CSCI 3342/6312 – Tutorial/Assignment #7 **Deadline: 3:00 pm on June 25, 2020**

– Quiz #7 (Separate *Container*) **Deadline: 3:00 pm on June 26, 2020**

Completion: Required **Submission: Required**

Last Name:_____ First Name:_____

This *tutorial* is to familiarize you with the Kivy code to enable Android App to interact with the DBOR and RPi.

1) How and what to submit?

Submit the following (upload in Blackboard to the available container) in "one" PDF document (not in docx or any other format):

i) The certification page (see next page) should be the first page, followed by
ii) your solution to the problems given on this assignment.

One way is to copy the certification page and your solution to the problems into a Word document and then save the Word document as PDF, and upload the PDF version (not docx version). Only the PDF version will be graded.

2) Only ONE upload attempt is allowed: Before submitting a document through Blackboard, you should review the document being uploaded to make sure that you are uploading the correct document (e.g. do not upload the assignment belonging to another course). To help you prevent uploading wrong documents, notes (titled "HelpOnSubmissionThroughBlackboard" on how to save & review drafts before final submission have been uploaded under Reference Material folder.

Certification Page

This page must be the first page of your uploaded document.
Your assignment will not be graded without this page (completed with your full name in the area provided) as the first page of your uploaded document.

I, _____, certify that the work I am uploading represents my own efforts, and is not copied from anyone else or any other resource (such as Internet). *Furthermore, I certify that I have not let anyone copy from my work.*

Tutorial Portion

Learning Objectives!

This tutorial is to familiarize you with the Kivy code to enable Android App to interact with the DBOR and, hence, with RPi.

Below are some references to learn development of Apps using Kivy:

https://www.youtube.com/watch?v=QUHnJrFouv8

https://www.youtube.com/watch?v=fGWHQA3LhJ8

https://www.youtube.com/watch?v=AS3b70pLYEU

https://www.youtube.com/watch?v=H8aWfu9aICc

https://www.youtube.com/watch?v=lnOTAq4NQfQ

In the previous tutorial you learned how to make a ***structure***/façade on the Android screen. In this tutorial, you will learn to add ***behavior*** to the ***structure***. Specifically, when you change the position of a designated *SwitchCompat* (e.g. LED1), it should change the respective value in DBOR. Also, if the value of a designated key/sensor (e.g. SW1) in DBOR changes, it should change the status of the respective *SwitchCompat* on the Android screen.

Below is a summary of the sections in this tutorial:

Sections 1 through 6: Building the intended Android App in stages using Kivy.

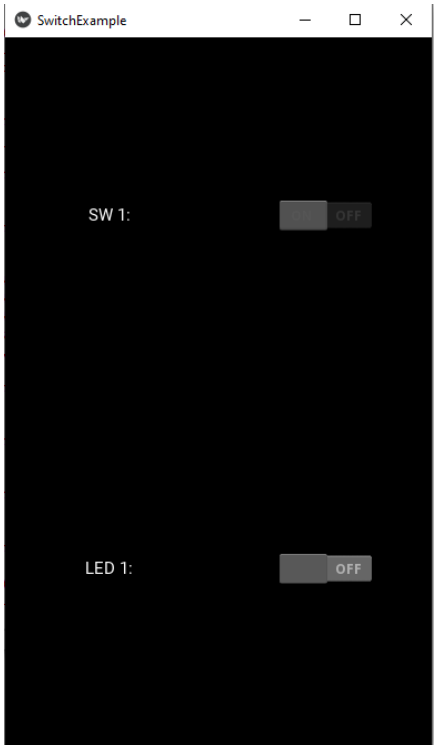
Sections 7: Installing the PHP script on *000webhost.com* to service the JSON requests from the Android App.

Section 8 and 9: Making the Android App interact with the DBOR ***device*** table.

Section 10: Making the Android App interact with RPi through the DBOR ***device*** table.

We will be using Windows based machine for this tutorial.

1. Using **Kivy**, we can make an app that enables an App-user to communicate with a database such as our DBOR in PaaS (*000webhost.com*). We will make a button (*SwitchCompat*) that can be used to turn the corresponding **RPi LED on or off**. The Kivy app **reads the status of the button and updates the status of the corresponding LED in the database**. Also, we will make another button (*SwitchCompat*) that **reads the status of SW1 from the database and updates its status on the Android app.** Our app should look as shown in the following structure/façade of Android screen (*note that SW1 button is not clickable, since we want it to only represent the status of the corresponding key or entity in the database*):



2. Create a .py file with the content given below which imports the required packages for this app. We, of course, use **Kivy** and some modules of it. ***Clock*** module allows us to schedule events every couple of seconds. We will use this component to periodically read/write from/to our DBOR. Additionally, we will use ***requests*** module to use *JSONRequest/JSONResponse* to interact with the DBOR:

```
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 new content shown below to the existing .**py** file that you have started building. In the new portion of the code, we are making a class called *SwitchContainer* which uses *GridLayout* module. This allows us to organize our two buttons (*SwitchCompats*) on the screen. Also, we are creating *two labels* and *two switches*, one corresponding to **SW1** and another for **LED1**:

```
class SwitchContainer(GridLayout):                  #create a class that uses the GridLayout module
    def __init__(self, **kwargs):
        super(SwitchContainer, self).__init__(**kwargs)
        self.cols = 2

        self.add_widget(Label(text="SW 1: "))                  #create a label for SW1
        self.sw1 = Switch(active=False)     #create a SwitchCompat for SW1 (default to OFF)
        self.add_widget(self.sw1)                  #add the created SwitchCompat to the screen
        self.sw1.disabled = True            #make SW1 unclickable on the app


        self.add_widget(Label(text="LED 1: "))                 #create a label for LED1
        self.led1 = Switch(active=False)    #create a SwitchCompat for LED1 (default to OFF)
        self.add_widget(self.led1)                 #add the created SwitchCompat to the screen
        self.led1.disabled = False          #by default a created SwitchCompat is clickable; so, there is no need
                                            #for this statement
```

4. We need to *periodically synchronize values with the DBOR*. To do this, we use the *Clock* module to **schedule the execution of the relevant function every *n* seconds** (*use a reasonable value for n in order not to offend 000webhost.com which may think it is a hacking event*). Add the new code below to the existing **.py** file that we are building. Note that we use *partial* so that we can specify our own parameters in the function being called. The function being called is *JSONrequest*, which will be defined in the next section. Note that *self* (referred to as *this* or *me* in other OOP languages) points to the current or local object:

```
#schedule the JSONrequest function to trigger every 5 seconds to read/write database
event = Clock.schedule_interval(partial(self.JSONrequest), 5)
```

5. Recall that in the previous section we had scheduled the function, *JSONrequest*, to be executed every 5 seconds. In this section, we will define *JSONrequest* function. Add the code given below in blue, making sure that "def JSONrequest (self, *largs)" is indented in sync with the earlier defined "def __init__(self, **kwargs)", otherwise, you may encounter errors. In JSONrequest function defined below, we are reading the status of the SwitchCompats on the app and converting the status to the respective integer values **(True = 1, False = 0)**. We, then, compile the payload (dictionary) for the *request* and send the *request* to our URL (*000webhost.com*) where *sync_app_data.php* script is triggered (PHP script is provided in a *later* section).

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

    def JSONrequest(self, *largs):
        if (self.sw1.active == True):              #Get the sw1 active status and convert it to an integer
            SW1 = 1
        else:
            SW1 = 0
        if (self.led1.active == True):             #Get the led1 active status and convert it to an integer
            LED1 = 1
        else:
            LED1 = 0
                                                   #below are json request payload, the request itself, and the response
        data = {'username': 'ben','password':'benpass', 'SW1':SW1, 'LED1': LED1}   #json request payload
        res = requests.post("https://yourdomain.000webhostapp.com/scripts/sync_app_data.php", json=data)
        r = res.json()          #json response

        if SW1 != r['SW1']:               #check the received value of SW1 & change it on the App if there is a mismatch
            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 need to add code for *building* and *running* our app (but ***don't run the app*** yet since we need to put the PHP script that the JSON request triggers on the server side to interact with our app). Add the code below to the ***.py*** file we are currently building (*note that there should be no indentation on this added code*):

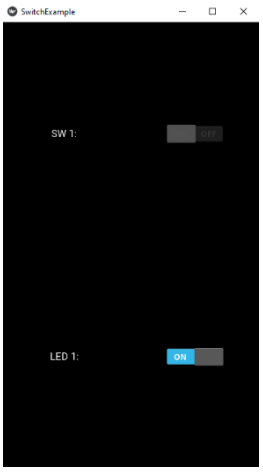
```
class SwitchExample(App):
    def build(self):                    #build
        return SwitchContainer()

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

7. Our app is now complete; however, we need to make sure that our server side is ready to handle the JSON requests from our app. Let's go to the server side. The PHP script below (which needs to be stored as ***sync_app_data.php*** under ***public_html➔scripts*** folder on 000webhost.com) expects a table named ***device*** with the following information (you may have different value for *devtype* such as *SENSOR instead of INPUT* and *ACTUATOR instead of OUTPUT*):

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

```
<?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
?>
```

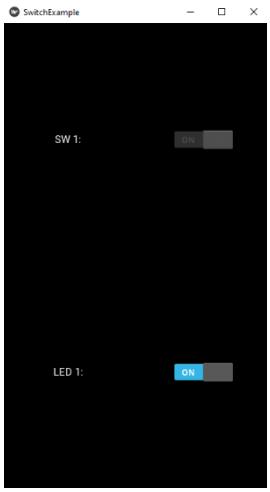
8. Test the app by running it. Swipe *SwitchCompat* LED1 on the Android screen. Refresh the DBOR ***device*** table to make sure that the change has taken place.



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

9. Change the value of SW1 in the DBOR ***device*** table to 1. After 5 seconds or so, the corresponding *SwitchCompat* (SW1) on Android screen should reflect the change (it should show ON).

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



10. Recall that in Assignment #4, you used Python program ***p2_t8.py*** on RPi to interact with the DBOR ***device*** table. Run ***p2_t8.py*** on your RPi and test the interaction between Android App and RPi through DBOR ***device*** table as follows:

    a) Change LED1 on the Android App. After some time (depending on how often RPi is connecting with the DBOR) LED1 on RPi's breadboard should reflect the change.

    b) Change the position of SW1 on RPi's breadboard. After some time SW1 SwitchCompat on the Android App should reflect the change.