

**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****

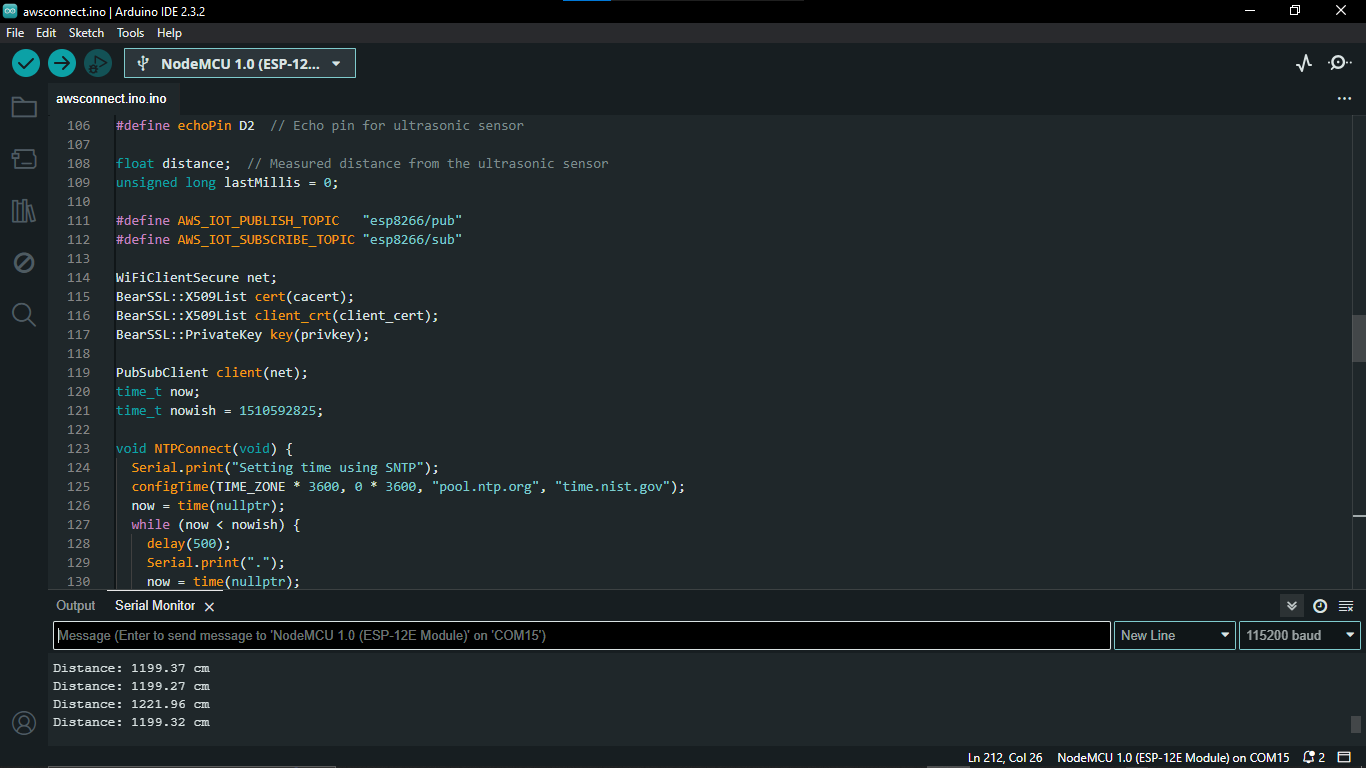
The system architecture consists of the following components:

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****

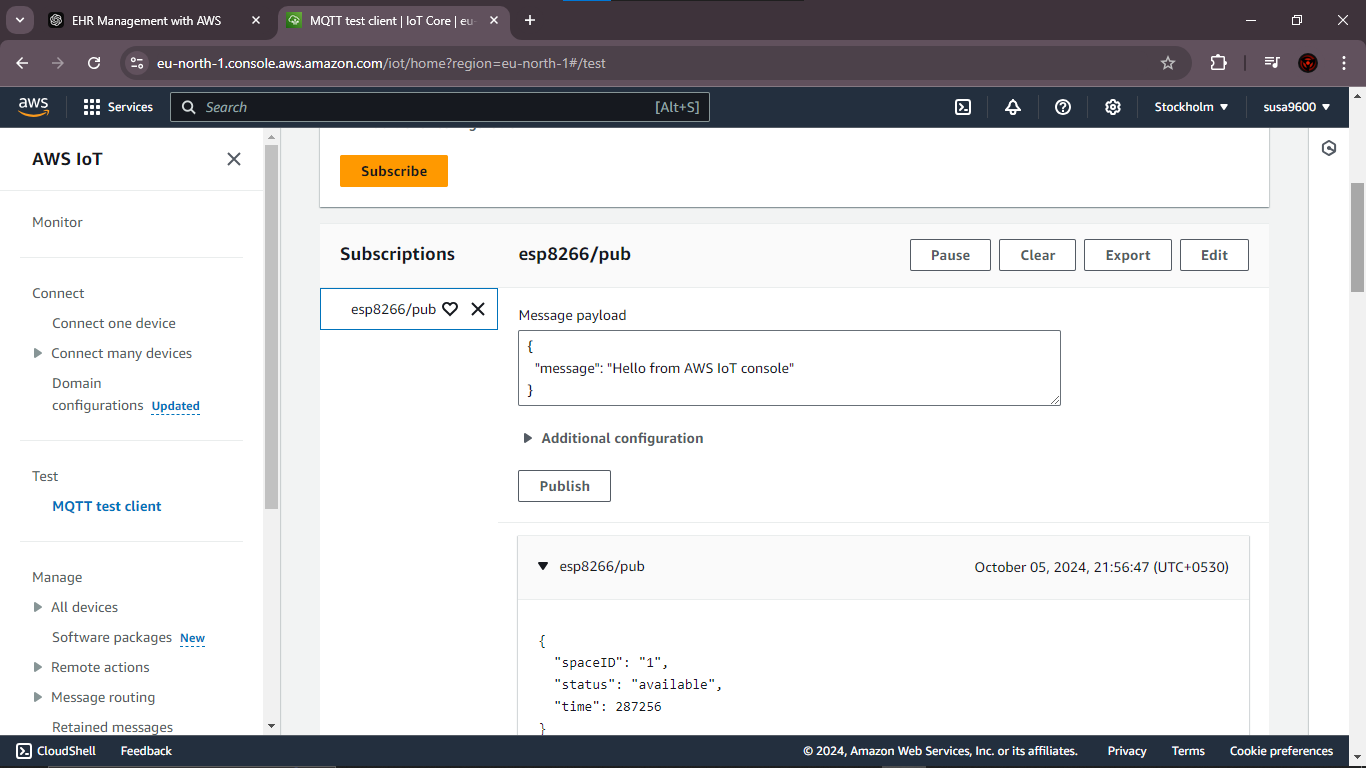
### ****4.1 Hardware Components****

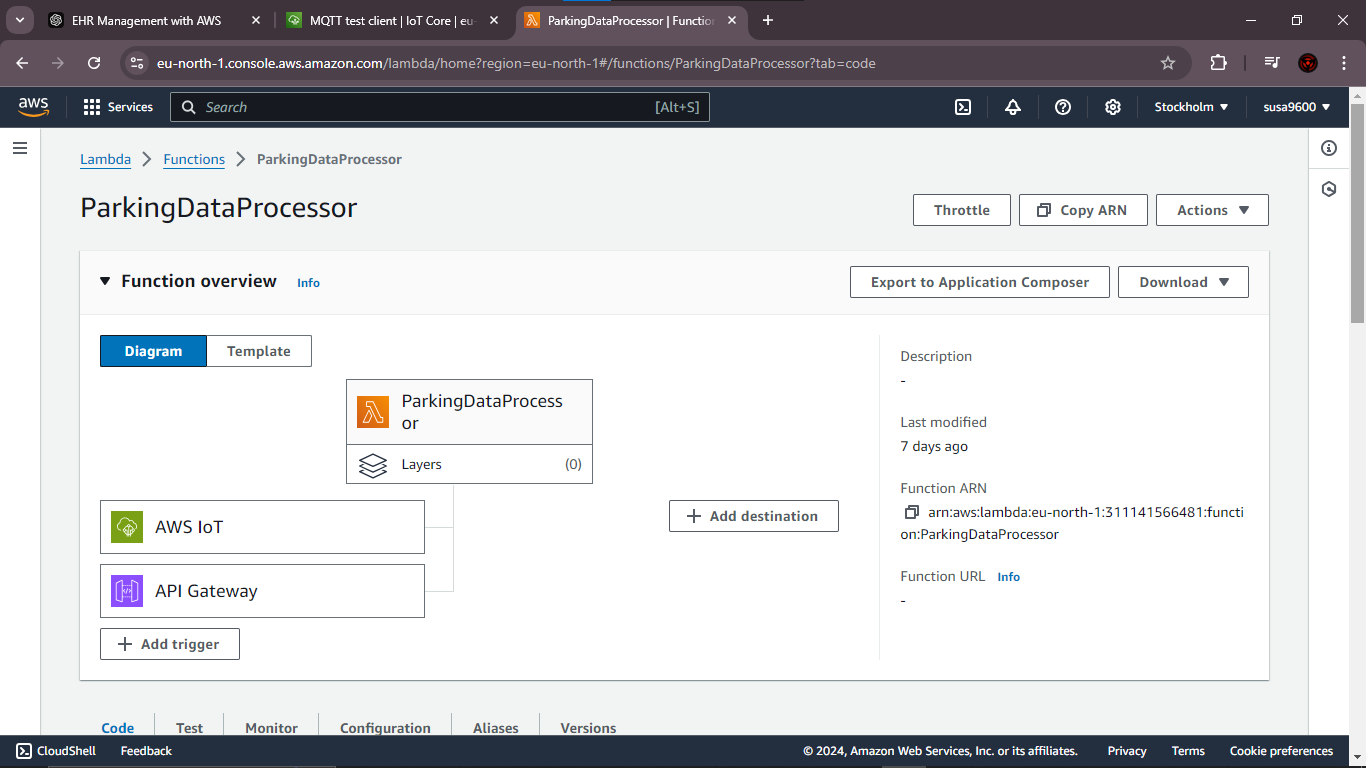
* **Ultrasonic Sensor**: This sensor is used to detect the occupancy of a parking space.
* **Microcontroller (ESP8266)**: Communicates with AWS IoT Core to transmit the sensor data to the cloud.
* **Programming Implementation in Arduino IDE** .

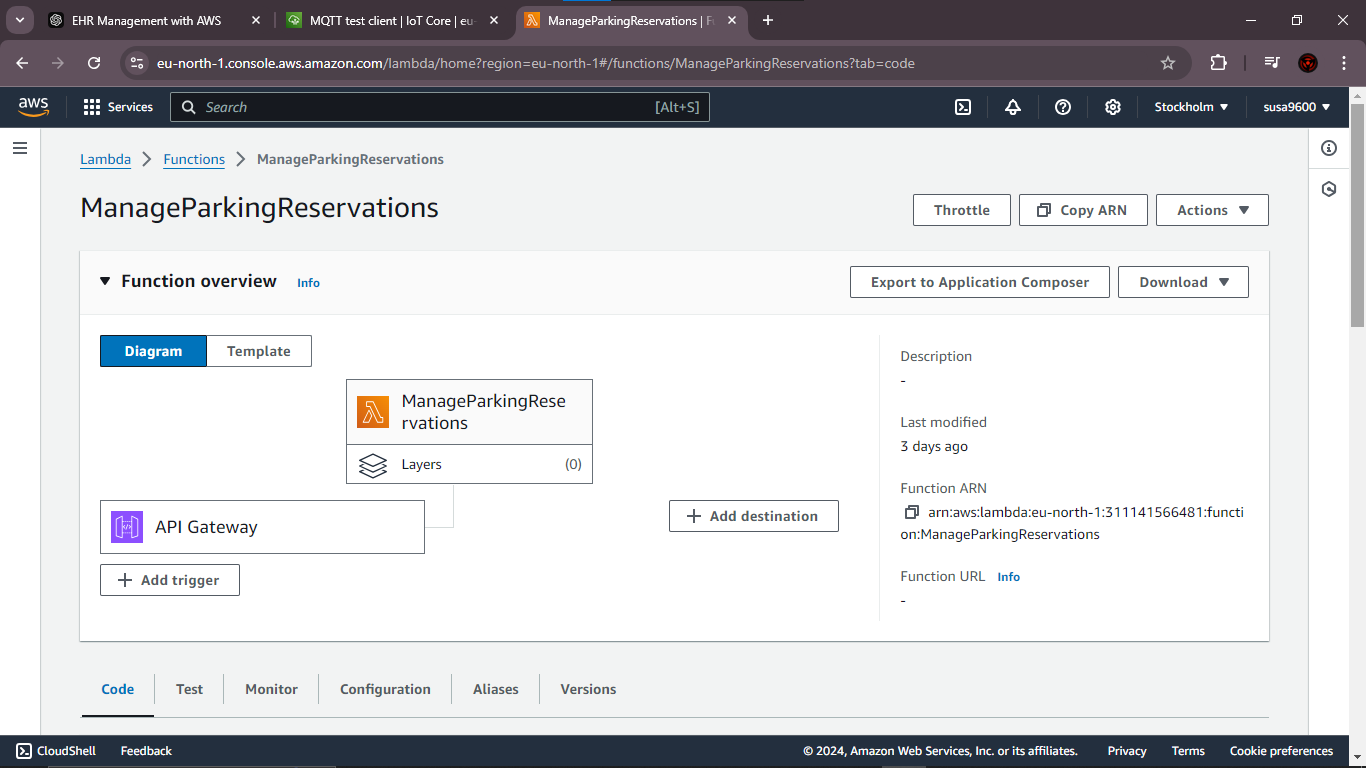


### ****4.2 AWS Infrastructure****

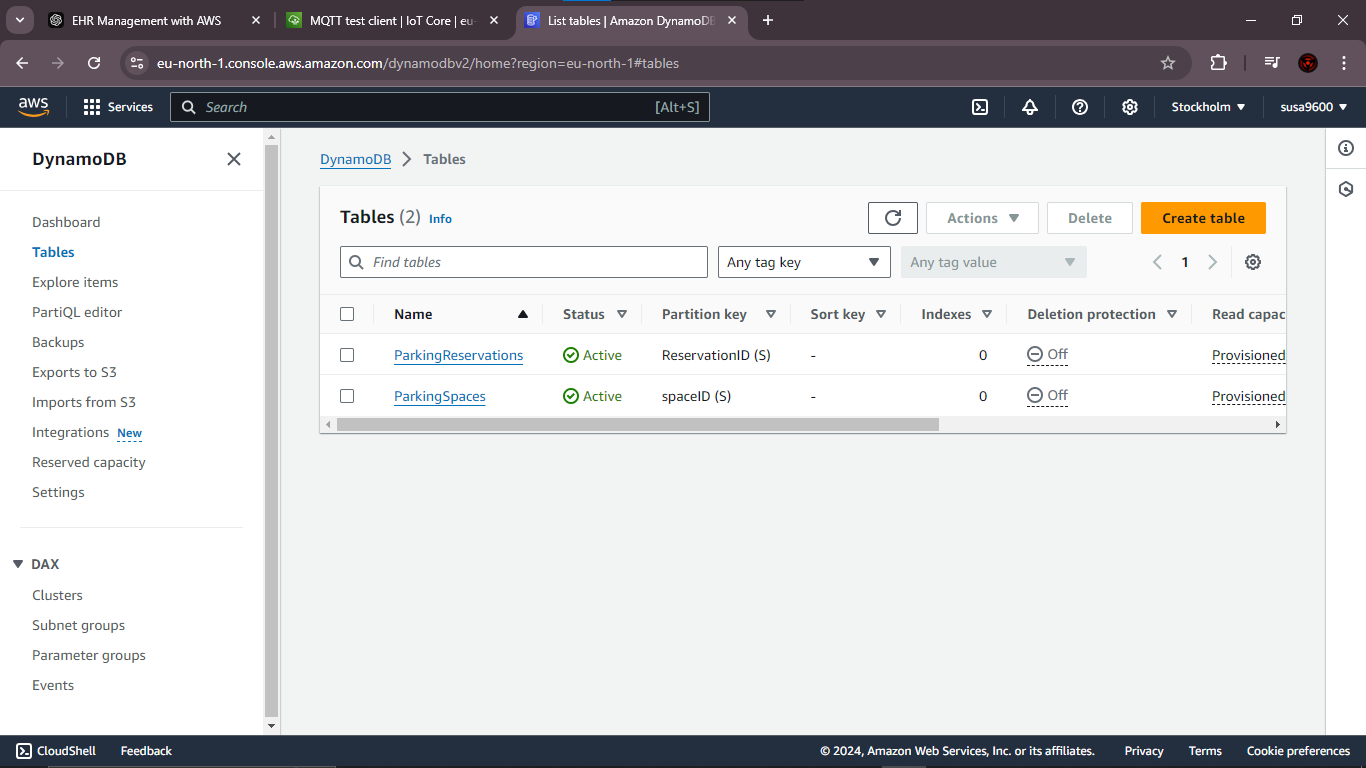
* **AWS IoT Core**: The sensor data is transmitted to AWS IoT Core. The data is then routed to a Lambda function that processes the information and stores it in DynamoDB.



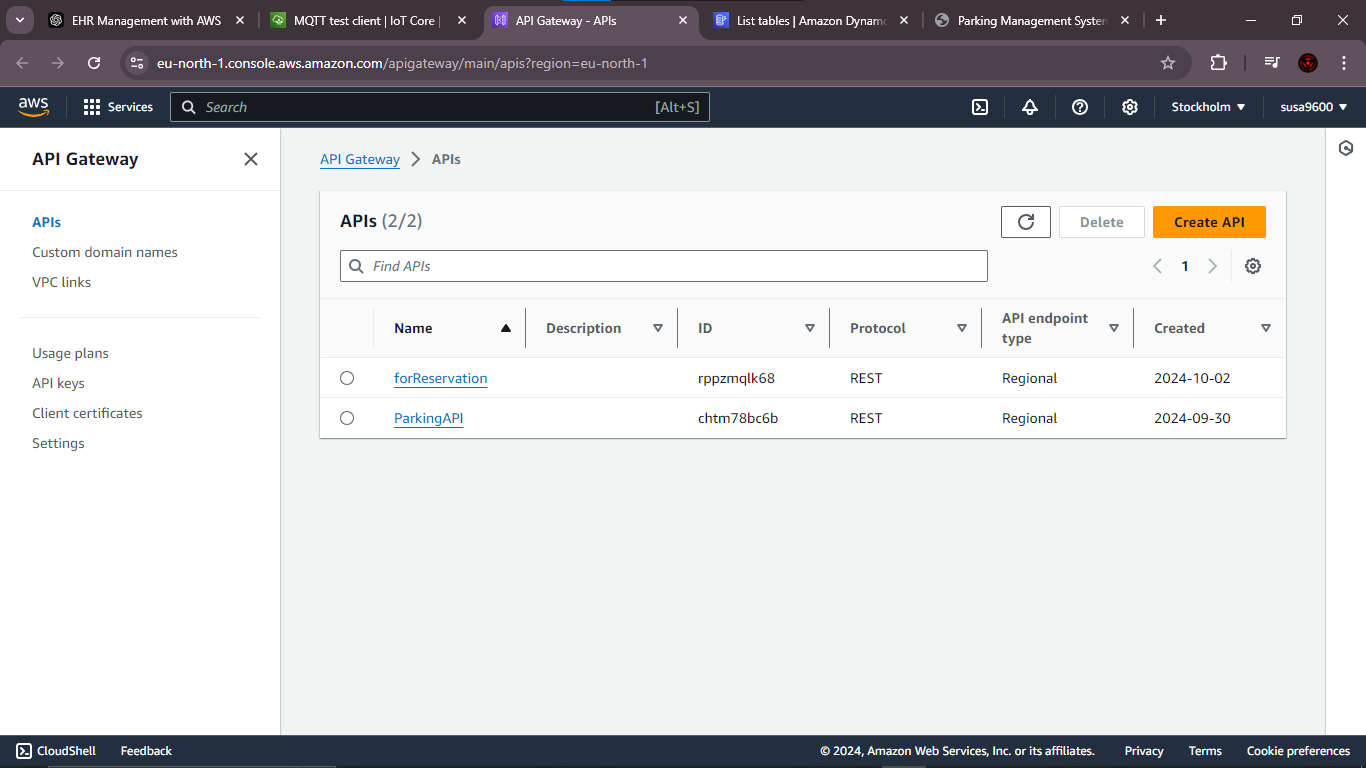
* **AWS Lambda**: Two Lambda functions are implemented:
  1. **Sensor Data Processing**: This function stores parking spot availability in the DynamoDB table based on sensor data.
  2. **Reservation Management**: This function handles reservation creation and expiry logic, updating the availability status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



* **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.



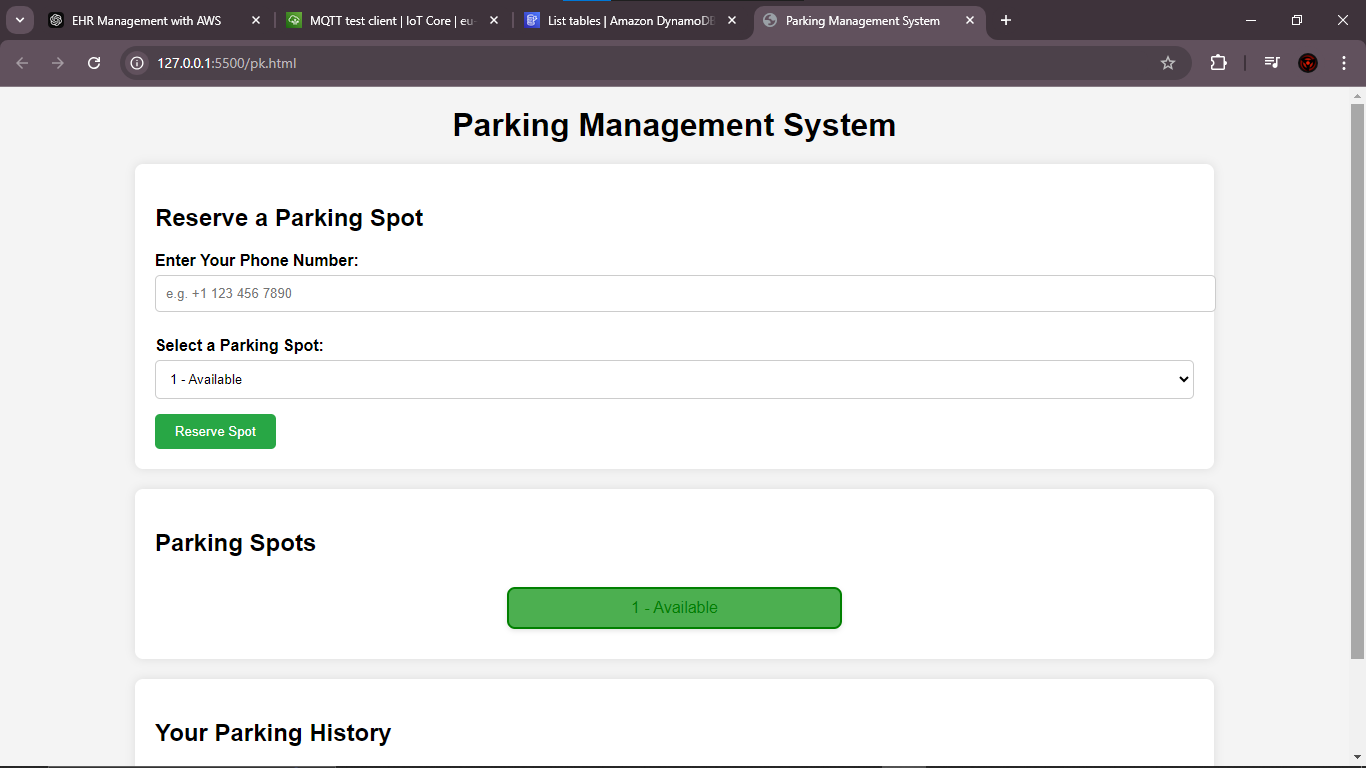
**API Gateway :**

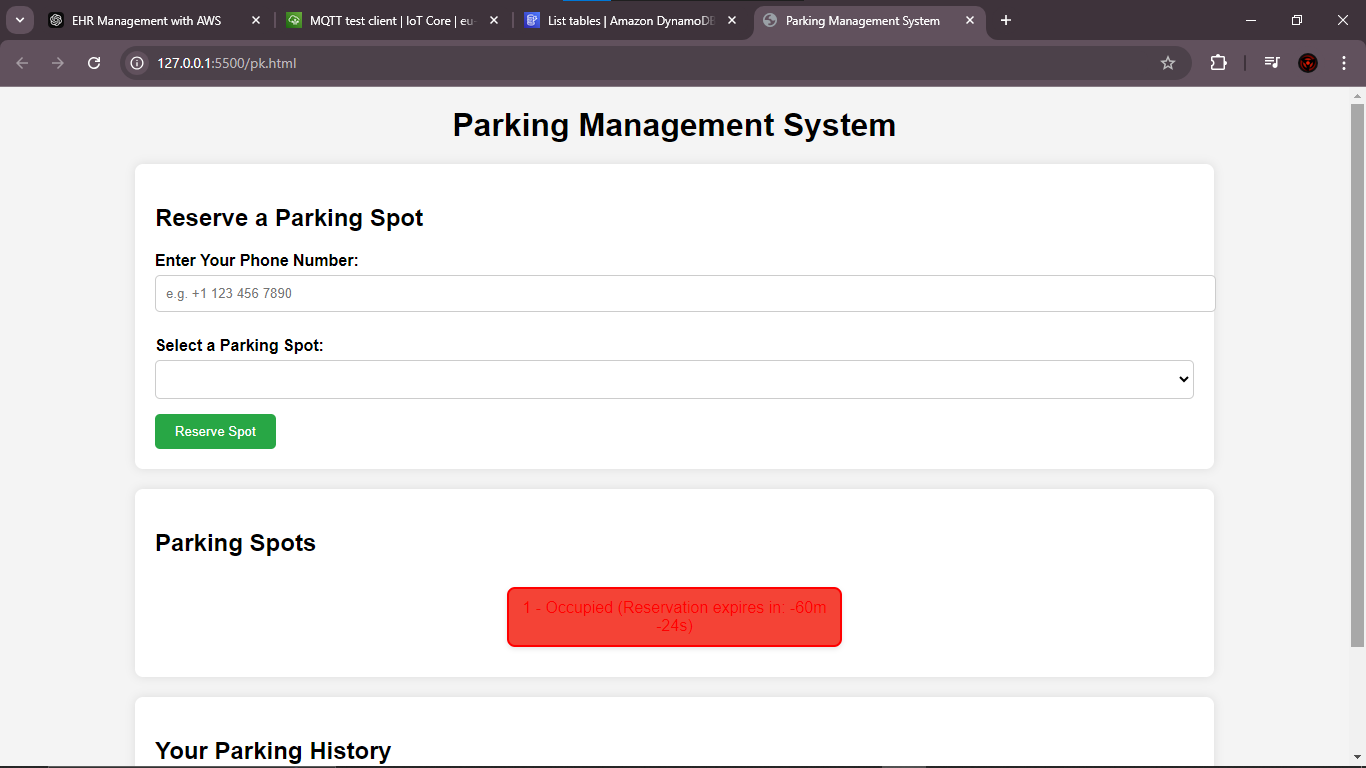
****

### ****4.3 Web Application (Frontend)****

The web application provides the following functionalities:

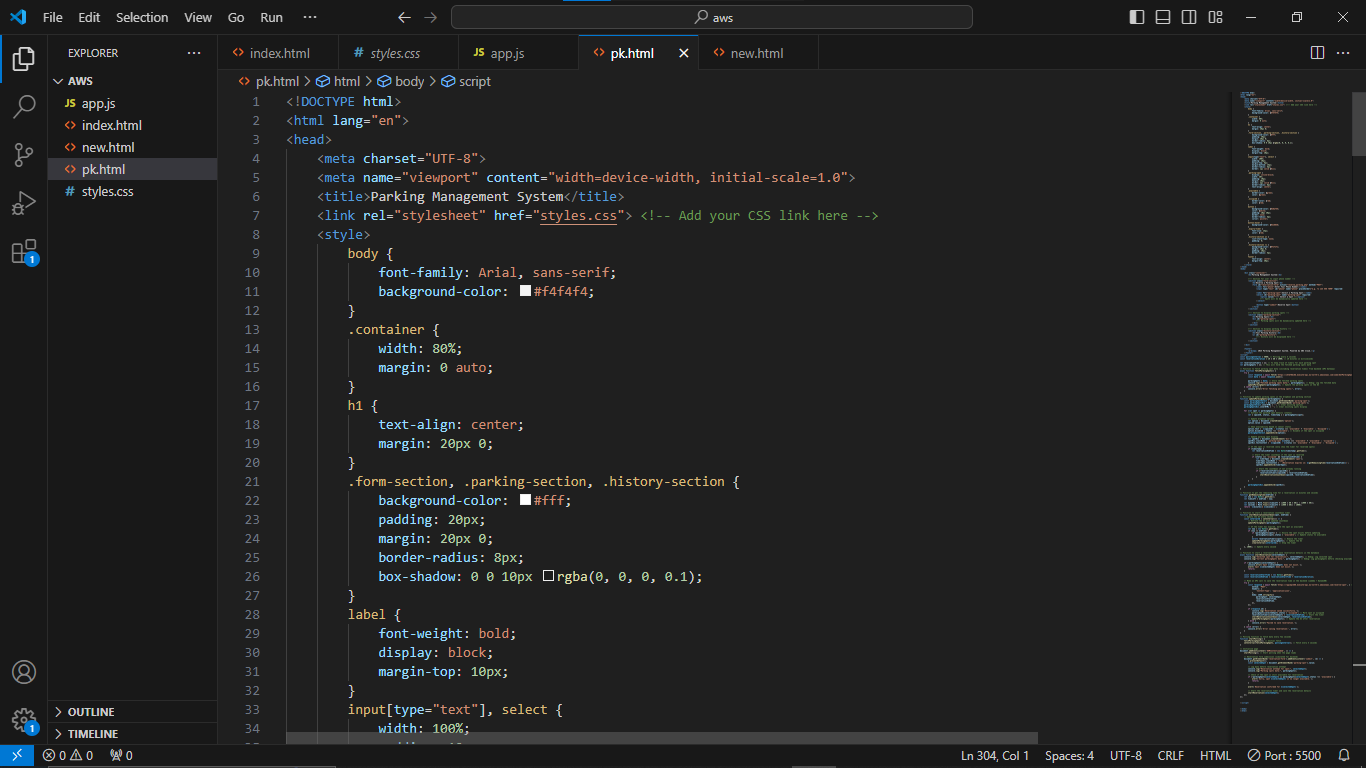
* Displays a list of parking spaces with real-time availability updates.
* Allows users to reserve available parking spots.
* Shows a reservation countdown timer, which is displayed until the reservation expires.
* Fetches parking spot data from the backend API every 5 seconds to ensure real-time updates.





#### ****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.



## ****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.

## ****workflow :****

**Ultrasonic Sensor**

**C:\Program Files\Microsoft Office\MEDIA\CAGCAT10\j0216858.wmf**

**DATA**

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)