LAMPIRAN

```
#include <FS.h>

#define BLYNK_PRINT Serial



#include <ESP8266WiFi.h>

#include <BlynkSimpleEsp8266.h>

BlynkTimer timer;



//needed for library

#include <DNSServer.h>

#include <ESP8266WebServer.h>

#include <WiFiManager.h>



#include <ArduinoJson.h>



#include <TimeLib.h>

#include <WidgetRTC.h>

WidgetRTC rtc;



char currentTime[9];

char blynk_token[34] = "BLYNK_TOKEN";

bool clockSync = false;



//flag for saving data
```

```
bool shouldSaveConfig = false;


//callback notifying us of the need to save config
void saveConfigCallback () {
  Serial.println("Should save config");
  shouldSaveConfig = true;
}


void setup()
{
  pinMode(14, OUTPUT);
  pinMode(12, OUTPUT);
  digitalWrite(14, HIGH);
  digitalWrite(12, HIGH);

  Serial.begin(115200);
  Serial.println();

  //SPIFFS.format();    //clean FS, for testing
  Serial.println("Mounting FS...");    //read configuration from FS json

  if (SPIFFS.begin()) {
    Serial.println("Mounted file system");
    if (SPIFFS.exists("/config.json")) {
      //file exists, reading and loading
```

```
    Serial.println("Reading config file");
    File configFile = SPIFFS.open("/config.json", "r");
    if (configFile) {
      Serial.println("Opened config file");
      size_t size = configFile.size();
      // Allocate a buffer to store contents of the file.
      std::unique_ptr<char[]> buf(new char[size]);

      configFile.readBytes(buf.get(), size);
      DynamicJsonBuffer jsonBuffer;
      JsonObject& json = jsonBuffer.parseObject(buf.get());
      json.printTo(Serial);
      if (json.success()) {
        Serial.println("\nparsed json");

        strcpy(blynk_token, json["blynk_token"]);

      } else {
        Serial.println("Failed to load json config");
      }
    }
  }
} else {
  Serial.println("Failed to mount FS");
}
```

```
  //end read


  // The extra parameters to be configured (can be either global or just in the setup)

  // After connecting, parameter.getValue() will get you the configured value

  // id/name placeholder/prompt default length

  WiFiManagerParameter custom_blynk_token("blynk", "blynk token", blynk_token, 33);     // was 32 length


  Serial.println(blynk_token);


  //WiFiManager

  //Local intialization. Once its business is done, there is no need to keep it around

  WiFiManager wifiManager;


  wifiManager.setSaveConfigCallback(saveConfigCallback);   //set config save notify callback


  //set static ip

  // this is for connecting to Office router not GargoyleTest but it can be changed in AP mode at 192.168.4.1

  //wifiManager.setSTAStaticIPConfig(IPAddress(192,168,10,111),   IPAddress(192,168,10,90), IPAddress(255,255,255,0));


  wifiManager.addParameter(&custom_blynk_token);   //add all your parameters here


  //wifiManager.resetSettings();  //reset settings - for testing
```

```
  //set minimu quality of signal so it ignores AP's under that quality

  //defaults to 8%

  //wifiManager.setMinimumSignalQuality();


  //sets timeout until configuration portal gets turned off

  //useful to make it all retry or go to sleep, in seconds

  wifiManager.setTimeout(600);


  //fetches ssid and pass and tries to connect, if it does not connect it starts an access point with the specified name

  //and goes into a blocking loop awaiting configuration

  if (!wifiManager.autoConnect("SmartRelay", "password123")) {

    Serial.println("Failed to connect and hit timeout");

    delay(3000);

    //reset and try again, or maybe put it to deep sleep

    ESP.reset();

    delay(5000);

  }


  //if you get here you have connected to the WiFi

  Serial.println("Smart  Relay is Connected");


  //read updated parameters

  strcpy(blynk_token, custom_blynk_token.getValue());
```

```
 //save the custom parameters to FS
if (shouldSaveConfig) {
  Serial.println("saving config");
  DynamicJsonBuffer jsonBuffer;
  JsonObject& json = jsonBuffer.createObject();
  json["blynk_token"] = blynk_token;

  File configFile = SPIFFS.open("/config.json", "w");
  if (!configFile) {
    Serial.println("Failed to open config file for writing");
  }

  json.printTo(Serial);
  json.printTo(configFile);
  configFile.close();
  //end save
}

Serial.println("local ip");
Serial.println(WiFi.localIP());

Blynk.config(blynk_token);
Blynk.connect();

timer.setInterval(60000L, activetoday);  // check every 60s if ON / OFF trigger time has been
```

```
reached
  timer.setInterval(1000L, clockDisplay);  // check every second if time has been obtained from the server
  timer.setInterval(100000L, sendWifi);
  timer.setInterval(1000, showCurrentTime);
}

BLYNK_CONNECTED() {
  rtc.begin();
}

void activetoday(){         // check if schedules should run today
  if(year() != 1970){
    Blynk.syncVirtual(V50);  // sync scheduler #1
    Blynk.syncVirtual(V53);  // sync scheduler #2
  }
}

void clockDisplay(){  // only needs to be done once after time sync
  if((year() != 1970) && (clockSync == false)){
    sprintf(currentTime, "%02d:%02d:%02d", hour(), minute(), second());
    Serial.println(currentTime);
    clockSync = true;
  }
}
```

```
BLYNK_WRITE(V50) {   // Scheduler #1 Time Input widget
  TimeInputParam t(param);
  unsigned int nowseconds = ((hour() * 3600) + (minute() * 60) + second());
  unsigned int startseconds = (t.getStartHour() * 3600) + (t.getStartMinute() * 60);
  unsigned int stopseconds = (t.getStopHour() * 3600) + (t.getStopMinute() * 60);
  int dayadjustment = -1;
  if(weekday() == 1){
    dayadjustment = 6; // needed for Sunday Time library is day 1 and Blynk is day 7
  }
  if(t.isWeekdaySelected((weekday() + dayadjustment))){ //Time library starts week on Sunday, Blynk on Monday
    //Schedule is ACTIVE today
    if(nowseconds >= startseconds - 31 && nowseconds <= startseconds + 31 ){     // 62s on 60s timer ensures 1 trigger command is sent
      Blynk.virtualWrite(V51, 1);  // turn on virtual LED
      Serial.println("Schedule 1 started");
    }
    if(nowseconds >= stopseconds - 31 && nowseconds <= stopseconds + 31 ){    // 62s on 60s timer ensures 1 trigger command is sent
      Blynk.virtualWrite(V51, 0);   // turn OFF virtual LED
      Serial.println("Schedule 1 finished");
    }
  }
}
```

```
BLYNK_WRITE(V53) {    // Scheduler #1 Time Input widget
  TimeInputParam t(param);
  unsigned int nowseconds = ((hour() * 3600) + (minute() * 60) + second());
  unsigned int startseconds = (t.getStartHour() * 3600) + (t.getStartMinute() * 60);
  unsigned int stopseconds = (t.getStopHour() * 3600) + (t.getStopMinute() * 60);
  int dayadjustment = -1;
  if(weekday() == 1){
    dayadjustment = 6; // needed for Sunday Time library is day 1 and Blynk is day 7
  }
  if(t.isWeekdaySelected((weekday() + dayadjustment))){ //Time library starts week on Sunday, Blynk on Monday
    //Schedule is ACTIVE today
    if(nowseconds >= startseconds - 31 && nowseconds <= startseconds + 31 ){     // 62s on 60s timer ensures 1 trigger command is sent
      Blynk.virtualWrite(V52, 1);  // turn on virtual LED
      Serial.println("Schedule 2 started");
    }
    if(nowseconds >= stopseconds - 31 && nowseconds <= stopseconds + 31 ){    // 62s on 60s timer ensures 1 trigger command is sent
      Blynk.virtualWrite(V52, 0);   // turn OFF virtual LED
      Serial.println("Schedule 2 finished");
    }
  }
}


BLYNK_WRITE(V54)
```

```
{
  int controlDigitalPins = param.asInt();
  if(controlDigitalPins == 1)
  {
     Blynk.virtualWrite(V51, 1);
     Blynk.virtualWrite(V52, 1);
  }
  else
  {
     Blynk.virtualWrite(V51, 0);
     Blynk.virtualWrite(V52, 0);
  }
}

void showCurrentTime()
{
  String CurrentDate = String(day()) + '-' + monthShortStr(month()) + '-' + year();
  String CurrentTime = String(hour()) + ':' + minute() + ':' + second();
  String formattedDate = CurrentDate + String(" | ") + CurrentTime;
  Blynk.virtualWrite(V1,formattedDate);
}

void sendWifi()
{
  Blynk.virtualWrite(V2, map(WiFi.RSSI(), -105, -40, 0, 100) );
```

```
}


  void loop()
{
  Blynk.run(); // Initiates Blynk
  timer.run(); // Initiates SimpleTimer
}
```