Smart Parking Availability System (IoT-Based)

Project Plan Presentation (PPP)

Garvin Patel

Computer Science — IoT Project

Problem Statement

Students in parking garages often:

- Drive through multiple levels to find open spots
- Waste time and fuel
- Create extra congestion inside the garage

Core issue: There's no simple way to know if a garage is full *before* entering.

Why This Matters

Impacts:

- **Time:** students arrive late to class
- **Traffic:** more cars circling inside the garage
- **Frustration:** repeated “search loops”
- **Efficiency:** poor usage visibility

Goal: Provide a simple “availability view” before and during entry.

Project Goal

Design an **IoT-based system** that:

1. Collects sensor data
2. Sends it wirelessly to a backend
3. Processes it into an “availability status”
4. Displays results on a web interface

Project Context (Provided IoT Skeleton)

We will build on the given hardware kit:

- **Arduino Nano** (main controller)
- **ESP32-C6** (WiFi/BLE communication)
- **TMP102 (I2C temperature sensor)** (representative sensor)

✅ TMP102 will **simulate parking-related signals** to validate:

- data flow
- wireless transmission
- backend + frontend pipeline

Proposed Solution (High-Level)

Pipeline:

1. TMP102 produces readings (representative data)
2. Arduino Nano reads + formats the data
3. ESP32-C6 transmits data to server (WiFi/BLE path)
4. Backend stores/processes
5. Frontend displays “availability status”

Key Features

1. **Sensor data collection** (TMP102 / simulated signal)
2. **Microcontroller data processing** (Arduino Nano)
3. **Wireless transmission** (ESP32-C6)
4. **Backend API** for receiving and storing readings
5. **Web dashboard** showing availability

Planned Technologies

Hardware

- Arduino Nano
- ESP32-C6 (WiFi/BLE)
- TMP102 (I2C)

Software

- Arduino C/C++
- Node.js + Express (API)
- MongoDB (data storage)
- React (web UI)

Architecture Overview (Data Flow)

Sensor → Controller → Wireless → Backend → Frontend

- TMP102 → Arduino Nano (I2C)
- Arduino Nano → ESP32-C6 (serial/UART suggested)
- ESP32-C6 → Backend (HTTP/MQTT; start with HTTP)
- Backend → MongoDB
- React frontend pulls status from API

Success Criteria (What “Working” Means)

By the end, we should demonstrate:

- Reliable sensor readings in the microcontroller
- Successful wireless data transmission to backend
- Data stored + retrievable from database
- UI displays availability status clearly
- End-to-end system works repeatedly (not “one-time”)

Project Plan & Milestones

Sprint 1 — Foundation

- Verify hardware setup + wiring
- Read TMP102 data via I2C
- Define data format + backend endpoints

Deliverable: stable sensor readings + documented data flow

Sprint 2 — Communication + Apps

- Send data from ESP32-C6 to backend (HTTP POST)
- Build backend API to receive/store data
- Create basic React UI (status + last update time)

Deliverable: data appears in DB and UI updates

Sprint 3 — Integration + Polish

- Full end-to-end integration testing
- Improve reliability & error handling
- Documentation + final demo readiness

Deliverable: consistent pipeline + final project report

Progress Tracking & Metrics

Tracking methods:

- GitHub commits + issues
- Sprint checklist completion
- Transmission success rate (messages delivered)
- System integration checkpoints (sensor → UI)

Risks & Mitigation

Risks

- IoT hardware learning curve
- Wireless communication instability
- Limited time for debugging integration

Mitigation

- Incremental testing (component-by-component)
- Simulated data if sensor constraints occur
- Use AI tools + documentation for troubleshooting

PPP Readiness Statement

All planning and system design is documented.

The project is ready for:

- PPP review
- stakeholder-style feedback
- milestone approval

Thank you!