

***Microprocessor***

***Final Project***

***Team 15***

|  |  |  |
| --- | --- | --- |
| Name | Section | ID |
| 1)Ahmed Mahmoud Abo Zeid | 2 | 40 |
| 2)Mohamed Ahmed Ali Hassan | 5 | 168 |
| 3)Yehia Ahmed Ibrahim Shahin | 8 | 271 |
| 4)Yehia Ehab Orfy Mohamed | 8 | 272 |
| 5)Youssef Mohamed Elsayed Mehdawy | 8 | 280 |

Table of Contents

[Introduction: 1](#_Toc62280252)

[**General Overview:** 1](#_Toc62280253)

[**Software Used:** 1](#_Toc62280254)

[Hardware: 3](#_Toc62280255)

[**Components:** 3](#_Toc62280256)

[**Schematic:** 4](#_Toc62280257)

[**PCB Design:** 7](#_Toc62280258)

[Software: 8](#_Toc62280259)

[**General Explanation:** 8](#_Toc62280260)

[**Mobile application Code:** 10](#_Toc62280261)

[**ESP32 Code:** 12](#_Toc62280262)

Parking System

# **Introduction:**

## **General Overview:**

A lot of studies indicate that garage workers are exposed chemicals that causes a wide range of biological effects depending on the level and duration of exposure. Biological monitoring of chemical exposure in the workplace has become increasingly important in the assessment of health risk as an integral part of the overall occupational health and safety strategy.

The idea of our project is to limit the time of presence inside of the garage or at least isolate them from the parking areas that have highest chemical exposure areas.

The project is simply a simulation for a parking system in which a garage has a number of empty locations that are displayed on a control board and a mobile application.

The system has 2 modes:

1. **Manual mode:** where you can increment, decrement and reset the display (number of empty locations in the garage) from either the control board using 3 push buttons or from the application where there are 3 similar buttons.
2. **Automatic mode:** where the display (on both the application and the control board) is changed automatically using 2 IR sensors that indicate whether a car is leaving or entering the garage.

## **Software Used:**

1. **Thonny Python IDE:** used for programming our microcontroller (ESP 32).
2. **MIT App Inventor:** Used for the development of our mobile application.
3. **Altium Designer:** Used for schematic and PCB design.

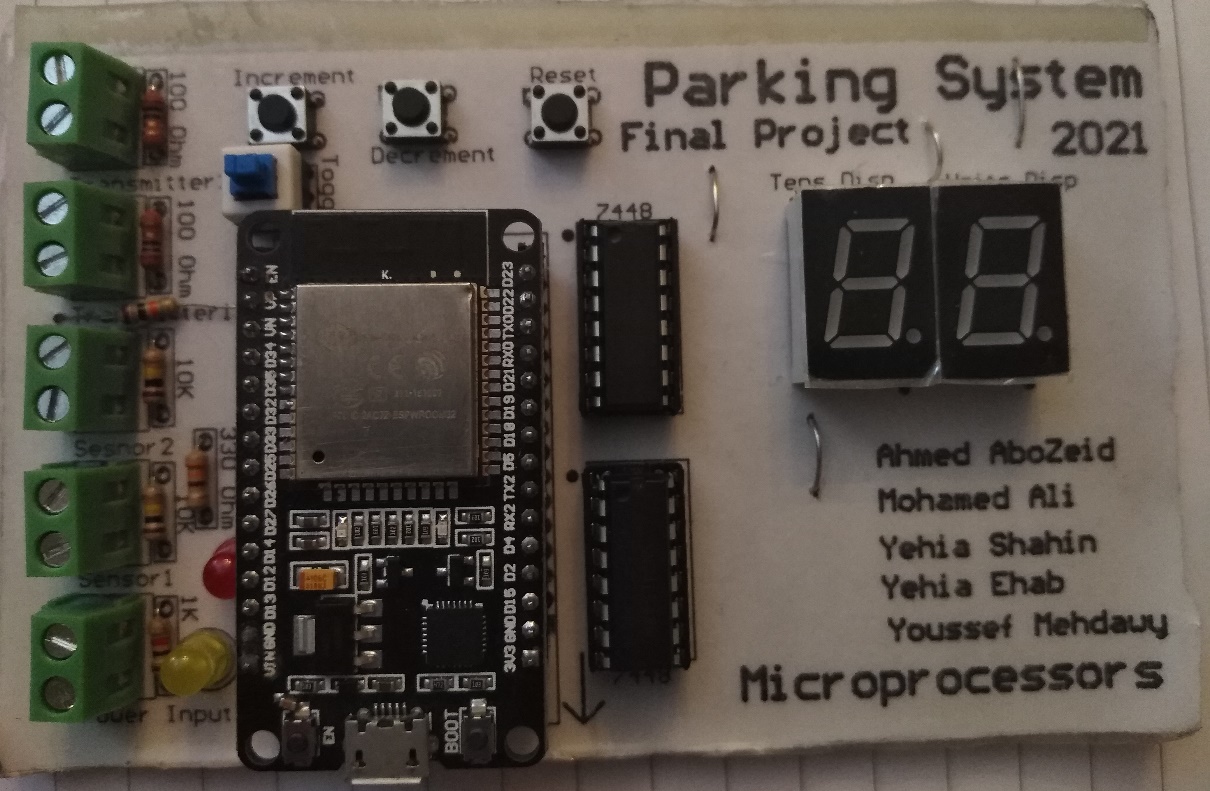




Figure 1: Control Board.

Figure 2: Mobile Application.

# **Hardware:**

## **Components:**

* Microcontroller (ESP32).
* 2 BCD to seven segment decoders (7448).
* 2 seven segment displays (common cathode).
* 3 push buttons (4 pins).
* 1 on-off push button (6 pins).
* 30 female headers.
* 5 terminal blocks (2 pins).
* 2 LEDs (red and yellow).
* 4 resistors (0 Ohm)
* 2 resistors (100 KOhm).
* 2 resistors (10 Ohm).
* 1 resistor (10 KOhm).
* 1 resistor (1 KOhm).
* 1 resistor (330 Ohm).
* Mobile (for application).

## **Schematic:**

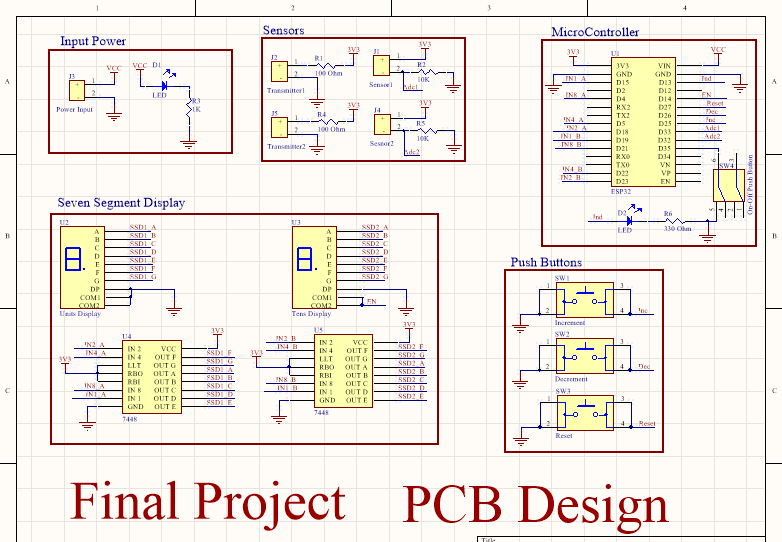


Figure 3: Schematic of our control board.

First, we tested every part separately on breadboards to check the validity of the components and also to test the logic of the circuit, and then we were able to merge all parts into this design.

The schematic is divided into 5 blocks as follows:

* **Input Power:**

The board can be powered from external power source (5-12V) that enters directly to the microcontroller and there is also an LED to indicate the presence of voltage.

* **Push buttons:**

There are 3 push buttons that are tied up to ground and their other sides are connected to the micro controller

When the button is pressed the controller reads logic 0 (Low).

There is no need for pull up resistor as the pins in the micro controller can be set to be pull up.

There is no need for a debouncing circuit as we already implement software debouncing in the code.

* **Sensors:**

As shown in the schematic there are 4 terminals in this block, there are 2 for the IR sensors (receiver) and another 2 for the IR Transmitters.

The IR receiver senses the IR rays, so we also use IR transmitter as a source of these rays.

Figure 4: IR receiver and transmitter.

The sensor and the transmitter are mounted beside each other. So, when there is no object in front of the sensor all the rays emitted from the transmitter travels away from the sensor and it reads a minimum value.

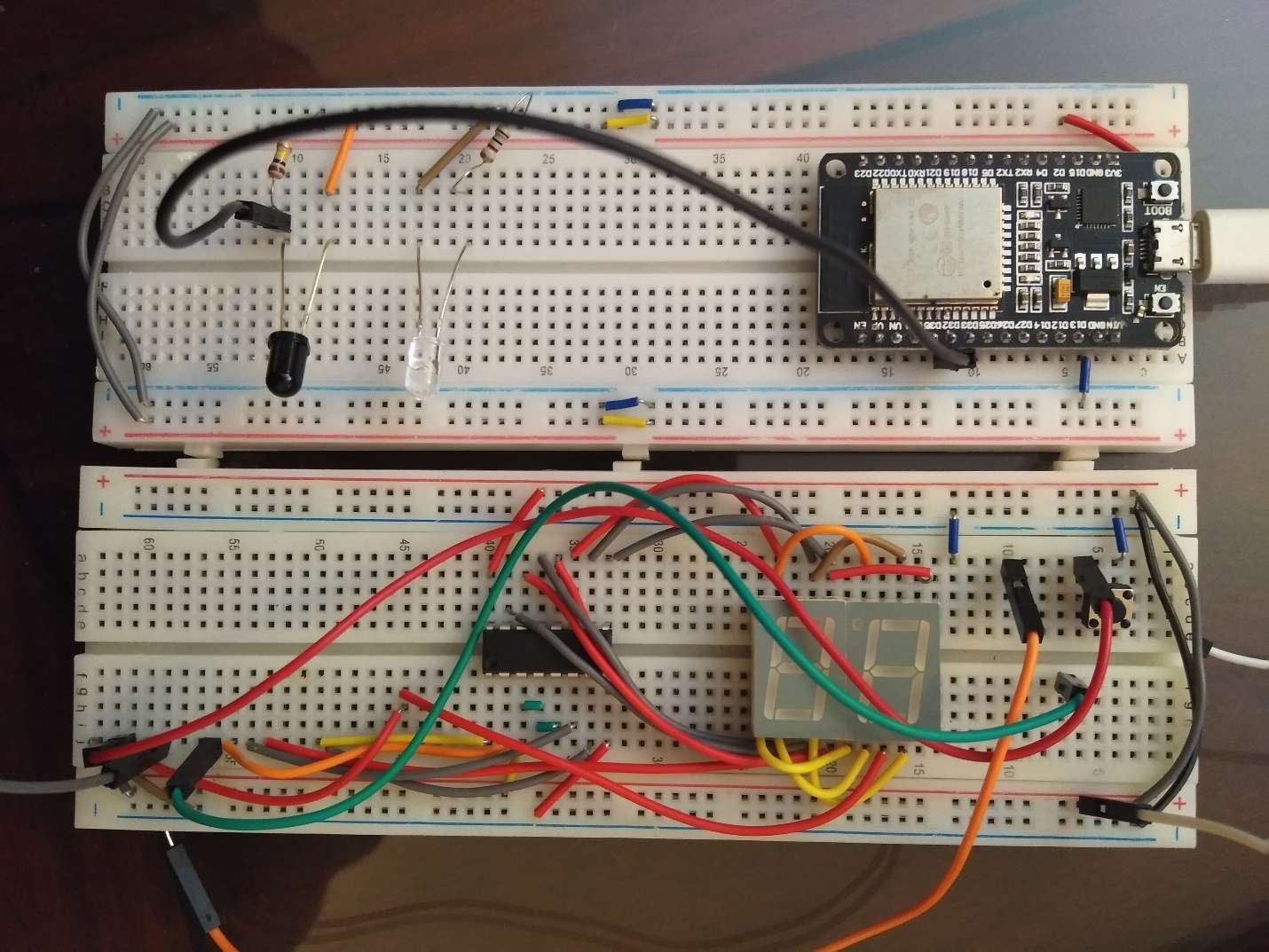
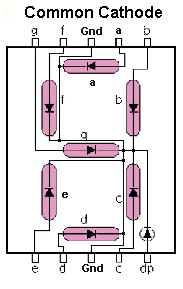


Figure 5: Mounting of sensor and transmitter.

When an object passes most of the IR rays sent from transmitter hit the object and travel back to sensor so it reads a very high value.

Finally the circuit of the transmitter is just like a normal LED, and that of the sensor is voltage divider, so the voltage change represents the change of level of IR and is read by microcontroller using ADC peripheral.

* **Seven Segment Display:**

We used 2 seven segment displays (common cathode) in order to display more than one digit.

The problem is each seven segment needs 7 or 8 pins from our microcontroller which means we need a total of 14 or 16 pins and we don’t have this number of pins available.

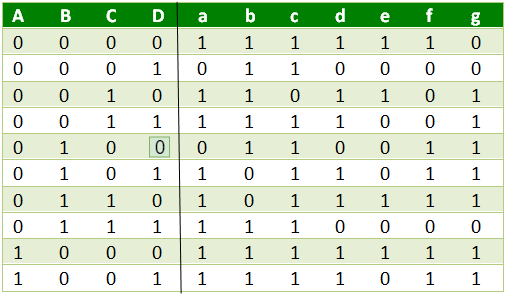
So, we used 2 BCD to seven segment decoders (7448) as well, because each decoder needs only four signals to interface with, which sums up to be 8 pins needed from the microcontroller.

Figure 6: Seven segment pinout.

So, the final circuit will be as shown in the schematic block, 2 BCD to seven segment decoders with their input connected to the microcontroller and their output is connected with the 2 seven segments (One will be for the units display and the other will be for the tens).

Figure 7: Truth Table of seven segment decoder.

Since we want the units display to be always active, we connected its common pin with the ground. However, the tens display will be only active if the number is greater than 9 so we connected its common pin with the microcontroller so we can enable it in the required time.

Finally, the decimal point pin in both seven segments is connected to ground since we don’t need any floating numbers.

* **Microcontroller:**

Finally, all the control signals in the previous blocks will be connected to the microcontroller.

In addition to that, we connected an LED to one of the controller pins to be used as indication for any desired action. In the program we used it as an indication that there are no remaining parking slots in the garage (when number reaches zero the LED turns on).

We also added an on-off switch to toggle between the two modes of operation of our program (manual and automatic modes).

## **PCB Design:**

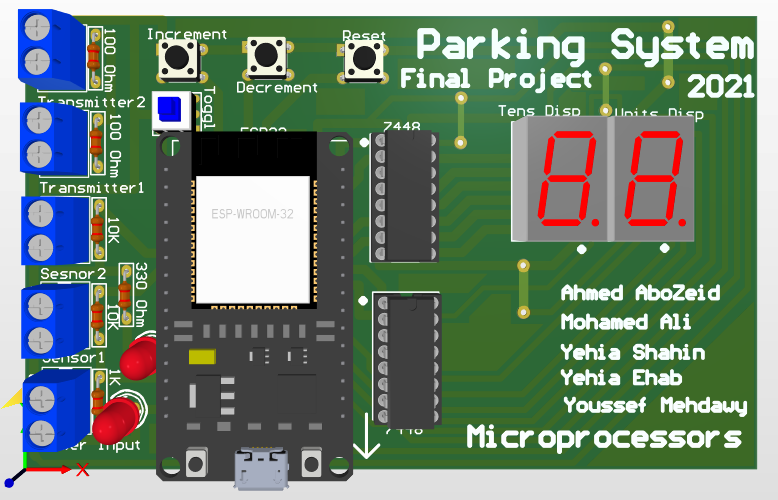


Figure 8: 3d model of our control board.

Finally, after all steps of testing, debugging and editing we reached to the final design of our control board and proceeded further to the PCB design as shown in figure 8 and we were able to fabricate the board successfully.

# **Software:**

## **General Explanation:**

Write 2

Write 1

**Mobile  
Application**

**Micro  
Controller**

**Server**

Read 2

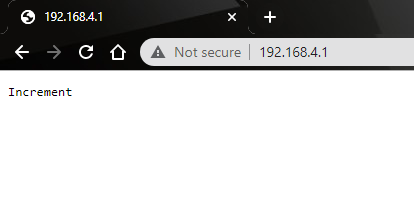
Read 1

Figure 9: Communication between ESP and Application.

As shown in figure 9, we need to have a 2-way communication between the controller and the application, which means every side should be able to write and read to and from the server.

We will initialize the ESP 32 as an access point so it will act as a router and we will open a socket with its IP address so that will be the server that we communicate through.

So, we have 4 operations that we have to handle:

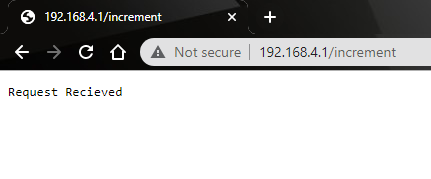
* **Write 1:**

The controller will update the server page and will print either increment decrement or reset that will be read by mobile application.

Figure 10: Server page during write request.

* **Read 1:**

The application will read the word sent from the controller and will take action according to the word sent.

* **Write 2:**

The application will access the server with the IP address of the controller and will send a message in the request to be read by controller.

Figure 11: Server Page during read request.

* **Read 2:**

The controller will get the link request, for example in figure 9 it was 192.168.4.1/increment and will take action according to the word sent after the IP.

* The implementation of all previous operations are going to be explained in the code in the following sections.

## **Mobile application Code:**

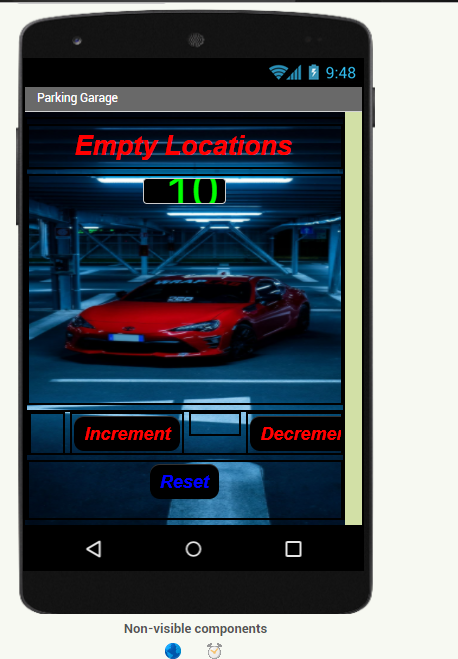


Figure 12: App layout from MIT App inventor.

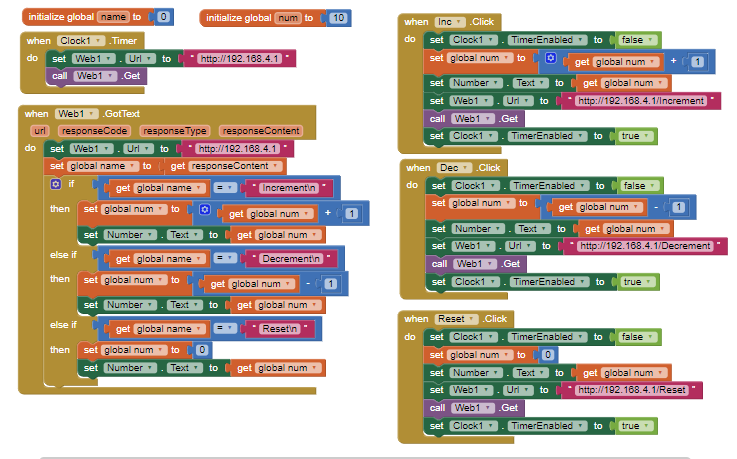


Figure 13: App code from MIT App inventor.

First, we designed the application layout. We tried to make user friendly and as simple as possible to be easy in its interface.

We have 2 invisible components: web and clock.

Then moving to the code part, as shown on figure 13, the right part is for writing on server and the left part is for reading.

* **Read part:**

As explained in page 8, the application needs to access the server page to read message sent from controller. But the problem is we don’t know when the message will be sent.

So, we initialized a clock (acting as a timer) of period=500ms. The timer will access the server at the end of each period and then we application will check if there is a message sent from the controller.

* **Write part:**

The write part will be much easier as we will only take action if one of the buttons was pressed.

We will first disable timer before each write request and then enable it again at the end of the request.

The message will be sent according to the button pressed and as explained in page 9 it will be sent beside the IP of the controller: 192.168.4.1/message.

## **ESP32 Code:**

First, we designed our code to be in a certain architecture where we can debug and edit it easily.

**Micro  
Controller**

**Libraries**

**Main  
Script**

Figure 14: Architecture of code.

As shown in figure 14, we designed libraries for each component we are going to interface with and each library was tested separately and then we used these libraries in the main script.

We have 8 libraries each one contains needed functions to interface with the needed components.

1. **LED.py:**

Library interfaces with on board LEDs

We have 2 LEDs: the indication LED connected in the control board and the built-in LED in ESP connected to pin 2.

We first initialize their pins as output using Pin module.

And then we define On and Off functions for both LEDs.

1. **SW.py:**

Library interfaces with switches.

We have 2 type of switches: push buttons and on-off switch.

We first initialize their pins as input using Pin module

We have only one on-off switch that toggles between two modes of operation and we have one function GetToggleValue for it that reads its current value.

We have 3 push buttons and we have one function GetKeyStatus that reads their values and return a list containing status of 3 switches (we also implemented software debouncing in this function).

1. **SSD.py:**

Library interfaces with seven segments.

We have 2 seven segments (total of 8 pins due to 7448) and we control the enable of the second one as explained in page 6. So, we have total of 9 pins.

We first initialize the pins as output using Pin module.

Then we have Enable and Disable function to control the second seven segment.

And we also have SendNumberUnits function to send number on first seven segment.

Another function is SendNumberTens to send number on second seven segment.

And the last function is SendNumber which handles the number if it was greater than 9 and divides it to be displayed on both seven segment displays.

1. **Sensor.py:**

Library interfaces with IR sensors.

We have 2 IR sensors to indicate whether car is entering or leaving the garage.

We first initialize their 2 pins connected with microcontroller using ADC and Pin module.

Then we have one function Read that gets the reading of both sensors and returns 1 if sensor reading passes threshold value (threshold value is determined experimentally after installing the sensors along with IR transmitters).

1. **Wifi.py:**

Library used to initialize the ESP32 as access point (router).

We used the network module.

We have 2 functions Connect and Disconnect functions to control Wi-Fi status on the controller.

1. **Server.py:**

Library establishing communication with server.

We used the usocket module.

We have 3 write functions: write\_inc, write\_dec, write\_res that raises flags to indicate desired operation that will be executed in listen function.

The last function is listen which establishes constant connection with server and send read or write requests according to raised flag.

1. **TIM.py:**

Library used for timer initialization.

We used the Timer module.

First, we have the Init function that initializes the timer and takes ISR function from user and executes it in the Timer\_ISR function after making it global using Global\_Function.

The last function is Disable that deactivates timer.

1. **INT.py:**

Library used for interrupt initialization.

First, we have the Init function that initializes the 3 Interrupts of the 3 push buttons and takes 3 ISR functions from user to be executed in Inc\_ISR, Dec\_ISR and Res\_ISR functions after making them global using Global\_Function1, Global\_Function2 and Global\_Function3.

In the ISR functions we disable the interrupt at the beginning of the ISR and then enable it at the end to eliminate the bouncing effect of the switches.

The last function is Disable that deactivates all 3 interrupts.

* **Main.py:**

This is the main script where we use all the previous functions.

First, we import all the 8 libraries we created and then define ISRs of the interrupts that we will use in the code.

First, we have Timer\_ISR that is responsible for sending the sensor data on the server each 500ms.

Then we have Inc\_ISR, Dec\_ISR and Res\_ISR that are responsible to send to the server if any of the push buttons was pressed.

We then initialize the timer and pass the Timer\_ISR function to it.

Finally, before entering main loop we activate the Wi-Fi in the ESP32 and define:

1. The initial value of the number to be displayed on the seven segments.
2. The maximum value that this number can reach (maximum number of empty locations in garage).
3. Write flag that will be used in automatic mode.

After entering the main loop, we first check the on-off switch value to determine the mode (whether its manual or automatic)

* If we are in manual mode, we initialize the 3 interrupts of 3 push buttons and pass the 3 ISR functions defined before main loop.

Then we establish a connection with server and check for messages received from application and take action upon this message.

The ISR of the 3 defined interrupts will be responsible for sending data on server if any switch was pressed.

* If we are in automatic mode, we disable the interrupts as we don’t need to send data from switches.

We get the reading from IR sensors and take action if any car was detected entering or leaving the garage.

We also establish a connection with the server to send data upon any signal taken from sensors.

The ISR of the Timer will be responsible for sending data on server in this mode.

The final action in the loop is to check whether the number reached 0 or not and then it will activate an LED on reaching 0 indicating that the garage is full and no more cars can enter.