



SMART PARKING SYSTEM USING IOT

PHASE IV

TEAM MEMBERS:

SIVAPRAKASH A (810021106079)

TONY CHACKO THOMAS (810021106089)

SINDHUJA U (810021106076)

SHRIVARSHA P (810021106075)

SRIHARI VV (810021106303)

VEERANANDHA KUMAR (810021106310)

SMART PARKING MANAGEMENT USING IOT

INTRODUCTION:

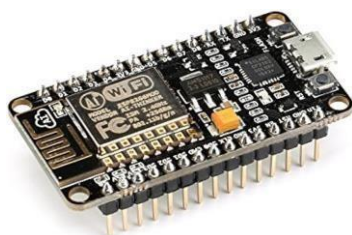
In today's rapidly evolving urban landscape, efficient parking management is becoming increasingly critical to address traffic congestion and enhance the overall quality of urban life. The integration of the Internet of Things (IoT) into parking systems has given rise to Smart Parking, a technology-driven solution that optimizes the way we find, reserve, and utilize parking spaces. As we delve into the world of web development for the "Smart Parking System Using IoT," we embark on a journey that harnesses the power of IoT components like server motors, IR sensors, and ESP8266 devices to create an intelligent, data-driven parking ecosystem. This ecosystem necessitates the development of web applications that serve as the bridge between IoT components and end-users, providing real-time information on parking availability, reservations, and navigation. The development of these web applications plays a pivotal role in transforming the conventional parking experience into one that is dynamic, efficient, and responsive to the needs of modern cities. In this exploration of web development for Smart Parking, we will uncover how the fusion of IoT and web technologies not only enhances convenience for drivers but also contributes to effective traffic management, environmental sustainability, and the realization of smarter, more connected urban environments.

DESIGN FOR SMART PARKING MANAGEMENT:

COMPONENTS REQUIRED:

HARDWARE:

- Node MCU ESP8266
- IR Sensor- 5 nos
- Servo Motor-2nos



ESP8266

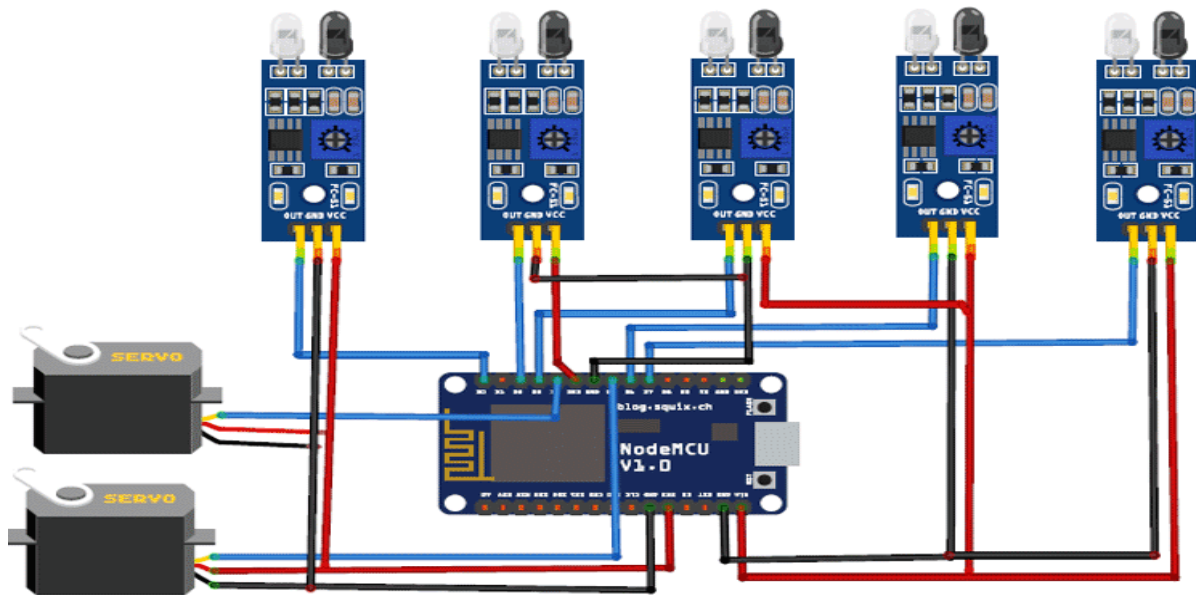


IR sensor



Servo motor

CIRCUIT DIAGRAM:



In this Smart Parking System using IOT, we are using five IR Sensors and two servo motors. IR sensors and Servo motors are connected to the NodeMCU. NodeMCU controls the complete process and sends the parking availability and parking time information to Adafruit IO so that it can be monitored from anywhere in the world using this platform. Two IR sensors are used at entry and exit gate so that it can detect the cars at entry and exit gate and automatically open and close the gate. Two servo motors are used as entry and exit gate, so whenever the IR sensor detects a car, the servo motor automatically rotates from 45° to 140°, and after a delay, it will return to its initial position. Another three IR sensors are used to detect if the parking slot is available or occupied and send the data to NodeMCU. Adafruit IO dashboard also has two buttons to manually operate the entry and exit gate.

CODE:

1. HTML (index.html):

```
html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Smart Parking System</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Smart Parking System</h1>
  <div class="parking-spot" id="spot1"></div>
  <div class="parking-spot" id="spot2"></div>
  <!-- Add more parking spots as needed -->
  <script src="script.js"></script>
</body>
</html>
```

2. CSS (style.css):

```
css
body {
  text-align: center;
  font-family: Arial, sans-serif;
}
.parking-spot {
  width: 100px;
  height: 100px;
  margin: 10px;
  display: inline-block;
  background-color: green;
```

```
}  
.occupied {  
  background-color: red;  
}
```

3. JavaScript (script.js):

```
javascript  
  
// Function to update parking spot status  
function updateParkingSpot(spotId, isOccupied) {  
  const spot = document.getElementById(spotId);  
  if (isOccupied) {  
    spot.classList.add('occupied');  
  } else {  
    spot.classList.remove('occupied');  
  }  
}  
  
// WebSocket connection to ESP8266  
const socket = new WebSocket('ws://your-esp8266-ip-address');  
socket.addEventListener('open', (event) => {  
  console.log('Connected to ESP8266');  
});  
socket.addEventListener('message', (event) => {  
  const data = JSON.parse(event.data);  
  updateParkingSpot(data.spotId, data.isOccupied);  
});
```

4. ESP8266 Code:

You'll need code for the ESP8266 to collect data from the IR sensor and servo motor and send it to the server.

```
cpp  
  
#include <ESP8266WiFi.h>  
#include <WebSocketsClient.h>  
  
const char* ssid = "your_SSID";
```

```

const char* password = "your_PASSWORD";
const char* serverAddress = "your_server_address";
WebSocketsClient webSocket;

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }
    webSocket.begin(serverAddress, 80, "/");
    webSocket.onEvent(webSocketEvent);
}

void loop() {
    webSocket.loop();
    // Code for reading IR sensor and controlling servo motor goes here
}

void webSocketEvent(WStype_t type, uint8_t * payload, size_t length) {
    switch(type) {
        case WStype_DISCONNECTED:
            Serial.println("Disconnected from server");
            break;
        case WStype_CONNECTED:
            Serial.println("Connected to server");
            break;
        default:
            break;
    }
}

```

5. Server (Node.js with Express):

You'll need a server to handle WebSocket connections and serve the web page. Create a new directory for your project, navigate into it, initialize a new Node.js project with ``npm init -y``, and install necessary packages with ``npm install express ws``.

Create a file named `server.js` with the following content:

```
javascript
const express = require('express');
const http = require('http');
const WebSocket = require('ws');
const app = express();
const server = http.createServer(app);
const wss = new WebSocket.Server({ server });
app.use(express.static('public'));
wss.on('connection', (socket) => {
  console.log('Client connected');

  socket.on('close', () => {
    console.log('Client disconnected');
  });
});
server.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

WORKING OF THE CODE:

HTML (`index.html`):

This HTML file serves as the front-end of the Smart Parking System web application. It defines the structure of the page, including the title, parking spot display elements, and references to the associated CSS and JavaScript files.

CSS (`style.css`):

The CSS file defines the visual styling of the parking spot elements. It sets the initial appearance of the parking spots with a green background, and when a parking spot is marked as occupied, it changes the background color to red. This is done using CSS classes.

JavaScript (script.js):

The JavaScript file contains two key functions. The `updateParkingSpot` function is responsible for updating the visual status of a parking spot based on whether it's occupied or vacant. The script establishes a WebSocket connection to an ESP8266 device, listens for messages from it, and updates the corresponding parking spot on the web page in real-time based on the received data.

ESP8266 Code:

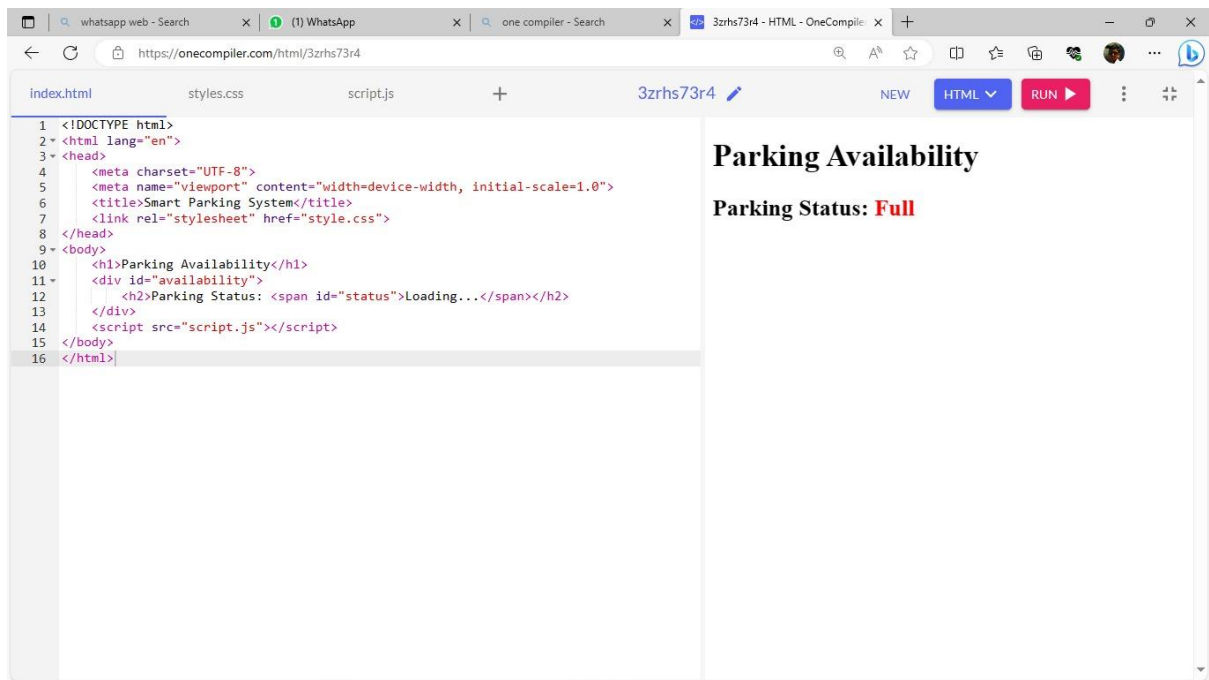
The ESP8266 code is responsible for reading data from IoT components like IR sensors and servo motors. It sends this data to the server via WebSocket messages, indicating whether a parking spot is occupied or vacant. The ESP8266 establishes a WebSocket connection with the server to facilitate real-time communication.

Server (Node.js with Express):

The server-side code is built using Node.js with Express. It serves the HTML, CSS, and JavaScript files to clients, making the web application accessible. It also creates a WebSocket server using the 'ws' library, allowing communication between the ESP8266 and the web page. The server handles WebSocket connections, logs connection and disconnection events, and serves the web application on port 3000.

This code setup creates a real-time Smart Parking System using IoT. The web page displays parking spot statuses and updates them in real-time as data is received from IoT components via WebSocket. The ESP8266 collects data from sensors and communicates it to the server, which, in turn, updates the web page for users to check parking spot availability.

OUTPUT:



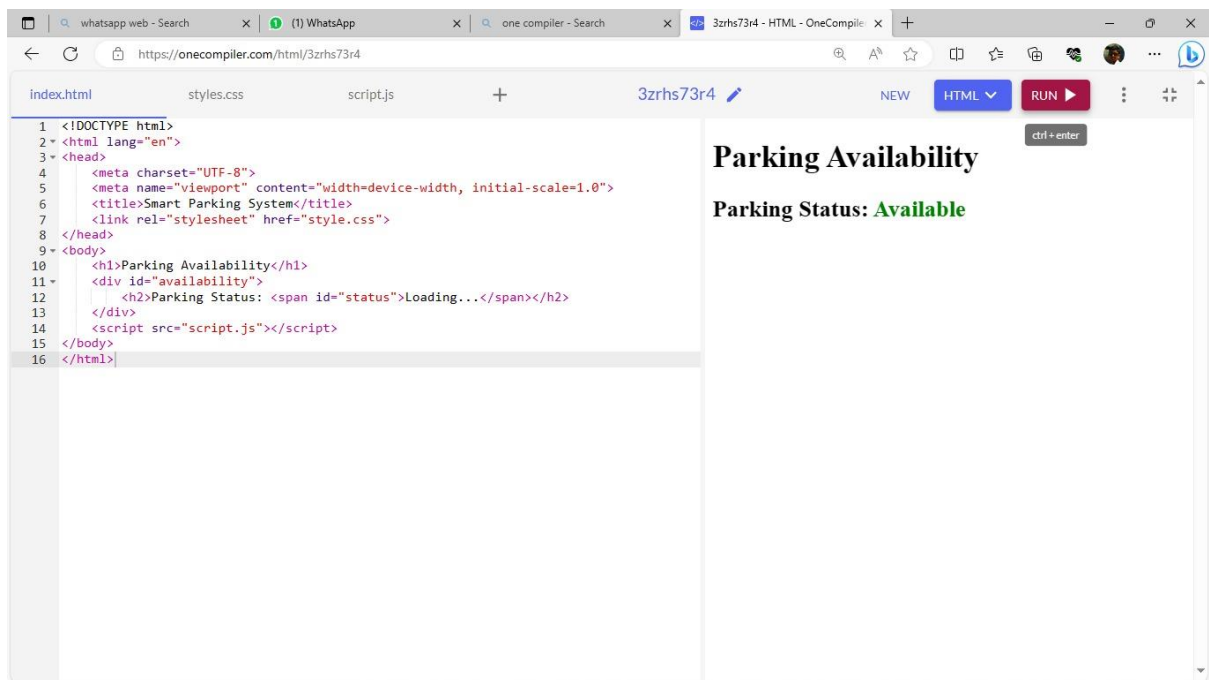
The screenshot shows the OneCompiler interface with the following HTML code in the editor:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Smart Parking System</title>
7   <link rel="stylesheet" href="style.css">
8 </head>
9 <body>
10  <h1>Parking Availability</h1>
11  <div id="availability">
12    <h2>Parking Status: <span id="status">Loading...</span></h2>
13  </div>
14  <script src="script.js"></script>
15 </body>
16 </html>
```

The browser preview on the right displays the rendered output:

Parking Availability

Parking Status: **Full**



The screenshot shows the OneCompiler interface with the same HTML code as the first screenshot. The browser preview on the right now displays the updated output:

Parking Availability

Parking Status: **Available**

A small tooltip with the text "ctrl + enter" is visible near the top right of the browser preview area.

CONCLUSION:

In the context of modern urban living, where effective parking management is an increasingly critical concern due to rising traffic congestion, the fusion of web development and IoT technology has given birth to the innovative concept of Smart Parking. This dynamic solution optimizes the discovery, reservation, and utilization of parking spaces. It is within this transformative landscape that the code provided illustrates a comprehensive system where IoT components, including server motors, IR sensors, and the ESP8266 device, work in harmony to create an intelligent, data-driven parking ecosystem. Web development acts as the crucial bridge between these IoT components and end-users, providing real-time parking availability, reservation capabilities, and navigation guidance. As a result, the conventional parking experience is elevated to one that is responsive, efficient, and adept at addressing the evolving needs of contemporary cities. This amalgamation of IoT and web technologies not only enhances driver convenience but also contributes significantly to robust traffic management, environmental sustainability, and the realization of smarter, interconnected urban environments.