JURNAL EMACS
[Engineering, MAthematics and Computer Science] Vol.5 No.3 September 2023: 129-135 e-ISSN: 2686-2573
DOI: 10.21512/emacsjournal.v5i3.9697

Relay Driver Based on Arduino UNO to Bridge the Gap of The Digital Output Voltage of The Node MCU ESP32

Yulianto
Computer Science Department, School of Computer Science,
Bina Nusantara University,
Jakarta, Indonesia 11480
yulianto003@binus.ac.id

*Correspondence: yulianto003@binus.ac.id

Abstract – *The IoT could control the devices that need a high current voltage to operate. The voltage control here means that the IoT could give the command to turn on and turn off the electric current by using a relay module. One of the devices that are most frequently used in many research projects is Node MCU ESP8266 and Node MCU ESP32. Those microcontrollers work with the maximum supply is 3.3-volt direct current (DC). On the other hand, the relay module commonly needs a voltage supply of 5-volt DC and the relay needs to be controlled by a single transistor to make a trig on. The relay will be active when the transistor's basis pin is grounded into the ground, so the relay will get the current flow. However, the relay module which is controlled using Node MCU could not work properly, caused Node MCU only provides the digital out is 3-volt maximum from its digital Input Output pins (I/O). Meanwhile, the driver relay based on a single transistor needs a bias input amount of 5 volts to make the relay module active well. If the bias voltage doesn't reach 5 volts or just 3.3 volts will make the relay can't switch on properly which can result in bad contact. To overcome that problem this research proposed the driver relay based on Arduino UNO. The novel of this research is adding the Arduino UNO module between Node MCU and the relay module which has task to bridge the voltage difference between the output digital output ESP that only maximum 3.3 volt converted by Arduino to be digital output which can reach the voltage of 5 volt. The Arduino JSON library was also involved to wrap the commands that produced by Node MCU then deserialized on Arduino to parse and convert to be digital output to control the relay module.*

Keywords: *Relay Module; Arduino UNO; Node MCU; Arduino JSON*

I. INTRODUCTION

The Internet of Things (IoT) concept has been being an important key to supporting the digitalization of smart-era technology (Nižetić et al., 2020). With the implementation of IoT architecture, various electronic hardware and systems software like physical devices, home appliances, and other embedded systems that equipped with sensor, network architecture, can communicate each other to share and exchange information data (Ivanov et al., 2021) (Zheng et al., 2021). The IoT allows for the seamless flow of information between devices and enables the automation of various processes, with the aim to increase efficiency and improve daily life experiences. IoT is a framework that could read the surrounding condition i.e., temperature, humidity, lighting, etc. that are assisted by sensors (Ding et al., 2020). The result of the reading sensor then can be monitored by people anywhere (Rahman et al., 2020). People also have the capability to control the environment through their gadgets or other mobile technology (Xu et al., 2020)(Shen et al., 2021). One of the existing components in IoT framework that was used as couple between the low current control to high current voltage control was relay module. A relay module has several roles, i.e., to control the high-voltage supply or high-current that flows on device target-controlled (Parab & Prajapati, 2019).

Relay designed to isolate the low current electricity area from that connected to microcontroller with high current electricity or hot area that was used to supply other electronic devices. Relay works mechanically. Although the relay separates the control area between low voltage or current with high voltage, the relay still needs a driver component for a bridge between the core of the source signal with the low voltage area. The relay still needs an additional driver component coupled between the digital output pin of the microcontroller to the relay. The driver relay model is needed because to activate the relay module, the relay's pin can't connect directly to the microcontroller because can damage the microcontroller because of the high current flowing into the microcontroller's pin.

The driver relay can be a single transistor, optocoupler or integrated circuits (IC) (Uguru-Okorie et al., 2022). The relay module that still uses a single transistor to drive the relay switch is known to have a deficiency in activating the relay. The bias voltage must be 5 volts. Whereas the digital output of Node MCU ESP32 maximum is 3 volts. The digital output of Node MCU ESP32 that only produces 3 volts to trigger the relay's driver makes the relay can't activate properly. The example of the relay module with a single transistor as a driver can be seen in **Figure 2** and for the detail of schematic diagram relay's module from **Figure 2** can be seen in **Figure 3**.

To overcome the problems related to the gaps in voltage between Node MCU's output that only can trigger 3 volts with the relay module that needs 5 volts for input trigger, this study aims to propose a relay's driver module that consists of Hardware based on Arduino UNO and software that involved a JSON Arduino library. The intent of using Arduino UNO as a liaison between Node MCU ESP32 and with relay module is to separate the task and to change the digital output from Node MCU ESP32 originally only 3 volts to 5 volts. So, it can be said the Node MCU ESP32 focuses on computing other task programs or other advanced processing, and the Arduino UNO is used to control the driver relay. All the command output from Node MCU ESP32 is wrapped in JSON format and then delivered to the Arduino UNO through serial communication. The Arduino UNO then does the process of deserializing the incoming command that is sent from Node MCU ESP32 to be converted as a digital output command to control the relay module.

Research about home automation for controlling devices remotely or wirelessly to help elderly people, disabled people, and for efficiency of time has been conducted Amoran et al (Amoran et al., 2021). The home automation system that has been proposed in the last study can be controlled by the smartphone to turn on and off the lamps. The control of lamps was done by a relay module. The relay module that has been proposed is equipped with Darlington IC ULN2003A as the driver module. A changed manual switch with a relay module has also been conducted by Perkasa et al (Perkasa et al., 2021)(Hermanu et al., 2022) (Hassan et al., 2022), where the implementation of the system is to make the switch room can operate on or off automatically. The controlling switch using a relay also has been done by Sun et al (Sun et al., 2021) where their research aimed to minimize the spark that arises when switching in mode on or off with paralyzed relay and MOSFET together. The research on the implementation of relays for circuit breakers has been done by Siu et al (Siu et al., 2020) where the relay is used to control the direct current in a microgrid system.

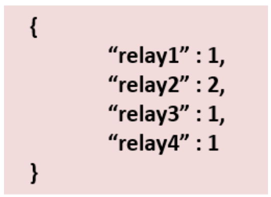
Although there are many studies that proposed a specific component associated with Node MCU ESP32, especially that used for separating between low voltage with high voltage areas, the fact found that the research focuses to connect 5 volts of relay module with a single driver transistor to Node MCU ESP32 is still limited. It's because the power supply to work is different between the relay module and Node MCU ESP32 will result in the relay can't working properly when the relay module is triggered to activate.

On the previous research to overcome the voltage's gap between output Node MCU with relay module, the utilization of relay module which has the same working voltage with the Node MCU which is 3 volts has been proposed by El-Sayed et al (El-Sayed et al., 2021). Inspired by the several pieces of literature mentioned before in this session, this research tries to keep using a relay module with a voltage of 5 volts using the adding Arduino UNO module as an additional driver, on the relay module.

II. METHODS

Node MCU ESP32-The Node MCU ESP32 is a device microcontroller known to be used in many IoT projects. This microcontroller has also been equipped with 2.4 GHz Wi-Fi and Bluetooth standard wireless communication. For example, the Node MCU ESP32 was used in the temperature and monitoring project proposed by Yahya et al (Yahya et al., 2020). Project related to the monitoring of air quality in mesh protocol network topology that has been conducted by Khan et al was also utilize the Node MCU ESP32 (Khan et al., 2022). The detailed specification for wireless communication in Node MCU ESP32 is the Wi-Fi standard 802.11 n with a maximum bandwidth of 150 Mbps attached. The Bluetooth specification used is the Bluetooth low energy (BLE) version 4.2 controller, with a maximum transmission power for Bluetooth of +10 dBm. The sensitivity for the Bluetooth receiver was -98 dBm.

Arduino JSON-JavaScript Object Notation that abbreviated from JSON is known as a lightweight document format for expressing the nature of data (Bourhis et al., 2020). The JSON format consists of a key and value pair (Liu et al., 2020), where the key is typed in the left part and the value is typed in the right part (see **Figure 1**). **Figure 1** is the example of JSON document which contains 4 keys there are relay1, relay2, relay3, and relay 4, and on every key paired with value. For example, key of relay1 assigned with the value 1, relay2 assigned with value of 2 etc.



```
{
        "relay1" : 1,
        "relay2" : 2,
        "relay3" : 1,
        "relay4" : 1
}
```

Figure 1. The example representation of the JSON format

Where the key refers to the name of the variable and the value data can be a string, number, array, object, etc. (Mironov et al., 2020)(Lee et al., 2021). Arduino JSON is a library that can be used to connect and exchange information data from the microcontroller to web service or between other microcontroller boards through serial protocol communication (Blanchon, n.d.). The Arduino JSON library is a third-party library and must be manually installed. Arduino JSON could wrap multiple keys or variables and values together, which is called the operation of serialization. On the opposite side the reverse operation is used to extract information from the JSON format that is known as deserializing JSON. Arduino JSON really works on low-resource computation because this library just needs less room of Random Access Memory (RAM) and less central processing unit (CPU) cycles like small microcontrollers like Arduino or other edge devices that support wireless technology like ESP8266, ESP32, and so on. Arduino JSON library can also be used for multipurpose programming that can be implemented on the personal computer (PC) with different types of operating systems like Linux, MacOS, and Windows.

Relay-As shown in **Figure 2**, this is an example of the 1-channel relay card. The module operates at a low level of 5-volt direct current (VDC). The relay can be used to switch electricity with a maximum voltage of 250 volts alternating current (VAC) 10 amperes (A). if the amperage is higher that 10 A for example up to 15A, then the maximum voltage that allowed to flow through the relay is 125 VAC maximum. The relay module shown in **Figure 2** can also be used as a pair of switches for direct current with a maximum current of 10A for 30 VDC. To control this relay, an operating voltage of 5 VDC with a current of 15-20 milliamperes (mA) is required.

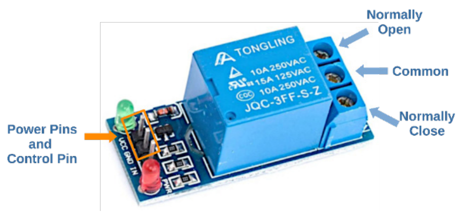


Figure 2. An example of low level 5V relay with a single driver NPN transistor.

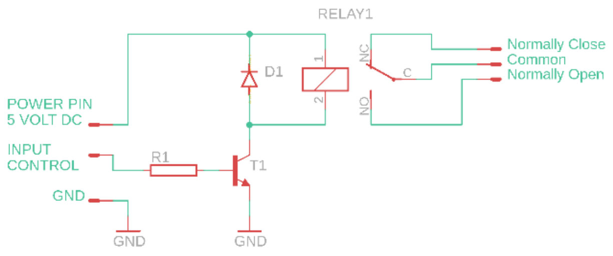


Figure 3. Schematic of a low-level 5-volt relay from Figure 2 that simplified for the principle.

Figure 3 is a schematic diagram of **Figure 2**. The relay module, shown as RELAY 1 in **Figure 3**, is equipped with a negative-positive-negative (NPN) transistor, that is written with code as T1. The single-channel relay module is fitted with 3-pin connectors. The connector pins are normally closed (NC), common (C), and normally open (NO). The word "normally" means that if T1 is not biased, the C pin on the relay will be connected to the NC pin on the relay. Otherwise, if the base pin on T1 is biased, the collector and emitter of T1 will conduct and make RELAY1 active, so the C pin of the relay will be connected to the NO pin of the relay.

To control the relay module, the Node MCU ESP32 is connected directly to the relay module because the relay will being weak mechanically to make contact between the common (C) relay pin and the normally open (NO) relay pin. The weak state of the relay contact is caused by the fact that the 5V supply to the relay is insufficient to power it. Insufficient current electricity to activate a relay is caused by the pin base driver transistor, which controls the relay, only receiving a maximum of 3 volts bias.

The base pin driver transistor is connected to the Node MCU ESP32. The digital output of the Node MCU ESP32 is 3 volts at high output conditions, which is insufficient to drive the bias transistor on the relay module. **Figure 4** is a general description of the proposed method to overcome the gap bias electricity between the digital output of Node MCU ESP32 with the relay module that requires a 5 volt as bias input.

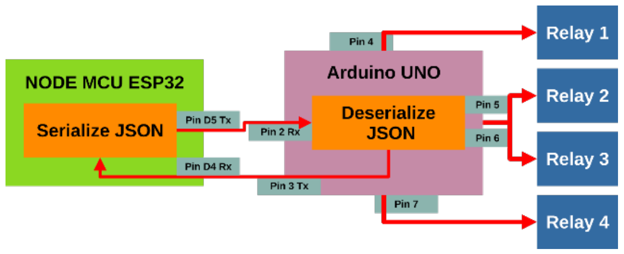


Figure 4. The diagram of the proposed method overall, where the Node MCU ESP32 does not control the relay directly, but instead through Arduino UNO which stands for relays driver.

As shown in **Figure 4**, the Node MCU ESP32 is connected to the Arduino UNO via the serial protocol using a pair of cables. The serial protocol consists of digital pin 5 of the ESP32, which acts as a distributor serial data (Tx) pin, connected to digital pin 2 of the Arduino UNO, which acts as a recipient serial data (Rx) pin. Arduino pin 3 acts as the Tx pin connected to the Node MCU ESP32 on digital pin D4 as the Rx pin. This method also uses serial JSON library to encapsulate the command from ESP32 sent to Arduino UNO. After the JSON data is received by Arduino UNO, the JSON is deserialized and extracted to obtain the command for further use to control the digital input/output (I/O) pin. The digital I/O pins of Arduino UNO that used in this study are Pin 4, 5, 6, 7 that connected to Relay module 1, 2, 3, and 4 respectively (see **Figure 5**).

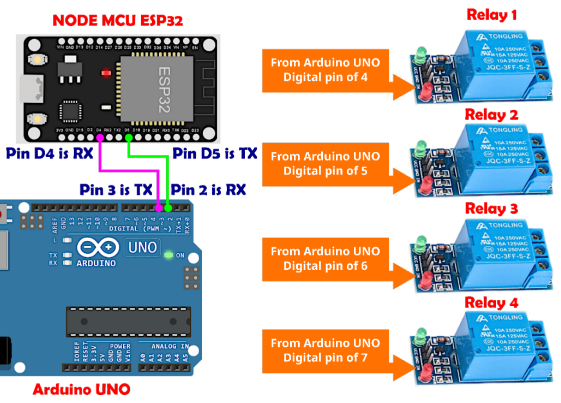

NODE MCU ESP32
Pin D4 is RX
Pin D5 is TX
Pin 3 is TX
Pin 2 is RX
Arduino UNO
Relay 1
Relay 2
Relay 3
Relay 4
From Arduino UNO Digital pin of 4
From Arduino UNO Digital pin of 5
From Arduino UNO Digital pin of 6
From Arduino UNO Digital pin of 7

Figure 5. The detail of the proposed method to overcome the lack of control from the Node MCU ESP32 to the relay module, caused by a gap of different voltages, by adding the Arduino UNO module.

Figure 5 is the actual implementation of the study proposal from **Figure 4**. As shown in **Figure 5**, to give a command from Arduino UNO to the Relay module, the digital output pin number 4 from Arduino UNO is connected to the Relay 1 module. Digital output pin number 5 is connected to Relay 2. The digital output pin number 6 is connected to Relay 3 and the digital output pin number 7 of the Arduino is connected to Relay 7.

The step-by-step experiment carried out will be discussed in this session. Where the experiment consists of 4 steps. The first step is the installation of the hardware, the second step is the programming of the Node MCU ESP32, the third step is the programming of the Arduino module and the last or fourth step is an evaluation by doing the experiment of switching on and off the relay that was done by the Node MCU ESP32.

The first step is the stringing hardware process, which consists of making a hardware connection by soldering each pin Tx/Rx between the Node MCU ESP32 and Arduino UNO using the cable and soldering the digital output pin from Arduino to the relay module, also using a cable. The details of the wiring hardware can be seen in **Figure 6**.

In the second step, a program displayed on **Code 1** is used to program the Node MCU ESP32. In **Code 1** there are four different variables that have been used. These are Relay1, Relay2, Relay3, and Relay4. For example, when the process is initialized as `doc["relay1"] = 1,` it means the Node MCU ESP32 will send the command to turn off the relay that will deliver to Arduino UNO. The ESP32 will send the command to turn on the relay module if the value set as 2, for example the program in Node MCU ESP32, the Arduino JSON form can be expressed as `doc["relay3"] = 2.` The step on the **Code 1**, the SoftwareSerial.h library was used to configure the custom of serial pin hardware freely.

```
#include <ArduinoJson.h>
#include <SoftwareSerial.h>

SoftwareSerial s(5, 4);

void setup() {
  s.begin(9600);
}

void loop() {
  StaticJsonDocument<255> doc;
  doc["relay1"] = 1;
  doc["relay2"] = 1;
  doc["relay3"] = 2;
  doc["relay4"] = 1;

  serializeJson(doc, s);
}
```

Code 1: The programs for Node MCU ESP32 that contained library ArduinoJson for wrapping or serializing of four different variables then sent to custom serial that declared on digital pin 5 for serial transmitter and digital pin 4 for serial receiver.

The third step was, after the programming of the Node MCU ESP32 using the source code shown in **Code 1**, the development of the Arduino UNO module was continued by programming using the source code shown in **Code 2**. For the Arduino UNO module, the Software was also used to custom initialize the new Rx/Tx pin configuration. The baud rate that was used for serial communication between Node MCU with Arduino was 9600.

The fourth step is to evaluate a way to give the command to switch on or off by programs statically on ESP32 and then measure the output result of Arduino UNO connected to the relay module.

```
#include <SoftwareSerial.h>
#include <ArduinoJson.h>
SoftwareSerial s(3, 2);
int relay1 = 4;
int relay2 = 5;
int relay3 = 6;
int relay4 = 7;
void setup() {
  s.begin(9600);
  pinMode(relay1, OUTPUT);
  pinMode(relay2, OUTPUT);
  pinMode(relay3, OUTPUT);
  pinMode(relay4, OUTPUT);

  digitalWrite(relay1, LOW);
  digitalWrite(relay2, LOW);
  digitalWrite(relay3, LOW);
  digitalWrite(relay4, LOW);
}
int relay_1 = 0;
int relay_2 = 0;
int relay_3 = 0;
int relay_4 = 0;

void loop() {
  StaticJsonDocument<255> doc;
  DeserializationError err = deserializeJson(doc, s);
  if (err == DeserializationError::Ok) {
    relay_1 = (int)doc["relay1"];
    if(relay_1 == 1) digitalWrite(relay1, LOW);
    else if(relay_1 == 2) digitalWrite(relay1, HIGH);

    relay_2 = (int)doc["relay2"];
    if(relay_2 == 1) digitalWrite(relay2, LOW);
    else if(relay_2 == 2) digitalWrite(relay2, HIGH);

    relay_3 = (int)doc["relay3"];
    if(relay_3 == 1) digitalWrite(relay3, LOW);
    else if(relay_3 == 2) digitalWrite(relay3, HIGH);

    relay_4 = (int)doc["relay4"];
    if(relay_4 == 1) digitalWrite(relay4, LOW);
    else if(relay_4 == 2) digitalWrite(relay4, HIGH);
  }
}
```

Code 2: The programs that implemented on Arduino UNO as a driver relay module.

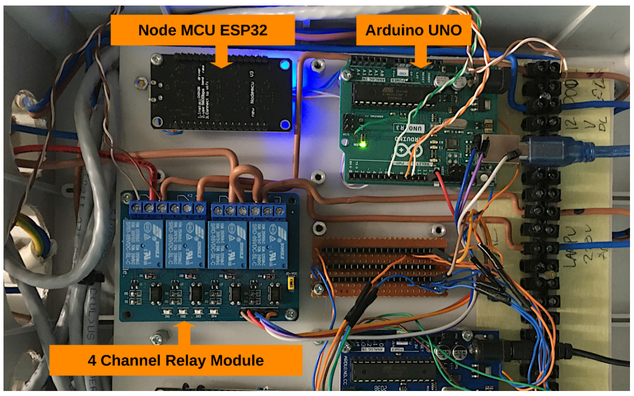


Figure 6. The result of experimenting with a combined Node MCU ESP32 connected to an Arduino UNO and a 4 channel Relay module.

III. RESULTS AND DISCUSSION

In this section, 5 different experiments were carried out to prove the result. The detailed result can be seen in **Table I**. The first experiment is to make relay module 1, relay module 2, relay module 3, and relay module 4, being inactive or turned off. It has been done by programming the Node MCU ESP32 to set all the key of Relay1, Relay2, Relay3, and Relay 4 set to 1 in JSON form. By using the command of "`serializeJson(doc, s)`" in **Code 1**, all commands will wrapped and stored in "**`doc`**" variable and distributed to custom serial pin configuration that initialized as "**`s`**" variable. The data in JSON format that is distributed by Node MCU ESP32 then received by Arduino UNO module. In the Arduino module then the deserialization process with purpose to get the value from key variables is carried out. If the value is 1 then the Arduino will execute the process of digital write zero to digital pin output. The result is the digital output pin will in zero-volt condition and will make the relay module not triggered, and as a result the relay module will not active or OFF.

In **Table I**, the number of sequence experimental that denoted as 2, 3, and 4 is attempt to turned-on and turned-off the relay module by set the Relay's key with integer number of 2. Sequence experiment number 5 in **Table I** will attempt to switch on all the relay modules by setting the All-relays keys to number 2. The reason why in experiment the integer number 1 was used to turned off command and number 2 used to turn on instead of using number 0 to turned off and number 1 to turned on was there possibilities from the deserialize process in Arduino UNO could product the zero number. These conditions can make the false decision for the relay module which can result in the relay module not being stable, or experience turned on or off in irregular conditions. As a global result, 5 sequences of experimental can execute precisely to control when the relay must be turned on or off.

Although this study can control the relay module successfully, this study has disadvantages in communication where the command that is sent from the Node MCU ESP32 to Arduino is still in only one way, and asynchronous. For future work, several machine-to-machine communication techniques like serial protocol interface (SPI), Inter-Integrated Circuit (i2c), and or combination of serial communication with SPI or i2c can be implemented to make the connection between Node MCU ESP32 to Arduino UNO module.

Table I. The experimental results of five trials to control turn-on and turn-off relay modules that begin by giving a command from Node MCU ESP32 that is encoded in JSON format and then transformed or encoded by Arduino UNO being a trigger command for the relay module to turn-on or off

Sequence Experimental	Node MCU ESP32 Command	Arduino UNO Module				Relay Module			
		Digital Pin 4	Digital Pin 5	Digital Pin 6	Digital Pin 7	Relay 1	Relay 2	Relay 3	Relay 4
1	{ "Relay1" : 1, "Relay2" : 1, "Relay3" : 1, "Relay4" : 1, }	0 Volt	0 Volt	0 Volt	0 Volt	OFF	OFF	OFF	OFF
2	{ "Relay1" : 2, "Relay2" : 1, "Relay3" : 1, "Relay4" : 1, }	**5 Volt**	0 Volt	0 Volt	0 Volt	**ON**	OFF	OFF	OFF
3	{ "Relay1" : 2, "Relay2" : 2, "Relay3" : 1, "Relay4" : 1, }	**5 Volt**	**5 Volt**	0 Volt	0 Volt	**ON**	**ON**	OFF	OFF
4	{ "Relay1" : 2, "Relay2" : 2, "Relay3" : 2, "Relay4" : 1, }	5 Volt	5 Volt	5 Volt	0 Volt	**ON**	**ON**	**ON**	OFF
5	{ "Relay1" : 2, "Relay2" : 2, "Relay3" : 2, "Relay4" : 2, }	5 Volt	5 Volt	5 Volt	5 Volt	ON	ON	ON	ON

IV. CONCLUSION

To overcome the weakness of Node MCU ESP32 for controlling the relay module, which is caused by the difference gap that is too far between the digital output Node MCU that can only release a 3-volt maximum and the relay driver that requires the trigger input amount of 5 volts, the addition of Arduino UNO as an additional driver module proposed. The Node MCU ESP32 is connected to Arduino by using serial communication protocol. The Arduino JSON library was also involved to wrap and encode the command to turn on and turn off relay in Node MCU ESP32. Arduino UNO module plays a role to decode the incoming JSON data that delivered by Node MCU to be a command to activate the digital Arduino Pins to be digital output for trigger the relay module. As a result, the command from ESP32 can be maintained by Arduino UNO to be 5 volts to trigger the relay module.

REFERENCES

Amoran, A. E., Oluwole, A. S., Fagorola, E. O., & Diarah, R. S. (2021). Home automated system using Bluetooth and an android application. *Scientific African*, *11*, e00711. https://doi.org/10.1016/j.sciaf.2021.e00711

Blanchon, B. (n.d.). *ArduinoJson. 2018*. Retrieved November 23, 2022, from https://arduinojson.org/

Bourhis, P., Reutter, J. L., & Vrgoč, D. (2020). JSON: Data model and query languages. *Information Systems*, *89*, 101478. https://doi.org/https://doi.org/10.1016/j.is.2019.101478

Ding, J., Nemati, M., Ranaweera, C., & Choi, J. (2020). IoT connectivity technologies and applications: A survey. *IEEE Access*, *8*, 67646–67673. https://doi.org/10.1109/ACCESS.2020.2985932

El-Sayed, W. T., Azzouz, M. A., Zeineldin, H. H., & El-Saadany, E. F. (2021). A Harmonic Time-Current-Voltage Directional Relay for Optimal Protection Coordination of Inverter-Based Islanded Microgrids. *IEEE Transactions on Smart Grid*, *12*(3), 1904–1917. https://doi.org/10.1109/TSG.2020.3044350

Hassan, C. A. U., Iqbal, J., Khan, M. S., Hussain, S., Akhunzada, A., Ali, M., Gani, A., Uddin, M., & Ullah, S. S. (2022). Design and Implementation of Real-Time Kitchen Monitoring and Automation System Based on Internet of Things. *Energies*, *15*(18). https://doi.org/10.3390/en15186778

Hermanu, C., Maghfiroh, H., Santoso, H. P., Arifin, Z., & Harsito, C. (2022). Dual Mode System of Smart Home Based on Internet of Things. *Journal of Robotics and Control (JRC)*, *3*(1), 26–31. https://doi.org/10.18196/jrc.v3i1.10961

Ivanov, D., Tang, C. S., Dolgui, A., Battini, D., & Das, A. (2021). Researchers' perspectives on Industry 4.0: multi-disciplinary analysis and opportunities for operations management. *International Journal of Production Research*, *59*(7), 2055–2078. https://doi.org/10.1080/00207543.2020.1798035

Khan, A. U., Khan, M. E., Hasan, M., Zakri, W., Alhazmi, W., & Islam, T. (2022). An Efficient Wireless Sensor Network Based on the ESP-MESH Protocol for Indoor and Outdoor Air Quality Monitoring. *Sustainability (Switzerland)*, *14*(24). https://doi.org/10.3390/su142416630

Lee, J., Anjos, E., & Satti, S. R. (2021). SJSON: A succinct representation for JSON documents. *Information Systems*, *97*. https://doi.org/10.1016/j.is.2020.101686

Liu, Z. H., Hammerschmidt, B., McMahon, D., Chang, H., Lu, Y., Spiegel, J., Sosa, A. C., Suresh, S., Arora, G., & Arora, V. (2020). Native JSON Datatype Support: Maturing SQL and NoSQL convergence in Oracle Database. *Proceedings of the VLDB Endowment*, *13*(12), 3059–3071. www.scopus.com

Mironov, V., Gusarenko, A., Yusupova, N., & Smetanin, Y. (2020). Json documents processing using situation-oriented databases. *Acta Polytechnica Hungarica*, *17*(8), 29–40. https://doi.org/10.12700/APH.17.8.2020.8.3

Nižetić, S., Šolić, P., López-de-Ipiña González-de-Artaza, D., & Patrono, L. (2020). Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future. *Journal of Cleaner Production*, *274*. https://doi.org/10.1016/j.jclepro.2020.122877

Parab, R., & Prajapati, S. (2019). IoT based relay operation. *International Journal of Engineering and Advanced Technology*, *9*(1), 6515–6520. https://doi.org/10.35940/ijeat.A1415.109119

Perkasa, R., Wahyuni, R., Melyanti, R., Herianto, & Irawan, Y. (2021). Light control using human body temperature based on arduino uno and PIR (Passive Infrared Receiver) sensor. *Journal of Robotics and Control (JRC)*, *2*(4), 307–310. https://doi.org/10.18196/jrc.2497

Rahman, M. S., Peeri, N. C., Shrestha, N., Zaki, R., Haque, U., & Hamid, S. H. A. (2020). Defending against the Novel Coronavirus (COVID-19) outbreak: How can the Internet of Things (IoT) help to save the world? *Health Policy and Technology*, *9*(2), 136–138. https://doi.org/10.1016/j.hlpt.2020.04.005

Shen, G., Zhang, J., Marshall, A., Peng, L., & Wang, X. (2021). Radio Frequency Fingerprint Identification for LoRa Using Deep Learning. *IEEE Journal on Selected Areas in Communications*, *39*(8), 2604–2616. https://doi.org/10.1109/JSAC.2021.3087250

Siu, K. K. M., Ho, C. N. M., & Li, D. (2020). Design and analysis of a bidirectional hybrid DC circuit breaker using AC relays with long life time. *IEEE Transactions on Power Electronics*, *36*(3), 2889–2900. https://doi.org/10.1109/TPEL.2020.3013612

Sun, G.-., Yun, J.-., & Cheon, M.-. (2021). Parallel Switch Configuration for High Voltage DC Switching to Secure PV Power System Safety. *Transactions on Electrical and Electronic Materials*, *22*(1), 108–113. https://doi.org/10.1007/s42341-020-00279-9

Uguru-Okorie, D. C., Adebimpe, A. M., Oni, T. O., & Omoyemi, P. (2022). Development of an automated bitter leaf processing machine. *Scientific African*, *17*. https://doi.org/10.1016/j.sciaf.2022.e01311

Xu, X., He, C., Xu, Z., Qi, L., Wan, S., & Bhuiyan, M. Z. A. (2020). Joint Optimization of Offloading Utility and Privacy for Edge Computing Enabled IoT. *IEEE Internet of Things Journal*, *7*(4), 2622–2629. https://doi.org/10.1109/JIOT.2019.2944007

Yahya, O. H., Alrikabi, H. T. S., & Aljazaery, I. A. (2020). Reducing the data rate in internet of things applications by using wireless sensor network. *International Journal of Online and Biomedical Engineering*, *16*(3), 107–116. https://doi.org/10.3991/ijoe.v16i03.13021

Zheng, T., Ardolino, M., Bacchetti, A., & Perona, M. (2021). The applications of Industry 4.0 technologies in manufacturing context: a systematic literature review. *International Journal of Production Research*, *59*(6), 1922–1954. https://doi.org/10.1080/00207543.2020.1824085