كلية علوم الحاسب والمعلومات
College of Computer and Information Sciences

CS330–Computer Networks Project

Car Parking Management System
a Client/Server Application

Students Names:

Name#1: **440013412 - عمر الخويتم**

Name#2: **440017330 -عبدالرزاق الثويني**

Name#3: **440015920 - عبدالإله ناصر الجربوع**

Section: 171

Date of Submission

2022/5/19

Instructor's Name:

Dr. Abdulaziz I. Almohimeed

Project Description:

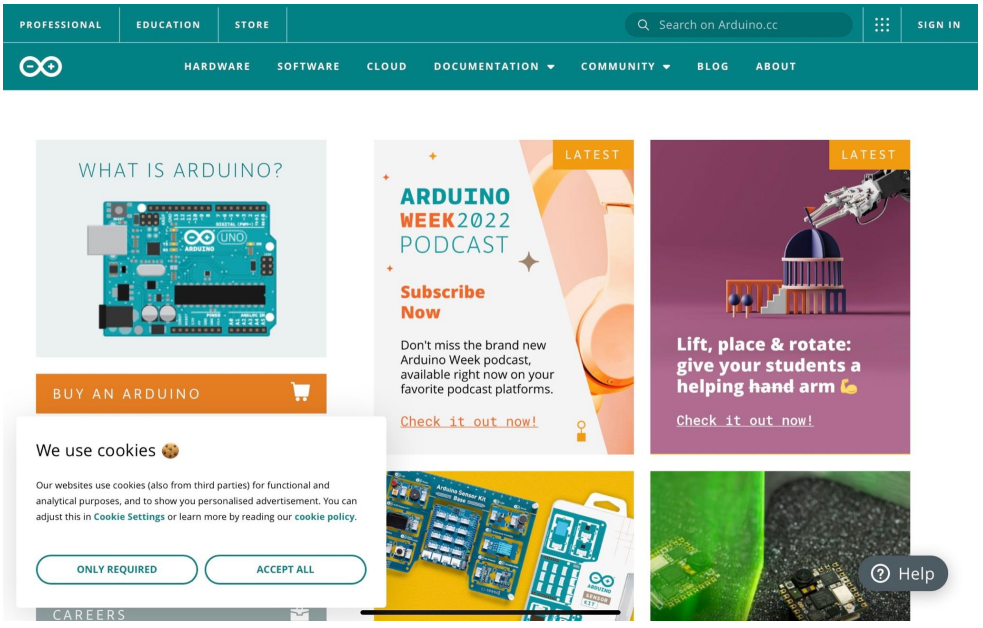
This project is going to focus on building a parking spot sensor from the ground up. We are going to cover everything from setting up the IDE and the setting up for the arduino to the code and how the server works, until we have a fully working system.
we choose this project because we live in a big city with alot of people so we know how it feels to try to find a parking spot when there is’nt any and there is no parking sensors installed.
we wanted to try and build an efficient and robust System but at the same time cheap to implement, and with that we encourage more institutions to add it to their parking places!

1-Setting up the Programming Environment

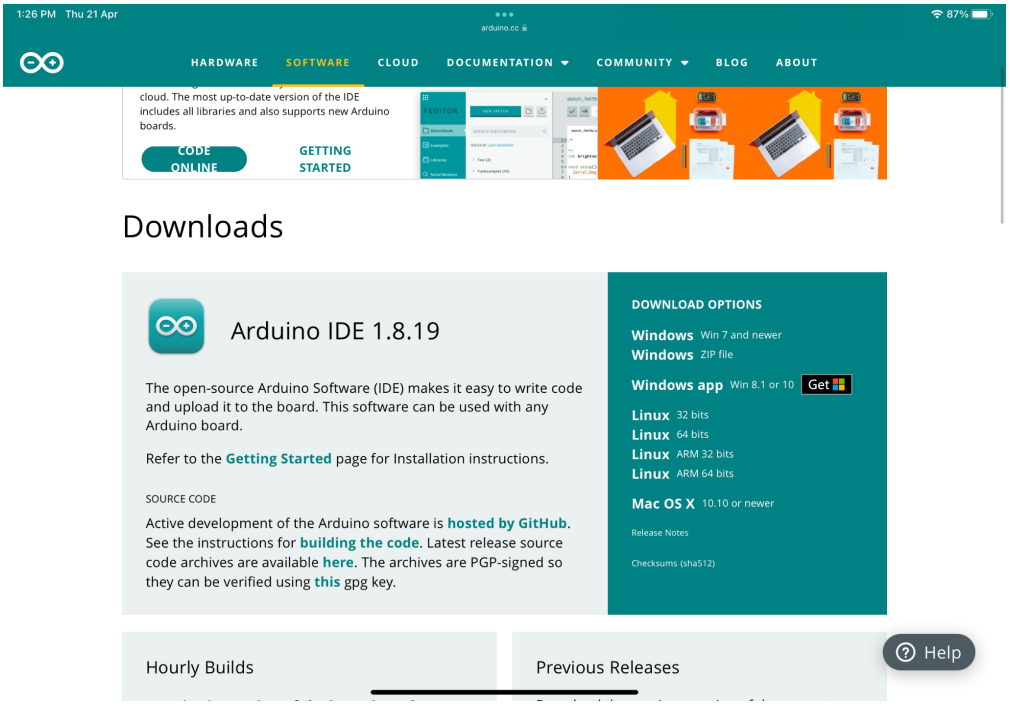
1.1-Programming languages

The programming languages we are going to use are c/c++ to program the Arduino board. and we are going to use java to build the program that is going to receive the messages “The Server” from the Arduino because we are familiar with it and we have been using it for the last three years.

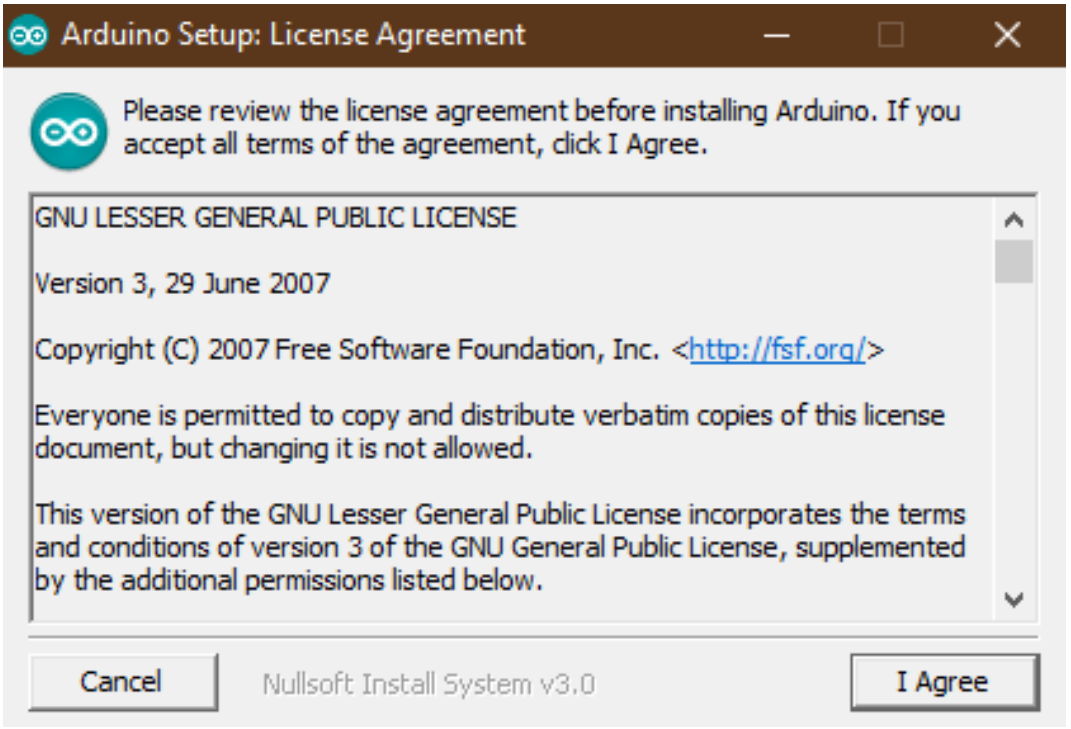
To download the Arduino IDE we have to go Arduino Then press Software



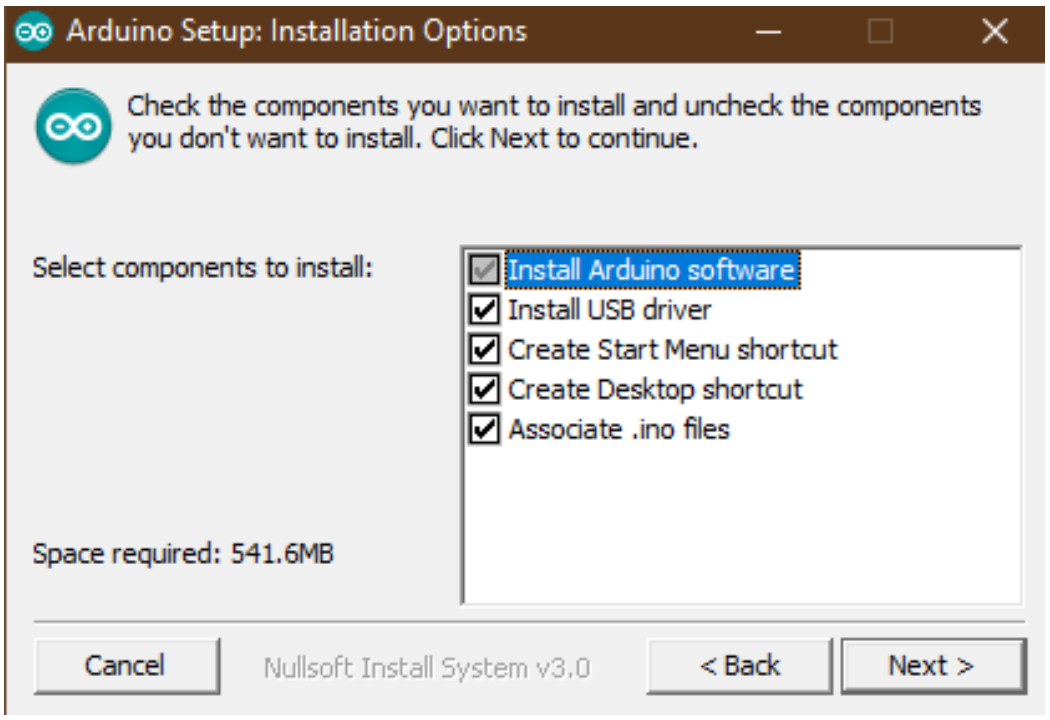
choose the operating system you are using and then the download will begin.



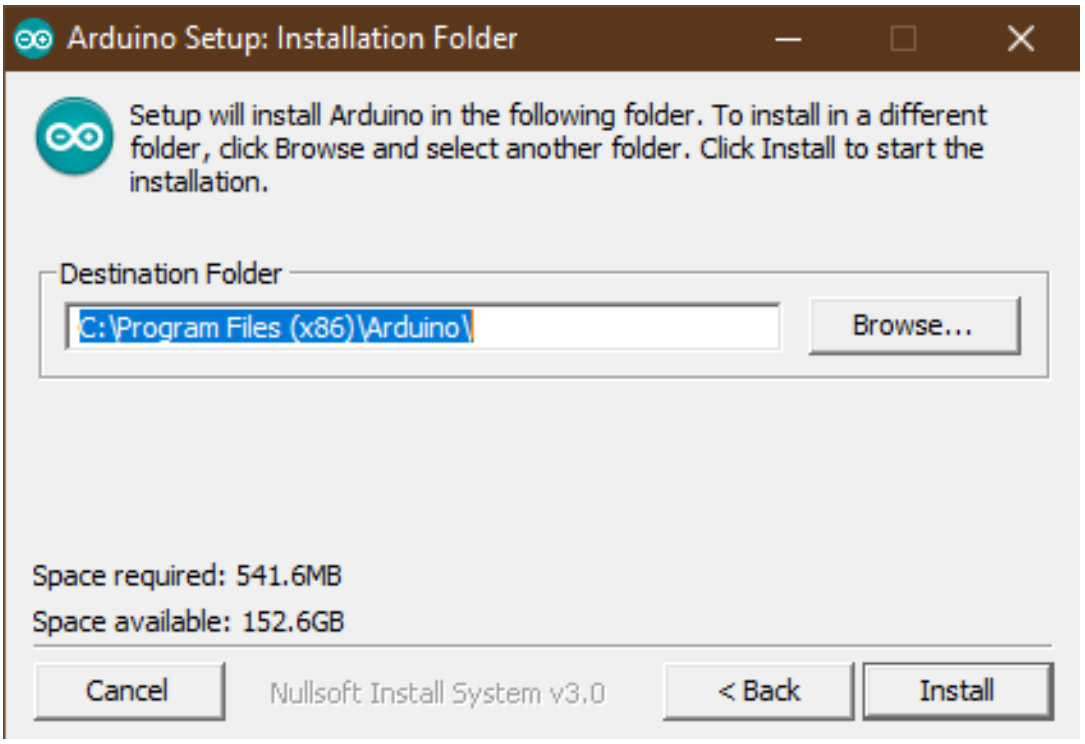
After the setup file is downloaded, run it and agree to the License



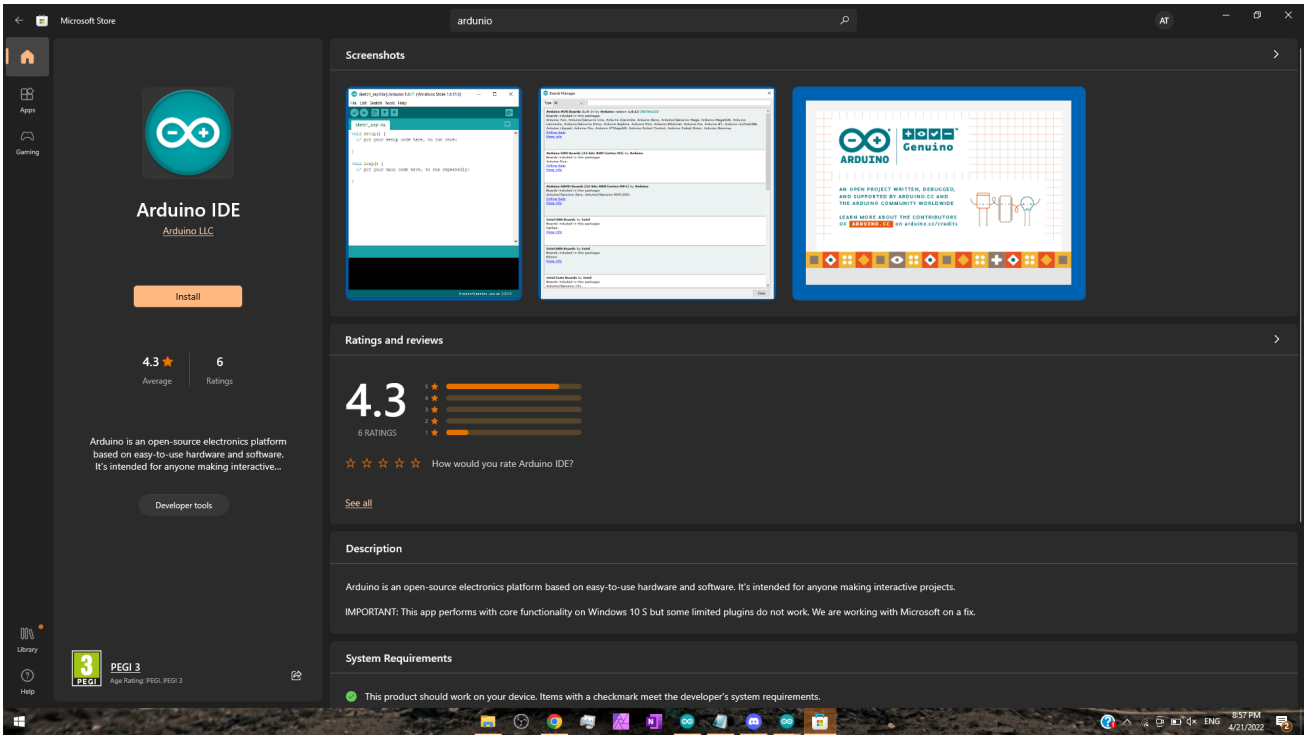
Next, select the choices you want



Finally chose where you want to install the program and press install

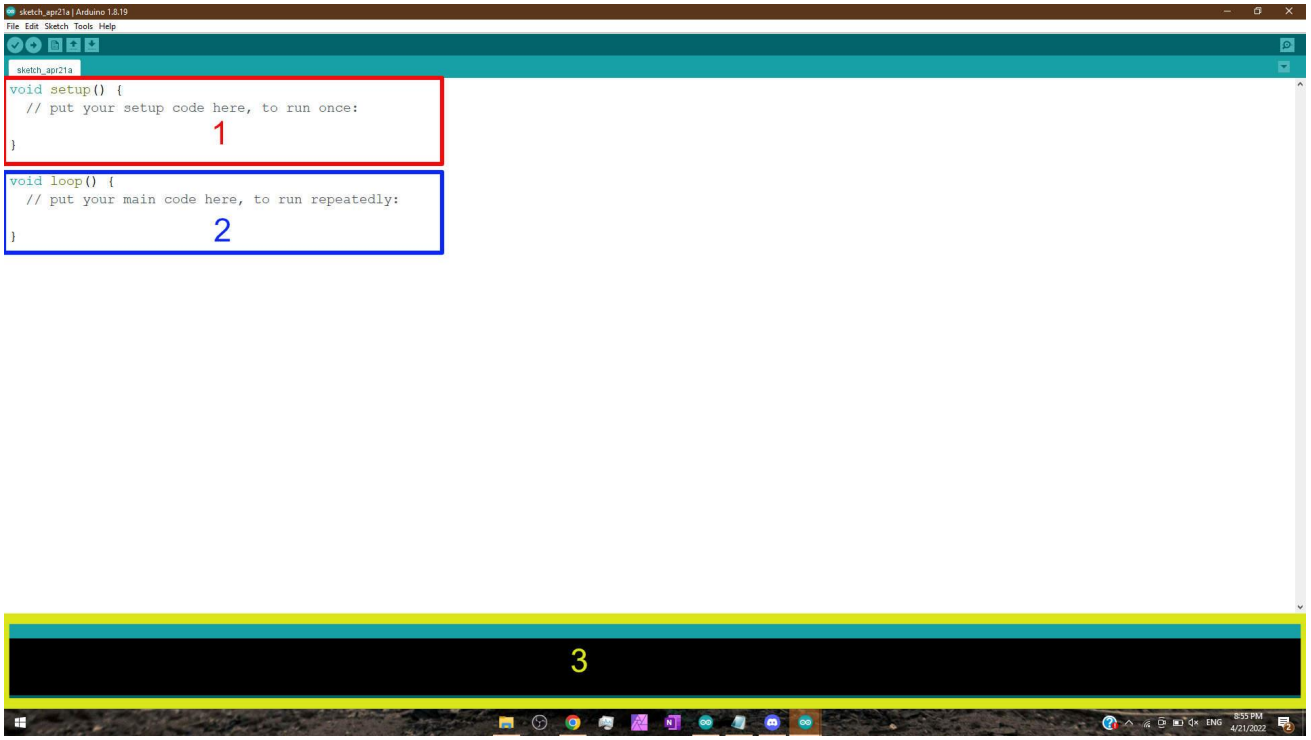


If you are using windows 8.1 or newer you can download the IDE from the windows store by going to the Microsoft store and searching Arduino IDE



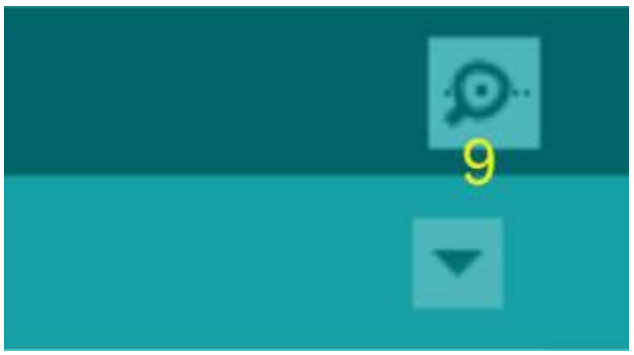
1.2 program user interface

The user interface of Arduino's IDE is simple, easy to use, and beginner-friendly



The red area is for the code that will run when the Arduino first boots, it enters this part **only** once. After it finishes the setup, it will go to the blue area. This method will run in a loop and execute the commands until the Arduino turns off, this is where you are going to write most of the code that is going to be executed. The yellow area is where you are going to see if there are any errors or the uploading percentage and any terminal output.

On the top left-hand corner, there are 5 buttons. number 4 is called Verify, and what it does is compile the code and check for any missing libraries and syntax errors. number 5 is called Upload and is for uploading the code to the board, before uploading the code it has to check if there are any errors just like the verify button. Button number 6 is called New. lets you open a new file in a separate window. button number 7 is called Open. What it does is that it lets you open previous files you worked on or open pre-loaded files that show you how to use functions. and button number 8 lets you save the file you are working on.



on the top right corner on the same bar, there is a Serial monitor. It gives you the ability to see what the Arduino board is outputting so you can print special messages that will help you with either debugging or having a separate window to see the output of your Arduino board when everything is happening in real-time.

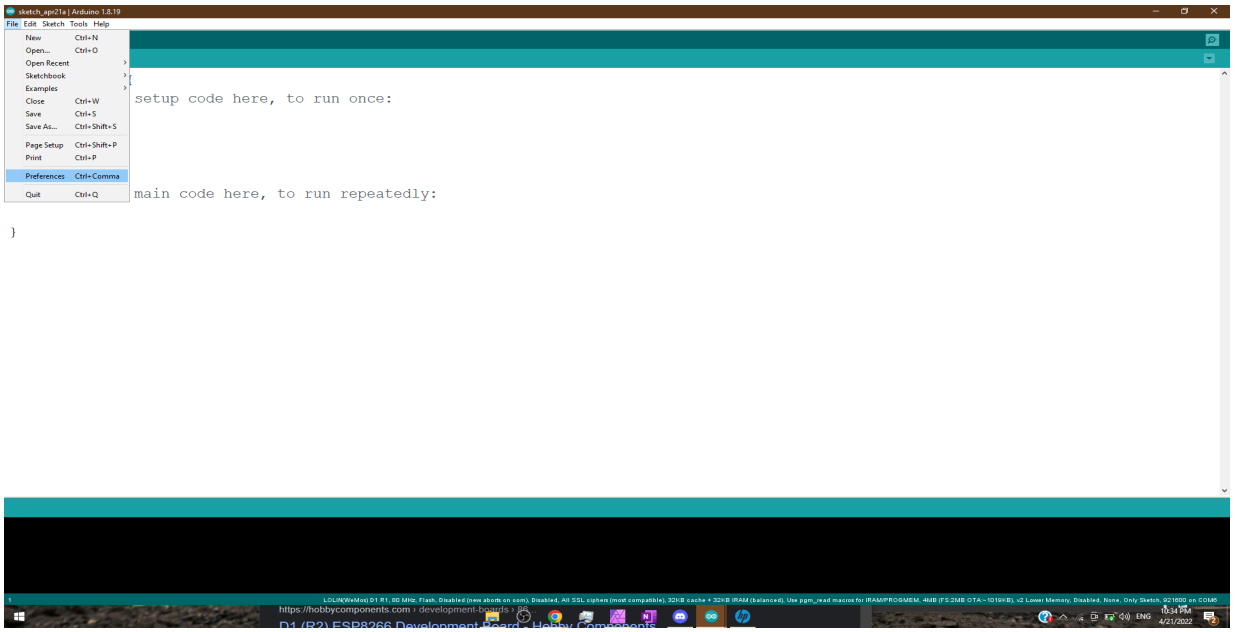
1.3- Setting up The IDE

Arduino boards come in many different shapes and sizes, and a lot of them are not made by Arduino. So if you want them to work you have to download their drivers and board managers. For example, the board we used in our project is called WeMos D1 R1. It is exactly like the Arduino Uno but it was wifi modular and based on ESP8266. To download the board manager we have to copy this link

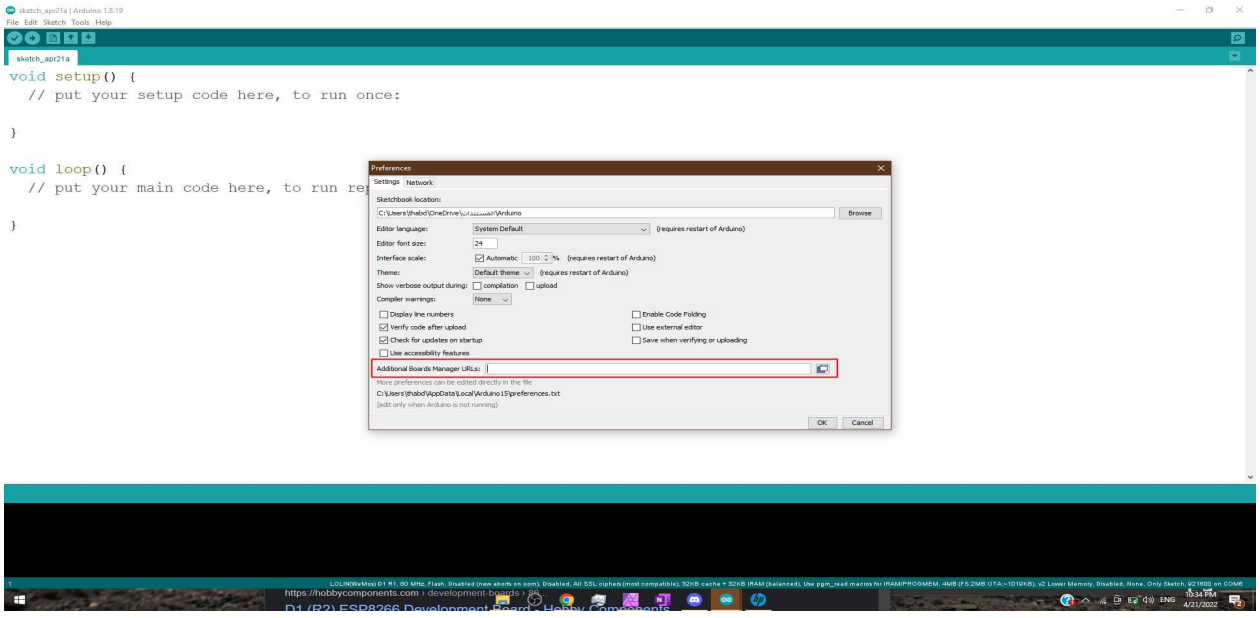
http://arduino.esp8266.com/stable/package_esp8266com_index.json

Then go to File-> preferences then past the link in the additional board manager URLs box

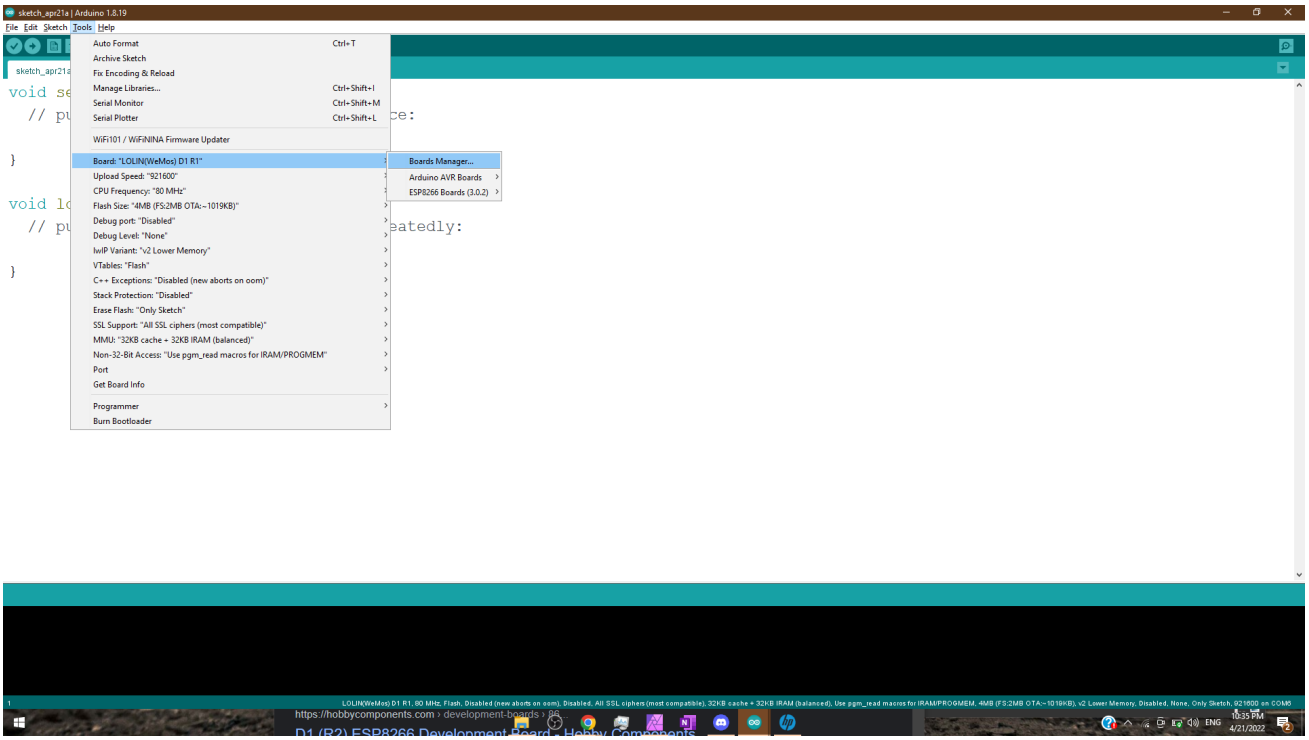
1-



2-



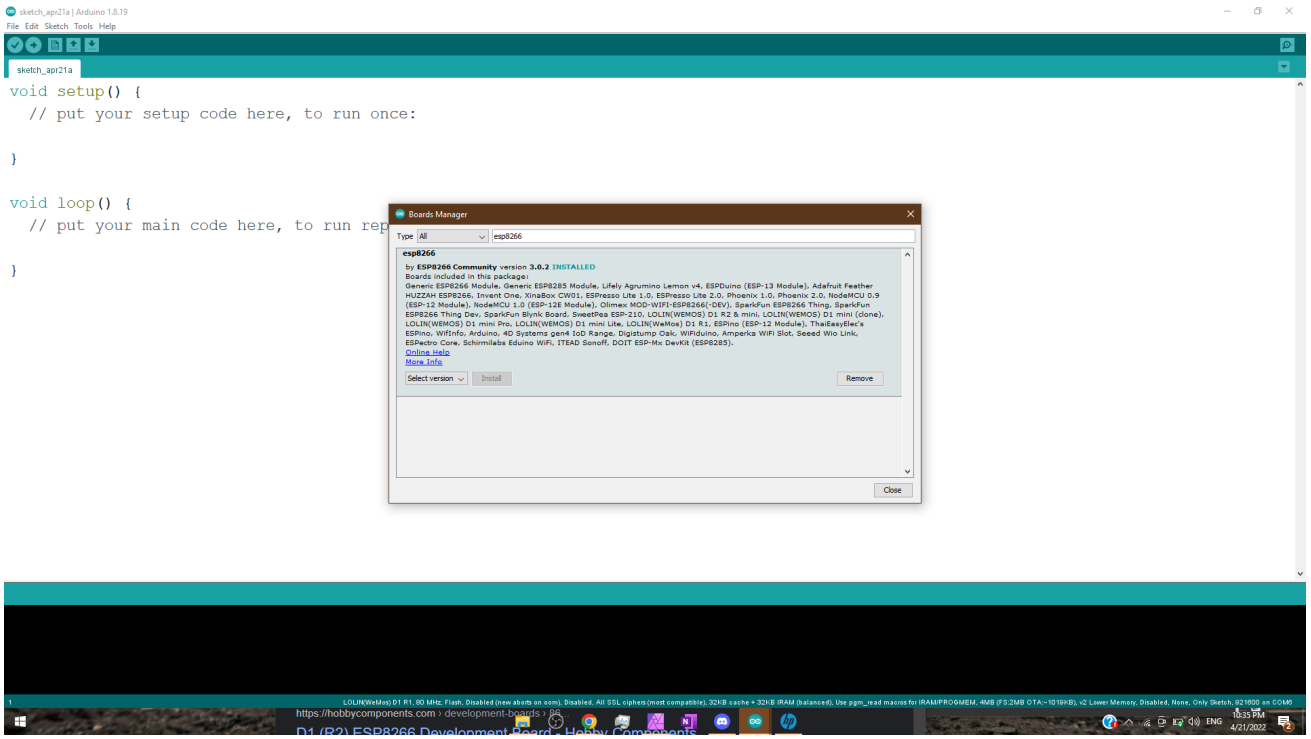
To download the board manager we have to go to tools -> board -> board manager and in the search bar we write ESP8266 and download the latest version of the board manager



3-

Because I already downloaded it, it appears in the board manager.

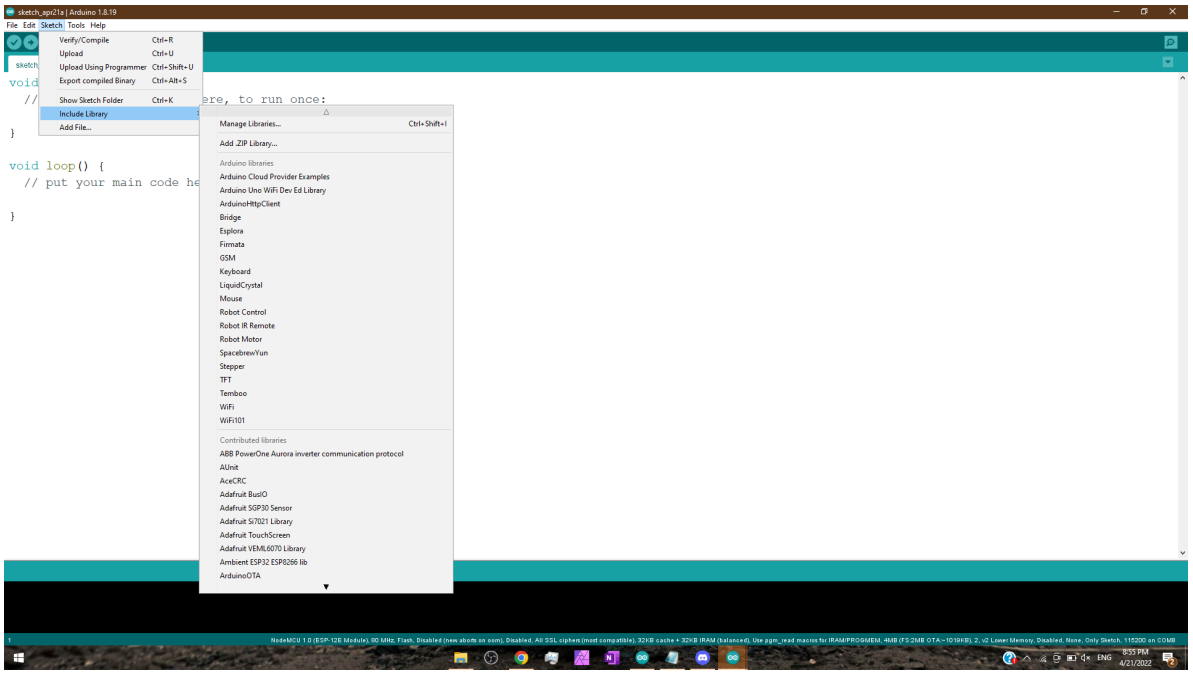
4-



And after that, we close the board manager window and go to tools-> board manager -> ESP8266 boards. In my case, I will choose LOLIN(WoMos) D1 R1. Now we are ready to start writing codes.

1.3.1 How to Download Libraries

When you first download the IDE it is only going to come with the necessary libraries to run the boards. So if you want to add libraries you will have to go to Sketch-> include library, if you have the library downloaded on your computer you are going to choose Add . ZIP library**(Note it has to be a zip file).** else you will choose Manage libraries and write the library name in the search box just like we did with the board manager.



1.4-Used Libraries

The libraries we had to use while working in the Arduino were ESP8266WiFi so we can connect the ESP board to the wifi and also connect to the server using socket programming, and the second library was NewPing, we need it to get the distance reading from the ultrasonic sensor and any other function that is associated with the sensor.

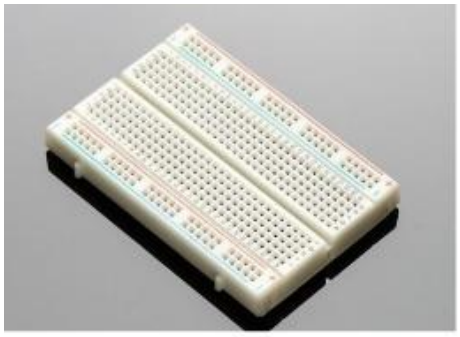
On the server-side, we used java and the following libraries,

java.net for the core of socket programming and use of sockets. java.io for any input/output that is associated with the server and how to get input from the client and so on.

2- board Hardware, setup, and programming

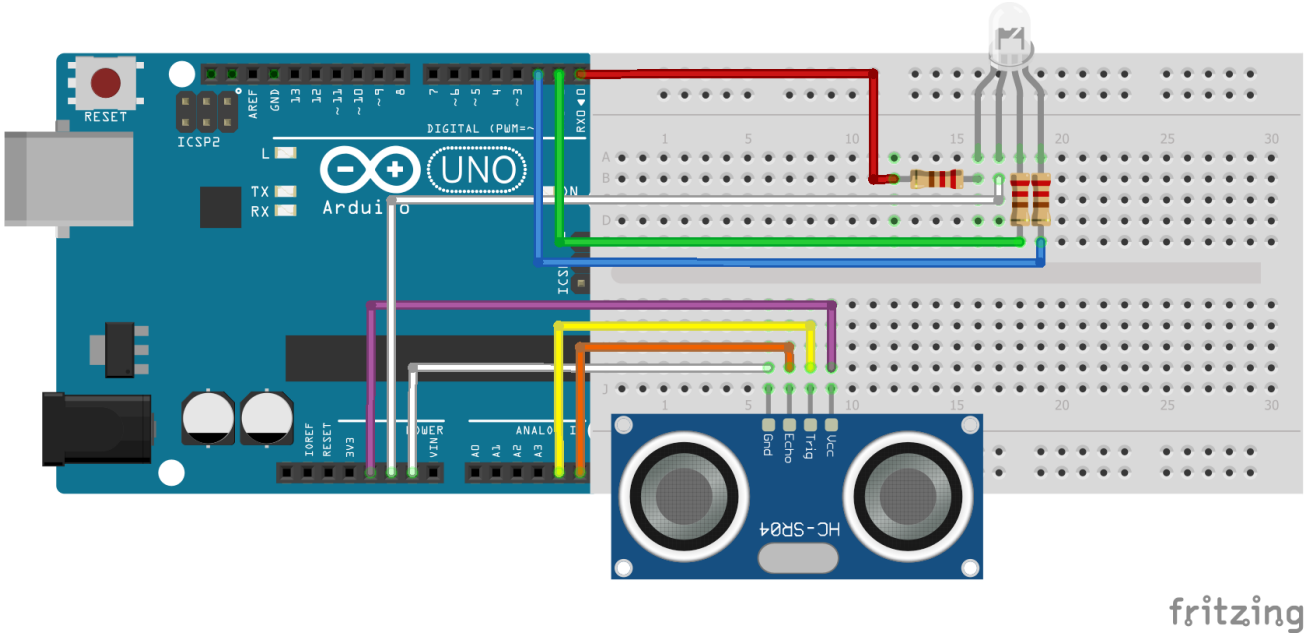
2.1- Hardware

To start with the project we first needed a couple of items which were the WeMos D1 R1 board, wires, RGB LED light, breadboard, and HC-SR04 ultrasonic sensor.






2.2- assembly



***The Board shown is not the real board and the pins used are not in the same places but the result is the same for visualizing purposes only.**

1 WeMos D1 R1 - 1 Ultrasonic Sensor - 1 RGB LED - 3 Resistors - 8 Wires - 1 micro-USB

We used Fritzing, a program for the visualizing and prewiring and it helped us with the hundreds of boards and electrical components, and an easy-to-use interface with a drag and drop way of handling things.
We first found out the best way to wire the board and then drew it in the program multiple times until we were satisfied with the result.
Then we started wiring the actual board and started testing if it worked or not.
Lastly, when we finished with the coding we uploaded the code to the board and tested it.

3-Steps for TCP/IP socket programming for client-server connection

We used C/C++ for Arduino and Java for the server socket programming to implement the client-server communication over TCP/IP protocol.
Java.net which is being used for the server is using TCP/IP, and we didn't need to make our own protocol obviously because we did not have any extra specifications and the TCP/IP protocol covers all our needs.
We used the TCP/IP to guarantee that the signal will be received by the server because we need the signal to be sent, delivered, and not thrown. the server will lose an important message and that will result in a failure in the main functionality of the system.
in the resources below we learned how to adapt the socket programming in the Arduino using the ESP8266WiFi library and how to set up the client and send the information we need by using the following:

1. In the setup function, we first start connecting the Arduino to the WiFi using an SSID which is the WiFi name and the password, and then make sure the board is connected before going to the next step.
2. Connect the board to the server by specifying the port and the host IP "in our case the server".
3. In the loop function we start sending the information by using Client.write().

- ESP32 Arduino Tutorial: Sending data with socket client - DFRobot

For the Server side, we used java.net and java.io and in the resources below we learned how to use the libraries by using the following:

1. Creating an object of the ServerSocket class for socket connection
2. set the port that we desire "We made sure the port is safe and not used by checking **IANA** database Service Name and Transport Protocol Port Number Registry"
3. Establishing a connection by making a socket and accepting messages
4. checking the message and changing the counter for the message

- Writing the Server Side of a Socket (The Java™ Tutorials > Custom Networking > All About Sockets)

- [Socket Programming in Java - GeeksforGeeks]
- [Java Socket Programming (Java Networking Tutorial) - javatpoint]

1- The server will create the socket using:

```
ServerSocket serverSocket = new ServerSocket(28002);
```

we make an object of class ServerSocket named “serversocket” that will bound the port 28002 and initialize it and then wait for the connection

```
Socket clientSocket = serverSocket.accept();
```

Now we establish the connection. If everything goes well, the server *accepts* the connection. Upon acceptance, the server gets a new socket, *clientSocket*, bound to the same local port, *28002*, and also has its remote endpoint set to the address and port of the client.

```
InputStreamReader ir =
new InputStreamReader(clientSocket.getInputStream());
BufferedReader br = new BufferedReader(ir);
int message;
```

Here we are just trying to get the message from the client by first using inputStreamReader that will get the input from the client Socket and then we use that in the bufferReader so we can say **br.read** when we want.

```
while(true){
        message = br.read();//READ FROM THE CLIENT
        if(message == 1)// 1 == PARKING TAKEN
          {
               numOfParking--;
               System.out.println("Available Parking:"
               + numOfParking);
         counter.setText(Integer.toString(numOfParking));

          }
          else if (message == 2)//2 PARKING NOT TAKEN
          {
             numOfParking++;
             System.out.println("Available Parking:"
             + numOfParking);
         counter.setText(Integer.toString(numOfParking));

          }
         if (!stage.isFullScreen()) {//to close the app if
                                     the window got minimized
                    clientSocket.close();
                    serverSocket.close();
```

```
                    Platform.exit();
                    System.exit(0);
                }}
```

And here is the final snippet of the Server-side with the conditionals and what to do with the messages from the client and at the end close the connection if needed.

2- The client will first establish a WiFi connection by the following code

```
WiFi.mode(WIFI_STA);
WiFi.begin(SSID,password);
```

the WiFi name and password are already defined at the top as follows with the port number and host IP address

```
const char* SSID ="iPhone"; //the HotSpot that is going to be
connected to
 const char* password="123456789"; //the password for it
 const uint16_t port = 28002; //the port specified by the server
 const char* host = "192.168.100.20"; //server IP address
```

and then will call a method that will try to connect to the WiFi and connect to the server and at the same time will blink the light **Blue** until the connection is made then will change to static **green** for 5 seconds and then will initiate the server connecting sequence that will blink **purple** until the connection is made.

```
void connect_toWiFi(){
Serial.print("WiFi connecting to  ");
Serial.print(SSID);
Serial.println("Connecting");
while(WiFi.status() != WL_CONNECTED){//WiFi not connected loop
      rgb(0,0,255);//Blue
      delay(500);
      rgb(0,0,0);//OFF
      delay(200);
 }

rgb(255,170,0);//green for 5 seconds
delay(5000);
rgb(0,0,0);


Serial.print("\nWiFi connected Success!");
Serial.println("NodeMCI IP Address");
Serial.println(WiFi.localIP());
```

```
while (!client.connect(host, port)) {//try to connect to
                                     server
 Serial.println("Connection to Server failed");
 }
}
```

3- Now we go into the main functionality of the Arduino, and that is the sensor. the board will check for WiFi connection every single loop and whenever we lose connection will go to connect_toWiFi function and re-establish it again. and if everything goes well we will take the readings from the sensor and go through two if statements, the first if the sensor does sense anything lower than 60cm then turn the LED **RED** and send the information to the server to update the counter. The second is if the sensor does not sense anything lower than 61cm then turn the LED **Green**. both will have a flag called isParked so that we don't send the same reading more than once.

```
bool isParked = true;
void loop() {
          //checking if connected to wifi if not go to
     connect_toWiFi function
    if(WiFi.status() != WL_CONNECTED ||!client.connected())
    connect_toWiFi();

  //the sensor conditionals
  //1st -> if there is something under 60cm light LED RED
  //2nd -> if there is nothing under 60cm light LED GREEN
  if(sensor.ping_cm()<60 && isParked == true){
  rgb(255,0,0);//RED
  isParked = false;//flag if the first time
  client.write(1);//send to the Server
  delay(100);
  delay(100);
  } else if(sensor.ping_cm() >61 && isParked == false){
  rgb(0,255,0);//GREEN
  isParked = true;//flag if the first time
  client.write(2);//send to the server
  delay(100);
  }delay(100);
```

4-Steps for setting up the network

We have the HotSpot working as the WiFi for the two and a pc/laptop to run the server in and WeMos Arduino.
First, we run the code for the server and make sure we put the right SSID and Password in the server code, and at the same time connect the laptop to the same WIFi.
then we turn the Arduino on and initiate the WiFi/server connection sequence,
which is the following
1-light blinking blue -> looking for the HotSpot
2-static green for 5 seconds -> WiFI connection established
3-no light -> trying to connect to the server
4-green light ->done with WiFI/Server connection sensor is working and active
*whenever the connection is lost the sequence will repeat

5- Codes and comments:

Code of server side:

```
package server;

//javafx libraries
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;

import java.io.*;
import java.net.*;
```

```
public class Server extends Application {
    // now we only have one sensor so only one parking spot
    static int numOfParking = 1;

    @Override
    public void start(Stage stage) throws IOException {
        Group root = new Group();
        Scene scene = new Scene(root, Color.BLACK);
        stage.getIcons().add(new Image("file:src/app_icon.png"));
        stage.setTitle("Server Parking Counter");
        Text text = new Text();
        text.setText("Available Parking");
        text.setX(690);
        text.setY(400);
        text.setFill(Color.rgb(51, 255, 0));
        Font font = Font.loadFont("file:src/digital-7 (mono).ttf", 70);
        text.setFont(font);
        root.getChildren().add(text);
        Text counter = new Text();
        counter.setText(Integer.toString(numOfParking));
        counter.setX(950);
        counter.setY(500);
        counter.setFill(Color.rgb(51, 255, 0));
        counter.setFont(font);
        root.getChildren().add(counter);
        stage.setFullScreen(true);
        stage.setScene(scene);
        stage.show();

        ServerSocket serverSocket = new ServerSocket(28002);
        System.out.println("The Server is waiting for a client on port
        28002");//only for the terminal
        Socket clientSocket = serverSocket.accept();// Accepts the
   connection for the client socket
        System.out.println("Connected");//only for the terminal


        new Thread(() -> {//made another thread so the main one won't clog
                          up and freeze the app
            try (InputStreamReader ir = new
InputStreamReader(clientSocket.getInputStream())) {
```

```
                BufferedReader br = new BufferedReader(ir);
                int message;
                while (true) {
                    message = br.read();
                    if (message == 1) {
                        numOfParking--;
                        System.out.println("Available Parking: " +
                    numOfParking);//only for the terminal
                        counter.setText(Integer.toString(numOfParking));

                    } else if (message == 2) {
                        numOfParking++;
                        System.out.println("Available Parking: " +
                    numOfParking);//only for the terminal
                        counter.setText(Integer.toString(numOfParking));

                    }
                    if (!stage.isFullScreen()) {//to close the app if the
                                                window got minimized
                        clientSocket.close();
                        serverSocket.close();
                        Platform.exit();
                        System.exit(0);
                    }
                }
            } catch (IOException e)    {
                e.printStackTrace();
            }
        }).start();


    }

    public static void main(String[] args) throws IOException {
        launch();
    }
}
```

Code for client side:

```
#include <ESP8266WiFi.h>
#include <NewPing.h>

 const char* SSID ="iPhone"; //the HotSpot that is going to be connected to
 const char* password="123456789"; //the password for it
 const uint16_t port = 28002; //the port specified by the server
 const char* host = "192.168.100.20"; //server IP address

WiFiClient client;
#define red D0
#define green D1
#define blue D2
#define trig D6
#define echo D5
NewPing sensor(trig,echo, 400);//trigger then echo pin then the maximum distance


     void setup() {
    Serial.begin(9600);

    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID,password);

    connect_toWiFi();

    //making a new client and connecting to the host and the designated port
    WiFiClient client;
      while (!client.connect(host, port)) {
      Serial.println("Connection to Server failed");
      }
      rgb(0,255,0);

}
bool isParked = true;
void loop() {
    //checking if connected to wifi if not go to connect_toWiFi function
    if(WiFi.status() != WL_CONNECTED || !client.connected())
    connect_toWiFi();
```

```
  //the sensor conditionals
  //1st -> if there is something under 60cm light LED RED
  //2nd -> if there is nothing under 60cm light LED GREEN
  if(sensor.ping_cm()<60 && isParked == true){
  rgb(255,0,0);
  isParked = false;
  client.write(1);
  delay(100);
  delay(100);
  } else if(sensor.ping_cm() >61 && isParked == false){
  rgb(0,255,0);
  isParked = true;
  client.write(2);
  delay(100);
  }delay(100);

}

void rgb(int i, int j, int k){
  analogWrite(red, i);
  analogWrite(green,j);
  analogWrite(blue,k);

  }

   void connect_toWiFi(){
    Serial.print("WiFi connecting to  ");
    Serial.print(SSID);
    Serial.println("Connecting");
    while(WiFi.status() != WL_CONNECTED){
      rgb(0,0,255);
      delay(500);
      rgb(0,0,0);
      delay(200);
    }

    rgb(0,255,0);
    delay(5000);
    rgb(0,0,0);

    Serial.print("\nWiFi connected Success!");
    Serial.println("Wemos D1 IP Address");
    Serial.println(WiFi.localIP());
```

```
while (!client.connect(host, port)) {
  rgb(128,0,128);
  delay(1000);
  rgb(0,0,0);
  }

}
```

6- Snapshots of the application outputs.

*no car detected

*car detected

```
The Server is waiting for a client on port 28002
Connected
Available Parking: 0
Available Parking: 1
Available Parking: 0
Available Parking: 1
Available Parking: 0
Available Parking: 1
Available Parking: 0
```

*Terminal output

7- Costs:

The cost of one WeMos R1D1 board is 16.9 riyals, a pack of 40 jumper wires goes for 3.4 riyals, a pack of 300 220Ω resisters goes for 14.4 riyals, one HC-SR04 ultrasonic sensor is 1.43 riyals, a pack of 100 RGB lights is 16.87 riyals. We used a breadboard for the prototype, but in the real world the jumper wires are soldered so a breadboard will not be necessary.

If we want to scale up and connect multiple lights and ultrasonic sensors to a single board. We will be able to connect 2 of each, for power we will use an external power supply. To connect the led light we need 3 digital pins one for each channel red, green, and blue, and 2 pins for the ultrasonic sensor for echo and target. We have 13 digital pins and we need 5 to connect a light and an ultrasonic sensor, so that is why we are able to cover 2 parking spots with a single board.

If we want to build a parking system that has 50 cars, we will need 25 boards, 50 sensors, 50 lights, 150 resistors, and 400 jumper wires. the cast are going to be:

	Quantity	Cost	Total cost
WeMos R1 D1	25	16.9	422.5
HC-SR04	50	1.43	71.5
4 pin rgb light	50	16.87 (comes in a pack of 100)	16.87
Jumper wires	400	3.4 (comes in a pack of 40)	34
Resisters	150	14.4 (comes in a pack of 300)	14.4

So the total cost for all of these components comes to 559.27 Riyals. The system is feasible and easy to adopt because of the low price and the simplicity of the system. everything can be set up and up and running in less than a week if we are talking about a single parking field that is a single story tall, and upgradability is not going to be an issue. Because we just have to connect the new sensors to the local network and edit the java program.

In the beginning we are going to work with small businesses to try the system in action and get their feedback on what to improve and add. After the experiment period, we are going to trage big businesses and shopping centers to adopt the sensor to make it easier for visitors and to find parking spots, and for businesses to follow their parking situation and improve it if needed.

8- Problems and solutions:

Problem #: libraries not working well or being recognized
Solution: because they were missing or they had to be installed manually

Problem #: Arduino board not being recognized
Solution: the fix was installing drives for it either from the IDE or from the board website
CH340 Drivers for Windows, Mac and Linux

Problem #: board recognized but won't upload
Solution: changed the COM port that is connected to the PC/laptop

Problem #: code is not being uploaded to the board
Solution: the fix was changing the upload speed

Problem #: ultrasonic sensor reading inaccuracy
Solution: the fix was slowing down the reading speed

Problem #: RGB LED blinking on its own
Solution: adding a delay so any noise will be somewhat discarded

Problem #: ultrasonic sensor enters conditional even if nothing changes and sends information not needed to the server
Solution: isParked flag so the code never enters the same conditional unless something changed so the server will only get a message if someone parked or not and not the real-time readings that keep reading thousands of times a second

Problem #: the server incrementing/decrementing to infinity
Solution: we were taking the message once and it gets repeatedly checked in the loop so we put it inside the loop and every loop gets a new message
Problem #: LED and sensor freezing after WiFi/server connection is done
Solution: the flag was defined inside the loop so it will be always true and will never change so we put it as a global variable

Problem #: the server is not getting any messages
Solution: we were using the client.print() instead of client.Write()

Problem #: the server is crashing when launched
Solution: so the problem was that the main thread for the javafx can't work with while loops that are infinite "in our case the receiving of the message need to be in one" so we needed to make another thread and make the receiving of the messages in the background with that thread and the main gui in the main one
Problem #: the counter text in the GUI is not updating
Solution: needed to set the text and parse it into in integer every time the message is being read and checked

References:

[1] Arduino
[2] ESP32 Arduino Tutorial: Sending data with socket client - DFRobot
[3] Service Name and Transport Protocol Port Number Registry
[4] Writing the Server Side of a Socket (The Java™ Tutorials > Custom Networking > All About Sockets)
[5] Java Socket Programming (Java Networking Tutorial) - javatpoint
[6] https://openjfx.io
[7] Overview (JavaFX 8)
[8] JavaFX Tutorial - javatpoint
[9] Java Threads
[10] JavaFX Concurrency