

Blueprint: Demo Parking Management System (PMS)

Version: 1.0

Author: Gabriel Baziramwabo

Institution: Rwanda Coding Academy

Purpose: Educational Demonstration of an Automated Smart Parking Workflow

1. System Overview

The Parking Management System (PMS) is a practical, modular solution that replicates a real-world smart parking environment. It integrates **sensor-based automation**, **deep learning**, and **RFID-based payment systems** to manage vehicle entry, payment, and exit efficiently and intelligently.

The system is divided into three interoperable subsystems:

- **Car Entry System**
- **Payment System**
- **Car Exit System**

Each component operates independently while contributing to the synchronized workflow via shared logs and serial communication.

2. Hardware and Software Components

2.1 Materials

Component	Quantity	Location	Description
PC	2	Entry & Exit	Hosts Python applications for entry, exit, and payment.
Automatic Gate Demo Kit	2	Entry & Exit	Includes ultrasonic sensor, servo motor, red LED, blue LED, and buzzer.
Arduino UNO	3	Entry, Exit, PoS	Controls gate and reads sensor or RFID data.
RFID-Based Payment System (PoS)	1	Exit	Comprises PCD module + Arduino UNO, reads license plate and balance.

2.2 Firmware and Scripts

Subsystem	Firmware (.ino)	Python Script (.py)	PC Location
Car Entry	car_entry.ino	car_entry.py	Entry PC
Payment	process_payment.ino	process_payment.py	Exit PC
Car Exit	car_exit.ino	car_exit.py	Exit PC

3. Communication Architecture

Connection	Protocol Used	Description
PC ↔ Arduino	UART via USB	Used in all three subsystems.
RFID PCD ↔ Arduino	SPI	Used in the payment subsystem only.
Entry PC ↔ Exit PC	WiFi/Ethernet/LoRa/Zigbee/ BLE	Protocol depends on deployment environment and distance.
Log Synchronization	File Sync over Network	Ensures shared access to plates_log.csv.

4. System Logic and Data Flow

4.1 Car Entry Subsystem

Goal: Detect and log vehicle license plates, then control gate access.

Steps:

- 1. Car Detection:**
 - Ultrasonic sensor constantly checks for an approaching car.
 - If car detected, trigger image capture.
 - 2. License Plate Detection and Recognition:**
 - car_entry.py captures frame and runs YOLO-based detection.
 - Region of Interest (ROI) is cropped and passed to Tesseract OCR.
 - If a valid plate number is extracted:
 - Log plate and timestamp into plates_log.csv.
 - Send signal to Arduino to open gate (servo rotates).
 - Visual cues: Blue LED on, Red LED off, buzzer beeps briefly.
 - Else:
 - Gate remains closed. Red LED on.
-

4.2 Payment Subsystem

Goal: Retrieve license plate and balance from RFID card, calculate dues, process payment, and update payment status.

Steps:

1. **RFID Data Retrieval:**
 - RFID PCD module reads:
 - Plate number (Block 2)
 - Balance (Block 4)
 2. **Data Forwarding to PC:**
 - process_payment.ino sends this data via serial to process_payment.py.
 3. **Payment Logic in process_payment.py:**
 - Parses serial message for plate and balance.
 - Searches for plate in plates_log.csv.
 - Calculates parking duration: $\text{current_time} - \text{entry_time}$
 - Computes due amount (e.g., RWF 200/hour).
 - Logs amount in the amount column; keeps payment_status = 0.
 4. **Transaction Execution:**
 - Sends required amount back to Arduino.
 - Arduino deducts amount from PICC and rewrites new balance.
 - Sends “done” signal to PC upon success.
 - Python script updates payment_status = 1.
-

4.3 Car Exit Subsystem

Goal: Validate that payment is complete before permitting exit.

Steps:

1. **Car Detection and Plate Recognition:**
 - Ultrasonic sensor detects vehicle.
 - Camera captures image and car_exit.py uses YOLO + OCR to extract plate.
 2. **Payment Verification:**
 - Checks plates_log.csv for payment status.
 - If payment_status == 1:
 - Open gate, blue LED on, buzzer beeps.
 - Else:
 - Send '2' to Arduino to trigger warning buzzer.
 - Keep gate closed and red LED on.
-

5. Deep Learning Integration: YOLO for License Plate Detection

5.1 Rationale

Traditional OpenCV techniques (e.g., contours, Haar cascades) were inconsistent under real-world conditions — skewed angles, glare, shadows, or partial occlusions.

To improve robustness and precision, the system uses a **YOLOv8 deep learning model** for **real-time license plate detection**.

5.2 Training Setup

- **Framework:** Ultralytics YOLOv8
- **Dataset:** 100+ images of toy car plates at varying angles
- **Annotation Tool:** LabelImg (class = license_plate)
- **Split:** 80% training, 20% validation
- **Output Format:** YOLO format (.txt with normalized bounding boxes)

5.3 Inference Pipeline

1. Webcam captures image upon car detection.
2. YOLOv8 locates license plate bounding box.
3. ROI is cropped and passed to Tesseract OCR.
4. Text result is logged or verified for payment.

5.4 Benefits of YOLO Integration

- ✓ Real-time performance on standard PCs
 - ✓ Robust across diverse conditions
 - ✓ High accuracy with limited dataset
 - ✓ Easily upgradable or transferable to larger datasets
-

6. Shared Log File: plates_log.csv






This file is **central to synchronization** between systems.

Plate Number	Entry Time	Amount Due	Payment Status
RAA123B	2025-05-06 09:45:00	400	1
RAB321C	2025-05-06 10:20:00	200	0

- **Payment Status:**
 - 0 = Unpaid
 - 1 = Paid

This file must be updated and shared reliably between entry and exit systems for the PMS to operate correctly.

7. Key Features

-  **Synchronized Logging** between entry and exit systems
 -  **Secure Access Control** with AI-enhanced plate validation
 -  **Contactless Payment** with RFID deduction and verification
 -  **Flexible Networking** using WiFi, BLE, Zigbee, or wired Ethernet
 -  **Scalable Architecture** for both demonstration and real-world expansion
-

8. Future Enhancements

- **Cloud-Connected Dashboard** for remote monitoring and reporting
 - **Mobile App** for users to check parking time and balance
 - **Blacklist/Whitelist Support** to enable privileged access
 - **Vehicle Tracking** across multiple entry/exit points
 - **AI Payment Prediction Models** for real-time recommendations or alerts
-

9. Final Note to Learners: From Demo to Real World


This Demo Parking Management System may be implemented on a modest scale using entry-level components, but it represents **the full architecture of a real-world smart parking solution**.

If you understand and can build this integrated demo — complete with deep learning-powered license plate detection, synchronized logging, RFID-based payments, and automated entry/exit control — **you are already capable of developing a real-world system**.

In the field, only a few hardware upgrades are needed:

- **More powerful motors** to handle actual barrier gates.
- **Higher-resolution cameras** for wide-angle or long-range detection.
- **Cloud-based dashboards** for remote monitoring and analytics.

But the **core logic, protocols, and integration principles remain the same**.

 *Mastering the demo is mastering the real thing — the rest is just scaling up.*

Keep learning, keep building, and remember: **engineers are not made by tools, but by systems thinking**.

10. Conclusion

Absolutely! Since you've promoted the learner empowerment message to **Section 9**, here's the updated **Section 10: Conclusion**, refined to follow naturally and reinforce that transition:

10. Conclusion

The Demo Parking Management System stands as more than just a classroom project — it is a **complete, scalable foundation** for intelligent vehicle access control in real-world environments. Through a seamless integration of **deep learning**, **sensor automation**, and **secure payment systems**, learners gain practical experience with concepts that drive modern infrastructure.

By successfully developing and demonstrating this system, you are not merely simulating — you are **prototyping the real world**. With just a few hardware upgrades, the same codebase and architecture can power large-scale deployments in commercial or municipal settings.

This project equips you with:

- **AI integration skills** through YOLO-based detection.
- **Embedded systems expertise** with Arduino-based hardware.
- **Networked system design** for synchronized control points.
- **Problem-solving capacity** that bridges hardware and software.

The journey from learning to leading starts with hands-on systems like this. What you build today in the lab can shape the infrastructure of tomorrow.