Software Requirement Specification (SRS)

**Intelligent Urban Parking Management: Real-Time Vehicle Tracking using YOLOv8 and a User-Friendly GUI**

Group Members:

* *B Kavitha 23BD1A05AF*
* *K Nagashivashankar 23BD1A05AT*
* *Nida Madeeha 23BD1A05B4*
* *S Tarunika 23BD1A05BF*
* *T Akshaya 23BD1A05BK*

The **Intelligent Urban Parking Management System** is an advanced **real-time parking management solution** designed to optimize urban parking efficiency. Using **YOLOv8 and DeepSORT**, the system provides **real-time vehicle tracking** and **automatic entry/exit logging**. Integrated **IR and ultrasonic sensors** monitor parking slot availability, while an **OLED display** updates users with live status. Additionally, a **React web portal** enables for an **admin dashboard** which facilitates **slot management and monitoring**. The system is powered by **MongoDB** for secure data storage and designed for **scalability, security, and real-time operation**

Table of Contents

* **Introduction**  
  1.1 Purpose  
  1.2 Document Conventions  
  1.3 Intended Audience and Reading Suggestions  
  1.4 Project Scope  
  1.5 References
* **Overall Description**  
  2.1 Product Perspective  
  2.2 Product Features  
  2.3 User Classes and Characteristics  
  2.4 Operating Environment  
  2.5 Design and Implementation Constraints  
  2.6 Assumptions and Dependencies
* **Specific Requirements**  
  3.1 Functional Requirements  
  3.2 External Interface Requirements  
  - User Interfaces  
  - Hardware Interfaces  
  - Software Interfaces  
  - Communication Interfaces  
  3.3 Non-Functional Requirements
* **Appendices**
* **Glossary**

**1. Introduction**

**1.1 Purpose**

The **Intelligent Urban Parking Management System (IUPMS)** is designed to provide an automated solution for managing urban parking spaces efficiently. The system will integrate **real-time vehicle tracking, parking slot monitoring, automatic gate control, reservation system, and a web portal** for user convenience.

**1.2 Document Conventions**

* **FR-X**: Functional Requirements
* **NFR-X**: Non-Functional Requirements
* **UI-X**: User Interface Requirements

**1.3 Intended Audience and Reading Suggestions**

* **Developers** – For implementation details.
* **Project Managers** – To track project progress.
* **End Users (Admins & Customers)** – To understand system functionalities.
* **Testing**  – To verify compliance with requirements.

**1.4 Project Scope**

The **IUPMS** will:

* Track **real-time vehicle movement** using **YOLOv8** and **DeepSORT**.
* Monitor **parking slot availability** using **IR and ultrasonic sensors**.
* Provide an **automated gate system** with a servo motor and buzzer alerts.
* Detect and store the details the numberplate of vehicle using **OCR**
* Enable **online slot reservations** via **Node+Express+React web portal**.
* Store data in **MongoDB** for tracking vehicle entries, exits, and reservations.

**1.5 References**

* YOLOv8 & DeepSORT documentation
* PyQt5 for GUI
* MongoDB documentation

**2. Overall Description**

**2.1 Product Perspective**

This system will be **standalone for local deployment** and **integrated with a web portal**. It interacts with **hardware sensors (IR, ultrasonic), Machine Learning Algorithms and cameras, a servo motor, and a database (MongoDB).**

**2.2 Product Features**

1. **Real-time Vehicle Tracking** – Detects and tracks vehicles using **YOLOv8 & DeepSORT**.
2. **Parking Slot Monitoring** – Uses **IR sensors** to track occupied/free slots.
3. **Automated Gate System** – Opens/closes using a **servo motor & ultrasonic sensor**.
4. **Web-Based Reservation System** – Users can book slots via the **Node-Express-React web portal**.
5. **Role-Based Authentication** – Separate access for **Admins and Users**.

**2.3 User Classes and Characteristics**

* **Admin:** Manages parking slots, monitors real-time status, and assigns spaces manually if needed.
* **User:** Can **reserve** a slot and track their vehicle’s location in the parking lot.

**2.4 Operating Environment**

* **Hardware:**
  + **Cameras** (for vehicle detection)
  + **IR & Ultrasonic Sensors** (for slot detection)
  + **Servo Motor & Buzzer** (for gate control)
  + **OLED Display (128x64)** (for slot status updates)
* **Software:**
  + **Backend:**Node.js and Express.js
  + **Frontend:** React.js
  + **Database:** MongoDB
  + **GUI:** PyQt5 & Tkinter

**2.5 Design and Implementation Constraints**

* Must support **real-time processing** (GPU-accelerated for YOLOv8).
* Must work with **multiple cameras** simultaneously.
* Web portal should be **mobile-responsive**.

**2.6 Assumptions and Dependencies**

* Users will have **internet access** for the web portal.
* MongoDB must be **running locally** before the system starts.

**3. Specific Requirements**

**3.1 Functional Requirements**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requirement Description | Priority |  |
| FR-1 | The system shall detect vehicles using **YOLOv8** in real-time. | High |  |
| FR-2 | The system shall track vehicle movement using **DeepSORT**. | High |  |
| FR-3 | The system shall monitor parking slot occupancy via **IR sensors**. | High |  |
| FR-4 | The **servo motor** shall open the gate when a car is detected. | High |  |
| FR-5 | The system shall store **entry/exit logs** in MongoDB. | High |  |
| FR-6 | The web portal shall allow users to **reserve a parking slot**. | High |  |
| FR-7 | The system shall display **available parking slots** on an **OLED display**. | High |  |
| FR-8 | The admin shall manage parking slots through a **dashboard**. | High |  |
| FR-9 | The system shall support **multi-camera** real-time tracking. | Medium |  |
| FR-10 | The buzzer shall sound when the **gate opens**. | Medium |  |
|  |  |  |  |

**3.2 External Interface Requirements**

**User Interface (UI) Requirements**

* **UI-1**: The PyQt5 GUI must display **real-time vehicle tracking**.
* **UI-2**: The OLED display must update with **live slot availability**.
* **UI-3**: The web portal should have a **login/signup system** for users.

**Hardware Interface Requirements**

* The **IR sensors** must correctly detect vehicle presence.

**Software Interface Requirements**

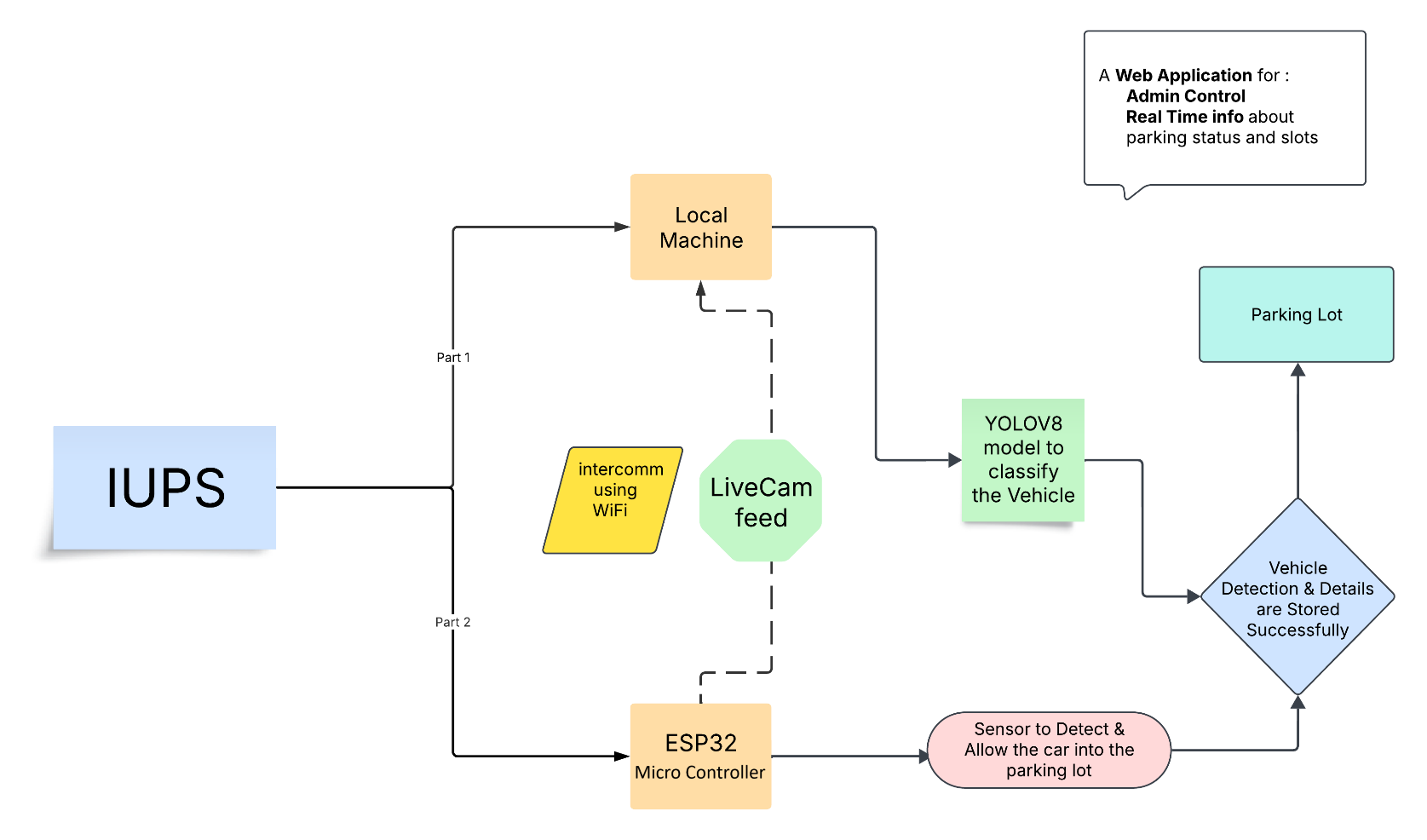
* YOLOv8 model must load within **5 seconds** on startup.
* Web portal must use **Flask API** for data exchange with MongoDB.

**3.3 Non-Functional Requirements**

|  |  |
| --- | --- |
| Category | Requirement |
| Performance | The system must process **live video feeds** at **30 FPS**. |
| Security | Users must authenticate via **email and password**. |
| Availability | The web portal must be accessible **24/7**. |
| Scalability | The system must support **at least 75 parking slots**. |
| Maintainability | The system must allow **easy sensor recalibration**. |

**4. Appendices**

* **System Architecture Diagram** (YOLOv8 + DeepSORT + MongoDB + Node+ Express + React).



**IUPS (Intelligent Urban Parking System)**

* Acts as the **central entity** of the system.
* Responsible for managing the overall parking workflow.
* Integrates **computer vision, IoT devices, web applications, and databases**.

**Live Camera Feed**

* Captures **real-time video** of incoming vehicles.
* Sends the feed to the **local machine** for processing.
* Works as an input for **YOLOv8-based vehicle detection**.
* Can be a **USB camera, IP camera, or CCTV camera**.

**Local Machine**

* The **core computing unit** responsible for:
  + **Processing the camera feed**.
  + Running the **YOLOv8 model** for vehicle detection.
  + Communicating with the **ESP32 microcontroller**.
  + Sending data to the **web application** for real-time monitoring.
* Can be a **PC, edge computing device, or cloud server**.

**YOLOv8 Model for Vehicle Classification**

* **Deep learning-based object detection model**.
* Identifies **vehicles entering/exiting the parking lot**.
* Detects **vehicle type, number plate, and motion tracking**.
* Uses **DeepSORT or ByteTrack** for continuous tracking.

**ESP32 Microcontroller**

* Acts as a **bridge between hardware components and software**.
* Uses **WiFi communication** to interact with:
  + **Local machine** (for receiving vehicle detection data).
  + **Sensors** (for vehicle presence detection).
  + **Gate control system** (to open/close the entry barrier).
* Runs **MicroPython or Arduino C** for MQTT-based communication.

**Intercommunication using WiFi**

* ESP32 and the **local machine communicate wirelessly**.
* Uses **MQTT protocol** for real-time data exchange.
* Enables **remote control and monitoring** without wired connections.

**Sensor to Detect Vehicles & Gate Control**

* Uses an **Ultrasonic Sensor or IR Sensor** to detect incoming vehicles.
* The ESP32 **monitors the sensor data**.
* If a vehicle is detected:
  + The **gate automatically opens**.
  + The system **logs the entry time & vehicle details**.
* The **servo motor controls the gate movement**.

**Vehicle Detection & Data Storage**

* Once a vehicle is identified:
  + **Vehicle details (number plate, type, timestamp) are stored in MongoDB**.
  + **Parking slot allocation is updated**.
* Uses **Node and Express** to send data to the web application.

**Web Application (Admin Control & Real-Time Info)**

* Built using **Node and Express(backend) and React.js (frontend)**.
* Displays **real-time parking status** to:
  + **Admins**: Full control over the system.
  + **Users**: Can check available slots.
* **Features:**
  + Live **dashboard** with parking slot status.
  + **Login system** with Admin/User roles.
  + **Real-time updates using Socket.IO or MQTT**.

**Parking Lot**

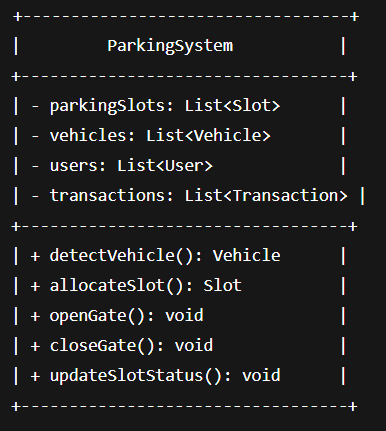
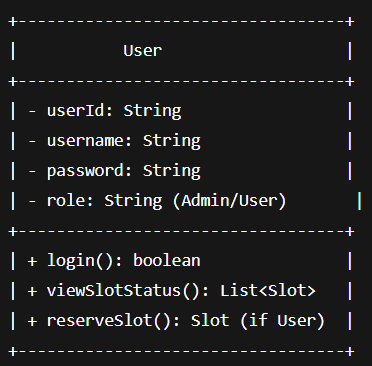
* The **final destination** where the vehicle is parked.
* The system **updates slot availability**.
* Ensures **efficient space management**.

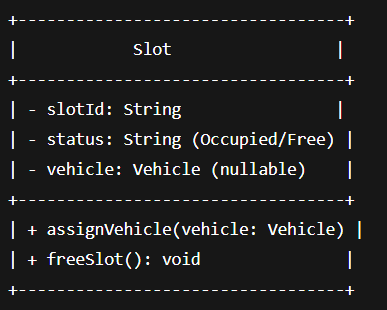
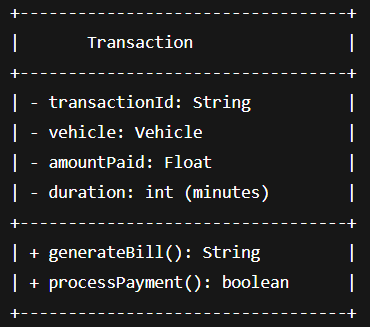
**How Everything Works Together**

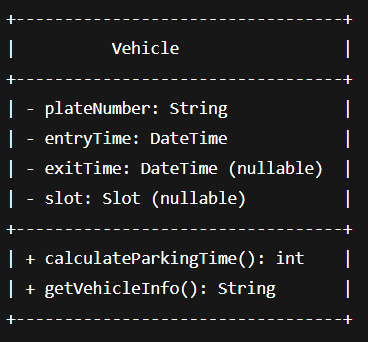
1. **Live Camera Feed** captures the incoming vehicle.
2. **Local Machine processes** the feed using YOLOv8.
3. **ESP32 communicates** with the machine via WiFi.
4. **Sensors detect the vehicle** and activate the **gate system**.
5. **Web Application updates** the parking status in real time.
6. **Admin can monitor & control** the parking lot remotely.

**Class Diagrams**

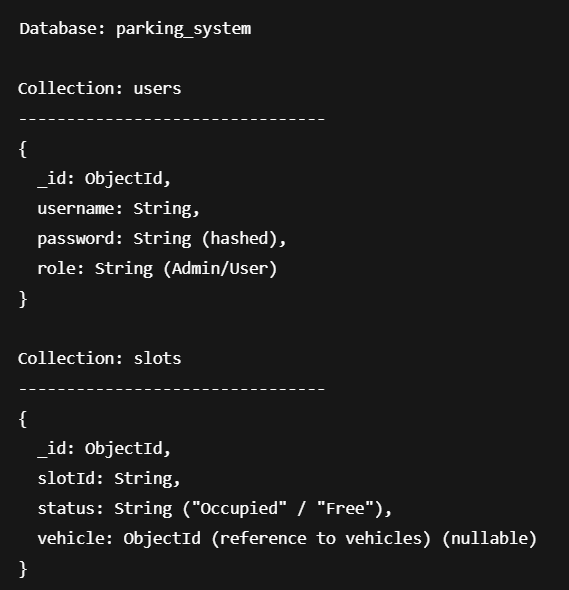
The **Class Diagram** represents the core structure of the **Intelligent Urban Parking System (IUPS)**, illustrating the relationships between different components. It follows an **object-oriented approach**, ensuring modularity and scalability.

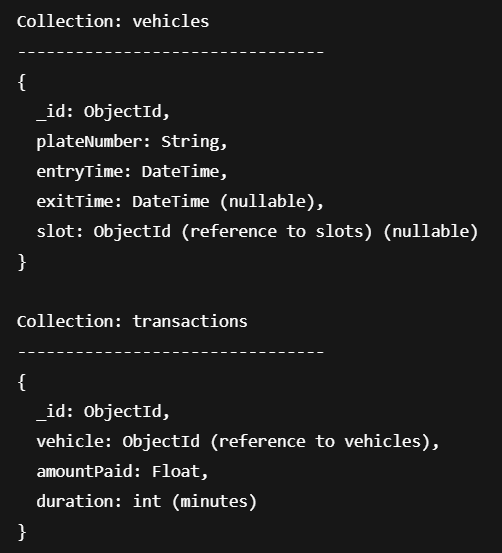




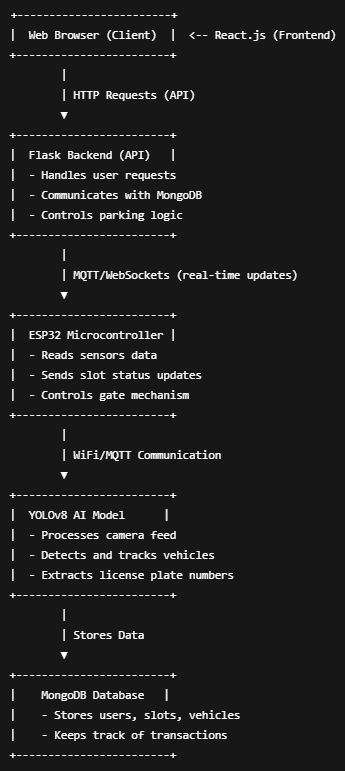


**Database Design (MongoDB Schema)**

****

****

**Deployment Diagram**

****

**5. Glossary**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| YOLOv8 | A deep learning model for real-time object detection. |
| DeepSORT | A tracking algorithm for multi-object tracking. |
| PyQt5 | A GUI framework for Python. |
| Express | A lightweight web framework. |
| MongoDB | A NoSQL database for storing parking data. |