Team Members:

R.Deepika Sri-921021104006

T. Muthuvaisali-921021104030

L.Luckymalathi-921021104023

M.Sowmiya-921021104047

**Smart Parking System Using IoT\_Phase4**

**Introduction**:

Smart parking systems are becoming increasingly popular as a way to address the challenges of urban congestion and parking scarcity. By using sensors and IoT connectivity, smart parking systems can provide real-time information on parking availability, guide drivers to open spots, and even automate the payment process.

This will build a simple smart parking system using an Arduino or ESP8266 microcontroller, ultrasonic sensors, and a cloud platform like Firebase. The system will detect the presence of vehicles in parking slots and send this information to the cloud in real time. Users can then access this information through a web or mobile app to see which parking slots are available and get directions to them.

**Components:**

* Arduino or ESP8266 microcontroller
* Ultrasonic sensors (one for each parking slot)
* LED indicator lights (one for each parking slot, optional)
* Breadboard and jumper wires
* Power supply
* Cloud platform

**Hardware Setup:**

* Connect the ultrasonic sensors to the microcontroller according to the sensor’s datasheet.
* Connect the LED indicator lights to the microcontroller (optional).
* Connect the microcontroller to the power supply.

**Software Setup:**

* Install the necessary libraries for the microcontroller and cloud platform.
* Write a program to read the ultrasonic sensors and send the data to the cloud.
* Create a web or mobile app to display the parking availability information from the cloud.

**Coding:**

**This is the code we given in to the arduino UNO:**

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

LiquidCrystal\_I2C lcd(0x27, 16, 2);  // Change the HEX address

#include <Servo.h>

Servo myservo1;

int IR1 = 2;

int IR2 = 4;

int SmokeDetectorPin = 6;  // Digital pin for the smoke detector

int BuzzerPin = 7;         // Digital pin for the buzzer

int Slot = 4;  // Enter Total number of parking Slots

bool flag1 = false;

bool flag2 = false;

unsigned long lastLcdUpdate = 0;  // Variable to track the time of the last LCD update

unsigned long lcdUpdateInterval = 1000;  // Update the LCD every 1000 milliseconds (1 second)

void setup() {

  lcd.begin(16, 2);  // Initialize LCD with 16 columns and 2 rows

  lcd.backlight();

  pinMode(IR1, INPUT);

  pinMode(IR2, INPUT);

  pinMode(SmokeDetectorPin, INPUT);

  pinMode(BuzzerPin, OUTPUT);

  myservo1.attach(3);

  myservo1.write(100);

  lcd.setCursor(0, 0);

  lcd.print("     ARDUINO    ");

  lcd.setCursor(0, 1);

  lcd.print(" PARKING SYSTEM ");

  delay(2000);

  lcd.clear();

  Serial.begin(9600);  // Start serial communication for debugging

}

void loop() {

  if (digitalRead(IR1) == LOW && !flag1) {

    if (Slot > 0) {

      flag1 = true;

      if (!flag2) {

        myservo1.write(0);

        Slot--;

      }

    } else {

      displayMessage("    SORRY :(    ", "  Parking Full  ");

    }

  }

  if (digitalRead(IR2) == LOW && !flag2) {

    flag2 = true;

    if (!flag1) {

      myservo1.write(0);

      Slot++;

    }

  }

  if (flag1 && flag2) {

    delay(1000);

    myservo1.write(100);

    Serial.println("Servo returned to initial position.");

    flag1 = false;

    flag2 = false;

  }

  // Update the LCD display with a delay

  if (millis() - lastLcdUpdate >= lcdUpdateInterval) {

    updateLcdDisplay();

    lastLcdUpdate = millis();

  }

  // ... (Rest of your code)

}

void updateLcdDisplay() {

  if (digitalRead(SmokeDetectorPin) == HIGH) {

    displayMessage("   WARNING!   ", " Smoke Detected ");

    digitalWrite(BuzzerPin, HIGH);  // Turn on the buzzer

  } else {

    displayMessage("    WELCOME!    ", "Slot Left: " + String(Slot));

    digitalWrite(BuzzerPin, LOW);   // Turn off the buzzer

  }

}

void displayMessage(const char \*line1, const String &line2) {

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print(line1);

  lcd.setCursor(0, 1);

  lcd.print(line2);

}

Here the json file for other altering things we need to include:

{

  "version": 1,

  "author": "Shraddha Trivedi",

  "editor": "wokwi",

  "parts": [

    { "type": "wokwi-arduino-uno", "id": "uno", "top": 10, "left": -112, "attrs": {} },

    { "type": "wokwi-servo", "id": "servo1", "top": -154.2, "left": -95.59, "attrs": {} },

    { "type": "wokwi-ir-receiver", "id": "ir1", "top": -99.35, "left": 279.95, "attrs": {} },

    { "type": "wokwi-ir-receiver", "id": "ir2", "top": 129.39, "left": 354.11, "attrs": {} },

    {

      "type": "wokwi-lcd1602",

      "id": "lcd1",

      "top": 264.82,

      "left": 323.78,

      "attrs": { "pins": "i2c" }

    },

    {

      "type": "wokwi-buzzer",

      "id": "bz1",

      "top": -215.97,

      "left": 154.63,

      "attrs": { "volume": "0.1" }

    },

    {

      "type": "wokwi-pir-motion-sensor",

      "id": "pir1",

      "top": 295.88,

      "left": -227.65,

      "attrs": {}

    }

  ],

  "connections": [

    [ "servo1:PWM", "uno:3", "green", [ "h211.26", "v36.1" ] ],

    [ "ir1:DAT", "uno:2", "green", [ "v19.37", "h-211.08" ] ],

    [ "ir2:DAT", "uno:4", "green", [ "v-195.46", "h-239.7" ] ],

    [ "ir1:GND", "uno:GND.2", "black", [ "v0" ] ],

    [ "ir2:GND", "uno:GND.3", "black", [ "v-116.8", "h-11.16", "v2", "h-0.67", "v-3.33" ] ],

    [ "ir1:VCC", "uno:5V", "red", [ "v207.21", "h-278.74" ] ],

    [ "ir2:VCC", "uno:5V", "red", [ "v-128.13", "h-13.43", "v124", "h-276" ] ],

    [ "lcd1:VCC", "uno:5V", "red", [ "h-4.56", "v-99.59", "h-275.86" ] ],

    [ "lcd1:SDA", "uno:A4", "blue", [ "h0" ] ],

    [ "lcd1:SCL", "uno:A5", "green", [ "h0" ] ],

    [ "servo1:V+", "uno:5V", "red", [ "h420.11", "v244.21", "h-274.13" ] ],

    [ "servo1:GND", "uno:GND.3", "black", [ "h410.97", "v81.11" ] ],

    [ "lcd1:GND", "uno:GND.2", "black", [ "h-269.94", "v-95.52" ] ],

    [ "bz1:1", "uno:GND.3", "cyan", [ "v337.36", "h-114.55" ] ],

    [ "pir1:VCC", "uno:5V", "orange", [ "v6.98", "h240.35", "v-193.34" ] ],

    [ "pir1:GND", "uno:GND.3", "#8f4814", [ "v-4.85", "h210.37" ] ],

    [ "pir1:OUT", "uno:6", "green", [ "v-411.27", "h249.03" ] ],

    [ "bz1:2", "uno:7", "violet", [ "v123.05", "h-119.07" ] ]

  ],

  "dependencies": {}

}

**About libraries.txt**:

The following libraries are included in the project:

LiquidCrystal: This library is used to control a 16x2 character LCD display.

Servo: This library is a great one to control the servo motors.

**Mobile App:**

Mobile app development framework like Flutter to create an app that displays realtime parking availability.

The Steps we followed to create the flutter application:

1. Set up the Flutter development environment.
2. Create a Firebase project and enable the Firestore database.
3. Write a backend service to retrieve the parking availability data from Firestore and send it to the Flutter app.
   * we use a programming language such as Python or Node.js to write the backend service.
   * The backend service should expose an API that the Flutter app can call to retrieve the parking availability data.
4. Develop the Flutter app.
   * The Flutter app should connect to the backend service and call the API to retrieve the parking availability data.
   * The Flutter app should display the parking availability data in a user-friendly way.

This is the Dart code for create Flutter Application:

import 'package:flutter/material.dart';

import 'package:http/http.dart' as http;

class ParkingAvailabilityApp extends StatefulWidget {

@override

\_ParkingAvailabilityAppState createState() => \_ParkingAvailabilityAppState();

}

class \_ParkingAvailabilityAppState extends State<ParkingAvailabilityApp> {

// The URL of the backend API

final String backendApiUrl = 'https://example.com/api/parking-availability';

// The parking availability data

List<ParkingSpace> \_parkingSpaces = [];

@override

void initState() {

super.initState();

// Fetch the parking availability data from the backend API

\_fetchParkingAvailabilityData();

}

Future<void> \_fetchParkingAvailabilityData() async {

// Make a GET request to the backend API

final response = await http.get(Uri.parse(backendApiUrl));

// Decode the JSON response

final parkingSpaces = jsonDecode(response.body) as List<dynamic>;

// Convert the JSON data to a list of ParkingSpace objects

setState(() {

\_parkingSpaces = parkingSpaces.map((parkingSpace) => ParkingSpace.fromJson(parkingSpace)).toList();

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Parking Availability'),

),

body: ListView.builder(

itemCount: \_parkingSpaces.length,

itemBuilder: (context, index) {

final parkingSpace = \_parkingSpaces[index];

return ListTile(

title: Text(parkingSpace.name),

subtitle: Text(parkingSpace.isOccupied ? 'Occupied' : 'Available'),

);

},

),

);

}

}

class ParkingSpace {

final String name;

final bool isOccupied;

ParkingSpace({required this.name, required this.isOccupied});

factory ParkingSpace.fromJson(Map<String, dynamic> json) {

return ParkingSpace(

name: json['name'],

isOccupied: json['isOccupied'],

);

}

}

Design app functions to receive and display parking availability data received from the Raspberry Pi.

We follow these steps to design app functions to receive and display parking availability data received from the Raspberry Pi.

1. Set up a communication channel between the Raspberry Pi and the mobile app. You can use a variety of communication channels, such as TCP/IP, UDP, or Bluetooth.
2. Develop a protocol for exchanging parking availability data between the Raspberry Pi and the mobile app. The protocol should define the format of the messages and the sequence of messages that will be exchanged.
3. Implement the communication protocol on the Raspberry Pi and the mobile app. This will involve writing code to send and receive messages, as well as to parse and interpret the messages.
4. Design the user interface of the mobile app to display the parking availability data. The user interface should be easy to use and understand, and it should provide the user with all of the necessary information about the parking availability.

This is implement the communication protocol on the Raspberry Pi and the mobile app:

**This is Python code for Raspberry Pi:**

import socket

# Create a TCP server socket

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Bind the server socket to the port 8080

server.bind(('localhost', 8080))

# Listen for incoming connections

server.listen(1)

# Accept an incoming connection

client, addr = server.accept()

# Receive a message from the client

message = client.recv(1024)

# Decode the JSON message

parking\_availability\_data = json.loads(message.decode())

# Close the client socket

client.close()

**This is Python code for Mobile App:**

import socket

import json

# Create a TCP client socket

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Connect to the server socket on the Raspberry Pi

client.connect(('localhost', 8080))

# Send a message to the server socket requesting the parking availability data

message = json.dumps({'request': 'parking-availability-data'})

client.sendall(message.encode())

# Receive a message from the server socket containing the parking availability data

response = client.recv(1024)

# Decode the JSON message

parking\_availability\_data = json.loads(response.decode())

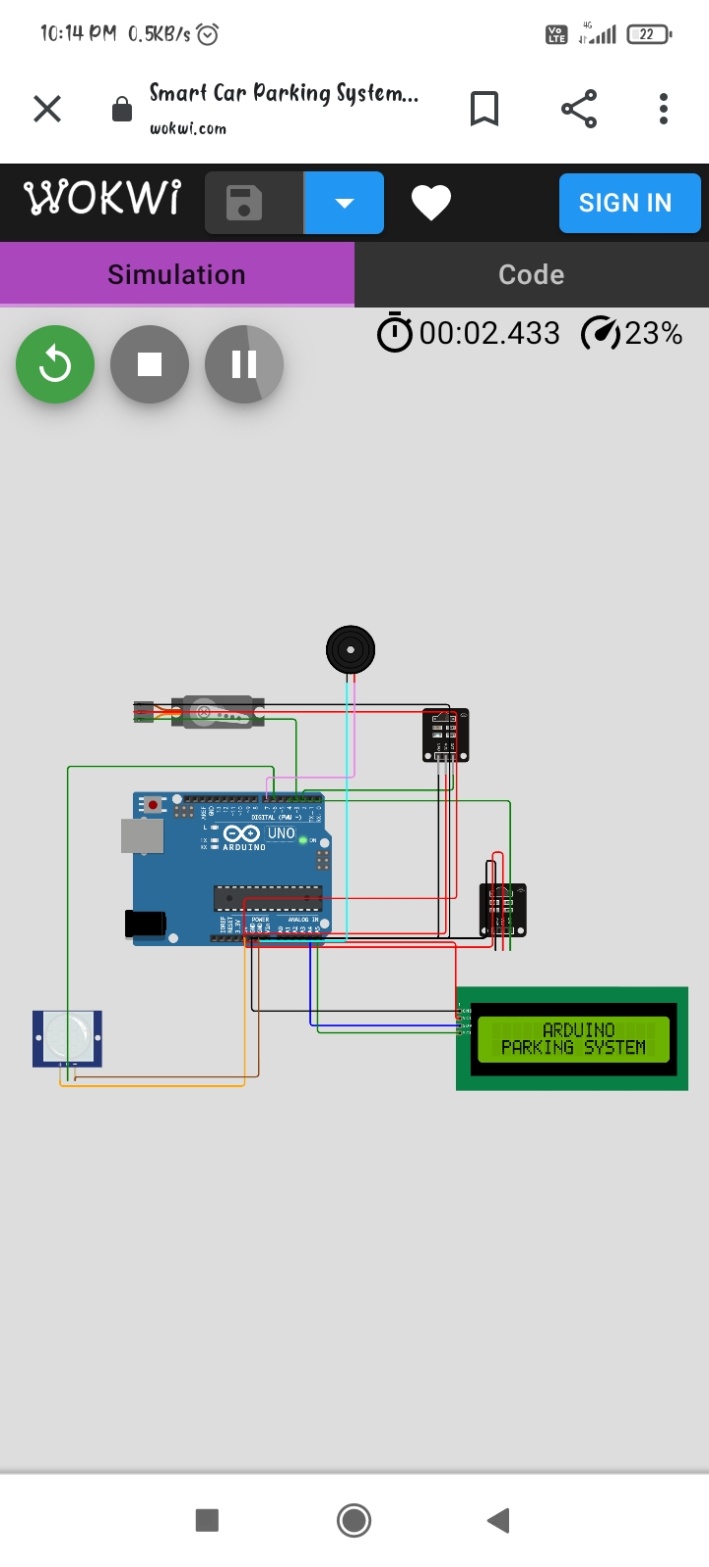
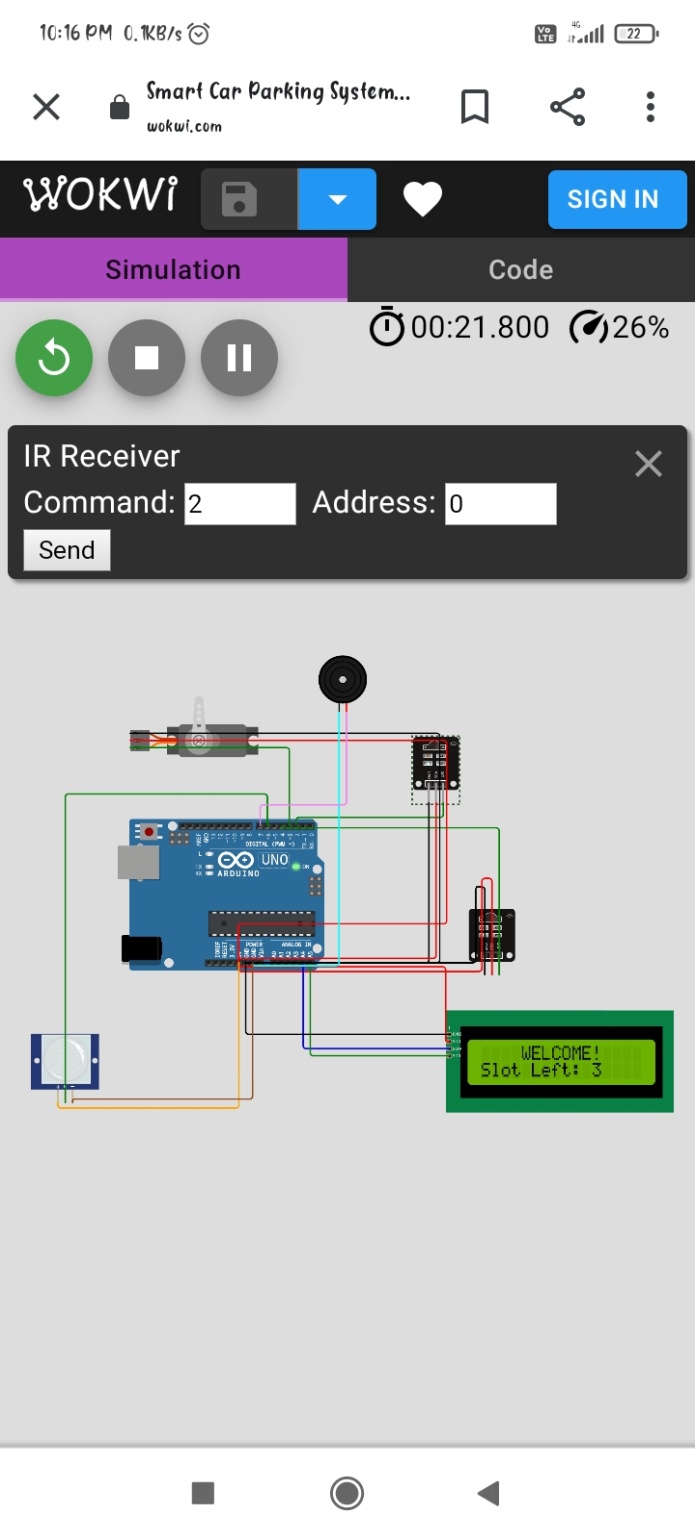
# Close the client socket

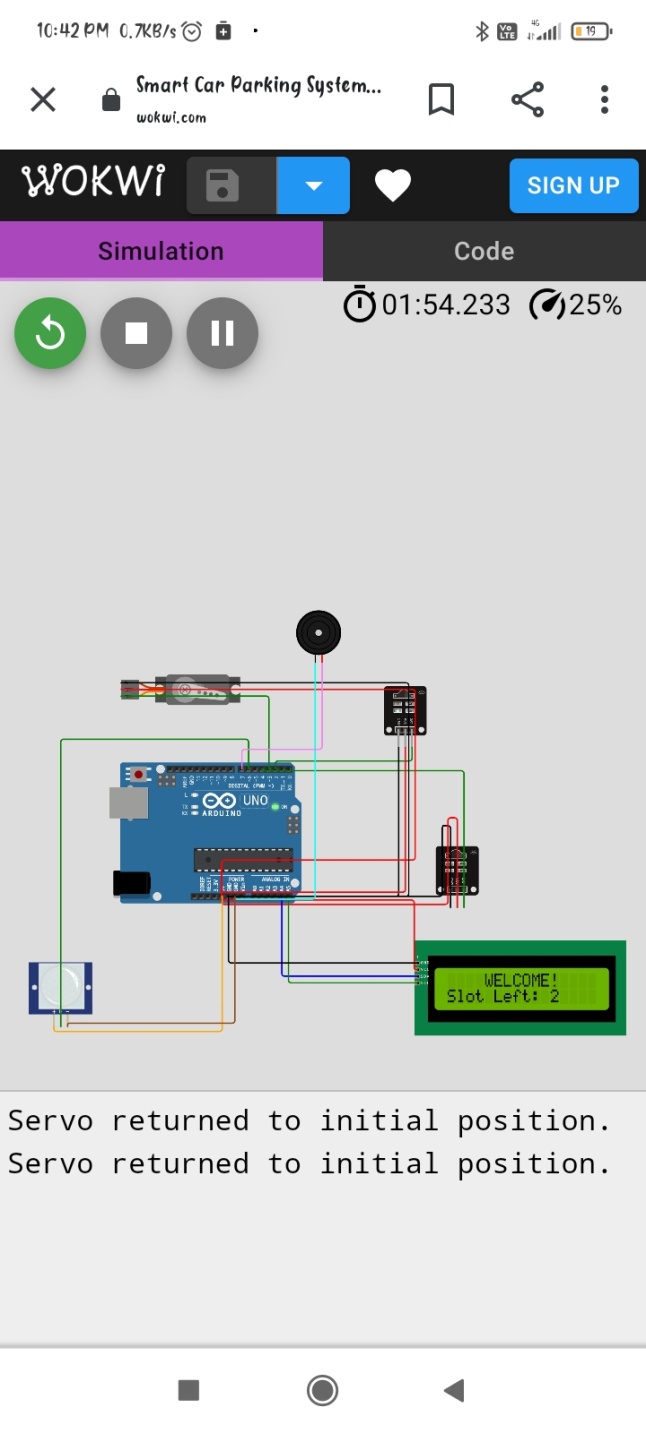
client.close()

# Display the parking availability data to the use

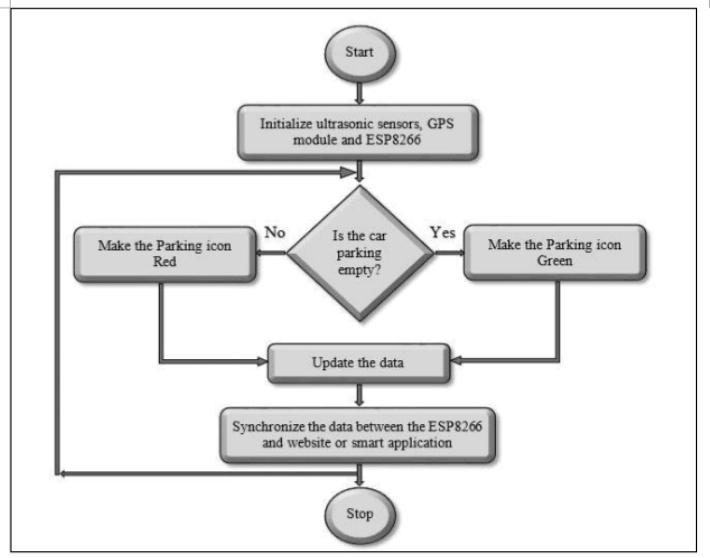
The link for the project: <https://wokwi.com/projects/378996446505722881>

**Simulation Results:**

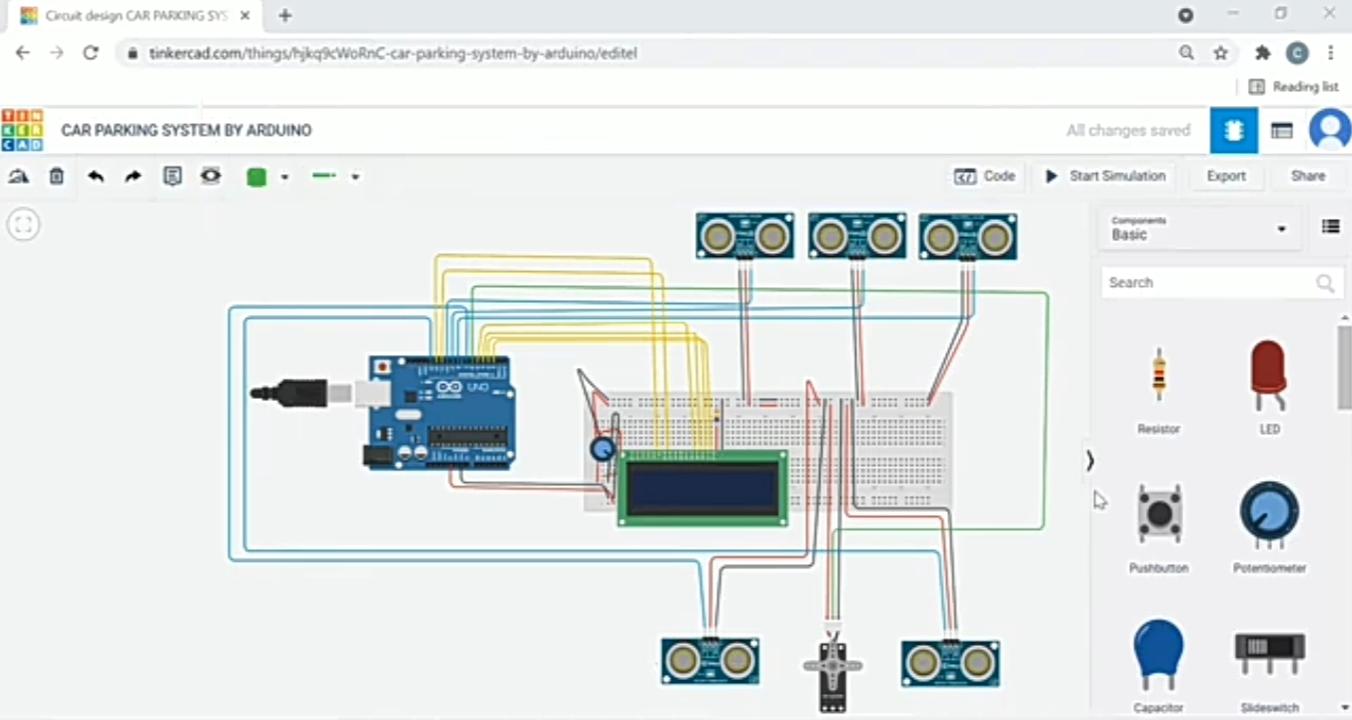
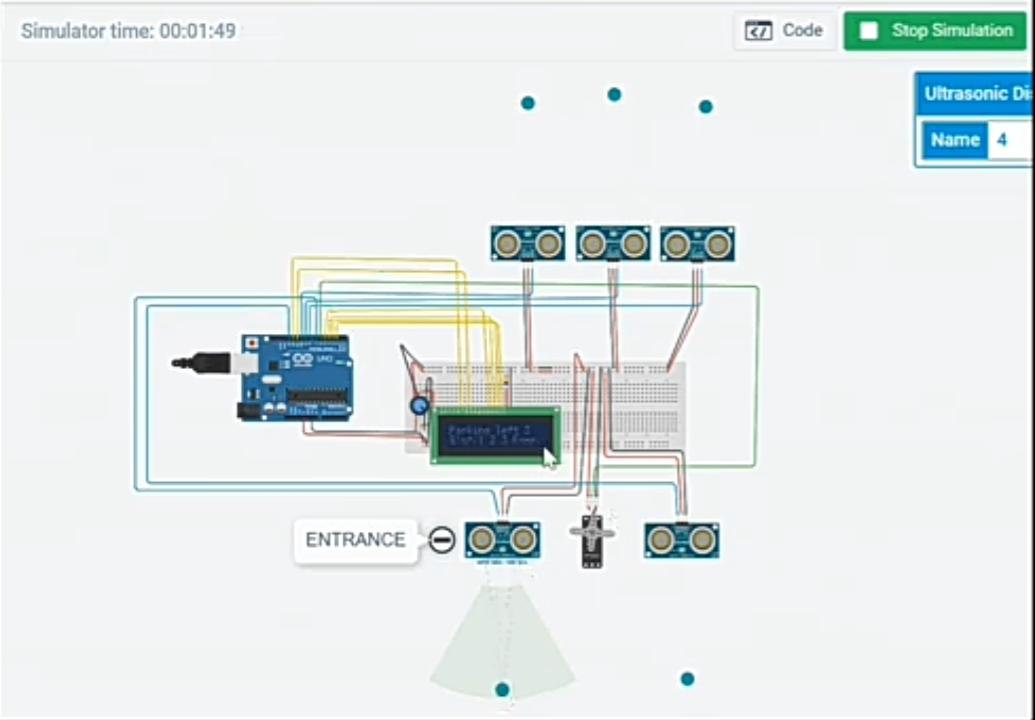
****

******

Since we had trouble using Wokwi Simulator in the PC web browser, we have implemented the project in the mobile web browser.

**Flowchart: **

**Tinkercad Results:**

****

**Conclusion:**

As a conclusion. This project will help in reducing the amount of time a driver has to spend around the parking just to find an available spot, reducing the amount of traffic around the parking and also reducing the bad parking around the parking space.