Smart Parking Availability System

IoT-Based Project Plan Presentation (PPP)

Student: Garvin Patel

Goal: Real car detection + live availability dashboard

Problem Statement

- Students waste time driving through parking garages searching for open spots
- No clear visibility into availability before entering
- Results in:
  - Time loss
  - Traffic congestion
  - Frustration

Project Goal

Build a **real, working** smart parking system that:

- Detects whether spots are occupied
- Sends occupancy data wirelessly to a server
- Displays availability in a web dashboard in near real-time

What “Detection” Means in This Project

Parking availability will be determined using **real vehicle detection**.

- **Primary detection sensor:** Ultrasonic distance sensor (per spot)
- A spot is **Occupied** when the measured distance is below a threshold
- A spot is **Available** otherwise

Hardware

From the provided kit/skeleton, the project can use:

- **ESP32-C6** (WiFi / BLE)
- **Arduino Nano** (controller)
- **TMP102 (I2C temperature sensor)** (optional add-on)

> For real parking detection, **ultrasonic sensors are required**.

Selected Approach

Approach A: ESP32-C6 Direct (Simpler)

Ultrasonic Sensor(s) → **ESP32-C6** → WiFi → Backend → Web UI

Why: fewer parts, simpler wiring, easier to scale.

Alternative Approach (If Needed)

Approach B: Arduino + ESP32-C6

Ultrasonic Sensor(s) → **Arduino Nano** → Serial (UART) → **ESP32-C6** → WiFi → Backend

Why: separates sensing/logic (Arduino) from networking (ESP32).

How Ultrasonic Detection Works

1. Sensor measures distance from sensor to ground/vehicle
2. Arduino/ESP32 reads distance repeatedly
3. Apply rule:
    - If distance < threshold → **Occupied**
    - Else → **Available**

> Threshold is calibrated during testing for the garage setup.

Data Flow (End-to-End)

Ultrasonic Sensor → Microcontroller (ESP32/Arduino) → WiFi → API → Database → Dashboard

Backend Overview (Node.js + Express)

- Receives sensor updates via REST API (HTTP)
- Validates incoming data
- Stores spot states + history
- Computes:
    - total spots
    - occupied count
    - available count

Database Overview (MongoDB)

Collections (example):

- **spots**: spotId, currentStatus, lastSeen
- **readings**: spotId, distanceCm, occupied, timestamp

Why MongoDB:

- Flexible JSON structure
- Works well with time-series style data

Frontend Overview (React)

Dashboard shows:

- Total available spots
- Occupied vs available
- Live updates (polling or WebSocket later)
- Simple, readable UI for quick decision-making

Learning With AI (Project-Relevant Skills)

AI will be used as a learning + debugging assistant for:

- **C/C++ (Embedded Programming):** sensor reading, timing, state logic
- **ESP32 Networking:** WiFi setup, HTTP requests, reliability retries
- **Node.js + Express:** building REST APIs, input validation, error handling
- **MongoDB:** schema design, queries, storing time-series data
- **React:** state management, rendering live spot status

Key Features

1. Real vehicle detection using ultrasonic sensors
2. Occupancy classification (Occupied/Available)
3. Wireless transmission to server
4. API + database storage
5. Web dashboard for availability

Milestones & Sprints

Sprint 1 — Hardware + Local Detection

- Wire ultrasonic sensor(s)
- Read distance reliably
- Convert distance to occupied/available
- Calibrate thresholds

Sprint 2 — Connectivity + Backend

- ESP32 sends data to server
- Build Express API endpoints
- Store data in MongoDB
- Verify end-to-end ingestion

Sprint 3 — Dashboard + Integration

- Build React dashboard
- Display live status counts
- Integration testing (hardware → UI)
- Improve reliability + documentation

Progress Tracking & Metrics

- GitHub commits (weekly)
- Hardware milestone completion
- Accuracy of occupancy classification
- Successful transmission rate
- End-to-end latency (sensor → dashboard)

Risks & Mitigation

Risks

- Sensor noise / false triggers
- WiFi drops / missed updates
- Scaling to multiple spots

Mitigation

- Smoothing (moving average) + hysteresis thresholds
- Retry + heartbeat updates
- Start with 1–2 spots (MVP), then expand

PPP Readiness Statement

- Real detection method selected (ultrasonic)
- Clear architecture + data flow defined
- Implementation plan with milestones prepared

Thank You

Questions & Feedback Welcome