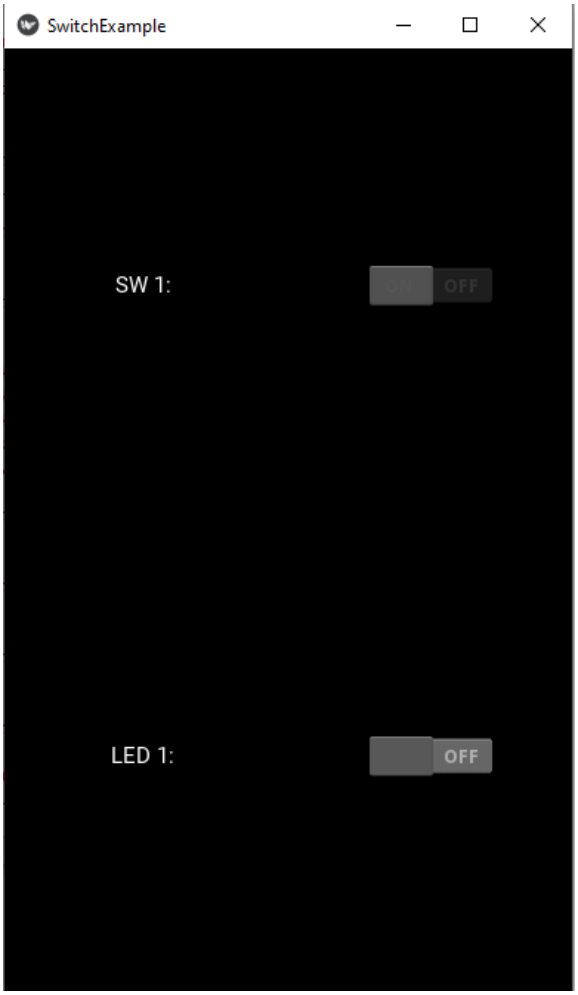
Kivy App – Communication with a Database

1. Using **Kivy**, we can make an app that will allow us to communicate with the database we have on 000webhost. We will make a button that can be used to turn the **RPi led on or off**. The Kivy app **reads the status of the button and updates the status of the led on the database**. Also, we will make another button that **reads the status of SW1 on the database and updates its status on the Kivy app.** Our app should look like this (The SW1 button is not clickable, since we want it to only read the database):



2. Create a .py file with the following content. Here we are simply importing every package we will be using for this app. We of course use **Kivy** and some components of it. The Clock component allows us to schedule events every couple of seconds. We will use this component to constantly read/write the database. Additionally we will use requests to use JSON, which acts as the middleman between the app and the database:

```
import kivy
from kivy.app import App
from kivy.uix.switch import Switch
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.clock import Clock
from functools import partial
import time
import requests
```

3. Add the following content to your **.py** file. We are making a class called *SwitchContainer* which uses the *GridLayout* module. This will allow us to organize our two buttons on the screen. Also we are creating two labels and two switches, one that corresponds to **SW1** and another for **LED1**:

```
import kivy
from kivy.app import App
from kivy.uix.switch import Switch
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.clock import Clock
from functools import partial
import time
import requests
```

```
class SwitchContainer(GridLayout): #Create a class that uses the GridLayout module
    def __init__(self, **kwargs):

        super(SwitchContainer, self).__init__(**kwargs)
        self.cols = 2

        #
        #Switch1
        #

        #switch label
        self.add_widget(Label(text="SW 1: "))

        #switch1 button
        self.sw1 = Switch(active=False)
        self.add_widget(self.sw1)
        self.sw1.disabled = True  #Make the switch unclickable on the app



        #
        #Led1
        #

        #led label
        self.add_widget(Label(text="LED 1: ")) #Create a label that displays "LED 1"

        #led1 button
        self.led1 = Switch(active=False)
        self.add_widget(self.led1) #Create a switch that can be turned off or on
```

4. We need to *constantly read and update values from the database*. To do this, we use the Clock package to **schedule a function execution every second.** Add the following code, we use partial so that we can specify our own parameters in the function being called. The function being called is self.JSONrequest, which will be defined in the next section:

```
import kivy
from kivy.app import App
from kivy.uix.switch import Switch
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.clock import Clock
from functools import partial
import time
import requests


class SwitchContainer(GridLayout): #Create a class that uses the GridLayout module
    def __init__(self, **kwargs):

        super(SwitchContainer, self).__init__(**kwargs)
        self.cols = 2

        #
        #Switch1
        #

        #switch label
        self.add_widget(Label(text="SW 1: "))
```

```
        #switch1 button
        self.sw1 = Switch(active=False)
        self.add_widget(self.sw1)
        self.sw1.disabled = True  #Make the switch unclickable on the app



        #
        #Led1
        #


        #led label
        self.add_widget(Label(text="LED 1: ")) #Create a label that displays "LED 1"


        #led1 button
        self.led1 = Switch(active=False)
        self.add_widget(self.led1) #Create a switch that can be turned off or on


        #schedule the JSONrequest function to trigger every second to read/write database
        event = Clock.schedule_interval(partial(self.JSONrequest), 1)
```

5. We've scheduled the function to execute every second, however, we haven't defined the function itself. Add the following code to define self.JSONrequest. Here, we are reading the status of the switches on the app and converting those values to integers **(True = 1, False = 0).** Then we send the JSONrequest with those values and update the database (or the app) accordingly**. Make sure to change the highlighted code "yourdomain" to your real domain.** (Note: If an error occurs, make sure that the indentation for the function self.JSONrequest is correct, it should line up with the def __init__ function.)

```
import kivy
from kivy.app import App
from kivy.uix.switch import Switch
```

```
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.clock import Clock
from functools import partial
import time
import requests

class SwitchContainer(GridLayout): #Create a class that uses the GridLayout module
    def __init__(self, **kwargs):

        super(SwitchContainer, self).__init__(**kwargs)
        self.cols = 2

        #
        #Switch1
        #

        #switch label
        self.add_widget(Label(text="SW 1: "))

        #switch1 button
        self.sw1 = Switch(active=False)
        self.add_widget(self.sw1)
        self.sw1.disabled = True  #Make the switch unclickable on the app



        #
        #Led1
        #

        #led label
```

```
        self.add_widget(Label(text="LED 1: ")) #Create a label that displays "LED 1"

        #led1 button
        self.led1 = Switch(active=False)
        self.add_widget(self.led1) #Create a switch that can be turned off or on

        #schedule the JSONrequest function to trigger every second to read/write database
        event = Clock.schedule_interval(partial(self.JSONrequest), 1)

#Make sure this following function's indentation matches with the def __init__ function above
def JSONrequest(self, *largs):
    #Get the sw1 active status and convert it to an integer
    if (self.sw1.active == True):
        SW1 = 1
    else:
        SW1 = 0

    #Get the led1 active status and convert it to an integer
    if (self.led1.active == True):
        LED1 = 1
    else:
        LED1 = 0



    #json request
    data = {'username': 'ben','password':'benpass', 'SW1':SW1, 'LED1': LED1}
    res = requests.post("https://yourdomain.000webhostapp.com/scripts/sync_app_data.php", json=data)
    r = res.json()
```

```
        #If the app sw1 doesn't match the DB sw1, change it on the app.
        if SW1 != r['SW1']:
            print("Changing SW1 status to the value in the database.")
            if self.sw1.active == True:
                self.sw1.active = False
            else:
                self.sw1.active = True
        else:
            return
```

6. We must build and run our app by adding the following code (Note: There should be no indentation on this added code):

```
import kivy
from kivy.app import App
from kivy.uix.switch import Switch
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.clock import Clock
from functools import partial
import time
import requests


class SwitchContainer(GridLayout): #Create a class that uses the GridLayout module
    def __init__(self, **kwargs):
```

```
super(SwitchContainer, self).__init__(**kwargs)
self.cols = 2

#
#Switch1
#

#switch label
self.add_widget(Label(text="SW 1: "))

#switch1 button
self.sw1 = Switch(active=False)
self.add_widget(self.sw1)
self.sw1.disabled = True  #Make the switch unclickable on the app



#
#Led1
#

#led label
self.add_widget(Label(text="LED 1: ")) #Create a label that displays "LED 1"

#led1 button
self.led1 = Switch(active=False)
self.add_widget(self.led1) #Create a switch that can be turned off or on

#schedule the JSONrequest function to trigger every second to read/write database
event = Clock.schedule_interval(partial(self.JSONrequest), 1)
```

```
def JSONrequest(self, *largs):
    #Get the sw1 active status and convert it to an integer
    if (self.sw1.active == True):
        SW1 = 1
    else:
        SW1 = 0

    #Get the led1 active status and convert it to an integer
    if (self.led1.active == True):
        LED1 = 1
    else:
        LED1 = 0




    #json request
    data = {'username': 'ben','password':'benpass', 'SW1':SW1, 'LED1': LED1}
    res = requests.post("https://yourdomain.000webhostapp.com/scripts/sync_app_data.php", json=data)
    r = res.json()



    #If the app sw1 doesn't match the DB sw1, change it on the app.
    if SW1 != r['SW1']:
        print("Changing SW1 status to the value in the database.")
        if self.sw1.active == True:
            self.sw1.active = False
        else:
            self.sw1.active = True
    else:
        return
```

```
class SwitchExample(App):
    def build(self):
        return SwitchContainer()


if __name__ == '__main__':
    SwitchExample().run()
```

7. Our app is now complete; however, we need to make sure our server side is ready to handle the requests from our app. **The app expects a table named 'device' with the following information**:

devnum	devtype	devname	ctrl	status
1	INPUT	SW1	RPI	0
2	OUTPUT	LED1	ANDROID	0

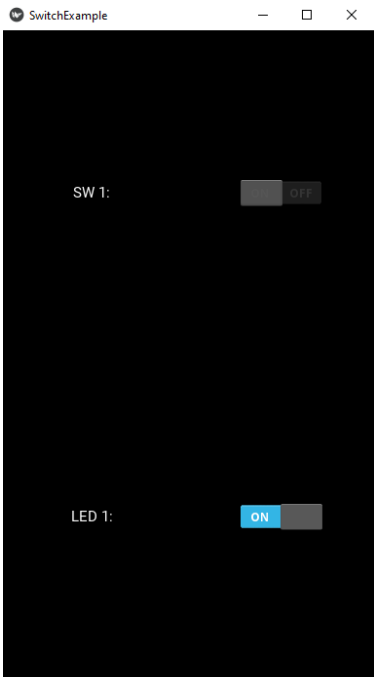
8. The app also expects the following **sync_app_data.php** script under ***public_html > scripts***.

```
<?php
require_once __DIR__ . '/../../required/db_connect.php';
$input = file_get_contents("php://input");
$error=0; $out_json = array(); $out_json['success'] = 1; //assume success
$SW1_status=0; $LED1_status=0;
if ($input) {
$json = json_decode($input, true); //check if it json input
if (json_last_error() == JSON_ERROR_NONE) {
if (isset($json["username"]) && isset($json["password"]) && isset($json["SW1"])
&& isset($json["LED1"])) {
```

```
$in_username = $json["username"];
$in_password = $json["password"]; //if the expected fields are not null, get them
$in_SW1 = $json["SW1"];
$in_LED1 = $json["LED1"];
if ($stmt=$mysqli->prepare("SELECT password FROM webuser WHERE pname = ? LIMIT 1")) {
$stmt->bind_param('s', $in_username);
$stmt->execute(); $stmt->store_result(); //store_result to get num_rows etc.
$stmt->bind_result($db_password); //get the hashed password
$stmt->fetch();
if ($stmt->num_rows == 1) { //if user exists, verify the password
if (password_verify($in_password, $db_password)) {
$stmt->close();
if ($stmt = $mysqli->prepare("UPDATE device set status=?
where devname = 'LED1'")) { //update LED1
$stmt->bind_param('i', $in_LED1); $stmt->execute();
} else {$error=1;}
$stmt->close();
if (!$error && ($stmt = $mysqli->prepare("SELECT status FROM device
where devname = 'SW1'"))) { //read SW1
$stmt->execute(); $stmt->bind_result($SW1_status); $stmt->fetch();
} else {$error=2;}
$stmt->close();
if (!$error && ($stmt = $mysqli->prepare("SELECT status FROM device
where devname = 'LED1'"))) { //read LED1
$stmt->execute(); $stmt->bind_result($LED1_status); $stmt->fetch();
} else {$error=3;}
$stmt->close();
} else {$error=4;}
} else {$error=5;}
} else {$error=6;}
} else {$error=7;}
```

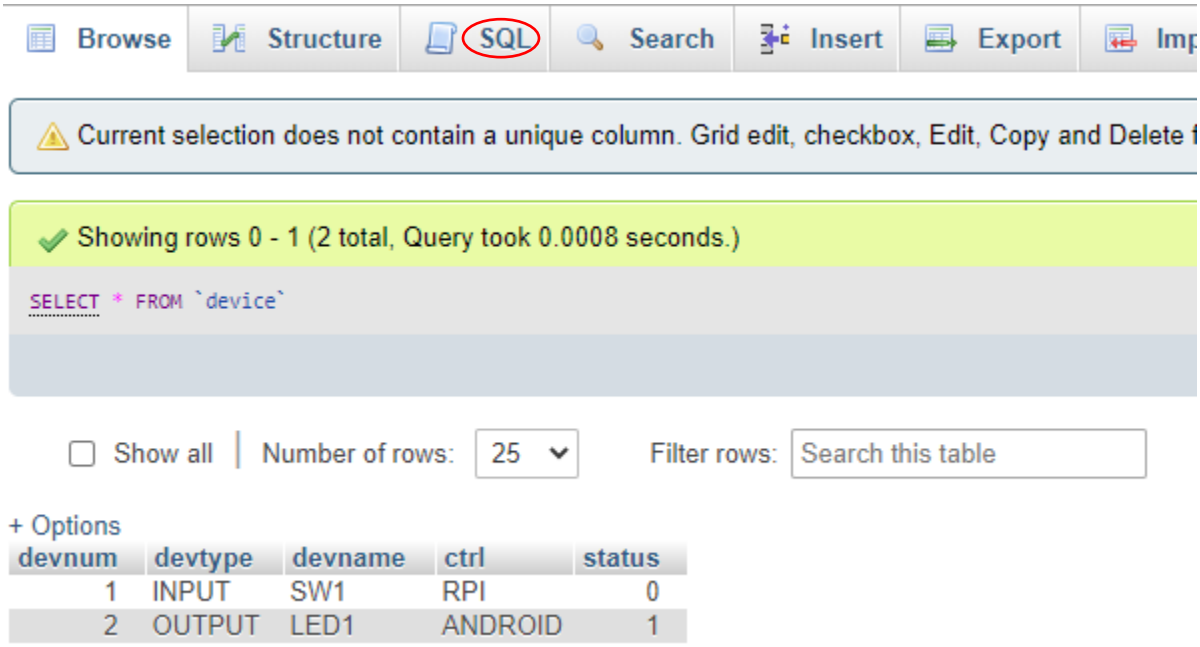
```
} else {$error=8;}

} else {$error=9;}

if ($error){

$out_json['success'] = 0; //flag failure

}

$out_json['SW1'] = $SW1_status; $out_json['LED1'] = $LED1_status;

$out_json['error'] = $error; //provide error (if any) number for debugging

echo json_encode($out_json); //encode the data in json format

?>
```

9. Test the app by running it and switching on the LED1 button. Then, refresh your database, the value should be updated.



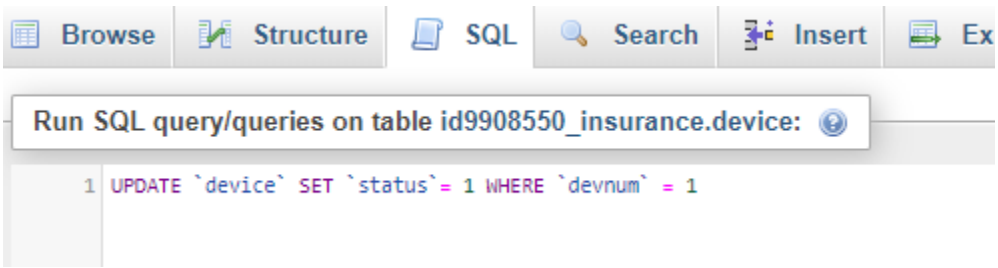
devnum	devtype	devname	ctrl	status
1	INPUT	SW1	RPI	0
2	OUTPUT	LED1	ANDROID	1

10. You can also change the value of SW1 on the database by going **to 000webhost > your website > Tools > Database Manager**. Then click on **Manage > phpMyAdmin**. Click on your 'device' table, then click on the SQL tab.

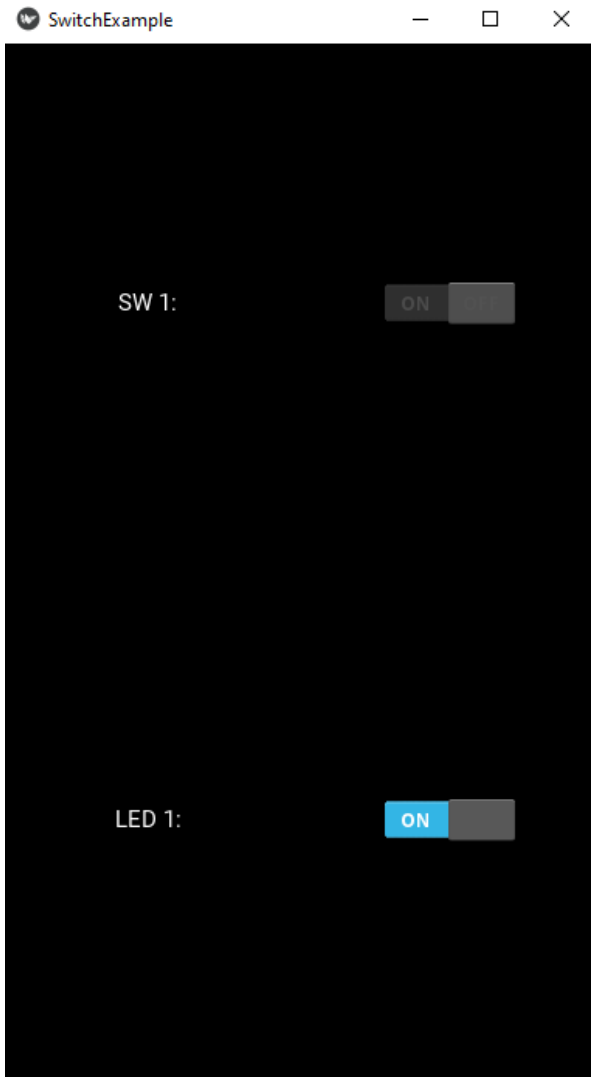


11. In the SQL tab, input the following query and click on **'GO'**:

```
UPDATE `device` SET `status`= 1 WHERE `devnum` = 1
```



12. Now check your app. The **SW1** button should be on the **ON** position as shown below:



devnum	devtype	devname	ctrl	status
1	INPUT	SW1	RPI	1
2	OUTPUT	LED1	ANDROID	1