LAMPIRAN

Kode Program Node Sensor

```
#include <WiFi.h>
#include <Firebase_ESP_Client.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <addons/TokenHelper.h>
#include <addons/RTDBHelper.h>
#include <time.h>
#define LED 2
#define relayNozzel 4
#define relayPump 15
#define WIFI_SSID "Sinau Tech"
#define WIFI_PASS "HURUFKECIL@321"
#define API_KEY   "AIzaSyC0Edmh6umgC2x7v9e8qCQPmZ-dt5Saco4"
#define USEREMAIL "cindypratiwisugiono001@gmail.com"
#define USERPASS  "gabisalogin"
#define DATABASE  "https://penyiram-taman-default-rtdb.asia-
southeast1.firebasedatabase.app/"
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
FirebaseJson json;
String uid, sensorPath, controlPath, notify, response, resData;
int cycle, soilMoistData, waterLvlData;
boolean setNozzel, setPump, modeAuto, setNotif, setNotif2;
const int soilMoistPin = 34;
const int waterLvlPin = 35;
void setup() {
  configTime(25200, 0, "pool.ntp.org"); //seting waktu, lokasi jakarta
  pinMode(LED, OUTPUT);
  pinMode(relayNozzel, OUTPUT);
```

```
pinMode(relayPump, OUTPUT);
digitalWrite(relayNozzel, HIGH);
digitalWrite(relayPump, HIGH);
const char *ssid = WIFI_SSID;
const char *password = WIFI_PASS;
digitalWrite(LED, HIGH);
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
while (WiFi.waitForConnectResult() != WL_CONNECTED) {
  delay(5000);
  ESP.restart();
}
digitalWrite(LED, LOW);
ArduinoOTA.setHostname("ESP32 - Cindy");
ArduinoOTA.onStart([]() {
  String type;
  if (ArduinoOTA.getCommand() == U_FLASH) type = "sketch";
  else type = "filesystem";
}).onEnd([]() {
}).onProgress([](unsigned int progress, unsigned int total) {
}).onError([](ota_error_t error) {
  if (error == OTA_AUTH_ERROR);
  else if (error == OTA_BEGIN_ERROR);
  else if (error == OTA_CONNECT_ERROR);
  else if (error == OTA_RECEIVE_ERROR);
  else if (error == OTA_END_ERROR);
});
ArduinoOTA.begin();
Firebase.reconnectWiFi(true);
fbdo.setResponseSize(4096);
config.api_key = API_KEY;
config.database_url = DATABASE;
```

```
  config.token_status_callback = tokenStatusCallback;
  config.max_token_generation_retry = 5;
  auth.user.email = USEREMAIL;
  auth.user.password = USERPASS;
  Firebase.begin(&config, &auth);
  while ((auth.token.uid) == "") delay(1000);
  uid = auth.token.uid.c_str();
  sensorPath = "/" + uid + "/dataSensor/";
  controlPath = "/" + uid + "/controlApp/";
  notify = "/" + uid + "/notification/";
  setNozzel = setPump = modeAuto = setNotif = setNotif2 = false;
  cycle = 0;
}
void loop() {
  ArduinoOTA.handle();
  nozzel();
  pump();
  setMode();
  sendDataString(sensorPath + "soilMoist", soilSensor());
  sendDataString(sensorPath + "waterLevel", waterLvl());
  notifKelembaban();
  notifPompa();
  delay(1000);
}
void sendDataString(String path, String value) {
  Firebase.RTDB.setString(&fbdo, path.c_str(), value);
}
void notifKelembaban() {
  soilMoistData = map(analogRead(soilMoistPin), 4095, 0, 0, 100);
  if ((soilMoistData <= 30) && (setNotif2 == false)) {
    String path = notify + (String)getTime();
    String title = "Tanah Kering";
```

```
    String desc = "Lakukan penyiraman untuk melembabkan tanah.";
    json.set("/title", title);
    json.set("/desc", desc);
    Firebase.RTDB.setJSON(&fbdo, path.c_str(), &json);
    setNotif2 = true;
   }
  if (soilMoistData > 30) setNotif2 = false;
}
void notifPompa() {
  if (digitalRead(relayPump) != setNotif) {
    String path = notify + (String)getTime();
    String title = (digitalRead(relayPump)) ? "Pompa OFF" : "Pompa ON";
    String desc = (digitalRead(relayPump)) ? "Tangki Nozzel " + waterLvl() :
"Tangki Nozzel Hampir Habis, Pengisian dimulai";
    json.set("/title", title);
    json.set("/desc", desc);
    Firebase.RTDB.setJSON(&fbdo, path.c_str(), &json);
    setNotif = digitalRead(relayPump);
   }
}
unsigned long getTime() {
  time_t now;
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) return (0);
  time(&now);
  return now;
}
void nozzel() {
  String readNozzel = controlPath + "nozzle";
  if (Firebase.RTDB.getString(&fbdo, readNozzel)) {
    resData = fbdo.stringData();
    if (resData == "on") setNozzel = true;
```

```
    if (resData == "off") setNozzel = false;
    digitalWrite(relayNozzel, !setNozzel);
  }
}
void pump() {
  String readPump = controlPath + "pump";
  if (Firebase.RTDB.getString(&fbdo, readPump)) {
    resData = fbdo.stringData();
    if (resData == "on") setPump = true;
    if (resData == "off") setPump = false;
    digitalWrite(relayPump, !setPump);
  }
}
void setMode() {
  String readPumpMode = controlPath + "pumpMode";
  if (Firebase.RTDB.getString(&fbdo, readPumpMode)) {
    resData = fbdo.stringData();
    if (resData == "on") modeAuto = true;
    if (resData == "off") modeAuto = false;
  }
}
String soilSensor() {
  soilMoistData = map(analogRead(soilMoistPin), 4095, 0, 0, 100);
  return (String)soilMoistData + "%";
}
String waterLvl() {
  String levelWater = "";
  waterLvlData = analogRead(waterLvlPin);
  sendDataString(controlPath + "sensor", (String)waterLvlData);
  if (waterLvlData < 500) {
    levelWater = "Habis";
    if (modeAuto) sendDataString(controlPath + "pump", "on");
```

```
  } else levelWater = "Sedang";
  if (waterLvlData > 1600) {
    levelWater = "Penuh";
    digitalWrite(relayPump, HIGH);
    if (modeAuto) sendDataString(controlPath + "pump", "off");
  }
  return levelWater;
}
```

Kode Program Main Activity Android

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        auth = Firebase.auth
        viewPager = findViewById(R.id.viewPager)
        tabLayout = findViewById(R.id.tabLayout)
        nickname = findViewById(R.id.username)
        npm = findViewById(R.id.npm)
        bellBtn = findViewById(R.id.bellBtn)
        user = auth.currentUser!!
        databaseUser =
FirebaseDatabase.getInstance().getReference(user.uid).child("user")
        databaseNotify =
FirebaseDatabase.getInstance().getReference(user.uid).child("notification")
        tabInit()
        setData()
        bellBtn.setOnClickListener {
            tabLayout.getTabAt(1)?.select()
        }
        createNotification()
        notificationService()
    }
    private fun setData() {
        databaseUser.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                nickname.text = dataSnapshot.child("nickname").value?.toString() ?: "-"
                npm.text = dataSnapshot.child("npm").value?.toString() ?: "-"
            }
            override fun onCancelled(error: DatabaseError) {}
        })
    }
```

```
    private fun tabInit() {
        adapter = PageAdapter(this)
        adapter.addFragment(HomeFragment(), "Home",
R.drawable.ic_round_home_24)
        adapter.addFragment(NotificationFragment(), "History",
R.drawable.ic_round_notifications_24)
        adapter.addFragment(SettingFragment(), "Settings",
R.drawable.ic_round_settings_24)
        viewPager.adapter = adapter
        TabLayoutMediator(tabLayout, viewPager) { tab, index ->
            tab.text = adapter.setTitle(index)
            tab.setIcon(adapter.setIcon(index))
        }.attach()
    }
    private fun createNotification() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val nameNotif = "Penyiraman dan Kelembaban Tanah"
            val desNotif = "Notifikasi pompa saat melakukan penyiraman dan
monitoring kelembaban tanah"
            val importance = NotificationManager.IMPORTANCE_DEFAULT
            val channel = NotificationChannel(channelId, nameNotif,
importance).apply {
                description = desNotif
            }
            val notifMan: NotificationManager =
getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
            notifMan.createNotificationChannel(channel)
        }
    }
    private fun notificationService() {
        val query: Query = databaseNotify.orderByValue().limitToLast(1)
        query.addChildEventListener(object : ChildEventListener {
```

```
            override fun onChildAdded(snapshot: DataSnapshot, previousChildName:
String?) {
                val title = snapshot.child("title").value?.toString() ?: "none"
                val text = snapshot.child("desc").value?.toString() ?: "none"
                if (stateApp) setNotif(title, text)
            }
            override fun onChildChanged(snapshot: DataSnapshot,
previousChildName: String?) {}
            override fun onChildRemoved(snapshot: DataSnapshot) {}
            override fun onChildMoved(snapshot: DataSnapshot, previousChildName:
String?) {}
            override fun onCancelled(error: DatabaseError) {}
        })
    }
    private fun setNotif(title: String, text: String?) {
        val builder = NotificationCompat.Builder(this, channelId)
            .setSmallIcon(R.drawable.ic_launcher_foreground)
            .setContentTitle(title)
            .setContentText(text)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        with(NotificationManagerCompat.from(this)) {
            notify(notificaitonId, builder.build())
        }
    }
    override fun onResume() {
        super.onResume()
        stateApp = false
    }
    override fun onPause() {
        super.onPause()
        stateApp = true
    }
```

```
override fun onBackPressed() {
    //super.onBackPressed()
    stateApp = true
    moveTaskToBack(true)
}
```

Kode Program Login Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
        auth = Firebase.auth
        loginLayout = findViewById(R.id.loginLayout)
        progress = findViewById(R.id.progress_circle)
        emailField = findViewById(R.id.emailField)
        passwordField = findViewById(R.id.pwField)
        loginBtn = findViewById(R.id.loginBtn)
        val currentUser = auth.currentUser
        if (currentUser != null) {
            startActivity(Intent(this, MainActivity::class.java))
            this.finish()
        }
    }
    override fun onStart() {
        super.onStart()
        loginBtn.setOnClickListener {
            val email = emailField.text.toString().trim()
            val password = passwordField.text.toString()
            if (email.isEmpty()) {
                emailField.error = "Email is Required!"
                emailField.requestFocus()
                return@setOnClickListener
            }
            if (!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
                emailField.error = "Email is Not Valid!"
                emailField.requestFocus()
                return@setOnClickListener
            }
            if (password.isEmpty()) {
```

```
                passwordField.error = "Password is Required"
                passwordField.requestFocus()
                return@setOnClickListener
            }
            loginLayout.visibility = View.GONE
            progress.visibility = View.VISIBLE
            auth.signInWithEmailAndPassword(email,
password).addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    val user = auth.currentUser
                    updateUI(user)
                } else {
                    updateUI(null)
                }
            }
        }
    }
    private fun updateUI(user: FirebaseUser?) {
        if (user != null) {
            startActivity(Intent(this, MainActivity::class.java))
            this.finishAffinity()
        } else {
            progress.visibility = View.GONE
            loginLayout.visibility = View.VISIBLE
            Toast.makeText(baseContext, "Auth Failed!",
Toast.LENGTH_LONG).show()
        }
    }
```

Kode Program Home Fragment

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        auth = Firebase.auth
        valueSoil = view.findViewById(R.id.percentageValueSoil)
        valueWater = view.findViewById(R.id.percentageValueWater)
        valueNozzle = view.findViewById(R.id.textNozzleControl)
        valuePump = view.findViewById(R.id.textWaterControl)
        switchNozzle = view.findViewById(R.id.switchNozzle)
        switchPump = view.findViewById(R.id.switchPump)
        val user = auth.currentUser
        if (user != null) {
            database = FirebaseDatabase.getInstance().getReference(user.uid)
            database.addValueEventListener(object : ValueEventListener {
                override fun onDataChange(dataSnapshot: DataSnapshot) {
                    valueSoil.text =
dataSnapshot.child("dataSensor").child("soilMoist").value?.toString() ?: "-"
                    valueWater.text =
dataSnapshot.child("dataSensor").child("waterLevel").value?.toString() ?: "-"
                    valueNozzle.text =
dataSnapshot.child("controlApp").child("nozzle").value?.toString() ?: "N/A"
                    valuePump.text =
dataSnapshot.child("controlApp").child("pump").value?.toString() ?: "N/A"
                    val nozzleSw =
dataSnapshot.child("controlApp").child("nozzle").value?.toString() ?: ""
                    if (nozzleSw == "on") switchNozzle.isChecked = true
                    else if (nozzleSw  == "off") switchNozzle.isChecked = false
                    val pumpSw =
dataSnapshot.child("controlApp").child("pump").value?.toString() ?: ""
                    if (pumpSw == "on") switchPump.isChecked = true
                    else if (pumpSw == "off") switchPump.isChecked = false
```

```
                val pumpMode =
dataSnapshot.child("controlApp").child("pumpMode").value?.toString() ?: ""
                if (pumpMode == "on") switchPump.visibility = View.INVISIBLE
                else if (pumpMode == "off") switchPump.visibility = View.VISIBLE
            }
            override fun onCancelled(databaseError: DatabaseError) {}
        })
    }
    switchNozzle.setOnCheckedChangeListener { _, isChecked ->
        if (isChecked)
database.child("controlApp").child("nozzle").setValue("on")
        else database.child("controlApp").child("nozzle").setValue("off")
    }
    switchPump.setOnCheckedChangeListener { _, isChecked ->
        if (isChecked) database.child("controlApp").child("pump").setValue("on")
        else database.child("controlApp").child("pump").setValue("off")
    }
```

Kode Program Notifikasi Fragment

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        textDump = requireView().findViewById(R.id.dumpData)
        dataHistory = requireView().findViewById(R.id.recycleView)
        val auth = Firebase.auth
        val user = auth.currentUser
        ref =
FirebaseDatabase.getInstance().reference.child(user!!.uid).child("notification")
        val query: Query = ref.orderByValue().limitToLast(100)
        query.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                val data = ArrayList<ListHistory>()
                snapshot.children.forEach { lists ->
                    val title = lists.child("title").value.toString()
                    val desc = lists.child("desc").value.toString()
                    val timestamp = lists.key.toString()
                    data.add(ListHistory(title, desc, checkTime(timestamp.toLong())))
                }
                adapter = HistoryAdapter(data)
                dataHistory.layoutManager = LinearLayoutManager(activity)
                dataHistory.adapter = adapter
            }
            override fun onCancelled(error: DatabaseError) {}
        })
    }
    @SuppressLint("SimpleDateFormat")
    private fun checkTime(epoc: Long): String {
        return SimpleDateFormat("HH:mm - dd MMMM yyyy").format(Date(epoc *
1000))
    }
```

Kode Program Setting Fragment

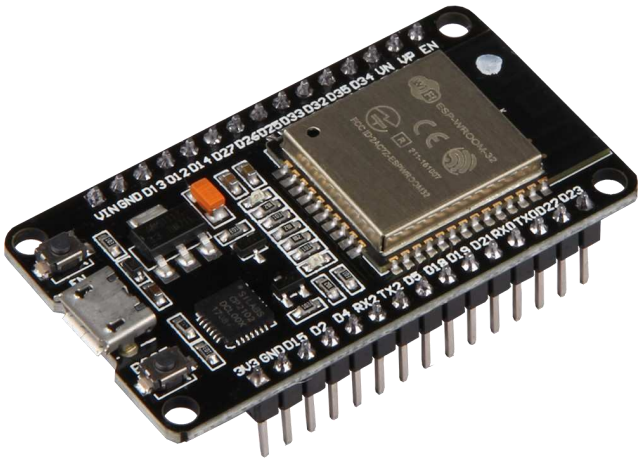
```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        auth = Firebase.auth
        editName = view.findViewById(R.id.editName)
        editNPM = view.findViewById(R.id.editNPM)
        nameText = view.findViewById(R.id.textName)
        npmText = view.findViewById(R.id.textNpm)
        emailText = view.findViewById(R.id.textEmail)
        switchPumpMode = view.findViewById(R.id.switchPumpMode)
        editBtn = view.findViewById(R.id.editBtn)
        saveBtn = view.findViewById(R.id.saveBtn)
        logoutBtn = view.findViewById(R.id.logoutBtn)
        logoutBtn.setOnClickListener {
            Firebase.auth.signOut()
            startActivity(Intent(this.activity, LoginActivity::class.java))
            this.activity?.finishAffinity()
        }
        val user = auth.currentUser
        if (user != null) {
            database = FirebaseDatabase.getInstance().getReference(user.uid)
            database.addValueEventListener(object : ValueEventListener {
                override fun onDataChange(dataSnapshot: DataSnapshot) {
                    nameText.text =
dataSnapshot.child("user").child("nickname").value?.toString() ?: "-"
                    npmText.text =
dataSnapshot.child("user").child("npm").value?.toString() ?: "-"
                    emailText.text = user.email
                }
                override fun onCancelled(databaseError: DatabaseError) {}
            })
```

```
        }
database.child("controlApp").child("pumpMode").get().addOnSuccessListener {
            if (it.value == "on") switchPumpMode.isChecked = true
            else if (it.value == "off") switchPumpMode.isChecked = false
        }
        switchPumpMode.setOnCheckedChangeListener { _, isChecked ->
            if (isChecked)
database.child("controlApp").child("pumpMode").setValue("on")
            else database.child("controlApp").child("pumpMode").setValue("off")
        }
        editBtn.setOnClickListener{
            editBtn.visibility = View.GONE
            nameText.visibility = View.GONE
            npmText.visibility = View.GONE
            saveBtn.visibility = View.VISIBLE
            editName.visibility = View.VISIBLE
            editNPM.visibility = View.VISIBLE
            database.child("user").child("nickname").get().addOnSuccessListener {
                editName.setText(it.value?.toString())
            }
            database.child("user").child("npm").get().addOnSuccessListener {
                editNPM.setText(it.value?.toString())
            }
        }
        saveBtn.setOnClickListener{
            editBtn.visibility = View.VISIBLE
            nameText.visibility = View.VISIBLE
            npmText.visibility = View.VISIBLE
            saveBtn.visibility = View.GONE
            editName.visibility = View.GONE
            editNPM.visibility = View.GONE
```

```
database.child("user").child("nickname").setValue(editName.text.toString())
            database.child("user").child("npm").setValue(editNPM.text.toString())
        }
    }
```

NodeMCU ESP32

Microcontroller Development Board



Technical Specifications

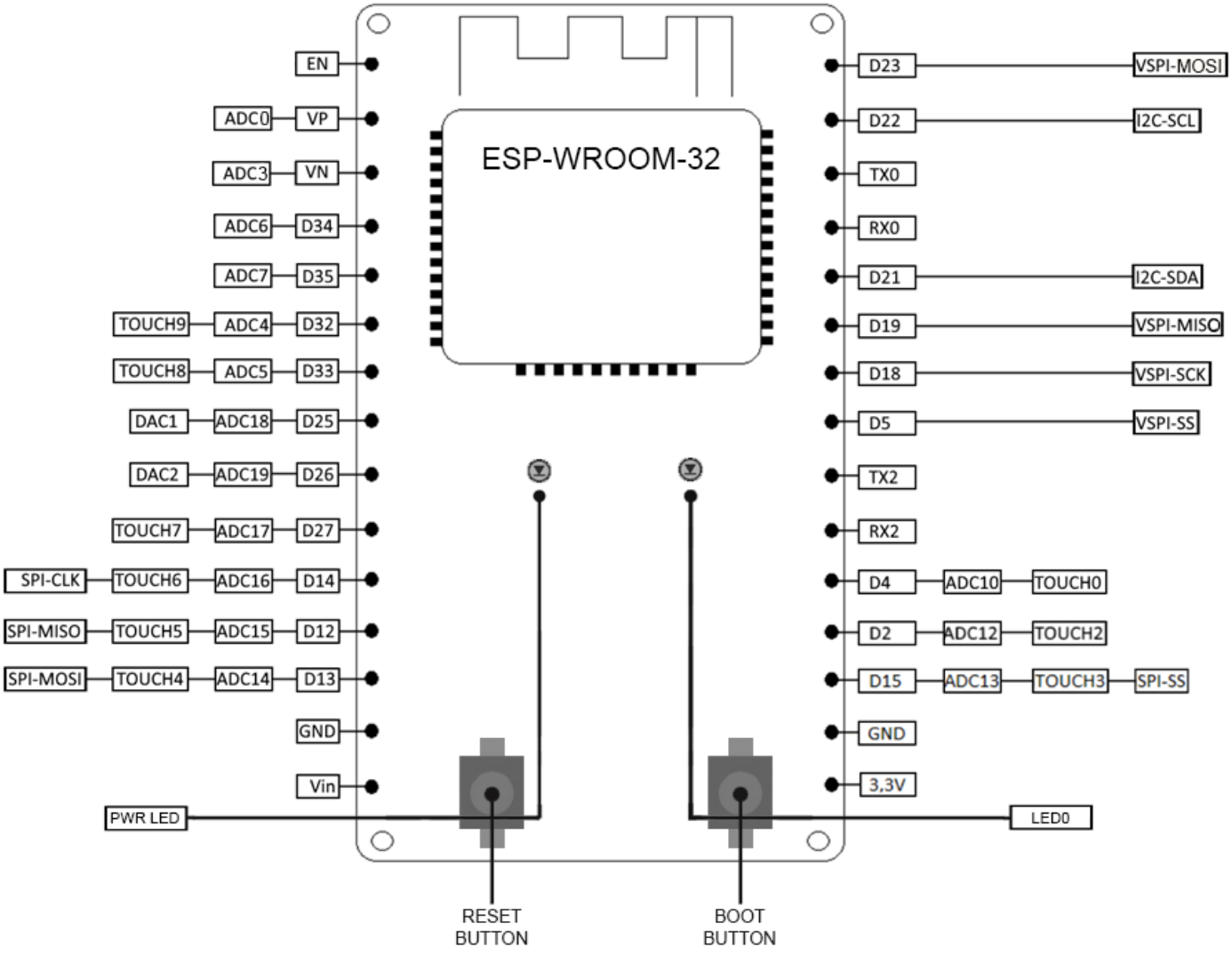
Model	NodeMCU ESP32
Article No.	SBC-NodeMCU-ESP32
Type	ESP32
Processor	Tensilica LX6 Dual-Core
Clock Frequency	240 MHz
SRAM	512 kB
Memory	4 MB
Wireless Standard	802.11 b/g/n
Frequency	2.4 GHz
Bluetooth	Classic / LE
Data Interfaces	UART / I2C / SPI / DAC / ADC
Operating Voltage	3,3V (operable via 5V-microUSB)
Operating Temperature	-40°C - 125°C
Dimensions (W x D x H)	48 x 26 x 11.5 mm
Scope Of Delivery	NodeMCU ESP32
EAN	4250236816104

NodeMCU ESP32

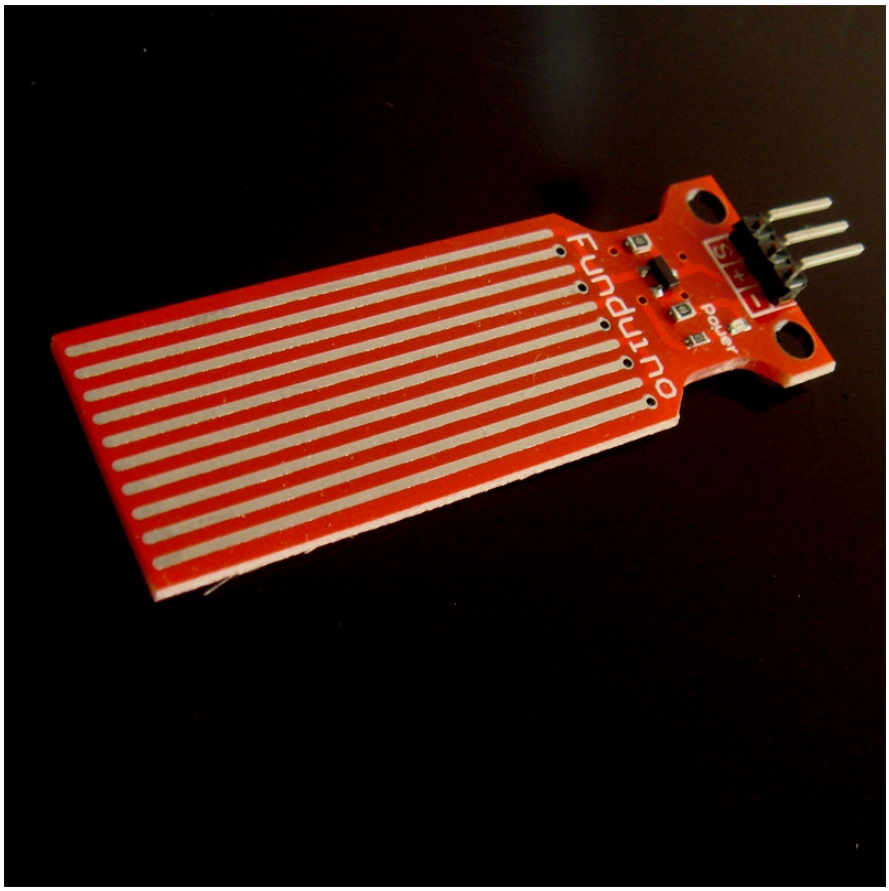
Microcontroller Development Board

An overview of the available pins can be seen in the following figure:

2x DAC	15x ADC	1x SPI
1x I²C	2x UART	



Water Sensor Module User's Manual



I. Notice

（1）did not carefully read the instructions before you do not give the driver board is powered ! Avoid faulty wiring caused permanent damage to the drive plate .

（2）Please carefully check pin function , attention condensed identifier , correct wiring ! Do not reverse the power cord , resulting in
Electronic devices burned.

2. the product introduction

2013 latest Water Sensor is a Easy to use, compact and lightweight , high cost of water , droplets identification and detection sensors. This sensor is working The principle is to measure the size of the trace amount of water droplets through the line with a series of parallel wires exposed . And domestic and foreign Products compared not only small , powerful, and cleverly designed with the following features : First, the amount of water to simulate Conversion ; Second, plasticity , based on the sensor output analog values ; Third, low power consumption , high sensitivity ; Fourth, can Directly connected to a microprocessor or other logic circuitry , and the controller board for a variety of , for example : ArduinoController , STC microcontroller , AVR microcontroller and so on.

3, the specification parameters

1 Product Name: water level sensor
2 Item :. K-0135
3 Operating voltage :. DC5V
4 Working current : less than 20mA
5 Sensor Type : Analog
6 detection area :. 40mm x16mm
7 Production process :. FR4 double-sided HASL
8 mounting hole size : 3.0mm
9 user-friendly design : half-moon -slip handle depression
10 Working temperature :. 10 ℃ -30 ℃
11 Operating Humidity : 10% ~ 90 % non -condensing
12 Weight :. 3g
13 Product Dimensions : 65mm x 20mm x 8mm

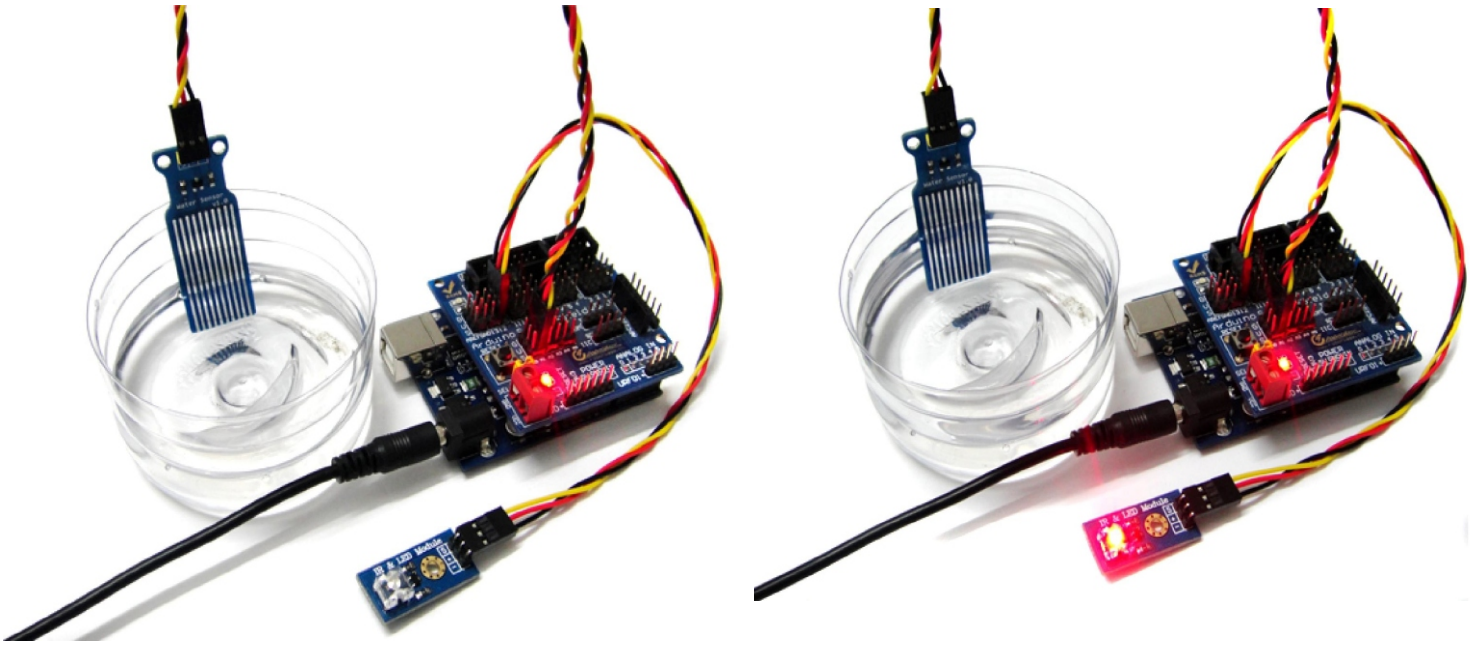
4., the test Water Sensor Module
We use the Arduino controller to be tested , need to use hardware devices as follows :
1, Arduino controller × 1
2, Arduino sensor expansion board × 1
3, Water Sensor Module × 1
4, 3P sensor cable × 2
5, IR & LED Modue (red) × 1
6, USB data communication cable × 1
Water Sensor DuPont line will be connected to the Arduino sensor expansion board interface
A1. The use of sensors
The red line will be connected to the Arduino piranha light sensor expansion board D8. After completing the hardware connection , the code is compiled
After downloading the Arduino inside .

Arduino experimental code .

```
int analogPin = 1; / / level sensor connected to an analog port
int led = 12; / / Piranha LED connected to digital port 12
int val = 0; / / define a variable val initial value of 0
int data = 0; / / define a variable data initial value of 0
void setup ()
{
pinMode (led, OUTPUT); / / define led to an output pin
Serial.begin (9600); / / set the baud rate to 9600
}

void loop ()
{
val = analogRead (analogPin); / / read the analog value to the variable val
if (val> 700) {/ / determine whether more than 700 variables val
digitalWrite (led, HIGH); when / / variable val is greater than 700 , Piranha LED lights
}
else {
digitalWrite (led, LOW); when / / variable val is less than 700 , the lamp goes out piranha
}
data = val; / / variable is assigned to the variable data val
Serial.println (data); / / Serial print variable data
delay (100);
}
```

After these steps are completed, we test the low water level, see experimental phenomena :



The water level does not reach the warning value , piranhas lamp is not lit
Water level reaches and exceeds the alert value , piranha lights , initiate alarm.

SMR110
SOIL MOISTURE RECORDER

Features

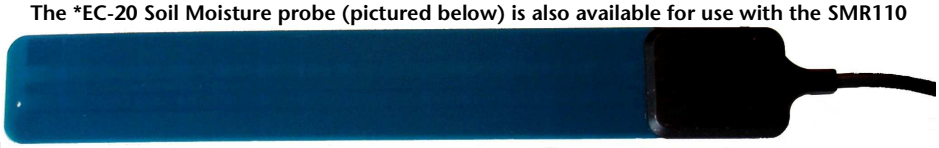
- High accuracy sensor
- Fast response time
- Simple soil insertion (vertical or horizontal orientation)
- Programmable start time and date
- Miniature size
- User calibration through MadgeTech software
- Weatherproof enclosure
- Probe is powered from data recorder's battery. No external power required.

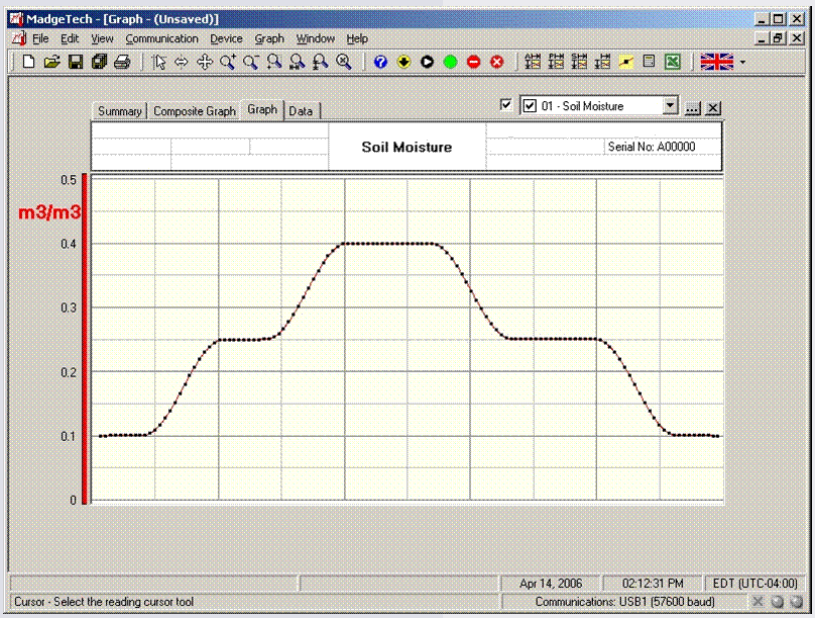
Applications

- Irrigation Management
- Potted plant and greenhouse monitoring
- Soil moisture trends
- Plant growth studies
- Peat Moss and organic soil studies

The SMR110 is a small, battery powered, soil moisture data recorder in a weatherproof enclosure. It provides accurate volumetric water content measurements over a complete range from dry to saturated. The SMR110 features a 10-year battery for long-term deployments and soil studies, over multiple seasons. With it's built-in memory, the data recorder can monitor and record over 32,767 soil moisture measurements. The EC-5, a high accuracy soil moisture probe (pictured above), has a response time of 10ms and can be placed in any type of soil. The EC-20* probe (below) is designed for use in medium-textured soils. For easy soil moisture analysis and trend information, MadgeTech software provides a time and date stamped graph. This data may be viewed, printed or exported to third-party programs (Excel, etc…) for more detailed analysis and irrigation management. Soil moisture data is essential in determining and scheduling irrigation for proper plant growth, care and research. This data is also useful in determining the health of the soil and plant root systems.

The *EC-20 Soil Moisture probe (pictured below) is also available for use with the SMR110





MadgeTech Data Recorder Software displays water per soil content data in an easy to use graph.

The Windows®-based software package allows the user to effortlessly collect, display and analyze data. A variety of powerful tools allow you to examine, export, and print professional looking data with just a click of the mouse.

Click MadgeTech Software for more information or to download the software.

879 Maple Street · Contoocook, NH 03229
PO Box 50 · Warner, NH 03278 · Phone: (603) 456-2011 · Fax: (603) 456-2012

SOIL MOISTURE RECORDER SPECIFICATIONS

Data Recorder

Measurement Range: +/- 1200mV
Accuracy: +/-0.01% FSR

Input range: 0 to 2.5V DC
Memory: 32,767 readings; software configurable memory wrap
Reading Rate: 1 reading every 2 seconds to 1 every 12 hours

Input Connection: Removable screw terminal plug
Operating Environment: -40 to +80°C (-40 to +176°F), 0-95% RH, non-condensing

Battery Type: 3.6V Lithium Battery included; **user replaceable**

Battery Life: 10 years typical @ 15 min reading rate

Dimensions: 0.8"x1.7"x2.7"(21mmx44mmx69mm)
Weight: 2 oz. (56 g)

Weatherproof Enclosure: Anodized aluminum case w/mounting flange. Communications port plug 3.5"x2.9"x1.1"(87mmx73mmx27mm) 7 oz. (198 g)

Soil Moisture Probes

Measurement Range: (EC-5) 0 to 100% VWC saturation
(EC-20) 0 to 40% VWC saturation
Measurement Type: VWC (Volumetric Water Content)

Measurement Accuracy: (EC-5) +/- 3% typical, ALL soils
(EC-20) +/- 4% typical on low EC and medium textured soils
(Both models) +/-1% with soil specific calibration

Calibration: Digital calibration through software
Calibration Date: Automatically recorded within device
Life Expectancy (probes): 3-5 years
Measurement Time: 10ms
Measurement Resolution: 0.002m^3/m^3
Operating Environment: -40 to +60 degC (-40 t0 +140 degF), 0-100% RH (EC-5 and EC-20 models)

Power Requirement: (probes) Powered from data recorder's internal battery. No external power required.

Dimensions (probes): (EC-5) 2.1"x0.6"x0.06" (55mm x 15mm x 1.5mm) (Cable length) - 16' (5 m), (tinned, wire leads)

(EC-20) 10"x1.25"x0.06" (255mmx32mm x1.5mm) Cable length) - 16' (5 m), (tinned, wire leads)

SOFTWARE FEATURES

Multiple Graphs: Simultaneously analyze data from several units or deployments; easily switch to a single data series

Real-Time Recording: Collect and display data in real-time while continuing to log

Data Table: Instantly access tabular view for detailed dates, times, values, and annotations

Scaling Options: Autoscale function fits data to the screen, or allows user to manually enter their own values

Printing: Automatically print graphical or tabular data

Statistics: Calculate averages, min, max and standard deviation

Export Data: Export data in a variety of common formats, or switch to Excel® with a single click

Calibration: Automatically calculate and store calibration parameters

Logger Configuration: Easy set up and launch of data loggers with immediate or delayed start, preferred sample rate, and device ID

Communications: Automatic or user-enabled communications port configuration (Baud rate, COM port selection)

ORDERING INFORMATION

Model	Description	Price (U.S.)
SMR110-5	Soil Moisture recorder w/EC-5 probe, weatherproof enclosure (Software NOT included)	$299.00
SMR110-20	Soil moisture recorder w/EC-20 probe, weatherproof enclosure (Software NOT included)	$299.00
EC-5	Soil moisture probe, ONLY	$130.00
EC-20	Soil moisture probe, ONLY	$130.00
IFC110	Software, manual and serial interface cable	$99.00
IFC200	Software, manual and USB interface cable	$119.00
LTC-7PN	Replacement battery for data recorder	$10.00

BATTERY WARNING: FIRE, EXPLOSION, AND SEVERE BURN HAZARD. DO NOT RECHARGE, DISASSEMBLE, HEAT ABOVE 212ºF, INCINERATE OR EXPOSE CONTENTS TO WATER.

ASK ABOUT OUR OTHER DATA RECORDERS

Temperature	Pulse/Event/State
Humidity	Low Level Current
Pressure	Low Level Voltage
pH	RF Transmitters
Level	Intrinsically Safe
Shock	Spectral Vibration
LCD Display	

For Quantity Discounts call 603-456-2011 or email sales@madgetech.com

MADGETECH
WE BUILD DATA LOGGERS

DOC-1125009-00 REV C 2009.07.01

879 Maple Street · Contoocook, NH 03229
PO Box 50 · Warner, NH 03278 · Phone: (603) 456-2011 · Fax: (603) 456-2012