IoT smart car parking

Phase 5: Project Documentation & Submission

Project statement:

The project aims to develop a Smart Parking solution that leverages IoT (Internet of Things) technology to enhance parking management and user experience. This system will provide real-time information about parking space availability, optimize parking space allocation, and streamline the parking process for both parking lot operators and users.

Objectives:

To tackle these challenges, we aim to achieve the following objectives:

Real-time Space Monitoring:

The primary objective of the project is to implement IoT sensors to enable real-time monitoring of parking space occupancy. By achieving this, the system will provide up-to-the-minute information on available parking spaces.

Optimized Space Allocation:

The project seeks to develop sophisticated algorithms that will dynamically allocate parking spaces based on real-time demand. This optimization aims to eliminate overcrowding in certain areas while ensuring efficient utilization of parking space.

User-friendly Mobile App:

A user-centric mobile application will be created to empower users with the ability to check parking space availability, reserve spots, and make contactless payments. This app will provide a seamless and convenient experience for drivers.

DESIGN THINKING:

In this phase, we will divide into the project's problem statement and outline a comprehensive design thinking approach to address it effectively.

Understanding the Problem:

current state of parking management in urban areas faces several challenges:

Parking Space Availability:

Users often struggle to find available parking spaces, leading to congestion and frustration.

Inefficient Space Allocation:

Parking lots may have uneven utilization of space, with some areas overcrowded while others remain underutilized.

Payment and Entry:

Traditional parking systems involve manual payment processes and entry/exit barriers that can cause delays and inconvenience.

IoT Sensor Design:

Sensor Selection:

Choose appropriate IoT sensors such as ultrasonic sensors or infrared sensors to detect the presence of vehicles in parking spaces. Additionally, utilize cameras for visual confirmation.

Sensor Placement:

Determine the optimal placement of sensors within each parking space to ensure accurate detection. Install cameras at entry and exit points for vehicle recognition.

Data Collection and Transmission:

Design a data collection mechanism to ensure reliable data gathering from sensors and cameras. Establish a data transmission protocol to send this information to a central server.

Real-Time Parking Management Platform User Interface Design:

Create an intuitive and user-friendly mobile app interface that displays real-time parking space availability, allows users to reserve spots, and make payments.

Data Processing:

Develop a backend system to process data received from IoT sensors and cameras. Implement algorithms for space allocation and availability updates.

Integration with Mobile App:

Ensure seamless integration between the backend system and the mobile app. The app should receive real-time updates on parking availability using firebase.

IoT Sensor Installation:

Procure and install appropriate IoT sensors (e.g., ultrasonic or infrared sensors) in each parking space across the parking lot. Install cameras at entry and exit points for vehicle recognition.

Gathering Components:

- Infrared sensor
- ESP3266
- jumper wires.
- power source.
- I2C LCD display
- Servo motor

NODEMCU ESP3266:

NodeMCU is an open-source firmware and development kit that helps you to prototype IoT products. It is based on the ESP8266 Wi-Fi module, a low-cost Wi-Fi chip with full TCP/IP stack and microcontroller capabilities. The NodeMCU development board is specifically designed for the NodeMCU firmware, and it features an ESP8266 chip, USB-to-serial programming interface, and several GPIO pins for hardware interfacing.

NodeMCU allows users to program the ESP8266 module with the Lua scripting language or the Arduino IDE. This makes it easier for beginners to work on IoT projects without having to learn the complexities of the underlying hardware. NodeMCU has gained popularity due to its ease of use, low cost, and robust capabilities, making it a popular choice for IoT prototyping and development.

Infrared sensor:

An IR (infrared) sensor is a device that detects and measures infrared radiation in its surrounding environment. Infrared radiation is a type of electromagnetic radiation with wavelengths longer than those of visible light, but shorter than those of radio waves. IR sensors are commonly used in various applications for detecting and measuring heat. They can detect the heat emitted by an object and convert it into an electrical signal, which can then be interpreted by a microcontroller or other electronic devices.

Script:

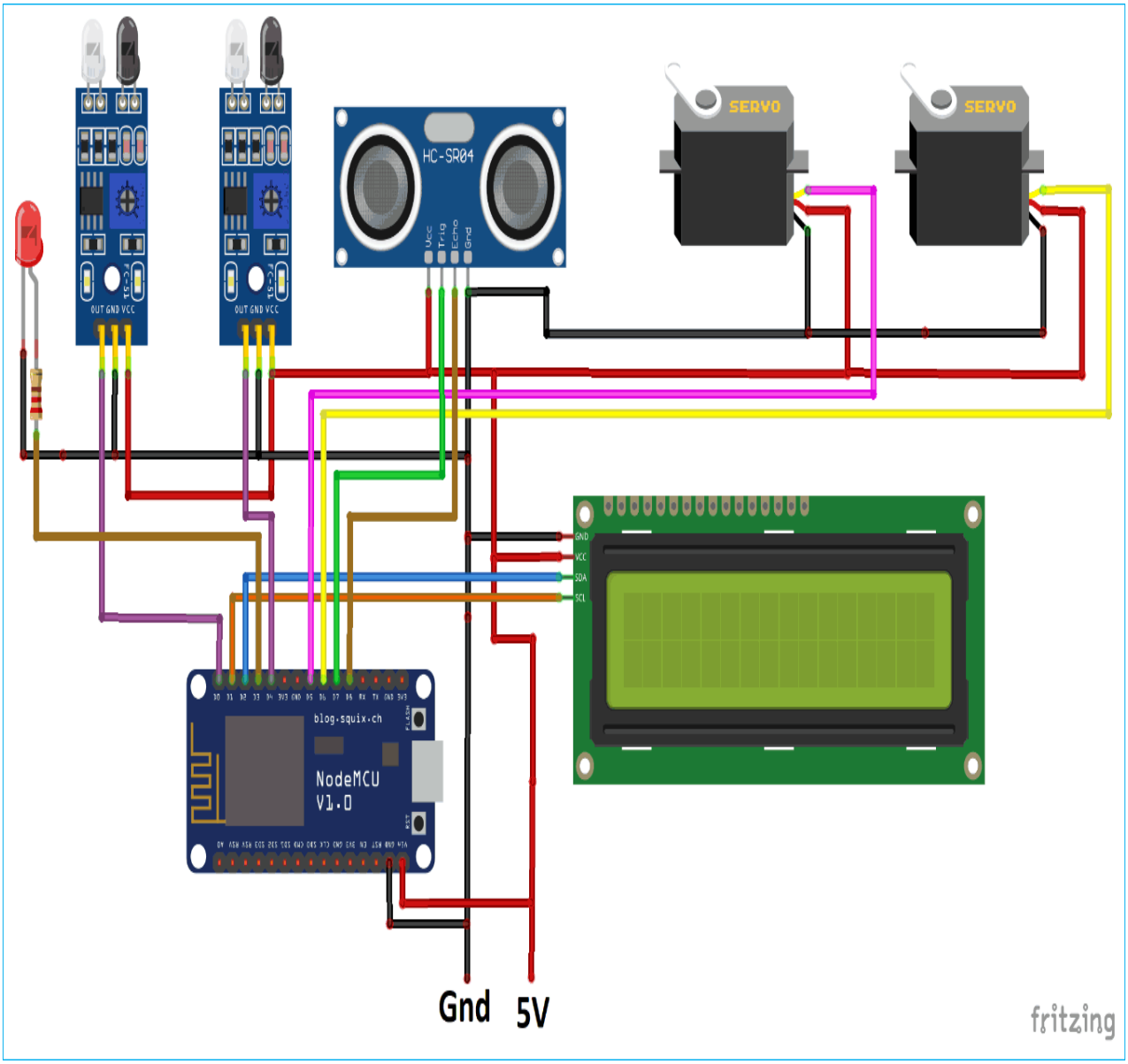
```
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

Servo gateServo1;
Servo gateServo2;
LiquidCrystal_I2C lcd(0x27, 16, 2); // Change the address if your
LCD is different

const int irSensor1 = D2; // IR sensor pins
const int irSensor2 = D3;
```

```
const int irSensor3 = D4;
const int irSensor4 = D5;
const int gatePin1 = D6;   // Servo motor control pins
const int gatePin2 = D7;

bool isOccupied1 = false;
bool isOccupied2 = false;

const char* ssid = "YourSSID";
const char* password = "YourPassword";
const char* serverURL = "http://yourserver.com"; // Change to your
server

void setup() {
  pinMode(irSensor1, INPUT);
  pinMode(irSensor2, INPUT);
  pinMode(irSensor3, INPUT);
  pinMode(irSensor4, INPUT);
  gateServo1.attach(gatePin1);
  gateServo2.attach(gatePin2);
  lcd.init();
  lcd.backlight();
  Serial.begin(115200);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
}

void loop() {
  int sensor1Value = digitalRead(irSensor1);
  int sensor2Value = digitalRead(irSensor2);
  int sensor3Value = digitalRead(irSensor3);
  int sensor4Value = digitalRead(irSensor4);
```

```
if (sensor1Value == LOW || sensor2Value == LOW) {
    openGate(1);
    isOccupied1 = true;
  } else {
    closeGate(1);
    isOccupied1 = false;
  }

  if (sensor3Value == LOW || sensor4Value == LOW) {
    openGate(2);
    isOccupied2 = true;
  } else {
    closeGate(2);
    isOccupied2 = false;
  }

  updateLCD(isOccupied1, isOccupied2);
  sendOccupancyStatus(isOccupied1, isOccupied2);
}

void openGate(int gate) {
  if (gate == 1) {
    gateServo1.write(90); // Open the first gate (adjust as needed)
  } else if (gate == 2) {
    gateServo2.write(90); // Open the second gate (adjust as needed)
  }
  delay(2000); // Open for 2 seconds (adjust as needed)
  closeGate(gate);
}

void closeGate(int gate) {
  if (gate == 1) {
    gateServo1.write(0); // Close the first gate
  } else if (gate == 2) {
    gateServo2.write(0); // Close the second gate
  }
}
```

```
void updateLCD(bool occupied1, bool occupied2) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Spot 1: ");
  lcd.print(occupied1 ? "Occupied" : "Vacant");
  lcd.setCursor(0, 1);
  lcd.print("Spot 2: ");
  lcd.print(occupied2 ? "Occupied" : "Vacant");
}

void sendOccupancyStatus(bool occupied1, bool occupied2) {
  HTTPClient http;

  if (WiFi.status() == WL_CONNECTED) {
    String url = serverURL + "/update?spot1=" + (occupied1 ?
"occupied" : "vacant") + "&spot2=" + (occupied2 ? "occupied" :
"vacant");

    http.begin(url);
    int httpCode = http.GET();

    if (httpCode == 200) {
      Serial.println("Status updated successfully");
    } else {
      Serial.println("Failed to update status");
    }

    http.end();
  }
```

Circuit:


HC-SR04
Vcc
Trig
Echo
Gnd
SERVO
SERVO
OUT GND VCC
OUT GND VCC
GND
VCC
SDA
SCL
blog.squix.ch
NodeMCU
V1.0
Gnd
5V
fritzing

App development:

Blynk is a popular platform that enables developers to build mobile applications for controlling and monitoring hardware projects. It provides a simple way to create interfaces for controlling and visualizing IoT (Internet of Things) devices. Blynk is particularly useful for beginners or those without extensive coding experience, as it simplifies the process of creating apps that interact with hardware.

To develop an app using Blynk, you typically follow these steps:

Set up your hardware: Choose the hardware you want to work with (such as an Arduino, Raspberry Pi, or ESP8266), and connect it to the internet.

Install the Blynk app: Download the Blynk app from the Google Play Store or the Apple App Store, depending on your device.

Create a Blynk account: Sign up for a Blynk account and log in to the Blynk app.

Create a new Blynk project: In the Blynk app, create a new project and select the hardware you're using.

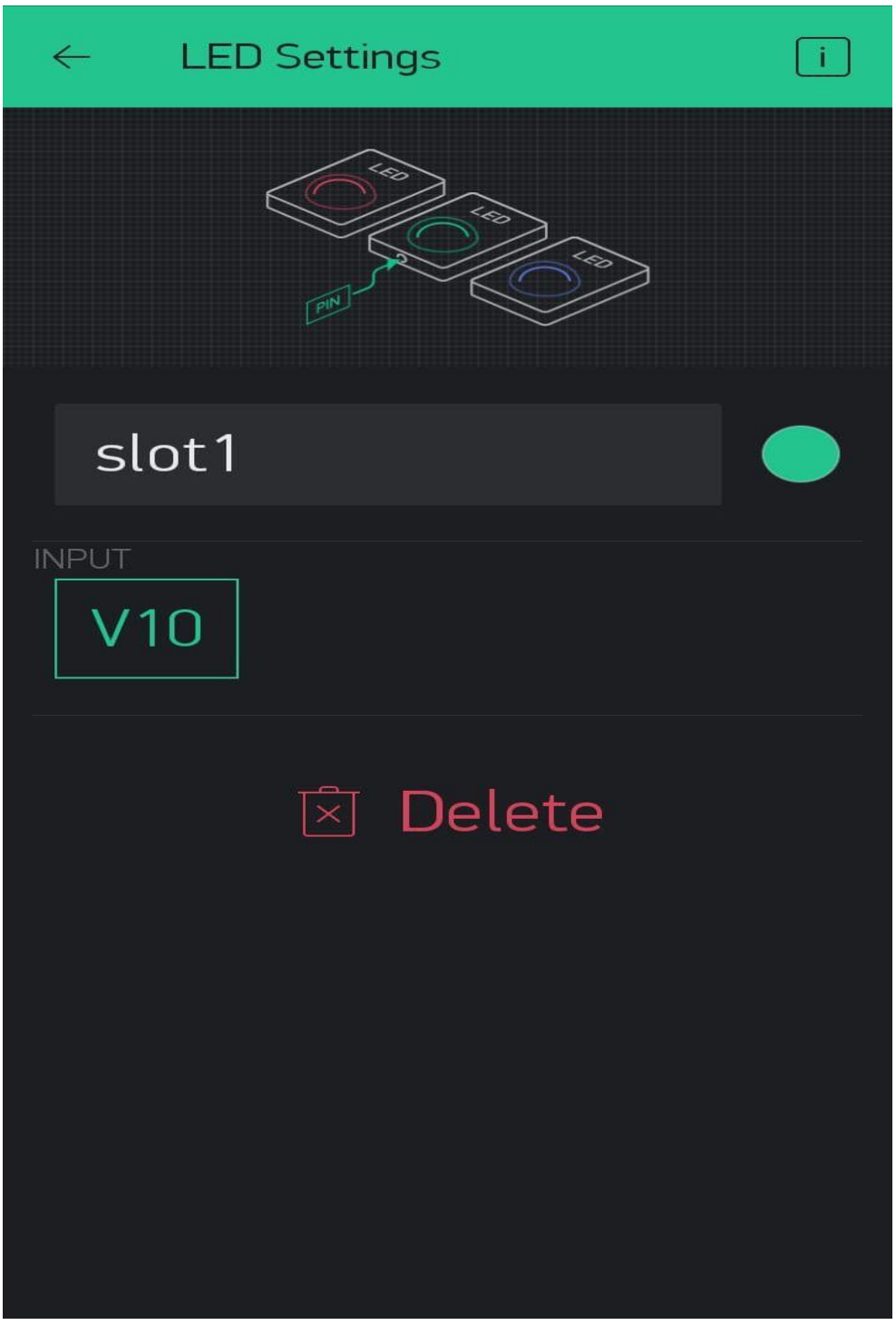
Configure the interface: Design the interface of your app by adding buttons, sliders, graphs, and other widgets to control and visualize your hardware.

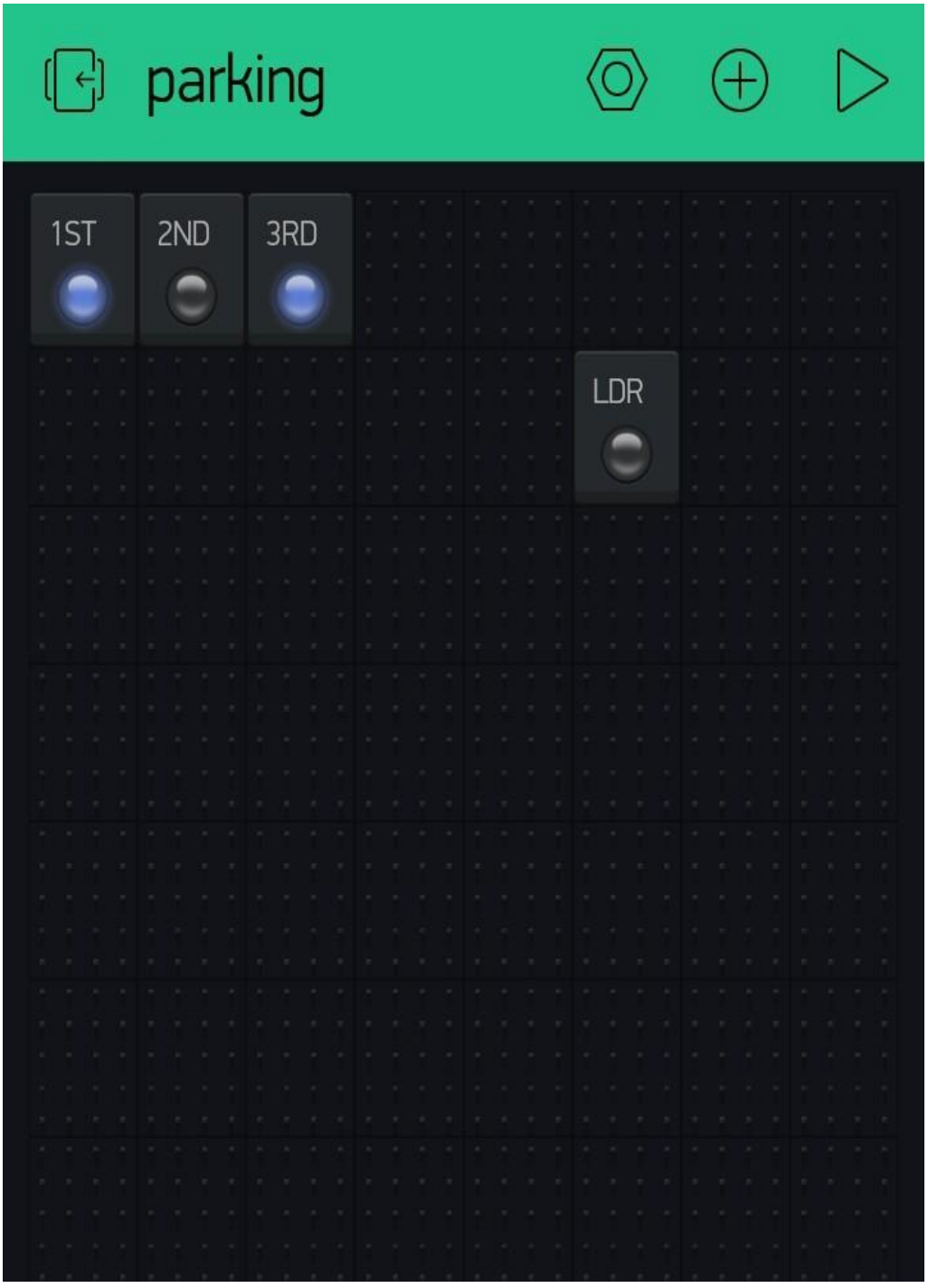
Obtain an authentication token: You will need an authentication token to establish a connection between your hardware and the Blynk app. The token is used to authenticate the hardware to the Blynk server.

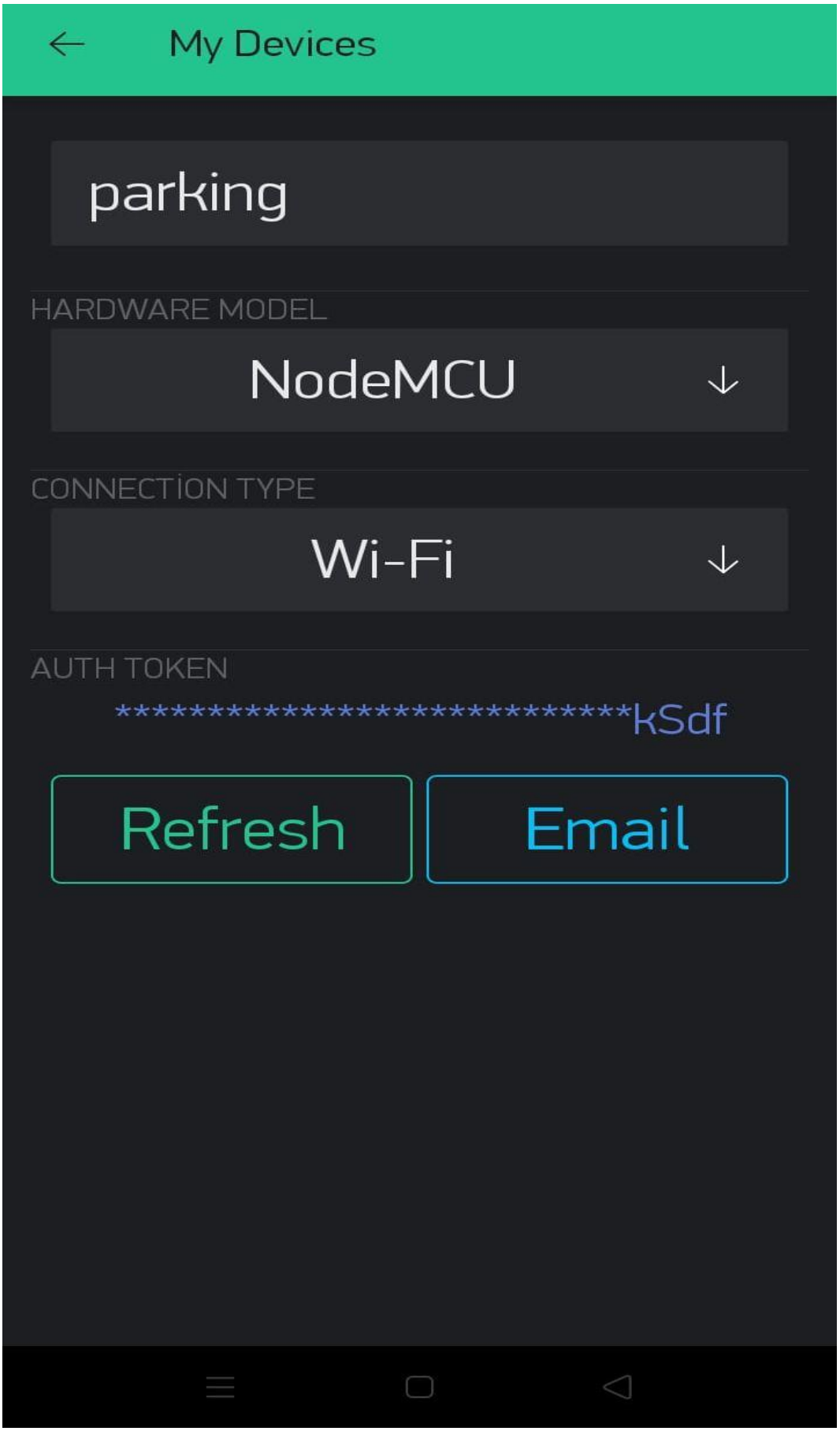
Code your hardware: Write the code for your hardware, incorporating the Blynk library and the authentication token. This code will enable your hardware to communicate with the Blynk app.

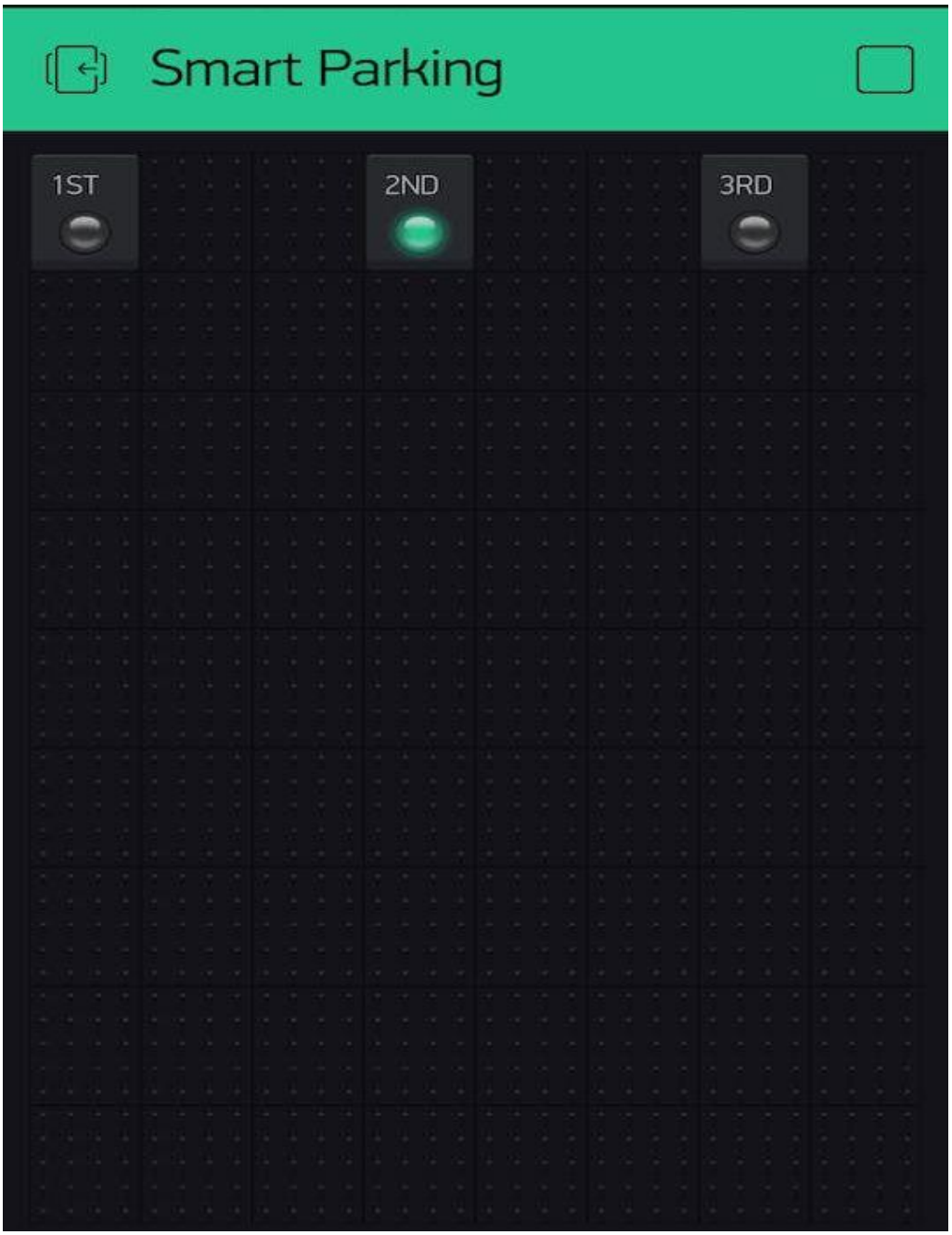
Test your project: Upload the code to your hardware, and test the connection between the hardware and the Blynk app.

Screenshot of the mobile application:




parking
1ST
2ND
3RD
LDR


My Devices
parking
HARDWARE MODEL
NodeMCU
CONNECTİON TYPE
Wi-Fi
AUTH TOKEN
**********kSdf
Refresh
Email


Smart Parking
1ST
2ND
3RD


CarParking
Not_ Available
Available
Available

COLLEGE CODE: 4212

PHASE 5

COLLEGE CODE: 4212

PHASE 5