VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Team Members :

22BIT0672 - Sathish K

22BIT0665 - kulasekara Muthu T

22BIT0671 - Balachandar Kv

22BIT0683 - Nithish

22BIT0676 - Sudharsanan G

Parking Management System In a Mall-Project Report

Abstract

This project presents a cloud-based **Parking Management System** that leverages IoT technology to detect real-time parking spot availability and enables users to reserve parking spaces. The system integrates ultrasonic sensors for detecting parking space occupancy, AWS IoT Core for handling sensor data, and AWS Lambda for backend logic. The status of each parking spot is updated in DynamoDB, and users can interact with the system through a web-based frontend to view parking availability or reserve a spot. The project demonstrates the application of IoT and serverless computing for efficient parking management in urban environments, providing an automated and scalable solution for smart cities.

1. Introduction

With the increasing number of vehicles, parking space management has become a significant challenge, especially in urban areas. The **Parking Management System** project aims to simplify and automate the process of managing parking spaces using IoT technology and cloud services. This system provides real-time parking space availability, manages reservations, and integrates sensor data to ensure seamless parking operations.

This report provides a comprehensive overview of the system, its architecture, functionality, implementation, and potential future enhancements.

2. Objectives

The primary objective of this project is to design and implement a scalable parking management system that:

- Detects parking space availability using IoT sensors.
- Provides real-time updates of parking spots through a web application.
- Allows users to reserve parking spaces.
- Manages reservation times and displays availability accordingly.
- Integrates AWS services such as DynamoDB, Lambda, and IoT Core for backend management.

---

3. System Architecture

The system architecture consists of the following components:

1. **IoT Devices (Ultrasonic Sensors)**: These sensors detect whether a parking spot is occupied or available.
2. **AWS IoT Core**: Facilitates the communication between IoT devices and the cloud infrastructure.
3. **AWS Lambda**: Executes the serverless backend logic, including sensor data handling, database interactions, and API responses.
4. **DynamoDB**: A NoSQL database used to store parking space data, including real-time sensor status and reservation information.
5. **API Gateway**: Exposes API endpoints for the web application to interact with the backend.
6. **Web Application (Frontend)**: Displays real-time parking information and allows users to reserve parking spots.

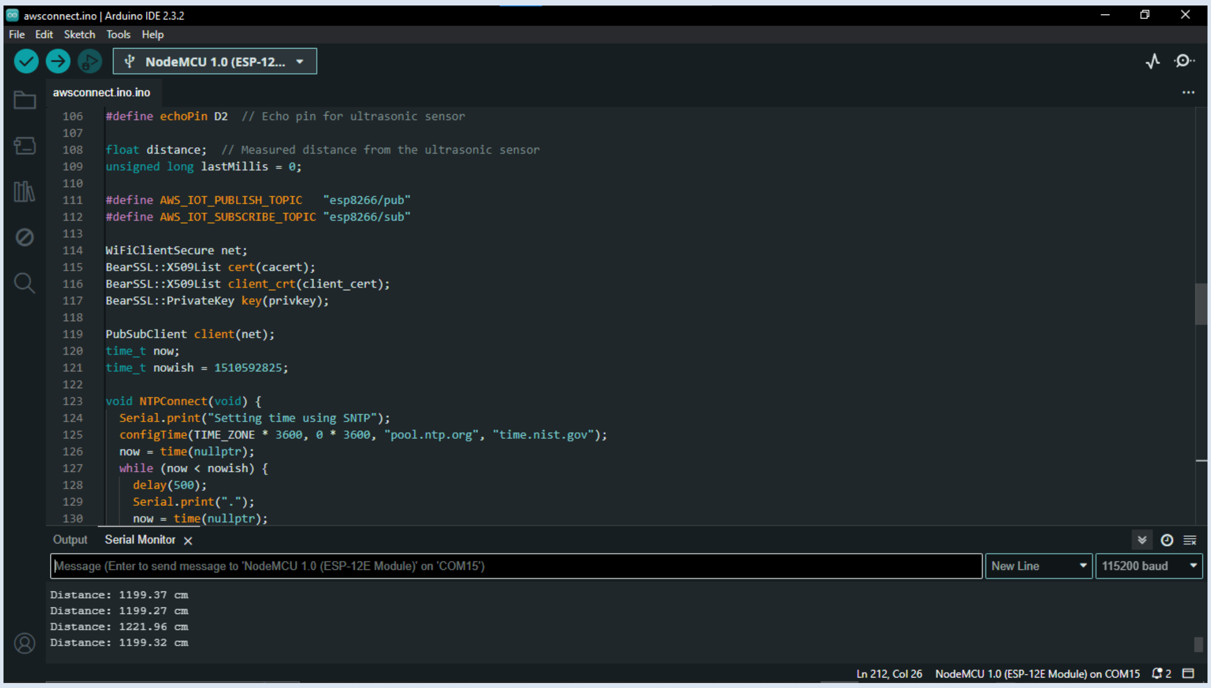
---

4. Implementation Details

4.1 Hardware Components

- **Ultrasonic Sensor**: This sensor is used to detect the occupancy of a parking space.

- **Microcontroller (ESP8266)**: Communicates with AWS IoT Core to transmit the sensor data to the cloud.
- **Programming Implementation in Arduino IDE .**


awsconnect.ino | Arduino IDE 2.3.2
File Edit Sketch Tools Help
NodeMCU 1.0 (ESP-12...
awsconnect.ino.ino

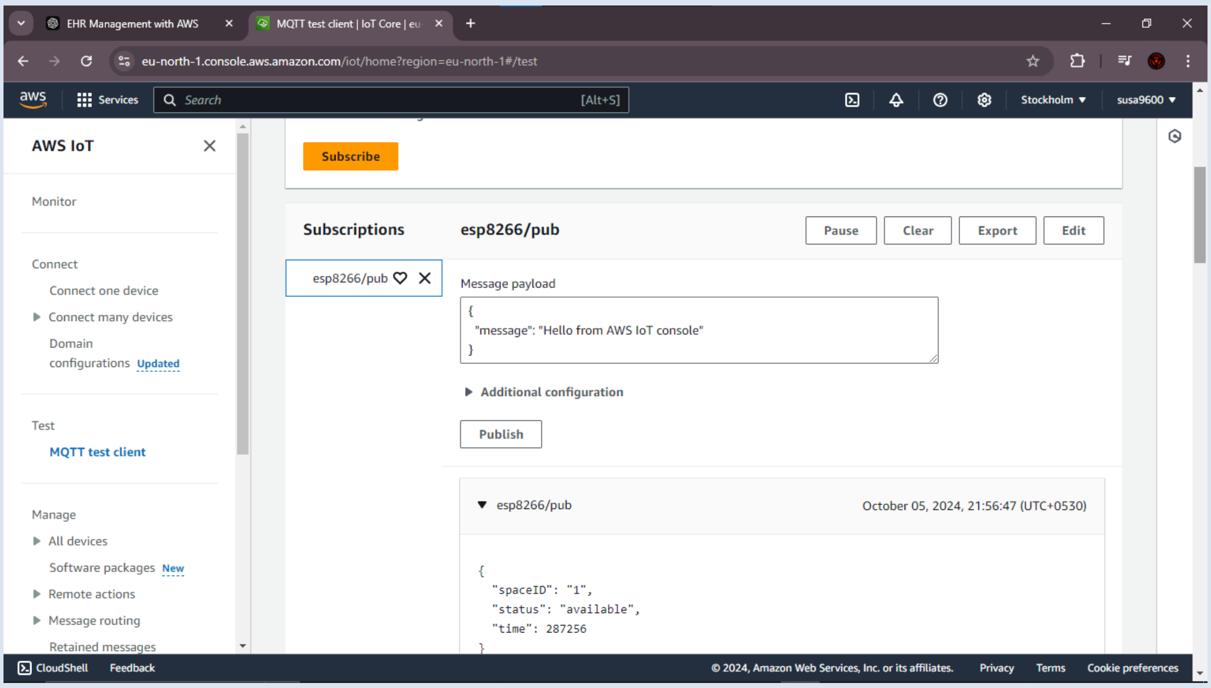
```
#define echoPin D2  // Echo pin for ultrasonic sensor

float distance;  // Measured distance from the ultrasonic sensor
unsigned long lastMillis = 0;

#define AWS_IOT_PUBLISH_TOPIC   "esp8266/pub"
#define AWS_IOT_SUBSCRIBE_TOPIC "esp8266/sub"

WiFiClientSecure net;
BearSSL::X509List cert(cacert);
BearSSL::X509List client_crt(client_cert);
BearSSL::PrivateKey key(privkey);

PubSubClient client(net);
time_t now;
time_t nowish = 1510592825;

void NTPConnect(void) {
  Serial.print("Setting time using SNTP");
  configTime(TIME_ZONE * 3600, 0 * 3600, "pool.ntp.org", "time.nist.gov");
  now = time(nullptr);
  while (now < nowish) {
    delay(500);
    Serial.print(".");
    now = time(nullptr);
```

Output Serial Monitor
Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM15')
New Line
115200 baud

```
Distance: 1199.37 cm
Distance: 1199.27 cm
Distance: 1221.96 cm
Distance: 1199.32 cm
```

4.2 AWS Infrastructure

- **AWS IoT Core**: The sensor data is transmitted to AWS IoT Core. The data is then



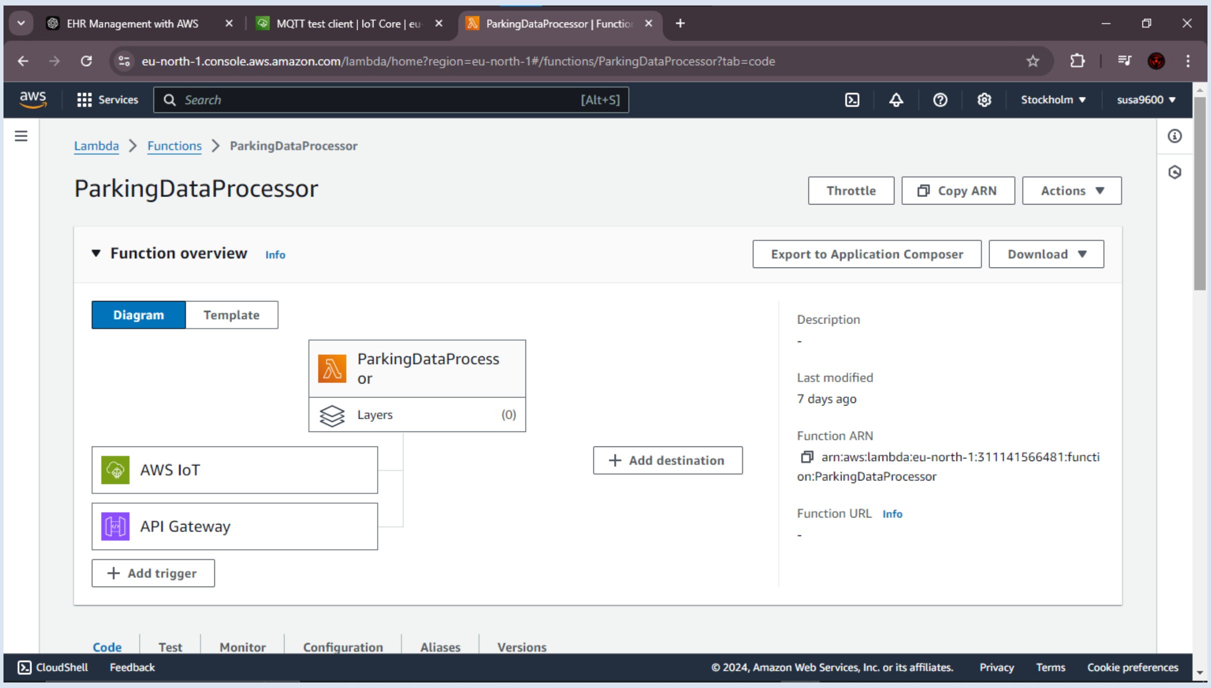
```
{
  "message": "Hello from AWS IoT console"
}
```

esp8266/pub October 05, 2024, 21:56:47 (UTC+0530)

```
{
  "spaceID": "1",
  "status": "available",
  "time": 287256
}
```

routed to a Lambda function that processes the information and stores it in DynamoDB.

- **AWS Lambda**: Two Lambda functions are implemented:
    1. **Sensor Data Processing**: This function stores parking spot availability in the DynamoDB table based on sensor data.
    2. **Reservation Management**: This function handles reservation creation and expiry logic, updating the availability status accordingly.

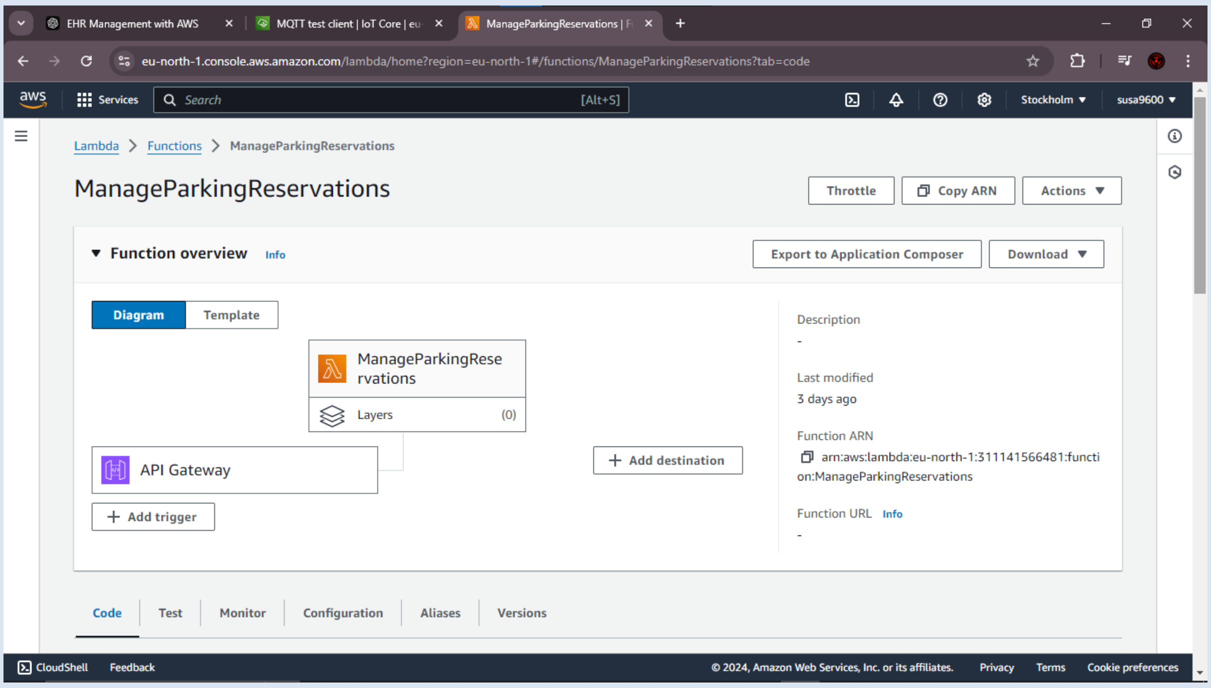


Description
-

Last modified
7 days ago

Function ARN
arn:aws:lambda:eu-north-1:311141566481:function:ParkingDataProcessor

Function URL Info
-


EHR Management with AWS
MQTT test client | IoT Core | eu
ManageParkingReservations | F
eu-north-1.console.aws.amazon.com/lambda/home?region=eu-north-1#/functions/ManageParkingReservations?tab=code
Services
Search
[Alt+S]
Stockholm
susa9600
Lambda > Functions > ManageParkingReservations
ManageParkingReservations
Throttle
Copy ARN
Actions
Function overview
Info
Export to Application Composer
Download
Diagram
Template
ManageParkingReservations
Layers (0)
API Gateway
Add trigger
Add destination
Code
Test
Monitor
Configuration
Aliases
Versions
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Description
-

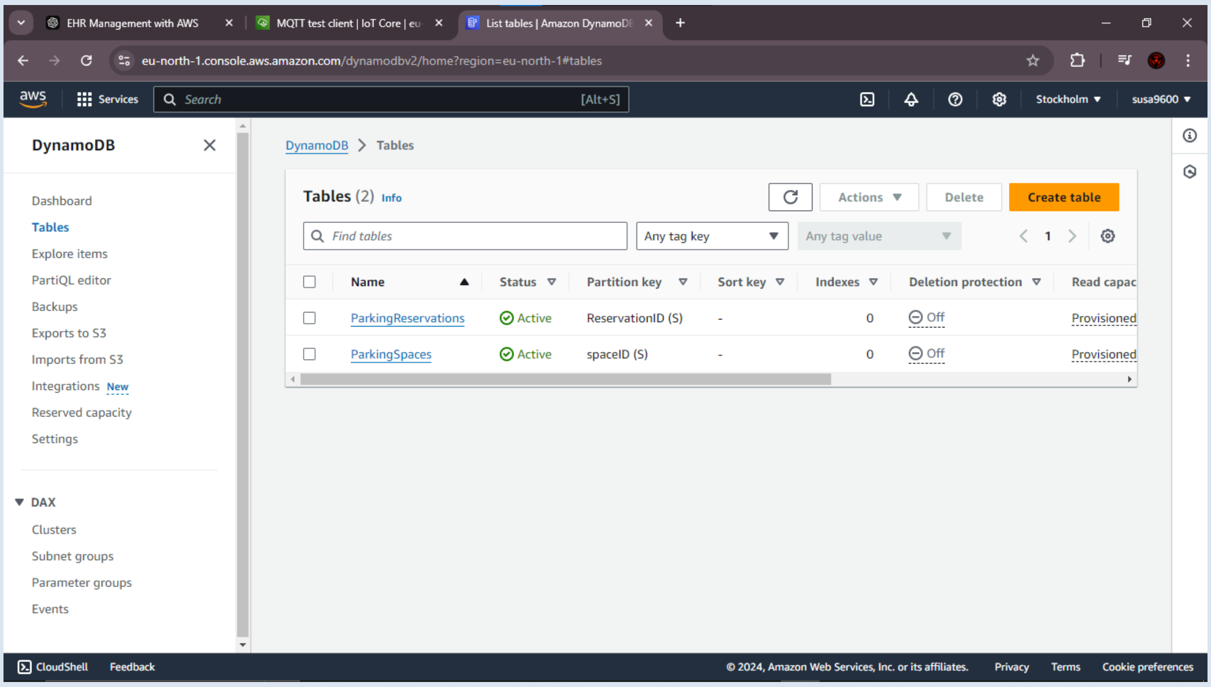
Last modified
3 days ago

Function ARN
arn:aws:lambda:eu-north-1:311141566481:function:ManageParkingReservations

Function URL Info
-

- **DynamoDB**: Two tables are created:

1. **ParkingSpaces Table**: Stores the real-time status of parking spaces (occupied/available).
2. **Reservations Table**: Stores reservation details, including start time and expiry time.


EHR Management with AWS | MQTT test client | IoT Core | eu | List tables | Amazon DynamoDB

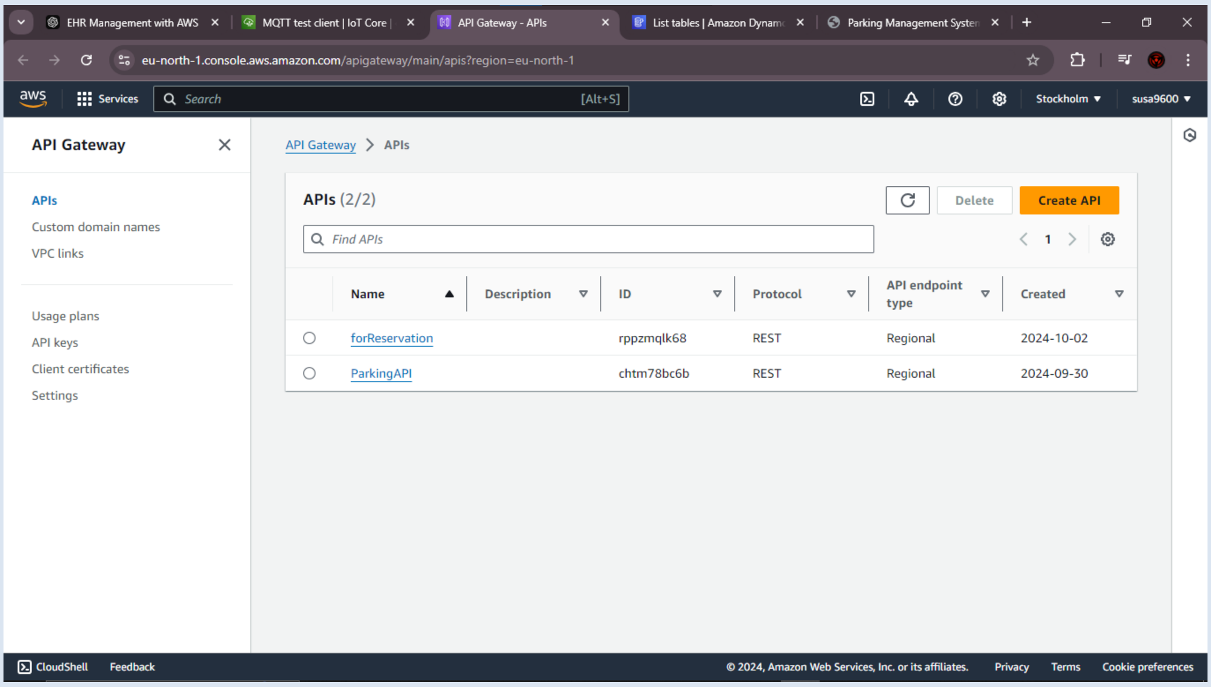
eu-north-1.console.aws.amazon.com/dynamodbv2/home?region=eu-north-1#tables

DynamoDB > Tables

Tables (2) Info

	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capac
☐	ParkingReservations	✓ Active	ReservationID (S)	-	0	Off	Provisioned
☐	ParkingSpaces	✓ Active	spaceID (S)	-	0	Off	Provisioned

API Gateway :


EHR Management with AWS | MQTT test client | IoT Core | API Gateway - APIs | List tables | Amazon Dynam | Parking Management Systen

eu-north-1.console.aws.amazon.com/apigateway/main/apis?region=eu-north-1

API Gateway > APIs

APIs (2/2)

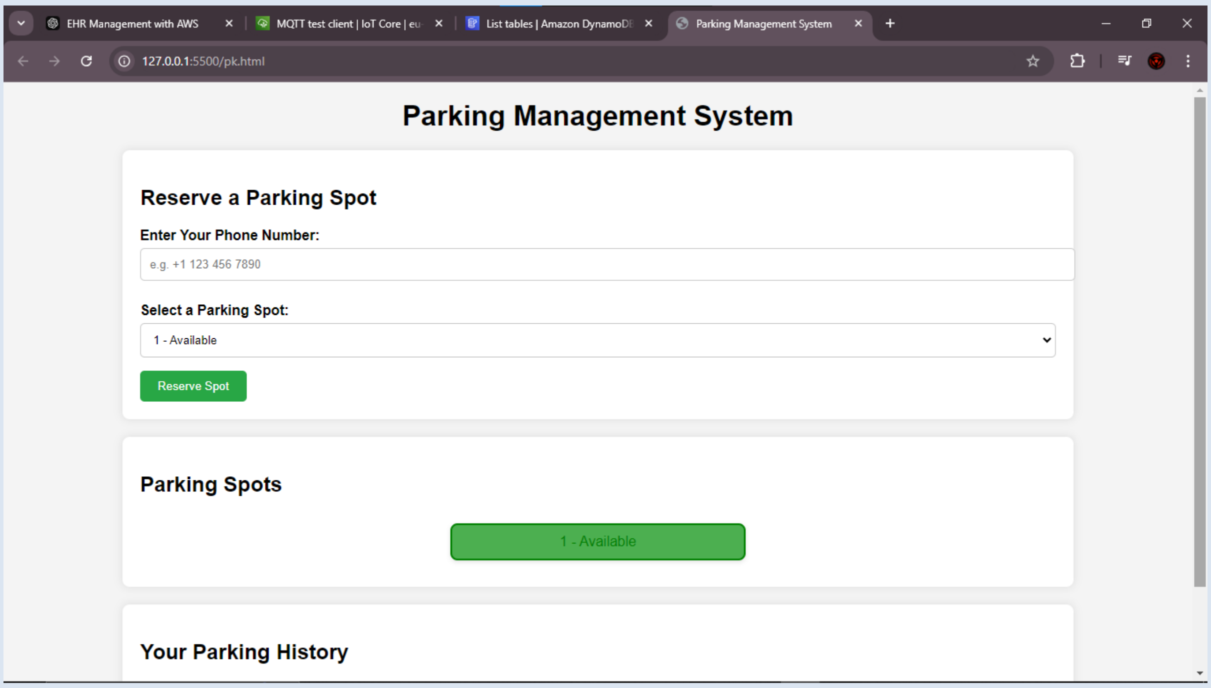
	Name	Description	ID	Protocol	API endpoint type	Created
○	forReservation		rppzmqlk68	REST	Regional	2024-10-02
○	ParkingAPI		chtm78bc6b	REST	Regional	2024-09-30

4.3 Web Application (Frontend)

The web application provides the following functionalities:

- Displays a list of parking spaces with real-time availability updates.

- Allows users to reserve available parking spots.
- Shows a reservation countdown timer, which is displayed until the reservation expires.
- Fetches parking spot data from the backend API every 5 seconds to ensure real-time updates.


EHR Management with AWS
MQTT test client | IoT Core | eu
List tables | Amazon DynamoDB
Parking Management System
127.0.0.1:5500/pk.html

Parking Management System

Reserve a Parking Spot

Enter Your Phone Number:

e.g. +1 123 456 7890

Select a Parking Spot:

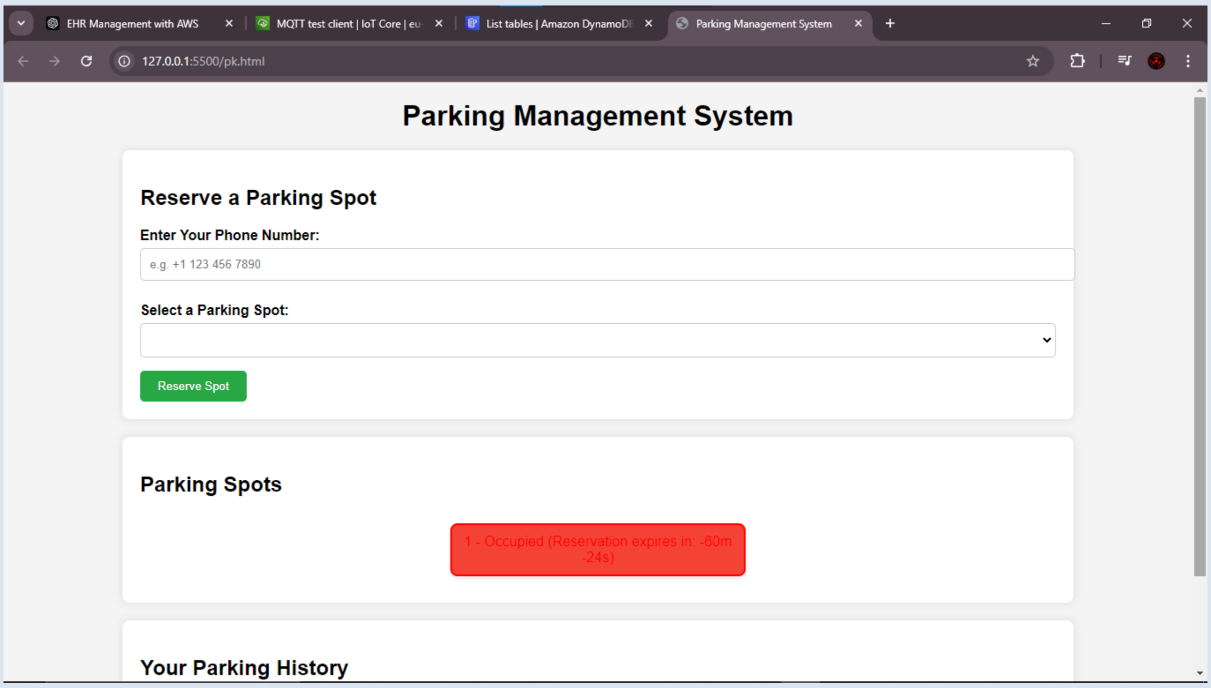
1 - Available

Reserve Spot

Parking Spots

1 - Available

Your Parking History


EHR Management with AWS
MQTT test client | IoT Core | eu
List tables | Amazon DynamoDB
Parking Management System
127.0.0.1:5500/pk.html

Parking Management System

Reserve a Parking Spot

Enter Your Phone Number:

e.g. +1 123 456 7890

Select a Parking Spot:

Reserve Spot

Parking Spots

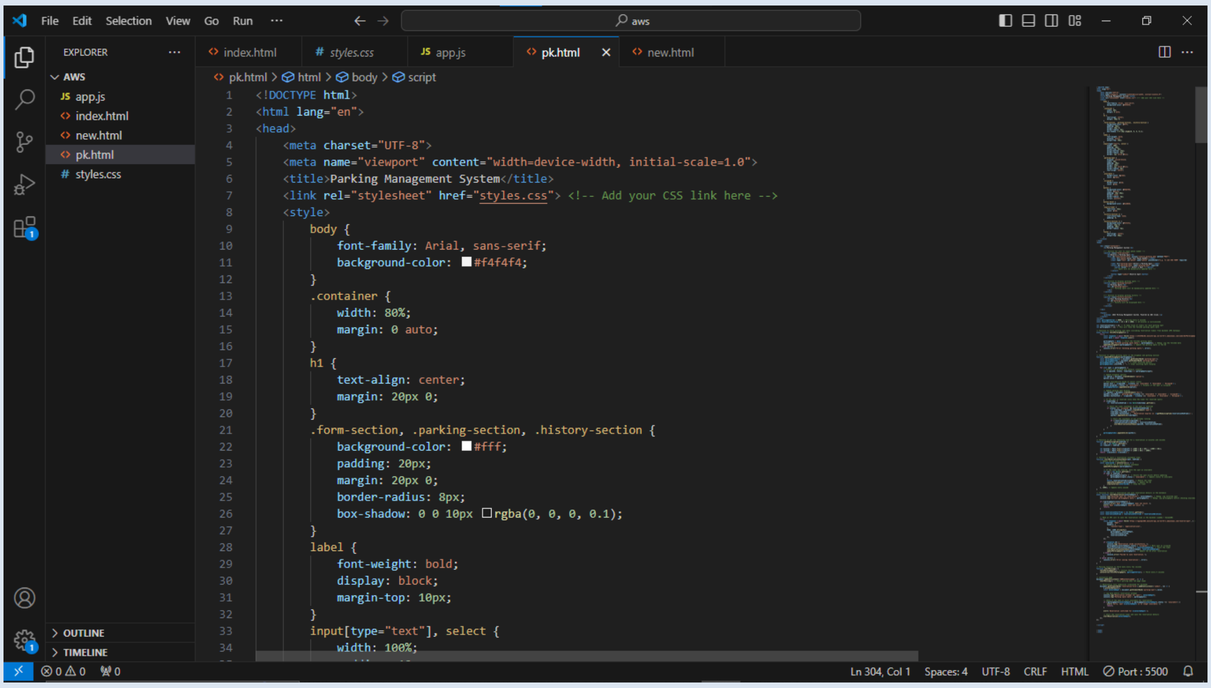
1 - Occupied (Reservation expires in: -60m -24s)

Your Parking History

Frontend Code Overview:

- **JavaScript/HTML**: Used to fetch data from AWS API Gateway and dynamically update the parking spots in the UI.
- **Reservation Logic**: Users can reserve parking spaces if they are available. If a parking space is occupied or reserved, the system prevents further reservations for that spot.

- **Reservation Timer**: Displays a countdown timer for reserved spots.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Parking Management System</title>
    <link rel="stylesheet" href="styles.css"> <!-- Add your CSS link here -->
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
        }
        .container {
            width: 80%;
            margin: 0 auto;
        }
        h1 {
            text-align: center;
            margin: 20px 0;
        }
        .form-section, .parking-section, .history-section {
            background-color: #fff;
            padding: 20px;
            margin: 20px 0;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        label {
            font-weight: bold;
            display: block;
            margin-top: 10px;
        }
        input[type="text"], select {
            width: 100%;
```

---

5. Key Functionalities

1. **Real-Time Parking Spot Detection**:
    - The ultrasonic sensor detects if a parking space is occupied or free.
    - The sensor data is sent to AWS IoT Core, where it is processed by the Lambda function and stored in DynamoDB.
2. **Reservation System**:
    - Users can reserve available parking spots through the web interface.
    - The reservation time is stored in DynamoDB, and the spot is marked as "occupied" for the duration of the reservation.
    - If a parking spot is reserved, the system shows a countdown timer indicating the time left before the reservation expires.
3. **Availability Management**:
    - If a parking spot is occupied based on sensor data, it is shown as unavailable, and the system blocks further reservations.
    - Once the sensor detects that a car has left the spot, the status is updated automatically.

---

6. AWS Lambda Code Snippets

Sensor Data Processing Lambda Function:

This Lambda function processes the data received from AWS IoT Core and updates the parking spot's status in the DynamoDB table.

```
python
Copy code
import json
import boto3
import logging

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('ParkingSpaces')

def lambda_handler(event, context):
    try:
        # Process incoming sensor data
        sensor_data = json.loads(event['body'])
        space_id = sensor_data['spaceID']
        status = sensor_data['status']

        # Update DynamoDB with the current status
        table.update_item(
            Key={'spaceID': space_id},
            UpdateExpression='SET #s = :val',
            ExpressionAttributeNames={'#s': 'status'},
            ExpressionAttributeValues={':val': status}
        )
        return {'statusCode': 200, 'body': json.dumps('Success')}
    except Exception as e:
        logging.error(f"Error: {str(e)}")
        return  {'statusCode':  500,  'body':  json.dumps('Error  processing
data')}
```

Reservation Management Lambda Function:

This Lambda function handles reservation creation, updating the reservation time and status.

```
python
Copy code
import json
import boto3
from datetime import datetime, timedelta

dynamodb = boto3.resource('dynamodb')
reservations_table = dynamodb.Table('Reservations')
spaces_table = dynamodb.Table('ParkingSpaces')

def lambda_handler(event, context):
    try:
        body = json.loads(event['body'])
        space_id = body['parkingSpot']
        reservation_end_time = body['reservationEndTime']

        # Store reservation in DynamoDB
        reservations_table.put_item(
            Item={
                'parkingSpot': space_id,
                'reservationEndTime': reservation_end_time
            }
        )

        # Mark the parking spot as occupied
        spaces_table.update_item(
            Key={'spaceID': space_id},
            UpdateExpression='SET #s = :val',
            ExpressionAttributeNames={'#s': 'status'},
            ExpressionAttributeValues={':val': 'occupied'}
        )

        return {'statusCode': 200, 'body': json.dumps('Reservation successful')}
    except Exception as e:
        return {'statusCode': 500, 'body': json.dumps(f"Error: {str(e)}")}
```

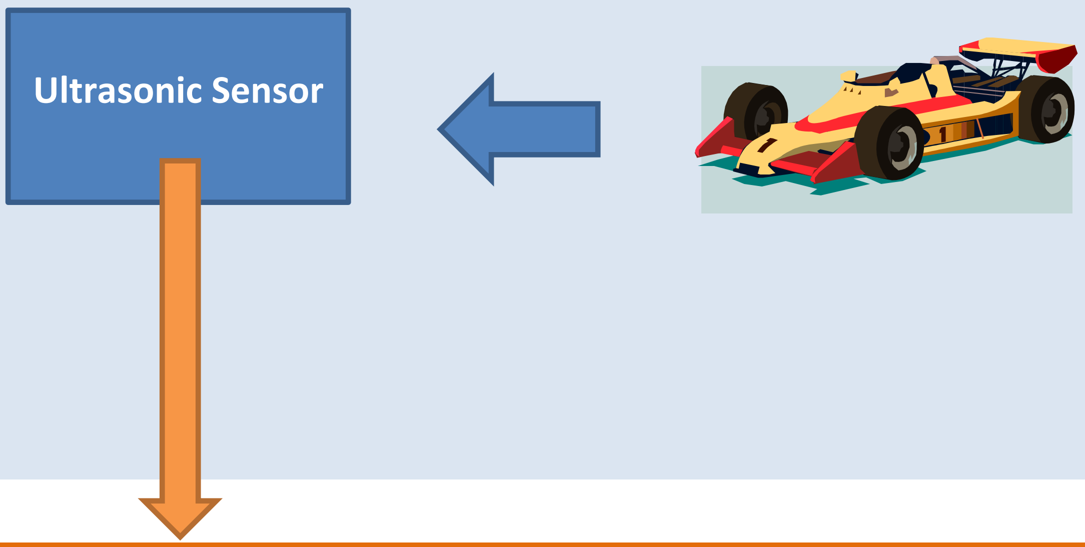
---

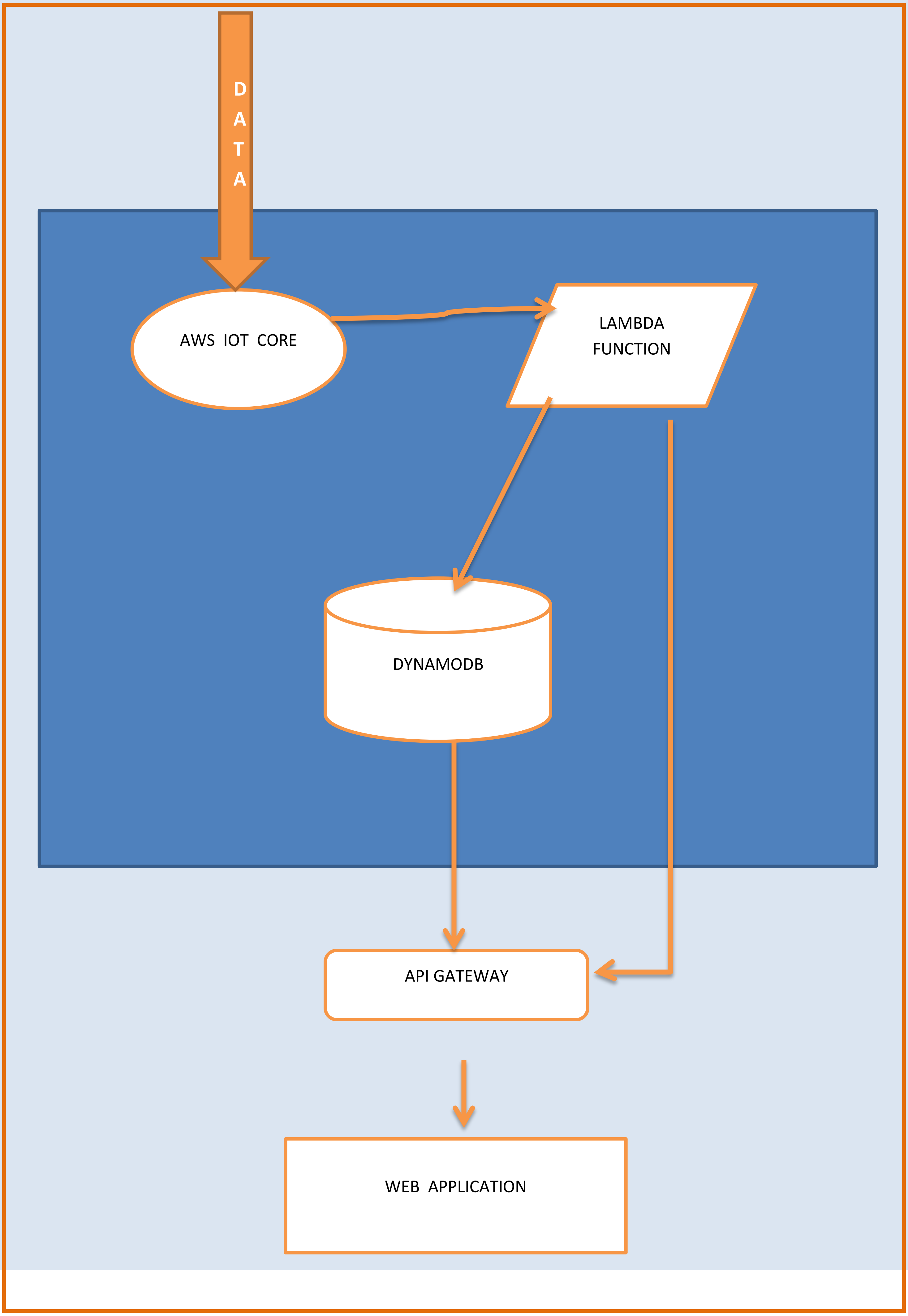
7. Challenges Faced

- **Real-Time Synchronization**: Ensuring the real-time synchronization between the IoT devices, backend, and frontend required a careful design of data flow, especially when multiple users are interacting with the system.
- **Reservation Timer**: Implementing an accurate countdown timer for reservations and handling scenarios where the sensor detects occupation was a challenge. This was addressed by managing separate states for sensor and reservation statuses.
- **CORS Issues**: Initially, there were CORS header issues when the frontend attempted to communicate with the AWS backend. This was resolved by adjusting API Gateway settings.

---

7. workflow :




DATA
AWS IOT CORE
LAMBDA FUNCTION
DYNAMODB
API GATEWAY
WEB APPLICATION

9. Conclusion

The Parking Management System successfully demonstrates the use of IoT and cloud services to manage parking spaces efficiently. It provides real-time updates on parking space availability, offers a reservation system, and integrates AWS services to ensure scalability and reliability. The system has potential for future expansion, including mobile integration, payment processing, and enhanced security features.

Appendix: Technologies Used

- **Frontend**: HTML, JavaScript
- **Backend**: AWS Lambda, API Gateway
- **Database**: AWS DynamoDB
- **IoT**: ESP8266, Ultrasonic Sensor
- **Cloud Platform**: AWS (IoT Core, Lambda, DynamoDB, API Gateway)