> dans l'ESP8266 en cliquant sur le fichier avec le bouton droit puis <b>Upload to /</b></p> <p>Vérifier l'utilisation de la bibliothèque dans la console :</p> <pre>&gt;&gt;&gt; import blynklib_mp as blynklib &gt;&gt;&gt; dir(blynklib) ['_class_', '__name__', 'const', 'select', 'sleep_ms', 'socket', 'struct', 'ticks_ms', 'time', '__version__', 'IOError', 'LOGO', 'stub_log', 'BlynkError', 'RedirectError', 'Protocol', 'Connection', 'Blynk'] &gt;&gt;&gt;</pre>
---	--

### Remarque :

Un fichier `.mpy` est un format de fichier binaire contenant du code précompilé et qui peut être importé comme un module `.py` normal. Ces fichiers `.mpy` contiennent du [bytecode](#) qui est généré à partir de fichiers source Python (fichiers `.py`) via le programme [mpy-cross](#). Cela permet d'optimiser l'espace mémoire Flash de l'ESP8266 et la bibliothèque sera exécutée plus rapidement.

La ligne de commande pour générer un fichier `.mpy` pour un esp8266 est :

```
mpy-cross blynklib_mp.py -march=xtensa
```

`mpy-cross` nom de l'utilitaire de pré compilation,  
`blynklib_mp.py` Fichier python à précompiler,  
`-march=xtensa`, option pour préciser l'architecture de destination (ESP8266).