Smart Parking System with Raspberry Pi & AWS

Dalair Franzen

May 21, 2025

CLDE2291 – Capstone Project

Problem & Goal

Problem Statement:

Urban campuses like Dunwoody face parking inefficiencies due to a lack of real-time space tracking. This leads to wasted time, confusion, and driver frustration.

Goal:

Build a real-time IoT parking system using Raspberry Pi, Arduino, sensors, AWS, and a web app to improve visibility and space usage.

Solution Overview

System Overview

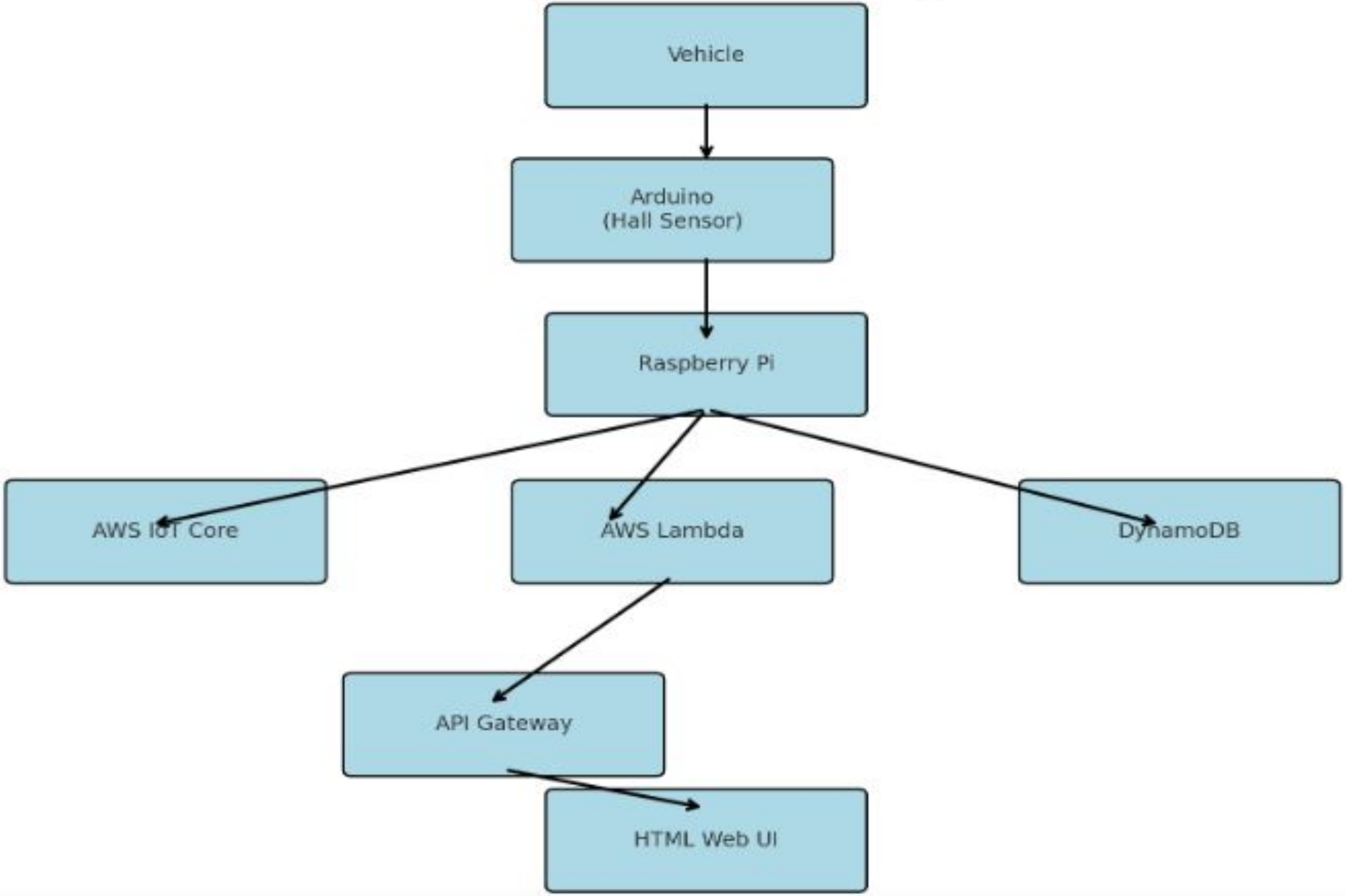
- Raspberry Pi reads parking data from Arduino
- Data is published to AWS IoT Core
- Lambda function stores data into DynamoDB
- API Gateway shows parking status
- HTML frontend displays a live parking grid

Solution Overview

Key Innovations & Value

- Real-time updates with minimal latency
- Low-cost prototype using open-source tools
- Serverless AWS integration which reduces infrastructure overhead
- Scalable to 1,000+ spaces
- User-friendly web UI

Slide 5: System Architecture Diagram



Technology Stack

Hardware:

- **Arduino Uno R3** - reads hall effect sensor
- **Raspberry Pi 4** - Edge device handling MQTT & Python

Software:

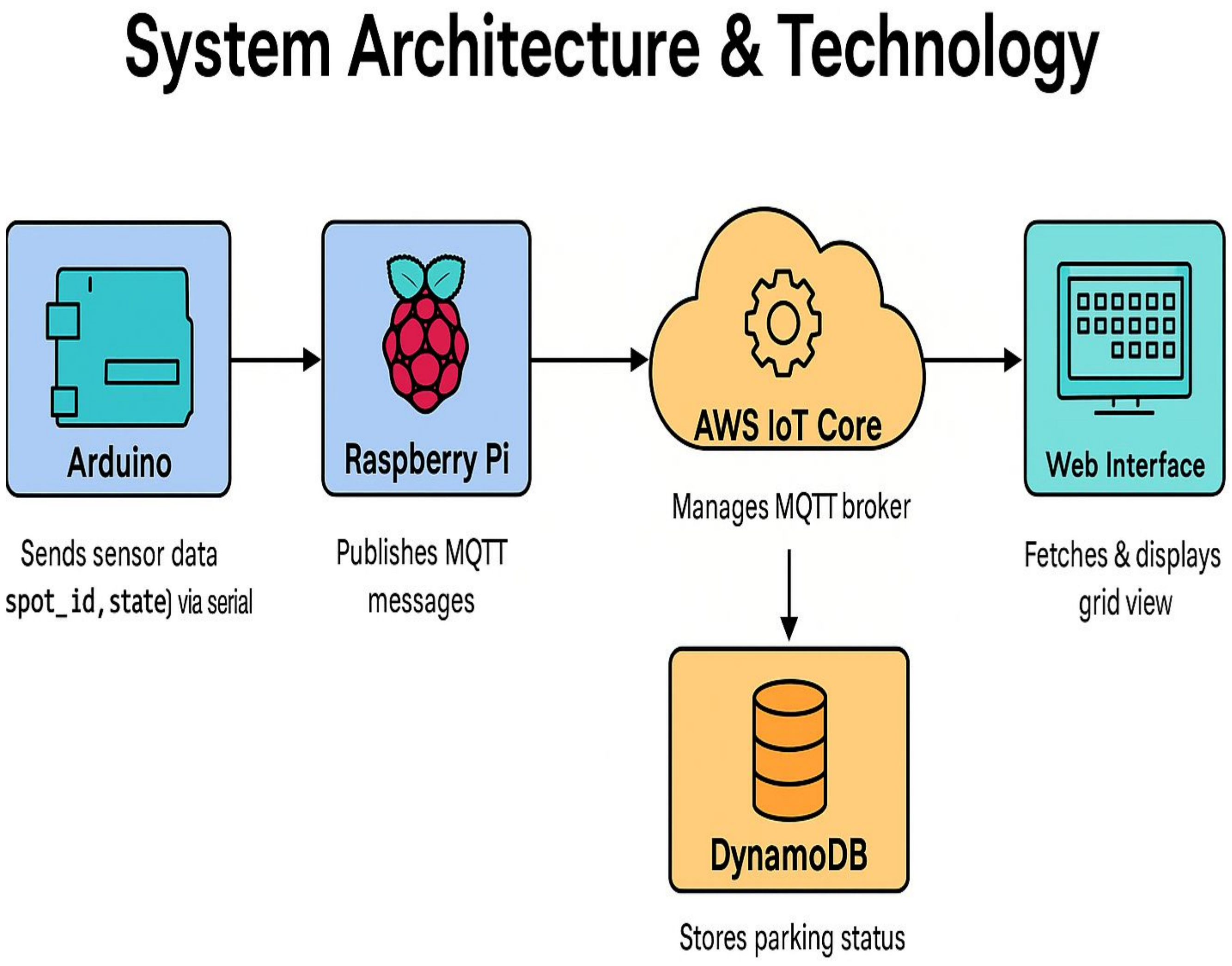
- **Python 3.11** – Core Scripting Language
- **Libraries** -
  - Boto3 (DynamoDB), AWSIoTPythonSDK (MQTT), dotenv (.env variables), pyserial (serial communication)

Cloud Services:

- **AWS IoT Core** - MQTT message broker
- **AWS Lambda** – Fetches parking data from database
- **Amazon DynamoDB** – Stores real-time spot status
- **Amazon API Gateway** – Rest endpoint to frontend
- **CloudWatch** – Monitors performance & errors

Deployment:

- Python script runs on Raspberry Pi
- Static website served over Local HTTP
- API fetches real-time status from DynamoDB



Design Decisions & Trade-offs

Decision	Benefit	Limitation
Raspberry Pi as edge	Handles local data & fast control	Limited processing power
Real-time updates via Pi	Instant updates with low delay	Needs stable power & connection
Static HTML UI	Quick to build and load	No interactivity or user logins

Key Decisions:

Why MQTT + AWS IoT Core?

- Lightweight, and ideal for real-time model sensor data
- Uses secure, certification-based communication

Why DynamoDB?

- Scales easily and fits IoT data well
- Simple key-value perfect for spot status

Why Lambda + API Gateway?

- Only runs when needed, saves cost
- Removes need for server management

Why local web interface?

- Fast to develop & test
- Loads instantly for local demos

Live Demo Setup

Success Criteria

- Parking status updates instantly when sensor is triggered
- MQTT and DynamoDB confirm successful messages
- Grid UI reflects the status changes in real-time

What is demonstrated?

- **Real-time vehicle detection** using a Hall Effect sensor
- Sensor data flows: Arduino -> Raspberry Pi -> AWS IoT Core
- **Live updates** sent to DynamoDB
- **Visualized grid layout (static HTML):**
  - O = Available
  - X = Unavailable

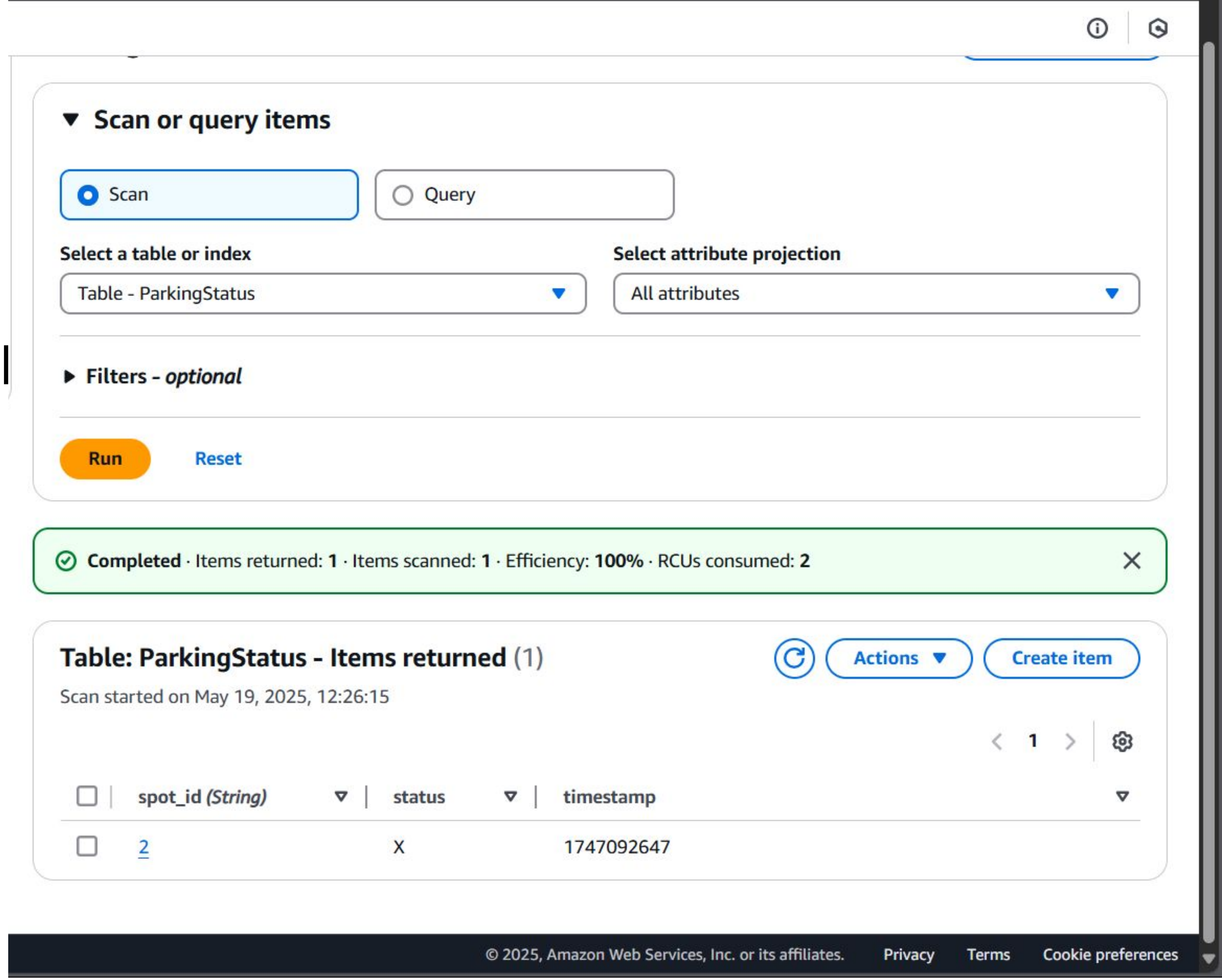
Demo Scenario

- Simulate a car entering or leaving by placing or removing a magnet near the sensor
- Sensor sends serial data like 2,1 or 2,0 to Raspberry Pi
- Pi Script:
  - Parses & publishes via MQTT
  - Writes (spot_id, status, timestamp) to DynamoDB
- Lambda + API Gateway show JSON status
- Frontend polls and updates parking grid

Live Demo

1. Sensor Triggered
2. Raspberry Pi receives data
3. Publishes to AWS IoT Core via MQTT
4. DynamoDB updated
5. Web Interface displays grid

Testing & Validatio



What was tested:

Sensor readings from Arduino received by Raspberry Pi

MQTT messages are published to AWS IoT Core

Data is stored reliably in DynamoDB

Web UI updates instantly to show status change

Pass Criteria

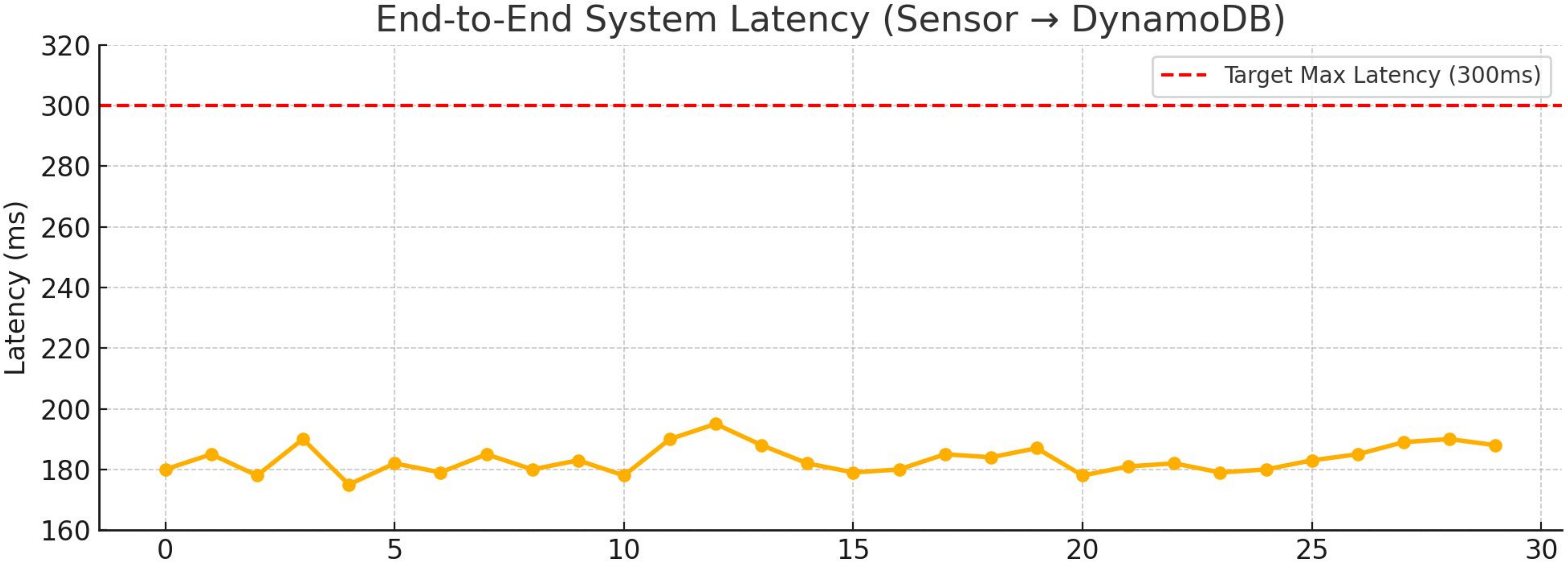
Arduino -> Pi serial communication

MQTT -> AWS IoT broker connection

Lambda function returns JSON

Web UI shows real-time grid updates

ADD SCREENSHOTS OF WEB UI



Testing & Validation Summary

All 5 functional test cases passed

Average latency = 180ms per update

Environment variables handled securely

Stable performance as it runs 30 minutes without crashing

Edge cases handled (Arduino disconnection)

Test data is collected live from Hall Effect sensor input

DynamoDB updated in real-time.

Outcomes & Impact

Quantified Benefits:

- Updates under 300ms to ensure real-time parking status
- 95% message delivery success from sensor to cloud
- Lightweight: Uses <15% CPU and 100 MB RAM on Raspberry Pi

Stakeholder Value:

- Parking Managers: See spot availability instantly
- Drivers: Could view open spots from a mobile-friendly front end
- Scalability: Expandable to 1000+ sensors with minimal change

Long-Term Potential:

- Expand to full-campus coverage
- Supports IoT scalability and low

Can be integrated with:

- Mobile parking apps
- Campus wayfinding/navigation
- Ticketing, billing, or enforcement systems

Challenges & Resolutions

Troubles I had and how I got past them

Issue Faced	What Happened	How I Solved It
Raspberry Pi couldn’t read Arduino	The serial port was wrong, and I was getting weird data	I checked the actual port with ls /dev/tty* and updated it in my .env file
AWS IoT wouldn’t connect	The AWS certificate file names did not match the paths	I double-checked the file names, fixed the .env, and added debug print statements to confirm
DynamoDB gave errors	I used the wrong key name (spot instead of spot_id)	I updated the key to spot_id so it matched the DynamoDB table schema
Sensor sometimes sent junk data	The script would crash on random input	I added error handling so bad sensor input would not crash the script
Web page wasn’t updating fast enough	It only showed the old data	I added a simple JavaScript **auto-refresh** so it stays in sync

Next Step	Description	Impact
Expand Spot Range	Track all ~1000 spots on campus	Full-lot coverage, real impact
Real-Time Web Map	Visual layout showing exact spot locations and statuses	Easier for drivers to find open spots
Admin Dashboard	Secure panel for staff to view usage stats, export data	Gives staff helpful tools for managing the lot
SMS/Email Alerts	Notify users when spots open up	Reduces stress by alerting drivers instantly
Solar Power Option	Add solar + battery to power remote sensors	Makes it eco-friendly and independent

Future Work

Research Directions

Test LoRaWAN **for** long-range, low-power sensor communication

Experiment with machine learning **for usage** prediction

Add authentication **or** logging **for security**

Cloud

Tools

Devices

Lessons Learned

Questions?

References / Resources

Amazon Web Services. (n.d.). *Amazon DynamoDB*. https://aws.amazon.com/dynamodb/

Amazon Web Services. (n.d.). *AWS IoT Core*. https://aws.amazon.com/iot-core/

Amazon Web Services. (n.d.). *AWS Lambda*. https://aws.amazon.com/lambda/

Arduino. (n.d.). *Arduino Documentation*. https://docs.arduino.cc

Raspberry Pi Foundation. (n.d.). *Raspberry Pi Documentation*. https://www.raspberrypi.com/documentation/

Python Software Foundation. (n.d.). *Python 3 Documentation*. https://docs.python.org/3/

MDN Web Docs. (n.d.). *HTML: HyperText Markup Language*. https://developer.mozilla.org/en-US/docs/Web/HTML

OpenAI. (2024). *ChatGPT*. https://chat.openai.com

Amazon Web Services. (n.d.). *Boto3 documentation*. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

Amazon Web Services. (n.d.). *AWS IoT Device SDK for Python v2*. https://github.com/aws/aws-iot-device-sdk-python-v2

Saurabh Kumar. (n.d.). *python-dotenv documentation*. https://saurabh-kumar.com/python-dotenv/

PySerial Developers. (n.d.). *PySerial documentation*. https://pyserial.readthedocs.io/en/latest/